POLITECNICO DI TORINO

ENGINEERING FACULTY

MASTER'S DEGREE COURSE

IN

MECHATRONIC ENGINEERING

MASTER'S DEGREE THESIS

Enhancement of Data Analysis and Visualization

Supervisor:

Professor Enrico Masala

Student:

Bisola Adeniyi

----Academic Year 2018/2019----

APRECIATION

Special appreciation to my thesis supervisor, Professor Enrico Masala for his patience towards me, lenient with my terms and always ready to impart more knowledge in me and to guide me through any problem I was having with the thesis. Being able to manage a distance thesis student is not the easiest of things to do, but Professor Enrico Masala saw me through the thesis phases until completion and this thesis would not be possible without his help.

Appreciation to all the professors that helped in passing this knowledge and in instilling and guiding me through my years here as a student.

TABLE OF CONTENTS

CHAPTER I – Introduction

This thesis is to research and investigate into making some modifications to the GUI of W.E.K.A to allow better customization for analysing data via csv files. Normally, these files come in extremely large data sets that are difficult or too many to analyse. The aim of this thesis is to enable a better analysis of the data sets in any particular uploaded CVS file and to make them easier to be understood, analysed, visualized, queried and manipulated by users of Waikato Environment for Knowledge Analysis Software (W.E.K.A). In the course of this thesis, I made changes to enhance the differences between the datasets through colours, colour bars, plots, data points, to distinguish between certain features using different shapes and tick boxes and to improved visualization by adding a feature to minimize and maximize data sets within a confined plot. The software package which I worked on (W.E.K.A), is an open source software package and I was able to adapt and extend its functionalities to better suit the need of dynamic interactive representations of the information included in databases with objective quality metrics, in particular, extending its data visualization capabilities. WEKA is released as open source software, to foster diffusion and usage in the international research community.

A few bits of the knowledge of W.E.K.A and its origination is helpful to understand how the software came about. According to University of Waikato, Weka is a collection of machine learning algorithms for data mining tasks. It contains tools for data preparation, classification, regression, clustering, association rules mining, and visualization. Found only on the islands of New Zealand, the Weka is a flightless bird with an inquisitive nature. The name is pronounced like this, and the bird sounds like this. Weka is open source software issued under the GNU General Public License.

An exciting and potentially far-reaching development in computer science is the invention and application of methods of machine learning (ML). These enable a computer program to automatically analyse a large body of data and decide what information is most relevant. This crystallised information can then be used to automatically make predictions or to help people make decisions faster and more accurately.

According the W.E.K.A, their objectives are to

- make ML techniques generally available;
- apply them to practical problems that matter to New Zealand industry;
- develop new machine learning algorithms and give them to the world;

- contribute to a theoretical framework for the field.

W.E.K.A software team has incorporated several standard ML techniques into a software "workbench" called W.E.K.A, for Waikato Environment for Knowledge Analysis. With it, a specialist in a field is able to use ML to derive useful knowledge from databases that are far too large to be analysed by hand. Weka's users are ML researchers and industrial scientists, but it is also widely used for teaching. Recently, our team has also worked on Massive Online Analysis(**MOA**), an environment for mining data streams

CHAPTER II – Context and Motivation

In the section I would be describing the data that we want to visualize; this data can come different file format that are presently supported by the W.E.K.A version 3.9.2 (which was the latest version as of the inception of this project). The file format type. arff, arff.gz, names, .data, .csv, .json, .json.gz, .libsvm, .m, .dat, .bsi, xrff, .xrff.gz : For the purpose of this thesis, I solely did all my testing with files that have the data format arff.gz as this was what I had available. Basically, the data set files contain instances and certain combination of these instances is what we actually want to visualize. An instance is a combination of attributes and it is this attribute that we would work on. An attribute normally contains video quality values with the like of (PSNR SSIM VIF etc...), and input parameters or "features" with the likes of (seq, rate, etc...). These video values are normally numerical as they contain integers and double representing video data, likewise the input parameters and features are normally nominal as they just contain values.

What we aim to achieve is a thorough investigation or a close to perfect understanding of how these "features" affect the video quality values. To do these there are there are panels and inbuilt features already provided by the W.E.K.A software application to hep us out. These can be done through the Panels provided by the application. The panels provided by default by W.E.K.A are:

Preprocess Panel – initially process the data we intend using, filter out unwanted attributes and see a thumbnail view of out the attributes look like and also see their properties.

Classify Panel – this panel lets the user pick an attribute from the data sets and see all the relevant property associated with it in a detailed view, there by classifying it.

Clustered Panel – this panel lets us choose an inbuilt clusterer present in the application for clustered the selected data you want, but it must be a nominal value.

Associate Panel – this panel creates an associated output for each of the attribute values that is selected.

Visualize Panel – this panel is responsible for seeing each of the data points in a plotted graph via a GUI. It also allows for manipulation and in-depth investigation of all data points. The base of the thesis and majority of the newly added features was be on this panel

Select Attributes Panel – this panel displays the properties of a selected attribute based on certain choices.

Auto-WEKA Panel – this panel is used automated different set of data set based on the attribute selected.

It is good to know that more panels that aid with understand data can also be added to the W.E.K.A application via plugins. This can be done by clicking on tools from the Weka GUI Chooser Menu. Then packet manager and select and install the plugins you prefer. As said earlier most of the work that would be discussed in this thesis would be on the Visualize Panel as this is the panel that is mostly utilised by researchers and would go a great effort improving the usability of the W.E.K.A application.

The thesis covers the previous and existing features that existed prior to the thesis inception. The progress that was made during the thesis development, the life cycle of the development, the result attained from each of the newly added featured to W.E.K.A application explained in detail and the comparisons, advantages and disadvantages. Majorly, all the features added to the thesis increases the robustness of the software application. Making it richer, easier to access, easier to use, easier to understand, easier to manipulate data, easier to distinguish between data and easier to filter out data. All features added to this application comes in very handy because they are used for research purposes for scientists, governments and schools, they are used for learning purposes as well.

In addition, in the sight of motivation, the main aim for this thesis was to make working in a data visualization environment easier. It enables a better and quality analysis of multimedia research data in particularly large data sets of several quality measures. Large data sets comprise of amount of up to 100s of thousands of data that is to be manipulated. As a researcher a lot of newly added features would come in handy and it would make life seamlessly easier. A lot of inferences and conclusions would be arrived at faster due to the enhanced ability of the user interface, the richer graph plots and the newly added dynamic features. There is always a room for improvement, so the features added to the application can also be upgraded as it is deemed fit. W.E.K.A been an open source software has made it easy for online community of software engineers and researchers to add to its functionalities and improve its service continuously.

Finally, efforts to determine whether the features improved upon and added during this thesis are solidly tested and won't constitute a nuisance to the end users have been taken. So,

the software would be durable, reliable and reusable for very large data sets with very good error handling measures taken into consideration. In the view of these, I am certain that the improvement to the W.E.K.A application would help a lot of user spend lesser time analysing data and spend more time been productive.

CHAPTER III – Contribution

This chapter would focus on the contributions I have made to the W.E.K.A development environment. It is a work through from the inception of the practical work for my thesis to the end. All the features I added, the disadvantages and limitations the application had and the advantages and versatility my code changes gave to the application. This dissertation contributes to the area of pure experimental computer science. Specifically, it introduces problem solving and techniques to the fields of operating big data systems, data visualization systems, and experimental systems research in general. The primary objective of this dissertation is to test the hypothesis that:

1. enables a better and quality analysis of multimedia research data in particularly large data sets,
2. enables manipulation of data in particularly large data sets of objective video quality measures

It should be noted that while it is possible to formally prove the correctness or falsehood of this hypothesis, it is not possible for this thesis to be the final upgrade for the software application because it would always be in progress. Instead, this dissertation is limited to providing, hopefully strong, evidence and feature additions to the W.E.K.A software application. It does so by breaking down all the features enhancements into smaller features that would be discussed later. The thesis would be written by the application platform provided by the university of Waikato. The university is located in New Zealand, is committed to delivering a world-class education and research portfolio, providing a full and dynamic university experience, distinctive in character, and pursuing strong international links to advance knowledge.

Further down this chapter we would also pick the features that have been developed one after the order, we would identify the disadvantage before the feature was added, the obstruction is caused and the hindrances it had. Also, after the new features have been added we would discuss numerous advantages as well to the application and how is feels better compared to the way it was before. This contributions to the application would make the software easier to use, faster to use and data would be analysed faster in an even better way.

Chapters 4 present the results of all the added feature. What has been added and what has been upgraded and how it all blends in together to form a new interesting feature for the application.

The three major contributions from this study are that:

1. improvement of user interface for the software application which was formally a bottleneck for research with large data sets
2. dynamic feature content and graph plots were all improved upon in various areas and
3. the three major feature addition resulted in significant performance improvements.

The first result increases aesthetics and clarity: while it has long been known that the presentation of data points is a primary bottleneck for large data sets with various attributes, it was generally accepted that the existent feature was enough to do the job that is needed to be done, this brought forth problems and limitations in data research bringing forth a lot of errors and thus a big problem when using the wrongly analysed data. It is useless. This study improves on the already existent user interface and makes it better and more usable to the end users and researchers. The second result implies that more advanced features were added under study, features like filtering data sets, removing selected data sets and making sure that the filtered data sets are filtered in all places that use the datasets including plugins. This comes in extremely handy for separating datasets for certain purposes and scattering them sparsely for easier investigation. Also, to enrich the graph plot, certain features were added to the panel below the plot to make it easier to interact with the data. For example, tick boxes for filtering the data, bars represent the colours and shapes of data sets and also when these bars are clicked on a window displays where we can easily modify their present value. The third result is rounds up all the previous, because a combination of all the features written above improves the performance and usability of the software application. This chapter would show the codes changes and additions written in a way to evict and counter act existent bugs so as to make it even easier for users to work with and implement the features that would be discussed in chapter 4. With the contribution that has been made to this project, more detailed oriented data visualization and investigation can be perform in a lesser amount of time. but in analysing the reasons for its failure, a better understanding of the technique is obtained. With this improved understanding, it is possible to enumerate the scenarios in which the technique might be employed beneficially.

Chapter 5 speaks about the conclusion and the summary of the dissertation work, an outline of future research directions, and some concluding remarks. Below are the contributions made for each of the section investigated:

User Interface

As discussed above this is one of the important features added to this thesis, the addition of all the features under user interface has proved to be:

Improved label readability

For improved label readability in the class panel tab under in the visualization panel this was added to the ClassPanel.java class in the PaintNornimal() method for create the U.I that is discussed in chapter 4.

```
jj.setLocation(x, y+10);
```

This was used to move the location of the attribute numbers slightly above by 10 pixels to the y-axis , while x axis position is maintained.

```
gx.setColor(m_colorList.get(i % m_colorList.size()));
gx.fillRect(x, y + 25, 20, 10);
```

The function above gets the present colour of the attribute, sets its for the graphic, draws a rectangle with the following dimension and fill it with the colour gotten from the attribute.

Changeable shape for plotted points

Instead of just displaying data points with a cross sign when analysing data, we can now display these points in different shapes as well, these shapes are outlined here, and the code changes made as well.

```
public static final int MAX_SHAPES = 7;
public static final int ERROR_SHAPE = 1000;
public static final int MISSING_SHAPE = 2000;
public static final int X_SHAPE = 0;
public static final int PLUS_SHAPE = 1;
public static final int DIAMOND_SHAPE = 2;
public static final int TRIANGLEUP_SHAPE = 3;
public static final int TRIANGLEDOWN_SHAPE = 4;
public static final int DEFAULT_SHAPE_SIZE = 2;
```

These shapes are defined as static final integers assigned a corresponding number which is also known as its index so that they can be used as an enum. When this index is called automatically the logic should draw the respective shape. There are other static variables like Error, Missing and Default and Max and their names connote their meaning.

```
public void setCSindex(int c) {

	m_shIndex = c;
	m_setShape = true;
	m_setAttrShape = false;

	for (int i = 0; i < m_plots.size(); i++) {

		PlotData2D temp_plot = (m_plots.get(i));

		if(temp_plot.m_plotInstances.attribute(i).isNominal())
			m_plots.get(i).setShindex(m_shIndex);

	}

	determineBounds();
	m_axisChanged = true;
	this.repaint();

   }
```

This is the method that calls in the shape change for each attribute. It is in the Plot2D.java class, this method setCSindex, is called to set every index of the data point. Where the parameter is an integer value represent the shape that has been selected. For the value to be set, every data point in the instance of the plot that requires this value and is nominal will be set. After this is done, the method determineBounds() is called to reconfigure the boundary of the x and y axis, the Boolean status of axis changed is true and the plot is repainted again to reflect the changes made.

```
switch (shape) {

    case X_SHAPE:
      drawX(gx, x, y, size);
      break;

    case PLUS_SHAPE:
      drawPlus(gx, x, y, size);
      break;

    case DIAMOND_SHAPE:
```

```
        drawDiamond(gx, x, y, size);
        break;

    case TRIANGLEUP_SHAPE:
        drawTriangleUp(gx, x, y, size);
        break;

    case TRIANGLEDOWN_SHAPE:
        drawTriangleDown(gx, x, y, size);
        break;

    case CIRCLE_SHAPE:
        drawCircle(gx, x, y, size);
        break;

    case SQUARE_SHAPE:
        drawSquare(gx, x, y, size);
        break;

    case ERROR_SHAPE: // draws the nominal error shape
        gx.drawRect((int) (x - size), (int) (y - size), (size * 2), (size * 2));
        break;

}
```

This code shows how the colours are set. Note that the ***mshIndex*** has the same value as the shape. And when the shape is set, the switch method looks for which of the is the appropriate enum number and performs the method that belongs to it. Below I would outline all the method responsible for each of the shapes we would be using.

For X shape representation

```
private static void drawX(Graphics gx, double x, double y, int size) {

    gx.drawLine((int) (x - size), (int) (y - size), (int) (x + size),
        (int) (y + size));

    gx.drawLine((int) (x + size), (int) (y - size), (int) (x - size),
        (int) (y + size));
```

```
}
```

For Square shape representation

```
private static void drawSquare(Graphics gx, double x, double y, int size) {

      gx.drawLine((int)(x-size),(int)(y+size),(int)(x+size),(int)(y+size));
      gx.drawLine((int)(x-size),(int)(y+size),(int)(x-size),(int)(y-size));
      gx.drawLine((int)(x-size),(int)(y-size),(int)(x+size),(int)(y-size));
      gx.drawLine((int)(x+size),(int)(y-size),(int)(x+size),(int) (y+ size));
}
```

For Plus Shape representation

```
private static void drawPlus(Graphics gx, double x, double y, int size) {

    gx.drawLine((int)(x-size),(int)(y),(int)(x+size),(int)(y));

    gx.drawLine((int)(x),(int)(y-size),(int)(x),(int)(y+size));
}
```

For Diamond shape representation

```
private static void drawDiamond(Graphics gx, double x, double y, int size) {

    gx.drawLine((int) (x - size), (int) (y), (int) (x), (int) (y - size));

    gx.drawLine((int) (x), (int) (y - size), (int) (x + size), (int) (y));

    gx.drawLine((int) (x + size), (int) (y), (int) (x), (int) (y + size));

    gx.drawLine((int) (x), (int) (y + size), (int) (x - size), (int) (y));
  }
```

For Triangle up representation

```
private static void drawTriangleUp(Graphics gx, double x, double y, int size) {

    gx.drawLine((int) (x), (int) (y - size), (int) (x - size), (int) (y + size));
```

```
    gx.drawLine((int) (x-size),(int)(y+size), (int) (x + size), (int) (y + size));

    gx.drawLine((int) (x + size), (int) (y + size), (int) (x), (int) (y - size));

  }
```

For Triangle down representation

```
private static void drawTriangleDown(Graphics gx, double x, double y, int size) {

    gx.drawLine((int) (x), (int) (y + size), (int) (x - size), (int) (y - size));

    gx.drawLine((int)(x-size),(int)(y-size), (int) (x + size), (int) (y -size));

    gx.drawLine((int) (x + size), (int) (y - size), (int) (x), (int) (y + size));

  }
```

For Circle representation

```
private static void drawCircle(Graphics gx, double x, double y, int size) {
        gx.drawOval((int)x, (int)y, 2*size, 2*size);
}
```

For all this to be displayed in the user interface, it has to be set for every instance of the data point in the system engine. Below is a snippet of what we are using to set the code for the shape changes on data points.

```
if(m_setShape) {

   int newShapeIndex = 0;

   for(int kk=0; kk<temp_plot.m_plotInstances.instance(i).numValues(); kk++) {

      if(temp_plot.m_plotInstances.instance(i).value(m_shIndex) == kk) {
```

```
            newShape = kk%7;
        }
    }

    if (temp_plot.m_connectPoints[i] == true) {

        drawDataPoint(x, y, prevx, prevy, temp_plot.m_shapeSize[i], newShape, gx);

    } else {

        drawDataPoint(x, y, temp_plot.m_shapeSize[i], newShape, gx);

    }

}
```

For this code to be hit, m_setShape has to be true, that way the engine loops though all instances of the data sets, picks out the shape and if the shape index is kk(which is a variation of all the variables in the data plots instances), the newshapeIndex is set to have an index of kk. This new shape index is now passed into the draw data point method which is the method with the switch statements explained earlier. These create a rectangle shape for each of the existing attributes and then fills the rectangle with the colour corresponding to the attribute. There by making it appear like a label and easier to read and giving it more aesthetics as well. The class were all these were added was in the ClassPanel.java class.

Background colour modification possible

For this feature to be added, series of code changes were added to the VisualizePanel.java and the Plot.java classes.

VisualizePanel.java class

```
protected JButton m_BckClrBut = new JButton("Bgd Colour");
```

This statement creates a button for the visualization panel class, where we can click in order to change the background colour. For this button to display in the window panel, it is also added to the main Janel in the constructor of the visualizaPanel class. The name of the button would be called “Bgd Colour”

```
m_BckClrBut.setToolTipText("Change Background Colour");
```

```
m_BckClrBut.addActionListener(new ActionListener() {

 @Override
  public void actionPerformed(ActionEvent e) {

   openColourPanel(e);

  }
});
```

This is the action listener which performs an action once the Background colour button is clicked. The action it performs in our case is the openColourPanel method.

```
protected void openColourPanel(ActionEvent e)
  {
        int m_iindex = 0;
      if ((e.getModifiers() & InputEvent.BUTTON1_MASK) == InputEvent.BUTTON1_MASK)
{
          Color tmp = JColorChooser.showDialog(VisualizePanel.this,
            "Select new Color", m_colorList.get(m_iindex));

          if (tmp != null) {
            m_colorList.set(m_iindex, tmp);
            VisualizePanel.this.repaint();
            rememberBgdCol = tmp;
            m_plot.setBackground(tmp);
          }
        }
  }
```

This method is now the method that opens a GUI colour chooser on the button click, when the user selects a colour and applies it, the class would repain itself via the VisualizePanel.**this**.repaint(); to apply the changes that have been selected. Then the background colour is set in the m_plot class via the method setBackground(). To now change the background colour, we only need to click on the Bgd Colour and a colour chooser would open for the colour to be selected.

Personalize the colour of the axes

For this feature to be added, series of code changes were added to the VisualizePanel.java, Plot2D.java and the Plot.java classes.

```
protected JButton m_AxisBut = new JButton("Axis Color");
```

This statement creates a button for the visualization panel class, where we can click in order to change the axis colour. For this button to display in the window panel, it is also added to the main JPanel in the constructor of the visualizaPanel class. The name of the button would be called “Axis Colour”

```
m_AxisBut.setToolTipText("Change Axis Colour");
m_AxisBut.addActionListener(new ActionListener() {
 @Override
public void actionPerformed(ActionEvent e) {

   openAxisColourPanel(e);

  }
});
```

This is the action listener which performs an action once the Background colour button is clicked. The action it performs in our case is the openColourPanel method.

```
 protected void openAxisColourPanel(ActionEvent e){

        int m_iindex = 0;

    if ((e.getModifiers() & InputEvent.BUTTON1_MASK) == InputEvent.BUTTON1_MASK) {

          Color tmp = JColorChooser.showDialog(VisualizePanel.this,
            "Select new Axis Color", m_colorList.get(m_iindex));

          if (tmp != null) {

            m_colorList.set(m_iindex, tmp);

            VisualizePanel.this.repaint();

            m_plot.m_plot2D.m_axisColour = tmp;
```

```
            }
        }
    }
```

This method is now the method that opens a GUI colour chooser on the button click, when the user selects a colour and applies it, the class would repaint itself via the VisualizePanel.**this**.repaint(); to apply the changes that have been selected. Then the background colour is set in the m_plot2D that is in the Plot.java class via the m_axisColour variable. To now change the axis colour, we only need to click on the Axis Colour and a colour chooser would open for the colour to be selected.

Set minimum and maximum axis

Below is code changes and addition that were implemented for this feature to work. The additions were also made in the VisualizePanel.java and Plot2D.java classes.

VisualizePanel.java

```
protected JButton m_setPlot = new JButton("Set Plot");
```

This statement creates a button for the visualization panel class, where we can click to set the same plot given newly defined maximum and minimum x and y axes. For this button to display in the window panel, it is also added to the main JPanel in the constructor of the visualizaPanel class. The name of the button would be called "`Set Plot`".

```
protected JButton m_resetPlot = new JButton("Reset Plot");
```

This statement creates a button for the visualization panel class, where we can click to reset the same plot to its original maximum and minimum x and y axes. For this button to display in the window panel, it is also added to the main JPanel in the constructor of the visualizaPanel class. The name of the button would be called "`Reset Plot`".

```
m_setPlot.setToolTipText("Change Plot Configuration");
```

This is just a tooltip setter that displays an information "`Change Plot Configuration`" when the mouse hovers over "`Set Plot`"

```
m_setPlot.addActionListener(new ActionListener() {

@Override
public void actionPerformed(ActionEvent e) {
   JLabel Xmin = new JLabel("X min: ");
   xmin0 = m_xmin;
   JLabel Xmas = new JLabel("X max: ");
   xmax0 = m_xmas;

   JLabel Ymin = new JLabel("Y min: ");
   ymin0 = m_ymin;
   JLabel Ymas = new JLabel("Y max: ");
   ymax0 = m_ymas;
   JLabel TickSize = new JLabel("tick Size: ");
   newTickSize = m_tickSize;

   JButton doneBt = new JButton("Done");

   final JDialog jd =
            new JDialog((JFrame) VisualizePanel.this.getTopLevelAncestor(),
              "Set Plot", ModalityType.DOCUMENT_MODAL) {
   private static final long serialVersionUID = -269823533147146296L;

   @Override
   public void dispose() {

     m_plot.m_plot2D.m_XaxisStart = Integer.parseInt(xmin0.getText());
     m_plot.m_plot2D.m_XaxisEnd = Integer.parseInt(xmax0.getText());
     m_plot.m_plot2D.m_YaxisStart = Integer.parseInt(ymin0.getText());
     m_plot.m_plot2D.m_YaxisEnd = Integer.parseInt(ymax0.getText());
           super.dispose();
       }
   };

   doneBt.addActionListener(new ActionListener() {

   @Override
   public void actionPerformed(ActionEvent ae) {
       resetAxisPlotPanel(ae);
       setAxisPlotPanel(ae);
```

```
            jd.dispose();
        }
    });
            GridBagLayout gbl = new GridBagLayout();
            GridBagConstraints gbc = new GridBagConstraints();
            JPanel p1 = new JPanel(gbl);
            gbc.anchor = GridBagConstraints.WEST;
            gbc.fill = GridBagConstraints.HORIZONTAL;
            gbc.insets = new Insets(0, 2, 2, 2);
            gbc.gridwidth = GridBagConstraints.RELATIVE;
            p1.add(Xmin, gbc);
            gbc.weightx = 0;
            gbc.gridwidth = GridBagConstraints.REMAINDER;
            gbc.weightx = 1;
            p1.add(xmin0, gbc);
            gbc.insets = new Insets(0, 2, 2, 2);
            gbc.gridwidth = GridBagConstraints.RELATIVE;
            p1.add(Xmas, gbc);
            gbc.gridwidth = GridBagConstraints.REMAINDER;
            gbc.weightx = 1;
            p1.add(xmax0, gbc);
            gbc.insets = new Insets(8, 2, 2, 2);

            gbc.gridwidth = GridBagConstraints.RELATIVE;
            p1.add(Ymin, gbc);
            gbc.weightx = 1;
            gbc.gridwidth = GridBagConstraints.REMAINDER;
            gbc.weightx = 1;
            p1.add(ymin0, gbc);
            gbc.insets = new Insets(0, 2, 2, 2);
            gbc.gridwidth = GridBagConstraints.RELATIVE;
            p1.add(Ymas, gbc);
            gbc.gridwidth = GridBagConstraints.REMAINDER;
            gbc.weightx = 1;
            p1.add(ymax0, gbc);
            gbc.insets = new Insets(8, 2, 2, 2);

            gbc.gridwidth = GridBagConstraints.RELATIVE;
            p1.add(TickSize, gbc);
            gbc.gridwidth = GridBagConstraints.REMAINDER;
```

```
            gbc.weightx = 1;
            p1.add(newTickSize, gbc);
            gbc.insets = new Insets(8, 2, 2, 2);

            JPanel p3 = new JPanel(gbl);
            gbc.fill = GridBagConstraints.HORIZONTAL;
            gbc.gridwidth = GridBagConstraints.REMAINDER;
            gbc.weightx = 1;
            gbc.weighty = 0;
            p3.add(p1, gbc);
            gbc.insets = new Insets(8, 4, 8, 4);
            p3.add(doneBt, gbc);

            jd.getContentPane().setLayout(new BorderLayout());
            jd.getContentPane().add(p3, BorderLayout.NORTH);
            jd.pack();
            jd.setLocation(m_setPlot.getLocationOnScreen().x,
              m_setPlot.getLocationOnScreen().y - jd.getHeight());
            jd.setVisible(true);
        }
    });
```

This is the action listener for the set plot button. Once the button is click, the function above is called. What this function typically does is to open a panel, for setting the new maximum and minimum values of the x and y axes. It the “done” button that called the method that sets the plot. This function is found just below.

```
protected void setAxisPlotPanel(ActionEvent e) {
    if ((e.getModifiers() & InputEvent.BUTTON1_MASK) == InputEvent.BUTTON1_MASK){
        int x,y,c,sh;

        x = m_plot.m_xIndex;
        y = m_plot.m_yIndex;
        c = m_plot.m_cIndex;
        sh = m_plot.m_shIndex;

        double x1,x2,y1,y2, tick, xvalue, yvalue;

          x1 = Integer.parseInt(xmin0.getText());
          x2 = Integer.parseInt(xmax0.getText());
          y1 = Integer.parseInt(ymin0.getText());
          y2 = Integer.parseInt(ymax0.getText());
```

```
        tick = Integer.parseInt(newTickSize.getText());

  Instances insts = new
  Instances(m_plot.m_plot2D.getMasterPlot().m_plotInstances, 500);

        for(int noa=0; noa<m_plot.m_plotInstances.numInstances(); noa++) {

           xvalue = m_plot.m_plot2D.getMasterPlot().m_plotInstances
                    .instance(noa).value(m_plot.m_xIndex);

           yvalue = m_plot.m_plot2D.getMasterPlot().m_plotInstances
                    .instance(noa).value(m_plot.m_yIndex);

           if ((xvalue <= x2)&&(xvalue >= x1)&&(yvalue <= y2)&&(yvalue >= y1)) {

                  insts.add(m_plot.m_plot2D.getMasterPlot().m_plotInstances
                  .instance(noa));
                }
            }

           if(insts.numInstances()==0) {

                  resetAxisPlotPanel(e);
           }else {

          PlotData2D tempd = new PlotData2D(insts);

          m_plot.m_plot2D.removeAllPlots();

          try {
             addPlot(tempd);
          }catch(Exception ex) {

                ex.printStackTrace();
                System.err.println(ex.getMessage());

          }
          m_plot.m_plot2D.setPlotAxis((int)x1, (int)x2,
          (int)y1, (int)y2, (int)tick, true);

           try {
                  VisualizePanel.this.setXIndex(x);
                  VisualizePanel.this.setYIndex(y);
                  VisualizePanel.this.setCIndex(c);
                  VisualizePanel.this.setShIndex(sh);

           } catch(Exception ex) {}
      }
    }
}
```

This is the actual engine responsible for setting the maximum and minimum x and y axes values. This function receives an input of the newly desired x and y axes, runs a for loop of all instances and picks only values that fall within the newly inputted values. This is just a way for filtering out the unneeded values, after that the values that fulfilled the condition of the for loop are then replotted on the graph plot.

```
m_resetPlot.setToolTipText("Reset Plot");
m_resetPlot.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        resetAxisPlotPanel(e);
    }
});
```

This is the action listener for the reset plot button. Once the reset button is click, the function above is called. What this function typically does is to restore the graph plot originally to the way it was before set plot button was clicked. This function is found just below.

```
protected void resetAxisPlotPanel(ActionEvent e)  {

    if ((e.getModifiers() & InputEvent.BUTTON1_MASK) == InputEvent.BUTTON1_MASK) {

         if(m_plot.m_plot2D.keyPressed) {
           int x = m_plot.m_xIndex;
           int y = m_plot.m_yIndex;
           int c = m_plot.m_cIndex;
           int sh = m_plot.m_shIndex;

                       m_plot.m_plot2D.removeAllPlots();

                       try {
                         VisualizePanel.this.addPlot(unSetPlot);
                       } catch (Exception ex) {
                         System.err.println(ex);
                         ex.printStackTrace();
                       }
           m_plot.m_plot2D.setPlotAxis(0, 0, 0, 0, 0, false);

           try {
             VisualizePanel.this.setXIndex(x);
             VisualizePanel.this.setYIndex(y);
             VisualizePanel.this.setCIndex(c);
             VisualizePanel.this.setShIndex(sh);
           } catch (Exception er) {
             System.out.println("Error : " + er);
           }

       VisualizePanel.this.repaint();
         }
       }
  }
```

This is the actual engine responsible for the plot to have their original values. Note that this must be reset for all action listeners as well so that they would be able to remember the original state of the graph plot.

```
public void removeAllPlots() {
    m_masterPlot = null;
    m_plotInstances = null;
    m_plots = new ArrayList<PlotData2D>();
    m_xIndex = 0;
    m_yIndex = 0;
    m_cIndex = 0;
  }
```

This is useful when setting new plots because all existent plots have to be purged out for the new plot to be plotted successfully.

```
private void plotReset(Instances inst, int cIndex, int sIndex) {
      .
      .
      .
      .
      .
      .
      m_xIndex = xIndex; //- for the x axis
      m_yIndex = yIndex; //- for the y axis

      //this was my addition
      m_cIndex = cIndex;
      m_sIndex = sIndex;
      cancelShapes();
}
```

So, the issue was that when reset button is clicked and all values are reset, the x and y axis are reset, but the colour and shape axes were previously not set. The lines in dots are beyond the scope of this thesis.

Dynamic Feature Value Selection

Filtering data set

Below are the code changes that were added to the existing software application to make the feature possible. The code additions were done mainly in the ClassPanel.java class.

```
NomCheckBx cc = new NomCheckBx(i);

if((IndexCollections.contains(String.valueOf(i)))&&(checkBxCollections !=null)) {
```

```
  boolean fff = true;
  if(checkBxCollections.get(i) == "false")
       fff = false;
  cc.setSelected(fff);
}
else {

   if(strAttr.contains(String.valueOf(i+1))) {
       cc.setEnabled(false);
       cc.setSelected(false);
   }else
       cc.setSelected(true);
   }
   int testVal = m_Instances.attributeStats(m_cIndex).nominalCounts[i];
   if (testVal == 0) {
      cc.setSelected(false);
      cc.setEnabled(false);
   }

   cc.setSize(m_labelMetrics.stringWidth(jj.getText()),
   m_labelMetrics.getAscent() + 4);
   this.add(cc);
   cc.setLocation(x, y-3);
```

This code snippet is found to the PaintNominal() method and is repeated 4 times for every condition that is dictated in the for loop looping through every instance. It replicates the checkbox for all the instances, checks if the checkbox is enabled (has been clicked). The creation of this feature would not have been easy or even possible without the checkbox component that was utilised.

```
private class NomCheckBx extends JCheckBox {

      /** for serialization */
     private static final long serialVersionUID = -4686613106474820655L;
     private int m_index = 0;
     public NomCheckBx (int id) {
         super();
         m_index = id;
         defaultColor.add(m_colorList.get(m_index));
```

```
JCheckBox pass = this;
for(int k=0; k<m_colorList.size(); k++) {
     checkBxCollections.add(String.valueOf(true));
}

pass.addActionListener(new ActionListener() {
  @Override
  public void actionPerformed(ActionEvent e) {
    if(!pass.isSelected()) {
          m_valCheckBxSel = false;
          pass.setSelected(m_valCheckBxSel);
          m_colorList.set(m_index, VisualizePanel.rememberBgdCol);
          sltdIndex = String.valueOf(m_index+1);
          sltdAttr = String.valueOf(m_cIndex+1);
          dnomIndexArr.put(m_index, sltdIndex);

     }else {

           m_valCheckBxSel = true;

           pass.setSelected(m_valCheckBxSel);

           dnomIndexArr.remove(m_index);

           m_colorList.set(m_index, defaultColor.get(m_index));

     }



   checkBxCollections.set(m_index, String.valueOf(m_valCheckBxSel));

    IndexCollections.add(String.valueOf(m_index));

    List<String> result2 = dnomIndexArr.values().stream()
                .collect(Collectors.toList());

    RemoveWithValues rem = new RemoveWithValues();
```

```
            int intarray[] = new int[result2.size()];
            int ii=0;

            for(String str:result2) {

                intarray[ii]=Integer.parseInt(str)-1;
                ii++;
            }

            rem.setNominalIndicesArr(intarray);
            rem.setAttributeIndex(sltdAttr);

            try {
                  gblRem = rem;
            } catch (Exception ex) {}

            m_oldWidth = -9000;
            ClassPanel.this.repaint();

            if (m_Repainters.size() > 0) {
                  for (int i = 0; i < m_Repainters.size(); i++) {
                        (m_Repainters.get(i)).repaint();
                  }
             }

        });
```

This nomcheck box class extends the Jcheckbox class, so that it can contain all its features. The constructor of this call is where all the action happens, action listeners that detects when the checkboxes are clicked or not are present in the constructor. When they are click, the data points values are filtered out with the background colour of the graph, to hide the data plot for that attribute. And when they are selected the attributes are given back their original colour and thereby are now visible on in the graph. Notice also, that upon election and deselection, there is a remove filtered that is called and used, this would be explained in detail in the next feature.

Filtering data for plugins

Below are the code changes that was added to the existing software application to make the feature possible. The code additions were done in the ClassPanel.java, MatrixPanel.java and the PreprocessPanel.java classes.

ClassPanel.java

RemoveWithValues rem = **new** RemoveWithValues();

```
int intarray[] = new int[result2.size()];
int ii=0;
for(String str:result2) {
    intarray[ii]=Integer.parseInt(str)-1;
    ii++;
}
rem.setNominalIndicesArr(intarray);
rem.setAttributeIndex(sltdAttr);
```

This RemovewithValue class is an inbuilt filter property from the WEKA software application. And it came in handy in the implementation of this feature across platform. It enabled effective filtering to be done by value once the checkbox is unselected.

MatrixPanel.java

```
jf.addWindowListener(new java.awt.event.WindowAdapter() {
    //@Override
    public void windowClosing(java.awt.event.WindowEvent e) {
            jf.dispose();

            gblRem = ClassPanel.gblRem;
             PreprocessPanel.gblRem = gblRem;
             PreprocessPanel.m_TFilter.doClick();
            System.exit(0);
          }
   });
```

This is the action listener that closes the visualization panel, it makes it possible for the Class panel filtered out attributes to be stored so that they can then be applied to the Preprocess panel, it then triggers an automatic click function doClick().

PreprocessPanel.java

```
public static JButton m_TFilter = new JButton("Filter Check boxes");
```

This initialises an invisible button called "`Filter Check boxes`" for applying the filters on checkboxes been selected. It is invisible on purpose because this button does not need to be seen in the GUI but only used in the backend when required.

```
m_TFilter.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
      try {

        applyFilter(gblRem);
        m_TFilter.setEnabled(false);
      } catch (Exception ex) {
        if (m_Log instanceof TaskLogger) {
          ((TaskLogger) m_Log).taskFinished();
        }
        // Pop up an error optionpane
        JOptionPane.showMessageDialog(PreprocessPanel.this,
          "Problem filtering instances:\n" + ex.getMessage(),
          "Remove Attributes", JOptionPane.ERROR_MESSAGE);
        m_Log.logMessage("Problem removing attributes: " + ex.getMessage());
        m_Log.statusMessage("Problem removing attributes");
        ex.printStackTrace();
      }
    }
});
```

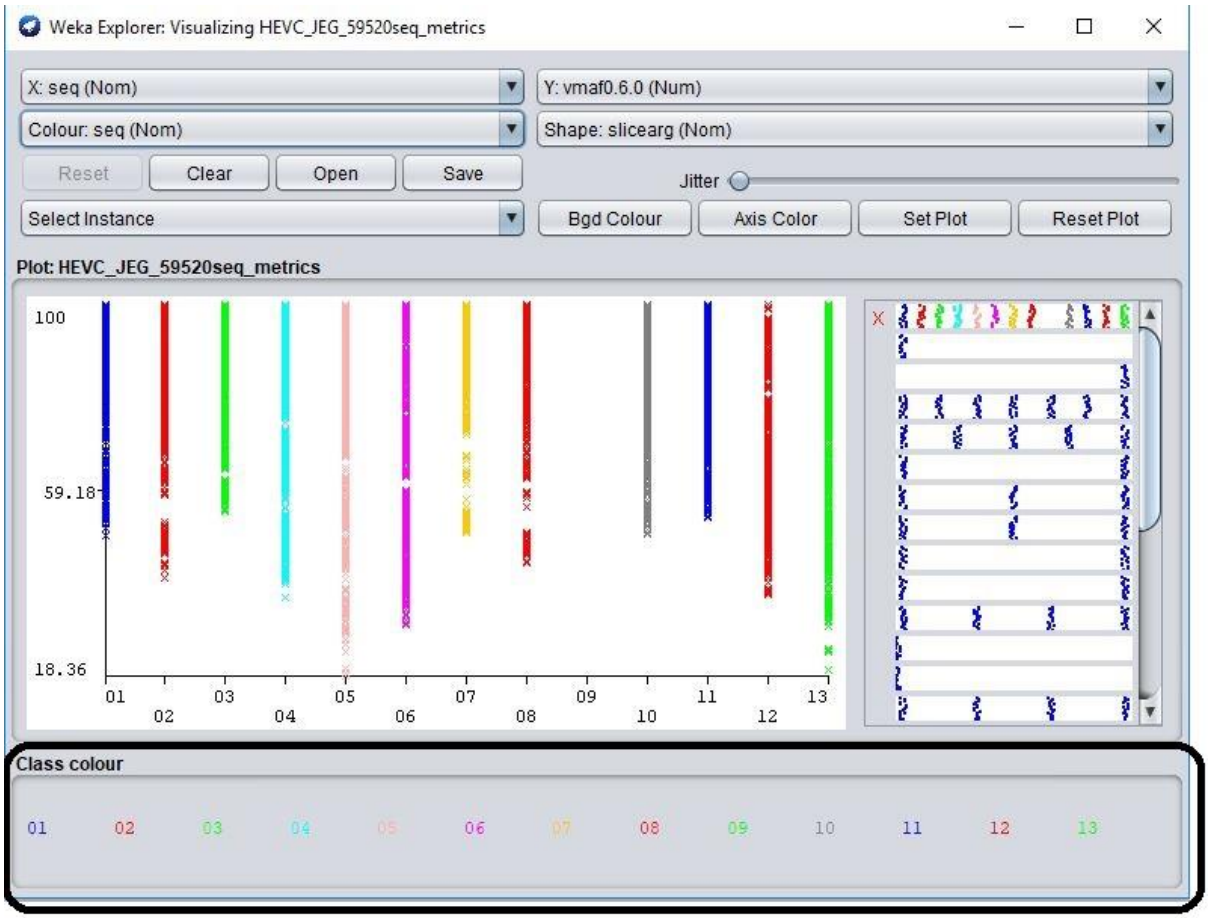
This is the function that basically implements the filter and applies it to all instances of the software application. This was it is available across panels and plugins as well.

CHAPTER 4 – Results

User Interface

1. Improved Label readability

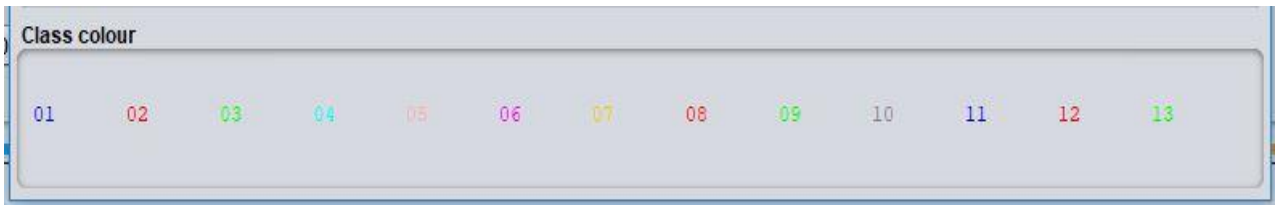
This added feature is just about being able to better read the labels i.e. having labels coloured black then the colour associated to it on the side, not colouring just the name of the label. Previously, colours for data plots can be changed by clicking on the label but now the colour can be easily changed still by click on the label but there is an additional panel that easily depicts the colour selected. This make the software more user friendly and easy to understand which attributes have been set to a particular colour. It also allows for easy change in modification of the attributes colour if the case need be.



4a.1.1 Visualization Panel with data plot

It is good to note that this feature is solely meant for nominal values and not string nor numeric values. For the purpose of understanding in the X, Y, Colour and Shape section of the image that was shown above, Nominal values are represented with "(Nom)", Numerical values are represented with "(Num)", while string values are represented with "(Str)" This feature, can be found in the class panel, as shown above. Notice the section highlighted in bold black at the bottom of the image, this is the class panel. And this is where the colours for data points values are represented. Also, you would observe that for each of the number of attributes present in the plot page (13 in this instances), there are 13 labelled coloured values in the class panel.

Previously, data points colours were represented in class panels, but this was rather inconvenient at time. Most especially when the colour that is selected, is close to the colour of the class panel background. Also, since the labels were represented on numbers, it was very difficult to see since numbers have a small size. It was also difficult at times when the user what to change the colour because of the number size and because by looking at the numbers it is difficult to tell that they are clickable. This makes it very unfriendly from the user point of view. Below is a clearer and bigger picture of what the previous class colour panel looked like.

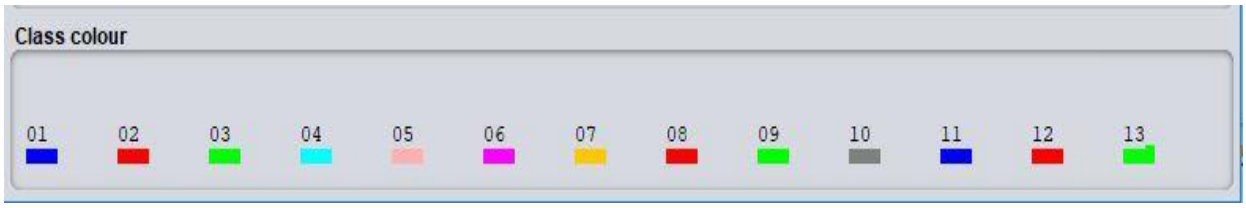


4a.1.2 Class Panel with coloured numbers

While this diagram portrays the previous problems highlighted above, it also gave the software application at empty fill and made the user interface looked scanty. With the introduction of this new features all the problems highlighted so far have been solved. There are now separate labels that fit just below the numbers (which are not black in colour by default), to represent the colour of the data points. Also, these labels are rectangle, they are easily noticeable and can be use for changing easily the colours of the attributes. The good part is that it is user intuitive i.e. it is easy for the user to figure out that I could change my attribute colours from here. Another interesting way of manipulation that I would say.

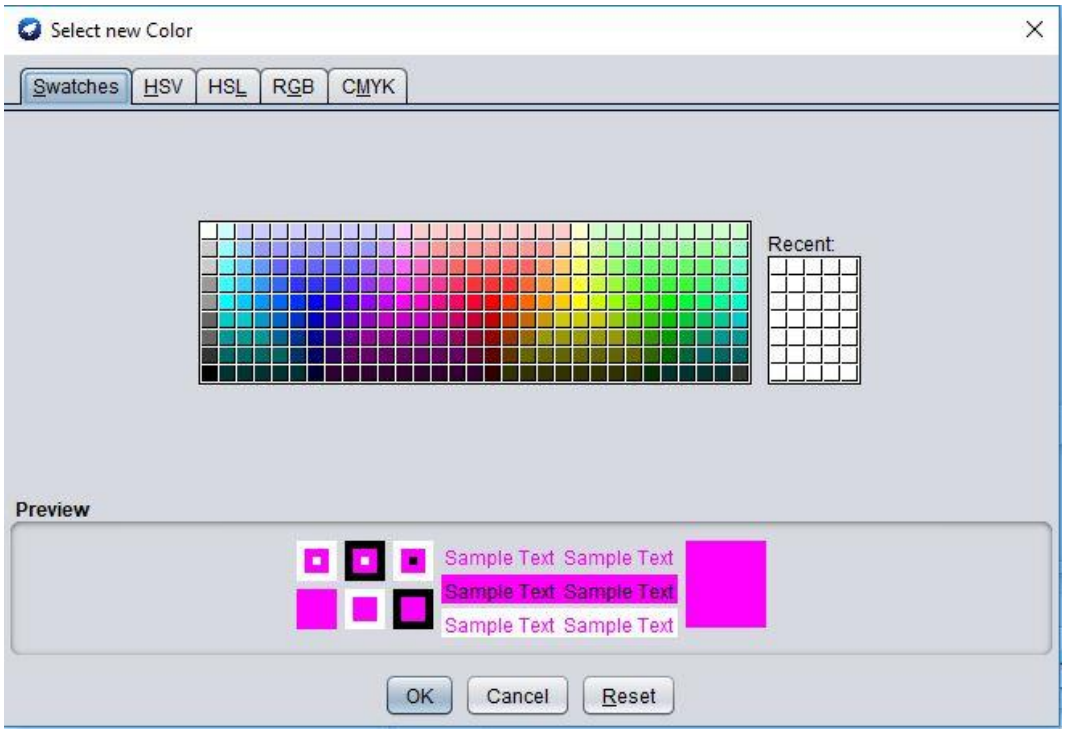
To make this possible some lines of code was added to the existing code which has been discussed in chapter 3. The class were all these were added was in the ClassPanel.java class.

This class is responsible for all the logic and user interface responsible for the Class colour section of the image Below is an image showing the newly added feature.



4a.1.3 Class Panel with black numbers and coloured label.

Looking at the diagram above we can see now the difference. The label colour representation perfectly fits under the number fonts. And it is easy to distinguish the name of the attributes from the colour of the attributes. Also, by clicking on any of the colour shapes (labels), a pop up would open with a colour chooser where we would be able to change the colour of the attributes to whatsoever we desire.



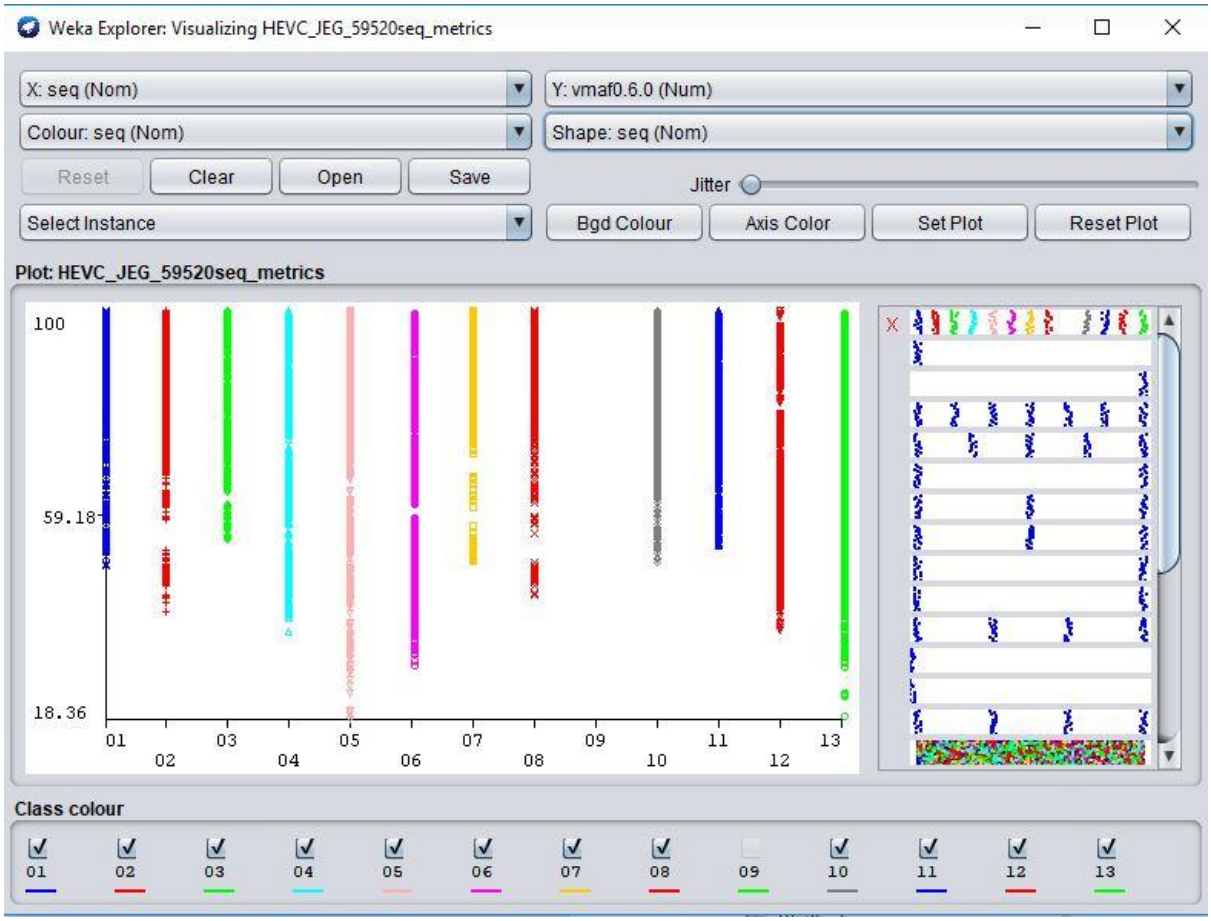
4a.1.4 Colour Chooser Panel.

2. Changeable shapes for plotted points

This is just the ability to be able to change the shape of plotted points, previously the shapes of all plotted points were just generic. They were only of the shape “x”. This limited the usability of the application because users could not manipulate data with respect of reference to shapes but only the colours. One interesting thing about this feature is that, it has made it possible for the data points to be changed to different shapes. But this version of the

improvement, we made it possible for the shapes to be up to seven. “cross - X”, “plus - +”, “square ”, “circle”, “Diamond”, “upward triangle” and “downward triangle”. These shapes are defined as static final integers assigned a corresponding number which is also known as its index so that they can be used as an enum. When this index is called automatically the logic should draw the respective shape. There are other static variables like Error, Missing and Default and Max and their names connote their meaning. For all these to be displayed in the user interface, it has to be set for every instance of the data point in the system engine. Below is a snippet of what we are using to set the code for the shape changes on data points.

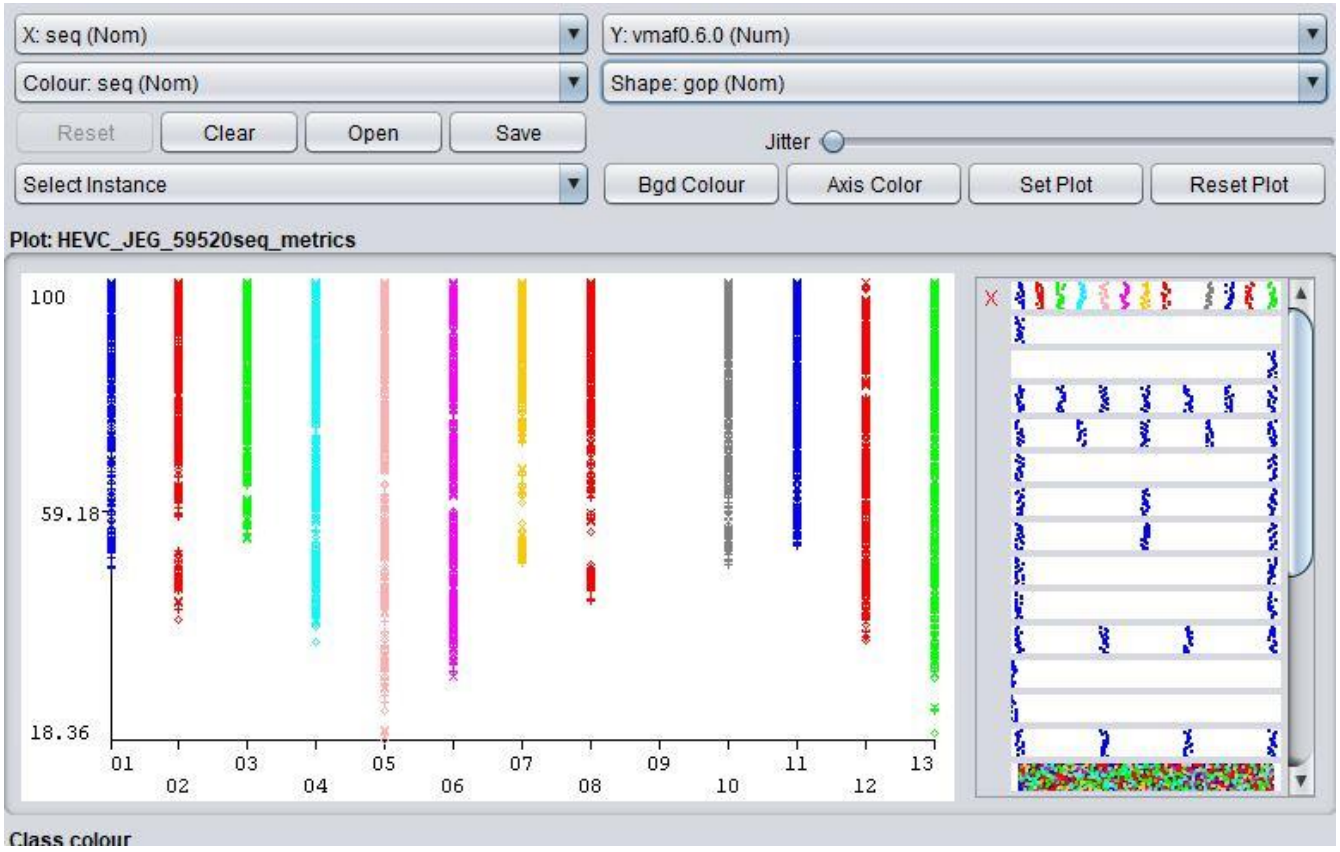
In the next is an image showing a plot with different set instance, you should notice that this plot has different shapes belonging to a particulate attribute and that these attributes can only have their shape changed when they are nominal. And depending on the nominal attributes the dispersion of shape change differs. Just like for colour changes as well.



4a.2.1 Visualization Panel with data plot showing different shape colour.

You would observe that for this particular selected attribute, the shapes representations are repeated after a particular number. This is because of the modulus sign (%) used in the code

above. Since we only have 7 different shapes, we reuse them again if the case need be. Also, the instance in this example where the attribute X axis is Seq(Nom), Y axis is vmaf0.6.0 (Num), Colour is seq(Nom) and shape selected is seq(Nom) every attribute has a different shape. But this is only because the seq(Nom) is selected. But this is only because the seq(Nom) is selected. If another attribute was to be selected, it would have a different arrangement for the shape representation. Now let's take a look at the next diagram where all other selected values are still the same except the shape. X axis is Seq(Nom), Y axis is vmaf0.6.0 (Num), Colour is seq(Nom) and shape selected is gop(Nom), we would notice that the shapes are dispersed in a way to explain the data manipulation. And an attribute could contain more that one shape. Take attribute 8 for example, it contains the plus shape representation, the diamond shape representation as well as the x shape representation.

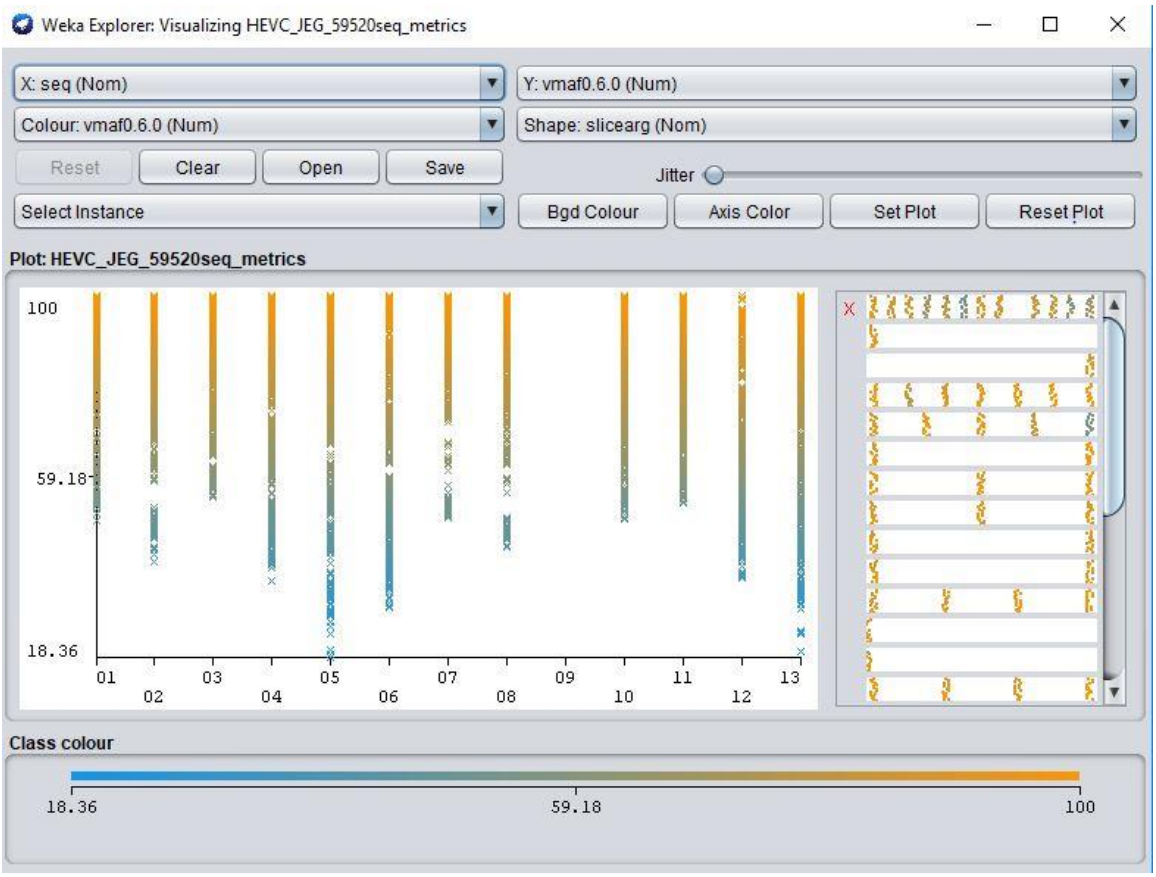


4a.2.2 Visualization Panel with data plot showing different shape colour.

3. Background Colour modification feature has been added to the project.

Previously, the background colour of the Visualization plot was unchangeable and was always white. The could cause inconvenience when investigation data as some of the data

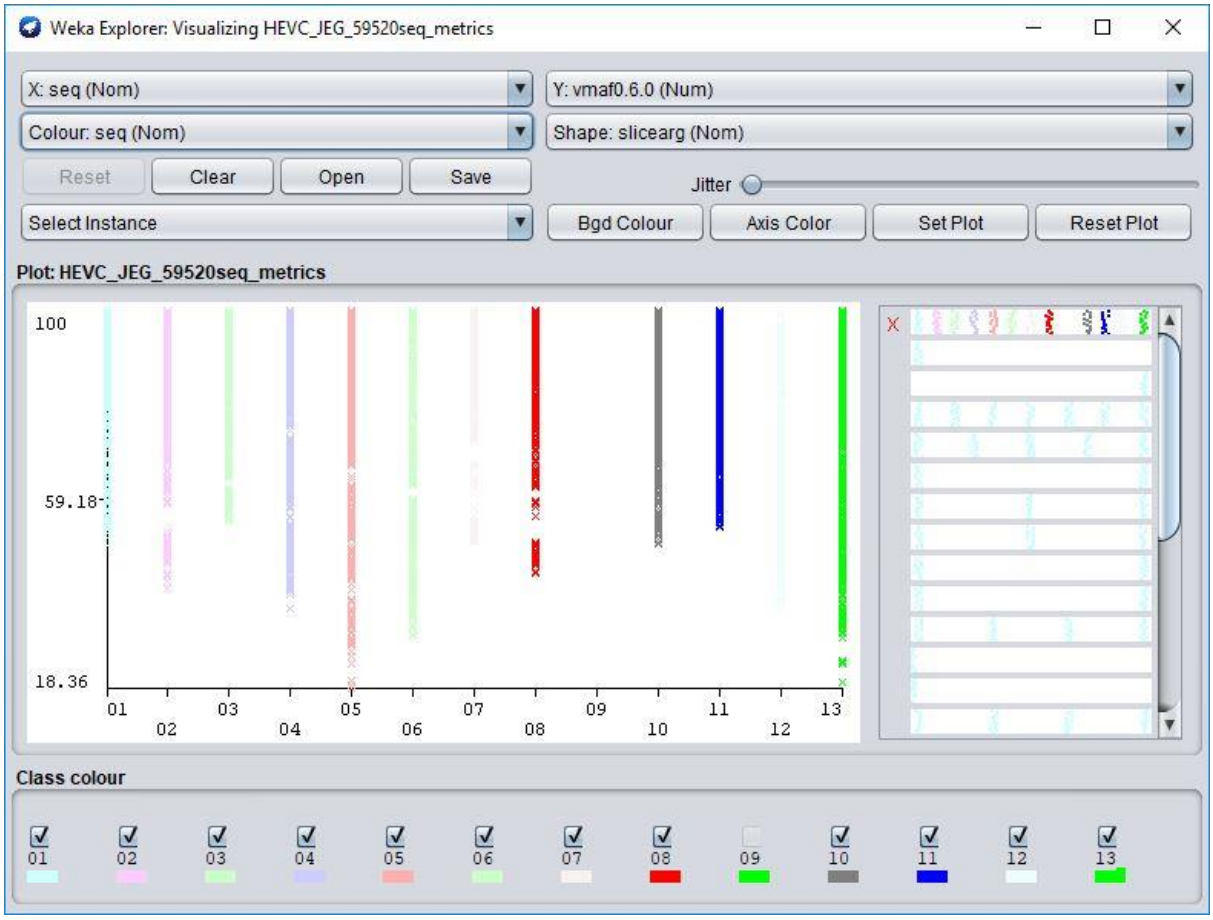
attributes can have colours that are very closely related to the background colour in terms of property making it tricky and very difficult to see or tell the difference. For example, if the data set is identified by a light or whitish colour, it would be difficult to understand the data point for the data as the background is of simple white colour. For numeric data values this it now a problem as data sets utilities only a range between 2 set of colours as in the image below. But data visualization becomes a problem when we are viewing nominal data sets. The diagram below is a depiction od the fact that this application needed improvement. While it might not have been easily noticeable from the beginning, the fact that the plot background colour was only white and could not be changed before proposed many hinderances.



4a.3.1 Visualization Panel with data plot on white background.

This looks good and clear now but can easily get vague and confusing when the colour properties of the attribute have been changed and are now very similar to the background colour, I would change the colour property of some of the data sets below and you would notice how unclear some of them are. Note that, I manually changed the data property colours for this illustration, but the opaqueness normally happens automatically sometimes when a nominal attribute is selected. Let's say for example, a user selects seq(Nom), and he/she decided to

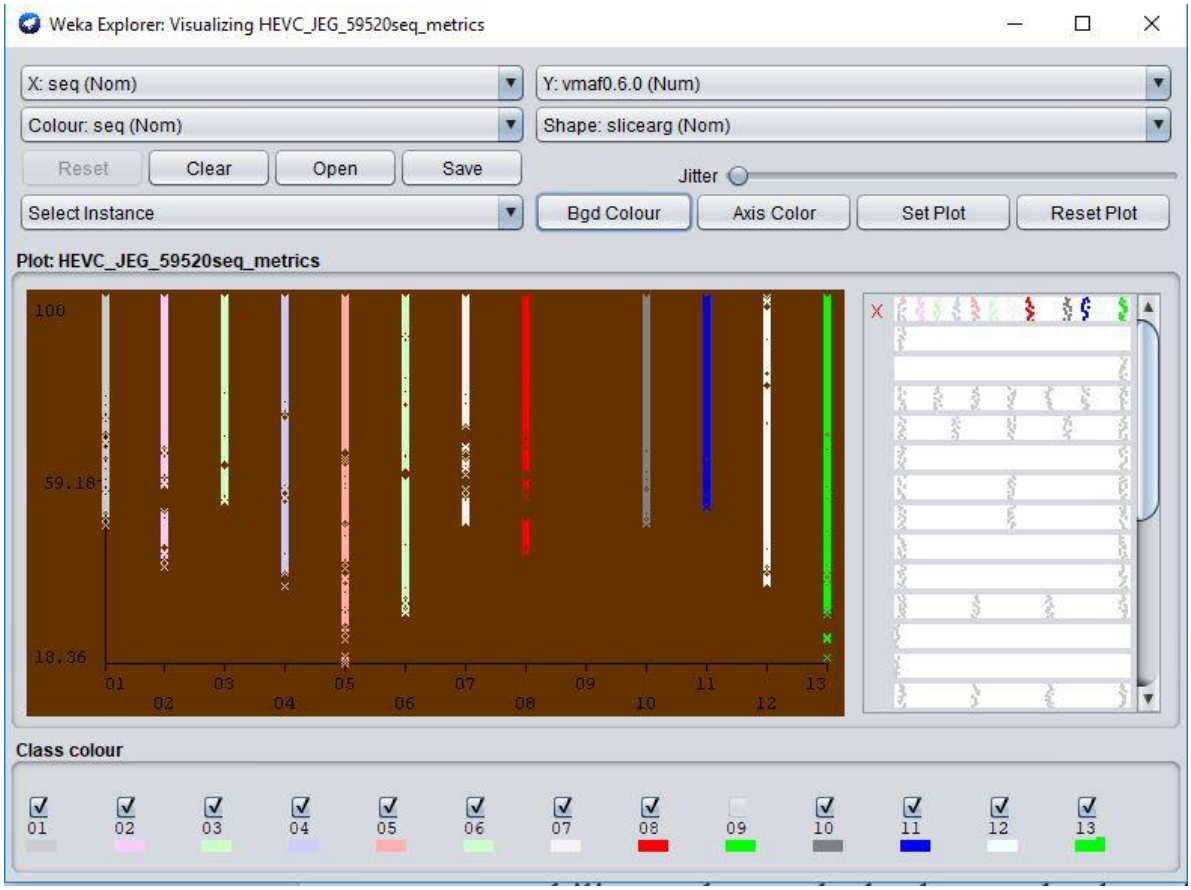
change to modify the background colour to a colour very vague and extremely similar to white. This could cause unclarity, but it is a good thing that he can always change it back to a more suitable colour. But what if this colour change was automatically cause by the colour filter within the code and there is no way to change the colour and neither the background colour as we already know. This would not be a pleasant issue for a user to find a work around to. In the diagram below is portrayed a nearly perfect example the one described earlier.



4a.3.2 Visualization Panel with data plot (light colours) on white background.

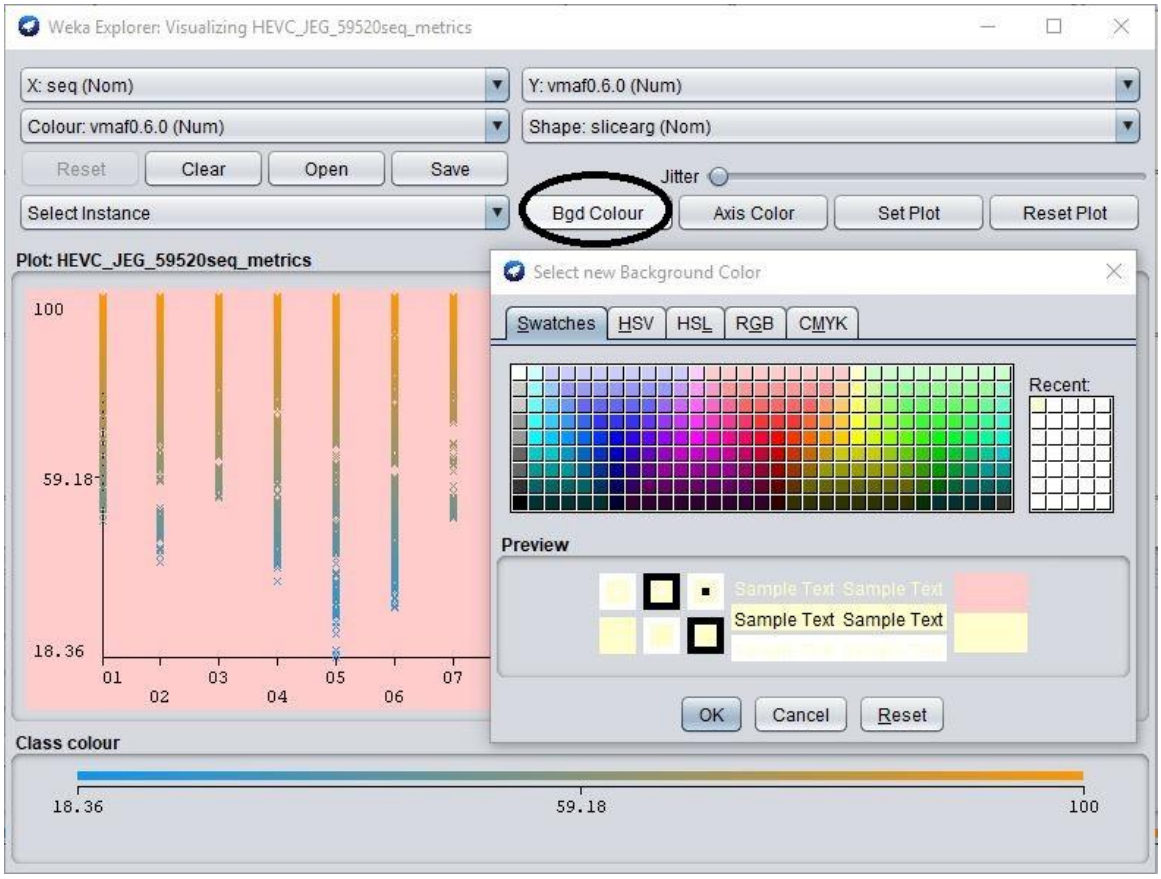
You would notice the data plot colour for the attributes, most of them are very unclear with attribute 12 been the most unclear. Note that the colours of this data points can be changed because seq(nom) was selected as colour and it is a nominal value. This diagram is a nightmare for a user trying to distinguish between data sets, but with the ability to change the background colour, data sets would always appear clear.

For this feature to be added, series of code changes were added to the VisualizePanel.java and the Plot.java classes which have been discussed in chapter 3.



4a.3.3 Visualization Panel with data plot (light colours) on modified background.

Another advantage of this ability to change the background colour as well is that, the data sets can easily be views anywhere. Previously, if it was very sunny outside, it could be difficult to understand your datasets plots but with the ability to change the background colour, this issued is automatically solved.

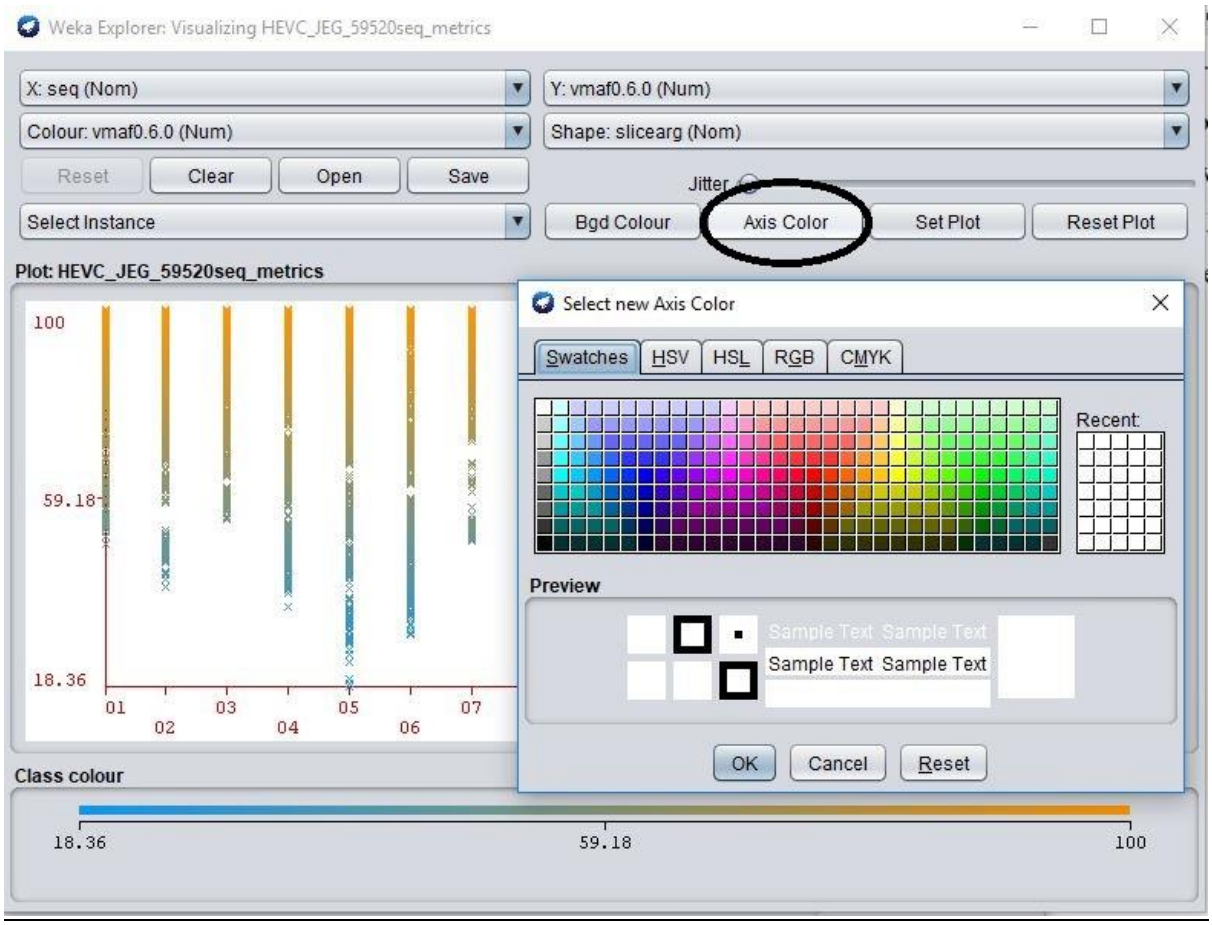


4a.3.4 Visualization Panel after click on Background button.

4 Personalize the colour of the Axes

Looking at the feature added in the previous point, another interesting feature to the software application would now be a little step forward. The ability for use to be able to change the colours for our x and y axes as well. Previously, the axis colour of the Visualization plot was unchangeable and was always black. The could cause inconvenience when investigation data as some of the data attributes can have colours that are very closely related to the background colour in terms of property making it tricky and very difficult to see or tell the difference. For example, if the background of the plot is identified by a dark or blackish colour, it would be difficult to understand the data point for the data compared to the background of a lighter or white colour. For numeric data values this it now a problem as data sets utilities only a range between 2 set of colours as in the image below. But data visualization becomes a problem when we are viewing nominal data sets. The previous diagram below is a depiction of the fact that this application still needed improvement. While it might not have been easily noticeable from the beginning, the fact that the plot axis was only black and could not be changed before proposed many hinderances. For this feature to be added, series of code

changes were added to the VisualizePanel.java, Plot2D.java and the Plot.java classes discussed in chapter 3.



4a.4.1 Visualization Panel after click on Background button.

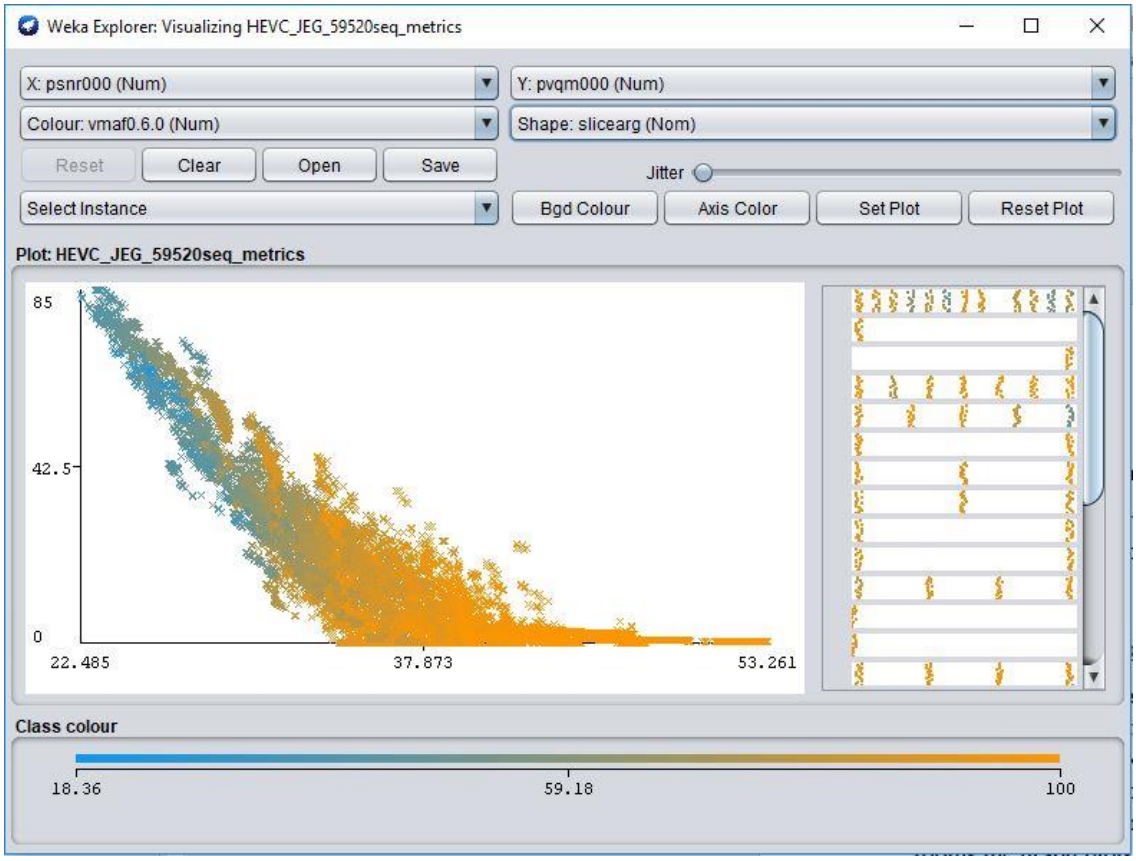
As can the observed the present colour of the axis above is red. This means that axis colour can now be easily changed in the advent of any unclear data changes or for easy readability of data.

5 Set minimum and maximum for the axes

Previously, with the software application only displays graph plots of data set loaded into it. But suppose n the visualize panel a user wants to inspect the data sets between a particular area in the y axis and the x axis, plotting this to a specific point would be impossible because until now the feature did not exist. Now, we can pick out particular areas of the plot we want to see by just specifying the maximum and minimum values for the x and y axes and this automatically zooms the graph plots to the measure given. This comes in very handy to users of the software for research, data manipulation and thorough understanding of data. The diagram below shows an image of a graph plot with this particular configuration.

X – psnr000 (Num)

Y – pvqm000 (Num)

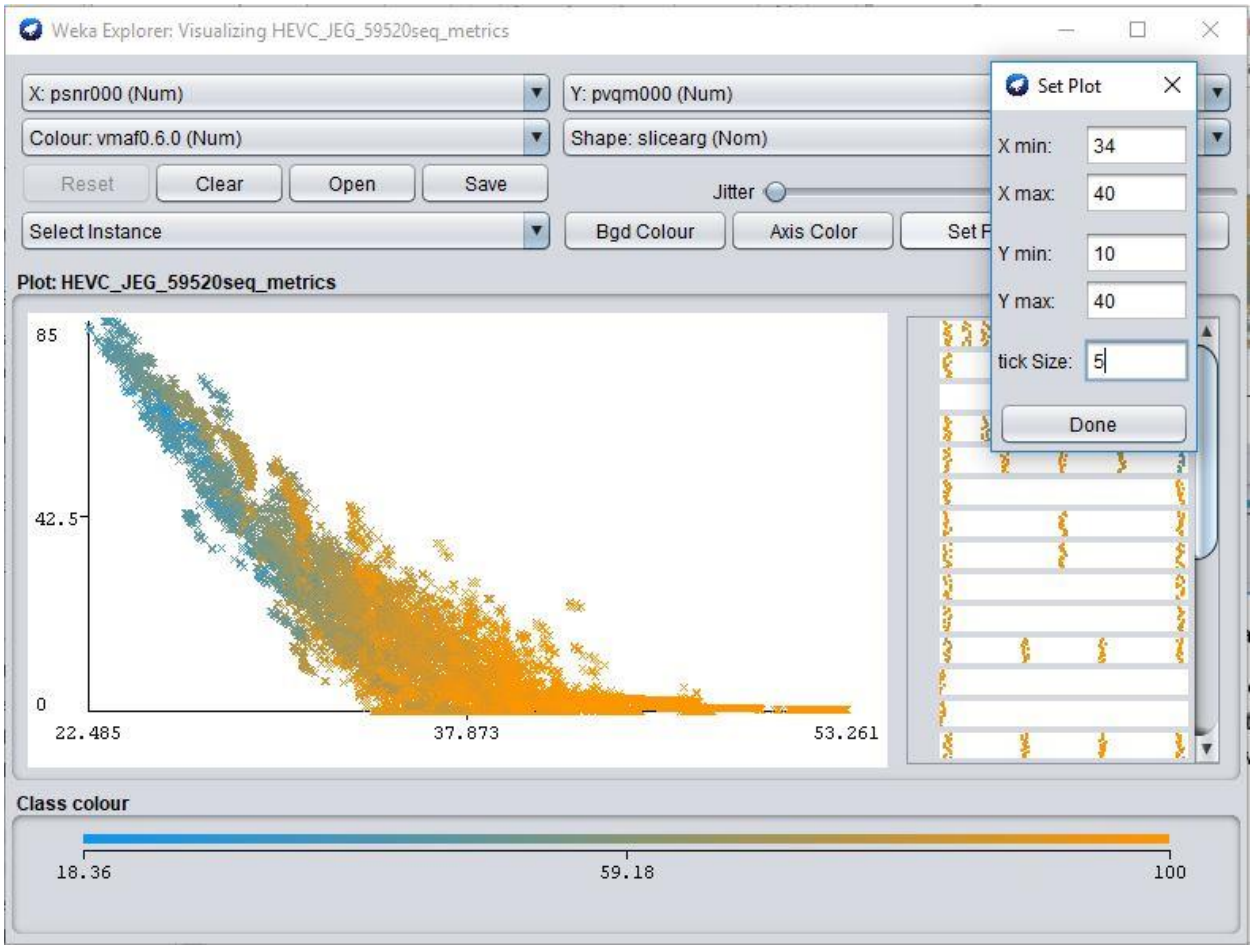


4a.5.1 Visualization Panel with an instance graph plot.

Looking at the image, we can see a dispersion of data, but the information is not so helpful, if we can zoom into it to understand the data more. It's a good thing we have a tool for doing that for us, which I would talk about later, but this tool is limited. With the addition of this feature we can specify exactly to which minimum and maximum values we want the graph plots to be displayed.

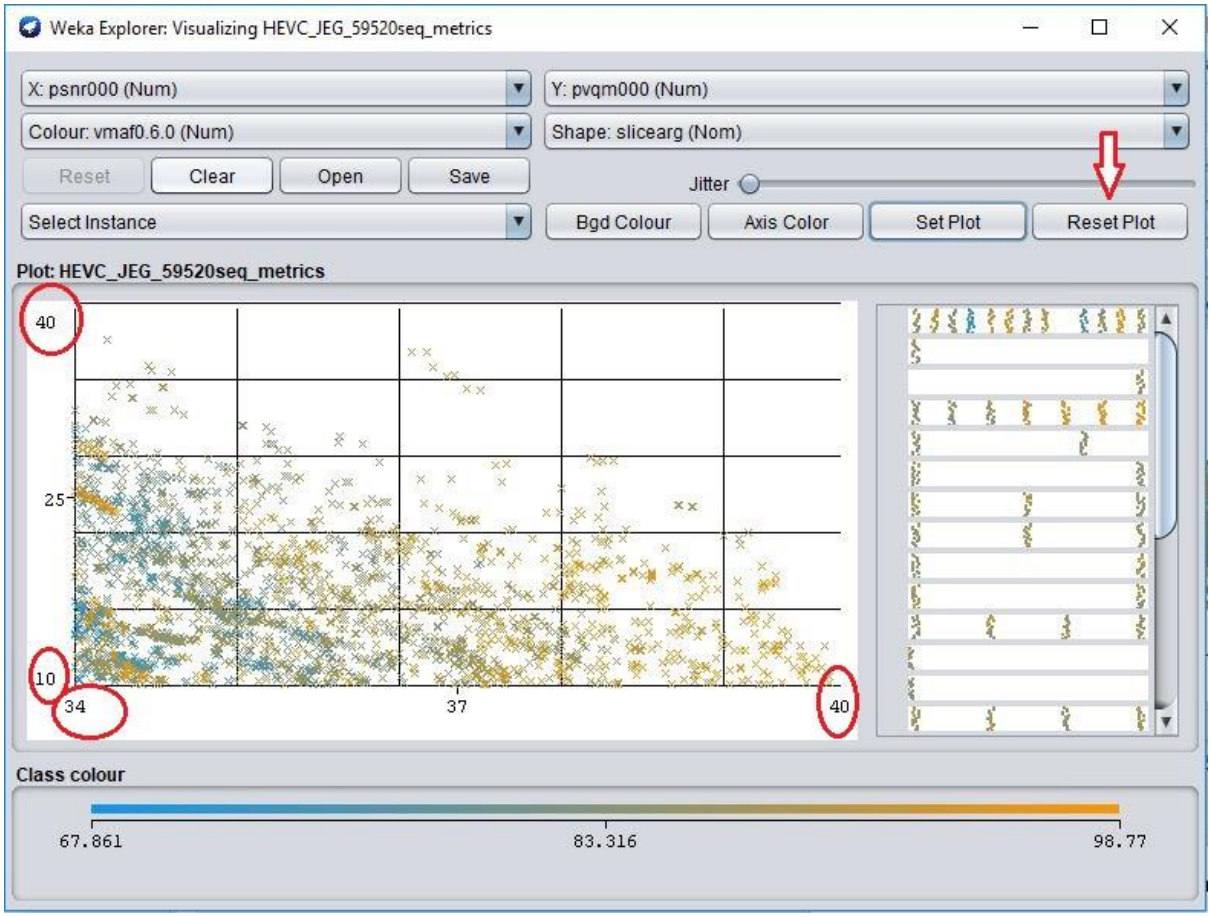
Let's assume we want to see only graph plots for X axis between 34 – 40 metrics, Y axis between 10 – 40 metrics, with a tick size of 5.

The tick size is just to divide the plot view into grids for an even better visualization.



4a.5.2 Visualization Panel with an instance graph plot with set plot button clicked.

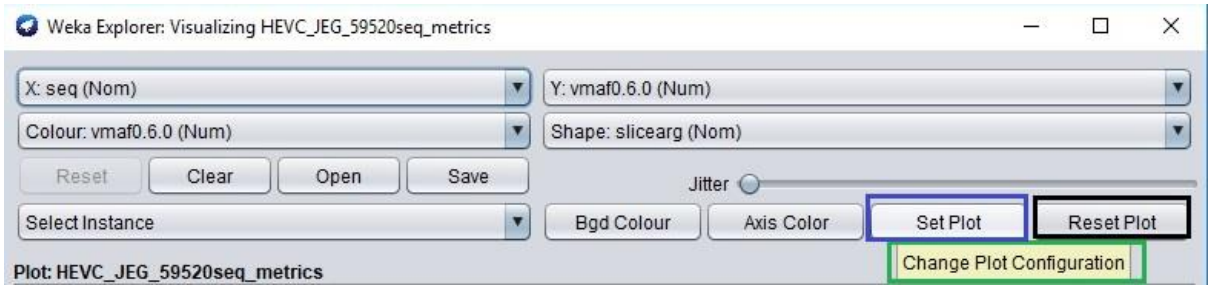
After the done button is clicked, the software engine, runs the application in the background and applies all the changes that have been set. And a zoomed-out plot is presented. An illustration of this is seen in the image below. Fig 4.5.3. Notice how sparsely dispersed the data points now are and how easier it is to understand the graph. We can also see from the graph that the minimum and maximum values for x and y axes circled in red colour are displayed just as was set in the set plot variables and the grid boxes in the plot are 5 in number. This plot ca further be set to even lesser values which mean that more precise access to data points can be gotten if need be.



4a.5.3 Visualization Panel with an instance graph plot zoomed out.

But for this illustration, I would explain how we can return to the original plot. To return to the original plot, we just must click on the reset plot button. Yes, its as easy as that clicking on this button takes every of the plot settings back to the way it was before the plots were set. This feature for sure makes this software a very handy tool in the hands of potential researchers and users.

Below is code changes and addition that were implemented for this feature to work. The additions were also made in the VisualizePanel.java and Plot2D.java classes discussed in Chapter 3.

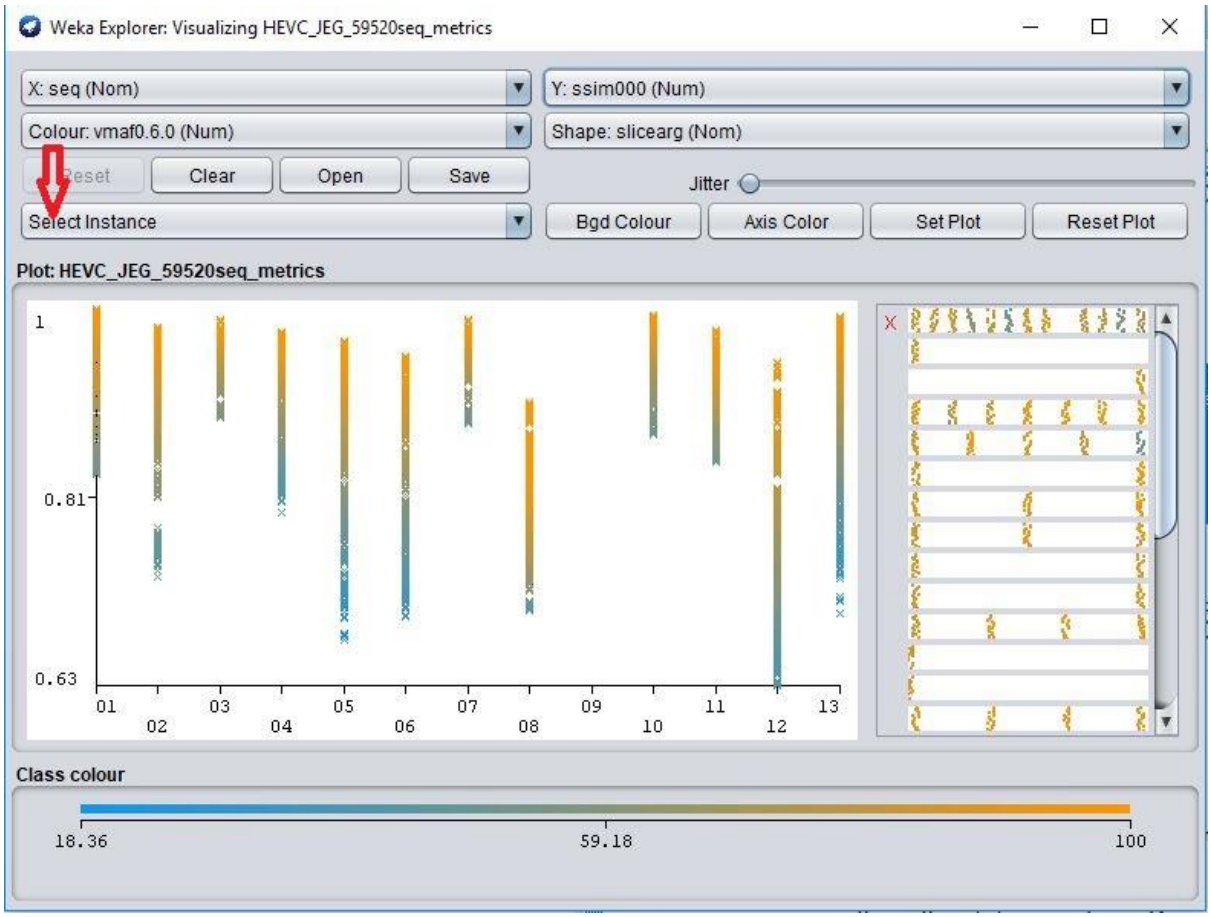


4a.5.4 Visualization Button Panel.

From the image above we can ser the set plot button highlighted in blue, the reset button highlighted in black and the tooltip text for set plot button highlighted in green.

6. REMEMBER the settings when changing visualization

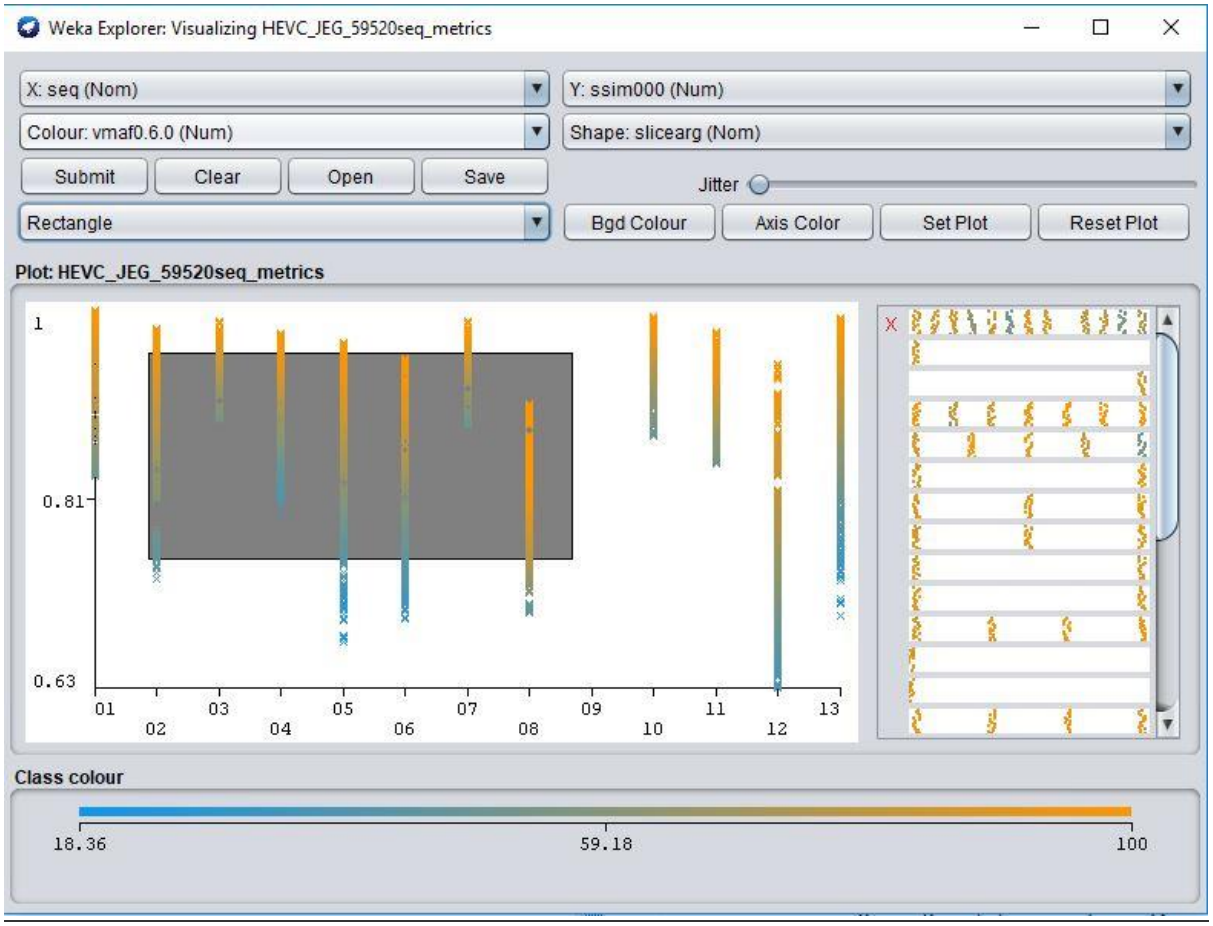
This feature was more like fixing a bug than added a whole new functionality. The added value to the user is like that of the feature described above. The main difference is that while the user can enter the exact points for x and y axes they want to inspect in the previous feature, for this feature they just must select the points they one to see zoom out directly on the graph. While this is very user friendly and handy, its down fall is that the user could not be able to enter the enter number for the maximum and minimum values of the x and y axes. This issue as said, has now been fixed and added already as a feature. But there is one problem though, when a portion of the graph is selected to be zoomed out, upon reset the graph returns to its original plot but does not remember all its set attributes.



4a.6.1 Visualization Button Panel with a graph plot instance.

This could be very worrying, most especially if the plot attributes and properties have taken time and expertise to set. Setting them again could be time consuming and could lead to mistakes. Fig 4.6.1 shows an image with a graph plot instance with the following properties:

X: seq (Nom), Y: ssim000 (Num), Colour: vmaf0.6 (Num), Shape: slicearg (Nom).



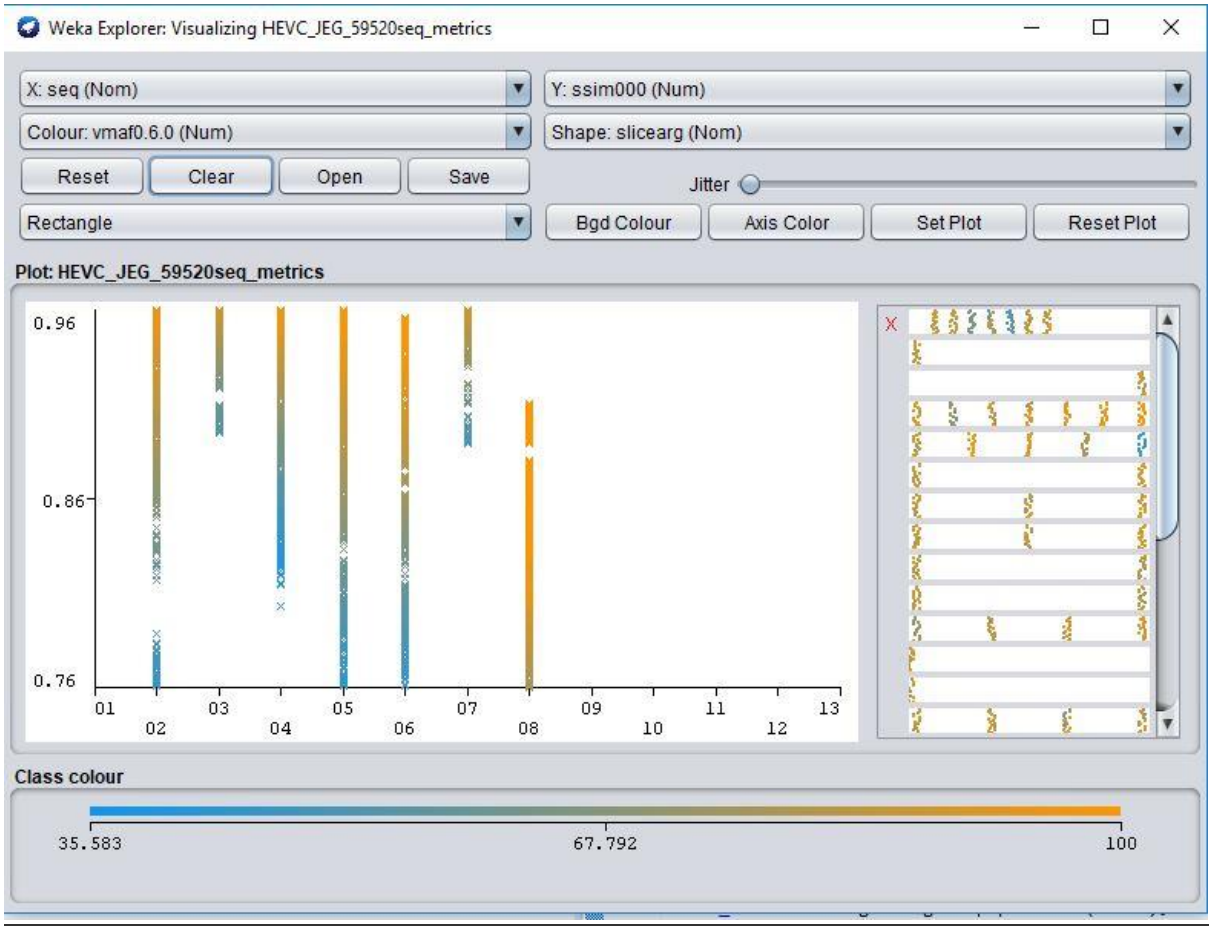
4a.6.2 Visualization Button Panel with a section highlighted.

Let try to zoom out a section of the graph so be able to read the data from this graph plot properly, to be able to do that, we must click on the select instance button that is shown under the red arrow in Fig 4.6.1. Then select Rectangle. This would enable the user to draw a rectangular shape on the section they plan to zoom out. In our scenario, a sample of this is given in Fig 4.6.2. Now in other for this section selection to be zoomed out and the graph data points plotted out again, the submit button must be Clicked. Clicking it would give the image below in Fig 4.6.3, you would see that Fig.3 is an exact replica of the highlighted section of Fig 4.6.2 only that is its maximised. Note also the buttons beside the submit button, which are clear, open and save buttons.

Clear – this would clear the highlighted section symbol and cancel the operation

Save – This would save the data values for the highlighted section in .arff format.

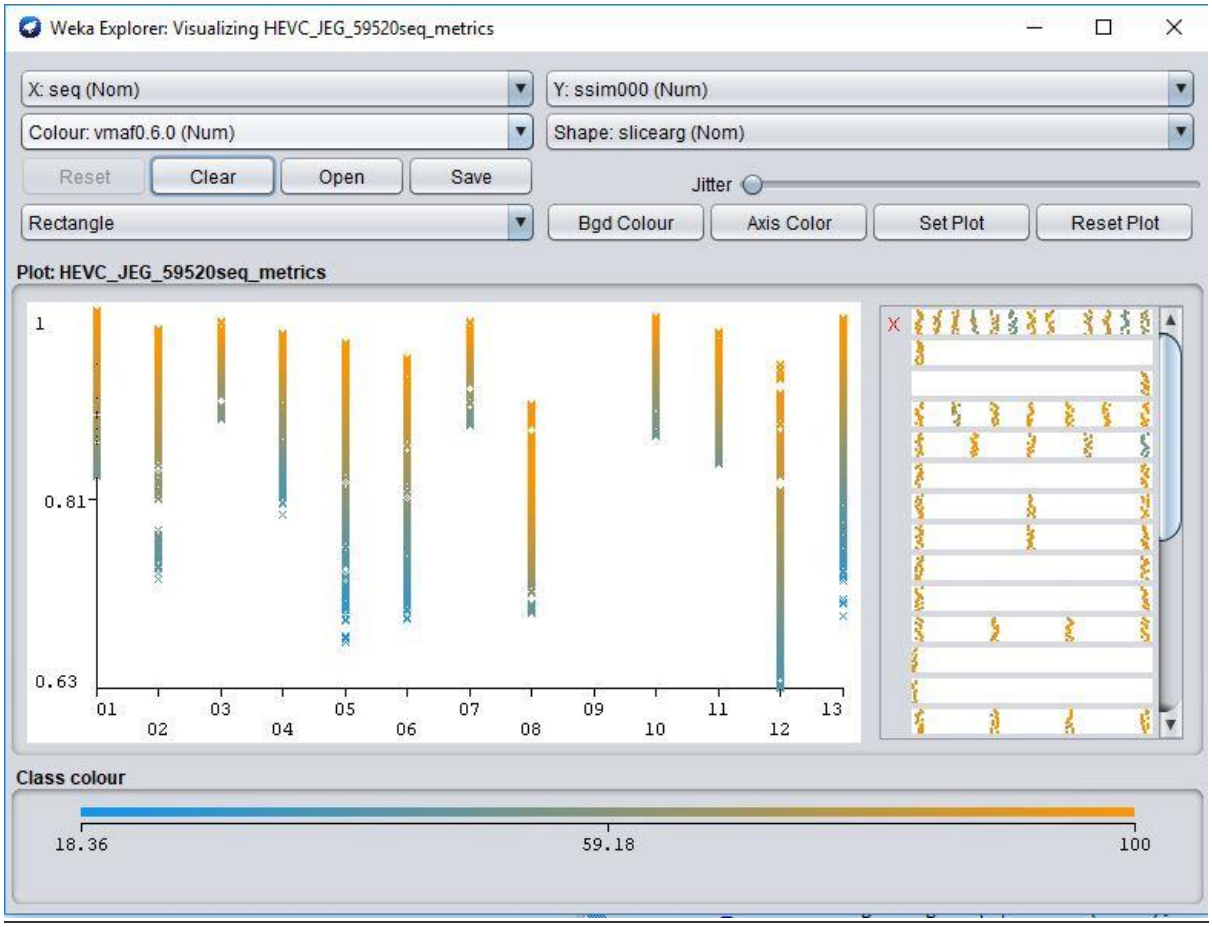
Open – This would file that have been saved with already highlighted section.



4a.6.3 Visualization Button Panel zoomed out.

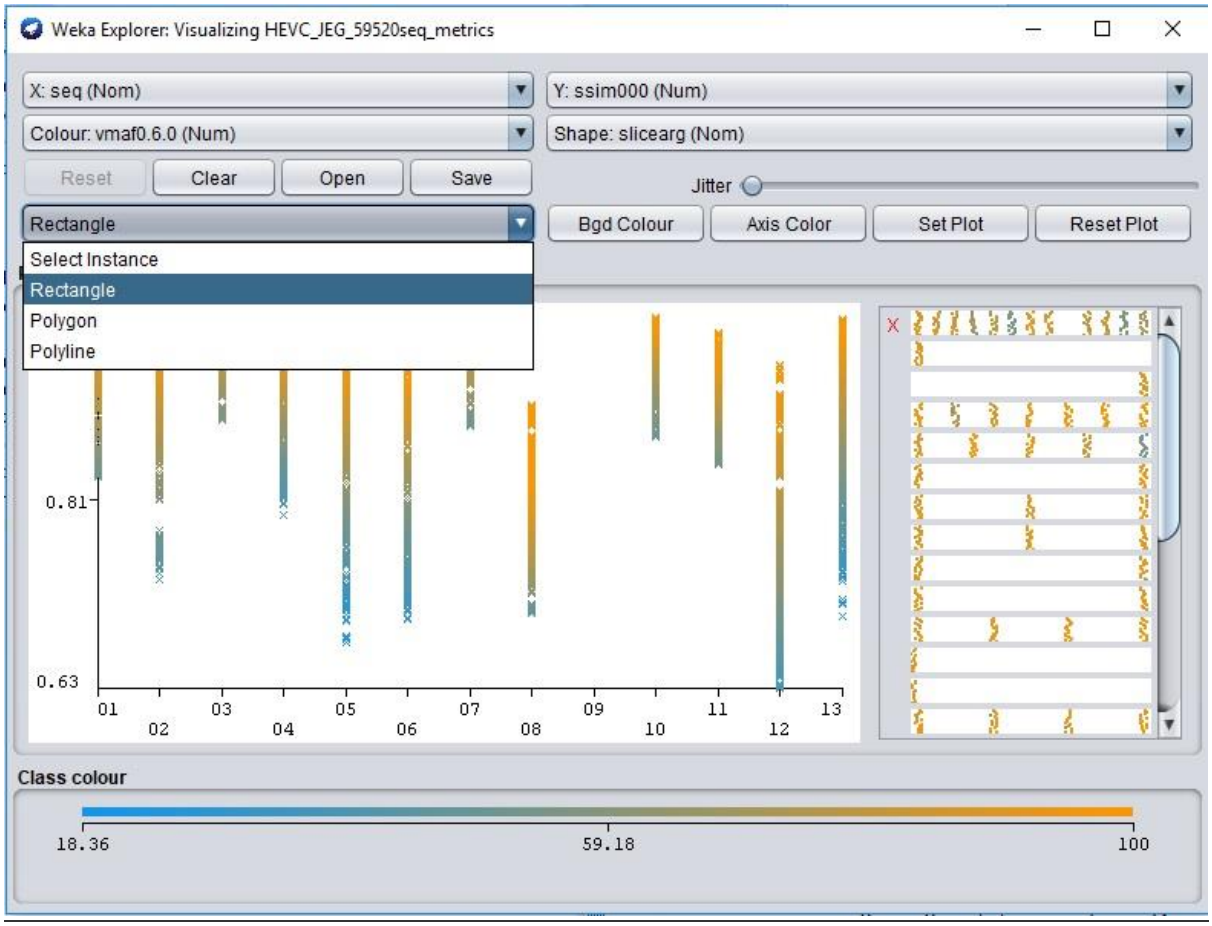
As said earlier, the problem with this cool feature was that the reset button was buggy. When the reset button is clicked, it resets the plot to its original form, but the colour attributes and shape attributes are no longer remembered. This has now been fixed and when the reset button is clicked all attributes and properties are reset correctly, including the colours and shapes.

The code changes for this one was a rather simple one, which was implemented in the Plot.java class found in the visualizePanel.java class as discussed in chapter 3.



4a.6.4 Visualization Button Panel.

One more important and interesting thing to know is that, when selecting an instance that would be used for zooming out the graph, rectangle is not the only option available. There are also other shapes that the user can choose from to enable him zoom out. The shapes available are Rectangle, Polygon and Polyline, like seen in the image below Fig 4.6.5 This option also make it easier for the user to choose the one they prefer the most and makes it easy to point pick certain areas for data manipulation.



4a.6.5 Visualization Button Panel with all instances

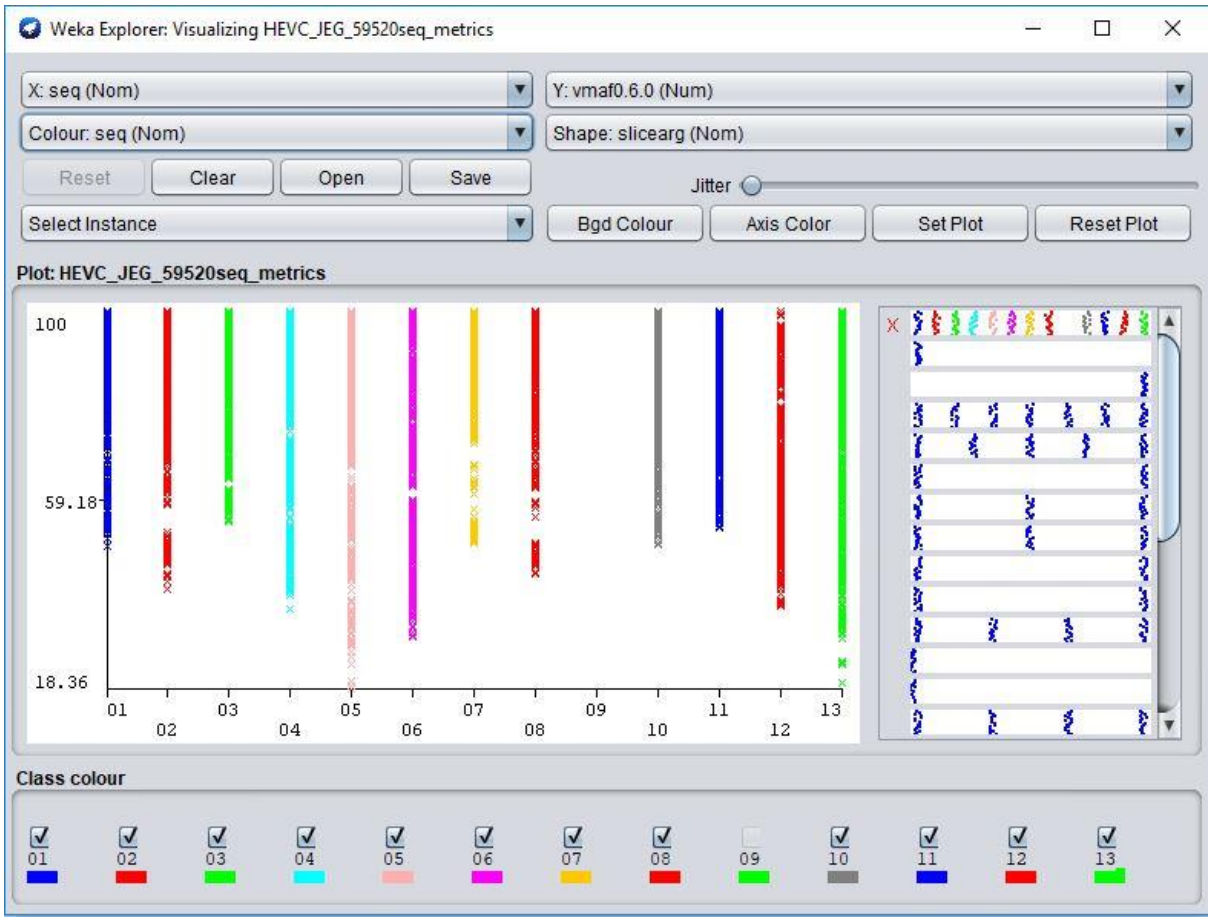
Dynamic feature value selection

This section is all about improving feature selections. When I say dynamic features, I mean the features that adds to the manipulative advantage of the software. These features make working around the software application easier and more fun. The best part of it is that specific part of the application can be selected for you at runtime. The strong point of this feature is the filtering of data points, not only in the visualize panel for users to see and use but also across all other panels and plugins. Below I would give a more detailed illustrations of the features and enhancement added to the software application, along with the code changes and addition made during development.

1. Filterable subset of data

This feature added the ability to be able to quickly filter out and "filter in" a subset of the data. This is a very useful feature as users of the software would love from time to time to filter in and out of the data they have. If a user for example, decides that certain data points are

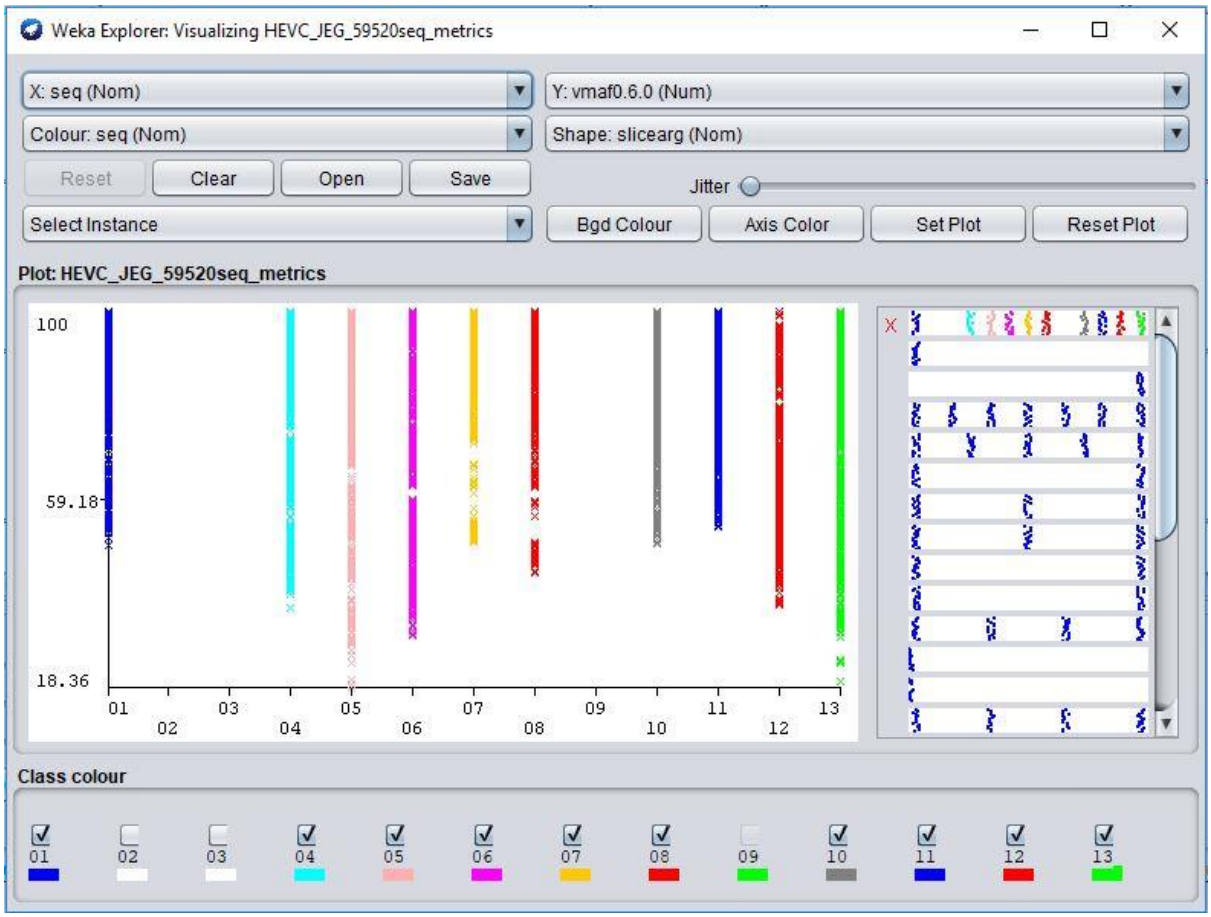
useless from the data points on the graph plot and decides that he wants to remove them temporarily or entirely from the plotted graph, this was impossible to be done because prior till now, a remedy did not exist. But thanks, to this feature, users can add and remove data points and/or attributes from the graph plots, just as easy and they can also replace it back to the way it was as far as they have not closed the window. The user now can easily filter the data sets depending as they want. It is useful to know that there already exists an in-built filtering option that is used to for filtering values in the PreprocessPanel where the file is first loaded, but apart from the fact that this filtering handles different data values, it is also limited in its use and can not be applied to the visualization panel.



4b.1.1 VisualizationPanel with all data sets

Fig4b.1.1 shows the application panel with a ploted graph of the following selections, X – seq(Nom), Y – vmaf0.6.0(Num), Colour – seq(Nom), Shape – slicearg(Nom). Notice how all the data points on the graph plot are displayed. The selection has 13 attributes, numbered 1 to 13. Also, the attribute 9 is missing as it does not exist. As we have discussed earlier, colour selection, shape representation are very good ways to manipulate and investigate the data

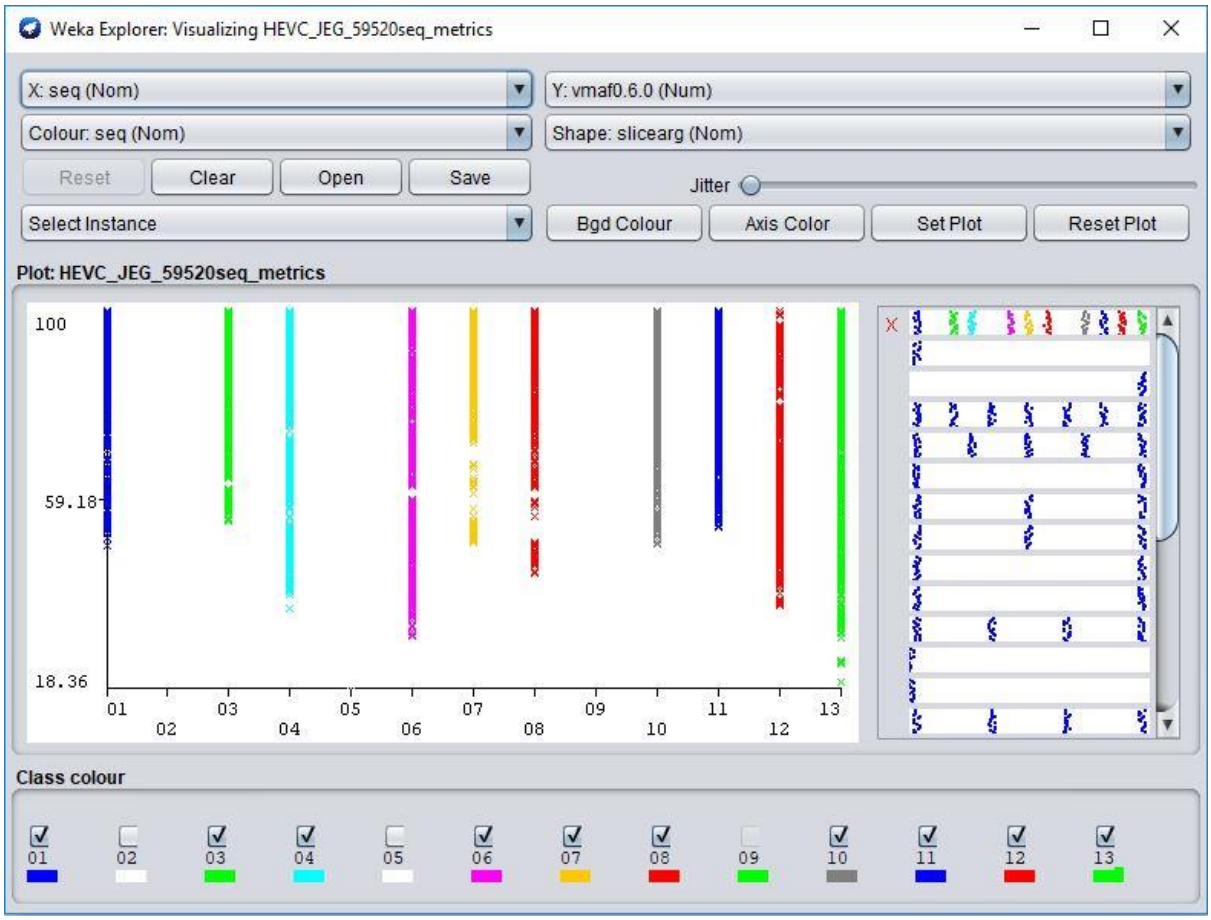
points, but they do not help us if we are only concerned about some data points. It can then further be argued that the zoom in/out features, or the maximization/minimization axis feature can also be used to fulfil this request, but while this is true, we have to bear in mind that if the user is trying to exclude some attributes this is not an effective way either. Filtering would make it easier to select off the exact attribute that the user does not want.



4b.1.2 VisualizationPanel with attributes 2 and 3 removed from data set attributes

For example, if we want to remove an attribute from a data plots, all we have to do is to unselect the checkbox to remove the attribute and select it to put it back, as far as the attribute is a nominal attribute. When the attribute is unselected, the colour of the attribute label automatically becomes white. In the image above, we want to remove data sets from the graph plot of attribute 2 and 3. This is possible by clicking on the checkbox corresponding to the number and colour of the concerned attribute to unselect. the image in fig4b.1.2, notice how easy this is done just with a click and it is immediately reflected on the graph and the plot is automatically replotted. Another interesting thing is that it is also possible to selected random attribute number from the set of data points given. We can choose to filter off the first and ninth

attributes, just for illustration. In the image below (Fig4b1.3), we would notice that attribute number 2 and 5 are filtered out, while attribute number 3 is reselected again. This like the previous example automatically resets the plot and resets the attributes the way they would now be in their desired positions.

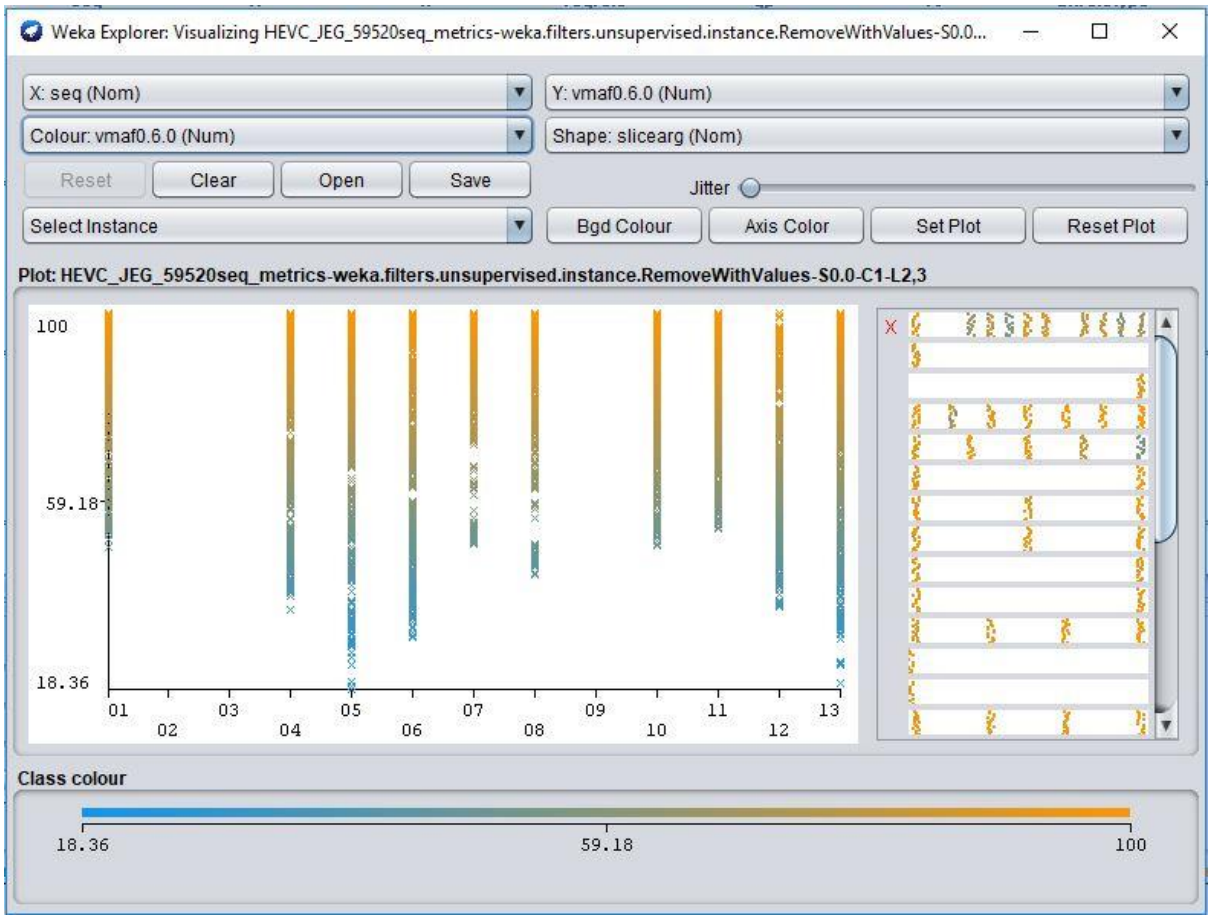


4b.1.3 VisualizationPanel with attributes 2 and 5 removed from, 3 added to data set attributes

The selection and deselection basically work the same way and no matter the number of attributes be select to be filtered off, their attribute would automatically be filtered out. Also, when we reselect them, they would appear again on the graph plot. This makes certain data analysis very easy to manage and to understand. It also makes the users life easy in terms of implementation, if he/she had a lot of attributes, say 100, all they have to do is to select the attributes they want, and it would automatically be filtered in/out of the graph plot.

The interesting thing about this filtered data sets is that they are applicable across all the instances. When the desired selection has been made as a nominal value, even if the colour combo box is switched to that of a numeric selection, it would still reflect that some data has

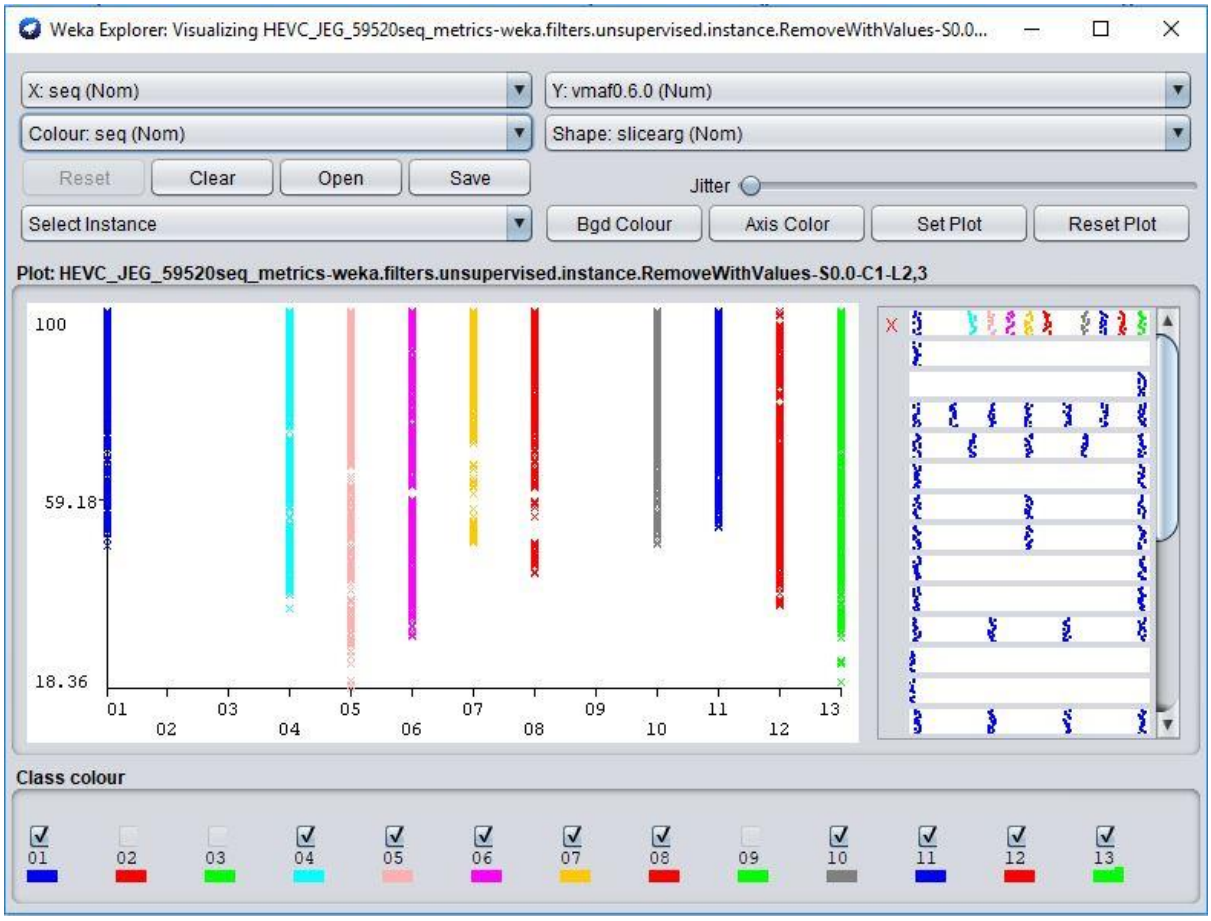
been filtered out. Let's consider Fig 4b1.4, this image is based on the selection from Fig 4b1.2, even though the filtering was done when colour was selected as a nominal value, the data sets still are represented correctly, when another colour attribute is selected. This is the same as well for the shape representation via the combo box. As far as the selection for X and Y in their combo boxes are the same.



4b.1.4 VisualizationPanel displaying to numeric data set with filtered attributes

Now when all the appropriate selection is made, and the user decides to close their visualization panel window, something interesting happens. The state of the filtered attribute values is remembered upon closure. This way the user can pick up from exactly where they left off, and they do not need to reset and prepare their visualization panel again for the data they are working on. This comes in also, very handy because it relieves the stress of trying to remember where you left off and just automatically populated only the filtered in values. When the Visualization panel window is opened again, the checkboxes for the already filtered out attributes are become greyed out and their data plots values are also filtered out just as expected. Let's look at the image below. Fig4b.1.5, you would be able to observe that the attribute values

for attributes 2 and 3 are greyed out, just like the attribute number 9. This is because there are no data plots for this attribute numbers as we can see in the graph. Also, reopened a visualization panel window after it has been filtered automatically produces this effect. For easy remembrance of data. It is important to note that the user should now exist the current window if they do not choose to go on with the present filtered values they have, this is because they would lose all the filtered-out data and would to depend on a new instance of the filtered in data. The user can always reload the file again if they need the data they have already filtered out.



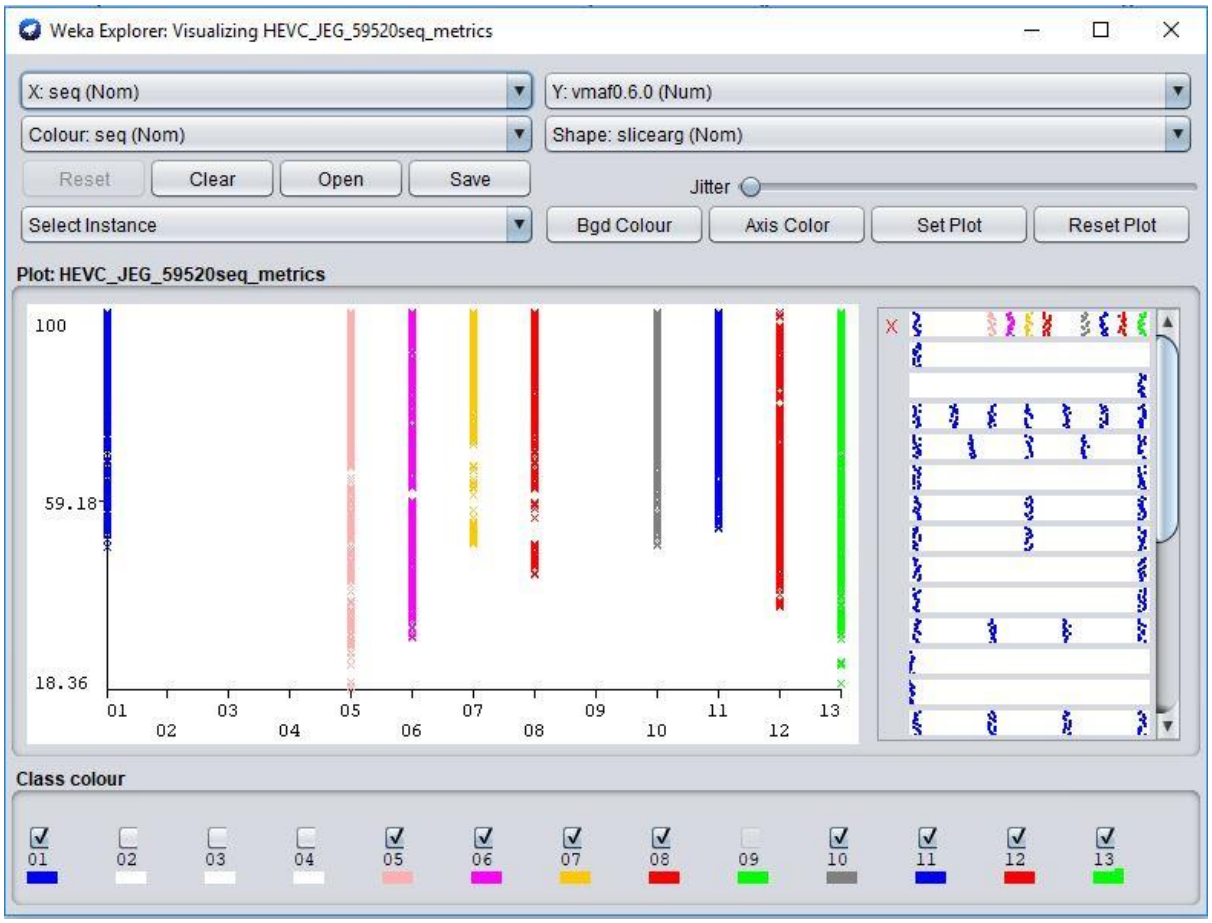
4b.1.5 VisualizationPanel remembers data plots after reopened.

code changes added to the existing software application to make the feature possible are found in the ClassPanel.java class and have been discussed in chapter 3.

2. Filtered data reflected in plugins

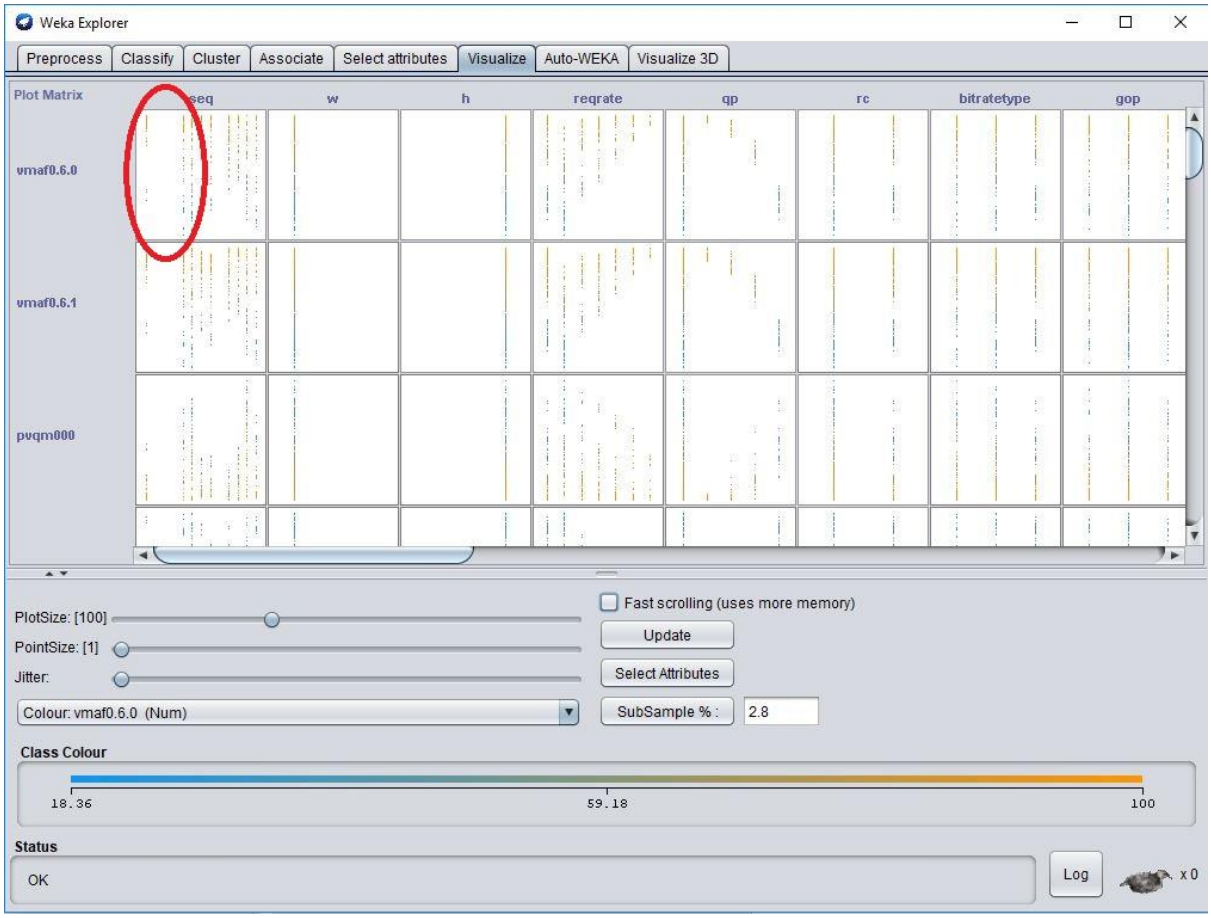
So far, we have been talking about filtering data and also how the addition of this feature has greatly fostered this application and can be of great help to other potential users and

researchers. But this is not all to this feature. We have seen how selecting check boxes can filter and unfilter values from a data plot. Let's take a look at the image below.



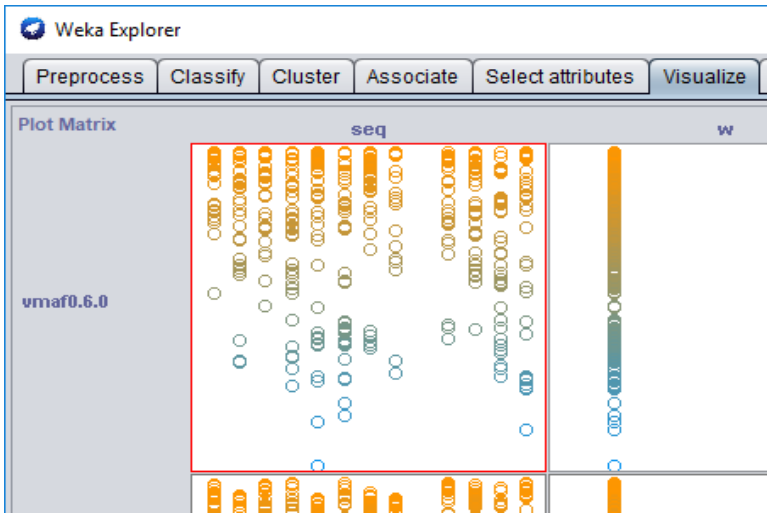
4b.2.1 VisualizationPanel with attributes 2, 3 and 4 removed from data set attributes

In this image, we can see that the attribute numbers for 2, 3 and 4 have been unselected, hence the reason the data plots are missing as well. Let's assume that this is the way I intend my data sets values to look like and I would like this to be applied throughout the application so that I can easily use it in another panel. All we must do is just to click on the exit button and close the window. It automatically gets saved, this is of course taking into consideration that this is the final state of the sets as explained above. Once the window is closed, we are left with the matrix panel. And this panel should automatically reflect the changes that have occurred in the visualization panel. Let's take a look at the image in Fig4b.2.2 for a perfect illustration, we would notice that the graph plots of the matrix panel have been changed when compared to its original version (a closer screen shot of the 2 version are available below for clarity). This confirms that was discussed in the previous feature. The plot instances of our new data sets have been saved.

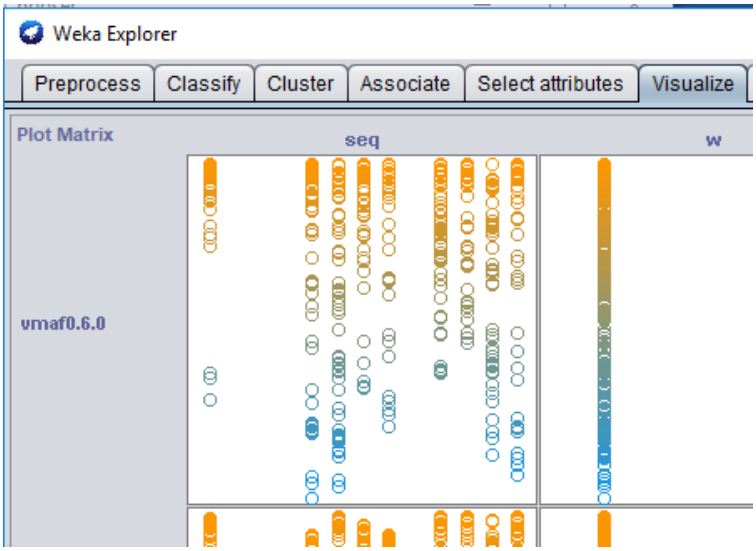


4b.2.2 Matrix Panel with Filtered out attributes

The area concerned, i.e. that has the filtered-out values have been circled in red in the image above.

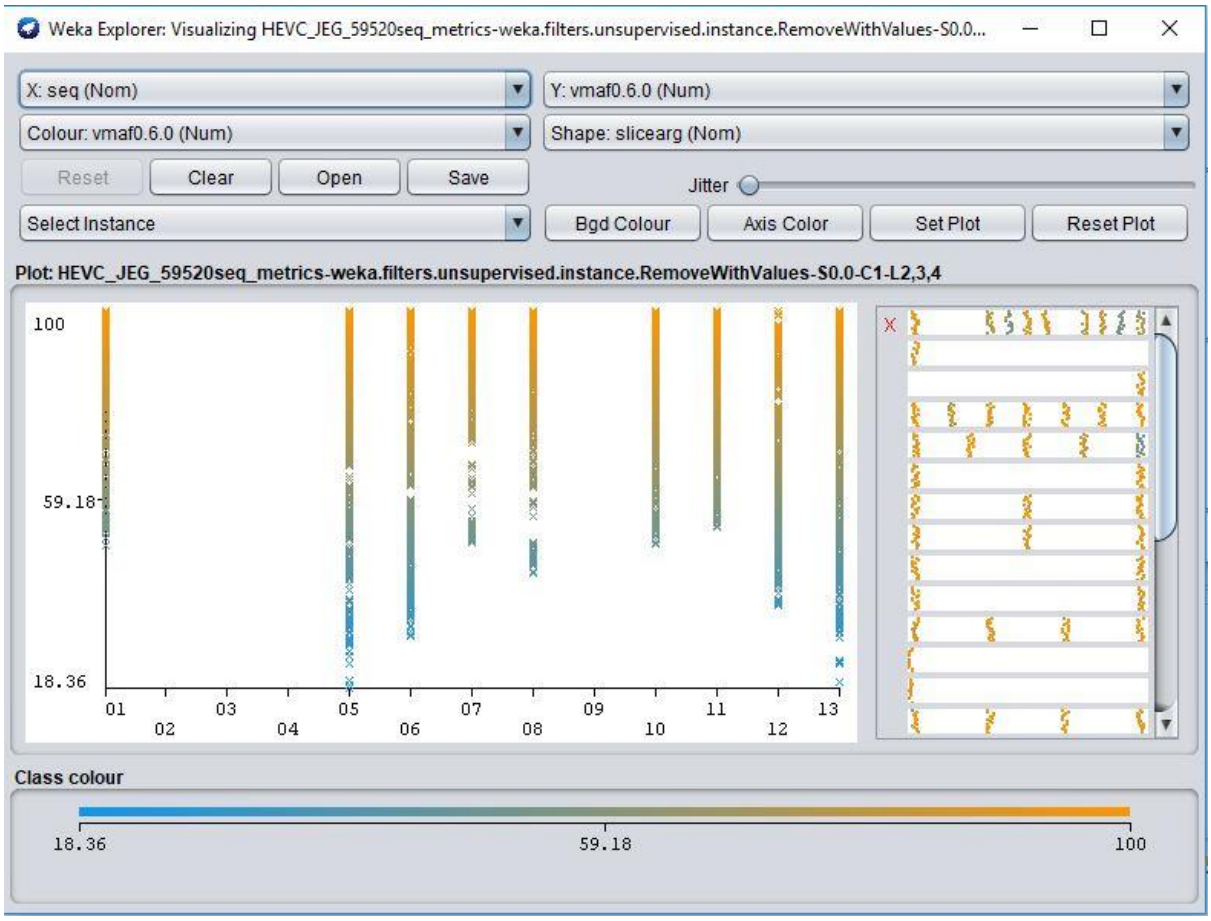


Original plot



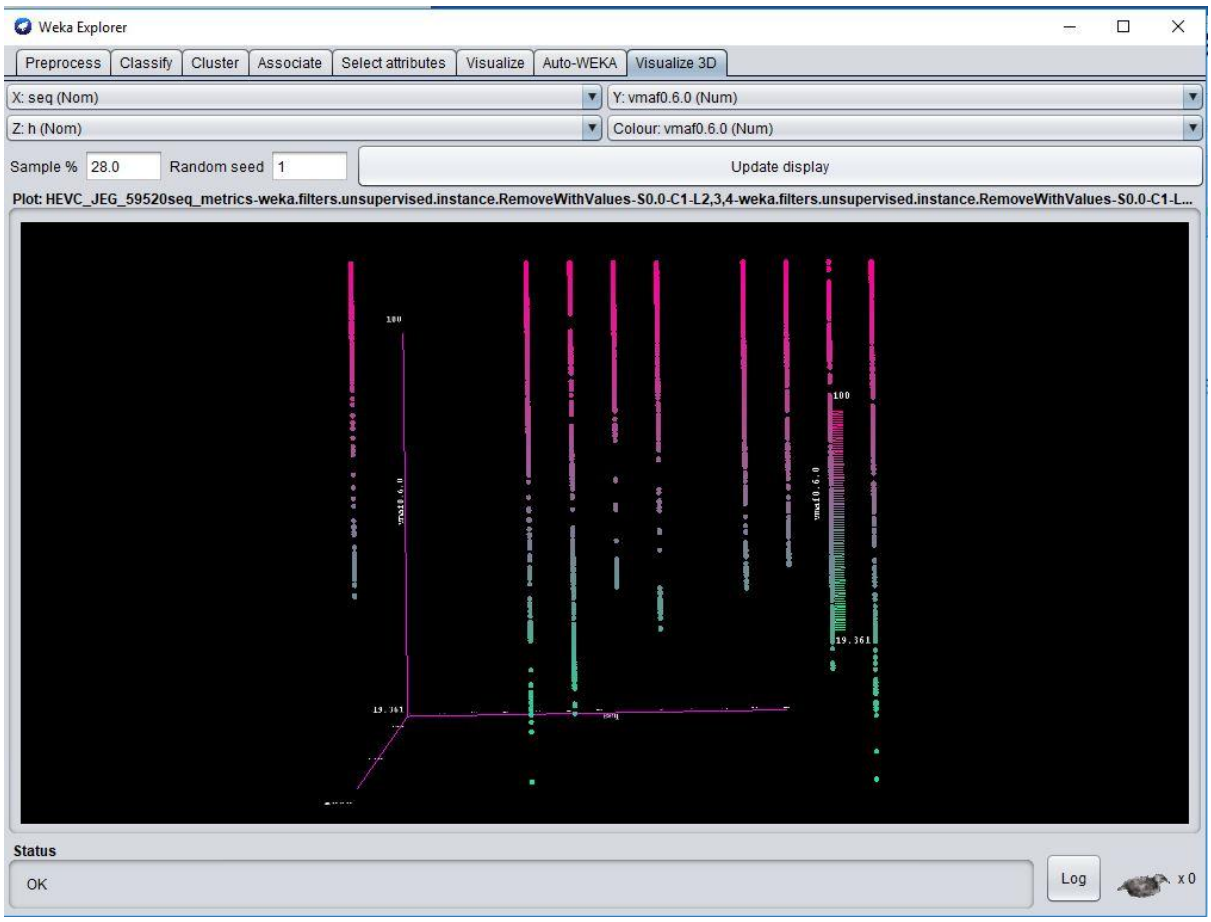
Newly filtered plot.

You would be able to know the difference in the newly filtered plot, that some data points have been removed.



4b.2.3 Visualize Panel with filtered out data sets

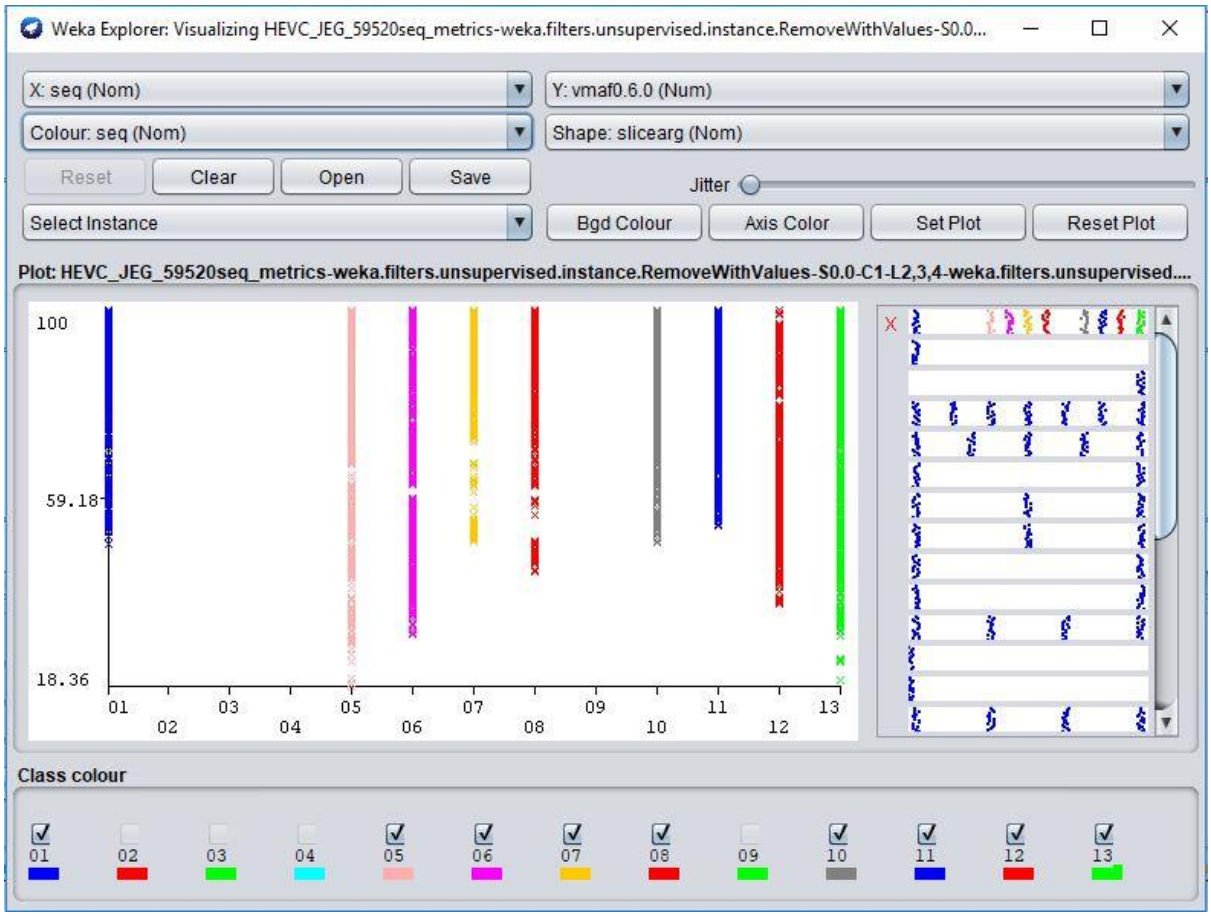
A very interesting thing happens now when we go back to the visualization panel by clicking on the Newly filtered plot. The plot values are still retained. It remembers which values have been filtered out and it does not show this value. Fig 4b.2.3 is a perfect example. Another interesting thing is the ability for the changes made to be reflected across platforms other than just panel. The filterable datasets also reflect in plugins as well, for example 3D plugins. This is known as the 3D visualization panel.



4b.2.4 Visualize 3D panel with filtered data set.

After filtering of data points has been done in the visualization panel, the window is closed, and the user clicks on visualize 3D panel, which is not a pre-installed panel and has to be installed via plugins found in the tool menu, the exact filtered out changes now reflects as well on the 3D plugins. This gives the advantage of also been able to see our set of newly filtered out data in 3D view, giving us the same existent advantage to inspect every part of it as a 3D model. A good example of this can be seen in the image above Fig4b.2.4. as we can see the same filtered data plots are now present in 3D form and we can see that they are properly filtered and can be used for investigation and easy data analysis. Also, all the

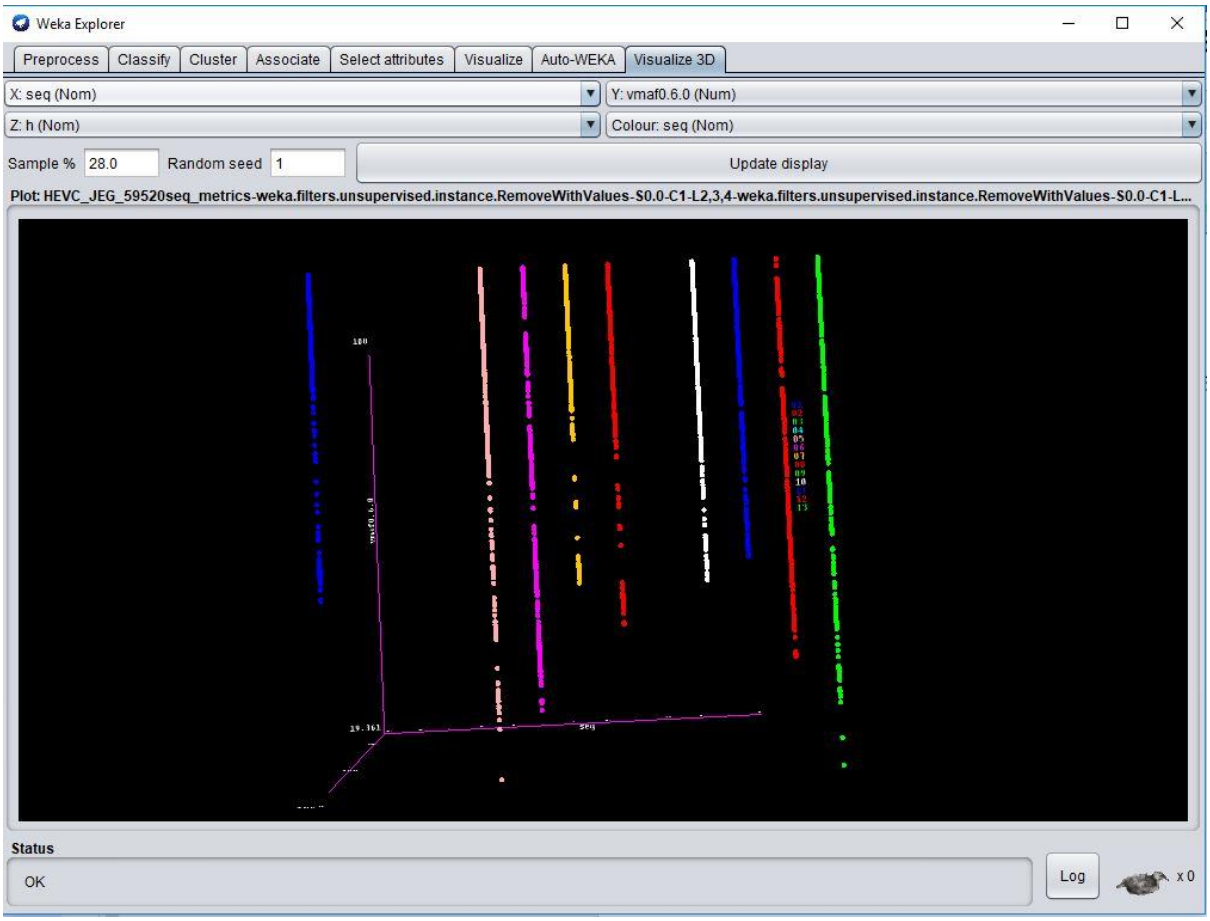
settings in this panel works as usual and when a change is made in this panel, the Update display button applies these changes.



4b.2.5 Visualize panel with filtered data set as nominal.

So far for this feature, we have been talking about filtered data sets across platforms and across panels, but just for numeric values, but it is astonishing to know that this feature also applies to nominal values as well. In the image above Fig4b.2.5, we can see that this is now a nominal attribute and the colour combo box now has “seq (Nom)”, selected as the colour. Automatically converting it to a nominal value. We can also see that the filter properties work as anticipated with no problem. The filtered-out attribute numbers have filtered out checkboxes and their data sets are filtered out as well, with no inherent bugs or problem. Lastly, if we close the window and open the Visualize 3D panel, we would see that the attributes are reflected exactly as they should again in 3D format. With the colour attributes selected as appropriate and with the right instances been filtered out. Below in Fig4b.2.6, we would see an illustration of what has been explained so far and how exact the anticipated properties are replicated. It is also good to know that in other to zoom in

and out of this 3D graph plot is it enough to place your mouse on the screen and scroll in or out of it.



4b.2.6 Visualize 3D panel with filtered nominal data set.

Code changes that was added to the existing software application to make the feature possible were done in the ClassPanel.java, MatrixPanel.java and the PreprocessPanel.java classes, they have also to discussed ore elaborately in chapter 3.

Richer Graph Plots

In this section I am going to speak about and show features that mainly affects the cosmetic aspect of the software application. I have spoken about the user interface, how the background colour and axis colour can now be changed, how attributes can now be arranged and analysed with respect to shapes and not only colours, how the class panel has been rearranged for a better user experience. I have spoken as well about how the ability to filter data has been added to the application. In this section, we would be exploring richer graph plot capabilities. i.e. majorly

the features that I would discuss here would be just like by-products of the previous features I have discussed earlier. A combination of all the other features produce a more powerful application with more interesting feature outreach that we would be exploring.

1. Data points on graph can be filtered by colour and shapes

One of the things that makes the graph plot for the W.E.K.A application richer is that data points can now be filtered by colour and by shape. When I say filtered, I am not referring to filtered out but to data attributes been distinguished. Prior to new, this separation and distinguishing was only possible through colours, but a very unsettling way for the users. Most times it was difficult for the user to understand how to use the application properly, or to understand it due to lack of certain features. Take a look for example at the image below

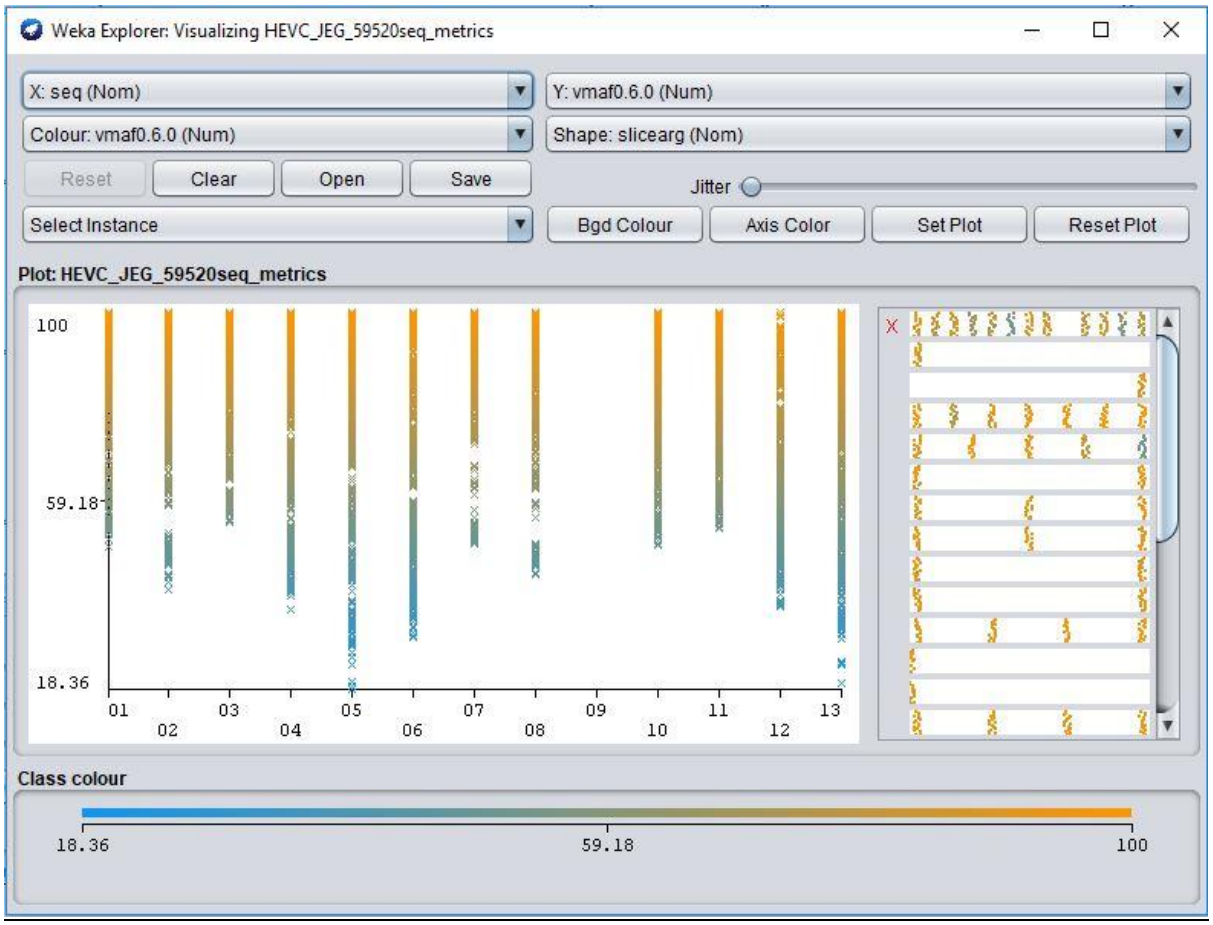


Fig4c.1.1 Visualize Panel

This is a visualization panel with numerical values plotted again nominal values. The colour of the graph is set to numerical as well hence for the way we see the graph now. When the colour is changed to a nominal value as in Fig4c.1.2 as we can see below, we would observe

that the data plots are now separated by attributes that can be easily distinguishable by their colour. But this is a feature that was already existent in the application.

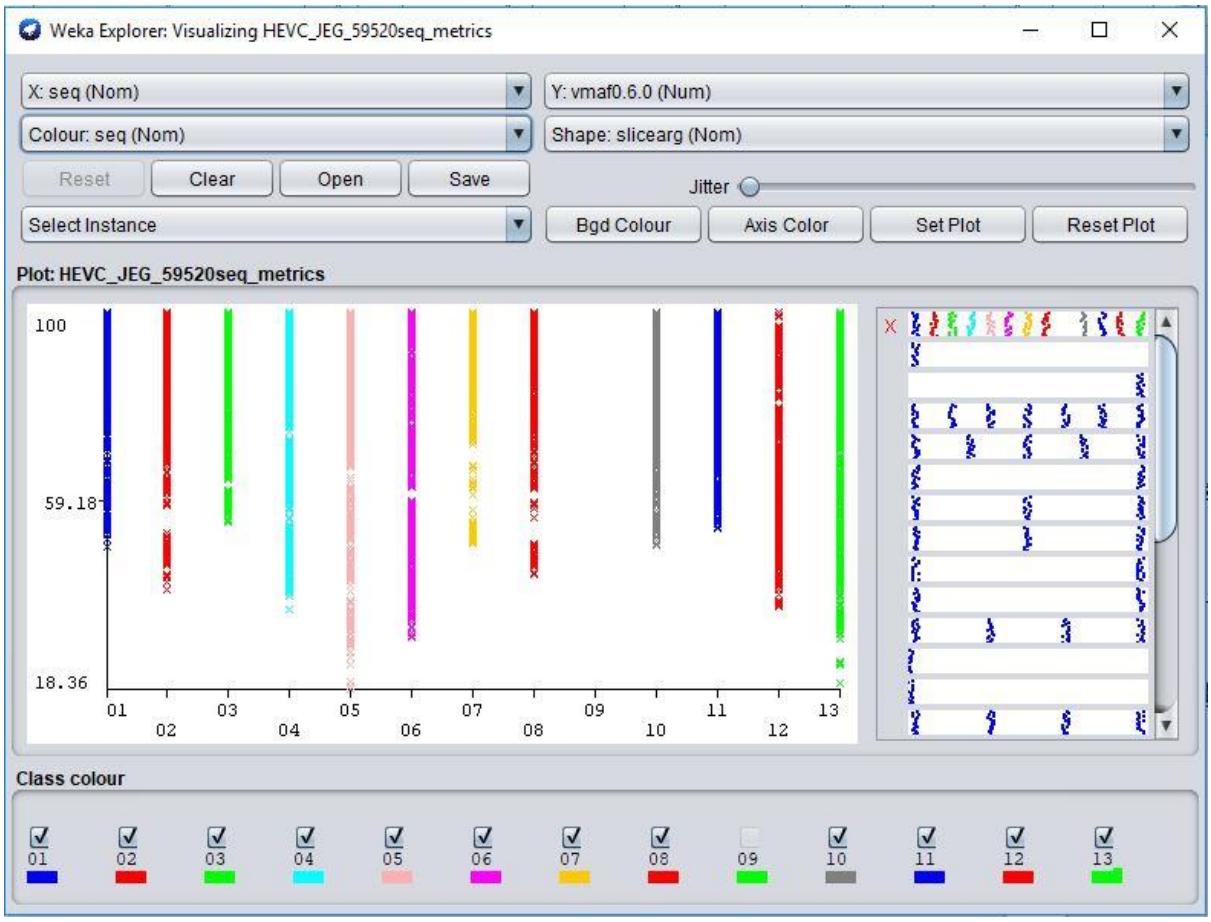


Fig4c.1.2 Visualize Panel with nominal colour attributes

The feature that was added now to W.E.K.A application makes the graph plot also possible to be distinguishable by shapes. For example, under the same instance as the previous diagram, with all settings remaining the same, lets select the shape attribute which has now been added as a new feature to be “bitratetype (Nom)” as is in the case of Fig 4c.1.3. We will notice that the data plots on the graphs have different shapes and not just a single shape “x”. Also, we would notice that the shapes are dispersed as in the bitratetype. Some of the attributes have different shapes within them. Let’s consider attribute number 8 for example. In this attribute, we have the plus shape representation, we have the cross-shape representation, we have the up-triangle shape representation and we have the diamond shape representation. It is a clear fact that this separation of attributes by shapes is not so clear when compared to if they are separated by colour, but it is good to know that a combination of these two features gives a very good user experience. And, sometimes the user might still prefer to use the shape

representation even, so an added option is always a good thing and another reason for the user to enjoy using the W.E.K.A application. Just for clarity again, I would zoom out a portion of the graph plot so that it can be clearer and more visible. I would analyse the data points between attribute number 3 and number 7.

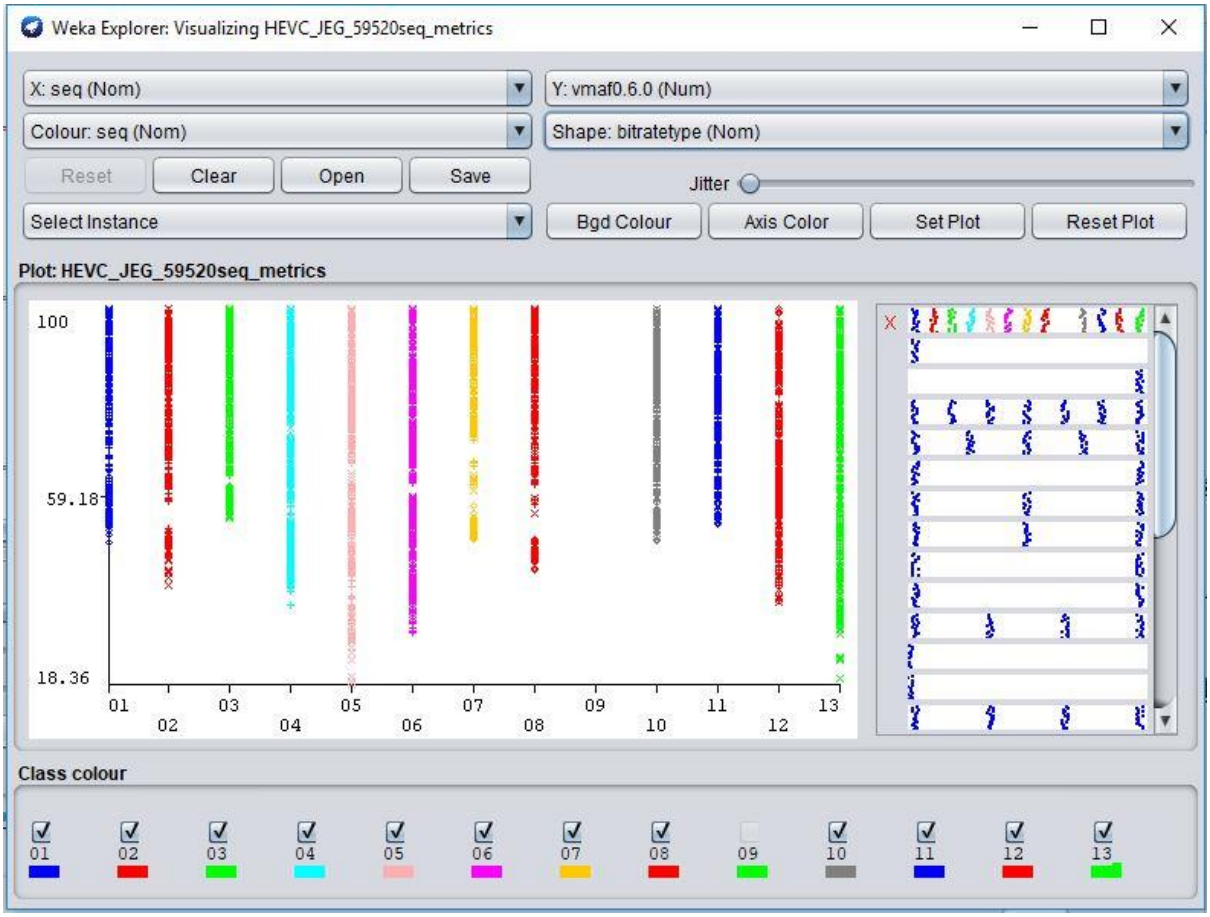


Fig4c.1.3 Visualize Panel with nominal shape attributes

An already zoom out image of this graph can be seen in Fig4c.1.4, we would see that the data points attribute number 3 to 7 are have been zoomed out and can be seen in a clearer way. This was possible from the select instance combo box that can be found above the data plot. The rectangle instance is used to get a decent representation for attributes number 3 to 7. Another thing we would observe thing is that maximized data plot is only along the y axis, while the x axis remains the same. It does not change while the points in the y axis modulate according to the image drawn via the selected instance, this is because the x axis has a nominal attribute selected, but this limitation is resolved from the minimize and maximise feature. We would also be able to able to see the shape representation for each attribute clearer and more distinctively. Notice how the checkboxes selected only represent the attribute data points displayed in the graph. This was it is not misleading to the user of the software application, since all other attributes have been temporarily filtered out. Also, on the selection of this

instance, the colours of each attributes are retained correctly, just as they were before the instance was selected. And the program does not crash or develop abnormalities.

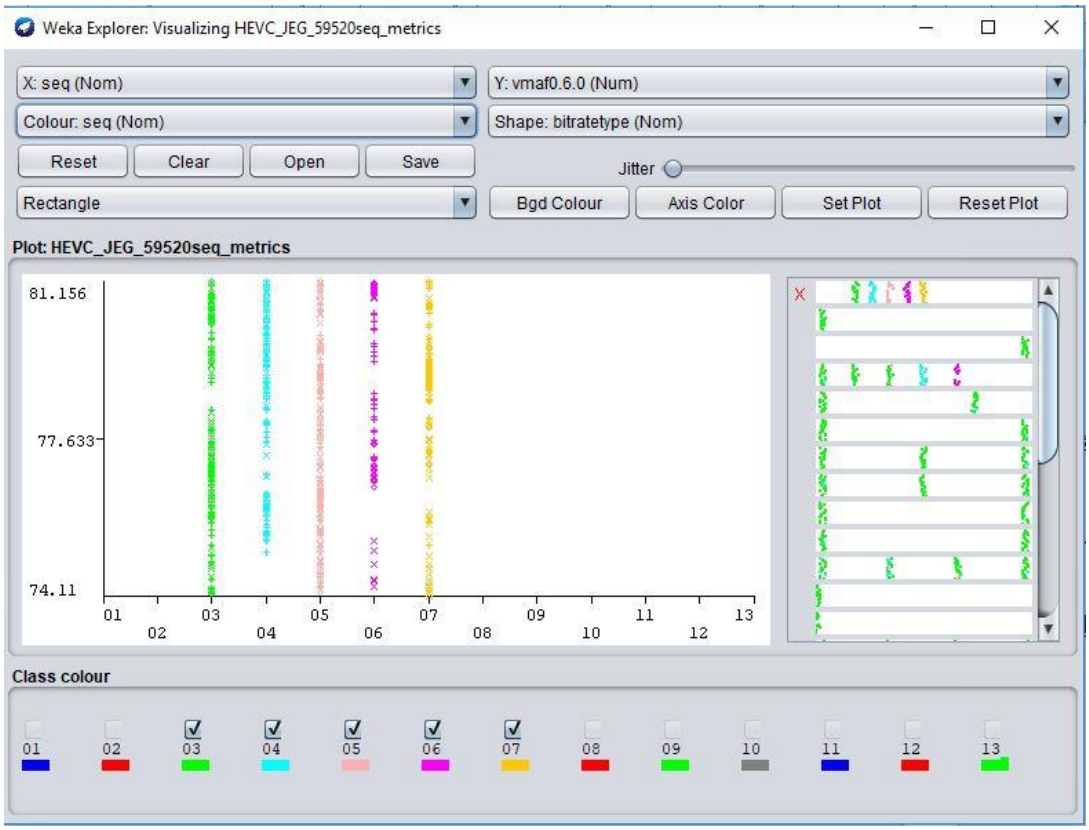


Fig4c.1.4 Visualize Panel with nominal shape attributes zoomed out

Another good example of shape representation can be found in the example below in Fig4c.1.5 and 6.

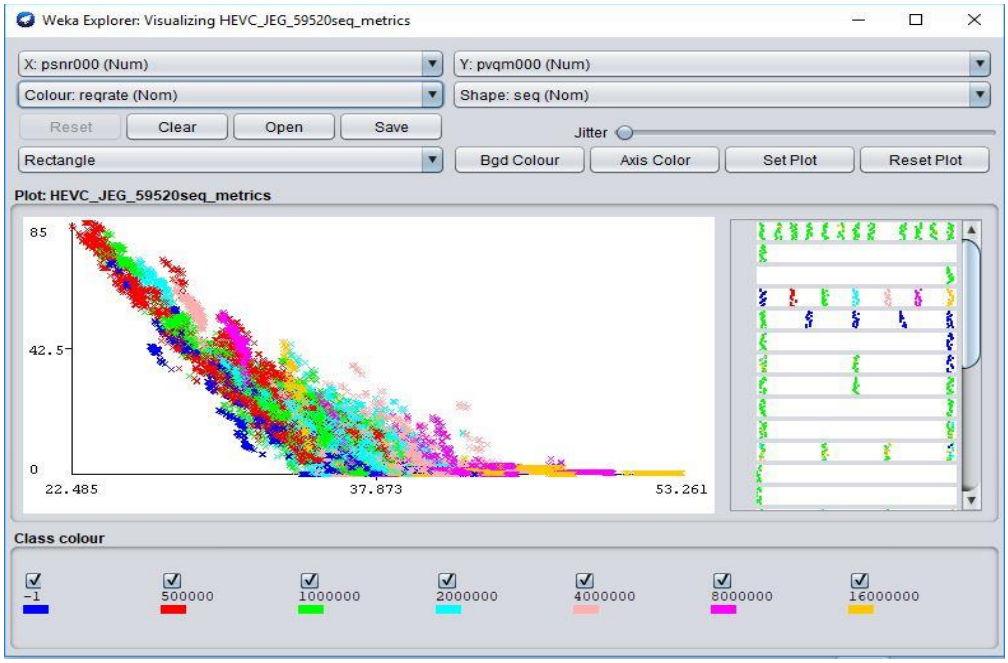


Fig4c.1.5 Visualize Panel with nominal shape attributes zoomed in

Looking, the data point plot of which the colour attributes are x – pnsr000(Num), y – pvqm000(Num), colour – reqrate(Nom), shape – seq(Nom), we would see the advantage and beauty of have the shape attribute feature implemented.

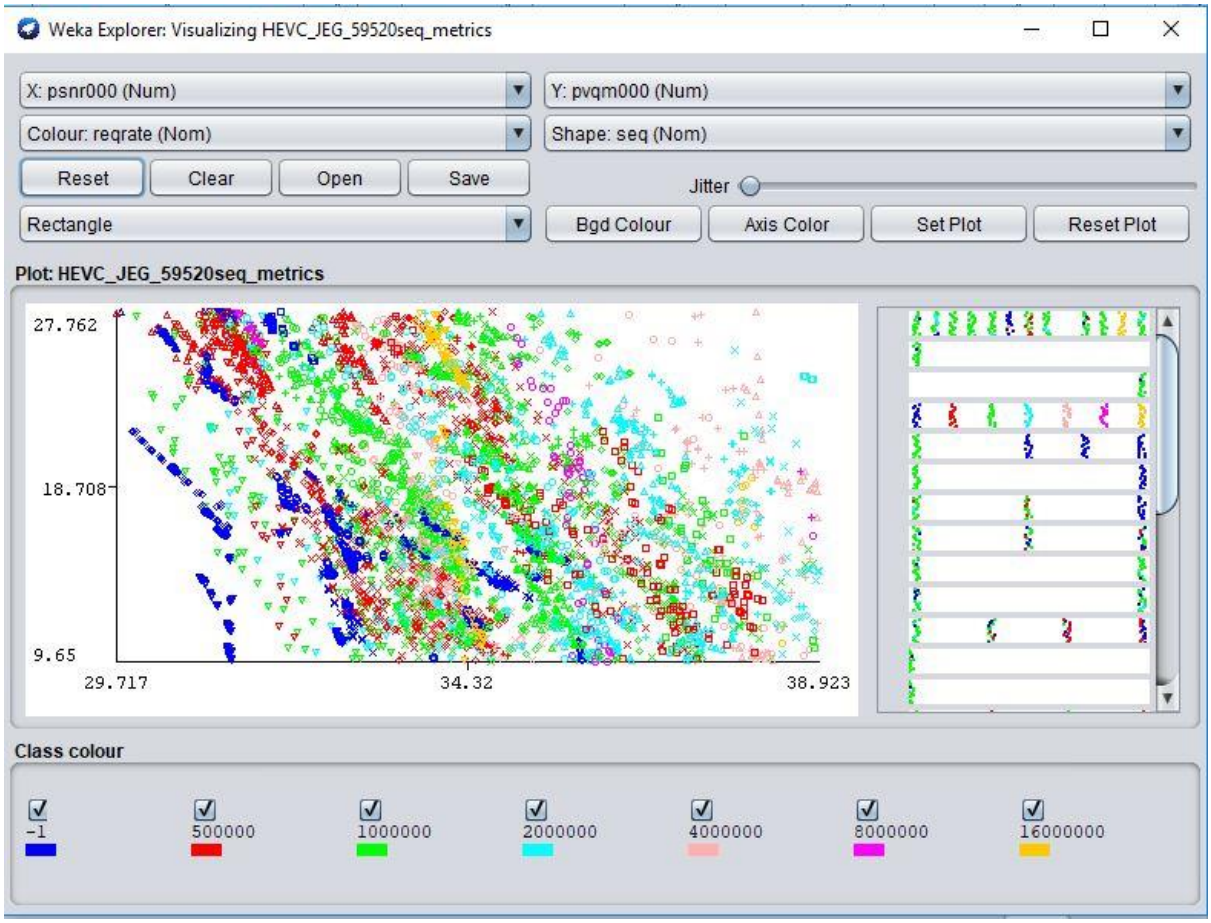


Fig4c.1.6 Visualize Panel with nominal shape attributes zoomed in

With Rectangle selected as the select instance, and the points mapped out by the rectangle is 9.65 – 27.762 by the y axis and 29.718 – 38.923 by the x axis. Also, you would notice now that both the x and y axes are zoom out properly, and this is because both properties selected it in the combo box above are numerical properties.

2. Bottom bar used to easily set colours

One interesting feature also worth discussing is the ability of the W.E.K.A software application to make it possible for colours to be easily set and changed by the bottom bar found in the class panel. Even if this feature was existent before, the panel to which it belongs has been endowed with other features (the colour labels and the checkboxes), and yet it still works

as expected. More would not be said about how this feature is implemented but rather what it does, because this has already been discussed in detail earlier. From the previous image, when we reset the plot we would get the image we have below in Fig 4c.2.1.

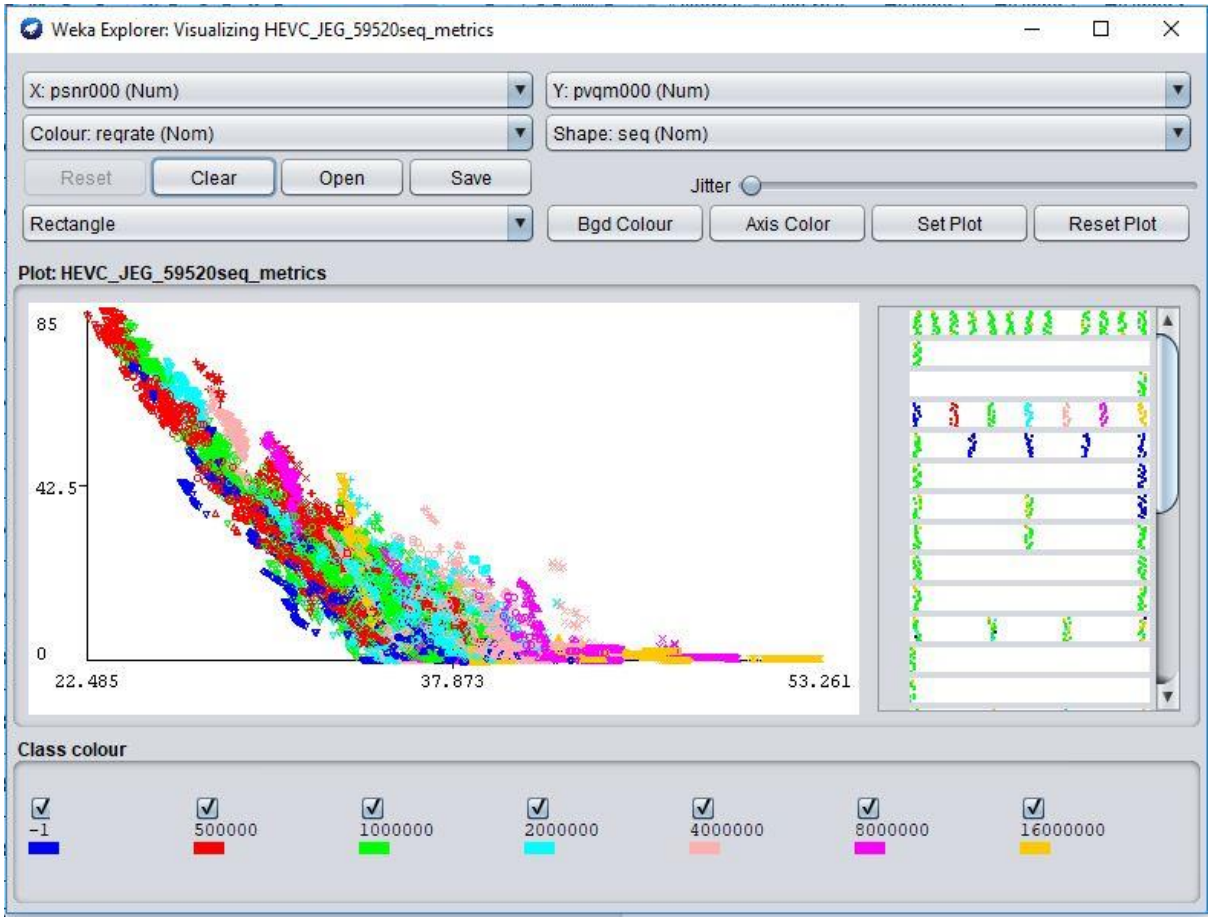


Fig4c.2.1 Visualization panel with properties psnr000 and pvqm000

We would see that the data points on the graph plot are ore clustered and with different colours. In Fig4c.2.1, there are 7 attributes named -1, 500000, 1000000, 2000000, 4000000, 8000000, 16000000. Each of these attributes have different colours. If the attribute differences are not clear, we can decide to change the attribute colour by selecting on the concerned attribute or it colour label and changing it easily. Changing the colour for attribute 50000 from red to yellow, we would have the image below in fig4c.2.2. We can now see a clear difference between the colours in the attribute and this makes the user understand their work even easier. If the user is still undecisive of the way the attributes colours should look individually on the graph plot, he/she can always change their colours again from the bottom bar.

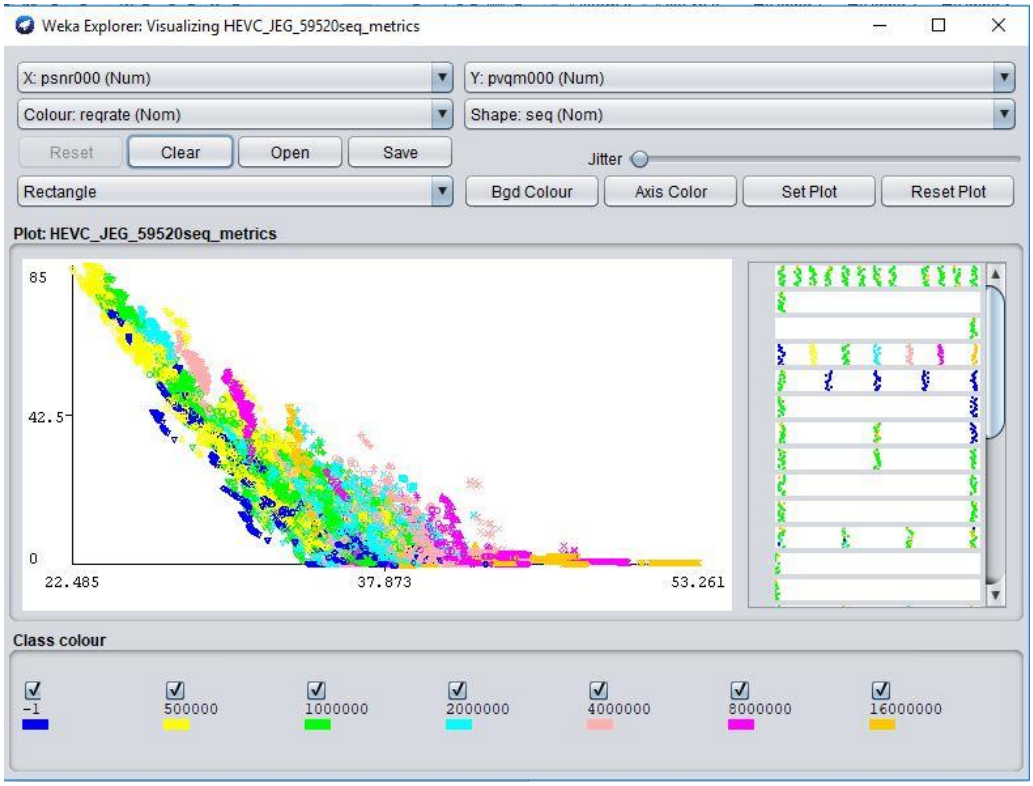


Fig4c.2.2 Visualization panel with properties psnr000 and pvqm000

An illustration of this is in Fig4c.2.3, I have changed the attribute colour for number 3 (1000000) and number 7 (16000000) from green to greenish brown and from orange to red respectively.

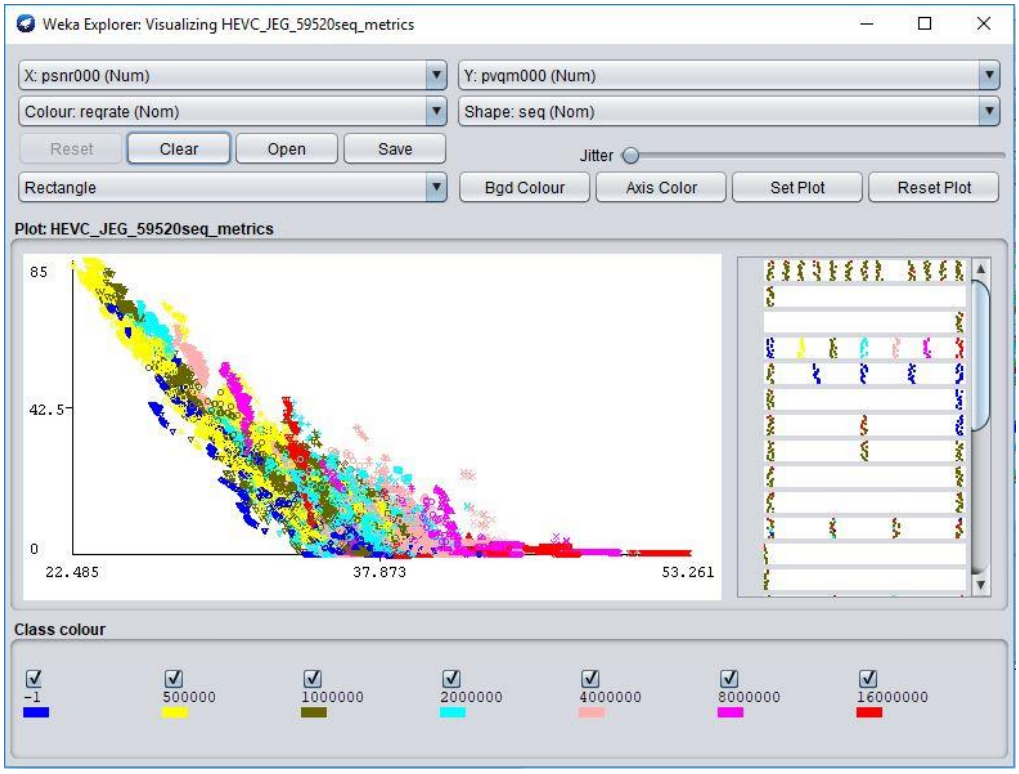


Fig4c.2.3 Visualization panel with properties psnr000 and pvqm00

3. Filter colour of data set with tick boxes

The last feature I would be elaborating about that added to the richness of the graph plot is the luxury of filtering our data sets with check boxes. Prior to this feature there was not way of filtering data sets from the visualization panel and saving it. At least there was not easy way. We would only highlight, and instance of the graph plot we want to see or change the property of the graph plot, also we would filter out the attributes from the data set in the pre-process panel, but this was a hard method that filters out all the attributes values in a data set for the attribute we have selected. It had almost no manipulability and effective if we need to remove just certain values from the data set.

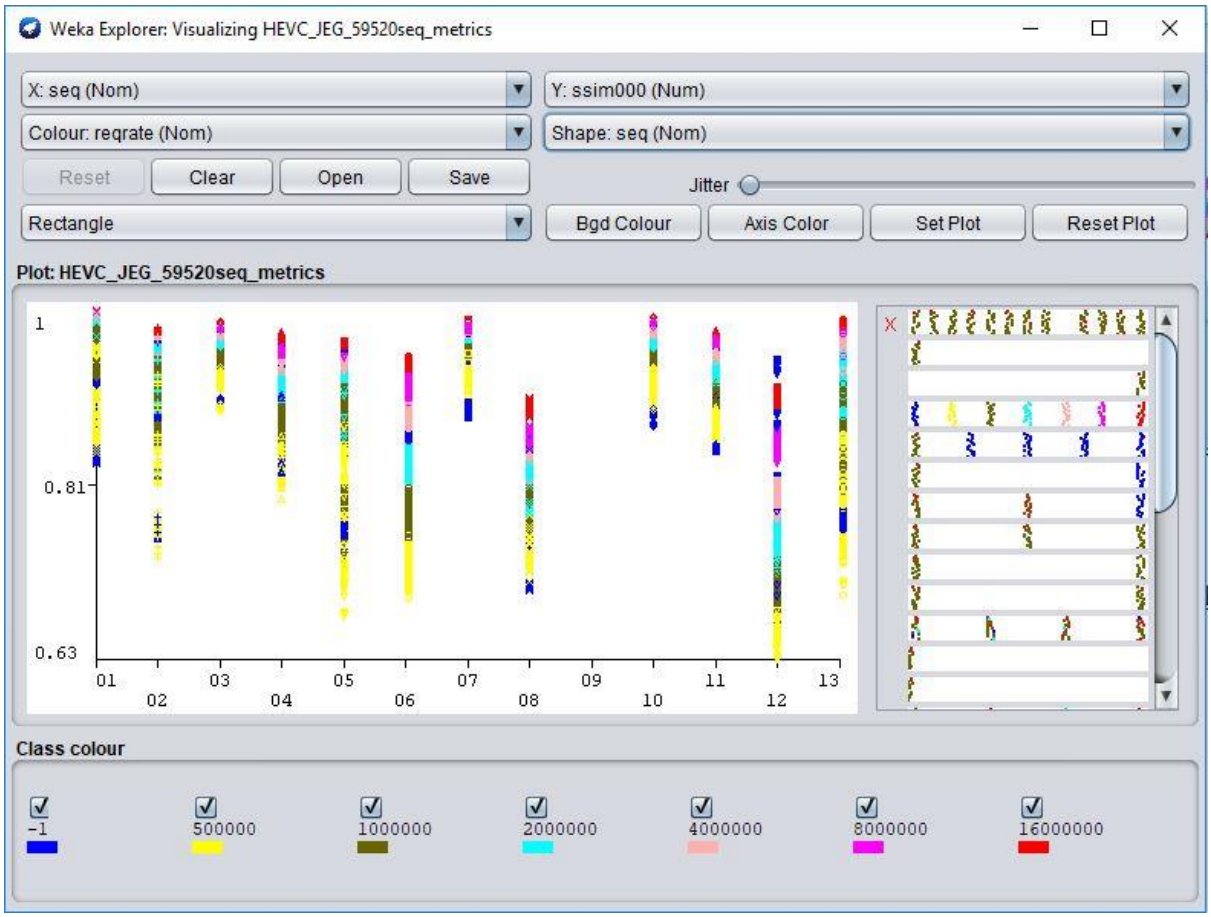


Fig4c.3.1 Visualization panel with properties seq and ssim000

But the addition of this feature eliminated this problem and datasets can now be easily filtered out with the highest dexterity as desired by the user. It can be done just by ticking or unticking the checkbox of the attribute concerned. As can be seen in the Fig 4c.3.1 and

Fig4c.3.2, the second attribute of the select instance in the graph plot can be filtered out just by unticking its checkbox.

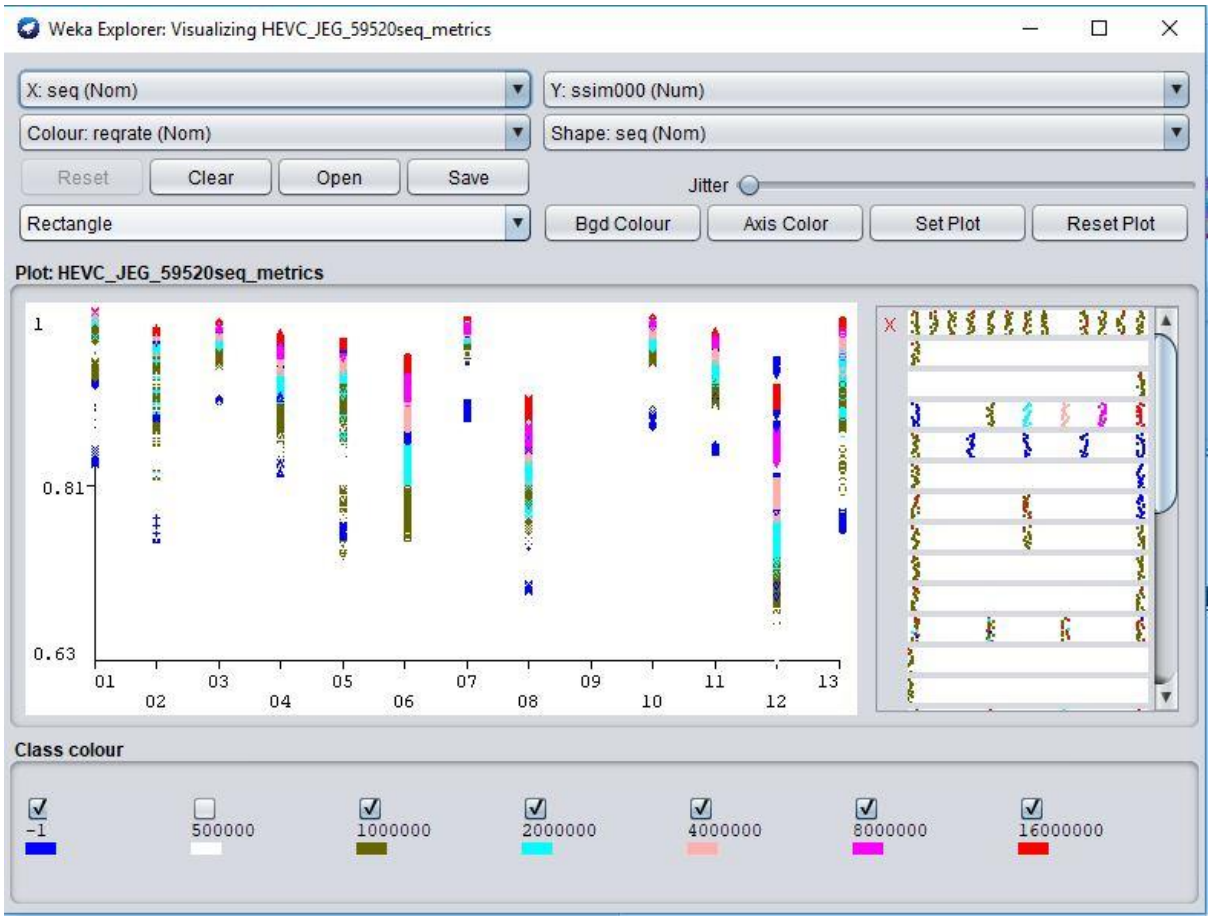


Fig4c.3.2 Visualization panel with properties seq and ssim000 with filter

Notice how the yellow colour of the attribute is filtered out and it now automatically represented by the colour white which is the colour of the graph plot background.

CHAPTER 5 – Conclusions

Data mining can be a tedious task, most especially when the tools needed to make it happen are lacking. Also, creating a solution for reading and investigating large chunk of data can be a laborious task. With the help of my supervisor Prof. Enrico Masala, I was able to come up with an improvement to the WEKA software where. This way, future usage and investigations with this software can be done easily. I also mainly concentrated on the visualization aspect of the software because it is the most utilised feature when mining data. Now when a scientist, student, researcher or user what to use WEKA for data mining, it makes their job an easier as features like, selecting distinct attributes, isolating preferred instances, modifying the minimum and maximum value of plot axis, displaying different attributes by colour and shapes, ability to change the colour and shape for better visualization have been added to the project.

REFERENCES

1. Machine Learning Group at the University of Waikato. University of Waikato, 2018, https://www.cs.waikato.ac.nz/ml/weka/, accessed 17th May 2018.
2. Stryker JP, Kuh CV, Voytuk JA, A Data Based Assessment of Research Doctorate in the US, Context and Motivation, Washington National Academic Press 2011, accessed 26th August 2018.
3. Machine Learning Group at the University of Waikato. University of Waikato , 2018, https://www.cs.waikato.ac.nz/ml/index.html, accessed 18th May 2018.
4. Computer Science Paper at the University of Arizona, University of Arizona, https://www2.cs.arizona.edu/projects/scout/Papers/mosberger/doc012.html, access 29th August, 2018.
5. Margaret Rouse, https://searchmicroservices.techtarget.com/definition/user-interface-UI, TechTarget, User Interface, accessed 4th September 2018.
6. Junaid Rehman, Advantage and Disadvantages of Graphic User interface, IT Release, http://www.itrelease.com/2017/11/advantages-disadvantages-graphical-user-interface/, accessed 7th September 2018.
7. Visualr Insights, 10 Advantages of data visualization, VisualR, https://visualrsoftware.com/advantages-data-visualization/, accessed 9th September 2018.
8. Arden Manning, Business Intelligence and Analytics, Yseop, Top 4 limitations of data visualization tools. https://yseop.com/blog/top-4-limitations-of-data-visualization-tools-2/, accessed 9th September 2018.
9. Robert Cordray, Big Data Zone, DZone, 7 Benefits of Data Visualization, https://dzone.com/articles/6-ways-data-visualization-can-change-your-company, accessed 9th September 2018.