

Politecnico di Torino

Department of Mechanical and Aerospace Engineering



**Politecnico
di Torino**

Master's Degree in Aerospace Engineering

Implementation of a Python-based software approach in aviation data analysis

Supervisor

Prof. Giorgio GUGLIERI

Author

Giovanna Francesca
CALLEGARI

Co-supervisors

Beatrice CONTI, PhD

Giuliano ANTONICIELLO, PhD

April, 2022

Abstract

Aviation safety is considered a vital topic to study and enforce periodically as recommended by the proper authorities. At European level every state uses their own methodology for safety data analysis, it is then important to revise and update the procedures. The purpose of this thesis is to investigate an alternative method for processing safety data in aviation. The results suggest a potential future shared approach.

A *Python-based user-friendly software pipeline* has been developed, which seeks to be comprehensive and collective. The choice of the programming language has been motivated by the need for an efficient manipulation and an optimal elaboration of the input data. The process started with Excel spreadsheets containing Italian safety data, which were cleaned, organized, and then used as a benchmark for our analysis.

The selected topics are *Safety Performance Indicators* and *Dangerous Goods*, respectively. The former are parameters defined to measure safety performances and the latter are any hazardous substances that happen to occur in in-flight or in ground operations. Using input files provided by ENAC, several templates have been derived that give as an outcome the analysis embedded in the yearly ENAC Safety Report¹. In addition, the method allows to assess the accuracy of the analysis through warnings and validation, and it can be easily updated for incoming years. The presented results are twofold: tabular data results and graphical visualisations.

¹<https://sites.google.com/enac.gov.it/enacsafetyreport/introduction?authuser=0>

Abstract

Nell'ambito dell'aviazione civile la safety è considerata uno tra gli aspetti più essenziali da esaminare e migliorare periodicamente secondo le direttive delle autorità competenti. A livello europeo ogni stato utilizza le proprie metodologie nell'analizzare i dati riguardanti la safety, è dunque importante verificare e rinnovare i processi. Lo scopo di questa tesi è ricercare un metodo alternativo per processare i dati safety. I risultati ottenuti suggeriscono un potenziale approccio condiviso in futuro.

È stato sviluppato un *software pipeline* basato su Python, tale da essere intuitivo, generale ed esauriente. La scelta del linguaggio di programmazione è derivata dalla necessità di manipolare ed elaborare i dati di input in modo efficiente. Partendo da fogli di lavoro in Excel, in seguito ad una pulizia ed ad un'organizzazione dei dati a livello italiano, questi sono stati utilizzati come riferimento per la nostra analisi. Gli argomenti selezionati sono gli *Safety Performance Indicators* e i *Dangerous Goods*. I primi sono parametri utilizzati nella misurazione del livello di safety performance, mentre i secondi sono qualunque sostanza pericolosa segnalata durante operazioni in volo o a terra. Utilizzando come input i documenti forniti da ENAC, sono stati realizzati vari template tali da fornire come output l'analisi annuale presente nell'ENAC Safety Report². Il metodo, inoltre, permette di accertare l'accuratezza dell'analisi attraverso messaggi di errore e validazioni, oltre ad essere facilmente aggiornabile per i prossimi anni. I risultati proposti sono di duplice natura: in forma tabulare e in forma grafica.

²<https://sites.google.com/enac.gov.it/enacsafetyreport/introduction?authuser=0>

To the people I care the most

Table of Contents

List of Tables	VII
List of Figures	VIII
Acronyms	XI
1 Introduction	1
1.1 General view on Safety	1
1.2 Organisations and safety programs	5
2 Safety at Italian level	9
2.1 Safety Report Portal	9
3 Topics of analysis	15
3.1 Dangerous Goods	15
3.1.1 General definition	15
3.1.2 Classification and coding	17
3.1.3 European state of art	19
3.2 Safety Performance Indicators	25
3.3 ENAC reporting system	28
4 Python-based Pipeline: Dangerous Goods	31
4.1 Python	31
4.2 Input data	33
4.2.1 Excel Files	33
4.2.2 Reading files	35
4.3 Event types	36
4.3.1 Grouping	38
4.3.2 Checks and validation	40
4.4 Dangerous Goods types	43
4.4.1 Gathering all data	44

4.4.2	ICAO codes	47
4.4.3	Grouping	50
4.4.4	Checks and validation	52
5	Python-based Pipeline: Safety Performance Indicators	55
5.1	Input data	55
5.1.1	Reading Excel files	57
5.2	General template	57
5.3	Event types template	59
5.4	Checks and validation	60
6	Output graphics	63
6.1	Reproducing Safety Portal results	63
6.1.1	Dangerous Goods	63
6.1.2	Safety Performance Indicators	66
6.2	Finding new visualisation options	69
6.2.1	Dangerous Goods	70
6.2.2	Safety Performance Indicators	76
7	Conclusions	81
A	Python-based Pipeline: DGs template	83
B	Python-based Pipeline: SPI template	111
C	Implementation of a Python-based dashboard	123
C.0.1	Plotting Adaptation	124
C.0.2	Designing the layout	125
	Bibliography	149

List of Tables

1.1	Categories of safety risk probability and severity.	2
1.2	Safety risk tolerability.	3
3.1	ENAC summary table on Outcome oriented SPI. (Tabella 1 – Indicatori operativi from [10])	26
4.1	Table model of Event types counting output.	38
4.2	Resulting table of Event types analysis.	41
4.3	Intermediate resulting table of Dangerous Goods analysis.	46
4.4	ICAO codes and classes.	48
4.5	Table model of DG types counting and matching output.	49
4.6	Division of codes into class 9 groups.	50
4.7	Resulting table of Dangerous Goods types analysis.	51
5.1	Resulting table of general SPI analysis on <i>Runways Excursions</i> . . .	59
5.2	Resulting table of general SPI analysis on <i>Runways Incursions</i> . . .	59
5.3	Resulting table of Event types analysis on <i>Runways Incursions</i> . . .	60

List of Figures

1.1	Acceptable level of safety performance (ALoSP). (Figure 8-4 from [1])	5
2.1	Data analysis on Runway Excursions (RE) from ENAC Safety Report Portal.	11
2.2	Event types analysis from ENAC Safety Portal.	12
2.3	Dangerous Goods Classes analysis from ENAC Safety Portal.	13
2.4	Legends of Dangerous Goods analysis in ENAC Safety Portal.	13
3.1	An example of how Dangerous Goods list is presented by ICAO.[12]	17
3.2	Reported events on RAMP from 2019 report in Austria.	19
3.3	Occurrences related to the transport of Dangerous Goods reported by Belgian organizations.	20
3.4	Hazardous goods incidents reported to the LBA in Germany.	21
3.5	Summary of the 6,697 occurrence reports submitted by Irish AOC holders during 2017. (DGs on the 16 th row)	21
3.6	Dangerous Goods reported in the Netherlands.	22
3.7	Reports on Dangerous Goods in Norway from 2019 report.	23
3.8	Reports on Dangerous Goods in Poland from 2018 report.	24
3.9	Reports on Dangerous Goods in Spain from 2016 report.	24
4.1	Flowchart of the Dangerous Goods analysis.	32
4.2	Example of input data for the time period 2014-2019. ('Source' and 'UTC date' are classified and indicated generally.)	34
4.3	Example of input data for the time period 2020-2021. ('Source' and 'UTC date' are classified and indicated generally.)	34
4.4	Example of input data in the attached files.	34
4.5	Example of multiple Event types in a single Excel cell. ('Source' and 'UTC date' are classified and indicated generally.)	42
4.6	Python report for 2019 on analysis validation.	42
4.7	Example of different descriptions for the same DG type from 2019 data. ('Source' and 'UTC date' are classified and indicated generally.)	43

4.8	Example of exceptions in DG type definitions from 2019 data. ('Source' and 'UTC date' are classified and indicated generally.) . .	43
4.9	Code output of the validation check on attached files' length from 2020 data.	45
4.10	Code output of the validation check on attached files' length from 2020 data.	46
4.11	Python output for codes' check.	47
4.12	Python report for 2019 on analysis validation.	52
4.13	Python report for 2020 on analysis validation.	53
5.1	Flowchart of the Safety Performance Indicators analysis.	56
5.2	Example of general input data of SPIs. (Sensible data is classified and indicated generally.)	56
5.3	Example of input data of SPIs with Event types data. (Sensible data is classified and indicated generally.)	56
6.1	Event types output chart.	64
6.2	Legend of figure 6.1	64
6.3	Dangerous Goods Types output chart.	65
6.4	Legend of figure 6.3.	65
6.5	Bar plot on occurrences per year and rate per year of <i>Runway Excursion</i> , as included in ENAC safety portal.	66
6.6	Trend of movements per year.	67
6.7	Bar plot on occurrences per year and rate per year of <i>Runway Incursion</i> , as included in ENAC safety portal.	67
6.8	Trend of fights per year.	68
6.9	Trend of occupancy duration per year.	68
6.10	Bar plots on Event types analysis of 'Runway Incursions', as included in ENAC safety portal.	69
6.11	Trend of total amount of Event types reported yearly.	70
6.12	Stacked bar plot of Event types focusing on the years.	71
6.13	Stacked bar plot of Event types focusing on the types.	72
6.14	Legends of stacked plots of Event types analysis.	72
6.15	Multiple bar plots of Event types focusing on their values.	73
6.16	Trend of total amount of Dangerous Goods types reported yearly. .	74
6.17	Stacked bar plot of DG types focusing on the years.	75
6.18	Stacked bar plot of DG types focusing on the classes.	75
6.19	Legends of Dangerous Goods analysis in ENAC Safety Portal. . . .	76
6.20	Trend of fights per year for <i>Runway Excursions</i>	77
6.21	Trend of fights per year for <i>Runway Incursions</i>	77
6.22	Bar plots and line plots on Event types analysis of <i>Runway Incursions</i> . 78	

6.23	Legend of figure 6.24.	78
6.24	Stacked bar plot on Event types analysis of <i>Runway Incursions</i> . . .	79
6.25	Line plot on Event types analysis of <i>Runway Incursions</i> with rate trends together.	79
C.1	Flowchart relating to the implementation of the dashboard.	124
C.2	Example of a tooltip in an interactive plot included in the dashboard.	124
C.3	Drop-down menus in the <i>SPI</i> page of the dashboard	126
C.4	Drop-down menu in the <i>Dangerous Goods</i> page of the dashboard. .	126
C.5	<i>Home</i> page of the dashboard.	126
C.6	<i>Safety Performance Indicators</i> page of the dashboard.	127
C.7	<i>Dangerous Goods</i> page of the dashboard.	127
C.8	<i>About</i> page of the dashboard.	127

Acronyms

AAR

Consolidated Annual Activity Report

ADREP

Accident/incident data reporting

ALoSP

Acceptable level of safety performance

ANSV

Agenzia nazionale per la sicurezza del volo

API

Application programming interface

ASR

Annual Safety Review

DG

Dangerous Good(s)

Doc

Document

EASA

European Union Aviation Safety Agency

ECAC

European Civil Aviation Conference

eE-MOR

electronic ENAC - Mandatory Occurrence Reporting

ENAC

Italian Civil Aviation Agency

ICAO

International Civil Aviation Organisation

MOR

Mandatory Occurrences report

MST

Member State Tasks

PAS

European Plan for Aviation Safety

SMS

Safety management system(s)

SPAS

State Plan for Aviation Safety

SPD

Single Programming Document

SPI

Safety performance indicator

SPT

Safety performance target

SRM

Safety risk management

SSP

State safety programme

UN

United Nations

Chapter 1

Introduction

1.1 General view on Safety

The concept of safety describes a state or a place where someone is safe and not in danger and it is a concern valued in many fields regarding everyday life. Working, travelling and living in a safer environment has become a fundamental topic to study, analyze and implement.

In civil aviation, safety is:

'the state in which risks associated with aviation activities, related to, or in direct support of the operation of aircraft, are reduced and controlled to an acceptable level' [1].

As aviation itself has been evolving through the years, also safety is a dynamic concept: since the early 1900s, different approaches have been implemented to continuously identify and mitigate new hazards and risks. In order to define an international standard to manage safety, since 2006 the International Civil Aviation Organization (ICAO) has been publishing periodically the document *Safety management manual* (Doc 9859)[1] and in November 2013 published the Annex 19, regarding *Safety Management*. This concept aims at directly addressing safety risks mitigation and accident/incident prevention and it supports a state managing its activities more accurately toward risks and more efficiently in term of resources. The implementation of these actions is made possible by two main tools: the State Safety Programme (SSP) and the Safety Management System (SMSs). The first one is a collection of regulations and activities performed by each state in order to improve safety, whereas the second one is a systematic approach to identify hazards, collect and analyze data and continuously manage safety risks, aiming at a better safety performance. The benefits of enforcing safety management can include strengthened safety culture, better understanding of safety-related interfaces and

relationship, enhanced early hazard detection, data-driven decision-making, and cost avoidance [1].

Particular emphasis should be given to the subject of safety culture. This topic is a key element to understand how safety is perceived, valued and prioritized by management and employees. Organisations with a positive safety culture are more likely to achieve their goals thanks to the collaboration of all stakeholders. A positive safety culture relies on a high degree of trust and respect between personnel and management, which is the foundation of an efficient safety reporting. As a matter of facts, the reporting system helps to gather data and information in order to detect existing and potential safety deficiencies and hazards [1].

Looking into ICAO definition of safety, it is important to understand how risk is reduced and controlled and what can be considered as an acceptable level. In particular, Safety Risk Management(SRM) is a continuous activity that follows the aviation evolution over time and includes hazard identification, safety risk assessment, safety risk mitigation, and risk acceptance. Hazards are inevitable components of aviation since they are considered as dormant potentials for harm, but they are not an issue as long as they are controlled and mitigated by understanding their consequences. The aim of hazard identification is to identify risks before they actually lead to dangerous events. This process can include reporting system, inspections, audits, brainstorming sessions, expert judgments and also reviews of internal and external investigation reports on accidents or incidents [1].

As far as defining safety risks, ICAO presents an indication of *safety risk tolerability*, which is based on the measurements of *safety risk probability* and *safety risk severity*. However, these are only guide-lines: every safety organisation adapts them to its specific needs and complexities. Safety risk probability is the likelihood of an occurrence to happen, whereas safety risk severity is the level of danger that can be expected from that occurrence.

The safety risk probability and severity may be categorized as shown in Table 1.1.

Likelihood	Severity
Frequent	Catastrophic
Occasional	Hazardous
Remote	Major
Improbable	Minor
Extremely improbable	Negligible

Table 1.1: Categories of safety risk probability and severity.

With the categories of Table 1.1, it is possible to build a matrix assigning to each

slot the acceptable level of safety. Safety risk tolerability can be divided into three levels as shown in Table 1.2 [1]:

- Intolerable (red), requiring to take immediate actions to mitigate the risk or stop the activity;
- Tolerable (orange), requiring to pay attention to the mitigation process;
- Acceptable (yellow), requiring no additional mitigation.

Probability/Severity	Catastrophic	Hazardous	Major	Minor	Negligible
Frequent					
Occasional					
Remote					
Improbable					
Extremely improbable					

Table 1.2: Safety risk tolerability.

With a look at safety risk mitigation, the purpose of the process is to reach an acceptable safety level through the application of appropriate controls. Such goal can be achieved by reducing the severity of the potential consequences, the probability of occurrences or the exposure to that safety risks. Usually the second option is more common to be applied. Mitigation actions often result in changes in operating procedures, equipment or infrastructures and can be categorized as:

- Avoidance, cancelling the operation;
- Reduction, reducing the frequency of the operation;
- Segregation, isolating the effects of the consequences of the risk.

In order to choose the appropriate mitigation alternative, different perspectives should be taken into account in the interests of finding an optimal solution. Therefore the decision should be postponed after examining effectiveness, cost/benefits, practicality, acceptability, enforceability, durability, residual safety risks, unintended consequences, and time [1].

Another fundamental process is the safety performance management, that helps to verify the effectiveness of implemented safety activities towards the objectives,

checking the functionality of SSPs and SMSs. Therefore, a set of *Safety performance indicators (SPIs)* need to be defined to measure performances and to understand whether the taken actions are adequate or further mitigation is required. They both describe the ongoing scenario and support decision-making. The choice of these parameters is based on the available data for each specific State.

The safety objectives to pursue are:

- *Process-oriented*: stated in terms of safe behaviours expected from operational personnel or the performance of actions implemented by the organization to manage safety risk;
- *Outcome-oriented*: encompassing actions and trends regarding containment of accidents or operational losses.

SPIs generally can be divided into: quantitative, qualitative, lagging, and leading indicators. A quantitative indicator is expressed with a number or a rate, whereas a qualitative one is descriptive based on quality. Quantitative indicators can be counted and compared, so that they are a preferable choice over the qualitative indicators. Lagging indicators, also known as 'outcome-base SPIs', measure past events, so they allow to study the overall long-term efficiency. They can be classified into low probability/high severity and high probability/low severity. Leading SPIs are known as "activity or process SPIs" and they measure conditions that can potentially result in a specific outcome. For leading SPIs the focus is on inputs and procedures, in order to anticipate failures [1].

Additional parameters to set safety achievements are the *Safety performance targets (SPTs)*, which define short-term and medium-term goals. Once SPIs are defined and data is analyzed, trends will stand out and indicate the direction along which the safety performances are steering, so the organisation can set reasonable and achievable SPTs for each SPI. This process should be periodically reviewed to guarantee that targets are pointed toward safety performance improvements [1].

The combination of SPIs and SPTs defines the *Acceptable level of safety performance (ALoSP)*, which indicates the safety levels that a State demands for its aviation system. It represents what the State evaluates as significant in this matter and needs to be developed according to the safety objectives in the SSP. It includes targets for each sector and measures to determine the effectiveness of safety activities [1].

As an indication of the relation between SPIs, SPTs and ALoSP, Figure 1.1 is presented.

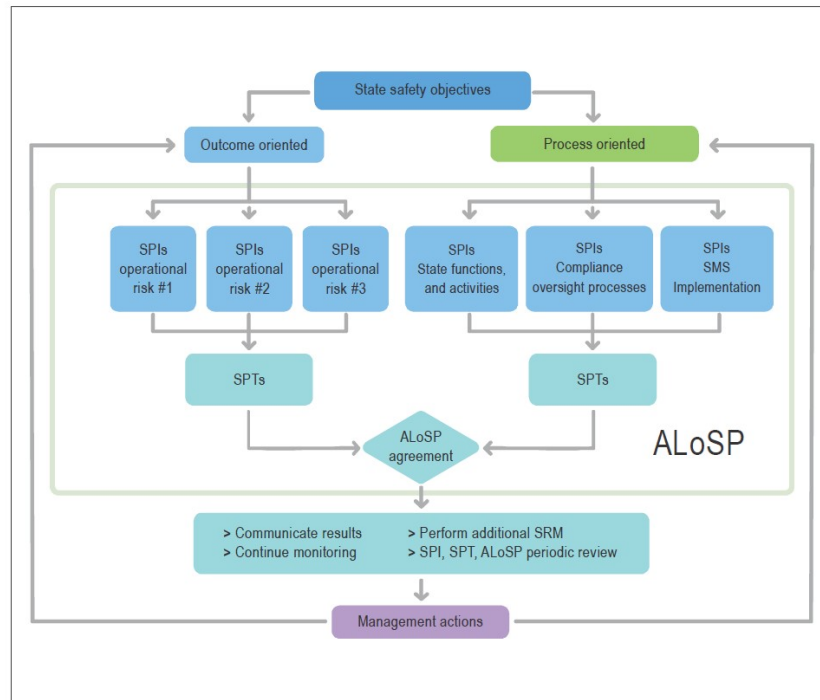


Figure 1.1: Acceptable level of safety performance (ALoSP). (Figure 8-4 from [1])

1.2 Organisations and safety programs

As aviation industry had grown in the last decades, regulations and controls became essential. Almost every country owns and flies aircraft around the globe, so it is important to manage and control the scenario and to avoid dangerous events. As a result, associations and agencies were founded to the purpose.

ICAO [2] is the International Civil Aviation Organization, found in 1944 at the Chicago Convention. It is formed by 193 national governments and it supports their diplomacy and cooperation in air transport in order to accomplish a sustainable growth of the global civil aviation system. Its mission includes developing policies and standards, undertaking compliance audits, performing studies and analyses, providing assistance, and building aviation capacity through many other activities and the cooperation of its Member States and stakeholders.

The organisation is committed to strengthen the aviation system by providing suggestions and solutions, that the Member States are encouraged to follow but they are not obligated.

ICAO collective objectives can be summarized as:

- Safety
- Air Navigation Capacity and Efficiency
- Security & Facilitation
- Economic Development of Air Transport
- Environmental Protection.

At European level, EASA [3] is the European Union Aviation Safety Agency, founded in 2002 and based in Cologne, Germany. The member states who collaborate are 27 from the European Union plus Switzerland, Norway, Iceland, and Liechtenstein. Its mission can be summarised as following:

- Ensure the highest common level of safety protection for EU citizens;
- Ensure the highest common level of environmental protection;
- Single regulatory and certification process among Member States;
- Facilitate the internal aviation single market and create a level playing field;
- Work with other international aviation organisations and regulators.

The agency works publishing periodically programs and reports, which contain documentations useful to the Member States. There are four key programming documents [4]: Single Programming Document (SPD), Consolidated Annual Activity Report (AAR), European Plan for Aviation Safety (EPAS), and Annual Safety Review (ASR). The first one contains the activities and the resources required to achieve a safer European Aviation Space. The second one reports how the annual work program, budget and staff resources have been implemented evaluating the results in term of objectives, performance indicators and timetable set, the risks associated with those activities, the use of resources and the general operations of the Agency, and the efficiency and effectiveness of the internal control systems. The third one is a key component of EASA's integrated Safety Management System (SMS) at the European level, it covers a five-years period and addresses three key-issues: systemic issues, operational issues and emerging issues. The last one consists in a report of the most common key risk areas and associated safety issues that lead to accidents in each of the different operational aviation domains.

In particular, the European Plan for Aviation Safety [5] provides a regional safety plan for the Member States at European level, including the strategic priorities,

strategic enablers, main risks affecting the aviation system and the necessary actions to mitigate those risks. The purpose of this document is to ensure that the system for the management of aviation safety in the EU delivers the highest level of safety performance, uniformly enjoyed across the whole Union, and continuing to improve over time, while taking into account other important objectives, such as environmental protection. It explains the functioning of the European aviation system to manage the safety of civil aviation in the EU in accordance with *Regulation (EU) 2018/11393* (*‘Basic Regulation’*). It describes the processes, roles and responsibilities of the different actors and lays down general principles for European safety management, including safety action planning. The document is divided in three volumes: Vol. I contains strategies and key-indicators, Vol. II contains the actions and Vol. III the "Safety Risk". The actions in Vol. II are divide in:

- Evaluation Tasks
- Memeber State Tasks
- Reasearch Tasks
- Rulemaking Tasks
- Safety Promotion Tasks.

In particular, the Member State Tasks (MST) are actions directly addressed to the members and have to be considered for their State Plan for Aviation Safety (SPAS).

ENAC [6] is the Italian Civil Aviation Authority, regarding technical regulation, certification, surveillance, and control. Its main activities includes ensuring safety during both in-flight and ground operations and security. It is also monitoring the observance of *The Passengers’ Bill of Rights* and subcontracts airport national properties. Among its roles, it defines and recommends developing programs for the airport national system, ensures both operations in national air space and services of air navigation and implements policies against air and noise pollution. Lastly, it represents Italy at international level, collaborating with the major aviation organisations; for instance ICAO, ECAC, EASA, and EUROCONTROL.

As demanded by art.8 of *Regulation (EU) 2018/1139*, every EU member state is required to write a Safety Plan for Aviation Safety (SPAS), following EASA directions included in the EPAS and its timeline.

The aim of this document is reaching a higher level of safety performance through a constant improvement of institutional activities. The actions the state promotes to achieve the goals are based on the available national safety data. The safety performance is valued in terms of *Safety Performance Indicators*, which are compared

with the *Safety Performance Targets*. The measurements and analyses of these indicators are included in the *Safety Report*, which is published annually, according to art.13 of *Regulation (EU) 376/2014* [7].

With reference to *Regulation (EU) 376/2014*, there are no specific indications on what this report must include, therefore every state has the authority to decide on what and how to displays data and analyses. For this reason, the Safety Reports appear really different depending on the states that is considered and consultation could be challenging [8].

Chapter 2

Safety at Italian level

Each EU member states needs to provide an indication of the level of safety performance, defined by Safety Performance Indicators, as mentioned in section 1.2. It is mandatory to publish at least once a year the Safety Report, as required by art. 13 of *Regulation (EU) no. 376/2014*.

In order to comply with European and aviation requirements, ENAC had published Safety Reports in paper format until 2019, from 2020 the agency has implemented a Safety Report Portal¹ [9].

2.1 Safety Report Portal

An overview of the Safety Report Portal is important to understand how Italy evaluates safety at Italian level and what is identified as crucial to analyze. ENAC has implemented a portal to share the results of safety data analysis, that used to be included in the Safety Report in paper format. The change is based on the modern technological habits that encourage the use of a user-friendly Web page instead of a paper document.

The website contains information allocated in multiple sections. Some sections introduce descriptions and regulatory references and they are: *Introduction*, *Why a Safety Portal?*, *Safety at Italian level*, *State Plan for Aviation Safety*, and *Occurrence Class*. Some other sections contain data analyses, which differ from one another in term of considered years and outputs and they are *Accident Rate*, *SPI-O*, *SPI-S*, *Dangerous Goods*, *Unruly Pax*, and *Clear Air Turbulence*. Lastly, the final

¹From now on, as **Safety Report Portal**, it is intended ENAC Safety Report published on the related Web portal [9], whereas as **Safety Report** or **Safety Review**, it is intended the traditional paper document.

sections present informative material, and they are: *Final remarks*, *Information Sources*, *Past Safety Report*, and *Useful links*. From now on, the focus is on the conducted analyses.

In the section *Accident Rate*, ENAC reports a comparison of the accident rate registered in Italy, Europe and the entire world. The outputs are line plots and the timeline is from 2008 to 2020.

The *SPI* section reports the analysis of Safety Performance Indicators as they are described in the document '*Safety Performance Indicators*' [10]. They are divided in the two categories: outcome oriented (O) and process oriented (S). The fifteen considered SPI-O are:

- Runway Excursions (RE)
- Runway Incursions (RI)
- Loss of Control in-Flight (LOC-I)
- TCAS Resolution Advisories (TCAS)
- Activation of TAWS (TAWS)
- Ramp events (RAMP)
- Collision while taxiing to or from a runway in use (GCOL)
- Fire or smoke on aircraft (F-IN)
- Laser beam interferences with flight operations (LASER)
- BRI (Bird Strike Index) (BIRD)
- Airspace Infringements (UPA)
- Separation Minimum Infringement number (SMI)
- Technical Failure (ATM Failure)
- Interferences of APR with manned aircraft during take-off or landing (APR Interference).

Generally, they are displayed with three graphs: rate per year, occurrences per year and movements/flights per year in a timeline from 2015 to 2020, by bar and line plots. There are, however, some exceptions: BIRD and ATM failure present only the occurrences per year graph, UPA and SMI consider the occupancy time in the sky. Then LASER and RAMP reports additionally geographical references

and RI and TCAS consider further examinations. As a general example, Figure 2.1 displays the graphs relating to Runway Excursions.

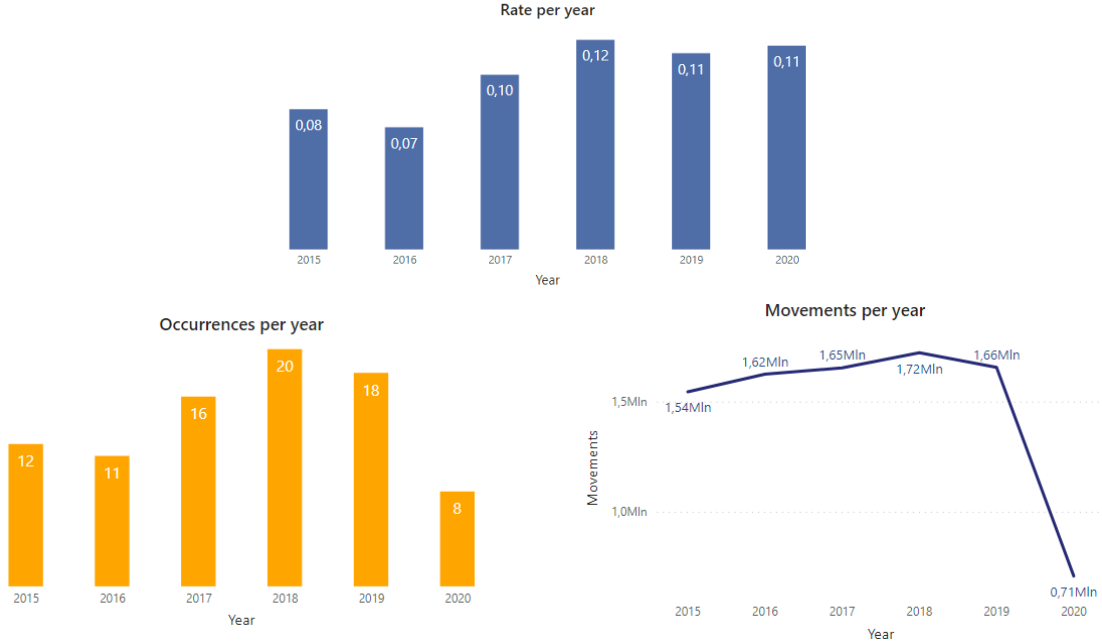


Figure 2.1: Data analysis on Runway Excursions (RE) from ENAC Safety Report Portal.

SPI-O are explained in details in section 3.2, as they are one of the subjects of this thesis.

Regarding the other category, the five considered SPI-S are: Inspections Performed, Occurrences reported by private pilots, Occurrences reported by APR pilot/operators, Training activities on aviation safety, and Reactivity index to Safety Recommendations issued by ANSV. In this case, the analysis is specific to each one of the them and it is more varied, but the most common visualization method is the bar plot.

The *Dangerous Goods* section displays information about hazardous goods reported at ENAC. This is one of the subject of this thesis and it is examined in depth in section 3.1. In the safety portal analysis, four different plot are addressed: Dangerous Good Undeclared, Event Type’s role in DGORs², Classes’ role in DGORs and Event Classification per Year; in a timeline from 2015 to 2020. The first chart represents a category of Event types and, as well as the fourth, it is a bar plot.

²The definitions of Event types and DG classes are addressed in section 3.3

Whereas the two middle charts are pie charts. Moreover the fourth plot represents the level of safety effects related to the DG reports. In particular, the second and the third plots will be fundamental to the development of this project, so they are presented in Figure 2.2 and Figure 2.3.

Then, *Unruly Pax* section considers passengers who do not respect one or multiple rules related to the airplane or the airport. The displayed plots are three: rate per year, occurrences per year and flights per year in a timeline from 2015 to 2020, by bar and line plots. This approach is similar to the one used in SPI-O section (see figure 2.1).

Lastly, in the *Clear Air Turbulence* section, it is presented the number of turbulence occurrences per year and the number of those with injury. Both the subjects are displayed as bar plots in a timeline from 2015 to 2020.

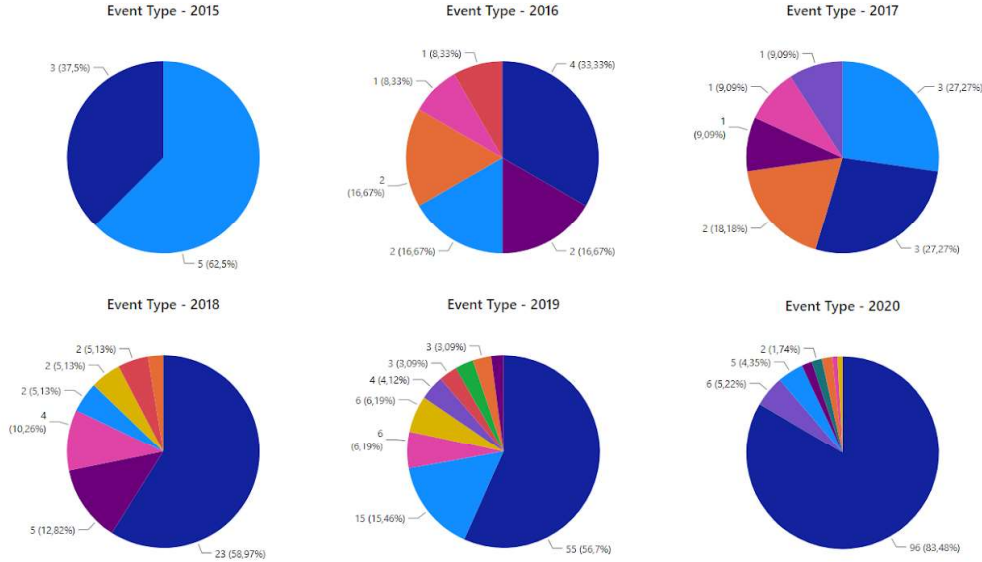


Figure 2.2: Event types analysis from ENAC Safety Portal.

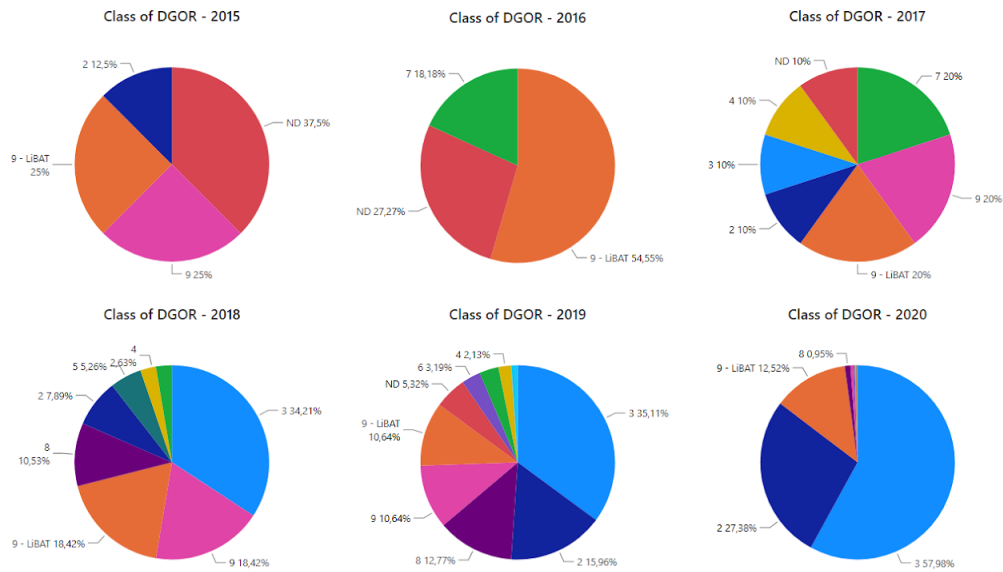


Figure 2.3: Dangerous Goods Classes analysis from ENAC Safety Portal.

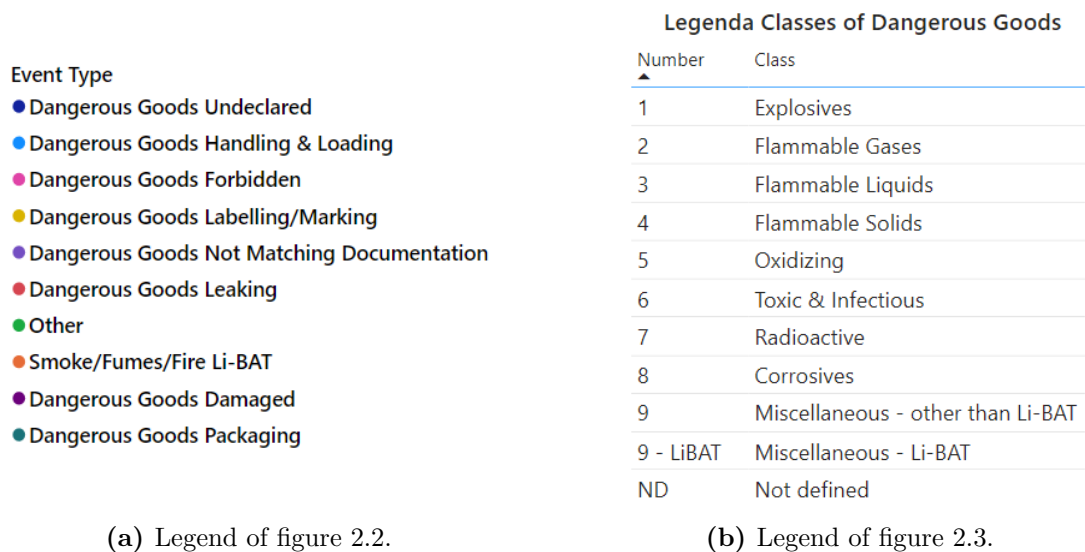


Figure 2.4: Legends of Dangerous Goods analysis in ENAC Safety Portal.

Chapter 3

Topics of analysis

The analyses developed in this thesis have two main focuses: *Dangerous Goods* and *Safety Performance Indicators*, which are two of the topics included in the ENAC Safety Report Portal. In this section, both these two topics are explained in-depth in the interests of understanding the decision-making process of the analyses, explained in chapter 4 and chapter 5, and the presented results. We provide general definitions and categorisations, and in addition a description of the current European scenario on DGs. Lastly, at the bottom of the chapter there is a brief summary on the Italian reporting system.

3.1 Dangerous Goods

3.1.1 General definition

Following ICAO definition, Dangerous goods are defined as

"articles or substances which are capable of posing a risk to health, safety, property or the environment and which are shown in the list of dangerous goods in the Technical Instructions or which are classified according to those Instructions" [11].

To guarantee safety at any level, DGs need to be listed and identified in order to be carried by air without placing an aircraft or its passengers at risk. The classifications and the recommended procedures are described in the *Technical Instruction for the Safe Transport of Dangerous Goods* (ICAO Doc 9284) [12] and revised periodically. Generally, DGs are divided into different classes or divisions according to the hazard they present. For instance some goods can be carried both in passengers and all-cargo aircraft following required conditions, but others are restricted to only all-cargo aircraft or with specific approval from the concerned

States, and some DGs are too dangerous to be carried on any aircraft. In transporting, they need to be packed, marked and labelled to be easily recognized. In addition, the pilot-in-command needs to be informed on what is on board because in case of emergency DGs could be crucial when deciding on actions.

Considering packing and labelling, packages must be in good quality, strong enough to resist the transport and the related loads. They need to be closed in the interest of avoiding leakage. Shippers must follow packing instructions and evaluate the package conditions.

Every Dangerous Good (identified as so by ICAO) must be tagged with a danger class label for the specific hazard it entails. Labels are required to be fully visible firmly affixed to or printed on the package. Moreover, labels must resist open weather exposure without loss of effectiveness and they must follow precise design in term of colour, symbols and general format, for instance contrasting colour in the background, solid outer boundary line, etc [12].

There are two types of occurrences related to DGs: *accident* and *incident*. The first one is "an occurrence associated with and related to the transport of dangerous goods by air which results in fatal or serious injury to a person or major property or environmental damage" [12]. The second one is "an occurrence, other than a dangerous goods accident, associated with and related to the transport of dangerous goods by air, not necessarily occurring on board an aircraft, which results in injury to a person, property or environmental damage, fire, breakage, spillage, leakage of fluid or radiation or other evidence that the integrity of the packaging has not been maintained. Any occurrence relating to the transport of dangerous goods which seriously jeopardizes the aircraft or its occupants is also deemed to be a dangerous goods incident" [12].

Both DGs accidents and incidents have to be reported so the authority can determine the causes and take action to prevent a recurrence, whenever possible.

ICAO provides a table containing the list of all Dangerous Goods most commonly carried, it can not be considered exhaustive but it illustrates all dangerous substances of commercial importance. The above-mentioned table is named as 'Table 3-1. Dangerous Goods List' in the ICAO relating document. There are 13 columns, each of them with specific information on the DG considered. The notable columns for this thesis are the second one and the third one, called 'UN no.' and 'Class or division', respectively. The first column is called as 'Name' and indicates the proper alphabetically shipping name of the good, but it is not essential for following developments.

In order to show how the table is built, a row is presented as an example in Figure 3.1.

3-2-4									Part 3			
Name	UN No.	Class or division	Subsidiary hazard	Labels	State variations	Special provisions	UN packing group	Excepted quantity	Passenger and cargo aircraft		Cargo aircraft only	
									Packing instruction	Max. net quantity per package	Packing instruction	Max. net quantity per package
1	2	3	4	5	6	7	8	9	10	11	12	13
Acetonitrile	1648	3		Liquid flammable			II	E2	353 Y341	5 L 1 L	364	60 L

Figure 3.1: An example of how Dangerous Goods list is presented by ICAO.[12]

3.1.2 Classification and coding

Classification

As a general rule, ICAO provides a classification of DGs in nine classes according to the hazard or the most predominant hazard they present. However the categorization is affected by state variations, as a matter of facts the appropriate national authority is required to define it.

A summary of the classes can be represented as following.

- **Class 1: Explosives**

Division 1.1: Substances and articles which have a mass explosion hazard

Division 1.2: Substances and articles which have a projection hazard but not a mass explosion hazard

Division 1.3: Substances and articles which have a fire hazard and either a minor blast hazard or a minor projection hazard or both, but not a mass explosion hazard

Division 1.4: Substances and articles which present no significant hazard

Division 1.5: Very insensitive substances which have a mass explosion hazard

Division 1.6: Extremely insensitive articles which do not have a mass explosion hazard

- **Class 2: Gases**

Division 2.1: Flammable gases

Division 2.2: Non-flammable, non-toxic gases

Division 2.3: Toxic gases

- **Class 3: Flammable liquids**

- **Class 4:** Flammable solids; substances liable to spontaneous combustion; substances which, on contact with water, emit flammable gases

Division 4.1: Flammable solids, self-reactive and related substances and desensitized explosives

Division 4.2: Substances liable to spontaneous combustion

Division 4.3: Substances which, in contact with water, emit flammable gases

- **Class 5:** Oxidizing substances and organic peroxides

Division 5.1: Oxidizing substances

Division 5.2: Organic peroxides

- **Class 6:** Toxic and infectious substances

Division 6.1: Toxic substances

Division 6.2: Infectious substances

- **Class 7:** Radioactive material

- **Class 8:** Corrosive substances

- **Class 9:** Miscellaneous dangerous substances and articles, including environmentally hazardous substances.

The considered order is not a representation of the degree of danger, which is furthered in the relating document. For each class, ICAO defines definitions, subdivisions, packing instructions and specific properties. To each considered good, the number of the class/division is assigned, as shown in the third columns of Figure 3.1.

Coding

Each Dangerous Good is identified by a serial number assigned to the article or substance under the United Nations classification system. This number is composed of four digit, as shown in Figure 3.1. There are, however, two exceptions: goods that do not have assigned a UN number are identified with a temporary identification number in the 8000 series and those that can be forbidden on aircraft under any circumstance are labelled as 'FORBIDDEN'.

3.1.3 European state of art

EU Member states have the authority to manage data on their reports as they consider more appropriate, so it is important to look into the current outline of the European states to understand if and how these states display their data and what can be identified as the preferable approach. In this comparison the research considers national safety reports from official websites and authorities. It must be noted that the analysis takes place on a qualitative level.

The countries that are going to be examined are: Austria, Belgium, Germany, Ireland, Italy, Netherlands, Norway, Poland and Spain. Some of them use their official language in their reports, so a common online translator is used to read the papers in English.

In the following investigation it will be clear that the scenario is really varied. Therefore there is not indeed a common way to analyze this topic. It is in this regard that our developed method may set the starting point for a future common approach.

Austria

On Austrian civil aviation authority website all safety reports since 2017 are available and there are some relating content. From 2018 to 2020 Dangerous Goods are included in the RAMP category, which considers Ground Handling activities. An example of the trend of reported events is showed in Figure 3.2, from 2019 report. Whereas, in the 2017 report no data is included [13].

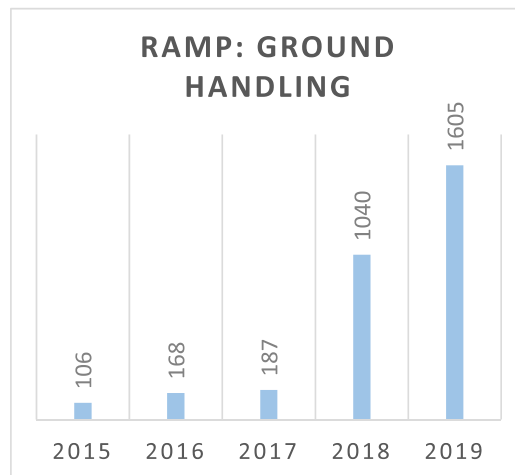


Figure 3.2: Reported events on RAMP from 2019 report in Austria.

Belgium

About Belgium aviation safety, there are available two safety plans: 2016-2020 and 2020-2024. In both documents, DGs are considered as a Safety Performance Indicator and there are explanations about actions/objectives on handling DGs, however only in the second one some data is presented [14][15]. In Figure 3.3 it is shown information of 2019 occurrences through percentages.

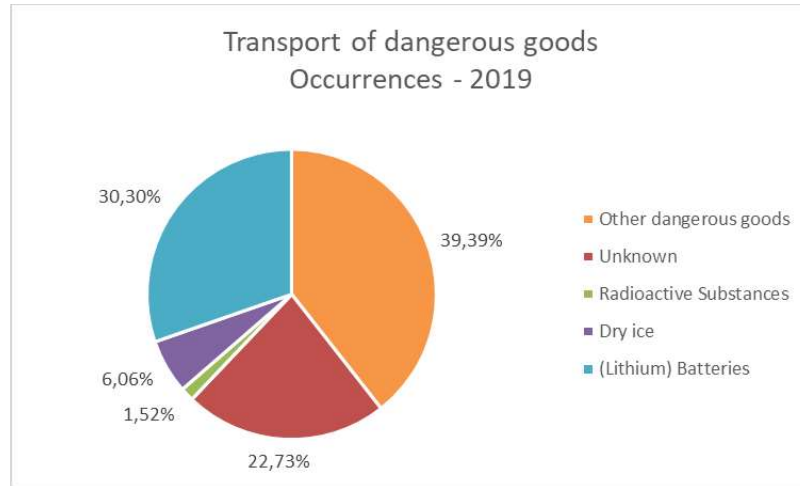


Figure 3.3: Occurrences related to the transport of Dangerous Goods reported by Belgian organizations.

Germany

The German civil aviation authority (LBA) publishes safety reports since 1995. Germany does a continuous monitoring on the topic including its trend throughout the years. In particular, since 2006 all air carriers and airports have additionally been reporting all dangerous goods incidents/accidents to the LBA [16]. Figure 3.4 represents the trend from 2000 to 2018; only in 2017-2019 report it is displayed as a graph, whereas in the other reports only by tabular data.

Ireland

Consulting safety reviews from 2016 to 2020, Irish authority takes into account DGs only in some years. In particular, in 2016 and 2017 DGs are considered individually, whereas in 2019 and 2020 there is no evident specification. Figure 3.5 represents one of the graphs included in 2016 and 2017 reports, it shows how many occurrences correspond to DGs. A second graph is added which represent a summary of MOR reports in a 4-years period for the Irish AOC holders [17, 18, 19].

An das LBA gemeldete Gefahrgutzwischenfälle

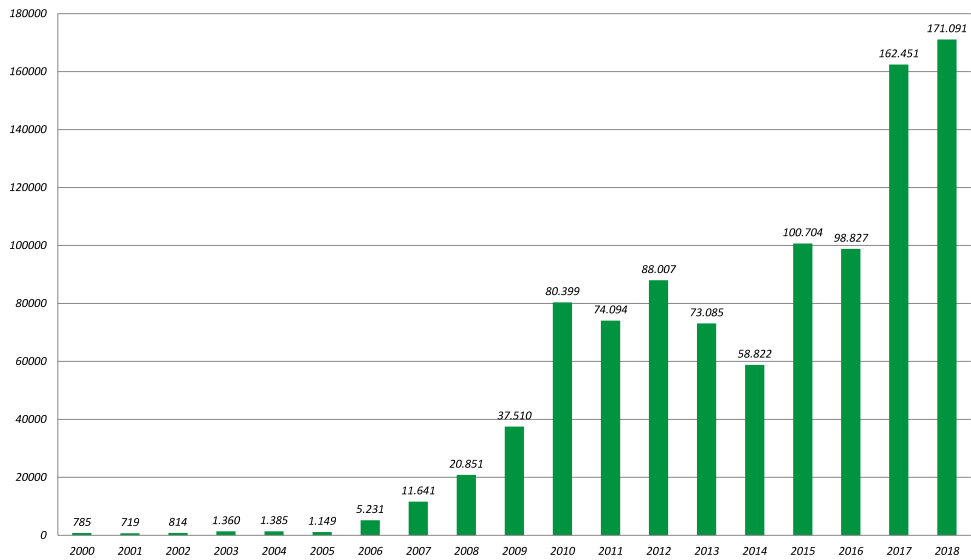


Figure 3.4: Hazardous goods incidents reported to the LBA in Germany.

Figure B.4: Summary of the 6,697 occurrence reports submitted by Irish AOC holders during 2017
(The AOC holders conducted 1,018,827 flights over this period)

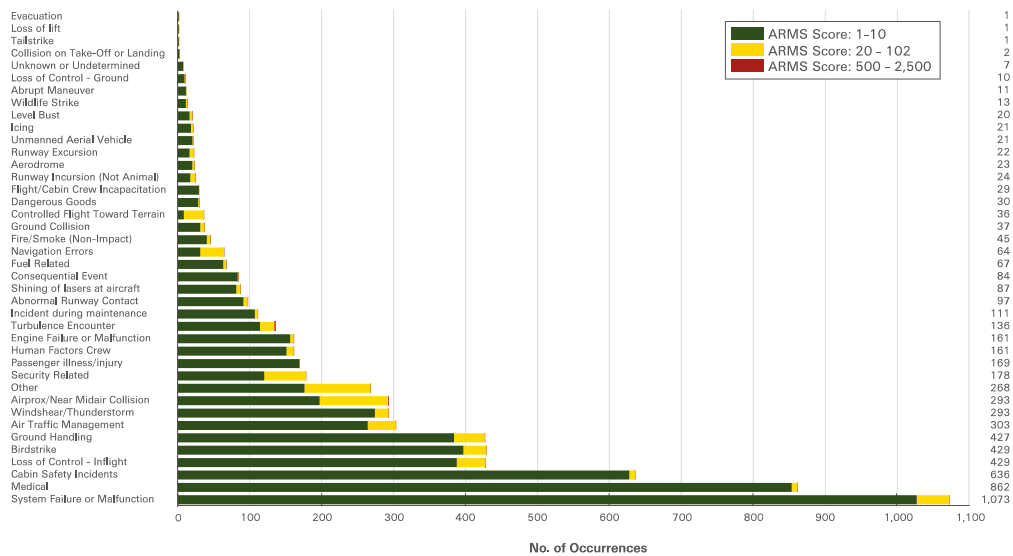


Figure 3.5: Summary of the 6,697 occurrence reports submitted by Irish AOC holders during 2017. (DGs on the 16th row)

Italy

In the current Italian scenario, safety reports refer to DGs only in 2014-2015, whereas there are no data available from 2016 to 2019. However in 2020 a Safety Portal has been created where data from 2015 is displayed [20, 21, 22][9]. There are different reported data, in particular the analysis considers the topic in terms of Event types and DG classes.

For additional details, consult chapter 2.

Netherlands

On the Dutch website it is presented a dashboard, showing safety indicators, including DGs. The timeline considered is between 2017 and 2021 and the reported occurrences are divided in notifications about batteries, leakage notifications and other, as showed in the Figure 3.6.

It is noticeable that the dashboard contains several filters, to show information separately in terms of subtopic, type of aviation and location [23].

Gevaarlijke goederen

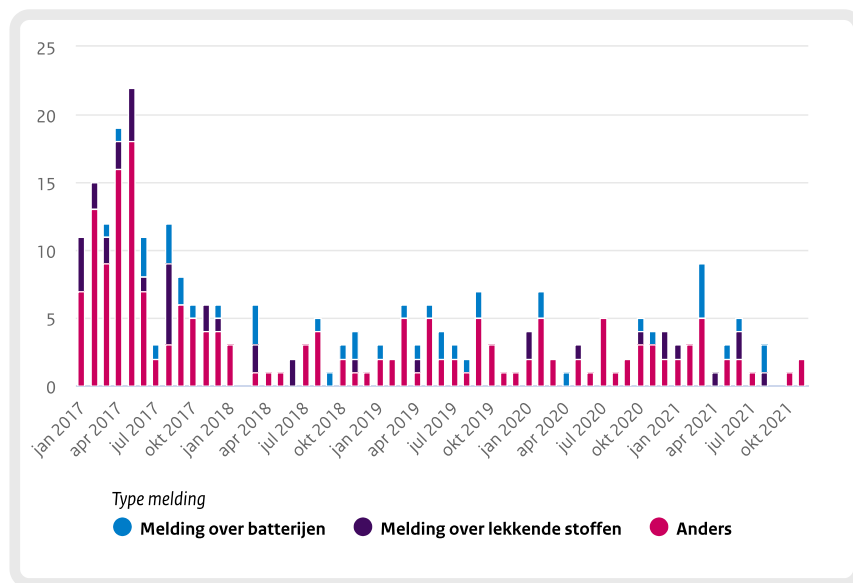


Figure 3.6: Dangerous Goods reported in the Netherlands.

Norway

The only document available is the 2019 safety review and it does reference to DGs. Norwegian civil aviation authority presents the trend on DG reporting occurrences

from 2012 to 2019, as shown in Figure 3.7 [24].

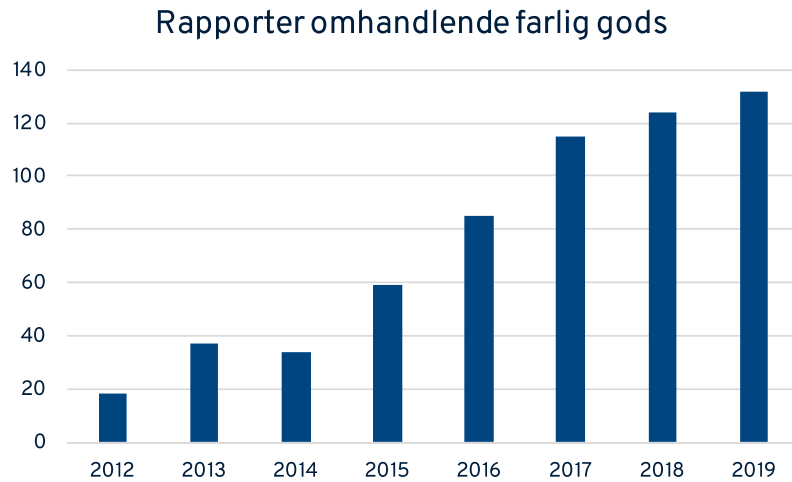


Figure 3.7: Reports on Dangerous Goods in Norway from 2019 report.

Poland

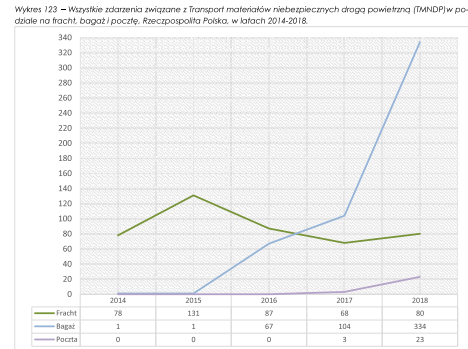
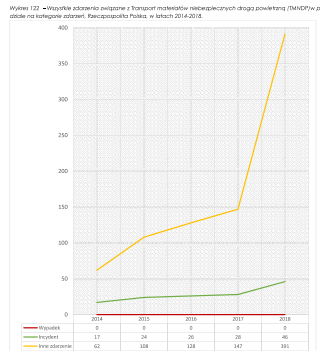
Safety reviews of 2017 and 2018 are available on the official website and Poland analyzes DGs and their trends. Dangerous goods are considered as an additional Safety Level Indicator.

Looking at the Figure 3.8 from 2018 report, it is clear that both graphical and tabular data are presented, however in 2017 report only tabular data can be found. In particular the Figure 3.8a refers to occurrences, divided in incident, accident and other, whereas the Figure 3.8b refers to type of events, such as freight, luggage and mail. There are also two more graphs included in the report presenting incidents in terms of passengers only, and cargo only plus mail [25] [26].

Spain

On the website only reports from 2015, 2016 and 2017 are available. The Spanish authority considers DGs included in the Handling category and it analyses event rate divided in categories and specifically by types, as shown in Figure 3.9. In 2016 report it is reported data from 2015 and 2016, whereas in 2015 report from 2014 and 2015 [27][28].

In the 2017 report the handling category is not included [29].



(a) All incidents related to Transport of Dangerous Goods by air, broken down by event categories, Republic of Poland, in 2014-2018.

(b) All events related to Transport of Dangerous Goods by air, broken down into freight, luggage and mail, the Republic of Poland, in 2014-2018.

Figure 3.8: Reports on Dangerous Goods in Poland from 2018 report.

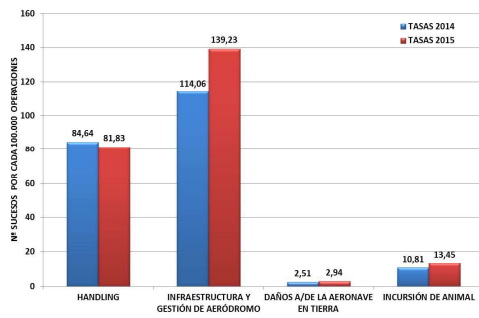


Figura 8-6. Tasa de sucesos de Entorno Aeroportuario (2014/15)



Figura 8-7. Tasa de Sucesos de Entorno Aeroportuario por tipos

(a) Airport Environment event rate (2015/16).

(b) Airport Environment Event Rate by type.

Figure 3.9: Reports on Dangerous Goods in Spain from 2016 report.

Other cases

In Europe, there are some states that do not report any information about Dangerous Goods. There are also some others that do not make available to the public their Safety Report. Further investigations should take place, if interested.

3.2 Safety Performance Indicators

Safety Performance Indicators are parameters that help the aviation authorities measuring the characteristic level of safety performance of a state. They are chosen depending on the safety data available to be analyzed and they need to illustrate the safety national objectives.

Each SPIs should include:

- a description of what it measures;
- its purpose;
- the units of measurement and any requirements for its calculation;
- the person in charge for collecting, validating, monitoring, reporting and acting on the SPI;
- where or how the data should be collected;
- the frequency of reporting, collecting, monitoring and analysis of the SPI data [1].

Regarding safety at Italian level, ENAC has drawn up the recommended SPIs in the document *Safety Performance Indicators* [10]. A work group identified them and ENAC Safety Board evaluated and validated them.

SPIs are divided in two categories:

- *Outcome oriented*. They derive from the measurement of events that could be the precursors of "undesired events" (accident or serious incident) and normally they are measured considering mandatory reports received in the eE-MOR system. These indicators have been chosen taking into account types of events which are particularly relevant in all the domains of civil aviation: Aerodrome, Air Traffic Control, Airworthiness, Operations and UAS.
- *Process oriented*. They derive from the most typical processes of the Civil Aviation Authority and plan to measure the effectiveness of ENAC activities trying to ensure the highest possible level of safety of the aeronautical operations [10].

ENAC provides two tables containing the required information for each SPI. Since the focus of this thesis is on SPI-outcome oriented, Table 3.1 illustrates them as displayed in the related document. The first column indicates the specific code of the considered SPI; the second its extended name; the third a short description of what it defines; the last one indicates the sources of the data (eE-MOR system, ENAC or EUROCONTROL).

Table 3.1: ENAC summary table on Outcome oriented SPI. (Tabella 1 – Indicatori operativi from [10])

Codice	Safety Performance Indicator (SPI)	Descrizione dell'indicatore	Origine dei dati
SPI-O-01	RE - Rateo di Runway Excursions	Number, every 10.000 movements, of occurrences involving a veer off or overrun off the runway surface	Sistema eE-MOR / Movimenti aeroportuali (fonte ENAC)
SPI-O-02	RI - Rateo di Runway Incursions	Number, every 10.000 movements, of occurrences involving the incorrect presence of an aircraft, vehicle or person on the protected area of a surface designated for the landing and take-off of aircraft	Sistema eE-MOR / Movimenti aeroportuali (fonte ENAC)
SPI-O-03	LOC-I - Rateo di casi di perdita di controllo in volo	Number, every 10.000 flights, of occurrences with loss of aircraft control while, or deviation from intended flight path, in flight	Sistema eE-MOR / Numeri di voli (fonte Eurocontrol)
SPI-O-04	TCAS – Rateo di Resolution Advisories (RA)	Number, every 10.000 flights, of TCAS Resolution Advisories following a TCAS activation	Sistema eE-MOR / Numero di voli (fonte Eurocontrol)

SPI-O-06	TAWS – Rateo di attivazione del TAWS	Number, every 10.000 flights, of Terrain and Avoidance Warning System activations	Sistema eE-MOR / Numero di voli (fonte Eurocontrol)
SPI-O-07	RAMP – Rateo di eventi di rampa (Rateo di eventi di rampa nei quali ci sia stato l'urto di un mezzo e/o di un'apparecchiatura con un aeromobile fermo al parcheggio)	Number of occurrences, every 10.000 movements, where a collision occurred while servicing, boarding, loading, and deplaning the aircraft	Sistema eE-MOR / Movimenti aeroportuali (fonte ENAC)
SPI-O-08	GCOL – Rateo di collisioni a terra (Rateo di collisioni, di mancate collisioni o conflitti di traffico che coinvolgano veicoli/mezzi ed aeromobili)	Number of occurrences, every 10.000 movements, where an aircraft comes into contact with another aircraft, a vehicle, a person, a structure, a building or any other obstacle while moving under its own power in any part of the airport other than the active runway, excluding power pushback	Sistema eE-MOR / Movimenti aeroportuali (fonte ENAC)
SPI-O-09	F-NI - Rateo di eventi classificabili come "Fire or smoke on aircraft"	Number, every 10.000 flights, where fire or smoke was detected on an aircraft, in flight, or on the ground	Sistema eE-MOR / Numero di voli (fonte Eurocontrol)
SPI-O-10	LASER - Rateo di interferenze di raggi laser con aeromobili durante le operazioni di volo	Number of occurrences, every 10.000 movements, in which a laser beam interfered with the flight operations of an aircraft taking off or landing	Sistema eE-MOR / Movimenti aeroportuali (fonte ENAC)

SPI-O-11	BIRD - Rateo di Bird/Wildlife Strikes (BRI - Bird Strike Index)	Si veda Circolare APT-01 (ultima revisione)	Sistema eE-MOR / Movimenti aeroportuali (fonte ENAC)
SPI-O-12	UPA - violazioni (airspace infringements) di spazi aerei controllati	Number of airspace infringements that occur when an aircraft enters notified airspace without previously requesting and obtaining clearance from the ATC or enters the airspace under conditions that were not contained in the clearance	Sistema eE-MOR
SPI-O-13	SMI - Separation Minimum Infringements	Numbers of occurrences, every 10000 flight hours, where prescribed separation between aircraft minima was not maintained	Sistema eE-MOR
SPI-O-14	ATM Failures - Numero di avarie gravi nei settori terminali ATM	Number of serious technical failures affecting the safe provision of air traffic services	Sistema eE-MOR
SPI-O-15	APR Interferences - Numero di interferenze di APR con aeromobili pilotati durante le fasi di decollo e/o atterraggio	Number of occurrences where an APR interferes with the flight of a manned aircraft during take-off or landing	Sistema eE-MOR

3.3 ENAC reporting system

Hazard identification and prevention are built on the effectiveness of the authority reporting system. The reported safety data and information are the foundation of safety analyses and investigations. ICAO describes in the *Safety Management Manual* [1] what is intended as a successful reporting system. Specifically, a reporting system is based on the reciprocal and continuous exchange of information between organizations and individuals and the protection of those information and the related sources is fundamental to maintain the operator's trust in the system.

The operator is more likely to report hazards and errors when protected and treated in a fair and consistent manner guaranteeing continuity of information.

Until January 1st, 2022¹, ENAC had been using the eE-MOR system in reporting. The acronym eE-MOR stands for *electronic ENAC - Mandatory Occurrence Reporting*, and it consists of the combination of WebDAS interface and ECCAIRS 5 software to manage the reporting dataset at European level [31]. The interface allows the users to enter the data in the system. The software is provided by the Joint Research Center of the European Commission [32]; the center supports EU policies with independent scientific evidence throughout the whole policy cycle [33]. Inside the software, the definitions of the reported events follow ICAO guidelines defined in the ADREP Taxonomy, which is a compilation of attributes and the related values [32][34].

Considering European and Italian regulations on reporting, *Regulation (EU) No 376/2014* handles reporting, analysis and follow-up of occurrences in civil aviation. In addition to general instructions and safety manners, it defines what is considered as *Mandatory reporting* (art. 4) and how it is required to be done. In particular, an *occurrence* is defined as "any safety-related event which endangers or which, if not corrected or addressed, could endanger an aircraft, its occupants or any other person and includes in particular an accident or serious incident". Specifically, the *Regulation (EU) 2015/1018* lays down a list classifying occurrences in civil aviation to be mandatorily reported according to *Regulation (EU) No 376/2014*. When an occurrence happens, the authorized personnel must report the event to the related organization, who has 72 hours to inform the competent authority [35]. All ENAC reports are performed through this system.

Regulation (EU) No 376/2014 has been revised in 2020 with the publication of *Regulation (EU) 2020/2034*, that will enter in service in 2023.

As far as Dangerous Goods reporting is concerned, during Chicago Convention in 1944 ICAO released Annex 18, named '*The Safe Transport of Dangerous Goods by Air*', which represents the international standard; it was followed by the Doc 9284 presenting specific transporting instructions. In 1948 Italy subscribed to the ICAO Annexes and from 1997 ENAC has been in charge of providing technical regulations in Civil Aviation at Italian level, according to Decreto Legislativo n. 250. Since 2008, following *Regulation (EC) No 216/2008* European standards have been EASA responsibilities, in order to establish and maintain a high uniform level

¹From January 2022, ENAC has started using ECCAIRS 2 system in reporting, but this change is not part of this thesis [30]. Dangerous Goods are not subjected to this change.

of civil aviation safety in Europe [36].

Details for the following analysis

In order to understand the following analysis on Dangerous Goods, it is important to define two terms which are used in ENAC reporting system. The first one is *Event type*, which means the general happening while an occurrence arises; for instance when the package of DG is not compliant to the packaging rules, the corresponding event type may be 'Dangerous Goods Labelling/Marking'. The second one is *Dangerous Goods type*, which indicates the kind of substance that is discovered, and its definition follows ICAO indications. These two terms are the two main topics on which the data analysis is based.

The term *Event type* is relevant also to the SPI analysis.

In addition, in the definitions of SPIs in section 3.2, there are three terms that need to be defined because they are involved in the following analysis on this topic. The first one is *movement*, which means "an aircraft take-off or landing at an airport; for airport traffic purposes one arrival and one departure is counted as two movements" [37]. The second one is *(domestic) flights*, which indicate "all flights of national or foreign aircraft in which all the airports are located in the territory of the same State; in both cases the flight shall be considered as the operation of an aircraft on a stage or number of stages with an unchanging flight number" [37]. The third one is *occupancy duration/flight hours*, which illustrates the occupancy time in the sky [9].

Chapter 4

Python-based Pipeline: Dangerous Goods

The development and the results of the Python-base pipeline are the core of this project. In this chapter and in the following one, every step of the building process is described highlighting the issues, the reasons behind specific decisions and the advantages of this approach. The theme here is the analysis on Dangerous Goods, divided in Event types analysis and DGs types analysis. The process starts from data-sets provided by ENAC, proceeds with data evaluation and ends with tabular and graphical results. In Figure 4.1, a flowchart is presented to illustrate the logical process of the analysis.

4.1 Python

The programming language that has been used to carry out the analysis is **Python 3**. Python is "an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together" [38].

It is known for its simplicity and appeal, since the focus is on readability. This reverberates on a less steep learning curve (compared to other programming languages) and in lower cost of maintenance, in term of overall resources required. Moreover, Python is open source, therefore all the tools necessary to our analysis are available to everybody for free. It also supports the use of custom made modules, which inherently allows a modular code design and makes easier to reuse and readapt already existing code to new purposes.

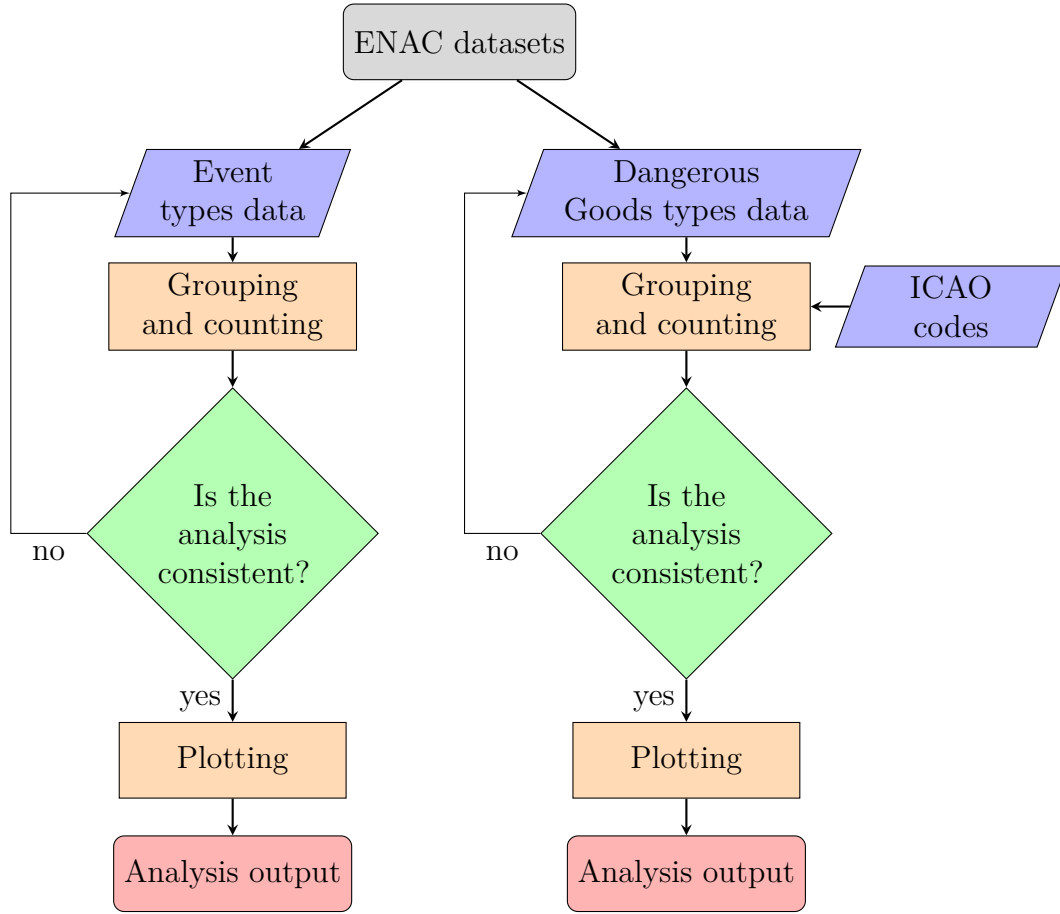


Figure 4.1: Flowchart of the Dangerous Goods analysis.

Traditionally in aviation the most used tool is **Microsoft Excel** (MS Excel) because of it is widely employed and easy to use for basic tasks. MS Excel is a spreadsheet-based software developed by Microsoft that offers calculation and graphing tools, pivot tables, and a macro programming language called Visual Basic for Applications (VBA)[39]. Although MS Excel is appealing for its advantages in terms of simplicity of use, a more versatile tool like Python offers more opportunities both in term of quality and performance. So Python has been chosen for this thesis, with the explicit goal to provide a different and improved approach to the current state of safety aviation data analysis.

In this thesis we mainly used the following Python software libraries: *NumPy* [40], *pandas* [41], *matplotlib* [42] and *seaborn* [43]. NumPy is a scientific computing package, offering a fast and versatile arrays computation and a wide range of numerical tools; Pandas is a package specifically designed to work with tabular data;

Matplotlib is a library for creating static, animated, and interactive visualizations, it allows the user to create quality plots and interactive figures but also to customize them entirely and export them. Lastly, Seaborn is a data visualization library well integrate into the NumPy and Matplotlib environment that provides a high-level interface for drawing attractive and informative statistical graphics.

The base of this work is a Python script, run through Jupyter Notebook by Anaconda, which contains all the necessary steps to run the data analysis. The notebook allows the user to create and edit documents that display cell by cell the input and output of Python scripts. Hence the final product of the project is a Jupyter notebook file with `.ipynb` extension.

4.2 Input data

The input files are provided by ENAC as Excel files. These files are the collection of all the reports and are subordinated to a manual intervention done by agency's operators. Any manually changes or adjustments are not subjects of this project, in fact the actual starting point is the Excel files obtained following ENAC supervision. Input files can be divided in a general file and some attached files. The first one contains all general data in terms of Event types and Dangerous Goods types, whereas the attached files concern multiple reports of DG types, that will be explained later. All input data is included in a folder, named '`Dangerous_Goods`', which itself contains two folders containing the attached files, named '`2020`' and '`2021`', respectively. In order to simplify the analysis and the logical process behind, the analysis is divided in two sectors: *Event types* and *Dangerous Goods types* (see section 3.3), therefore also the explanation process is divided just the same.

4.2.1 Excel Files

For both the considered sectors, we are presenting how the Excel files are assembled and which kind of data we are dealing with. First of all, the available time period is from 2014 to 2021. In the general file, a sheet is assigned to each year. Over the course of the years the reporting system has experienced changes and improvements, thus the data has also experienced different frameworks. With that in mind, it is easy to identify two scenarios: from 2014 to 2019 and from 2020 to 2021. In the first six-years period data is minimal, no attached files exist and the quantity of reports is limited, whereas in the last two-years period data is more complex and extended and attached files need to be taken into account.

In this project we are handling just the minimum amount of data required to conduct the analysis due to data classification restriction, which is not subject of

this thesis.

In the interest of understanding fully the input data, an example for each scenario is presented in Figures 4.2 and 4.3. In addition, in Figure 4.4 an example of attached files data is given.

Source [452]	UTC date [477]	Dangerous Good [688]	Event Type [390]
0000	dd/mm/yyyy	1845 - Carbon dioxide, solid	Dangerous Goods Handling and Loading Return to Stand

Figure 4.2: Example of input data for the time period 2014-2019. ('Source' and 'UTC date' are classified and indicated generally.)

Source [452]	UTC date [477]	Dangerous Good [688]	Event Type [390]	count
0000	dd/mm/yyyy	3480 - Lithium ion batteries 3481 - Lithium ion batteries contained in equipment	Dangerous Goods Handling and Loading	1

Figure 4.3: Example of input data for the time period 2020-2021. ('Source' and 'UTC date' are classified and indicated generally.)

Type of Goods
1950 - Aerosols, flammable

Figure 4.4: Example of input data in the attached files.

Event types

The data regarding Event types is defined in both figures 4.2 and 4.3 as '*Event Type [390]*' and only that column will be necessary to be proceeded.

Dangerous Goods types

In this part of the analysis data interconnections may be complex. First of all, the considered columns in Figures 4.2 and 4.3 are '*Source*', '*Dangerous Good [688]*' and '*count*', and also figure 4.4 is taken into account (only for 2020 and 2021). In particular, the first column represents the identification number of that report and the third column represents the number of DG types.

For the period 2014-2019, we extract the information from the general file according to the mentioned columns. For the years 2020 and 2021, when the '*count*' column contains a number equal to 1, the involved DG types data is included only in the '*Dangerous Goods [688]*' column. On the other hand, when the '*count*' column contains a number higher than 1, the involved DG types data is included in an attached file whose name is defined in the '*Source*' column. Lastly, when the '*count*' column is equal to 0 the report is not valid, so it does not contribute to the analysis¹.

4.2.2 Reading files

In order to use the data, it is necessary to read the Excel files and save the information. An efficient way to do that is to implement *pandas* reading commands². We provide an example used in the code to save 2014 data in a DataFrame³, accessing the general Excel file:

```
data_2014=pd.read_excel('Dangerous_Goods/DG_2021.xlsx',
sheet_name=years[0])
```

The command is defined by many parameters, whose that can be read above are the source file with its directory and the specific Excel sheet to consider. For each year (from 2014 to 2021) a different DataFrame is created (they will be identified as the 'original DataFrames' later). The resulting output is a sort of a table, that can be accessed by indexes and labels for rows and columns. This is really useful when extracting specific part of the DataFrame is required, as a matter of facts when we proceed with Event types analysis and DGs types analysis we define new DataFrames including just the necessary columns from the original data-sets.

The method for reading the attached files is the same, however the reading process

¹The reason why some reports are not valid needs to be addressed to ENAC.

²Ref. to code lines 18-25 in Appendix A.

³A DataFrame is a two-dimensional pandas structure containing labeled axes (rows and columns) [44].

is dynamic and it will be describe in section 4.4.1.

In reference to the code, it is shown how data is extracted for the different analyses⁴. In particular, Event types analysis needs column number 7 (titled '*Event Type [390]*') of the original DataFrames, whereas DGs types analysis needs columns number 0 (titled '*Source*'), 6 (titled '*Dangerous Goods [688]*') and 8 (titled '*count*')⁵.

Ultimately, it is important to underline that both the general file and the attached files contain empty columns due to data restrictions, even though in the analysis this detail has been considered in order to generalize the study, in Figures 4.2, 4.3 and 4.4 it is not shown.

4.3 Event types

The process behind Event types analysis starts with the identification of all the Event types categories that can be used in reporting, then it follows with the counting of them in the different years and it ends with the grouping into specific macro-groups.

The first part consists in reading the specific column for every year and append them together in order to create a list of every present item without the need to insert them manually⁶. At this point, it is important to clear this list deleting duplicates and removing special characters. In addition, it can happen that in the Excel file more than one item is written in a single cell, so it is fundamental to split them so they can be counted individually. The used command is `pandas.Series.str.split('\n')`. It allows the user to divide words that are linked with the specified string in brackets, which in this case represents the *Enter* command. Even though this issue can be solved easily, it can be problematic when checking if the total number of event types extracted is equal to the total number of event types reported, this matter will be addressed in section 4.3.2.

Now, using the `chain` command from *itertools* package, we are able to create a list and then we transform it in a *pandas* Series. In addition, after gathering those multiple types individually, it is required to clean further the data, deleting new duplicates.

As a result, the list of every Event types used in reporting is:

⁴Ref. code lines 37-41 and 377-415 in the Appendix A.

⁵DataFrame's indexes start as 0,1,..

⁶Ref. code lines 45-50 in Appendix A.

- Action Performed Incorrectly
- Aerodrome Emergency or Fire Services Deployed
- Baggage Non-Compliant Carriage of Load
- Baggage Security Check
- Cargo Labelling/Marking
- Crew Door Fails to Open/Close
- Dangerous Goods Damaged
- Dangerous Goods Exceeds Storage Compartment Limitations
- Dangerous Goods Forbidden
- Dangerous Goods Handling and Loading
- Dangerous Goods Labelling/Marking
- Dangerous Goods Leaking
- Dangerous Goods Load Weighting
- Dangerous Goods Loading/Unloading
- Dangerous Goods Not Matching Documentation
- Dangerous Goods Not Recorded
- Dangerous Goods Packaging
- Dangerous Goods Security Check
- Dangerous Goods Undeclared
- Dangerous Goods Unsecure without Shift
- External Load - Release
- Fire - Lithium Battery
- Fumes - Passenger Baggage
- Helicopter RPM Exceedance
- Lack of Communication

- Passenger Carry-On Baggage
- Qualifications
- Return to Stand
- Smell - PEDs
- Smoke - Lithium Battery
- Smoke - PEDs
- Smoke or Fumes in Aircraft
- Use of Emergency Equipment.

At this moment, we can proceed with the second part of the analysis⁷. It is necessary to count how many times each event type in the above list can be found in each considered year using the *pandas* `Series.value_counts()`. In particular, for years 2020 and 2021 we have to make sure to exclude those reports with *'count'* equal to 0. So it is possible to create a table with event types as rows and years as columns, as can be found in the code's outputs.

	Years
Event types	(counts)

Table 4.1: Table model of Event types counting output.

Before proceeding, we have to motivate the operation placed at line 27 in Appendix A. During the development process it was found out that in 2014 data the description of the event type 'Dangerous Goods Leaking' was in fact 'Dangerous Goods Leaking related event'. This discrepancy interfered with the process explaining in this section, because it looked like a different event type. Since this problem was registered only in this case, we decided to change the single datum, in order to make the code works and the analysis accurate.

4.3.1 Grouping

It is clear that studying such a long list could be chaotic, so a good way to reduce the amount of items is to group them together following a certain rule⁸. In this

⁷Ref. code lines 54-72 in Appendix A

⁸Ref. code lines 77-136 in Appendix A.

specific case, the macro-groups have already been defined by a previous examination by ENAC, so we are going to list them and identify the grouping method.

The macro-categories can be listed as:

- Dangerous Goods Undeclared
- Dangerous Goods Forbidden
- Dangerous Goods Labelling/Marking
- Dangerous Goods Leaking
- Dangerous Goods Damaged
- Dangerous Goods Packaging
- Dangerous Goods Handling and Loading
- Dangerous Goods Not Matching Documentation
- Smoke/Fumes/Fire Li-BAT
- Other.

Some items, such as *Dangerous Goods Undeclared*, *Dangerous Goods Forbidden*, *Dangerous Goods Labelling/Marking*, *Dangerous Goods Leaking*, *Dangerous Goods Damaged*, *Dangerous Goods Packaging*, are the same categories as considered before, whereas the other items are a combination of more than one event type.

Dangerous Goods Handling and Loading item contains:

- Dangerous Goods Handling and Loading
- Dangerous Goods Loading/Unloading
- Dangerous Goods Load Weighting
- Dangerous Goods Exceeds Storage Compartment Limitations
- Dangerous Goods Unsecure without Shift.

Dangerous Goods Not Matching Documentation item contains:

- Dangerous Goods Not Matching Documentation
- Dangerous Goods Not Recorded.

Smoke/Fumes/Fire Li-BAT item includes every event type which contains the words 'Smoke', 'Battery', 'Fumes' or 'Smell' and lastly *Other* item contains every other event type not mentioned above.

Particular emphasis should be made for the last item, *Other*, because it contains many different event types that sometimes are not consistent with the analysis. It can happen that, in reporting, operators do not know exactly the correct type to use, so a certain amount of event types should be removed in the interest of coherence. In this project, ENAC identified those that are not suitable: Action Performed Incorrectly, Baggage Non-Compliant Carriage of Load, Baggage Non-Compliant Carriage of Load, Baggage Security Check, Cargo Labelling/Marking, Crew Door Fails to Open/Close, External Load - Release', 'Helicopter RPM Exceedance, Lack of Communication, Passenger Carry-On Baggage, Qualifications, Return to Stand, Use of Emergency Equipment.

After due consideration, it is possible to obtain a comprehensive table where for each year the total counts of each macro-group is presented, as shown in Table 4.2.

4.3.2 Checks and validation

Data analysis results could look correct at first glance but they may hide some errors that can be prevented implementing some checks⁹. In this case, the control aims at comparing the total amount of event types extracted and the total amount of event types in the Excel file for each considered year.

The first one can be easily calculated summing each column of the general Table 4.1, whereas the second needs several steps. Firstly, we check the length of the Excel file for each year measuring the length of the original DataFrames, so we get the total number of reports (number of rows), but it can happen that more than one event type is written in a single cell. An example is provided in Figure 4.5.

So, secondly, we have to calculate the additional event types that are in single cells. To do that, we check every row in the original DataFrames and count how many times the *Enter* command is present. When breaking lines, Python registers `\n`, so if in that row there is one it means there are two event types together; if there are two it means there are three event types, and so on. However, sometimes if, in the Excel file, the *Enter* command had been used twice consecutively, Python registers `\n\n`, so we need to verify that reading two `\n` means either three event types or two event types.

⁹Ref. code lines 141-196 in Appendix A.

	2014	2015	2016	2017	2018	2019	2020	2021
Dangerous Goods Undeclared	2	3	4	3	23	55	96	255
Dangerous Goods Forbidden	1	0	1	1	4	6	1	0
Dangerous Goods Labelling/-Marking	0	0	0	0	2	6	1	5
Dangerous Goods Leaking	1	0	1	0	2	3	0	2
Dangerous Goods Damaged	0	0	2	1	5	2	2	0
Dangerous Goods Packaging	0	0	0	0	0	0	2	2
Dangerous Goods Handling and Loading	0	5	2	3	2	15	5	5
Dangerous Goods Not Matching Documentation	1	0	0	1	0	4	6	0
Smoke-Fumes-Fire Li-BAT	0	0	2	2	1	3	2	2
Other	0	0	0	0	0	3	0	0
Total	5	8	12	11	39	97	115	271

Table 4.2: Resulting table of Event types analysis.

Source [452]	UTC date [477]	Dangerous Good [688]	Event Type [390]
0000	dd/mm/yyyy	1415 - Lithium	Dangerous Goods Loading/Unloading Dangerous Goods Undeclared Smoke - Lithium Battery Fumes - Passenger Baggage

Figure 4.5: Example of multiple Event types in a single Excel cell. ('Source' and 'UTC date' are classified and indicated generally.)

The script includes some cases, however if a different multiple event types arrangement happens, Python provides a warning: 'Check the count!'.

The counting validation can be done comparing the sum of the total reports with the additional event types and the total extracted amount. If these two numbers do not correspond, the code will provide a warning: 'Missing data!'; in this scenario the user needs to take action. Additionally, as Figure 4.6 is showing, the code provides a yearly report of the numbers previously calculated to inform the user.

```
Total number of events in 2019: 103
Total number of reports in 2019: 87
Total additional event types: 16
2 event type: 6
3 event type: 1
4 event type: 1
5 event type: 0
6 event type: 1
```

Figure 4.6: Python report for 2019 on analysis validation.

In reference to Figure 4.6, the third line correspond to the linear combination of the rows below. In this specific case: $Total\ additional\ event\ types = (2\ event\ type)*1 + (3\ event\ type)*2 + (4\ event\ type)*3 + (5\ event\ type)*4 + (6\ event\ type)*5$.

4.4 Dangerous Goods types

The Dangerous Goods types analysis requires a different approach than the one used for Event types analysis, because the data is distributed differently and it demands an alternative manipulation. At first, we tried to implement a similar method listing all the individual DG types used in reporting. However this method was not beneficial because, in reporting, the way DG types are defined is not unique. To each UN code can correspond multiple descriptions, as shown in Figure 4.7. As you can see, the opening numerical part is the same, but the following description has variations, so these two DG types look different but they are exactly the same.

Source [452]	UTC date [477]	Dangerous Good [688]	Event Type [390]
0000	dd/mm/yyyy	3481 - Lithium ion batteries contained in equipment	Dangerous Goods Forbidden
0000	dd/mm/yyyy	3481 - Lithium ion batteries packed with equipment	Dangerous Goods Labelling/Marking

Figure 4.7: Example of different descriptions for the same DG type from 2019 data. ('Source' and 'UTC date' are classified and indicated generally.)

With this in mind, the new approach plans to exclude the descriptions and analyze the data relying just on the UN codes. Unfortunately, there are some exceptions that emerge during the analysis: not every report indicates the UN code, sometimes there can be the class number or no numbers at all. In Figure 4.8 some examples are displayed. These cases need to be managed individually, as shown in section 4.4.2.

Source [452]	UTC date [477]	Dangerous Good [688]	Event Type [390]
0000	dd/mm/yyyy	Battery, wet, filled with acid	Dangerous Goods Leaking
0000	dd/mm/yyyy	1 - Explosives	Helicopter RPM Exceedance Aerodrome Emergency or Dangerous Goods Not Matching Documentation
0000	dd/mm/yyyy	8000 - Consumer commodity	Dangerous Goods Loading/Unloading
0000	dd/mm/yyyy	0 - Wheelchair, electric with batteries	Dangerous Goods Forbidden
0000	dd/mm/yyyy	2.1 - Flammable Gas	Dangerous Goods Loading/Unloading
0000	dd/mm/yyyy	Not defined	Dangerous Goods Loading/Unloading

Figure 4.8: Example of exceptions in DG type definitions from 2019 data. ('Source' and 'UTC date' are classified and indicated generally.)

4.4.1 Gathering all data

Before proceeding with the analysis itself, it is necessary to import the required data¹⁰. This process needs to be diversified for 2014-2019 data and 2020-2021 data, because in the second scenario the attached files need to be read. What is common for both scenarios is defining new DataFrames extracting the required columns from the original DataFrames (see section 4.2.2), even though the required columns are different, as explained later, we unified the script for every year.

From 2014 to 2019

Regarding just data of years from 2014 to 2019¹¹, we need to consider just the '*DG types*' column and we have to manipulate it in order to get only UN codes and their counts for every year.

Firstly, we list all the DG types (in a year) paying attention to those that could be written together in a single cell, so we use the *pandas* split command and obtain a *pandas* Series. It is important to clean the data dropping NaN values, replacing special characters and sorting the list. Secondly, for each row of the series we split it at every blank space and we keep only the first part, which corresponds usually to the UN code. Then, using the *pandas* `Series.value_counts()` command we count how many times a UN code occurs yearly.

The final outputs are six DataFrames containing reported UN codes and their counts.

2020 and 2021

For both the considered years, the process is divided in two parts relating to the number in the '*count*' column. As a matter of fact, when extracting data we filter the rows in order to get the cases separated. When the '*count*' column is equal to 1, the process is the same explained for the 2014-2019 scenario, resulting with two *pandas* Series containing reported UN codes. When the '*count*' column is more than 1, the process changes because of the attached files. In this case, the required data is no longer read from the general file but from the specific appended file, so it is required to read also the '*source*' column (from the general file) which contains the relating appended file name.

As far as 2020 data is concerned¹², for each row with '*count*' greater than 1,

¹⁰Ref. code lines 377-415 in Appendix A.

¹¹Ref. code lines 426-471 in Appendix A.

¹²Ref. code lines 476-509 in Appendix A.

we identify the corresponding file name that is composed by its directory and the wording of the `'source'` column. For instance, assuming the file name is `'0000'`, `'Dangerous_Goods/2020/0000.xlsx'`. At this point, for every considered report we read the specific file and we add all the DG types extracted together in a list, then we split every row at every blank space and keep the first part, doing so the result is a *pandas* Series containing the reported UN codes.

In this code section we add a validation check that will be useful later, in section 4.4.4. Every time we read a file, we check that its length corresponds to the number in the corresponding `'count'` column. Those two numbers should be the same, but sometimes we can find some mistakes. In this regard, the Python code provides a warning when finding inconsistencies, as shown in Figure 4.9.

The attached file  contains 42 instead of [41] reported

Figure 4.9: Code output of the validation check on attached files' length from 2020 data.

As far as 2021 data is concerned¹³, the method is almost the same used for year 2020, the only difference is defining the attached files names. In this case, looking at the files in the `'2021'` folder the names have variations, so using the `'source'` as the name is not enough. For each row with `'count'` greater than 1, we define different possible file names, based on the options in the folder. There are six possibilities as following, assuming the source code is `'0000'`:

- `'Dangerous_Goods/2021/0000.xlsx'`
- `'Dangerous_Goods/2021/eE-MOR 0000.xlsx'`
- `'Dangerous_Goods/2021/EE-MOR 0000.xlsx'`
- `'Dangerous_Goods/2021/0000 (modificato).xlsx'`
- `'Dangerous_Goods/2021/EE-MOR_0000.xlsx'`
- `'Dangerous_Goods/2021/eE-MOR 0000 .xlsx'`.

At this point, we verify which of these possibilities exists in the directory, using `os.path.isfile()` command and then we read the file, extracting the DG types and listing them in a *pandas* Series.

Regarding the validation check on the attached files length, the process and the Python warning are the same explained for year 2020. This time the errors are

¹³Ref. code lines 513-568 in Appendix A.

shown in Figure 4.10.

```
The attached file ██████████ contains 4 instead of [6] reported
The attached file ██████████ contains 25 instead of [32] reported
```

Figure 4.10: Code output of the validation check on attached files' length from 2020 data.

Lastly, for both years, we append the two obtained Series (one for 'count' equal to 1 and one for 'count' greater than 1) resulting in a Series containing all reported UN codes, and using *pandas* `Series.value_counts()` we collect the final counts.

It is possible now to join all the data and create a general DataFrame with all years¹⁴. To do so, we use *pandas* `concat()` command, filling with '0' when some codes are not included in a year.

The result is displayed in Table 4.3.

Table 4.3: Intermediate resulting table of Dangerous Goods analysis.

	code	2014	2015	2016	2017	2018	2019	2020	2021
0	0	1	0	1	0	1	1	0	25
1	1	0	0	0	0	0	1	0	1
2	1033	0	0	0	0	0	0	1	0
3	1044	0	0	0	0	1	4	3	0
4	1050	0	0	0	0	0	1	0	0
...
86	9	0	0	0	0	3	4	0	2
87	Battery,	0	0	0	0	0	1	0	0
88	Not	0	0	0	0	0	4	0	1
89	UNK	0	3	3	1	0	0	0	0
90	radioactive	0	0	0	0	0	0	0	1

¹⁴Ref. code lines 651-655 in Appendix A.

4.4.2 ICAO codes

After gathering all the data and the counts for each year, it is important to underline that every Dangerous Goods type is defined by its code. However, this is not a good way to visualize results because of the amount of data, so we will group them into classes defined by ICAO (see section 3.1.2). The grouping process will be addressed in the following section, whereas now we focus on how to match codes and classes. In particular, we consider the main classes and not the sub-classes (divisions).

First of all, we need to create a list with all the possible combinations of code-class and this can be done reading a specific document published by ICAO, *Technical Instructions for the Safe Transport of Dangerous Goods by Air* [12]. There, from page 2 to 234 (the pdf source 'doc_9284_2019_2020' is a partial extraction of the manual) there are tables containing the searched relation. Then, thanks to *pandas* reading command we can read all those pages and obtain the resulting table, shown in Table 4.4. The reading process, however, is not simple because Python acquires the data in a odd way, so some manipulation is needed¹⁵. Using *tabula* package we can read the selected pages obtaining a table, then we have to clean the data depending on how Python acquires the pages. For instance, we have to divide codes and classes at \r character or deleting the lines named 'FORBIDDEN'. In addition, we want to keep only the class number ignoring the sub-class, since the sub-class number is reported as, for instance, '4.5', we delete any numbers after the point. It is important to note that not every page is acquired in the same way, so in the code there is a check that helps understanding if some necessary codes have not been acquired, thus the user can check if some pages have not been read correctly. As Figure 4.11 explains, Python provides a warning if there are some missing codes that are required in the analysis and in this case they will be displayed in order to the user to take action. The problem could be either a failure of the Python code or an outdated version of the ICAO document.

```
Codes not found in Icao document are:  
Series([], Name: code, dtype: object)  
Please check pdf reading or new versions of the document.
```

Figure 4.11: Python output for codes' check.

The misread pages are considered individually and then combined with the rest of

¹⁵Ref. code lines 658-703 in Appendix A.

the codes¹⁶; in particular, pages 130, 131 and 220.

The total number of codes obtained from the ICAO document is 2328.

codes	classes
1088	3
1089	3
1841	9
2332	3
2789	8
...	...
3125	6
3172	6
3462	6
0212	1
0306	1

Table 4.4: ICAO codes and classes.

A way to match codes and classes is to define a Python function that given an input code, it attributes the corresponding class, according to Table 4.4. The function used is defined as following:

```

1  def matching (x): #defining a function to match from the code the
    corresponding class
2  if codes_icao['codes'].str.contains(x).any(): #verifying if the
    string exist in the icao pdf
3
4      for i in range(0, len(codes_icao['codes'])):
5          if (x==codes_icao['codes'][i]):
6              match=codes_icao['classes'][i]
7  else:
8      match='Notfound'
9  return match

```

¹⁶Ref. code lines 667-679 in Appendix A.

In the above code, x is the input code, *code icao* is the DataFrame corresponding to Table 4.4, whereas *match* is the resulting class.

As the input codes are various¹⁷, we have to divide the different options when matching.

Input code

4 digits → use matching function

1 digit → keep it as resulting class

There are then some specific cases that need to be treated individually. For example, the next rows explain how items are processed:

Input word

'Battery,' → class 9

'Not' → ND (Not Defined)

'UNK' (Unknown) → ND (Not defined)

'radioactive' → class 7

As a result it is possible to create a table that combine yearly counts and ICAO classes, as shown in Figure 4.5.

	Years	
Codes	(counts)	Classes

Table 4.5: Table model of DG types counting and matching output.

¹⁷This diversity originates when splitting the DG types in order to get the UN codes, but not all the types are alike (see section 4.4), so sometimes the extracted first part of the type is not a 4 digit string.

4.4.3 Grouping

The grouping process is similar to what explained for Event types analysis: defined some macro-groups we fill them in with all the previous types¹⁸. This time, the macro-groups are already clarified by ICAO (see section 3.1.2) and also ENAC has introduced a request. In particular, it was required to split class 9 into two different groups: *Miscellaneous - Li-BAT* and *Miscellaneous - other than Li-BAT*. Particular emphasis should be made for class 9 grouping, as matter of facts it is not automatic but it needs the user intervention. From a previous examination by ENAC, the division of the two class-9 groups was defined and the considered codes were listed:

<i>Miscellaneous - other than Li-BAT</i>	<i>Miscellaneous - Li-BAT</i>
1845, 2807, 3077, 3082, 3316, 3363, 8000, 9, 2990, 3166	0, 3090, 3091, 3171, 3480, 3481, Battery,

Table 4.6: Division of codes into class 9 groups.

In the interest of making the process more reliable, the Python code provides a control whether some class 9-codes are not assigned to one of the two groups¹⁹. In this way the user can decide which class is more appropriate and add the new code to that class.

Lastly, there is an additional class, named *ND: Not Defined*, which contains DGs defined as 'Not Defined' and 'UNK'(Unknown) and also the number of empty cells existing in the Excel files (see section 4.4.4).

As a result, the Table 4.7 is the expected output.

¹⁸Ref. code lines 747-811 in Appendix A.

¹⁹Ref. code lines 789-795 in Appendix A.

	2014	2015	2016	2017	2018	2019	2020	2021
class 1: Explosives	0	0	0	0	0	1	0	1
class 2: Flammable Gases	0	1	0	1	3	15	375	336
class 3: Flammable Liquids	1	0	0	1	13	33	792	687
class 4: Flammable Solids	1	0	0	1	1	2	2	3
class 5: Oxidizing	0	0	0	0	2	0	1	1
class 6: Toxic and Infectious	0	0	0	1	0	3	4	3
class 7: Radioactive	0	0	2	2	1	3	1	3
class 8: Corrosives	0	0	0	0	4	12	13	16
class 9: Miscellaneous - other than Li-BAT	0	2	0	1	7	10	6	14
class 9-Bat: Miscellaneous - Li-BAT	3	2	6	2	7	10	171	252
ND: Not defined	0	3	3	1	0	5	1	1
Total	5	8	11	10	38	94	1366	1317

Table 4.7: Resulting table of Dangerous Goods types analysis.

4.4.4 Checks and validation

As well as for Event types analysis, also in this analysis it is necessary to check if there are any errors in the data results²⁰. The calculated values for each year are the total amount of DGs types extracted, the total amount of reports and the additional multiple types, besides the number of blank cells and the miscounts. The total amount of DGs types extracted comes from the sum of each columns of Table 4.7.

The total amount of reports is the length of the general Excel file or, for 2020 and 2021, of the collective attached files; to calculate it we sum the *'count'* column yearly (for years 2014-2019, the *'count'* columns was set to 1).

The additional types are the amount of multiple DGs types that are written in a single cell; we proceed in the same way done before: reading each row and detecting how many \n there are and thus how many additional DGs types.

Concerning the sum of blank cells, it can happen that some reports have empty cells corresponding to the searched data and we find them using *pandas Series.isna()* command. Counting them is important in order to make the numbers work because these empty cells are counted also as number of reports.

Lastly, with the term *'miscounts'* we imply possible errors in the Excel file *'count'* column. As a matter of facts, the number reported in that column should be equal to the length of the corresponding attached file, but this is not always the case. The calculation has been conducted in section 4.4.1.

As shown in Figures 4.12 and 4.13 Python reports the calculated numbers for each years, adding the miscounts for 2020 and 2021. In case the figures don't add up, the code provides a warning: *'Missing data!'*. The third row is the linear combination of the rows below, as indicated before in section 4.3.2, in Figure 4.6.

```
Total reports of Dangeruos Goods in 2019: 87
Total number of dangerous goods extracted in 2019: 93
Total additional DG types: 7
2 DG type: 3
3 DG type: 2
4 DG type: 0
5 DG type: 0
6 DG type: 0
Blank cells: 1
```

Figure 4.12: Python report for 2019 on analysis validation.

²⁰Ref. code lines 573-648 in Appendix A.

```
Total reports of Dangeruos Goods in 2020: 1356
Total number of dangerous goods extracted in 2020: 1365
Total additional DG types: 9
2 DG type: 6
3 DG type: 0
4 DG type: 1
5 DG type: 0
6 DG type: 0
Blank cells: 1
Number of missing counts:[1]
```

Figure 4.13: Python report for 2020 on analysis validation.

Since the code found out some errors in 2020 and also in 2021, ENAC has been informed and the errors have been corrected. The results presented in chapter 6 consider the adjustments.

Chapter 5

Python-based Pipeline: Safety Performance Indicators

The theme is the analysis of Safety Performance Indicators - outcome oriented. It aims at creating a template that can be used for any SPI study. In ENAC Safety Report Portal the displayed parameters are fourteen and they are mentioned in chapter 2. Following, we focus on two study cases, however the Python code can be run for any of the SPIs-O included in the Safety Report Portal. The chosen samples are *Runway Excursions* and *Runway Incursions*; the first one is characterized by a type of analysis that is common between many other SPIs, whereas the second one includes a further examination about Event types. In section 5.2, the whole code will be explained even though the graphical outputs in chapter 6 are only about the two cases mentioned above. Then, specifically in section 5.3 there are references to the Event types analysis.

Figure 5.1 presents the flowchart regarding the analysis.

5.1 Input data

The input data come from Excel files, one for each requested SPI. Every file contains as many sheets as the total number of considered years. In particular, the analysis covers a time-period from 2015 to 2020.

Generally, every file contains the data about the occurrences of the specific SPI, in Figure 5.2 an example of an Excel file is presented. The second studying case contains also data about Event types, as shown in Figure 5.3. The general data on occurrences and the data about Event types are assigned to different columns,

therefore the coexistence of them doesn't compromise one analysis or the other.

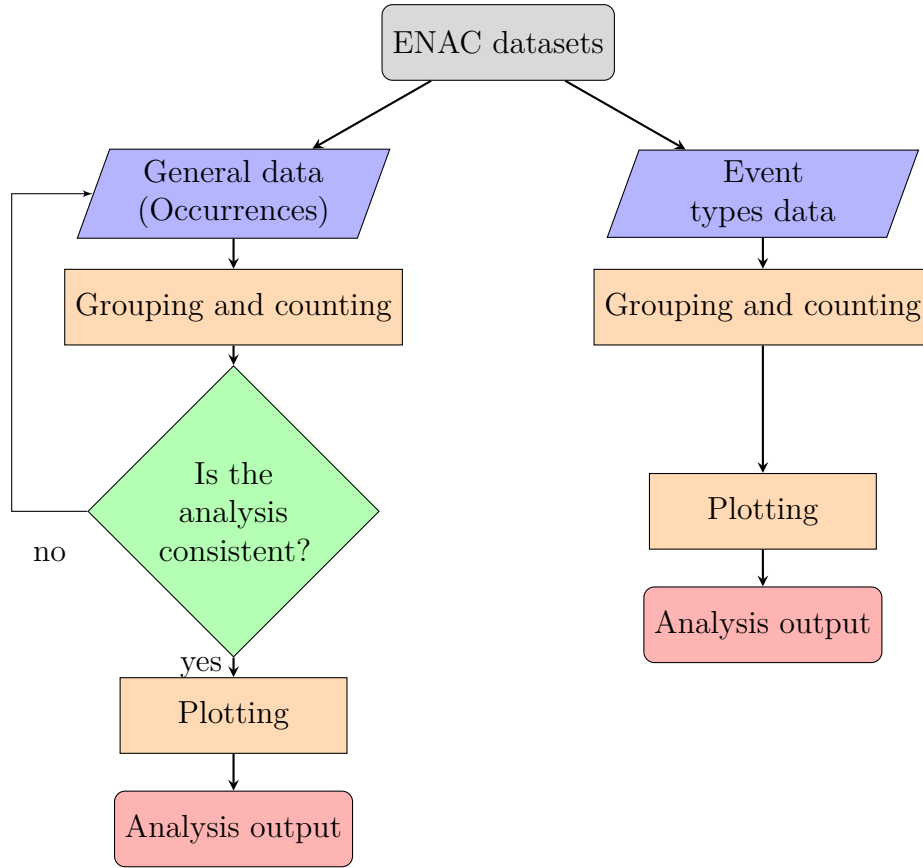


Figure 5.1: Flowchart of the Safety Performance Indicators analysis.

UTC date	UTC time	File number
dd/mm/yyyy	00:00	xxxx

Figure 5.2: Example of general input data of SPIs. (Sensible data is classified and indicated generally.)

UTC date	UTC time	File number	Event type
dd/mm/yyyy	00:00	xxxx	Incursions

Figure 5.3: Example of input data of SPIs with Event types data. (Sensible data is classified and indicated generally.)

In Figure 5.2, there are three filled columns, but in the analysis only the column named '*UTC date*' is used. In Figure 5.3 there are four filled columns, but the necessary ones are those named '*UTC date*' and '*Event type*'.

The input files include empty columns due to data restrictions¹, even though it is not shown in Figures 5.2 and 5.3.

5.1.1 Reading Excel files

To conduct the analysis it is necessary to read the files²: we need to read one file for each run because the analysis of one SPI ends in itself. Since the code is generalized, it provides the reading of all Excel files even though we present only the first two cases with their Excel files.

So, through the use of a if-condition it is specified which Excel file is required to be read. In particular, at line 14 in the code, the user has to indicate the name of the considered SPI, then according to that choice, the code selects three specific variables: `subject`, which is the extended name of the SPI, `name`, which is the contracted name (used in naming output files) and `file_name`, which is the name of the Excel file input.

At this point, it is possible to read the file using `pandas read_excel()` command³. In particular, we read every sheet of the file (according to the considered years) and we append them together, saving all in a single DataFrame (following referred as the 'original DataFrame').

5.2 General template

The analysis starting point is the original DataFrame, where all data is contained. The only necessary column is the number 5, where the dates are indicated, because it allows us to diversify data relating to the year and to count the total number of reports of that year. In SPIs analysis, differently from DGs analysis, we study the total number of events yearly, so the total number of reports of a specific year, that is also equal to the length of the specific Excel sheet. When extracting the data from the original DataFrame, we also convert the data to date format and then we keep only the indication of the year; to do so we use the `pandas to_datetime().dt.year`.

From now on, we need to differentiate the analysis according to the definition of

¹Data restrictions have to be addressed to ENAC and they are not subject of this thesis

²Ref. code lines are 20-39 in Appendix B.

³Ref. code lines 58-68 in Appendix B.

the specific SPI, however the process has the same steps: counting the occurrences, calculating the relating rate (only for the first three groups) and creating a resulting DataFrame. The different categories are four and consist of the following SPIs:

- *spi_mvt* : RE, RI, TCAS, RAMP, GCOL, F-NI, LASER, APR_interference
- *spi_flt* : LOC-I, TAWS
- *spi_occ* : BIRD, UPA, ATM_failure
- *spi_dur* : SMI

The first category considers SPIs whose rate is defined by the number of movements⁴; the second one whose rate is defined by number of flights⁵; the third one whose rate is defined by the occupancy duration; the fourth one considers SPIs whose only output is the number of occurrences yearly. In order to divide the code lines respectively, if-conditions are used.

For each of the first three categories, we count the total number of events yearly using the *pandas* `Series.value_counts()` command, then we calculate the rate. Changing category, the denominator of the fraction assigned and the multiplication factor change. In particular, the first group is calculated every 10000 movements, the second one every 10000 flights and the third one every 1 million minutes. As an example we provide the calculation for the first category in equation 5.1.

$$rate = \frac{yearly_occurrences}{movements} * 10000 \quad (5.1)$$

The values for *movements* and *occupancy duration* are provided by ENAC, while the number of *flights* comes from EUROCONTROL.

The resulting DataFrame is presented in Table 5.1 and Table 5.2.

Similar resulting DataFrame can be obtained for the other two groups.

As far as the fourth category is concerned, since their output is only the number of occurrences, using the *pandas* `Series.value_counts()` command the code will count the number of events yearly. The tabular result is a DataFrame composed of just one column.

⁴See section 3.3 for the definition.

⁵See section 3.3 for the definition.

	Movements	Number of events	Rate
2015	1544643	12	0.08
2016	1624966	11	0.07
2017	1653242	16	0.10
2018	1722254	20	0.12
2019	1655381	18	0.11
2020	708602	8	0.11

Table 5.1: Resulting table of general SPI analysis on *Runways Excursions*.

	Movements	Number of events	Rate
2015	1544643	80	0.52
2016	1624966	72	0.44
2017	1653242	127	0.77
2018	1722254	180	1.05
2019	1655381	187	1.13
2020	708602	86	1.21

Table 5.2: Resulting table of general SPI analysis on *Runways Incursions*.

5.3 Event types template

Regarding Event types analysis, it is automatically run if the required data is available, that is when the *'Event type'* column is filled.

To implement the analysis, columns number 5 and number 12 of the original DataFrame are extracted⁶. The data about dates is converted in date format using *pandas to_datetime()* command.

Differently from the DGs analysis, in this case we consider only specific Event types to count, which are requested by ENAC. In particular, they are three:

- Runway Incursions by an Aircraft

⁶Ref. code lines 254-269 in Appendix B.

- Runway Incursions by an Person
- Runway Incursions by an Vehicle/Equipment.

It is important to underline that these mentioned event types refer only to the SPI of *Runway Incursions*, but the code can work for any event type used in reporting, after appropriate changes.

At this point we count how many times the specific types appear and we save the scores in a new DataFrame. Then, we calculate the rate using the equation 5.1. In this case we consider the rate per 10000 movements and in addition, we round the results at third decimals. As a result we obtain a DataFrame containing six columns, as shown in Table 5.3.

	<i>Aircraft</i>	<i>rate Aircraft x10000mvt</i>	<i>Person</i>	<i>rate Person x10000mvt</i>	<i>Vehicle /Equipment</i>	<i>rate Vehicle /Equipment x10000mvt</i>
2015	46	0.298	3	0.002	5	0.019
2016	43	0.265	2	0.002	10	0.012
2017	75	0.454	10	0.003	28	0.060
2018	112	0.650	13	0.004	35	0.075
2019	111	0.671	4	0.004	40	0.024
2020	38	0.536	7	0.008	17	0.099

Table 5.3: Resulting table of Event types analysis on *Runways Incursions*.

5.4 Checks and validation

With the interests of verifying that the results are accurate, the implementation of a check is required. In particular, we calculate the number of rows in the Excel file, for each year, and the number of blank cells and we compared them with the the total number of occurrences extracted⁷.

The number of rows is calculated using the `len()` command, the number of blank

⁷Ref. code lines 64-65 and 121-123 in Appendix B.

cells using `isna().sum()` and the total number of occurrences are the values in the second column in Table 5.1 or Table 5.2. Whenever the numbers do not correspond the code provides a warning: `'Missing data!'`.

In the Event types analysis, implementing a check is ineffective because the analysis does not consider all the event types in the Excel file, so the validation results irrelevant.

Chapter 6

Output graphics

The graphical output of the project is twofold: the code aims, firstly, at obtaining the same results displayed in the Safety Portal and secondly, at identifying new and more appropriate graphical visualisations¹.

The first target helps to understand how using Python can be easier compared to the use of other visualisation software² and so, to target a possible change in the current scenario maintaining the current shape. The second goal tries to improve the current scenario, operating with different types of plots and new design.

6.1 Reproducing Safety Portal results

Regarding obtaining the same results of the Safety Portal, we focus on following the existing layout, therefore the chosen types of graph are pie charts, bar plots and line plots. The diagrams that we are going to replicate can be seen in section 2. The layouts are similar to the old ones but also reflect Python characteristics.

6.1.1 Dangerous Goods

Event types

Figure 6.1 represents the graphical output of Event Types analysis, following its legend in Figure 6.2. Looking at the evolution over the years, the changing in ENAC

¹Referring code lines of Event types plots are 202-369 and of Dangerous Goods types plots are 817-950 in Appendix A, whereas code lines of SPI are 123-249 and 274-433 in Appendix B.

²Generally avionic agencies make use of Microsoft Power Bi, which is an interactive data visualization software developed by Microsoft.

reporting system is evident. In 2014 and 2015 reporting was minimal as the event types reported are few; from 2016, and even more from 2018, the reporting system has become more and more efficient. An important variation is registered from 2020 when the attached files were introduced, as a matter of facts the possibility of reporting many events together has favored the notification of 'Dangerous Goods Undeclared' as a general definition of any events. This trend could lean to an inaccurate analysis, due to the fact that the input data is misrepresented.

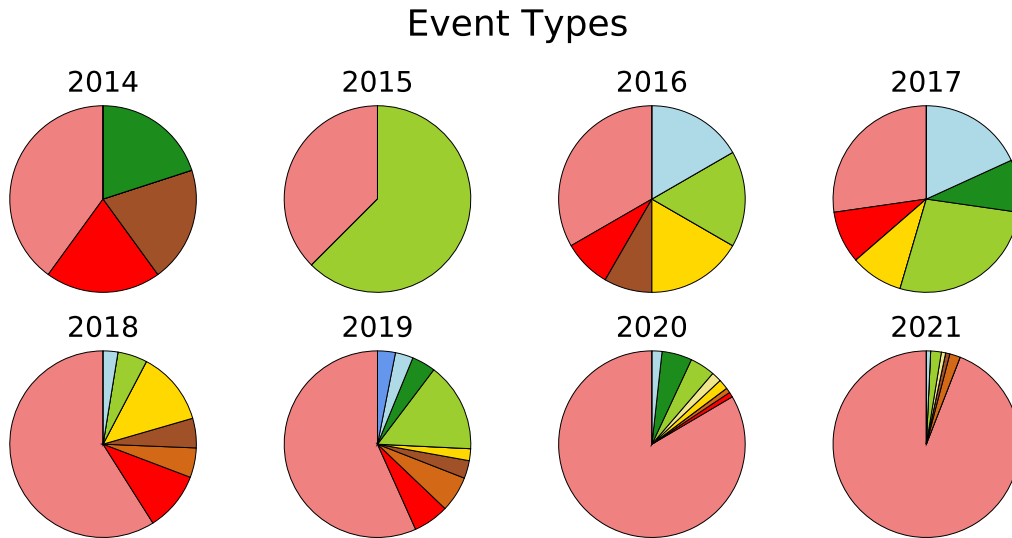


Figure 6.1: Event types output chart.

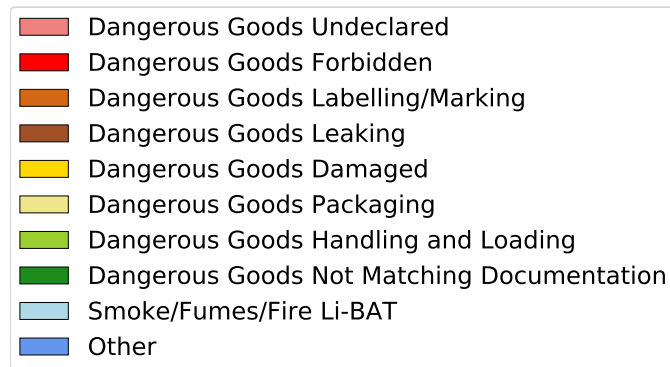


Figure 6.2: Legend of figure 6.1

Dangerous Goods types

Figure 6.3 represents the analysis output of Dangerous Goods types, followed by its legend in Figure 6.4. The registered trend over the years looks different from the one in Figure 6.1 because it is a different analysis and also it is affected differently by the changes in the reporting system. In particular, what stands out is the increase of 'Flammable Liquids' from 2018.

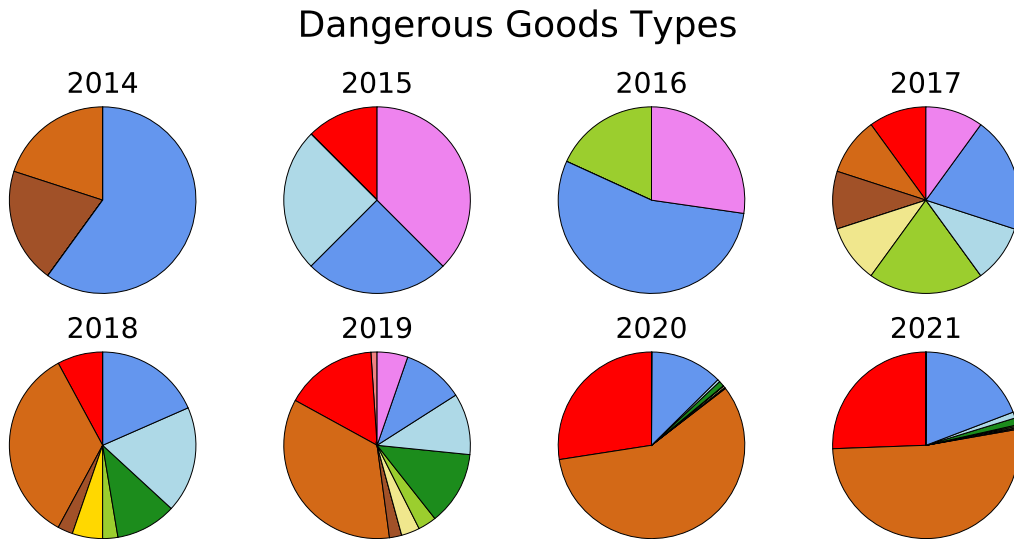


Figure 6.3: Dangerous Goods Types output chart.

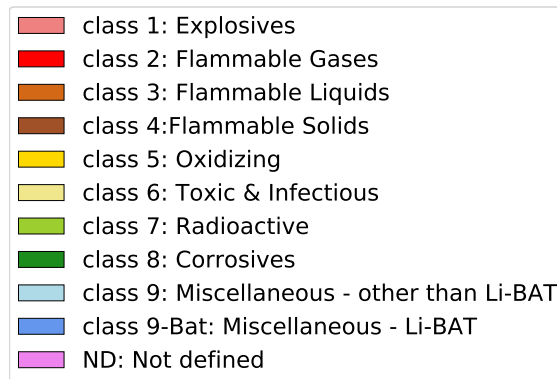


Figure 6.4: Legend of figure 6.3.

It is important to underline that we do not note the percentage values and the number of occurrences on the pie charts, because it can look disorganized. In respond to this omission, we suggest to attach a table (similar to Table 4.2), which

is easier to consult and to extract numerical and precise results.

6.1.2 Safety Performance Indicators

Regarding the general analysis, the requested plots are two bar plots and a line plot. The first two are about the number of occurrences per year and rate per year, whereas the third is about the trend about movements per year. The last graph can change according to the considered SPI; in some cases it can represent the trend of flights per year and in one other case the trend of occupancy duration in the sky.

These three graphs can be found for almost all the considered SPIs, clearly following the indicator's definition to conduct the analysis. Figure 6.5 presents the requested bar plots and Figure 6.6 the line plot. Since we examined two study cases, we present also the results for the second case in Figure 6.7. Looking at the bar plots on the number of events/occurrences, it is clear that the decrease in 2020 is a consequence of the Covid-19 pandemic, which can be also identified in the line plot in Figure 6.6.

In order to make the graphical outputs as complete as possible, we present, in figures 6.8 and 6.9, also the trends of flights and occupancy duration.

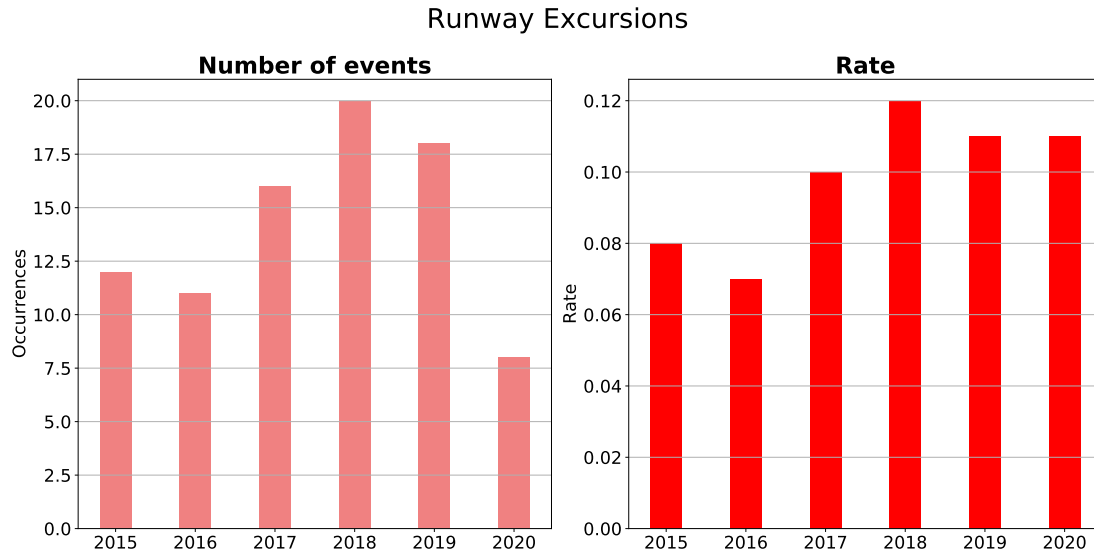


Figure 6.5: Bar plot on occurrences per year and rate per year of *Runway Excursion*, as included in ENAC safety portal.

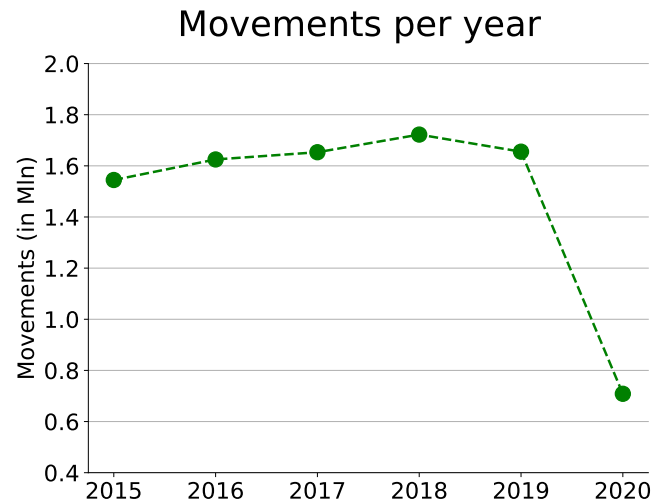


Figure 6.6: Trend of movements per year.

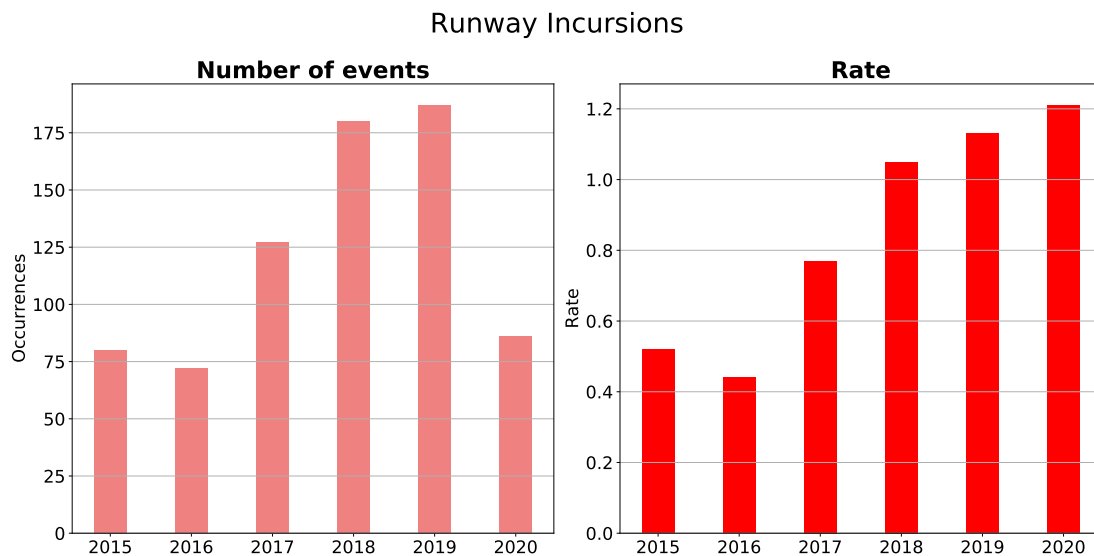


Figure 6.7: Bar plot on occurrences per year and rate per year of *Runway Incursion*, as included in ENAC safety portal.

Only three SPIs, such as BIRD, UPA, ATM failure require only the bar plot about occurrences per year. The SPIs, named RAMP and LASER, present also further geographical examinations, which are not considered in this project. In addition, we decide to not show the values at the top of the bar plots, because it would be a repetition since the vertical axis is always shown.

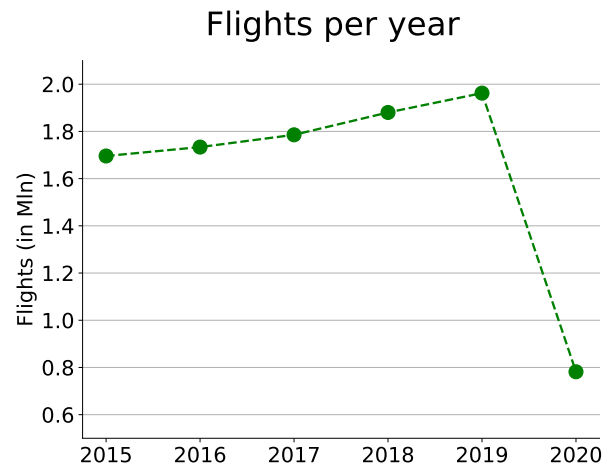


Figure 6.8: Trend of flights per year.

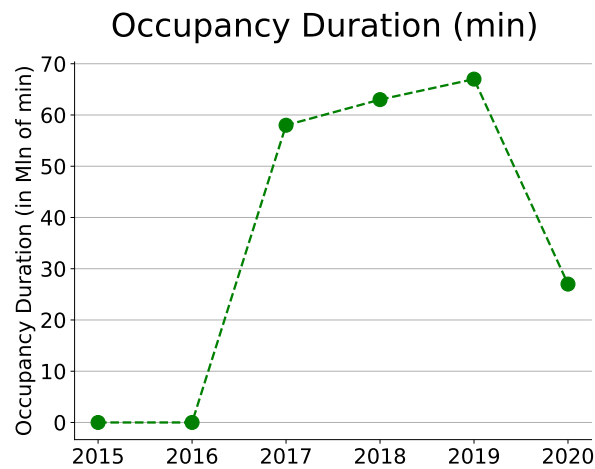


Figure 6.9: Trend of occupancy duration per year.

Regarding the Event types analysis, the requested plots are six bar plots, three on occurrences per year and three on rate per year. Figure 6.10 present the output as included in the Safety Portal. It is possible to understand the runway incursions by an aircraft are the more frequent types, followed by those by vehicle/equipment.

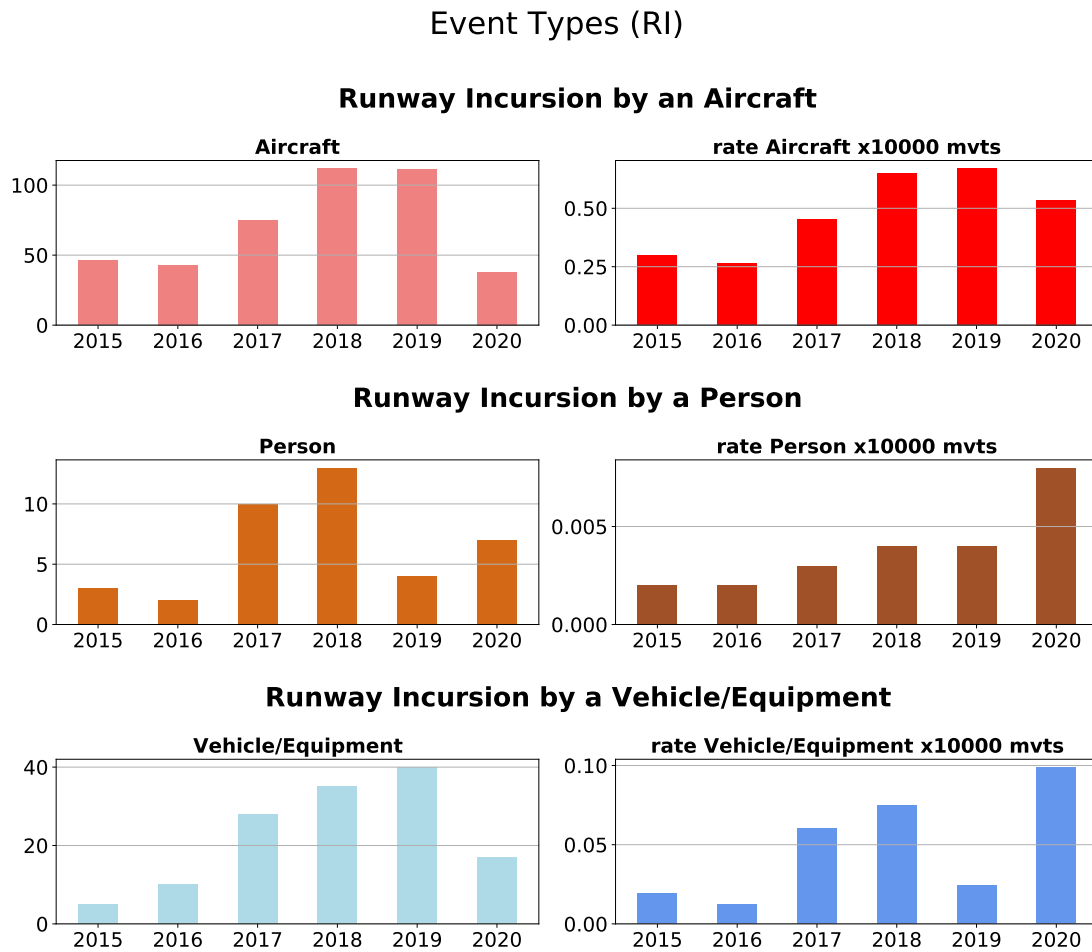


Figure 6.10: Bar plots on Event types analysis of 'Runway Incursions', as included in ENAC safety portal.

6.2 Finding new visualisation options

When presenting any data analysis to an audience³, it is fundamental to understand which information we are sharing and the best way to do so. In this regard, we focus the attention on the types of plots that can be used to visualise the data and their properties to make the representation clear and exhaustive. As mentioned before, Python offers advanced methods of visualisation. Those that have been chosen in this project will be discussed further on.

³ENAC Safety Report Portal is a public accessible site for consulting.

6.2.1 Dangerous Goods

Event types

Concerning Event types analysis, the first attempt of a different visualisation is a simple bar plot, which represents the amount of event types reported yearly. It can be useful to understand how the safety culture has spread over the years, displaying the total number of occurrences.

In this case we use `matplotlib.pyplot.bar()` command to generate the output.

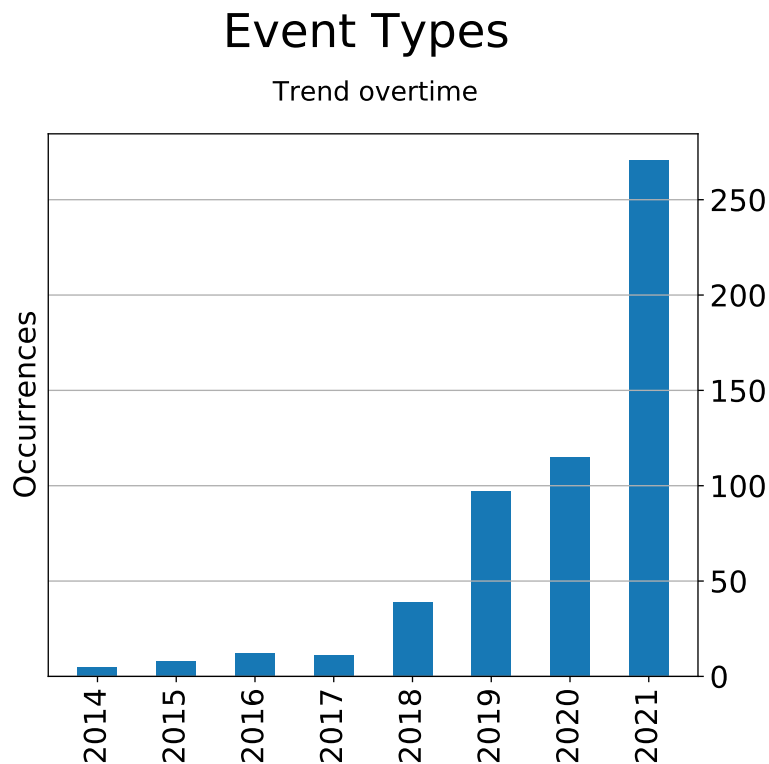


Figure 6.11: Trend of total amount of Event types reported yearly.

A second option is a horizontal stacked bar plot, with two different perspectives. The characteristic of this type of plot is that it shows the percentages in a clearer and more consistent way than a pie chart, when the amount of data is significant. It is possible to understand the mutual relation between the event types and their weights in each year, following the percentage indication at the bottom line of the graph. Figures 6.12 and 6.13 differ in term of the focus: the first one points out the years, whereas the second one the event types themselves.

We operate a *pandas* plotting command, `DataFrame.plot.barh()`, to generate both graphs.

In Figure 6.12 it is shown, as in Figure 6.1, the widely use of 'Dangerous Goods Undeclared' in reporting. The introduction of the second stacked bar plot helps to understand how the reporting of a specific type has evolved in the considered time-period. For instance, it is clear that reporting 'Dangerous Goods Undeclared' is more predominant in 2021 than in all the other years. The possible reason behind this trend is explained in section 6.1.1.

Lastly, in Figure 6.15 we present the values of every category of Event types. In this way we can have a clear idea on their order of magnitude. It is also represented the individual trend over the considered time-period. It is important to notice that the vertical axis is different for each graph.

In this last case, as well as the first bar plot, we use `matplotlib.pyplot.bar()` command.

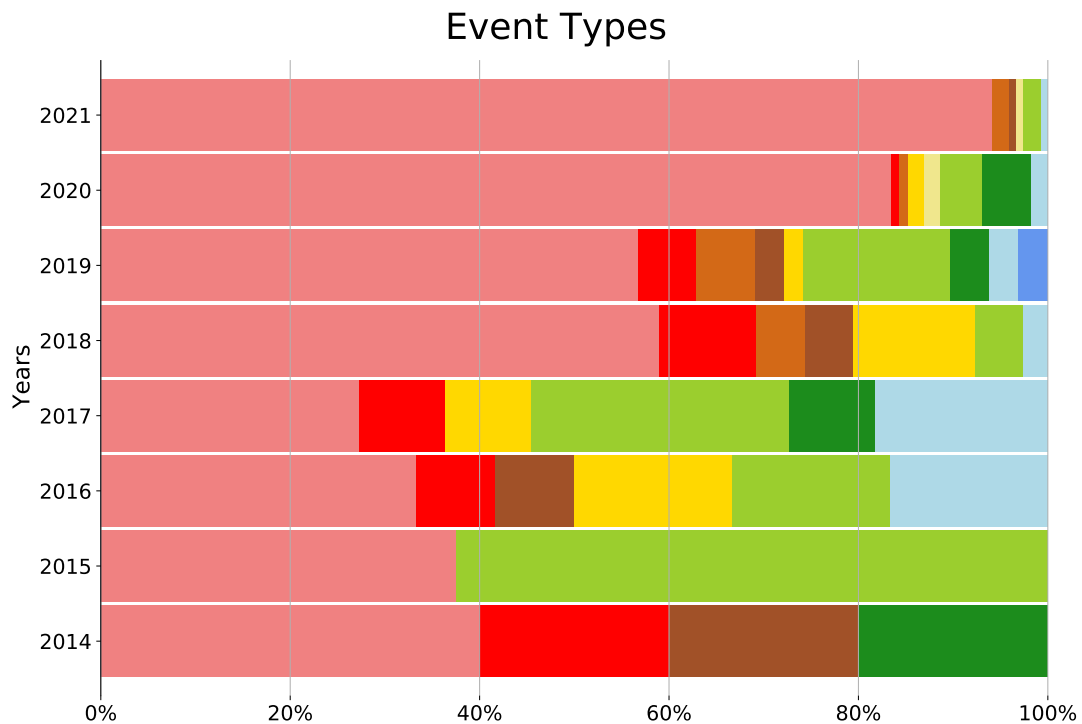


Figure 6.12: Stacked bar plot of Event types focusing on the years.

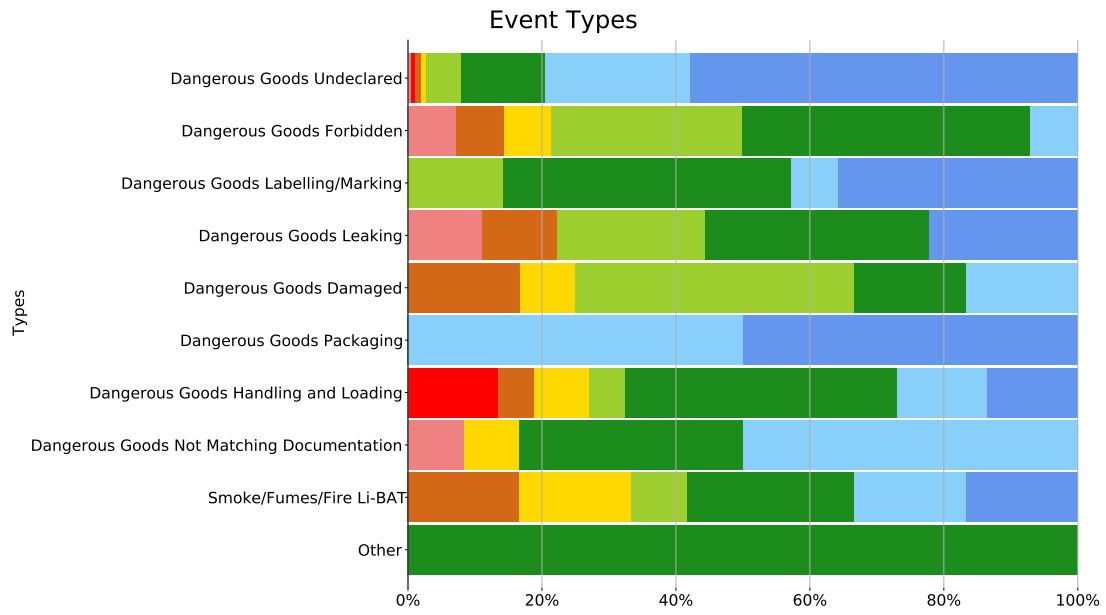
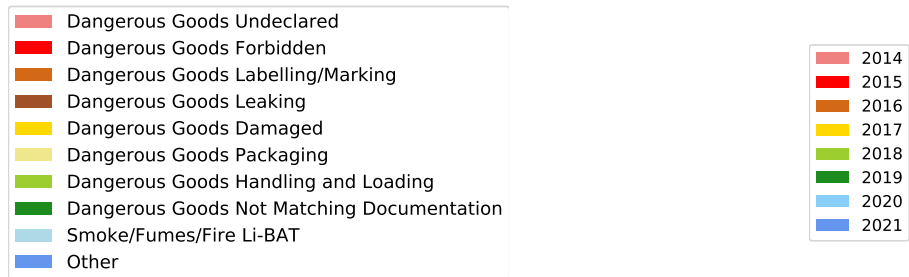


Figure 6.13: Stacked bar plot of Event types focusing on the types.



(a) Legend of figure 6.12.

(b) Legend of figure 6.13.

Figure 6.14: Legends of stacked plots of Event types analysis.

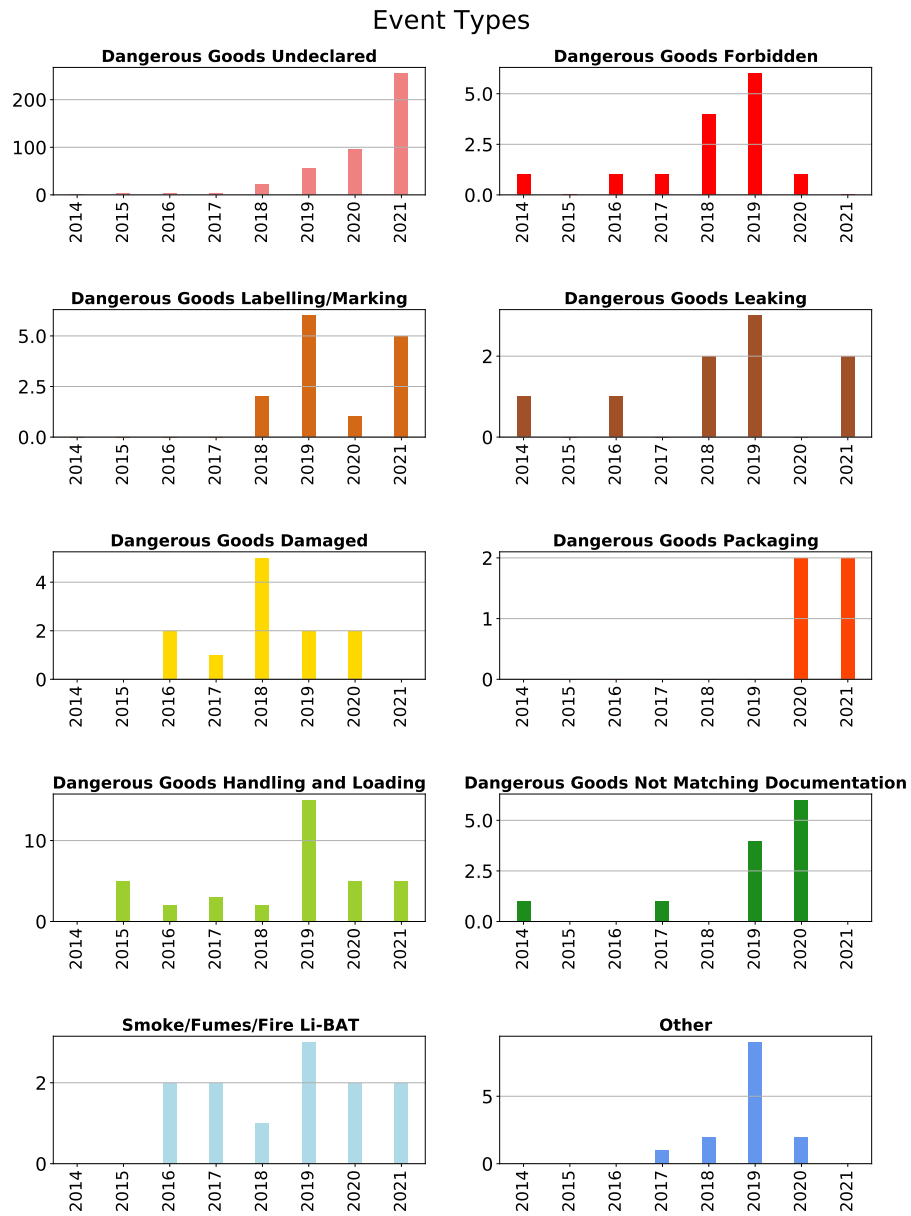


Figure 6.15: Multiple bar plots of Event types focusing on their values.

Dangerous Goods types

In this section we present the graphical outputs that suit better the results of DGs types analysis.

The first graphs are bar plots, which represent the total number of occurrences over the considered time-period. In Figure 6.16, there are two plots: the first one uses the decimal scale, whereas the second one uses the logarithmic scale. This choice can be explained looking at the left plot, where it is clear that data from 2014 to 2019 is misrepresented, so using a logarithmic scale it is better showing the orders of magnitude.

In this case we use `matplotlib.pyplot.bar()` command to generate the output.

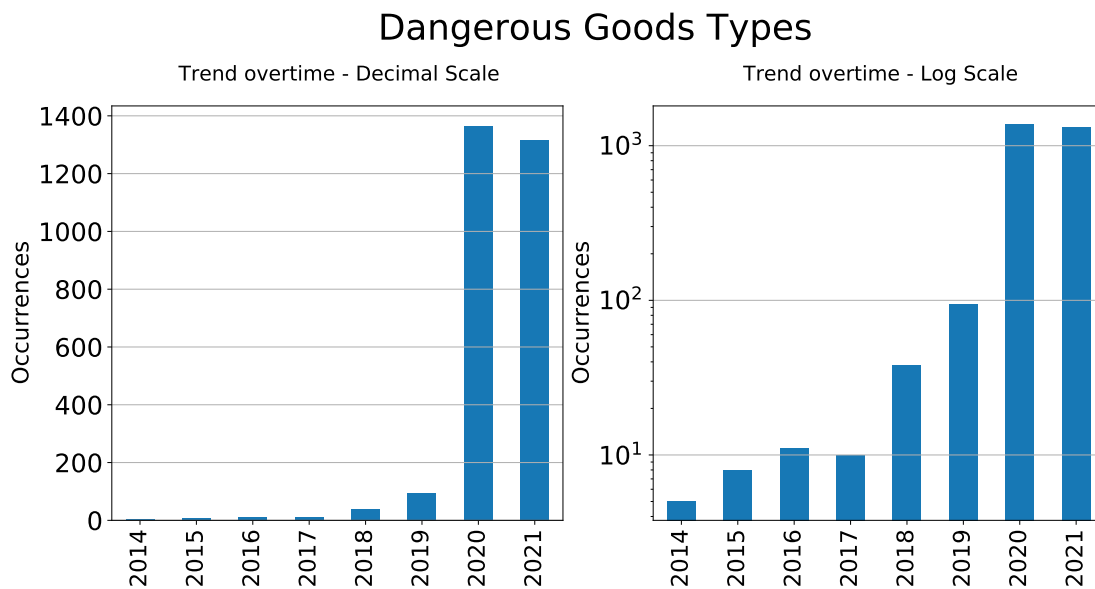


Figure 6.16: Trend of total amount of Dangerous Goods types reported yearly.

As a second possible visualisation, we present a stacked bar plot. This type of plot shows all important details in a better way than a pie chart, when the amount of data is significant. Figure 6.17 and Figure 6.18 show the relation between the total number of occurrences per class and their weights yearly, with different focal points.

We operate a *pandas* plotting command, `DataFrame.plot.barh()`, to generate both graphs.

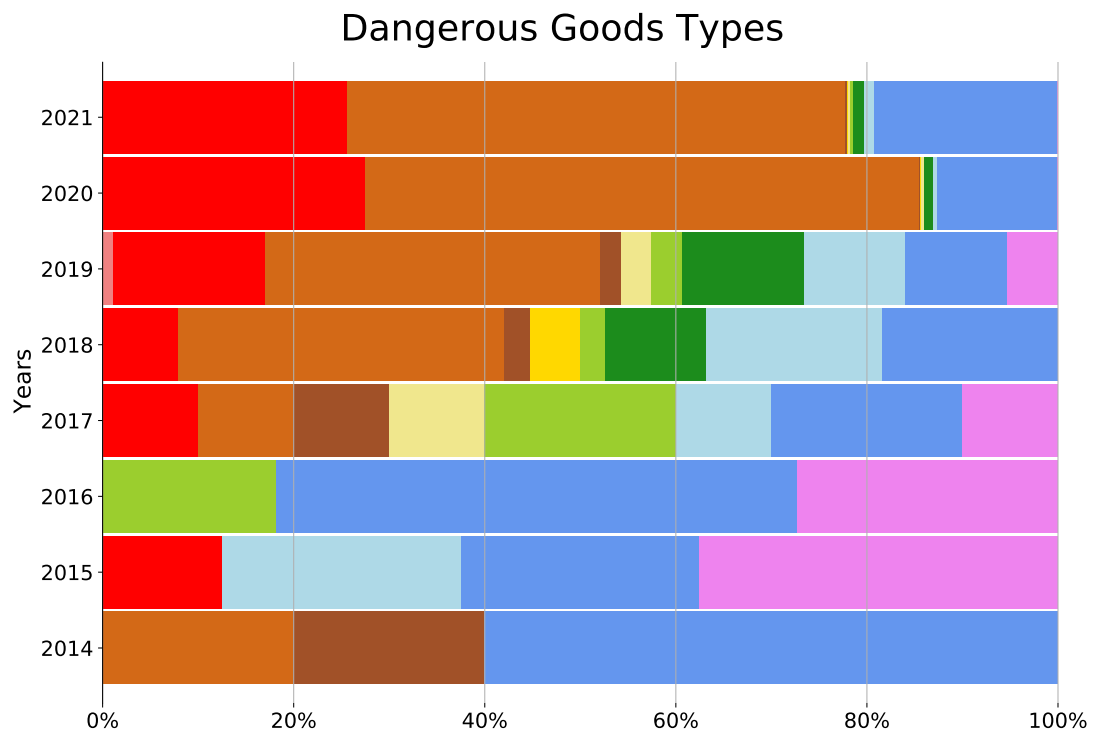


Figure 6.17: Stacked bar plot of DG types focusing on the years.

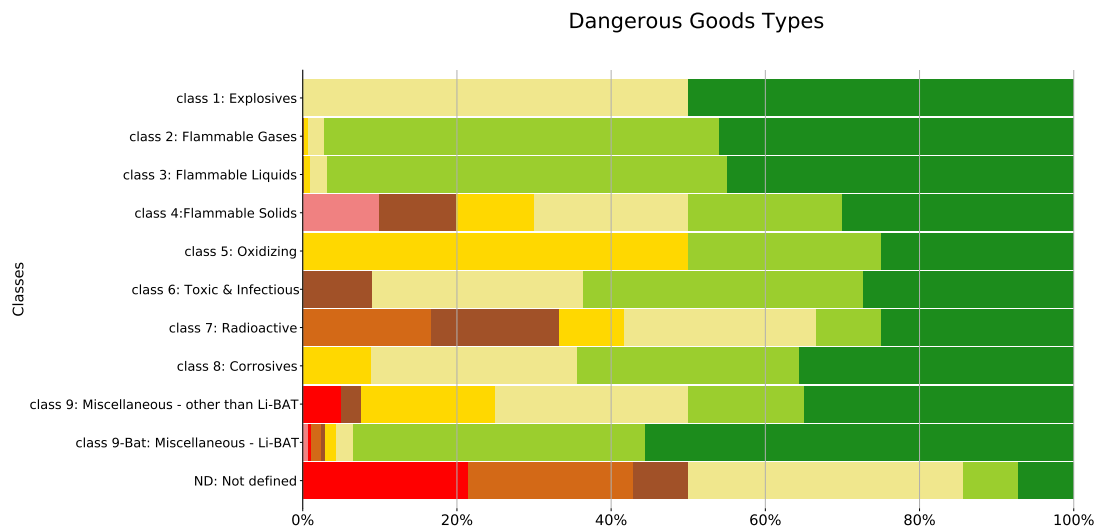
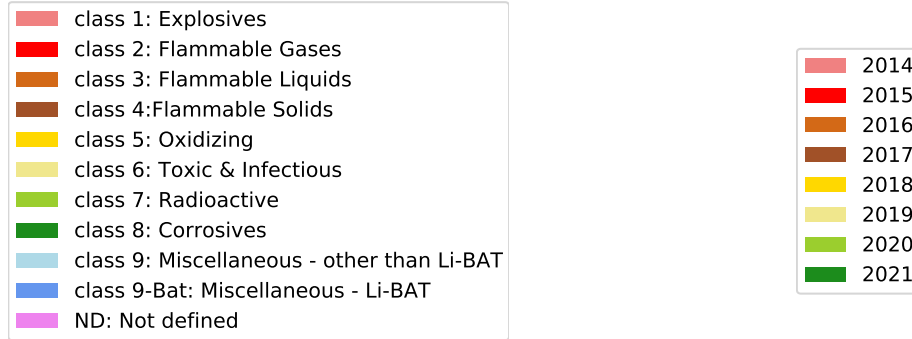


Figure 6.18: Stacked bar plot of DG types focusing on the classes.



(a) Legend of Figure 6.17.

(b) Legend of Figure 6.18.

Figure 6.19: Legends of Dangerous Goods analysis in ENAC Safety Portal.

While investigating which graphical visualisation could suit better the analysis, we conclude that with such many categories to display a tabular representation could be the best option (similar to Table 4.7).

6.2.2 Safety Performance Indicators

Considering new ways of visualization for SPI analysis, we decide to represent the rate as a line plot to highlight the trend instead of its specific values, as shown in Figure 6.20 and Figure 6.21.

Regarding the Event types analysis, Figure 6.22 present the output as a possible variation, using line plots of rate graphs instead.

In addition to Figure 6.22, we consider other two ways of visualising the Event types results. The first one studies the number of occurrences implementing a stacked bar plot, in order to show the relation of the three considered event types respectively. Figure 6.24 presents the weight of each event type relating to the others. It is important to underline that we do not use percentages because these event types are just a portion of all those reported.

The second implemented graph is a line plot that illustrates all three rate trends of the considered event types together. In this way, it is possible to compared them easily, as shown in figure 6.25.

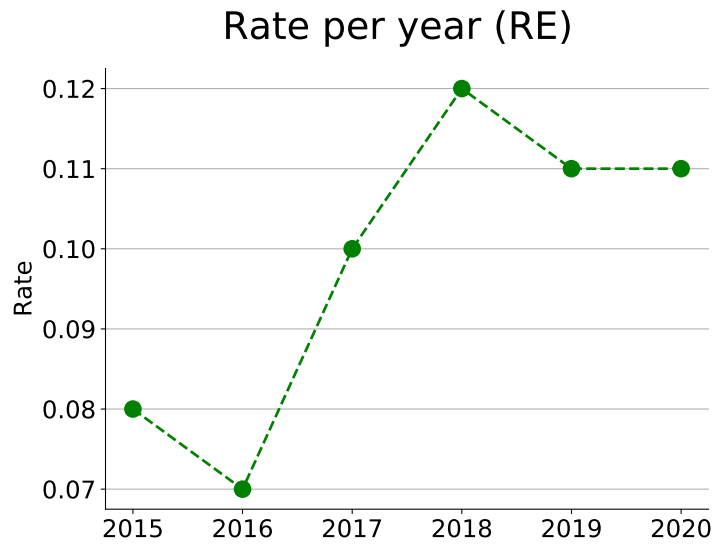


Figure 6.20: Trend of fights per year for *Runway Excursions*.

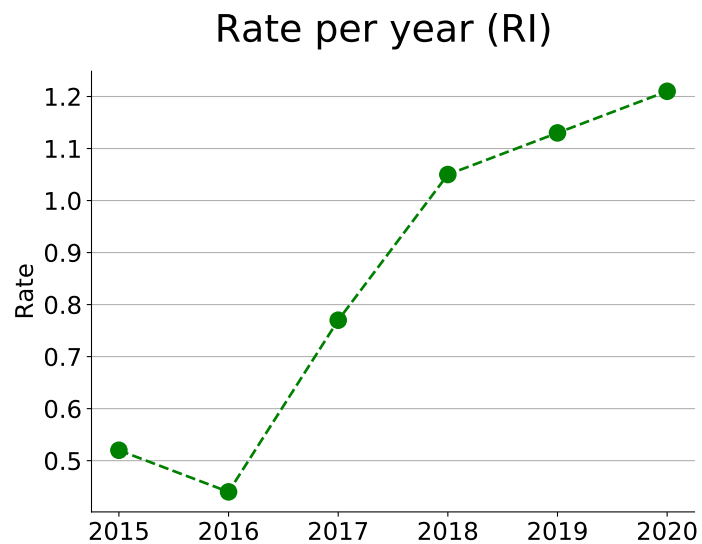


Figure 6.21: Trend of fights per year for *Runway Incursions*.

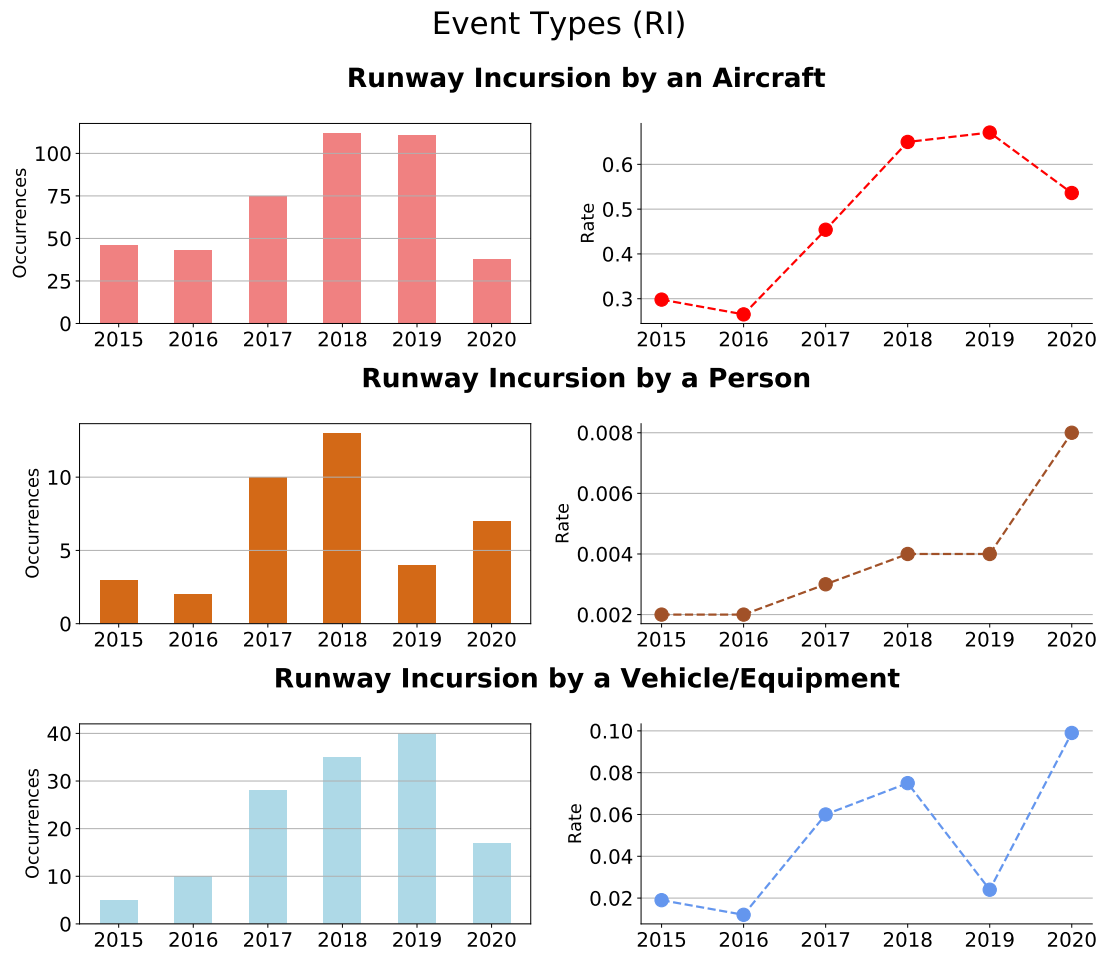


Figure 6.22: Bar plots and line plots on Event types analysis of *Runway Incursions*.

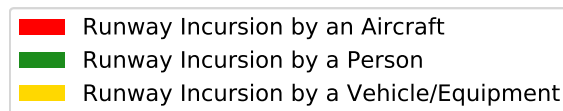


Figure 6.23: Legend of figure 6.24.

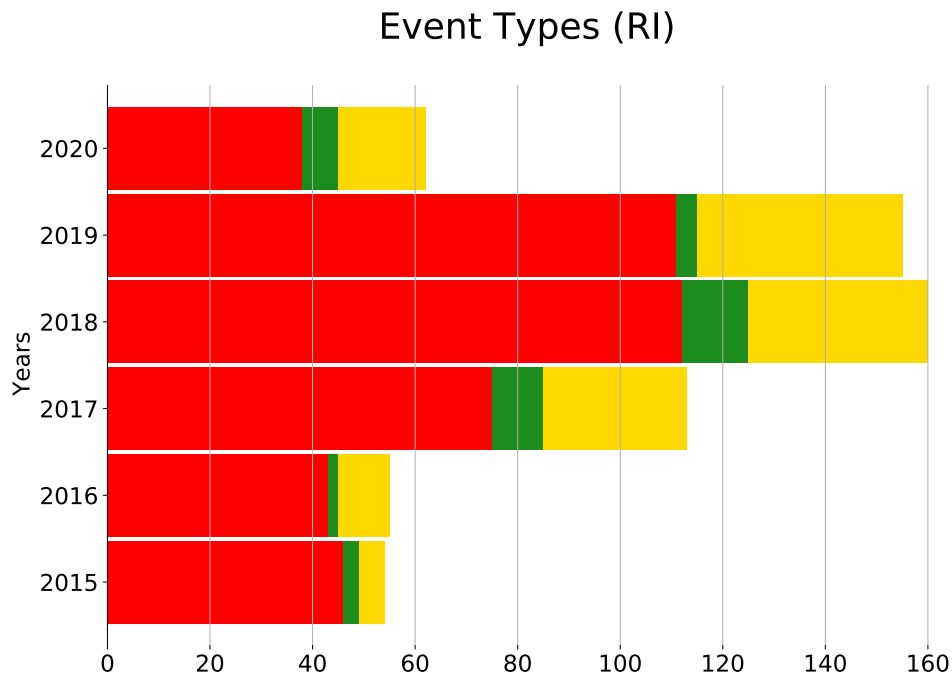


Figure 6.24: Stacked bar plot on Event types analysis of *Runway Incursions*.

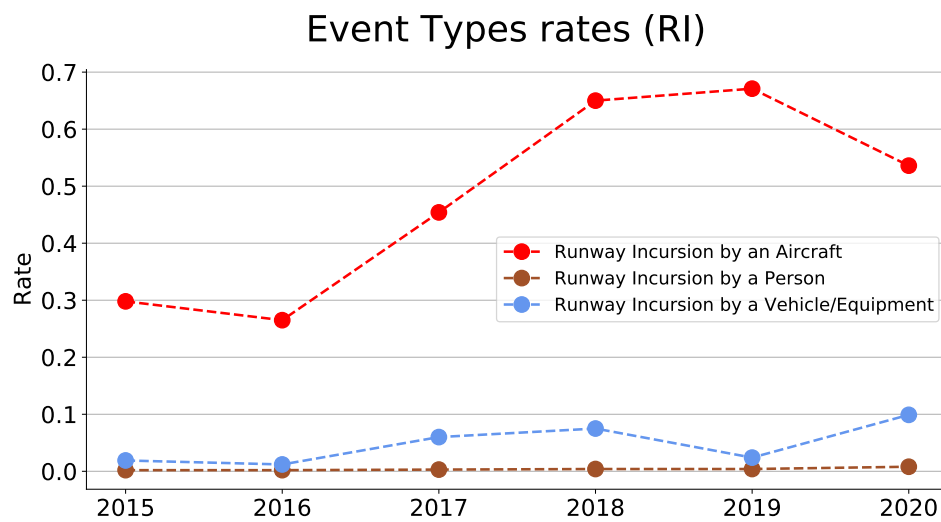


Figure 6.25: Line plot on Event types analysis of *Runway Incursions* with rate trends together.

Chapter 7

Conclusions

This thesis started from ENAC data-sets. Its output is a Python code that can analyze both Dangerous Goods data and Safety Performance Indicators data. The software pipeline managed to replicate ENAC Safety Report results¹ and to extend the analysis up to 2021 data. As a byproduct, two kinds of graphical visualisations have been presented: we reproduced a subset of already available plots and we suggested some new ones that provide a different and improved insight of the subject.

Since safety is such an essential matter in aviation, an accurate and user-friendly analysis of safety data is fundamental for a more effective prevention of potential hazards. This practice helps to deepen the knowledge on the Italian safety scenario and to understand what is important to monitor. It should be noticed that safety culture is growing, since the total number of reports has increased over the years. However a revision of the reporting system is desirable in order to make the results more accurate. Furthermore, results of 2020 and 2021 are not reliable without a discussion on the effects on aviation during the Covid-19 pandemic. The interpretation of such data is still an ongoing process, since an adequate time-frame is needed in order to gather a clear view of the scenario and the consequences on aviation.

Every national safety authority develops its unique method to process and analyze safety data, making it difficult to compare safety results and trends belonging to different states. An extended use of this Python-based pipeline could be the key to find a common ground on studying this matter. In this regard, this project aims at providing Excel format files as outputs, that could be used in already existing tools for post-processing. For instance, the use of Microsoft Power Bi to draw plots can be carried on: given the Excel files as inputs it works as usual.

¹As the publication date of this work, the ENAC Safety Report displays data up to 2020.

All presented Tables, 4.1, 4.2, 4.4, 4.5, 4.7, 5.1, 5.2 and 5.3, are saved in a folder, named 'Analysis_Excel' and allow the user to benefit from the data analysis outside the software. The connection between Python and Excel is provided by *pandas*, in particular using `DataFrame.to_excel('directory/filename.xlsx')` command.

A further improvement on the quality of data availability and presentation is a Python-based interactive dashboard. This tool is a Web interface that displays results that can be consulted by the interested parties and that are connected to the prior analysis.

This application does not required any technical knowledge to be used. The programming language used in the building process is Python and the performed analyses are based on the pipeline illustrated in chapter 4 and chapter 5. An extensive and technical explanation of the design of this dashboard can be found in Appendix C.

The programming language is the same used in the whole project, so no additional skills are required. The layout and the content can be modified based on the user's needs and changes are easy to implement. In addition, there is a connection between the input and the output, therefore any changes in the initial Excel files will affect directly the information displayed in the dashboard.

The dashboard consists of four pages and a sidebar, which allows to choose which page is visualized. The four pages are respectively: *Home* that is the opening front page, *Safety Performance Indicators*, *Dangerous Goods*, and *About* that gives some information about the project and the developers. In particular, the second and third pages collect the results given as output of the previous analyses: there is an introduction of the subject illustrating the main definitions and visualisations, both tabular and graphical, as provided in chapter 6. All the included plots are interactive, thus the interpretation of the analysis results is more easily delivered and the user is highly engaged.

This project could be the starting point for an actual shared approach on safety among EU Member states. Therefore, the optimization of the Python code, which is currently under development, is encouraged. We also underline the potential of the dashboard, which can become more refined, exhaustive and structured.

Appendix A

Python-based Pipeline: DGs template

This appendix contains the Python code of the Dangerous Goods analysis.

```
1 import pandas as pd
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import matplotlib.colors as mcol
5 import matplotlib as mpl
6 import seaborn as sns
7 from itertools import chain #to append lists of list
8 import matplotlib.ticker as mtick #to show percentage ticks in
    matplotlib graphs
9 import os.path #to verify the existence of a file in a specific
    directory
10 import tabula #to read pdf files
11 from matplotlib.gridspec import SubplotSpec #to title every row of in
    figure with subplots
12
13 #Indicating the years considered for the analysis.
14 years=['2014','2015','2016','2017','2018','2019','2020','2021']
15
16 #READING DATA
17 #Reading all data from each Excel sheet and creating a specific
    Dataframe for each year.
18 data_2014=pd.read_excel('Dangerous_Goods/DGOR_2021.xlsx',sheet_name=
    years[0])
19 data_2015=pd.read_excel('Dangerous_Goods/DGOR_2021.xlsx',sheet_name=
    years[1])
```

```

20 data_2016=pd.read_excel('Dangerous_Goods/DGOR_2021.xlsx',sheet_name=
    years[2])
21 data_2017=pd.read_excel('Dangerous_Goods/DGOR_2021.xlsx',sheet_name=
    years[3])
22 data_2018=pd.read_excel('Dangerous_Goods/DGOR_2021.xlsx',sheet_name=
    years[4])
23 data_2019=pd.read_excel('Dangerous_Goods/DGOR_2021.xlsx',sheet_name=
    years[5])
24 data_2020=pd.read_excel('Dangerous_Goods/DGOR_2021.xlsx',sheet_name=
    years[6])
25 data_2021=pd.read_excel('Dangerous_Goods/DGOR_2021.xlsx',sheet_name=
    years[7])
26
27 data_2014.iloc[2,7]='Dangerous Goods Leaking' #In 2014 the Event Type
    about Leaking is identified differently, so we changed it (It was
    'Dangerous Goods Leaking related event'.)
28
29 #Creating a dataframe to contain all data.
30 data=[data_2014, data_2015, data_2016, data_2017, data_2018,
    data_2019, data_2020, data_2021]
31 #=====
32 #=====
33 #EVENT TYPES ANALYSIS
34 #=====
35 #Extracting only event types data and defining a list of every type,
    without manually insert them.
36
37 ets_all=pd.Series(dtype=str) #new dataframe
38 for year, df in zip(years,data): #for each year dataframe,
    extracting the column of event type and appending them all
    together
39     et= pd.Series()
40     et=df.iloc[:,7] #Event Type
41     ets_all=ets_all.append(et) #putting together every event type
42
43 #from the whole list it is necessary to remove parenthesis, to drop
    duplicates and then to split the event types that are together in a
    single cell (divided by \n)
44
45 ets_all=ets_all.str.replace('(','').str.replace(')','').
    drop_duplicates(keep='first').str.split('\n').tolist()
46
47 #then we got a list of lists, so with the command chain, all the
    items are concatenated in a single list that is transformed into a
    Series, which can be manipulated deleting other duplicates,
    sorting in ascending order and deleting also blank spaces
48
49 et_all=pd.Series(list(chain(*ets_all))).str.lstrip().drop_duplicates(
    keep='first')

```

```

50 et_all=et_all.drop(et_all[et_all==''].index).sort_values(ascending=
    True)
51 #et_all is the list of 35 event types that can be used in reporting.
52
53 #COUNTING
54 eventtype_count=pd.DataFrame(index=et_all, columns=years) #new
    dataframe as columns the years and indexes all even types.
55 #for each year the count of each event type is recorded in the
    specific cell in the new dataframe
56
57 data_plus=[data_2014, data_2015, data_2016, data_2017, data_2018,
    data_2019, data_2020[data_2020['count']!=0], data_2021[data_2021['
    count']!=0]]
58 #for years 2020 and 2021 it is necessary to not consider those
    occurences with 'count'=0, so I defined a new general Dataframe(
    data_plus)
59
60 for year, df in zip(years, data_plus):
61     for dg in et_all:
62         eventtype_count.loc[dg,year]=df.iloc[:,7][df.iloc[:,7].str.
            contains(dg)].value_counts().sum()
63
64 #Dataframe with total count per year (uncomment following rows if
    necessary)
65 #total_counts_tot=pd.DataFrame(index=['Total'], columns=years)
66 #total_counts_tot[years]=eventtype_count[years].sum()
67 #eventtype_count=eventtype_count.append(total_counts_tot)
68
69 #=====
70 eventtype_count.to_excel('Analysis_Excel/eventtype_all.xlsx') #saving
71 #=====
72 eventtype_count
73
74 #GROUPING
75 #In order to make data more understandable, we define new groups
    merging some old ones together. The 10 new categories are listed
76 # in new_types, the first six are the same considered above, whereas
    the last four contain more groups.
77 new_type=['Dangerous Goods Undeclared', 'Dangerous Goods Forbidden',
    'Dangerous Goods Labelling/Marking', 'Dangerous Goods Leaking',
    'Dangerous Goods Damaged', 'Dangerous Goods Packaging', 'Dangerous
    Goods Handling and Loading', 'Dangerous Goods Not Matching
    Documentation', 'Smoke/Fumes/Fire Li-BAT', 'Other']
78
79 #At each new row of the new dataframe we insert specifically the data
    of the considered group.
80 newtype_count=pd.DataFrame(index=new_type, columns=years) #new
    DataFrame
81

```

```

82 #DG UNDECLARED
83 newtype_count.loc['Dangerous Goods Undeclared']=eventtype_count.loc['
    Dangerous Goods Undeclared']
84
85 #DG HANDLING AND LOADING
86 # DG Handling and Loading contains also: Dangerous Goods Loading/
    Unloading, Dangerous Goods Load Weighting, Dangerous Goods Exceeds
    Storage Compartment Limitations, Dangerous Goods Unsecure without
    Shift
87 load=['Dangerous Goods Handling and Loading','Dangerous Goods Loading
    /Unloading', 'Dangerous Goods Load Weighting','Dangerous Goods
    Exceeds Storage Compartment Limitations','Dangerous Goods Unsecure
    without Shift']
88 newtype_count.loc['Dangerous Goods Handling and Loading']=
    eventtype_count.loc[load].sum()
89
90 #DG FORBIDDEN
91 newtype_count.loc['Dangerous Goods Forbidden']=eventtype_count.loc['
    Dangerous Goods Forbidden']
92
93 #DG LABELLING/MARKING
94 newtype_count.loc['Dangerous Goods Labelling/Marking']=
    eventtype_count.loc['Dangerous Goods Labelling/Marking']
95
96 #DG NOT MATCHING DOCUMENTATION
97 #Dangerous Goods Not Matching Documentation contains also Not
    Recorded
98 doc=['Dangerous Goods Not Matching Documentation','Dangerous Goods
    Not Recorded']
99 newtype_count.loc['Dangerous Goods Not Matching Documentation']=
    eventtype_count.loc[doc].sum()
100
101 #DG LEAKING
102 newtype_count.loc['Dangerous Goods Leaking']=eventtype_count.loc['
    Dangerous Goods Leaking']
103
104 #DG DAMAGED
105 newtype_count.loc['Dangerous Goods Damaged']=eventtype_count.loc['
    Dangerous Goods Damaged']
106
107 #DG PACKAGING
108 newtype_count.loc['Dangerous Goods Packaging']=eventtype_count.loc['
    Dangerous Goods Packaging']
109
110 #SMOKE/FUMES/FIRE (Smoke/Fumes/Fire Li-BAT):
111 flame=pd.Series(eventtype_count.index)
112 flame=flame[flame.str.contains('|'.join(['Smoke','Battery','Fumes','
    Smell']))].tolist() #keeping only the categories that contain
    certain words.

```

```

113 newtype_count.loc['Smoke/Fumes/Fire Li-BAT']=eventtype_count.loc[
    flame].sum()
114
115 #OTHER
116 other=new_type[0:-4]+load+doc+flame
117 newtype_count.loc['Other']=eventtype_count.drop(other).sum() #
    combining all left groups, so all without those already considered
    just above
118
119 #In 'Other' category there are some event types that are not
    consistent with the analysis: from ENAC existing analysis, we can
    identify which event types had been ignored
120 et_ignored=['Action Performed Incorrectly','Baggage Non-Compliant
    Carriage of Load','Baggage Non-Compliant Carriage of Load','
    Baggage Security Check','Cargo Labelling/Marking','Crew Door Fails
    to Open/Close','External Load - Release','Helicopter RPM
    Exceedance','Lack of Communication','Passenger Carry-On Baggage','
    Qualifications','Return to Stand','Use of Emergency Equipment']
121
122 #defining a new dataframe to save this change: the first 10
    categories stay the same
123 newtype_count_wother=pd.DataFrame(index=new_type, columns=years)
124 newtype_count_wother.loc['Dangerous Goods Undeclared':'Smoke/Fumes/
    Fire Li-BAT']=newtype_count.loc['Dangerous Goods Undeclared':'
    Smoke/Fumes/Fire Li-BAT']
125 #NEW OTHER
126 other_riduced=new_type[0:-4]+load+doc+flame+et_ignored
127 newtype_count_wother.loc['Other']=eventtype_count.drop(other_riduced)
    .sum() #combining all left groups, so all without those already
    considered just above
128
129 #Dataframe with total count per year (uncomment following rows if
    necessary)
130 #total_counts_et=pd.DataFrame(index=['Total'], columns=years)
131 #total_counts_et[years]=newtype_count_wother[years].sum()
132 #newtype_count_wother=newtype_count_wother.append(total_counts_et)
133 #=====
134 newtype_count_wother.to_excel('Analysis_Excel/event_type_collapsed.
    xlsx') #saving
135 #=====
136 newtype_count_wother #df with modified OTHER category
137
138 #CHECKS AND VALIDATION
139 #As a check, we count the total number of events in each year(14-19
    and 20-21 are separated at the beginning with if conditions) this
    check is done considering OTHER category complete (no event types
    have been removed as ENAC intervention), because it allows us to
    proper check the accuracy of the analysis.
140

```

```

141 for year, df in zip(years, data):
142
143     total_event=newtype_count[year].sum() #total number of event type
        after manipulation
144     print(f'Total number of events in {year}: {total_event}')
145
146     if year=='2014'or year=='2015'or year=='2016'or year=='2017'or
        year=='2018'or year=='2019':
147
148         total_reports=len(df.iloc[:,7]) #total number of reports (
            number of excel file rows)
149         print(f'Total number of reports in {year}: {total_reports}')
150
151         multiple=list(df[df.iloc[:,7].fillna(' ').str.contains('\n')
            ].index) #delete blank cells and list which cells as \n
152
153         if year=='2020'or year=='2021': #for 2020 and 2021 just 'count'=1
            is evaluted (attached files souldn't have this necessity)
154
155             total_reports=len(df[df['count']!=0].iloc[:,7]) #total number
                of reports (number of excel file rows)
156             print(f'Total number of reports in {year}: {total_reports}')
157
158             multiple1=df[df['count']==1].reset_index(drop=True) #
                considering only rows with 'count'==1 because those could have
                multiple types in a single cell
159             multiple=list(multiple1[multiple1.iloc[:,7].fillna(' ').str.
                contains('\n')].index)
160             df=multiple1
161
162             num_dg_2=0 #initiating variables: _2: two event types, _3: three
                event types ...
163             num_dg_3=0
164             num_dg_4=0
165             num_dg_5=0
166             num_dg_6=0
167             for i in multiple:
168                 ex=df.iloc[i,7].count('\n') #counting how many \n
169                 if ex==1: #1 addition event type
170                     num_dg_2=num_dg_2+1
171                 if ex==2: #2 additional event type
172                     num_dg_3=num_dg_3+1
173                 if ex==3: #...
174                     num_dg_4=num_dg_4+1
175                 if ex==4:
176                     num_dg_5=num_dg_5+1
177                 exx=df.iloc[i,7].count('\n\n') #counting how many \n\n
                    consecutively
178                 if exx==1:#one addition event types

```



```

179         num_dg_3=num_dg_3-1 #deleting that '3 event types'
counted before
180         num_dg_2=num_dg_2+1 #considering 2 event types, so 1
additional
181         if exx==2:
182             num_dg_5=num_dg_5-1
183             num_dg_3=num_dg_3+1
184         if exx==5:
185             num_dg_6=num_dg_6+1
186         if exx!=1 and exx!=2 and exx!=5 and exx!=0 and ex>4: #
checking other options
187             print('Check the count!')
188         print(f'Total additional event types: {num_dg_2 +2*(num_dg_3)+
3*(num_dg_4)+4*(num_dg_5)+5*(num_dg_6)}')
189         print(f'2 event type: {num_dg_2}')
190         print(f'3 event type: {num_dg_3}')
191         print(f'4 event type: {num_dg_4}')
192         print(f'5 event type: {num_dg_5}')
193         print(f'6 event type: {num_dg_6}')
194         print(' ')
195         if total_event!=total_reports + num_dg_2 +2*(num_dg_3)+ 3*(
num_dg_4)+4*(num_dg_5)+5*(num_dg_6):
196             print('Missing data!')
197
198 #GRAPHICS
199 #
200 # Pie Charts about Event Types
201 #
202 fig , ax = plt.subplots(2, 4,figsize=(12, 6)) #creating figure box
203
204 fig.suptitle('Event Types', y=1,fontsize=28) #title
205
206 color=['lightcoral','red','chocolate','sienna','gold','khaki','
yellowgreen','forestgreen','lightblue','cornflowerblue'] #colors
207 #
208 ax = ax.ravel()
209
210 for i in range(len(years)): #using a loop to draw every pie chart
211
212     axes = ax[i]
213
214     axes.pie(newtype_count_wother[years[i]], colors=color, shadow=
False, labeldistance=1.1, startangle=90, radius=1.1, wedgeprops =
{"edgecolor" : "black", 'linewidth': 0.3, 'antialiased': True})
215
216     axes.set_title(years[i], fontsize=22, y=0.97)
217
218 legend=fig.legend(new_type, loc='center left', bbox_to_anchor=(1,
0.5), prop={'size': 14}) #legend

```

```

219 #plt.rc('legend', fontsize=22) # legend fontsize
220 #
221 fig.tight_layout()
222 fig.savefig('Graphics/event_types/event_type_general.pdf',format = '
    pdf', dpi = 300) #saving
223
224 fig.tight_layout()
225 fig.savefig('Graphics/event_types/event_type_general_withlgd.pdf',
    bbox_inches='tight', format = 'pdf', dpi = 300)#saving chart and
    legend
226
227 def export_legend(legend, filename='Graphics/event_types/
    event_type_ldg_pie.pdf'):
228     fig=legend.figure
229     fig.canvas.draw()
230     bbox=legend.get_window_extent().transformed(fig.dpi_scale_trans.
    inverted())
231     fig.savefig(filename, dpi=300, bbox_inches=bbox)
232
233 export_legend(legend) #saving only the legend
234
235 #
236 # Bar Chart about Event Types General
237 #
238 fig, ax = plt.subplots(1,1, sharex = False, sharey=False, figsize
    =(7, 6)) #creating figure box
239
240 fig.suptitle('Event Types',y=1.05,fontsize=28) #title
241
242 axisx=np.arange(len(years))
243 #
244 axes = ax
245
246 axes.bar(x=axisx, height=newtype_count_wother.sum(), width=0.5,
    tick_label=years, color='tab:blue')
247
248 axes.set_title('Trend overtime', fontsize=16, pad=20)
249
250 axes.yaxis.tick_right() #setting ticks on the right axis
251 axes.grid(axis='y') #showing only y grid
252 plt.xticks(rotation='vertical')
253 plt.rc('xtick', labels=16)
254 plt.rc('ytick', labels=16)
255 ax.set_ylabel('Occurrences', fontsize=18) #axis title
256 #
257 fig.savefig('Graphics/event_types/event_type_overtime.pdf',
    bbox_inches='tight', format = 'pdf', dpi = 300)#saving
258
259 #

```

```

260 #Stacked Bar chart with percentage for years
261 #
262 fig, ax = plt.subplots(1, 1, figsize=(12, 8)) #creatng figure box
263 fig.suptitle('Event Types', fontsize=28) #title
264
265 color=['lightcoral','red','chocolate','sienna','gold','khaki','
        yellowgreen','forestgreen','lightblue','cornflowerblue'] #colors
266
267 #
268 (newtype_count_wother/newtype_count_wother.sum()*100).T.plot.barh(
        stacked=True,color=color, legend=True, lw=0, width = 0.95, ax=ax)
269
270 legend=ax.legend(new_type, loc='center left', bbox_to_anchor=(1, 0.5)
        ) #legend
271 ax.set_ylabel('Years',fontsize=18) #axis title
272 ax.spines['top'].set_visible(False) #deleting top box line
273 ax.spines['right'].set_visible(False) #deleting right box line
274 ax.spines['bottom'].set_visible(False) #deleting bottom box line
275 ax.grid(axis='x') #shwoing only x grid
276 plt.rc('xtick', labels=18)
277 plt.rc('ytick', labels=18)
278
279 fmt = '%.0f%%' #setting percentages on x axis
280 xticks = mtick.FormatStrFormatter(fmt)
281 ax.xaxis.set_major_formatter(xticks)
282 #
283 ax.legend().set_visible(False)
284 fig.tight_layout()
285 fig.savefig('Graphics/event_types/
        event_type_stackedbar_normalized_nolgd.pdf', format = 'pdf', dpi =
        300)#saving chart without legend
286
287 ax.legend(new_type, loc='center left', bbox_to_anchor=(1, 0.5)).
        set_visible(True)
288 fig.savefig('Graphics/event_types/
        event_type_stackedbar_normalized_withlgd.pdf', bbox_inches='tight',
        , format = 'pdf', dpi = 300)#saving chart plus legend
289
290 def export_legend(legend, filename='Graphics/event_types/
        event_type_stackedbar_normalized_lgd.pdf'):
291     fig=legend.figure
292     fig.canvas.draw()
293     bbox=legend.get_window_extent().transformed(fig.dpi_scale_trans.
        inverted())
294     fig.savefig(filename, dpi=300, bbox_inches=bbox)
295
296 export_legend(legend) #saving only the legend
297
298 #

```

```

299 #Stacked Bar chart with percentage for event type
300 #=====
301
302 fig , ax = plt.subplots(1, 1, figsize=(18, 10))
303 fig.suptitle('Event Types', fontsize=28)
304
305 color=['lightcoral','red','chocolate','gold','yellowgreen','
        forestgreen','lightskyblue','cornflowerblue']
306
307 #=====
308
309 (newtype_count_wother.T/newtype_count_wother.T.sum()*100).T.plot.barh
    (stacked=True,color=color,legend=True, lw=0, width = 0.95, ax=ax)
310
311 legend=ax.legend(years, loc='center left', bbox_to_anchor=(1, 0.5))#
    legend
312 ax.set_ylabel('Types', fontsize=18) #axis title
313 ax.spines['top'].set_visible(False) #deleting top box line
314 ax.spines['right'].set_visible(False) #deleting right box line
315 ax.spines['bottom'].set_visible(False) #deleting bottom box line
316 ax.invert_yaxis()
317 ax.grid(axis='x') #shwoing only x grid
318 plt.rc('xtick', labels=18)
319 plt.rc('ytick', labels=18)
320
321 fmt = '%.0f%%' #setting percentages on x axis
322 xticks = mtk.FormatStrFormatter(fmt)
323 ax.xaxis.set_major_formatter(xticks)
324 #=====
325
326 ax.legend().set_visible(False)
327 fig.tight_layout()
328 fig.savefig('Graphics/event_types/
    event_type_stackedbar_normalized_nolgd2.pdf',bbox_inches='tight',
    format = 'pdf', dpi = 300)#saving chart without legend
329
330 ax.legend(years, loc='center left', bbox_to_anchor=(1, 0.5)).
    set_visible(True)
331 fig.savefig('Graphics/event_types/
    event_type_stackedbar_normalized_withlgd2.pdf', bbox_inches='tight
    ', format = 'pdf', dpi = 300) #saving
332
333 def export_legend(legend, filename='Graphics/event_types/
    event_type_stackedbar_normalized_lgd2.pdf'):
334     fig=legend.figure
335     fig.canvas.draw()
336     bbox=legend.get_window_extent().transformed(fig.dpi_scale_trans.
    inverted())
337     fig.savefig(filename, dpi=300, bbox_inches=bbox)

```

```

338
339 export_legend(legend) #savinh just the legend
340
341 #=====
342 # Bar Chart about Event Types
343 #=====
344 fig, ax = plt.subplots(5,2, sharex = False, sharey=False, figsize
    =(16, 22)) #creating figure box
345
346 fig.suptitle('Event Types', fontsize=28,y=0.92) #title
347 plt.subplots_adjust( hspace=0.9)
348
349 color=['lightcoral','red','chocolate','sienna','gold','orangered','
    yellowgreen','forestgreen','lightblue','cornflowerblue'] #colors
350
351 axisx=np.arange(len(years)) #defining x positions
352
353 #=====
354 ax = ax.ravel()
355
356 for i in range(len(new_type)): #using a loop to draw every bar chart
357
358     axes = ax[i]
359
360     axes.bar(x=axisx, height=newtype_count.iloc[i,:], width=0.3,
    tick_label=years, color=color[i])
361
362     axes.set_title(new_type[i], fontweight='bold', fontsize=18) #
    title
363     axes.grid(axis='y') #showing only y grid
364     axes.set_xticklabels(years,rotation = 90)
365
366 plt.rc('xtick', labelsiz=18)
367 plt.rc('ytick', labelsiz=20)
368 #=====
369 fig.savefig('Graphics/event_types/event_type_barchat.pdf', format = '
    pdf', dpi = 300)#saving
370
371 #=====
372 #=====
373 #DANGEROUS GOODS TYPES ANALYSIS
374 #=====
375 #Extracting Data from Excel files about DG Types from 2014 to 2021,
    creating a new dataframe for each year
376
377 dgs_2014 = pd.DataFrame()
378 dgs_2014['source']=data_2014.iloc[:, 0] #source
379 dgs_2014['goods type']=data_2014.iloc[:,6] #type
380 dgs_2014['counts']=1 #Number of goods set to 1

```

```

381 #2015
382 dgs_2015 = pd.DataFrame()
383 dgs_2015['source']=data_2015.iloc[:, 0]
384 dgs_2015['goods type']=data_2015.iloc[:,6]
385 dgs_2015['counts']=1 #Number of goods set to 1
386 #2016
387 dgs_2016 = pd.DataFrame()
388 dgs_2016['source']=data_2016.iloc[:, 0]
389 dgs_2016['goods type']=data_2016.iloc[:,6]
390 dgs_2016['counts']=1 #Number of goods set to 1
391 #2017
392 dgs_2017 = pd.DataFrame()
393 dgs_2017['source']=data_2017.iloc[:, 0]
394 dgs_2017['goods type']=data_2017.iloc[:,6]
395 dgs_2017['counts']=1 #Number of goods set to 1
396 #2018
397 dgs_2018 = pd.DataFrame()
398 dgs_2018['source']=data_2018.iloc[:, 0]
399 dgs_2018['goods type']=data_2018.iloc[:,6]
400 dgs_2018['counts']=1 #Number of goods set to 1
401 #2019
402 dgs_2019 = pd.DataFrame()
403 dgs_2019['source']=data_2019.iloc[:, 0]
404 dgs_2019['goods type']=data_2019.iloc[:,6]
405 dgs_2019['counts']=1 #Number of goods set to 1
406 #2020
407 dgs_2020 = pd.DataFrame()
408 dgs_2020['source']=data_2020.iloc[:, 0]
409 dgs_2020['goods type']=data_2020.iloc[:,6]
410 dgs_2020['counts']=data_2020.iloc[:,8] #Number of goods
411 #2021
412 dgs_2021 = pd.DataFrame()
413 dgs_2021['source']=data_2021.iloc[:, 0]
414 dgs_2021['goods type']=data_2021.iloc[:,6]
415 dgs_2021['counts']=data_2021.iloc[:,8] #Number of goods
416
417
418 #2014–2019
419 #Process for each year:
420 #Read data from the specific column, deleting blank rows and
    resetting the index
421 #Listing the dg types dividing those that are in a single cell(
    divided by \n), then removing special characters and sorting them
422 #Creating a new dataframe listing all the codes, taking only the
    string before the first space (usually xxxx)
423 #Counting how many time a code reoccurs.
424
425 #2014
426 goods_14=dgs_2014['goods type'].dropna().reset_index(drop=True)

```

```

427 goods_14=pd.Series(list(chain(*goods_14.str.split('\n').tolist()))).
    str.replace(' ','').str.replace(' ','').sort_values(ascending=True)
    ).reset_index(drop=True)
428 dgs_code14=pd.Series(dtype=str)
429 for i in range(len(goods_14)):
430     dgs_code14=dgs_code14.append(pd.Series(goods_14[i].split()[0]),
        ignore_index=True)
431 counts_14=pd.DataFrame(dgs_code14.value_counts(),columns=['2014']).
    astype(str) #counting
432
433 #2015
434 goods_15=dgs_2015['goods type'].dropna().reset_index(drop=True)
435 goods_15=pd.Series(list(chain(*goods_15.str.split('\n').tolist()))).
    str.replace(' ','').str.replace(' ','').sort_values(ascending=True)
    ).reset_index(drop=True)
436 dgs_code15=pd.Series(dtype=str)
437 for i in range(len(goods_15)):
438     dgs_code15=dgs_code15.append(pd.Series(goods_15[i].split()[0]),
        ignore_index=True)
439 counts_15=pd.DataFrame(dgs_code15.value_counts(), columns=['2015']).
    astype(str)
440
441 #2016
442 goods_16=dgs_2016['goods type'].dropna().reset_index(drop=True)
443 goods_16=pd.Series(list(chain(*goods_16.str.split('\n').tolist()))).
    str.replace(' ','').str.replace(' ','').sort_values(ascending=True)
    ).reset_index(drop=True)
444 dgs_code16=pd.Series(dtype=str)
445 for i in range(len(goods_16)):
446     dgs_code16=dgs_code16.append(pd.Series(goods_16[i].split()[0]),
        ignore_index=True)
447 counts_16=pd.DataFrame(dgs_code16.value_counts(), columns=['2016']).
    astype(str)
448
449 #2017
450 goods_17=dgs_2017['goods type'].dropna().reset_index(drop=True)
451 goods_17=pd.Series(list(chain(*goods_17.str.split('\n').tolist()))).
    str.replace(' ','').str.replace(' ','').sort_values(ascending=True)
    ).reset_index(drop=True)
452 dgs_code17=pd.Series(dtype=str)
453 for i in range(len(goods_17)):
454     dgs_code17=dgs_code17.append(pd.Series(goods_17[i].split()[0]),
        ignore_index=True)
455 counts_17=pd.DataFrame(dgs_code17.value_counts(),columns=['2017']).
    astype(str)
456
457 #2018
458 goods_18=dgs_2018['goods type'].dropna().reset_index(drop=True)

```

```

459 goods_18=pd.Series(list(chain(*goods_18.str.split('\n').tolist()))).
    str.replace(' ','').str.replace(' ','').sort_values(ascending=True)
    ).reset_index(drop=True)
460 dgs_code18=pd.Series(dtype=str)
461 for i in range(len(goods_18)):
462     dgs_code18=dgs_code18.append(pd.Series(goods_18[i].split()[0]),
    ignore_index=True)
463 counts_18=pd.DataFrame(dgs_code18.value_counts(), columns=['2018']).
    astype(str)
464
465 #2019
466 goods_19=dgs_2019['goods type'].dropna().reset_index(drop=True)
467 goods_19=pd.Series(list(chain(*goods_19.str.split('\n').tolist()))).
    str.replace(' ','').str.replace(' ','').sort_values(ascending=True)
    ).reset_index(drop=True)
468 dgs_code19=pd.Series(dtype=str)
469 for i in range(len(goods_19)):
470     dgs_code19=dgs_code19.append(pd.Series(goods_19[i].split()[0]),
    ignore_index=True)
471 counts_19=pd.DataFrame(dgs_code19.value_counts(), columns=['2019']).
    astype(str)
472
473 #The focus is on the occurrences with 'count' equal to 1, namely those
    without attached files. It is the same process that 2014–2019
474
475 #2020
476 goods_20=pd.Series(dtype=str)
477 goods_20=dgs_2020[dgs_2020['counts']==1]['goods type'].dropna().
    reset_index(drop=True)
478 goods_20=pd.Series(list(chain(*goods_20.str.split('\n').tolist()))).
    str.replace(' ','').str.replace(' ','').sort_values(ascending=True)
    ).reset_index(drop=True)
479 dgs_code20=pd.Series(dtype=str)
480 for i in range(len(goods_20)):
481     dgs_code20=dgs_code20.append(pd.Series(str(goods_20[i]).split()
    [0]), ignore_index=True)
482
483 #In this case the focus is on the occurrences with multiple 'count'
    reported, so it is necessary to read attached files correlated by
    the 'source'
484
485 #2020plus
486 #listing all the 'source' with 'number of goods' higher than 1 and
    than defining the strings to read corresponding attached files
487 goods_pos=dgs_2020[(dgs_2020['counts']!=1) & (dgs_2020['counts']!=0)
    ]['source'].str.lstrip()
488
489 good=pd.Series(dtype=str)
490 goods_20plus=pd.Series(dtype=str)

```



```

491 error_count20=0 #difference error to be considered in the check of
    total DGs extracted compared to the sum of 'count'
492
493 for it in goods_pos:
494     goods_source='Dangerous_Goods/2020/'+it+'.xlsx' #sttached file
    name
495     good=pd.read_excel(goods_source)['Unnamed: 3'].dropna().
    reset_index(drop=True).drop(index=0) #reading the columns in the
    attached files, removing blank cells as the title(index=0), adding
    a check about the length of the attached files and the number
    reported in the 'count' columns
496     num_count=dgs_2020[dgs_2020['source'].str.contains(it)]['counts'
    ].values
497     if (len(good)!=num_count).all(): #possible error in 'count'
    column
498         print(f'The attached file {it} contains {len(good)} instead
    of {num_count} reported')
499         error_count20=error_count20+(num_count-len(good))
500
501     goods_20plus=goods_20plus.append(good) #listing together all DG
    types from all the files
502
503 goods_20plus=pd.Series(list(chain(*goods_20plus.str.split('\n').
    tolist()))).str.replace(' ','').str.replace(' ','').sort_values(
    ascending=True).reset_index(drop=True)
504 dgs_code20plus=pd.Series(dtype=str)
505 for i in range(len(goods_20plus)):
506     dgs_code20plus=dgs_code20plus.append(pd.Series(str(goods_20plus[i
    ]).split()[0]), ignore_index=True) #taking only the first string(
    UN code)
507
508 #2020 TOTAL : adding together 2020 and 2020plus codes list and
    countig them.
509 counts_20tot=pd.DataFrame(dgs_code20.append(dgs_code20plus).
    value_counts(), columns=['2020']).astype(str)
510
511 #2021 occurrences with 'counts' equal to 1 need the same process of
    2020.
512 #2021
513 goods_21=pd.Series(dtype=str)
514 goods_21=dgs_2021[dgs_2021['counts']==1]['goods type'].dropna().
    reset_index(drop=True)
515 goods_21=pd.Series(list(chain(*goods_21.str.split('\n').tolist()))).
    str.replace(' ','').str.replace(' ','').sort_values(ascending=True
    ).reset_index(drop=True)
516 dgs_code21=pd.Series(dtype=str)
517 for i in range(len(goods_21)):
518     dgs_code21=dgs_code21.append(pd.Series(str(goods_21[i]).split()
    [0]), ignore_index=True)

```

```

519 counts_21=dgs_code21.value_counts()
520
521 #2021 occurrences with 'counts' higher than 1 are linked to attached
    files , but for 2021 the file names have variations.
522 #2021plus
523 #getting all the sources listed
524 goods_pos=dgs_2021[(dgs_2021['counts']!=1) & (dgs_2021['counts']!=0)
    ]['source'].str.lstrip()
525
526 good=pd.Series(dtype=str)
527 goods_21plus=pd.Series(dtype=str)
528 error_count21=0 #difference to be considered in the check of total dg
    extracted compared to the sum of 'count'
529
530 for it in goods_pos: #for each source we defined the possible file
    name between the different variations
531     goods_source1='Dangerous_Goods/2021/'+it+'.xlsx'
532     goods_source2='Dangerous_Goods/2021/eE-MOR '+it+'.xlsx'
533     goods_source3='Dangerous_Goods/2021/EE-MOR '+it+'.xlsx'
534     goods_source4='Dangerous_Goods/2021/'+it+' (modificato).xlsx'
535     goods_source5='Dangerous_Goods/2021/EE-MOR '+it+'.xlsx'
536     goods_source6='Dangerous_Goods/2021/eE-MOR '+it+'.xlsx'
537
538     if os.path.isfile(goods_source1): #we verify which of the
    possible names exists in the directory and then read the file
539         good=pd.read_excel(goods_source1)['Unnamed: 3'].dropna().
    reset_index(drop=True).drop(index=0)
540         elif os.path.isfile(goods_source2):
541             good=pd.read_excel(goods_source2)['Unnamed: 3'].dropna().
    reset_index(drop=True).drop(index=0)
542         elif os.path.isfile(goods_source3):
543             good=pd.read_excel(goods_source3)['Unnamed: 3'].dropna().
    reset_index(drop=True).drop(index=0)
544         elif os.path.isfile(goods_source4):
545             good=pd.read_excel(goods_source4)['Unnamed: 3'].dropna().
    reset_index(drop=True).drop(index=0)
546         elif os.path.isfile(goods_source5):
547             good=pd.read_excel(goods_source5)['Unnamed: 3'].dropna().
    reset_index(drop=True).drop(index=0)
548         elif os.path.isfile(goods_source6):
549             good=pd.read_excel(goods_source6)['Unnamed: 3'].dropna().
    reset_index(drop=True).drop(index=0)
550         else:
551             print(f'Name file not recognized {it}') #if the name is
    different or doesn't exist this is this check
552
553 #adding a check about the length of the attached files and the
    number reported in the 'count' columns

```

```

554     num_count=dgs_2021[dgs_2021['source'].str.contains(it)][ 'counts'
555     ].values
556
557     if (len(good)!=num_count).all():
558         print(f'The attached file {it} contains {len(good)} instead
559         of {num_count} reported')
560         error_count21=error_count21+(num_count-len(good))
561
562     goods_21plus=goods_21plus.append(good)
563
564     goods_21plus=pd.Series(list(chain(*goods_21plus.str.split('\n').
565     tolist()))).str.replace(' ','').str.replace(' ','').sort_values(
566     ascending=True).reset_index(drop=True)
567     dgs_code21plus=pd.Series(dtype=str)
568     for i in range(len(goods_21plus)):
569         dgs_code21plus=dgs_code21plus.append(pd.Series(str(goods_21plus[i
570         ]).split()[0]), ignore_index=True) #taking the UN code
571
572     #2021 TOTAL : adding together 2021 and 2021plus codes list and
573     coutig them.
574     counts_21tot=pd.DataFrame(dgs_code21.append(dgs_code21plus).
575     value_counts(), columns=['2021']).astype(str)
576
577     #CHECK AND VALIDATIONS
578     #for each year we count the number of total rows we get from the
579     Excel files and the total DGs we get after the manipulation: they
580     can be different because in one Excel cell can be more DGs. Then,
581     we compare the two numbers with those of blank cells, and those of
582     multiple data cells.
583
584     data_dgs=[dgs_2014, dgs_2015, dgs_2016, dgs_2017, dgs_2018, dgs_2019,
585     dgs_2020, dgs_2021]
586     data_code=[dgs_code14, dgs_code15, dgs_code16, dgs_code17, dgs_code18
587     , dgs_code19, dgs_code20, dgs_code21,dgs_code20plus,dgs_code21plus
588     ]
589
590     cell_blank=pd.DataFrame(index=['blank cells'], columns=years) #new df
591     to contain number of blank cells
592
593     for year, df in zip(years, data_dgs):
594
595         total_reports=df['counts'].sum() #total number of rows in Excel
596         Files
597         print(f'Total reports of Dangeruos Goods in {year}: {
598         total_reports}')
599
600         if year=='2014'or year=='2015'or year=='2016'or year=='2017'or
601         year=='2018'or year=='2019':

```

```

584     multiple=list(df[df['goods type'].fillna(' ').str.contains('\n')].index) #delete blank cells and list which cells as \n
585
586     total_dg=len(data_code[years.index(year)]) #total number of
dg after manipulation
587     print(f'Total number of dangerous goods extracted in {year}:
{total_dg}')
588
589     cell_blank[year]=df['goods type'].isna().sum() #number of
blank cell to be consider
590
591     if year=='2020'or year=='2021': #for 2020 and 2021 just 'count'=1
is evaluted (attached files souldn't have this necessity)
592     multiple1=df[df['counts']==1].reset_index(drop=True) #I
remove dropna() differently than what i did before because it gave
me an empty df
593     multiple=list(multiple1[multiple1['goods type'].fillna(' ').
str.contains('\n')].index)
594     df=multiple1
595
596     total_dg=len(data_code[years.index(year)]+len(data_code[
years.index(year)+2]) #total number of dg after manipulation ('+2'
is to indicate the index of 'plus')
597     print(f'Total number of dangerous goods extracted in {year}:
{total_dg}')
598
599     cell_blank[year]=df['goods type'].isna().sum() #number of
blank cell to be consider when 'count'=1 !!!!
600
601     num_dg_2=0 #initiating variables: _2: two event types, _3: three
event types ...
602     num_dg_3=0
603     num_dg_4=0
604     num_dg_5=0
605     num_dg_6=0
606     for i in multiple:
607         ex=df['goods type'][i].count('\n') #checkinh how many \n
608         if ex==1: #two types together
609             num_dg_2=num_dg_2+1
610         if ex==2: #three types together
611             num_dg_3=num_dg_3+1
612         if ex==3: #...
613             num_dg_4=num_dg_4+1
614         if ex==4:
615             num_dg_5=num_dg_5+1
616         exx=df['goods type'][i].count('\n\n') #checkinh how many \n\n
consecutively
617         if exx==1:
618             num_dg_3=num_dg_3+1 #no three types

```

```

619         num_dg_2=num_dg_2+1 #yes two types
620     if exx==2:
621         num_dg_5=num_dg_5-1
622         num_dg_3=num_dg_3+1
623     if exx==5:
624         num_dg_6=num_dg_6+1
625     if exx!=1 and exx!=2 and exx!=5 and exx!=0 and ex>4:
626         print('Check the count!') #check in case new arrangement
627 happens
628     print(f'Total additional DG types: {num_dg_2 +2*(num_dg_3)+ 3*(
629 num_dg_4)+4*(num_dg_5)+5*(num_dg_6)}')
630     print(f'2 DG type: {num_dg_2}')
631     print(f'3 DG type: {num_dg_3}')
632     print(f'4 DG type: {num_dg_4}')
633     print(f'5 DG type: {num_dg_5}')
634     print(f'6 DG type: {num_dg_6}')
635     print(f'Blank cells: {cell_blank[year].sum()}')
636
637     if year=='2014'or year=='2015'or year=='2016'or year=='2017'or
638 year=='2018'or year=='2019':
639         if total_dg+ cell_blank[year].sum()!=total_reports + num_dg_2
640 +2*(num_dg_3)+ 3*(num_dg_4)+4*(num_dg_5)+5*(num_dg_6):
641             print('Missing data!') #warning in case of miscounting
642             print(' ')
643         elif year=='2020': #taking into account 'count' column errors
644             print(f'Number of missing counts:{abs(error_count20)}')
645             if total_dg+ cell_blank[year].sum()!=total_reports + num_dg_2
646 +2*(num_dg_3)+ 3*(num_dg_4)+4*(num_dg_5)+5*(num_dg_6)-
647 error_count20:
648                 print('Missing data!')
649                 print(' ')
650         elif year=='2021': #taking into account 'count' column errors
651             print(f'Number of missing counts:{abs(error_count21)}')
652             if total_dg+ cell_blank[year].sum()!=total_reports + num_dg_2
653 +2*(num_dg_3)+ 3*(num_dg_4)+4*(num_dg_5)+5*(num_dg_6)-
654 error_count21:
655                 print('Missing data!')
656                 print(' ')
657
658 dgs_allyears=pd.DataFrame() #new dataframe where insert all the data
659 counts
660 dgs_allyears=pd.concat([counts_14,counts_15,counts_16,counts_17,
661 counts_18,counts_19,counts_20tot,counts_21tot],axis=1).fillna('0')
662 .sort_index()
663 dgs_allyears.reset_index(inplace=True)
664 dgs_allyears = dgs_allyears.rename(columns = {'index':'code'}) #
665 naming correctly codes columns.
666 dgs_allyears

```

```

656
657 #ICAO CODES READING
658 page=list(range(2,235)) #up to the next page in order to end at 235
659 file1 = "Dangerous_Goods/doc_9284_2019_2020.pdf"
660 table = tabula.read_pdf(file1 ,pages=page)
661 df=table #creating a dataframe with the data of the pages, table[0]
        corresponds to the first page etc..
662
663 #each table has all the interesting data at the second row in '
        Unnamed :0' and 'UN/rNo.' columns.
664 #from a check control below, we could verified that some pages didn't
        get read correctly with the above method, so for these pages (in
        this case 130,131,220) the method is adapted.
665
666 #PAGE 130
667 table1 = tabula.read_pdf(file1 ,pages=130)
668 df1=table1[0]
669 pg130=df1[['Unnamed: 2', 'Unnamed: 3']].dropna().reset_index(drop=True
        ).drop(index=[0,1])
670 #PAGE 131
671 table2 = tabula.read_pdf(file1 ,pages=131)
672 df2=table2[0]
673 pg131=df2[['Unnamed: 2', 'Unnamed: 3']].dropna().reset_index(drop=True
        ).drop(index=[0,1])
674 #PAGE 220
675 table3 = tabula.read_pdf(file1 ,pages=220)
676 df3=table3[0]
677 pg220=df3[['Unnamed: 2', 'Unnamed: 3']].dropna().reset_index(drop=True
        ).drop(index=[0,1])
678
679 more=pg130.append([pg131, pg220]).reset_index(drop=True).rename({'
        Unnamed: 2': 'codes', 'Unnamed: 3': 'classes'}, axis=1)
680 more['classes']=more['classes'].str.split('.', expand=True)[0]
681
682 #new variables
683 codes=pd.Series(dtype=str) #auxiliary variable: will become a columns
        of the dataframe
684 classes=pd.Series(dtype=str) #auxiliary variable: will become a
        columns of the dataframe
685 codes_icao=pd.DataFrame() # will contains codes and classes from the
        ICAO document
686
687 for i in range(0,len(page)): #both classes and codes need to be
        separated at \r
688     codes=codes.append(pd.Series(str(df[i].iloc[2,0]).split('\r'))).
        reset_index(drop=True)
689     classes=classes.append(pd.Series(str(df[i].iloc[2,1]).split('\r')
        )).reset_index(drop=True)
690

```

```

691 #What we actually need is just the general classes and not the sub-
        classes , so we can remove the number after the point and delete
        possible duplicates
692 for i in classes:
693     if len(i)>1:
694         i=i.split('.')[0]
695
696 #in the ICAO document there are some rows with 'FORBIDDEN' and those
        have to be deleted
697 codes_icao['codes']=codes[codes!='FOR']
698 codes_icao['classes']=classes[classes!='IDDE'].str.split('.',expand=
        True)[0]
699 codes_icao=codes_icao.drop_duplicates(keep='first',ignore_index=True)
        .append(more, ignore_index=True)
700 #=====
701 codes_icao.to_excel('Analysis_Excel/codes_icao.xlsx')
702 #=====
703 codes_icao
704
705 #MATCHING
706 def matching (x): #defining a function to match from the code the
        corresponding class
707     if codes_icao['codes'].str.contains(x).any(): #verifying if the
        string exist in the ICAO pdf
708
709         for i in range(0,len(codes_icao['codes'])):
710             if (x==codes_icao['codes'][i]):
711                 match=codes_icao['classes'][i]
712     else:
713         match='Notfound'
714     return match
715
716 matches=pd.Series(dtype=str)
717
718 for i in dgs_allyears['code']:
719     if len(i)<2: #classes already defined
720         et=i
721     elif len(i)==4: #codes of 4 numbers
722         et=matching(i)
723     elif i=='Battery,': #a specific case that needs to be considered
        individually
724         et='9'
725     elif i=='Not': #Not defined
726         et='ND'
727     elif i=='UNK': #Unknown
728         et='ND'
729     elif i=='radioactive': #a specific case that needs to be
        considered individually
730         et='7'

```

```

731     else:
732         (f' {i} not identified')
733         matches=matches.append(pd.Series(et)).reset_index(drop=True)
734
735     dgs_allyears['class']=matches #resulting table
736     #=====
737     dgs_allyears.to_excel('Analysis_Excel/dg_types_all.xlsx')
738     #=====
739     display(dgs_allyears)
740
741     #From the matching, codes not found in ICAO document are listed below
742     . This is a check to understand if the ICAO document is read
743     correctly: if the Series in EMPTY no mistakes have been committed,
744     on the contrary if there are nay codes showing a manual
745     intervention is needed.
746     print('Codes not found in Icao document are:')
747     print(dgs_allyears[dgs_allyears['class']=='Notfound']['code'])
748     print('Please check pdf reading or new versions of the document.')
749
750     #GROUPING
751     new_classes=['class 1: Explosives', 'class 2: Flammable Gases','class
752                 3: Flammable Liquids', 'class 4:Flammable Solids', 'class 5:
753                 Oxidizing', 'class 6: Toxic & Infectious', 'class 7: Radioactive',
754                 'class 8: Corrosives', 'class 9: Miscellaneous – other than Li-
755                 BAT', 'class 9–Bat: Miscellaneous – Li–BAT', 'ND: Not defined']
756
757     dgs_inclasses=pd.DataFrame(index=new_classes, columns=years) #new
758     dataframe to collect data divided in classes and years.
759
760     #Class 1: Explosives
761     cl1=dgs_allyears[dgs_allyears['class']=='1'].index
762     dgs_inclasses.loc[new_classes[0]]=dgs_allyears.loc[cl1][years].astype
763     (int).sum()
764
765     #Class 2: Flammable Gases
766     cl2=dgs_allyears[dgs_allyears['class']=='2'].index
767     dgs_inclasses.loc[new_classes[1]]=dgs_allyears.loc[cl2][years].astype
768     (int).sum()
769
770     #Class 3: Flammable Liquids
771     cl3=dgs_allyears[dgs_allyears['class']=='3'].index
772     dgs_inclasses.loc[new_classes[2]]=dgs_allyears.loc[cl3][years].astype
773     (int).sum()
774
775     #Class 4:Flammable Solids
776     cl4=dgs_allyears[dgs_allyears['class']=='4'].index
777     dgs_inclasses.loc[new_classes[3]]=dgs_allyears.loc[cl4][years].astype
778     (int).sum().astype(int)
779
780

```



```

767 #Class 5: Oxidizing
768 cl5=dgs_allyears[dgs_allyears['class']=='5'].index
769 dgs_inclasses.loc[new_classes[4]]=dgs_allyears.loc[cl5][years].astype
    (int).sum()
770
771 #Class 6: Toxic & Infectious
772 cl6=dgs_allyears[dgs_allyears['class']=='6'].index
773 dgs_inclasses.loc[new_classes[5]]=dgs_allyears.loc[cl6][years].astype
    (int).sum().astype(int)
774
775 #Class 7: Radioactive
776 cl7=dgs_allyears[dgs_allyears['class']=='7'].index
777 dgs_inclasses.loc[new_classes[6]]=dgs_allyears.loc[cl7][years].astype
    (int).sum()
778
779 #Class 8: Corrosives
780 cl8=dgs_allyears[dgs_allyears['class']=='8'].index
781 dgs_inclasses.loc[new_classes[7]]=dgs_allyears.loc[cl8][years].astype
    (int).sum()
782
783 #Class 9: Miscellaneous – other than Li–BAT /Class 9–Bat:
    Miscellaneous – Li–BAT
784 cl9=dgs_allyears[(dgs_allyears['class']=='0') | (dgs_allyears['class']
   )=='9').reset_index(drop=True) #filter in class 9 but also for
    code=0 which is 'a manual operation by enac'
785 miscthan=['1845','2807','3077','3082','3316','3363','8000','9','2990'
    , '3166'] #codes to be written MANUALLY
786 misc=['0','3090','3091','3171','3480','3481','Battery,'] #codes to be
    written MANUALLY
787 cl9miscthan=pd.DataFrame(columns=years)
788 cl9misc=pd.DataFrame(columns=years)
789 for i in range(0,len(cl9)):
790     if cl9['code'][i] in miscthan:
791         cl9miscthan=cl9miscthan.append(cl9.iloc[i])
792     if cl9['code'][i] in misc:
793         cl9misc=cl9misc.append(cl9.iloc[i])
794     elif cl9['code'][i] not in miscthan and cl9['code'][i] not in
    misc:
795         print(cl9['code'][i], 'is not considered in the classification
    of Class 9. A manual operation is needed.')
796
797 dgs_inclasses.loc[new_classes[8]]=cl9miscthan[years].astype(int).sum
    ()
798 dgs_inclasses.loc[new_classes[9]]=cl9misc[years].astype(int).sum()
799
800 #Class ND: Not defined
801 clnd=dgs_allyears[dgs_allyears['class']=='ND'].index
802 dgs_inclasses.loc[new_classes[10]]=dgs_allyears.loc[clnd][years].
    astype(int).append(cell_blank).sum() #Adding blank cells

```

```

803
804 #Dataframe with total count per year (uncomment following rows if
      necessary)
805 #total_counts=pd.DataFrame(index=['Total'], columns=years)
806 #total_counts[years]=dgs_inclasses[years].sum()
807 #dgs_inclasses=dgs_inclasses.append(total_counts)
808 #
809 dgs_inclasses.to_excel('Analysis_Excel/dg_types_collapsed.xlsx')
810 #
811 dgs_inclasses
812
813 #GRAPHICS
814 #
815 # Pie Charts about Event Types
816 #
817 fig, ax = plt.subplots(2, 4, figsize=(12, 6)) #creating figure box
818 fig.suptitle('Dangerous Goods Types', y=1, fontsize=28) #title
819
820 color=['lightcoral', 'red', 'chocolate', 'sienna', 'gold', 'khaki', '
      yellowgreen', 'forestgreen', 'lightblue', 'cornflowerblue', 'violet']
      #colors
821 #
822 ax = ax.ravel()
823
824 for i in range(len(years)): #using a loop to draw every pie chart
825
826     axes = ax[i]
827
828     axes.pie(dgs_inclasses[years[i]], colors=color, shadow=False,
      labeldistance=1.1, startangle=90, radius=1.1, wedgeprops = {
      "edgecolor" : "black", 'linewidth': 0.3, 'antialiased': True} )
829
830     axes.set_title(years[i], fontsize=22, y=0.97)
831
832 legend=fig.legend(new_classes, loc='center left', bbox_to_anchor=(1,
      0.5), prop={'size':14}) #legend
833 #
834 fig.tight_layout()
835 fig.savefig('Graphics/dg_types/dg_type_general.pdf',
836             format = 'pdf',
837             dpi = 300)
838
839 fig.tight_layout()
840 fig.savefig('Graphics/dg_types/dg_type_general_withlgd.pdf',
      bbox_inches='tight', format = 'pdf', dpi = 300) #saving chart and
      legend
841
842 def export_legend(legend, filename='Graphics/dg_types/
      dg_types_ldg_pie.pdf'):

```

```

843     fig=legend.figure
844     fig.canvas.draw()
845     bbox=legend.get_window_extent().transformed(fig.dpi_scale_trans.
inverted())
846     fig.savefig(filename, dpi=300, bbox_inches=bbox)
847
848 export_legend(legend) #saving only the legend
849
850 #=====
851 # Bar Chart about DG types General
852 #=====
853 fig, ax = plt.subplots(1,2, sharex = False, sharey=False, figsize
=(14, 6)) #creating figure box
854 fig.suptitle('Dangerous Goods Types', y=1.05, fontsize=28) #title
855 axisx=np.arange(len(years))
856 #=====
857 dgs_inclasses[years].sum().plot.bar(stacked=True, ax=ax[0]).grid(axis
='y')
858 ax[0].set_title('Trend overtime – Decimal Scale', fontsize=16, pad=20)
859 #=====
860 dgs_inclasses[years].sum().plot.bar(stacked=True, logy=True, ax=ax
[1]).grid(axis='y')
861 ax[1].set_title('Trend overtime – Log Scale', fontsize=16, pad=20)
862 #=====
863 plt.rc('xtick', labels=16)
864 plt.rc('ytick', labels=16)
865 ax[0].set_ylabel('Occurrences', fontsize=18) #axis title
866 ax[1].set_ylabel('Occurrences', fontsize=18) #axis title
867
868 fig.savefig('Graphics/dg_types/dg_types_overtime_withlog.pdf',
bbox_inches='tight', format = 'pdf', dpi = 300)
869
870 #=====
871 #Stacked Bar chart with percentage for years
872 #=====
873 fig, ax = plt.subplots(1, 1, figsize=(12, 8)) #creating figure box
874 fig.suptitle('Dangerous Goods Types', fontsize=28) #title
875
876 color=['lightcoral', 'red', 'chocolate', 'sienna', 'gold', 'khaki', '
yellowgreen', 'forestgreen', 'lightblue', 'cornflowerblue', 'violet']
#colors
877 #=====
878 (dgs_inclasses/dgs_inclasses.sum()*100).T.plot.barh(stacked=True,
color=color, legend=True, lw=0, width = 0.95, ax=ax)
879 legend=ax.legend(new_classes, loc='center left', bbox_to_anchor=(1,
0.5)) #legend
880 ax.set_ylabel('Years', fontsize=18) #axis title
881 ax.spines['top'].set_visible(False) #deleting top box line
882 ax.spines['right'].set_visible(False) #deleting right box line

```

```

883 ax.spines['bottom'].set_visible(False) #deleting bottom box line
884 ax.grid(axis='x') #shwoing only x grid
885 plt.rc('xtick', labelsiz=18)
886 plt.rc('xtick', labelsiz=18)
887
888 fmt = '%.0f%%' #setting percentages on x axis
889 xticks = mtick.FormatStrFormatter(fmt)
890 ax.xaxis.set_major_formatter(xticks)
891 #=====
892 ax.legend().set_visible(False)
893 fig.tight_layout()
894 fig.savefig('Graphics/dg_types/dg_types_stackedbar.pdf', bbox_inches=
    'tight', format = 'pdf', dpi = 300)#saving chart without legend
895
896 ax.legend(new_classes, loc='center left', bbox_to_anchor=(1, 0.5)).
    set_visible(True)
897 fig.savefig('Graphics/dg_types/dg_types_stackedbar_withlgd.pdf',
    bbox_inches='tight', format = 'pdf', dpi = 300)#saving chart plus
    legend
898
899 def export_legend(legend, filename='Graphics/dg_types/
    dg_types_stackedbar_lgd.pdf'):
900     fig=legend.figure
901     fig.canvas.draw()
902     bbox=legend.get_window_extent().transformed(fig.dpi_scale_trans.
        inverted())
903     fig.savefig(filename, dpi=300, bbox_inches=bbox)
904
905 export_legend(legend) #saving only the legend
906
907 #=====
908 #Stacked Bar chart with percentage for DGs
909 #=====
910 #defining a new dataframe to use in plotting(Problem in division by
    zero)
911 data_graph=pd.DataFrame(index=years, columns=new_classes)
912 for i in new_classes:
913     if dgs_inclasses.T[i].sum()!=0:
914         data_graph[i]=dgs_inclasses.T[i]/dgs_inclasses.T[i].sum()*100
915     else:
916         data_graph[i]=0
917
918 fig, ax = plt.subplots(1, 1, figsize=(18, 10)) #creating figure box
919 fig.suptitle('Dangerous Goods Types', fontsize=26) #title
920
921 color=['lightcoral', 'red', 'chocolate', 'sienna', 'gold', 'khaki', '
    yellowgreen', 'forestgreen', 'lightblue', 'cornflowerblue', 'violet']
    #colors
922 #=====

```

```

923 (data_graph).T.plot.barh(stacked=True,color=color, legend=True, lw
    =0, width = 0.95, ax=ax)
924 legend=ax.legend(years, loc='center left', bbox_to_anchor=(1, 0.5)) #
    legend
925 ax.set_ylabel('Classes', fontsize=18) #axis title
926 ax.spines['top'].set_visible(False) #deleting top box line
927 ax.spines['right'].set_visible(False) #deleting right box line
928 ax.spines['bottom'].set_visible(False) #deleting bottom box line
929 ax.invert_yaxis()
930 ax.grid(axis='x') #shwoing only x grid
931 plt.rc('xtick', labels=18)
932 plt.rc('ytick', labels=18)
933
934 fmt = '%.0f%%' #setting percentages on x axis
935 xticks = mtick.FormatStrFormatter(fmt)
936 ax.xaxis.set_major_formatter(xticks)
937 #=====
938 ax.legend().set_visible(False)
939 fig.savefig('Graphics/dg_types/dg_types_stackedbar2.pdf', bbox_inches
    ='tight', format = 'pdf',dpi = 300) #saving chart plus legend
940
941 ax.legend(years, loc='center left', bbox_to_anchor=(1, 0.5)).
    set_visible(True)
942 fig.savefig('Graphics/dg_types/dg_types_stackedbar_withlgd2.pdf',
    bbox_inches='tight', format = 'pdf', dpi = 300)#saving chart plus
    legend
943
944 def export_legend(legend, filename='Graphics/dg_types/
    dg_types_stackedbar_lgd2.pdf'):
945     fig=legend.figure
946     fig.canvas.draw()
947     bbox=legend.get_window_extent().transformed(fig.dpi_scale_trans.
    inverted())
948     fig.savefig(filename, dpi=300, bbox_inches=bbox)
949
950 export_legend(legend) #saving only the legend

```


Appendix B

Python-based Pipeline: SPI template

This appendix contains the Python code of the Safety Performance Indicators analysis.

```
1 import pandas as pd
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import matplotlib.colors as mcol
5 import matplotlib as mpl
6 import seaborn as sns
7 from itertools import chain #to append lists of list
8 import matplotlib.ticker as mtick #to show percentage ticks in
    matplotlib graphs
9 import os.path #to verify the existence of a file in a specific
    directory
10 import tabula #to read pdf files
11 from matplotlib.gridspec import SubplotSpec #to title every row of in
    figure with subplots
12
13 #Insert the requested SPI to analyze
14 subject_required='Runway Incursions' #the user decides which analysis
    to run
15
16 #According to the user choice, a file is selected to be read through
    if-conditions. Given the extended name of a SPI, the code acquire
    the contracted name (used for naming output files) and the source
    file name.
17
```



```

51 |
52 | #READING DATA
53 | #extracting the data for each year
54 | years=['2015','2016','2017','2018','2019','2020'] #considered years
55 | data=pd.DataFrame() #new DataFrame
56 | check_length=pd.DataFrame(index=['black cells','length'], columns=
    years) #new DataFrame for validation
57 |
58 | for i in range(len(years)):
59 |
60 |     data_excel=pd.read_excel('SPIs/'+file_name,sheet_name=years[i]) #
    reading data from excel file sheets
61 |     data=data.append(data_excel) #appending all the data together
62 |
63 |     #calculating the length of the Excel sheet and the the number of
    blank cells per sheet
64 |     check_length.iloc[0,i]=data_excel.iloc[:,5].isna().sum()
65 |     check_length.iloc[1,i]=len(data_excel.iloc[:,5])
66 |
67 | #keeping only the column with the required information about dates,
    converting data in date-format and keeping only the information
    about the year
68 | data_date=pd.to_datetime(data.iloc[:,5].dropna().reset_index(drop=
    True)).dt.year
69 |
70 | #For each SPI categories we run a different analysis
71 | if name in spi_mvt:
72 |     #resulting DataFrame
73 |     data_tab=pd.DataFrame(columns=['Movements','Number of events','
    Rate'])
74 |     data_tab['Number of events']=data_date.value_counts().sort_index(
    ascending=True) #counting the total number of occurrences
75 |     data_tab['Movements']=movements
76 |     #rate per 10000 movements, rounding at 2 decimal digits
77 |     data_tab['Rate']=round(data_tab['Number of events']/data_tab['
    Movements']*10000,2)
78 |
79 |     variation_name='every100000mvt' #specific name for saving the
    data
80 |
81 | elif name in spi_flt:
82 |     #resulting DataFrame
83 |     data_tab=pd.DataFrame(columns=['Flights','Number of events','Rate
    '])
84 |     data_tab['Number of events']=data_date.value_counts().sort_index(
    ascending=True) #counting the total number of occurrences
85 |     data_tab['Flights']=flights
86 |     #rate per 10000 movements, rounding at 2 decimal digits

```

```

87     data_tab['Rate']=round(data_tab['Number of events']/data_tab['
    Flights']*10000,2)
88
89     variation_name='every10000flt' #specific name for saving the data
90
91 elif name in spi_dur:
92     #resulting DataFrame
93     data_tab=pd.DataFrame(columns=['Occupancy duration','Number of
    events','Rate'])#counting the total number of occurrences
94     data_tab['Number of events']=data_date.value_counts().sort_index(
    ascending=True)
95     data_tab['Occupancy duration']=duration
96     #rate per 1Mln minutes, rounding at 2 decimal digits
97     data_tab['Rate']=round(data_tab['Number of events']/data_tab['
    Occupancy duration']*1000000,2)
98
99     variation_name='every1Mlnmins' #specific name for saving the data
100
101 elif name in spi_occ:
102     #resulting DataFrame
103     data_tab=pd.DataFrame(columns=['Number of events'])
104     data_tab['Number of events']=data_date.value_counts().sort_index(
    ascending=True)#counting the total number of occurrences
105
106     variation_name='' #no addition information in the output file
    name is required
107 else:
108     print('Input data do not corresond to study cases')
109
110 #implementing a check to verify the number of events extracted
    corresponds to the total events reported
111 for i in range(len(years)):
112     if data_tab.iloc[i,1]!=check_length.diff().iloc[1,i]:
113         print('Missing data!')
114
115 data_tab.to_excel('Analysis_Excel/'+name+variation_name+'.xlsx') #
    saving the resulting table in an Excel file
116
117 display(data_tab)
118
119 #According to the chosen SPI, one of the three graphs below will be
    performed.
120
121 # Movements Trends
122
123 if name in spi_mvt:
124
125     fig, ax = plt.subplots(1,1, sharex = False, sharey=False,
    figsize=(8, 6)) #creating a figure box

```

```

126     fig.suptitle('Movements per year', fontsize=28) #title
127     axisx=np.arange(len(years)) #defining x positions
128     #=====
129     ax.plot(years, movements/1000000, 'go—', linewidth=2, markersize
130             =12)
131
132     ax.spines['top'].set_visible(False) #deleting top box line
133     ax.spines['right'].set_visible(False) #deleting right box line
134     ax.set_ylabel('Movements (in Mln)', fontsize=18) #axis title
135     ax.grid(axis='y') #showing only y grid
136     plt.rc('xtick', labels=18)
137     plt.rc('ytick', labels=18)
138     plt.ylim((0.4, 2))
139     #=====
140     fig.savefig('Graphics/spi/SPI_movperyear.pdf', format = 'pdf', dpi
141               = 300)#saving
142     #=====
143     # Flights Trends
144     #=====
145     elif name in spi_flt:
146
147         fig, ax = plt.subplots(1,1, sharex = False, sharey=False,
148                               figsize=(8, 6))#creating a figure box
149         fig.suptitle('Flights per year', fontsize=28) #title
150         axisx=np.arange(len(years)) #defining x positions
151         #=====
152         ax.plot(years, flights/1000000, 'go—', linewidth=2, markersize=12)
153
154         ax.spines['top'].set_visible(False) #deleting top box line
155         ax.spines['right'].set_visible(False) #deleting right box line
156         ax.set_ylabel('Flights (in Mln)', fontsize=18) #axis title
157         ax.grid(axis='y') #showing only y grid
158         plt.rc('xtick', labels=18)
159         plt.rc('ytick', labels=18)
160         plt.ylim((0.5, 2.1))
161         #=====
162         fig.savefig('Graphics/spi/SPI_flightperyear.pdf', format = 'pdf',
163                   dpi = 300)
164         #=====
165         # Occupancy Duration
166         #=====
167         elif name in spi_dur:
168
169             fig, ax = plt.subplots(1,1, sharex = False, sharey=False,
170                                   figsize=(8, 6)) #creating a figure box
171             fig.suptitle('Occupancy Duration (min)', fontsize=28) #title
172             axisx=np.arange(len(years)) #defining x positions
173             #=====

```

```

169     ax.plot(years,duration/1000000,'go—',linewidth=2,markersize
170             =12)
171
172     ax.spines['top'].set_visible(False) #deleting top box line
173     ax.spines['right'].set_visible(False) #deleting right box line
174     ax.set_ylabel('Occupancy Duration (in Mln of min)',fontsize=18) #
175     axis title
176     ax.grid(axis='y') #showing only y grid
177     plt.rc('xtick', labels=18)
178     plt.rc('ytick', labels=18)
179     #plt.ylim((0.4e6,2e6))
180     #
181     fig.savefig('Graphics/spi/SPI_durationmin.pdf', format = 'pdf',
182               dpi = 300)
183
184 #OCCURRENCES and RATE (per year) GRAPHICS
185 #
186 if name in spi_mvt+spiflt+spidur: #plotting number of occurrences
187     and rate
188
189     fig, ax = plt.subplots(1,2, sharex = False, sharey=False,
190                           figsize=(16, 8)) #creating a figure box
191     fig.suptitle(subject, fontsize=28,y=1) #title
192     axisx=np.arange(len(years)) #defining x positions
193     color=['lightcoral','red'] #colors
194     topic=['Number of events','Rate'] #topic to visualize
195     #
196     ax=ax.ravel()
197
198     for i in range(len(topic)): #using a loop to plot the tw bar
199         plots
200
201         axes = ax[i]
202
203         axes.bar(x=axisx,
204                  height=data_tab[topic[i]],
205                  width=0.4,
206                  tick_label=years,
207                  color=color[i])
208         axes.set_title(topic[i], fontweight='bold',fontsize=24)
209         axes.grid(axis='y') #showing only y grid
210
211     #plt.ylim((0,24))
212     plt.rc('xtick', labels=20) #setting x ticks size
213     plt.rc('ytick', labels=20) #setting y ticks size
214     ax[0].set_ylabel('Occurrences', fontsize=18) #axis title
215     ax[1].set_ylabel('Rate', fontsize=18) #axis title
216     #
217     fig.tight_layout()

```

```

212     fig.savefig('Graphics/spi/SPI_occrateperyear_'+name+'.pdf',
213               format = 'pdf', dpi = 300)#saving
214 elif name in spi_occ: #plotting only the total number of occurrences
215
216     fig, ax = plt.subplots(1,1, sharex = False, sharey=False,
217                           figsize=(12, 6)) #creating a figure box
218     fig.suptitle(subject, fontsize=28,y=1) #title
219     color='lightcoral' #color
220     topic='Number of events'
221
222     ax.bar([1,2,3,4,5,6],height=data_tab[topic],width=0.4, tick_label
223            =years, color=color)
224     ax.set_title('Occurrences per year', fontweight='bold')
225     ax.grid(axis='y') #showing only y grid
226     plt.rc('xtick', labels=20) #setting x ticks size
227     plt.rc('ytick', labels=20) #setting y ticks size
228     ax.set_ylabel('Occurrences', fontsize=18) #axis title
229     #=====
230     fig.tight_layout()
231     fig.savefig('Graphics/spi/SPI_occrperyear_'+name+'.pdf', format =
232               'pdf',dpi = 300)#saving
233
234 #As a new way of visualisation, we present a different type of graph
235 # for plotting rates.
236 # Rate Trends with line plots
237 #=====
238 if name in spi_mvt+spi_flt+spi_dur:
239
240     fig, ax = plt.subplots(1,1, sharex = False, sharey=False,
241                           figsize=(8, 6)) #creating a figure box
242     fig.suptitle('Rate per year ('+name+)', fontsize=28) #title
243     axisx=np.arange(len(years)) #defining x positions
244     #=====
245     ax.plot(years,data_tab['Rate'],'go—', linewidth=2, markersize
246            =12)
247
248     ax.spines['top'].set_visible(False) #deleting top box line
249     ax.spines['right'].set_visible(False) #deleting right box line
250     ax.set_ylabel('Rate',fontsize=18) #axis title
251     ax.grid(axis='y') #showing only y grid
252     plt.rc('xtick', labels=18) #setting x ticks size
253     plt.rc('ytick', labels=18) #setting y ticks size
254     #=====
255     fig.savefig('Graphics/spi/SPI_rateperyear_'+name+'_line.pdf',
256               format = 'pdf', dpi = 300)#saving
257
258 #EVENT TYPES ANALYSIS (plus plots)

```

```

253 #=====
254 if data.iloc[:,12].notnull().values.any(): #for now it means only RE
255
256     data_events=pd.DataFrame({'date': pd.to_datetime(data.iloc[:,5]),
257                               'event_types': data.iloc[:,12]}).dropna()
258     data_events['date']=data_events['date'].dt.year
259
260     event_types=['Runway Incursion by an Aircraft','Runway Incursion
261 by a Person','Runway Incursion by a Vehicle/Equipment']
262
263     events=pd.DataFrame(columns=['Aircraft','rate Aircraft x10000
264 mvts','Person','rate Person x10000 mvts','Vehicle/Equipment','rate
265 Vehicle/Equipment x10000 mvts'])
266
267     for i in range(len(event_types)):
268         events.iloc[:,i+i]=data_events[data_events['event_types']==
269 event_types[i]]['date'].value_counts().sort_index(ascending=True)
270         events.iloc[:,(i+i)+1]=round(events.iloc[:,i]/movements
271 *10000,3)
272     #=====
273     events.to_excel('Analysis_Excel/'+name+'_eventtypes.xlsx')
274     #=====
275     display(events)
276
277 #=====
278 #Bar charts
279 #=====
280 fig, ax = plt.subplots(3,2, sharey = False,  figsize=(14, 12))
281 fig.suptitle('Event Types ('+name+)', fontsize=28, y=1)
282 plt.subplots_adjust( hspace=0.9)
283 color=['lightcoral','red','chocolate','sienna','lightblue','
284 cornflowerblue'] #colors
285 axisx=np.arange(len(years))
286 #=====
287 ax = ax.ravel()
288
289 for i in range(len(events.columns)):
290
291     axes = ax[i]
292
293     axes.bar(x=axisx,height=events[events.columns[i]],width=0.5,
294 tick_label=years,align='center',color=color[i])
295     axes.set_title(events.columns[i], fontweight='semibold',
296 fontsize=18)
297     axes.grid(axis='y') #showing only y grid
298     if i==0 or i==2 or i==4:
299         axes.set_ylabel('Occurrences',fontsize=16)
300     elif i==1 or i==3 or i==5:
301         axes.set_ylabel('Rate',fontsize=16)

```

```

293 plt.rc('xtick', labelsizes=18)
294 plt.rc('ytick', labelsizes=20)
295
296 #=====
297 #function to insert title at every row
298 def create_subtitle(fig: plt.Figure, grid: SubplotSpec, title:
299 str):
300     "Sign sets of subplots with title"
301     row = fig.add_subplot(grid)
302     # the '\n' is important
303     row.set_title(f'{title}\n', fontweight='bold', fontsize=24, y
304 =1.1)
305     # hide subplot
306     row.set_frame_on(False)
307     row.axis('off')
308
309 rows = 3
310 cols = 2
311 grid = plt.GridSpec(rows, cols)
312 create_subtitle(fig, grid[0, :], event_types[0])
313 create_subtitle(fig, grid[1, :], event_types[1])
314 create_subtitle(fig, grid[2, :], event_types[2])
315
316 #=====
317 fig.tight_layout()
318 fig.savefig('Graphics/spi/SPI_eventtypes_'+name+'.pdf', format =
319 'pdf', dpi = 300)
320
321 else:
322     print('The data-set does not contain Event types.')
323
324 #EVENT TYPES GRAPHICS
325 if data.iloc[:,12].notnull().values.any():
326
327     #=====
328     #Bar charts stacked —> not with percentage because these
329     events types are just a few of the total
330     #=====
331     only_events=events[['Aircraft', 'Person', 'Vehicle/Equipment']]
332
333     fig, ax = plt.subplots(1, 1, figsize=(12, 8)) #creatng figure box
334     fig.suptitle('Event Types ('+name+')', fontsize=28) #title
335     color=['red', 'forestgreen', 'gold', 'lightblue', 'cornflowerblue'] #
336     colors
337     #=====
338     only_events.plot.barh(stacked=True, color=color, legend=True, lw
339 =0, width =0.95, ax=ax)

```

```

335     legend=ax.legend(event_types, loc='center left', bbox_to_anchor
336                       =(1, 0.5)) #legend
337     ax.set_ylabel('Years', fontsize=18) #axis title
338     ax.spines['top'].set_visible(False) #deleting top box line
339     ax.spines['right'].set_visible(False) #deleting right box line
340     ax.spines['bottom'].set_visible(False) #deleting bottom box line
341     ax.grid(axis='x') #shwoing only x grid
342     plt.rc('xtick', labels=18)
343     plt.rc('ytick', labels=18)
344     #=====
345     ax.legend().set_visible(False)
346     fig.savefig('Graphics/spi/SPI_eventtypes_'+name+'_stacknolgd.pdf',
347               , format = 'pdf', dpi = 300)#saving chart without legend
348
349     ax.legend(event_types, loc='center left', bbox_to_anchor=(1, 0.5)
350               ).set_visible(True)
351     fig.savefig('Graphics/spi/SPI_eventtypes_'+name+'_stack.pdf',
352               , format = 'pdf', dpi = 300)#saving chart plus
353               legend
354
355     def export_legend(legend, filename='Graphics/spi/SPI_eventtypes_'
356                       +name+'_stack_onlylgd.pdf'):
357         fig=legend.figure
358         fig.canvas.draw()
359         bbox=legend.get_window_extent().transformed(fig.
360             dpi_scale_trans.inverted())
361         fig.savefig(filename, dpi=300, bbox_inches=bbox)
362
363     export_legend(legend) #saving only the legend
364     #=====
365     #Line plot of rate trends
366     #=====
367     fig, ax = plt.subplots(1,1, sharey = False, figsize=(12,6))
368     fig.suptitle('Event Types rates ('+name+')', fontsize=28)
369     plt.subplots_adjust(hspace=0.9)
370     color=['lightcoral', 'red', 'chocolate', 'sienna', 'lightblue', '
371           cornflowerblue'] #colors
372
373     axisx=np.arange(len(years))
374     #=====
375     axes= ax
376
377     axes.plot(years, events[events.columns[1]], color=color[1], marker=
378     'o', linestyle='dashed', linewidth=2, markersize=12)
379     axes.plot(years, events[events.columns[3]], color=color[3], marker=
380     'o', linestyle='dashed', linewidth=2, markersize=12)
381     axes.plot(years, events[events.columns[5]], color=color[5], marker=
382     'o', linestyle='dashed', linewidth=2, markersize=12)

```



```

373 axes.spines['top'].set_visible(False) #deleting top box line
374 axes.spines['right'].set_visible(False) #deleting right box line
375 axes.set_ylabel('Rate',fontsize=18) #axis title
376 axes.grid(axis='y') #showing only y grid
377 legend=ax.legend(event_types, prop={'size': 14}) #legend
378 plt.rc('xtick', labels=18)
379 plt.rc('ytick', labels=18)
380 #
381 fig.savefig('Graphics/spi/SPI_ratestoggether_et_'+name+'.pdf',
format = 'pdf', dpi = 300)
382
383 #
384 #Bar charts and line plots
385 #
386 fig, ax = plt.subplots(3,2, sharey = False, figsize=(14, 12))
387 fig.suptitle('Event Types ('+name+)', fontsize=28, y=1)
388 plt.subplots_adjust( hspace=0.9, wspace=0.3)
389 color=['lightcoral','red','chocolate','sienna','lightblue','
cornflowerblue'] #colors
390 axisx=np.arange(len(years))
391 #
392 ax = ax.ravel()
393
394 for i in [0,2,4]:
395
396     axes = ax[i]
397
398     axes.bar(x=axisx, height=events[events.columns[i]], width
=0.5, tick_label=years, align='center', color=color[i])
399     axes.set_ylabel('Occurrences',fontsize=16)
400     axes.grid(axis='y') #showing only y grid
401     plt.rc('xtick', labels=18)
402     plt.rc('ytick', labels=18)
403
404     for i in [1,3,5]:
405
406         axes= ax[i]
407
408         axes.plot(years,events[events.columns[i]],color=color[i],
marker='o', linestyle='dashed', linewidth=2, markersize=12)
409         axes.spines['top'].set_visible(False) #deleting top box line
410         axes.spines['right'].set_visible(False) #deleting right box
line
411         axes.set_ylabel('Rate',fontsize=16) #axis title
412         axes.grid(axis='y') #showing only y grid
413 #
414 #function to insert title at every row
415 from matplotlib.gridspec import SubplotSpec

```

```

416     def create_subtitle(fig: plt.Figure, grid: SubplotSpec, title:
417     str):
418         "Sign sets of subplots with title"
419         row = fig.add_subplot(grid)
420         # the '\n' is important
421         row.set_title(f'{{title}}\n', fontweight='bold', fontsize=24)
422         # hide subplot
423         row.set_frame_on(False)
424         row.axis('off')
425
426     rows = 3
427     cols = 2
428     grid = plt.GridSpec(rows, cols)
429     create_subtitle(fig, grid[0, :], event_types[0])
430     create_subtitle(fig, grid[1, :], event_types[1])
431     create_subtitle(fig, grid[2, :], event_types[2])
432     #
433     fig.tight_layout()
434     fig.savefig('Graphics/spi/SPI_eventtypes_'+name+'__withline.pdf',
435     format='pdf', dpi=300)

```

Appendix C

Implementation of a Python-based dashboard

One of the conclusive output of this thesis is an interactive dashboard. A dashboard is a Web interface that displays information that can be consulted by the interested parties. The aim is to provide a tool to present the results that is both available to the public and connected to the prior analysis.

The programming language on which the application is based is Python. In addition, it is required the used of some libraries provided by HoloViz, which is a collaboration between the maintainers of several Python packages [45]. The necessary Python libraries are *hvplot*, *Panel*, and *Param*. The first one is a high-level plotting API for the PyData ecosystem [46]. The second one is a library that allows creating custom interactive web apps and dashboards by connecting user-defined widgets to plots, images, tables, or text [47]. The last one is a library for handling all the user-modifiable parameters, arguments, and attributes that control your code [48].

The choice behind this package of libraries is its integration with Anaconda, even though it is partially still under development.

The concept behind the implementation of the dashboard is presented in Figure C.1. The starting point is the Excel files provided by ENAC, whereas the final outcome is the dashboard itself. The in-between process can be divided in two parts: data analysis and dashboard design. The data analysis is the one previously conducted and explained in chapter 4 and chapter 5. The building process includes both plotting adaptations and layout design.

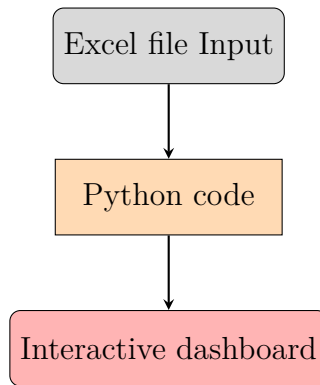


Figure C.1: Flowchart relating to the implementation of the dashboard.

C.0.1 Plotting Adaptation

With plotting adaptation, we indicate the process of implementing interactive plots with *hvplot* library. The starting point is the data in DataFrame format, which is the output of the data analyses. The types of plots are the same displayed in chapter 6, but the used commands are different.

An interactive plot is a plot with special characteristics which help gain a better understanding of the subject. In particular, all the provided charts are fitted with tooltips, as shown in Figure C.2.

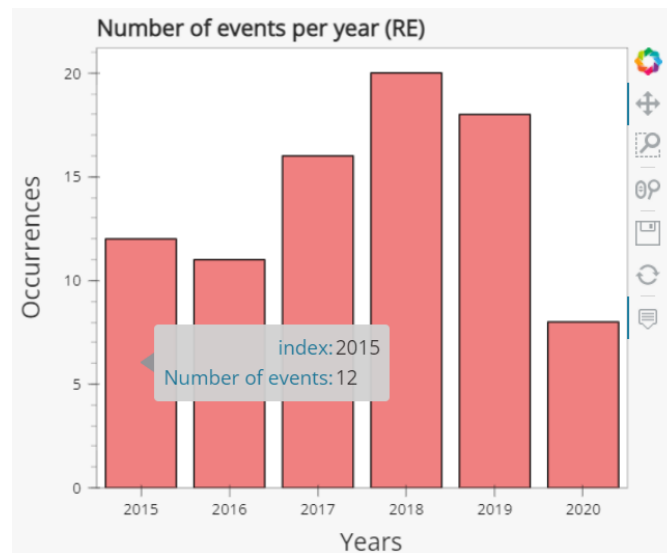


Figure C.2: Example of a tooltip in an interactive plot included in the dashboard.

Bar plots and line plots are the main types included and they are implemented with *hvplot* commands: `hvplot.bar()` and `hvplot.line()*hvplot.scatter()`. Whereas pie charts are not a plotting type included in the mentioned library, so we used *Bokeh*, which is a Python library for creating interactive visualizations for modern web browsers [49]. The required command for pie charts is `figure.wedge()`.

In the dashboard there are additionally tabular representations of the resulting data. In particular, the *panel* command `pn.widgets.DataFrame()` is used to present a `DataFrame` that is also interactive. Every column can be sorted and the rest of the cells adapts to that filtering.

C.0.2 Designing the layout

The layout of a dashboard is subjective. There are many ways of arranging the outputs on a blank page. The way we decided to proceed aims at displaying data as clearly and easily as possible.

The main structure of the dashboard consists of 4 pages and a sidebar. The sidebar allows to choose which page is visualized. The four pages are respectively: *Home*, *Safety Performance Indicators*, *Dangerous Goods*, and *About*. The first page works as the opening front page, the second one contains all the results about SPIs, the third one contains all the results about DGs and the four one gives some information about the project.

Each page is defined in a different way according to the content it contains. Generally, the writing sections are generated with *panel* command `pn.panel(HTML())`, that allows to use `html` programming language. The layout of these sections is made it possible through the *panel* commands `pn.Row()` and `pn.Column()`, which allow to arrange the context in rows or in columns. These two commands are also used in the arrangement of tables and plots.

As far as the data results, such as tables and plots they are displayed in the following order:

- tables,
- plots similar to those included in the ENAC Safety Report,
- additional plots.

In particular, the setup of the plots makes use of tabs, which are a technique that allows switching between multiple objects by clicking on the corresponding tab header. The *panel* command is `pn.Tabs()`.

In the *SPI* page, two drop-down menus are used to select what to visualize.

The first menu allows to select the requested SPI, whereas the second one the analysis to run, in this case the options are 'general analysis' and 'event types analysis', as shown in Figure C.3.

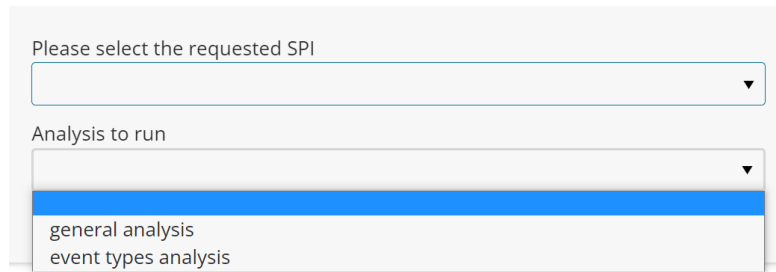


Figure C.3: Drop-down menus in the *SPI* page of the dashboard

In the *DG* page, a drop-down menu is used to select the requested analysis to display, as shown in Figure C.4. The options are 'Event types analysis' and 'Dangerous Goods analysis'.

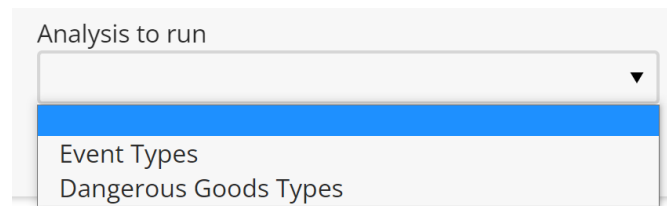


Figure C.4: Drop-down menu in the *Dangerous Goods* page of the dashboard.

The resulting layout is presented in Figures C.5, C.6, C.7, and C.8.

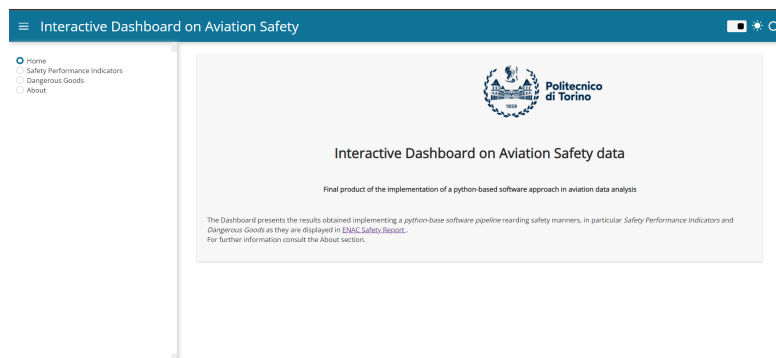


Figure C.5: *Home* page of the dashboard.

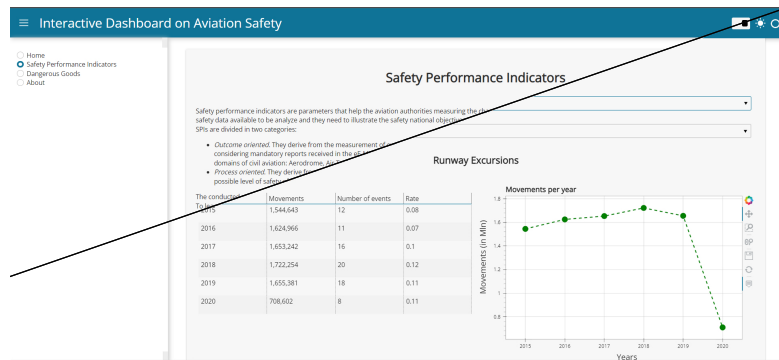


Figure C.6: *Safety Performance Indicators* page of the dashboard.

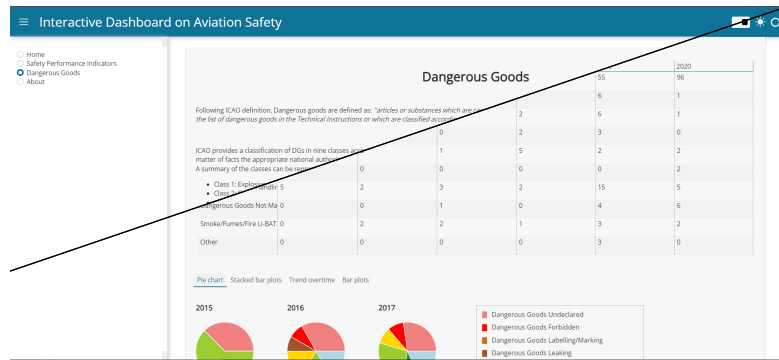


Figure C.7: *Dangerous Goods* page of the dashboard.

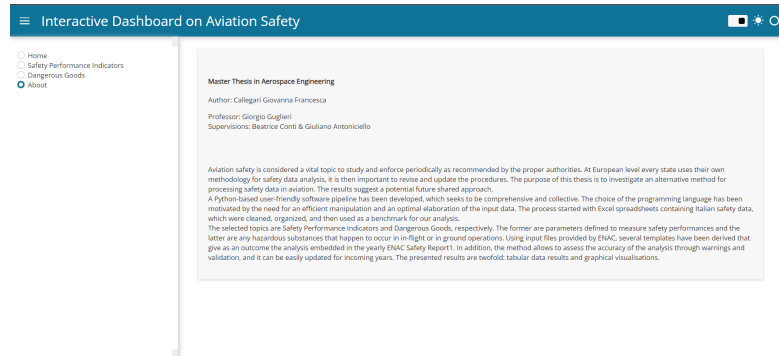


Figure C.8: *About* page of the dashboard.

The code lines regarding the plotting and the layout arrangements are illustrated in the cells below.

```

1 import pandas as pd
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import matplotlib.colors as mcol
5 import matplotlib as mpl
6 import seaborn as sns
7 from itertools import chain #to append lists of list
8 import matplotlib.ticker as mtick #to show percentage ticks in
   matplotlib graphs
9 import os.path #to verify the existence of a file in a specific
   directory
10 import tabula #to read pdf files
11 from matplotlib.gridspec import SubplotSpec #to title every row of in
   figure with subplots
12 from IPython.core.display import display , HTML
13 import ipywidgets as widgets
14 import holoviews as hv
15 import panel as pn
16 import xarray as xr
17 import hvplot.pandas # noqa: API import
18 import hvplot.xarray # noqa: API import
19 import holoviews as hv
20 import param
21 from math import pi
22 from bokeh.plotting import figure , show
23 from bokeh.io import output_notebook , show , save
24 from bokeh.transform import cumsum
25 from bokeh.layouts import row , column
26 pn.extension()
27 #=====
28 #HOME PAGE —> first page of the dashboard
29
30 pn.config.sizing_mode = 'stretch_width' #sizing configuration
31
32 #defining elements of home page
33 logo_poli=pn.pane.PNG( '../logo_polito.png' , width=250) #logo
   politecnico
34 title=pn.panel(HTML( '<p><h1><center>Interactive Dashboard on Aviation
   Safety data</center></h1></p>' ))
35 sub_title=pn.panel(( '<p><h4><center>Final product of the
   implementation of a python-based software approach in aviation
   data analysis</center></h4></p>' ))

```



```

36 expl=pn.panel(HTML( '''<p>The Dashboard presents the results obtained
    implementing a <em>python-base software pipeline</em> rearding
    safety manners, in particular <em>Safety Performance Indicators</
    em> and <em>Dangerous Goods</em> as they are displayed in <a href
    ="https://sites.google.com/enac.gov.it/enacsafetyreport/
    introduction?authuser=0" target="_blank">ENAC Safety Report </a
    >.<br>
37 For further information consult the About section.</p>''' ))
38
39 #buiding home page
40 home_page=pn.Column(pn.Row(pn.Spacer(), logo_poli), pn.Spacer(height
    =10), title, sub_title, expl)
41 #=====
42 #ABOUT PAGE —> last page of the dashobard
43
44 #defining elements of about page
45 authors=pn.panel(HTML( '<p><h4><strong>Master Thesis in Aerospace
    Engineering</strong></h4>Author: Callegari Giovanna Francesca<br
    ></p>'
46
    '<p>Professor: Giorgio Guglieri<br>Supervisions: Beatrice
    Conti & Giuliano Antoniciello</p>' ))
47 abstract=pn.panel(HTML( '''<p>Aviation safety is considered a vital
    topic to study and enforce periodically as recommended by the
    proper authorities. At European level every state uses their own
    methodology for safety data analysis, it is then important to
    revise and update the procedures. The purpose of this thesis is to
    investigate an alternative method for processing safety data in
    aviation. The results suggest a potential future shared approach.<
    br>
48 A Python-based user-friendly software pipeline has been developed,
    which seeks to be comprehensive and collective. The choice of the
    programming language has been motivated by the need for an
    efficient manipulation and an optimal elaboration of the input
    data. The process started with Excel spreadsheets containing
    Italian safety data, which were cleaned, organized, and then used
    as a benchmark for our analysis.<br>
49 The selected topics are Safety Performance Indicators and Dangerous
    Goods, respectively. The former are parameters defined to measure
    safety performances and the latter are any hazardous substances
    that happen to occur in in-flight or in ground operations. Using
    input files provided by ENAC, several templates have been derived
    that give as an outcome the analysis embedded in the yearly ENAC
    Safety Report1. In addition, the method allows to assess the
    accuracy of the analysis through warnings and validation, and it
    can be easily updated for incoming years. The presented results
    are twofold: tabular data results and graphical visualisations.</p
    >''' ))
50 #building home page

```

```

51 about_page=pn.Column(pn.Spacer(height=5), authors, pn.Spacer(height=20)
    , abstract)
52 #=====
53 #=====
54 #DANGEROUS GOODS
55 #EVENT TYPES ANALYSIS
56 ##Here the Event types analysis needs to be run. The code is the same
    reported in the Appendix A, resulting in a DataFrame named '
    newtype_count_wother' ##
57
58 dg_eventtypes_df=pn.widgets.DataFrame(newtype_count_wother, widths={'
    index':60}, width=600)#interactive dataframe to display in the
    dashboard
59 #=====
60 # Pie Charts about Event Types
61 #=====
62 years=['2015', '2016', '2017', '2018', '2019', '2020']
63
64 p1=figure(height=200, width=200, title=years[0], toolbar_location=None
    , tools="hover", tooltips="@types: @value", x_range=(-0.5, 1.0),
    background_fill_color='#00000000')
65 p2=figure(height=200, width=200, title=years[1], toolbar_location=None
    , tools="hover", tooltips="@types: @value", x_range=(-0.5, 1.0),
    background_fill_color='#00000000')
66 p3=figure(height=200, width=200, title=years[2], toolbar_location=None
    , tools="hover", tooltips="@types: @value", x_range=(-0.5, 1.0),
    background_fill_color='#00000000')
67 p4=figure(height=200, width=200, title=years[3], toolbar_location=None
    , tools="hover", tooltips="@types: @value", x_range=(-0.5, 1.0),
    background_fill_color='#00000000')
68 p5=figure(height=200, width=200, title=years[4], toolbar_location=None
    , tools="hover", tooltips="@types: @value", x_range=(-0.5, 1.0),
    background_fill_color='#00000000')
69 p6=figure(height=200, width=200, title=years[5], toolbar_location=None
    , tools="hover", tooltips="@types: @value", x_range=(-0.5, 1.0),
    background_fill_color='#00000000')
70 leg=figure(height=350, width=350, toolbar_location=None, tools="hover
    ", tooltips="@types: @value", x_range=(0, 1.0),
    background_fill_color='#00000000')
71
72 rest=[p1, p2, p3, p4, p5, p6, leg]
73
74 for i in range(0, len(years)):
75
76     x = newtype_count_wother[years[i]]
77
78     pie = pd.Series(x).reset_index(name='value').rename(columns={'
    index': 'types'})
79     pie['angle'] = pie['value']/pie['value'].sum() * 2*pi

```

```

80     pie['color'] = ['lightcoral', 'red', 'chocolate', 'sienna', 'gold', '
khaki', 'yellowgreen', 'forestgreen', 'lightblue', 'cornflowerblue']
81
82     rest[i].wedge(x=0, y=1, radius=0.5, start_angle=cumsum('angle',
include_zero=True), end_angle=cumsum('angle'), line_color="white",
fill_color='color', source=pie)
83     rest[6].wedge(x=0, y=1, radius=0.0001, start_angle=cumsum('angle'
, include_zero=True), end_angle=cumsum('angle'), line_color="white
", fill_color='color', source=pie, legend_field='types')
84
85     rest[i].axis.axis_label = None
86     rest[i].axis.visible = False
87     rest[i].grid.grid_line_color = None
88     rest[i].outline_line_color = None
89     rest[6].axis.axis_label = None
90     rest[6].axis.visible = False
91     rest[6].grid.grid_line_color = None
92     rest[6].outline_line_color = None
93 dg_events_pie= pn.pane.Bokeh(row(column(row(p1, p2, p3), row(p4, p5, p6)),
leg))
94 #
95 # Bar Chart about Event Types General
96 #
97 dg_events_bar=newtype_count_wother.sum().hvplot.bar(height=400,width
=550,shared_axes=False,grid=True,title='Trend overtime',xlabel='
Years',ylabel='Occurrences')
98 #
99 #Stacked Bar chart with percentage for years
100 #
101 color=['lightcoral', 'red', 'chocolate', 'sienna', 'gold', 'khaki', '
yellowgreen', 'forestgreen', 'lightblue', 'cornflowerblue']
102
103 dg_events_stack1=((newtype_count_wother/newtype_count_wother.sum()
*100).T).hvplot.barh(color=color, shared_axes=False, height=400,
width=1000,stacked=True,grid=True,xlabel='Years',ylabel='
Occurrences (%)').opts(legend_position='right')
104 #
105 #Stacked Bar chart with percentage for event type
106 #
107 color=['lightcoral', 'red', 'chocolate', 'gold', 'yellowgreen', '
forestgreen', 'lightskyblue', 'cornflowerblue']
108
109 dg_events_stack2=(newtype_count_wother.T/newtype_count_wother.T.sum()
*100).T.hvplot.barh(stack=True,shared_axes=False, height=400,
width=1000,xlabel='Years',ylabel='Occurrences (%)')
110 #
111 # Bar Chart about Event Types
112 #

```

```

113 color=['lightcoral','red','chocolate','sienna','gold','orangered','
114         yellowgreen','forestgreen','lightblue','cornflowerblue'] #colors
115 dg_events_bar1=(newtype_count_wother.T).hvplot.bar(y=
    newtype_count_wother.index[0], color=color[0], height=300,width
    =400,grid=True,shared_axes=False,title=newtype_count_wother.index
    [0],xlabel='Years',ylabel='Occurrences').opts(fontsize={'title':
    10})
116 dg_events_bar2=(newtype_count_wother.T).hvplot.bar(y=
    newtype_count_wother.index[1], color=color[1], height=300,width
    =400,grid=True,shared_axes=False,title=newtype_count_wother.index
    [1],xlabel='Years',ylabel='Occurrences').opts(fontsize={'title':
    10})
117 dg_events_bar3=(newtype_count_wother.T).hvplot.bar(y=
    newtype_count_wother.index[2], color=color[2], height=300,width
    =400,grid=True,shared_axes=False,title=newtype_count_wother.index
    [2],xlabel='Years',ylabel='Occurrences').opts(fontsize={'title':
    10})
118 dg_events_bar4=(newtype_count_wother.T).hvplot.bar(y=
    newtype_count_wother.index[3], color=color[3], height=300,width
    =400,grid=True,shared_axes=False,title=newtype_count_wother.index
    [3],xlabel='Years',ylabel='Occurrences').opts(fontsize={'title':
    10})
119 dg_events_bar5=(newtype_count_wother.T).hvplot.bar(y=
    newtype_count_wother.index[4], color=color[4], height=300,width
    =400,grid=True,shared_axes=False,title=newtype_count_wother.index
    [4],xlabel='Years',ylabel='Occurrences').opts(fontsize={'title':
    10})
120 dg_events_bar6=(newtype_count_wother.T).hvplot.bar(y=
    newtype_count_wother.index[5], color=color[5], height=300,width
    =400,grid=True,shared_axes=False,title=newtype_count_wother.index
    [5],xlabel='Years',ylabel='Occurrences').opts(fontsize={'title':
    10})
121 dg_events_bar7=(newtype_count_wother.T).hvplot.bar(y=
    newtype_count_wother.index[6], color=color[6], height=300,width
    =400,grid=True,shared_axes=False,title=newtype_count_wother.index
    [6],xlabel='Years',ylabel='Occurrences').opts(fontsize={'title':
    10})
122 dg_events_bar8=(newtype_count_wother.T).hvplot.bar(y=
    newtype_count_wother.index[7], color=color[7], height=300,width
    =400,grid=True,shared_axes=False,title=newtype_count_wother.index
    [7],xlabel='Years',ylabel='Occurrences').opts(fontsize={'title':
    10})
123 dg_events_bar9=(newtype_count_wother.T).hvplot.bar(y=
    newtype_count_wother.index[8], color=color[8], height=300,width
    =400,grid=True,shared_axes=False,title=newtype_count_wother.index
    [8],xlabel='Years',ylabel='Occurrences').opts(fontsize={'title':
    10})

```

```

124 dg_events_bar10=(newtype_count_wother.T).hvplot.bar(y=
    newtype_count_wother.index[9], color=color[9], height=300,width
    =400,grid=True,shared_axes=False,title=newtype_count_wother.index
    [9],xlabel='Years',ylabel='Occurrences').opts(fontsize={'title':
    10})
125
126 dg_events_barall=pn.Column(dg_events_bar1+dg_events_bar2,
    dg_events_bar3+dg_events_bar4,dg_events_bar5+dg_events_bar6,
    dg_events_bar7+dg_events_bar8,dg_events_bar9+dg_events_bar10)
127 #=====
128 #=====
129 #DANGEROUS GOODS TYPES ANALYSIS
130 ##Here the DGs types analysis needs to be run. The code is the same
    reported in the Appendix A, resulting in a DataFrame named '
    dgs_inclasses' ##
131
132 dg_types_df=pn.widgets.DataFrame(dgs_inclasses,widths={'index':300},
    width=600)
133 #=====
134 # Pie Chart
135 #=====
136 years=['2015','2016','2017','2018','2019','2020']
137
138 p1=figure(height=200, width=200,title=years[0], toolbar_location=None
    , tools="hover", tooltips="@types: @value",x_range=(-0.5, 1.0),
    background_fill_color='#00000000')
139 p2=figure(height=200, width=200,title=years[1], toolbar_location=None
    , tools="hover", tooltips="@types: @value",x_range=(-0.5, 1.0),
    background_fill_color='#00000000')
140 p3=figure(height=200, width=200,title=years[2], toolbar_location=None
    , tools="hover", tooltips="@types: @value",x_range=(-0.5, 1.0),
    background_fill_color='#00000000')
141 p4=figure(height=200, width=200,title=years[3], toolbar_location=None
    , tools="hover", tooltips="@types: @value",x_range=(-0.5, 1.0),
    background_fill_color='#00000000')
142 p5=figure(height=200, width=200,title=years[4], toolbar_location=None
    , tools="hover", tooltips="@types: @value",x_range=(-0.5, 1.0),
    background_fill_color='#00000000')
143 p6=figure(height=200, width=200,title=years[5], toolbar_location=None
    , tools="hover", tooltips="@types: @value",x_range=(-0.5, 1.0),
    background_fill_color='#00000000')
144 leg=figure(height=350, width=350, toolbar_location=None, tools="hover
    ", tooltips="@types: @value",x_range=(0, 1.0))
145
146 rest=[p1,p2,p3,p4,p5,p6,leg]
147
148 for i in range(0,len(years)):
149     x = dgs_inclasses[years[i]]
150

```

```

151     pie = pd.Series(x).reset_index(name='value').rename(columns={'
index': 'types'})
152     pie['angle'] = pie['value']/pie['value'].sum() * 2*pi
153     pie['color'] =['lightcoral','red','chocolate','sienna','gold','
khaki','yellowgreen','forestgreen','lightblue','cornflowerblue','
violet'] #colors
154
155     rest[i].wedge(x=0, y=1, radius=0.5,start_angle=cumsum('angle',
include_zero=True), end_angle=cumsum('angle'), line_color="white",
fill_color='color', source=pie)
156
157     rest[6].wedge(x=0, y=1, radius=0.0001,start_angle=cumsum('angle',
include_zero=True), end_angle=cumsum('angle'), line_color="white"
, fill_color='color', source=pie, legend_field='types')
158
159     rest[i].axis.axis_label = None
160     rest[i].axis.visible = False
161     rest[i].grid.grid_line_color = None
162     rest[i].outline_line_color = None
163     rest[6].axis.axis_label = None
164     rest[6].axis.visible = False
165     rest[6].grid.grid_line_color = None
166     rest[6].outline_line_color = None
167 dg_types_pie= pn.pane.Bokeh(row(column(row(p1,p2,p3),row(p4,p5,p6)),
leg))
168
169 # Bar Chart about Event Types General
170
171 dg_types_bar1=dgs_inclasses.sum().hvplot.bar(height=300,width=400,
grid=True, shared_axes=False, title='Trend overtime', xlabel='Years',
ylabel='Occurrences')
172
173 #Stacked Bar chart with percentage for years
174
175 color=['lightcoral','red','chocolate','sienna','gold','khaki','
yellowgreen','forestgreen','lightblue','cornflowerblue','violet']
#colors
176
177 dg_types_stack1=((dgs_inclasses/dgs_inclasses.sum()*100).T).hvplot.
barh(color=color,shared_axes=False, height=400,width=900,stacked=
True,grid=True,xlabel='Years',ylabel='Occurrences')
178
179 #Stacked Bar chart with percentage for event type
180
181 color=['lightcoral','red','chocolate','gold','yellowgreen','
forestgreen','lightblue','cornflowerblue','violet'] #colors
182
183 data_graph=pd.DataFrame(index=years, columns=new_classes)
184 for i in new_classes:

```

```

185     if dgs_inclasses.T[i].sum()!=0:
186         data_graph[i]=dgs_inclasses.T[i]/dgs_inclasses.T[i].sum()*100
187     else:
188         data_graph[i]=0
189
190 dg_types_stack2=(data_graph/data_graph.sum()*100).T.hvplot.barh(
    stacked=True,color=color,shared_axes=False)
191
192 #=====
193 #DANGEROUS GOODS ANALYSIS PAGE —>page 2 of the dashboard
194
195 #defining elements of the top section of the DGs page
196 title_dg=pn.panel(HTML('<p><h1><center>Dangerous Goods</center></h1>
    </p>'))
197 dg_descr=pn.panel(HTML('
    <p>Following ICAO definition , Dangerous goods are defined as: <em>''articles or substances which are capable of posing a risk to health, safety, property or the environment and which are shown in the list of dangerous goods in the Technical Instructions or which are classified according to those Instructions''</em>.</p><br>
    <p>ICAO provides a classification of DGs in nine classes according to the hazard or the most predominant hazard they present. However the categorization is affected by state variations , as a matter of facts the appropriate national authority is required to define it .<br>
    A summary of the classes can be represented as following.<br>
    <ul><li>Class 1: Explosives</li>
    <li>Class 2: Gases</li>
    <li>Class 3: Flammable liquids</li>
    <li>Class 4: Flammable solids; substances liable to spontaneous combustion</li>
    <li>Class 5: Oxidizing substances and organic peroxides</li>
    <li>Class 6: Toxic and infectious substances</li>
    <li>Class 7: Radioactive material</li>
    <li>Class 8: Corrosive substances</li>
    <li>Class 9: Miscellaneous dangerous substances and articles , including environmentally hazardous substances</li></ul>
    The considered order is not a representation of the degree of danger , which is furthered in the specific document. Besides , for each class , ICAO defines definitions , sub-divisions , packing instructions and specific properties. To each considered good , the number of the class/division is assigned , as shown in the proper tables.<br>

```

```

209 Each Dangerous Good is identified by a serial number assigned to the
    article or substance under the United Nations classification
    system. This number is composed of four digit. There are, however,
    two exceptions: firstly, some goods do not have assigned a UN
    number so they are identified with a temporary identification
    number in the 8000 series; secondly, some goods can be forbidden
    on aircraft under any circumstances, so they are labelled as "
    FORBIDDEN".<br>
210 To learn more about the topic, consult <a href="https://www.icao.int/
    safety/DangerousGoods/Pages/Doc9284-Technical-Instructions.aspx"
    target="_blank">ICAO document</a>.</p>''')
211 dg_analysis=pn.panel(HTML(''<p>The Dangerous Goods analysis is
    divided in two section: Event types analysis and Dangerous Goods
    types analysis. The required analysis can be selected below.''))
212 #building the top section of the DGs page
213 dg_page1=pn.Column(title_dg , dg_descr , dg_analysis)
214
215 #Event types analysis page
216 title_et_dg=pn.panel(HTML('<p><h3><center>Event types analysis</
    center></h3></p>'))
217 dg_page2=pn.Column(title_et_dg , pn.Row(dg_eventtypes_df) , pn.Spacer(
    height=20) , pn.Tabs(( 'Pie chart' , pn.Column(pn.Spacer( height=15) ,
    dg_events_pie)) , ( 'Stacked bar plots' , pn.Column(dg_events_stack1 , pn
    .Spacer( height=10) , dg_events_stack2)) , ( 'Trend overtime' ,
    dg_events_bar) , ( 'Bar plots' , dg_events_barall)))
218
219 #Dangerous Goods analysis page
220 title_ty_dg=pn.panel(HTML('<p><h3><center>Dangerous Goods types
    analysis</center></h3></p>'))
221 dg_page3=pn.Column(title_ty_dg , pn.Row(dg_types_df) , pn.Spacer( height
    =20) , pn.Tabs(( 'Pie chart' , pn.Column(pn.Spacer( height=15) ,
    dg_types_pie)) , ( 'Stacked plots' , pn.Column(dg_types_stack1 , pn.
    Spacer( height=10) , dg_types_stack2)) , ( 'Trend overtime' ,
    dg_types_bar1)))
222
223 #objects for the following selectors
224 objects2=[ ' ' , 'Event Types' , 'Dangerous Goods Types' ] #options
225 data_df2=[ ' ' , dg_page2 , dg_page3 ] #corresponding page to display
226
227 data2=' ' #initializing a variable, used in the class below
228 #defining class 'Selectingdata'
229 class Selectingdata(param.Parameterized):
230     Analysis_to_run=param.ObjectSelector(default=' ' , objects=objects2)
    #selector for DGs analyses
231     @param.depends( 'Analysis_to_run' )
232     def data_view(self): #function to display one analysis or the
    other
233         for obj , df2 in zip(objects2 , data_df2):
234             if obj==self.Analysis_to_run:

```



```

235         data2=df2
236     return data2
237 rd=Selectingdata(name='') #class propriety
238 dg_page=pn.Column(dg_page1,rd.param,rd.data_view)
239 #
240 #SAFETY PERFORMANCE INDICATORS—> page 3 of the dashboard
241
242 #defining elements of home page
243 title_spi=pn.panel(HTML('<p><h1><center>Safety Performance Indicators
    </center></h1></p>'))
244 spi_descr=pn.panel(HTML(''<p>Safety performance indicators are
    characteristic level of safety performance of a state. They are
    chosen depending on the safety data available to be analyze and
    they need to illustrate the safety national objectives.<br>
245 SPIs are divided in two categories:<br>
246 <ul><li><em>Outcome oriented</em>. They derive from the measurement
    of events that could be the precursors of "undesired events" (
    accident or serious incident) and normally they are measured
    considering mandatory reports received in the eE-MOR system. These
    indicators have been chosen taking into account types of events
    which are particularly relevant in all the domains of civil
    aviation: Aerodrome, Air Traffic Control, Airworthiness,
    Operations and UAS.</li>
247 <li><em>Process oriented</em>. They derive from the most typical
    processes of the Civil Aviation Authority and plan to measure the
    effectiveness of ENAC activities trying to ensure the highest
    possible level of safety of the aeronautical operations.</li></ul>
248 The conducted analysis considers only SPI—outcome oriented, as they
    are included in the Safety Portal.<br>
249 To learn more about the topic, consult <a href="https://www.enac.gov.
    it/sites/default/files/allegati/2020—Lug/
    Safety_Performance_Indicators_Edizione_2.pdf" target="_blank">
    ENAC document</a>.</p>
250 <p>(*) Additional graphical visualisations.</p>'''),width=600)
251 #bulding the top of the spi page
252 spi_page1=pn.Column(title_spi,spi_descr)
253
254 #defining elements to use in the following analysis
255 #SPIs to be analyzed
256 spi_selection=['Runway Excursions', 'Runway Incursions', 'Loss of
    Control In-Flight', 'TCAS Resolution Advisories',
257 'Activations of TAWS', 'Ramp events', 'Collision while taxiing to/from
    a runway in use', 'Fire or smoke on aircraft', 'Laser beam
    interferences with flight operations', 'Bird Wild Strike rate', '
    Airspace Infringements', 'Separation Minimum Infringement', 'ATM
    technical failures', 'Interferences of APR with manned aircraft
    during take-off or landing']
258 #short name of each SPI

```



```

295     external_data=pd.DataFrame(index=years , data={'movements':
movements, 'flight': flights , 'duration': duration})
296
297     #extracting the data for each year
298     data=pd.DataFrame() #new DataFrame
299     check_length=pd.DataFrame(index=['black cells', 'length'],
columns=years) #new DataFrame for validation
300
301     if name!='empty':
302         for i in range(len(years)):
303             data_excel=pd.read_excel('SPIs/'+file_name, sheet_name
=years[i]) #reading data from excel file sheets
304             data=data.append(data_excel) #appending all the data
together
305
306             #calculating the length of the Excel sheet and the
the number of blank cells per sheet
307             check_length.iloc[0,i]=data_excel.iloc[:,5].isna().
sum()
308             check_length.iloc[1,i]=len(data_excel.iloc[:,5])
309
310             #keeping only the column with the required information
about dates
311             #converting data in date-format and keeping only the
information about the year
312             data_date=pd.to_datetime(data.iloc[:,5].dropna().
reset_index(drop=True)).dt.year
313
314             #For each SPI categories we run a different analysis
315             if name in spi_mvt:
316                 #resulting DataFrame
317                 data_tab=pd.DataFrame(columns=['Movements', 'Number of
events', 'Rate'])
318                 data_tab['Number of events']=data_date.value_counts().
sort_index(ascending=True) #counting the total number of
occurrences
319                 data_tab['Movements']=movements
320                 #rate per 10000 movements, rounding at 2 decimal digits
321                 data_tab['Rate']=round(data_tab['Number of events']/
data_tab['Movements']*10000,2)
322
323                 variation_name='every10000mvt' #specific name for saving
the data
324
325             elif name in spi_flt:
326                 #resulting DataFrame
327                 data_tab=pd.DataFrame(columns=['Flights', 'Number of
events', 'Rate'])

```

```

328         data_tab['Number of events']=data_date.value_counts().
sort_index(ascending=True) #counting the total number of
occurrences
329         data_tab['Flights']=flights
330         #rate per 10000 movements, rounding at 2 decimal digits
331         data_tab['Rate']=round(data_tab['Number of events']/
data_tab['Flights']*10000,2)
332
333         variation_name='every10000flt' #specific name for saving
the data
334
335         elif name in spi_dur:
336             #resulting DataFrame
337             data_tab=pd.DataFrame(columns=['Occupancy duration',
Number of events', 'Rate'])#counting the total number of
occurrences
338             data_tab['Number of events']=data_date.value_counts().
sort_index(ascending=True)
339             data_tab['Occupancy duration']=duration
340             #rate per 1Mln minutes, rounding at 2 decimal digits
341             data_tab['Rate']=round(data_tab['Number of events']/
data_tab['Occupancy duration']*1000000,2)
342
343             variation_name='every1Mlnmins' #specific name for saving
the data
344
345             elif name in spi_occ:
346                 #resulting DataFrame
347                 data_tab=pd.DataFrame(columns=['Number of events'])
348                 data_tab['Number of events']=data_date.value_counts().
sort_index(ascending=True)#counting the total number of
occurrences
349
350                 variation_name='' #no addition information in the output
file name is required
351                 elif name!='empty':
352                     print('Input data do not correspond to study cases')
353
354                 #implementing a check to verify the number of events
extracted corresponds to the total events reported
355                 if name!='empty':
356                     for i in range(len(years)):
357                         if data_tab.iloc[i,1]!=check_length.diff().iloc[1,i]:
358                             print('Missing data!')
359                     data_to_print=data_tab
360                     data_to_print.index=data_tab.index.astype(str)
361                     data_df=pn.widgets.DataFrame(data_tab,widths={'index':40,
'Number of events':90},height=350)
362                 else: #default case (to not run the event types analysis)

```

```

363         data=pd.DataFrame(np.zeros((10, 14)))
364         data.iloc[:,12]=np.nan
365         data_df=''
366
367         #According to the chosen SPI, one of the three graphs below
368         #will be performed.
369         #=====
370         # Movements Trends
371         #=====
372         if name in spi_mvt:
373             line=(external_data/1000000).hvplot.line(x='index',y='
movements', color='green',line_dash= 'dashed',title='Movements per
year', ylabel='Movements (in Mln)', xlabel='Years',grid=True,
height=400,width=500)
374             dots=(external_data/1000000).hvplot.scatter(x='index',y='
movements',height=400,width=500).opts(color='green', size=12,
marker='o')
375             descr_line=line*dots
376         #=====
377         # Flights Trends
378         #=====
379         elif name in spi_flt:
380             line=(external_data/1000000).hvplot.line(x='index',y='
flights', color='green',line_dash= 'dashed',title='Flights per
year', ylabel='Flights (in Mln)', xlabel='Years',grid=
True,height=400,width=500)
381             dots=(external_data/1000000).hvplot.scatter(x='index',y=
'flights').opts(color='green', size=12, marker='o')
382             descr_line=line*dots
383         #=====
384         # Occupancy Duration
385         #=====
386         elif name in spi_dur:
387             line=(external_data/1000000).hvplot.line(x='index',y='
duration', color='green',line_dash= 'dashed',title='Occupancy
Duration (min)', ylabel='Occupancy Duration (in Mln of min)',
xlabel='Years',grid=True,height=400,width=500)
388             dots=(external_data/1000000).hvplot.scatter(x='index',y='
duration').opts(color='green', size=12, marker='o')
389             descr_line=line*dots
390
391         elif name=='empty': #default case
392             descr_line=''
393         #=====
394         #Occurrences per year and rate per year
395         #=====
396         if name in spi_mvt+spi_flt+spi_dur: #plotting number of
occurrences and rate
397

```

```

398         occ_bar=data_tab.hvplot.bar(x='index',y='Number of events
',height=400,width=500,color='lightcoral',title='Number of events
per year ('+name+')', xlabel='Years',ylabel='Occurrences')
399         rate_bar=data_tab.hvplot.bar(x='index',y='Rate',height
=400,width=500,color='red',title='Rate per year ('+name+')',
xlabel='Years',ylabel='Rate')
400
401         line2=data_tab.hvplot.line(x='index',y='Rate', color='red
',line_dash='dashed',title='Rate per year ('+name+')', ylabel='
Rate', xlabel='Years',grid=True,height=400)
402         dots2=data_tab.hvplot.scatter(x='index',y='Rate').opts(
color='red', size=12, marker='o')
403         rate_line=line2*dots2
404
405         rate_tab=pn.Tabs(('Bar chart',rate_bar),('Line chart (*)'
,rate_line))
406
407         elif name in spi_occ: #plotting only the total number of
occurrences
408             occ_bar=data_tab.hvplot.bar(x='index',y='Number of events
',height=400,width=500,color='lightcoral',title='Number of events
('+name+')', xlabel='Years',ylabel='Occurrences')
409             rate_bar=''
410             rate_line=''
411             rate_tab=''
412         else: #default case
413             occ_bar=''
414             rate_bar=''
415             rate_line=''
416             rate_tab=''
417
418         spi_title_selec=HTML('<p><h2><center>{spi}</center></h2></p>'
.format(spi=self.Please_select_the_requested_SPI))
419         spi_page2=pn.Column(spi_title_selec,pn.Row(pn.Column(pn.
Spacer(height=18),data_df),descr_line),pn.Row(pn.Column(pn.Spacer(
height=22),occ_bar),rate_tab))
420
421         if data.iloc[:,12].notnull().values.any(): #for now it means
only RE
422             data_events=pd.DataFrame({'date': pd.to_datetime(data.
iloc[:,5]), 'event types': data.iloc[:,12]}).dropna()
423             data_events['date']=data_events['date'].dt.year
424
425             event_types=['Runway Incursion by an Aircraft','Runway
Incursion by a Person','Runway Incursion by a Vehicle/Equipment']
426
427             events=pd.DataFrame(columns=['Aircraft','rate Aircraft
x10000 mvts','Person','rate Person x10000 mvts','Vehicle/Equipment
','rate Vehicle/Equipment x10000 mvts'])

```

```

428         for i in range(len(event_types)):
429             events.iloc[:, i+i]=data_events[data_events['event
430 types']==event_types[i]]['date'].value_counts().sort_index(
ascending=True)
431             events.iloc[:, (i+i)+1]=round(events.iloc[:, i]/
movements*10000,3)
432
433             events_to_print=events
434             events_to_print.index=data_tab.index.astype(str)
435             events_df=pn.widgets.DataFrame(events_to_print, widths={'
Aircraft':50, 'rate Aircraft x10000 mvts':150, 'Person':50, 'rate
Person x10000 mvts':150, 'Vehicle/Equipment':130, 'rate Vehicle/
Equipment x10000 mvts':220}, width=750)
436 #=====
437 #Bar charts
438 #=====
439         color=['lightcoral', 'red', 'chocolate', 'sienna', 'lightblue
', 'cornflowerblue'] #colors
440
441         et_bar1=events.hvplot.bar(x='index', y=events.columns[0],
height=250, width=350, color=color[0], title=events.columns[0]+' ('+
name+)', xlabel='Years', ylabel='Occurrences').opts(toolbar='right
', fontsize={'title': 10})
442         et_bar2=events.hvplot.bar(x='index', y=events.columns[1],
height=250, width=350, color=color[1], title=events.columns[1]+' ('+
name+)', xlabel='Years', ylabel='Rate').opts(toolbar='right',
fontsize={'title': 10})
443         et_bar3=events.hvplot.bar(x='index', y=events.columns[2],
height=250, width=350, color=color[2], title=events.columns[2]+' ('+
name+)', xlabel='Years', ylabel='Occurrences').opts(toolbar='right
', fontsize={'title': 10})
444         et_bar4=events.hvplot.bar(x='index', y=events.columns[3],
height=250, width=350, color=color[3], title=events.columns[3]+' ('+
name+)', xlabel='Years', ylabel='Rate').opts(toolbar='right',
fontsize={'title': 10})
445         et_bar5=events.hvplot.bar(x='index', y=events.columns[4],
height=250, width=350, color=color[4], title=events.columns[4]+' ('+
name+)', xlabel='Years', ylabel='Occurrences').opts(toolbar='right
', fontsize={'title': 10})
446         et_bar6=events.hvplot.bar(x='index', y=events.columns[5],
height=250, width=350, color=color[5], title=events.columns[5]+' ('+
name+)', xlabel='Years', ylabel='Rate').opts(toolbar='right',
fontsize={'title': 10})
447
448         air=et_bar1+et_bar2
449         pers=et_bar3+et_bar4
450         equip=et_bar5+et_bar6

```

```

451         et_bars=pn.Column(pn.panel(HTML( '<p><h3><center>Runway
Incursion by an Aircraft</center></h3><p>' )), air, pn.panel(HTML( '<p>
<h3><center>Runway Incursion by a Person</center></h3><p>' )), pers
, pn.panel(HTML( '<p><h3><center>Runway Incursion by a Vehicle/
Equipment</center></h3><p>' )), equip)
452     #=====
453     #Event types graphs all together
454     #Bar charts stacked ———> not with percentage because these events
types are just a few of the total
455     #=====
456         only_events=['Aircraft', 'Person', 'Vehicle/Equipment']
457         color=['red', 'forestgreen', 'gold', 'lightblue', '
cornflowerblue']
458
459         et_barstack=events.hvplot.barh(x='index', y=only_events,
stacked=True, shared_axes=False, legend='bottom_right', width=700,
color=color, title='Event Types total occurrences per year ('+name+
')', xlabel='Years', ylabel='Occurrences')
460     #=====
461     #Line plot of rate trends
462     #=====
463         only_rates=['rate Aircraft x10000 mvts', 'rate Person
x10000 mvts', 'rate Vehicle/Equipment x10000 mvts']
464         color=['red', 'sienna', 'cornflowerblue']
465
466         line3=events.hvplot.line(x='index', y=only_rates,
shared_axes=False, color=color, line_dash='dashed', title='Event
types rates per year ('+name+)', ylabel='Rate', xlabel='Years',
grid=True, width=800, height=400)
467         dots3=events.hvplot.scatter(x='index', y=only_rates,
shared_axes=False, color=color, marker='o', size=20)
468         et_rates_line=line3*dots3
469
470     #=====
471     #Bar charts and line plots
472     #=====
473         color=['lightcoral', 'red', 'chocolate', 'sienna', 'lightblue
', 'cornflowerblue'] #colors
474
475         line4=events.hvplot.line(x='index', y=only_rates[0], color
=color[1], line_dash='dashed', title=only_rates[0], ylabel='Rate',
xlabel='Years', grid=True, height=250, width=350).opts(fontsize={'
title': 10})
476         dots4=events.hvplot.scatter(x='index', y=only_rates[0]).
opts(color=color[1], size=12, marker='o')
477         et_line2=(line4*dots4).opts(toolbar='below')
478

```



```

479         line5=events.hvplot.line(x='index',y=only_rates[1], color
=color[3],line_dash='dashed',title=only_rates[1],ylabel='Rate',
xlabel='Years',grid=True,height=250,width=350).opts(fontsize={'
title': 10})
480         dots5=events.hvplot.scatter(x='index',y=only_rates[1]).
opts(color=color[5], size=12, marker='o')
481         et_line4=(line5*dots5).opts(toolbar='below')
482
483         line6=events.hvplot.line(x='index',y=only_rates[2], color
=color[5],line_dash='dashed',title=only_rates[2],ylabel='Rate',
xlabel='Years',grid=True,height=250,width=350).opts(fontsize={'
title': 10})
484         dots6=events.hvplot.scatter(x='index',y=only_rates[2]).
opts(color=color[5], size=12, marker='o')
485         et_line6=(line6*dots6).opts(toolbar='below')
486
487         airl=et_bar1+et_line2
488         pers1=et_bar3+et_line4
489         equip1=et_bar5+et_line6
490         et_barandline=pn.Column(pn.panel(HTML('<p><h3><center>
Runway Incursion by an Aircraft</center></h3><p>')),airl,pn.panel(
HTML('<p><h3><center>Runway Incursion by a Person</center></h3><p>
')),pers1,pn.panel(HTML('<p><h3><center>Runway Incursion by a
Vehicle/Equipment</center></h3><p>')),equip1)
491
492         #defining elements of spi page to lay the plots out
493         et_tab = pn.Tabs(('Only Bar charts',et_bars),('Bar and
line charts (*)',et_barandline),('More (*)',pn.Column(et_barstack,
et_rates_line)))
494         title_et=pn.panel(HTML('<p><h3>Event types</h3></p>'))
495         intro_et=pn.panel(HTML('''<p>The Event types analysis
aims at counting how many time a specific event has occurred. Two
values are then calculated: the total number of occurrences and
the rate. The rate follows the formula indicated above.<br>
In this case, we take into account only
three categories of event types:
<ul><li>Runway Incursions by an Aircraft
</li>
<li>Runway Incursions by an Person</li>
<li>Runway Incursions by an Vehicle/
Equipment</li></ul>'''))
500         #building the section of the spi page on plots of the
general analysis
501         spi_etpage=pn.Column(title_et,intro_et,events_df,pn.
Spacer(height=15),et_tab)
502
503         elif name!='empty': #defining the section of the spi page on
plots of the event types analysis

```

```

504         events_df, et_bars, et_barandline, et_barstack, et_rates_line
= 'None', 'None', 'None', 'None', 'None'
505         no_events_data='The data-set does not contain Event types
.'
506         spi_etpage=pn.Column(no_events_data)
507     else:
508         spi_etpage='' #empty string to display when 'default'
509
510         if self.Analysis_to_run=='general analysis': #selecting which
page to display: general analysis or event types analysis
511             page=spi_page2
512         elif self.Analysis_to_run=='event types analysis':
513             page=spi_etpage
514         else:
515             page='' #empty string to display when 'default'
516
517         return page
518
519 rd5=spi_page_building(name='') #class propriety
520
521 #building the page, depending on the selections
522 spi_page=pn.Column(spi_page1, rd5.param, rd5.spi_analysis)

```

```

1  #DASHBOARD
2  pn.extension(sizing_mode="stretch_width") #sizing property
3
4  pages = {
5      "Home": home_page,
6      "Safety Performance Indicators": spi_page,
7      'Dangerous Goods': dg_page,
8      'About': about_page, } #dict on the pages names and variables
9
10 def show(page): #function to show the pages
11     return pages[page]
12
13 starting_page = pn.state.session_args.get("page", [b"Home"])[0].
decode() #default starting page
14 #defining page selector
15 page = pn.widgets.RadioBoxGroup(
16     value=starting_page,
17     options=list(pages.keys()),
18     name="Page",
19     sizing_mode="fixed",
20     button_type="success")
21
22 ishow = pn.bind(show, page=page) #defining parameters for the
dashboard template

```

```
23 DEFAULT_PARAMS = {"accent_base_color": '#0E7397', "header_background"  
24 : '#0E7397'}  
25 #building the dashboard through the template  
26 dash=pn.template.FastListTemplate(  
27     title="Interactive Dashboard on Aviation Safety",  
28     sidebar=[page], #sidebar  
29     main=[ishow], #main page  
30     **DEFAULT_PARAMS).show()
```


Bibliography

- [1] ICAO. *Safety Managament Manual - Doc. 9859*. 4th. 2018 (cit. on pp. 1–5, 25, 28).
- [2] *ICAO Website*. URL: <https://www.icao.int/Pages/default.aspx> (cit. on p. 5).
- [3] *EASA Website*. URL: <https://www.easa.europa.eu/the-agency/the-agency> (cit. on p. 6).
- [4] *EASA Safety Programms*. URL: <https://www.easa.europa.eu/annual-programmes-reports> (cit. on p. 6).
- [5] EASA. *European Plan for Aviation Safety (EPAS 2020-2024)*. 2009 (cit. on p. 6).
- [6] *ENAC Website*. URL: <https://www.enac.gov.it/> (cit. on p. 7).
- [7] ENAC. *State Plan for Aviation Safety 2021-2025*. 2021 (cit. on p. 8).
- [8] Lucas Fernandes, Giorgio Guglieri, and Beatrice Conti. «A comprehensive analysis of Aviation Safety Reports in Europe». Politecnico di Torino, 2021 (cit. on p. 8).
- [9] *ENAC Safety Portal - Italy*. URL: <https://sites.google.com/enac.gov.it/enacsafetyreport/introduction?authuser=0> (cit. on pp. 9, 22, 30).
- [10] *Safety Performance Indicators*. Edizione 2 (cit. on pp. 10, 25, 26).
- [11] *ICAO Website - SAFETY*. URL: <https://www.icao.int/safety/airnavigation/ops/cabinsafety/pages/dangerous-goods.aspx> (cit. on p. 15).
- [12] ICAO. *Technical Instructions for the Safe Transport of Dangerous Goods by Air*. 2019-2020 (cit. on pp. 15–17, 47).
- [13] *Austrian Safety Reviews*. URL: <https://www.austrocontrol.at/luftfahrtbehoerde/safety/sicherheitsbericht> (cit. on p. 19).
- [14] *Belgium Safety Review 2018*. URL: https://mobilit.belgium.be/sites/default/files/DGLV/belgian_plan_for_aviation_safety_update_2018_en.pdf (cit. on p. 20).

- [15] *Belgium Safety Review 2010*. URL: https://mobilit.belgium.be/sites/default/files/DGLV/belgian_plan_for_aviation_safety_bpas_2020-2024_update_2020_en.pdf (cit. on p. 20).
- [16] *German Safety Reviews*. URL: https://www.lba.de/DE/Presse/Publikationen/_Funktion/Jahresberichte_node.html (cit. on p. 20).
- [17] *Irish Safety Reviews*. URL: <https://www.iaa.ie/publications?taxonomy=categories&propertyName=category&taxon=%2fpublication-categories%2fcorporate-publications%2fperformance> (cit. on p. 20).
- [18] *Irish Safety Review 2017*. URL: [https://www.iaa.ie/docs/default-source/publications/corporate-publications/performance/13161-iaa-safety-report-2017-v07-\(002\).pdf](https://www.iaa.ie/docs/default-source/publications/corporate-publications/performance/13161-iaa-safety-report-2017-v07-(002).pdf) (cit. on p. 20).
- [19] *Irish Safety Reviews 2018*. URL: https://www.iaa.ie/docs/default-source/publications/corporate-publications/performance/annual-safety-performance-review-2018.pdf?sfvrsn=c7b00f3_0 (cit. on p. 20).
- [20] *Italian Safety Review 2020*. URL: <https://drive.google.com/file/d/1f3csETwrM6QHRW3rGo5Zx7ulqrtNo-qI/view> (cit. on p. 22).
- [21] *Italian Safety Review 2019*. URL: https://drive.google.com/file/d/1f1c_iywltJSHcIhmv0z8IVMheDpbt_F/view (cit. on p. 22).
- [22] *Italian Safety Review 2016*. URL: <https://drive.google.com/file/d/1ezuhihW1zhHfXAMBaL8imTADoQIFwkfq/view> (cit. on p. 22).
- [23] *Dutch Safety Portal*. URL: <https://dashboards.ilt.rijkscloud.nl/luchtvaartvoorvallen/> (cit. on p. 22).
- [24] *Norwegian Safety REport 2019*. URL: <https://luftfartstilsynet.no/globalassets/dokumenter/flysikkerhet/norske-flysikkerhetsresultater/norske-flysikkerhetsresultater-2019.pdf> (cit. on p. 23).
- [25] *Polish Safety Review 2017*. URL: https://ulc.gov.pl/_download/bezpieczenstow_lotow/analizy/Sprawozdanie_o_stanie_bezpieczenstwa_lotnictwa_cywilnego_za_rok_2017_v1.pdf (cit. on p. 23).
- [26] *Polish Safety Review 2018*. URL: https://ulc.gov.pl/_download/bezpieczenstow_lotow/kultura-promocja-bezpieczenstwa/Sprawozdanie_o_stanie_bezpieczenstwa_lotnictwa_cywilnego_za_rok_2018_v1.pdf (cit. on p. 23).
- [27] *Spanish Safety Review 2015*. URL: https://www.seguridadaerea.gob.es/sites/default/files/memoria_2016.pdf (cit. on p. 23).
- [28] *Spanish Safety Review 2016*. URL: https://www.seguridadaerea.gob.es/sites/default/files/memoria_2015_sns_y_ceanita.pdf (cit. on p. 23).

- [29] *Spanish Safety Review 2017*. URL: https://www.seguridadaerea.gob.es/sites/default/files/memoria_2017.pdf (cit. on p. 23).
- [30] *ENAC occurrences reporting website*. URL: <https://reporting.enac.gov.it/eemor-4103/index.php> (cit. on p. 29).
- [31] ENAC. *VADMECUM - guida pratica all'utilizzo del sistema eE-MOR, R376/2014, R2015/1018*. Settembre 2019 (cit. on p. 29).
- [32] Ing. Rosario Concilio. «Il nuovo che avanza - Il sistema eE-MOR ed il Regolamento 376/2014». In: Rome, July 2015 (cit. on p. 29).
- [33] *Joint Research Center Website*. URL: <https://ec.europa.eu/jrc/en/about/jrc-in-brief> (cit. on p. 29).
- [34] *ADREP Taxonomy*. URL: <https://www.icao.int/safety/airnavigation/aig/pages/adrep-taxonomies.aspx> (cit. on p. 29).
- [35] ENAC. *LA SEGNALAZIONE OBBLIGATORIA DEGLI EVENTI AERONAUTICI (MANDATORY OCCURRENCE REPORTING) - GEN - 01E*. 2021 (cit. on p. 29).
- [36] ENAC. *REGOLAMENTO - TRASPORTO AEREO DELLE MERCI PERICOLOSE*. 2nd. 2019 (cit. on p. 30).
- [37] ICAO. *Reference Manual on the ICAO Statistics Programme - Doc. 9060/5*. 5th. 2013 (cit. on p. 30).
- [38] *Python Website*. URL: <https://www.python.org/doc/essays/blurb/> (cit. on p. 31).
- [39] URL: https://en.wikipedia.org/wiki/Microsoft_Excel (cit. on p. 32).
- [40] *Numpy Website*. URL: <https://numpy.org/> (cit. on p. 32).
- [41] *Pandas Website*. URL: <https://pandas.pydata.org/> (cit. on p. 32).
- [42] URL: <https://matplotlib.org/> (cit. on p. 32).
- [43] URL: <https://seaborn.pydata.org/> (cit. on p. 32).
- [44] URL: <https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html> (cit. on p. 35).
- [45] *HoloViz Website*. URL: <https://holoviz.org/> (cit. on p. 123).
- [46] *hvplot Website*. URL: <https://hvplot.holoviz.org/> (cit. on p. 123).
- [47] *Panel Website*. URL: <https://panel.holoviz.org/> (cit. on p. 123).
- [48] *Param Website*. URL: <https://param.holoviz.org/> (cit. on p. 123).
- [49] *Bokeh Website*. URL: <https://docs.bokeh.org/en/2.4.1/index.html#> (cit. on p. 125).