Table of Contents

Summary

The process of data preparation and classification used on this assignment has taught me plenty about data analysis. First of all there is no description of what the data set is, there are 5000 rows of data with 32 columns and the values do not show any clear pattern like a time series or clear relationships with one another.

Not having domain knowledge of the data to process comes with its pros and cons. Pros definitely being the confidence to make assumptions and taking the general steps of data cleaning. We work only with columns that have a certain threshold of non-missing values, the columns with too many missing values can be dropped whilst the other columns can have their numeric values replaced with the mean or median and categorical columns can be replaced with the mode. The other general steps include methods to display the data type, peeking at the start and tail end of values, describing the quarterly percentiles and averages as well as visually plotting a histogram to gather a range of values and their distribution. We can be satisfied with our data analysis even if our models are not perfect but can achieve a high accuracy score.

The cons however is that by not knowing what the column attributes represent we are unable to select specific columns to properly train our models for classification; create subsets of attributes to find distinct combinations of class representation (for example in our titanic dataset we can determine survival rates of all passengers based on combinations of passenger class, gender, age range, embarkation point etc.).

Despite not having any column labels the process I have undertaken has been rewarding. It has led me to doing plenty of online searching on good data preparation habits and learning how to tackle data of enormous size. Using the basic functions of peeking, describing the numeric value columns and listing the unique value counts provides a very good gist of the dataset. The use of visual representation is extremely helpful to get additional insight that sometimes descriptive functions can't easily provide. The use of heatmaps in particular greatly aided me in identifying highly correlated columns as well as using Confusion Matrix to validate the number of errors the model makes when mislabeling a class. The very detailed process of classification outlined in Practical 5 has been the most challenging. It requires correct sampling steps taken to ensure the results at the end are valid. For example when implementing the k-NN classifier code, it led me to read more about data clusters and how data points can affect neighbourhoods based on their weights being uniform (equally weighted) or distance (closer neighbours have greater influence than those that are further away). In the first attempt of coding I had the opportunity to increment the n count in the list of n_neighbours to see how it affects the machine learning model and the results it produced. Obviously feeding too many values will negatively affect the runtime of our k-NN model.

The library packages that help to achieve this are aplenty and indeed makes the otherwise arduous process very straightforward. I've had to look up plenty of documentation on the preprocessing package from scikit-learn and certainly gathered plenty of information not just on the models used but also the other modules

for other processes that could be implemented such as the KBinDiscretizer for discretising continuous values into discrete values and OneHotEncoder for binarising text columns(with low percentage of unique values) into categorical columns (represented as integers). These other methods I have read up on would definitely be very useful for further cleaning of data to work effectively with machine learning models.

Methodology

Data Preparation

Using our knowledge of manipulating dataframes in Jupyter Notebook, our very first step is loading the data using the appropriate package ‘sqlite3’. Other packages to import include ‘pandas’, ‘matplotlib’ and ‘scitkit-learn’ which we have had practice with in our Data Mining practicals. With an ‘sqlite3’ viewer it appears to show two tables conveniently labeled ‘train’ and ‘test.

We store the sql data into a variable which we will store as two separate data frames (one for train and one for test). We can have a brief look at our dataset by displaying the first and last five rows of our dataframe with the head and tail function.

The following steps are all part of examining our training data (Note: we repeat these actions for our test data to spot any similarities and differences)
Examining our data:

- Printing the labels of each column and their associated data types
    - Test data has no data type for the class label
- We use the shape method to get the dimensions of the data frame (how many columns and rows)
    - Both train and test data have the same number of columns but different quantities (train has 5000 rows while test has 500 rows)
- Using the info method also provides the data types for each column as well as the number of missing values
- We also plot a histogram to get a brief look at the range of values and their occurrences
    - At first glance Att00-Att10-Att13 and Att16-Att22 appear almost identical
    - Use of the describe method for these columns to check their similarity
    - Use of the equals method to check if they are duplicate columns and turns out they aren’t
- Using a slightly modified function from Practical 2 to list the percentage of missing values for each column (Att09 - 19.96%, Att23 - 59.64% and Att25 - 1%)
- Creating a function to list the percentage of unique values for each column(lower means less unique values)
    - Att01, Att12, Att17, Att26 and class are extremely low
    - Use of the value_counts method to list all the values and their occurrences(turns out they are categorical values (binary: 1-0 and text: ABCD))

Now that we have had a good look at our data set we go through the process of removing any irrelevant attributes and/or replacing the missing values in our almost filled columns. (Note: We repeat some if not all actions for our test data set)

Removing attributes:

- Although we encountered missing values in our training data whilst examining it we didn't encounter any for our test data set.
    - However we would still need to remove these attributes in the test data to maintain consistency in our models later
- We will remove these columns (We repeat these for our test data set):
    - Index - as it is only an index of rows it is not required for data classification
    - Att23 - as it has 59.64% missing values it will be inadequate for data classification
    - Att01, Att11 and Att12 - Although these are categorical values they contain extra values that our test data does not (Att01 - 1 count of HIWU, Att11 - 1 count of KIKU, Att12 - 1 count of NZEZ)
- We will replace the missing values in these columns with the mean
    - Use of the mean method to fill null with the mean of Att09 and Att25

Our final step in preparing our data is to tidy it. (Note: We will repeat these actions for our test data)
Tidy Data:

- In the examination phase the data type for our class attribute is listed as float64
    - Use of astype method to convert that column's data type to integer
- Standard Scaling of all attributes that have a normal distribution
    - Use of scikit-learn StandardScaler method to acquire a normal distribution of our values
    - After scaling we see that the peak of our normal distribution lies at 0 with the range of values drastically reduced (from -40,000 to -4). *(Refer to Figure 1a and 1b)*
- MinMax Scaling of attributes that have a uniform distribution
    - Att18, Att20 and Att28 have the MinMaxScaler method applied to now scale their range of values between 0 and 1. *(Refer to Figure 1a and 1b)*
- Splitting our training data
    - We create a copy of the training data frame without the values column, then stores the isolated class values column to an array (y_arr)
    - This is for the correlation matrix to observe any similarities between columns
- Identifying highly correlated columns
    - Using the corr method along with the seaborn plotting package we generate a heatmap of attributes compared to each other in a 2-D grid.
    - The correlation ranges from [-1,1], we will assume a value of greater than 0.95 to be highly correlated. *(Refer to Figure 2a)*
    - The attributes that satisfy this condition are Att16, Att19, Att22, Att28 and Att29 which will be dropped
    - We plot correlation matrix again with the columns now dropped to check if we have missed any other attributes. *(Refer to Figure 2b)*

At this point the training data is now considered cleaned and ready for classification.

Data Classification

Before we train our models, we need to split the train data into two subsets, one for model training and one for validation(comparing the predicted and actual classes). 25% of the train data will be set for validation under X_test whilst the remaining will be set under X_train. We use the train_test_split module from 'sci-kit-learn' to achieve this.

We also need to address the class imbalance to avoid any bias in our model training. The training models used will be the ones learned from the practicals: K-NN, Decision Tree and naive-Bayes.

Class Imbalance

- Using value_counts method we get the counts for each class label (0: 1010, 1: 1500, 2: 2490)
- Therefore we create a fair sample size by undersampling for n = 1010 using the RandomUnderSampler method

K-NN classifier:

- We calculate data similarity and distance using the KNeighborsClassifier method with a range of n-neighbours: [1,3,7,11,17,21] between uniform and distance weights.
- Our champion with the highest accuracy is revealed to be n_neighbour: 17 and weight: distance with an accuracy score ~ 89%
- We then create a confusion matrix using the appropriate module to compare the predicted and actual data labels. *(Refer to figure 3a)*
- The count of errors sit under or equal to 20 for all three classes which is a sign that the accuracy score is valid

Decision Tree classifier:

- Using our DecisionTreeClassifier module with a criteria of gini and entropy and sample split: [3,10,15,20] we have eight combinations to select from
- The champion has an accuracy score ~ 75% with the entropy criteria and a sample split of 3

Naive-Bayes classifier:

- Using the naive_bayes method we have an accuracy score ~74%
- The confusion matrix highlights that we have an error count of average 98.83. *(Refer to Figure 3b)*
- This reflects that our prediction model will mislabel 98.83 times for each class

Our best two models based on accuracy score appear to be K-NN and Decision Tree which we will use for our prediction against the unlabelled test dataset.

Prediction

FIrstly we drop the class column in our test dataframe as it is empty. We then use the predict method of the two models on the test dataset to obtain an array of predicted class labels. Finally we create a dataframe to store the predictions and save it as a ‘sqlite’ file.

Conclusion

Initially in the examining data phase I gathered that the text columns ‘Att01’, ‘Att11’ and ‘Att12’ contained additional values in the train dataset that was not present in the test dataset. Therefore I made the decision to drop it based on the reasoning that it would be inconsistent in the model training as well as text that appear to be acronyms without any real meaning. However in the later parts of the assignment I read up more on modules for discretising and binarising which led me to realise that the text values could have been binarised to categorical integers that can be fed to the machine learning model to train its classification. Regarding the additional text value that didn’t exist in the test dataset, since it was only 1 count I could have easily removed that row of values to not disrupt the training model.

WIth the accuracy score of the Decision Tree and naive-Bayes classifier sitting around 75%, it has led me to wonder if any more columns could have been dropped to improve the accuracy score. I also didn’t implement any methods to check for and remove outliers. Perhaps if a column contained a number of outlets above a certain threshold those columns could have been removed.

Overall this was a very insightful and challenging assignment that feels very grounded with industry level data analysis. Although I had very little knowledge of Data Mining coming into this course I certainly have gained awareness of how much more there is to expect beyond this unit.

References

1. Steps for basic data examination taken from Practical 2 of Data Mining
    a. These include the ordered steps of peeking the head and tail of the dataset
    b. Looking at the number of columns and rows
    c. Acquiring a description of the data type and count of missing values
2. List_missing function to list the percentage of missing values modified from Practical 2
    a. Modified the function of listing columns with missing values >80% to listing all column percentages
    b. Also reworked the function to display the percentage of unique values
3. Gathered several useful descriptive functions from pandas documentation online: value_counts(), info() from https://pandas.pydata.org
4. Identify duplicates and mean functions acquired from Practical 2
5. Steps for Data Classification taken from Practical 5 of Data Mining
6. Code to implement Standard Scaling acquired from Practical 5
    a. Code reworked to implement MinMax Scaling
7. Splitting Training Data knowledge gathered online from https://pandas.pydata.org
    a. loc and drop method to maintain dataframe and remove columns
    b. values method to extract rows and store into an array
8. Code to implement Correlation Matrix acquired from Practical 2
9. Undersampling code for RandomUnderSampler gathered online from http://imbalanced-learn.org
10. Splitting Training Data Code acquired from Practical 5
    a. train_test_split module imported from sklearn package from http://scikit-learn.org
11. Code to implement k-NN, Decision Tree and naive-Bayes classifier acquired from Practical 5 which are all from the sklearn package from http://scikit-learn.org
12. Class Label Prediction code gathered online from http://scikit-learn.org
13. Saving to SQL file code acquired from Practical 5

Appendices

Figure 1a (Before Scaling)

```
dfTest.hist(figsize=(24,12))
plt.show()
```

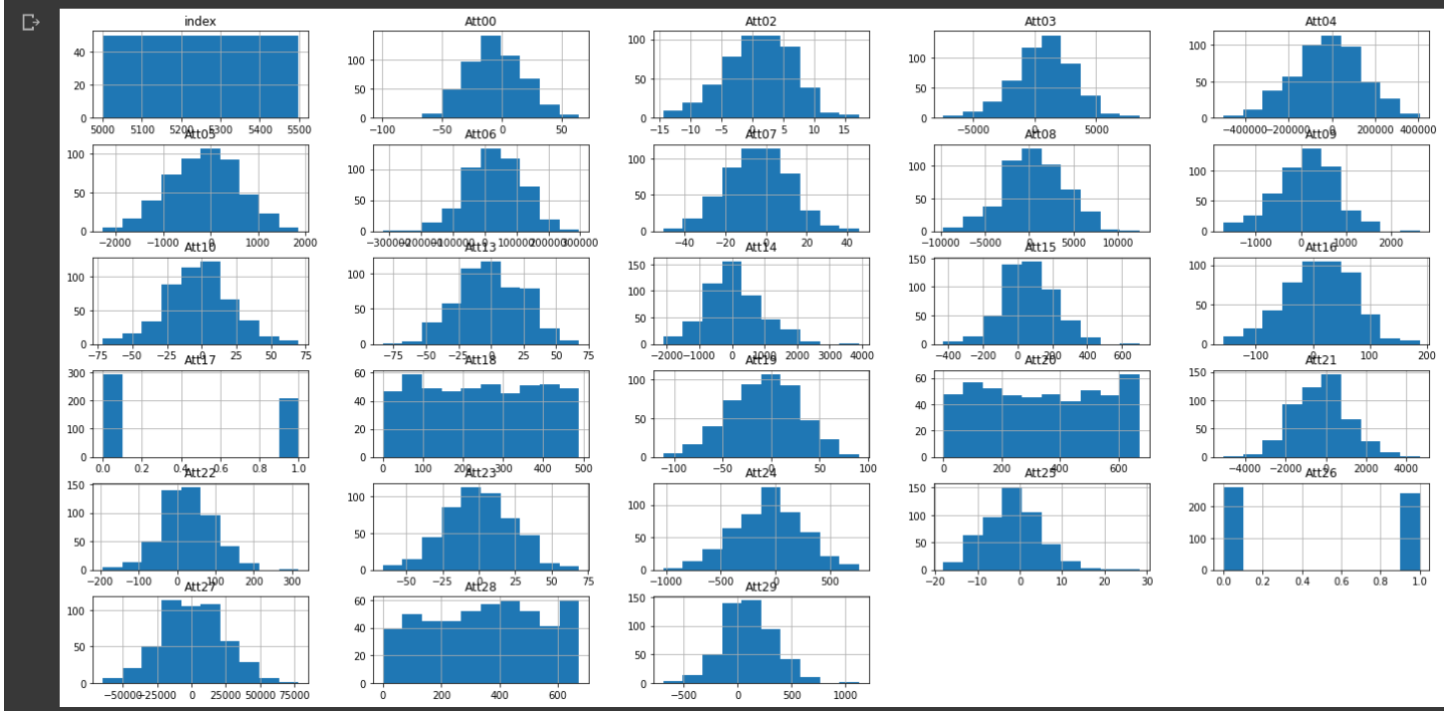

Figure 1b (After Scaling)

```
dfTrain.hist(figsize=(24,12))
plt.show()
```

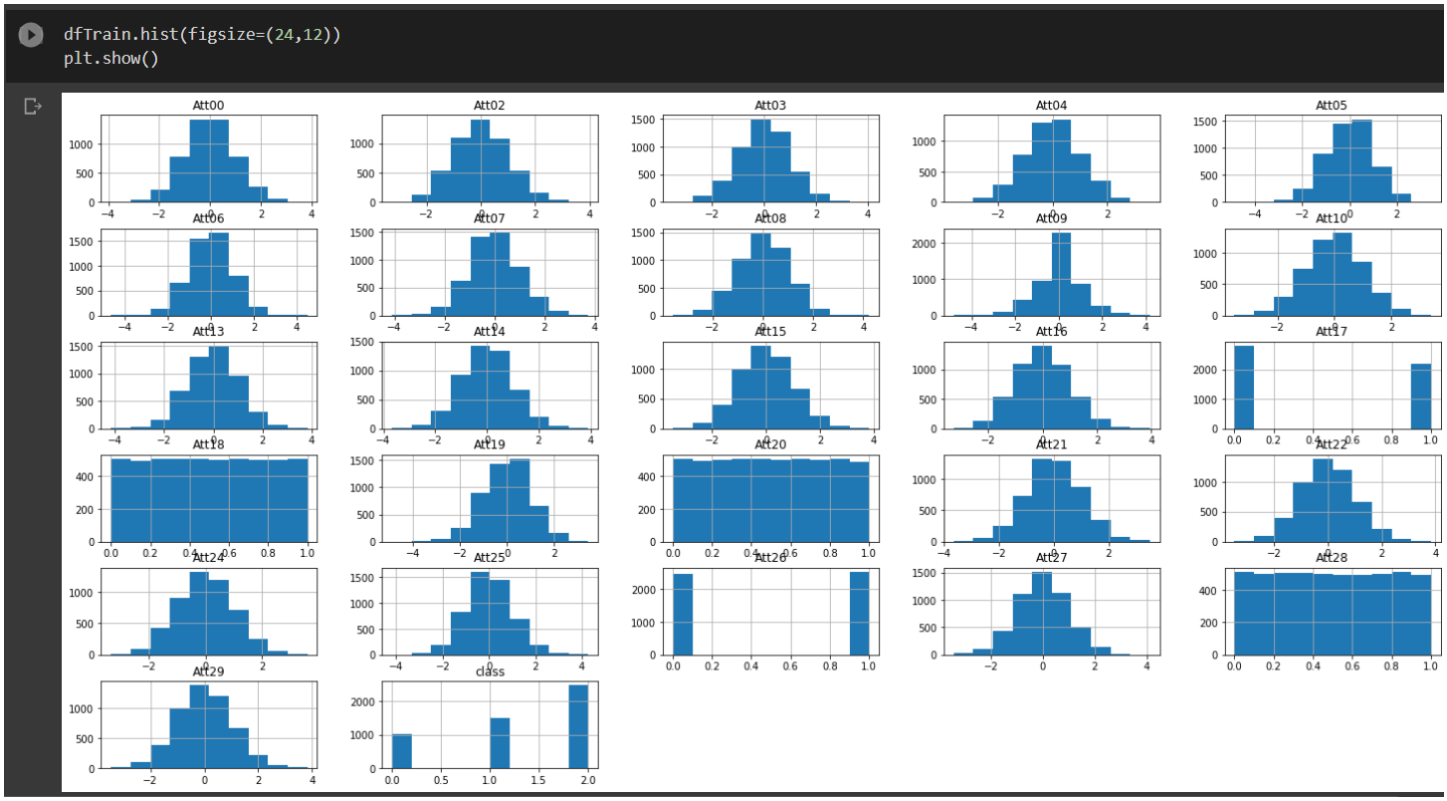

Figure 2a (Correlation Matrix)

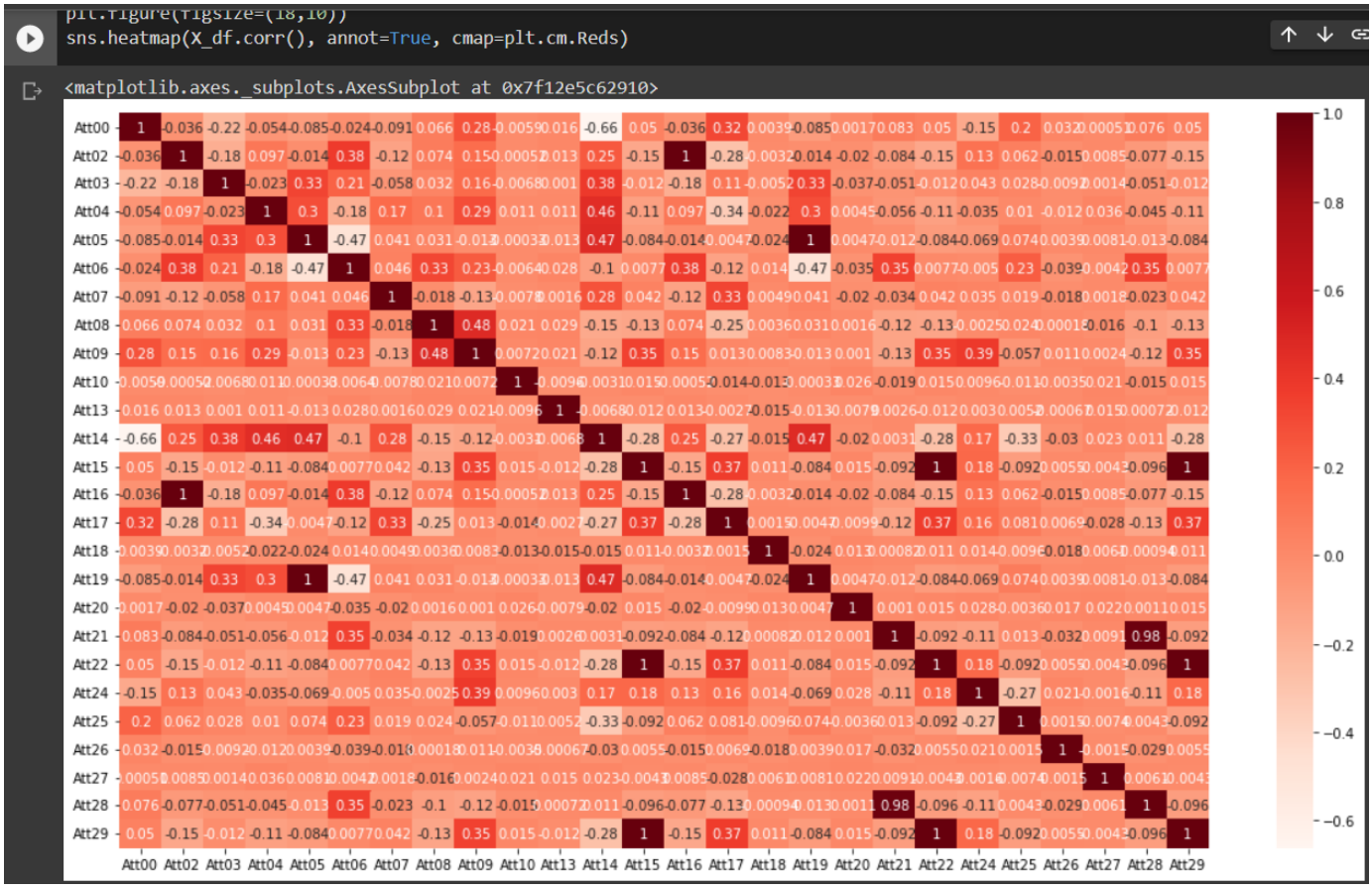

Figure 2b (After dropping highly correlated columns)

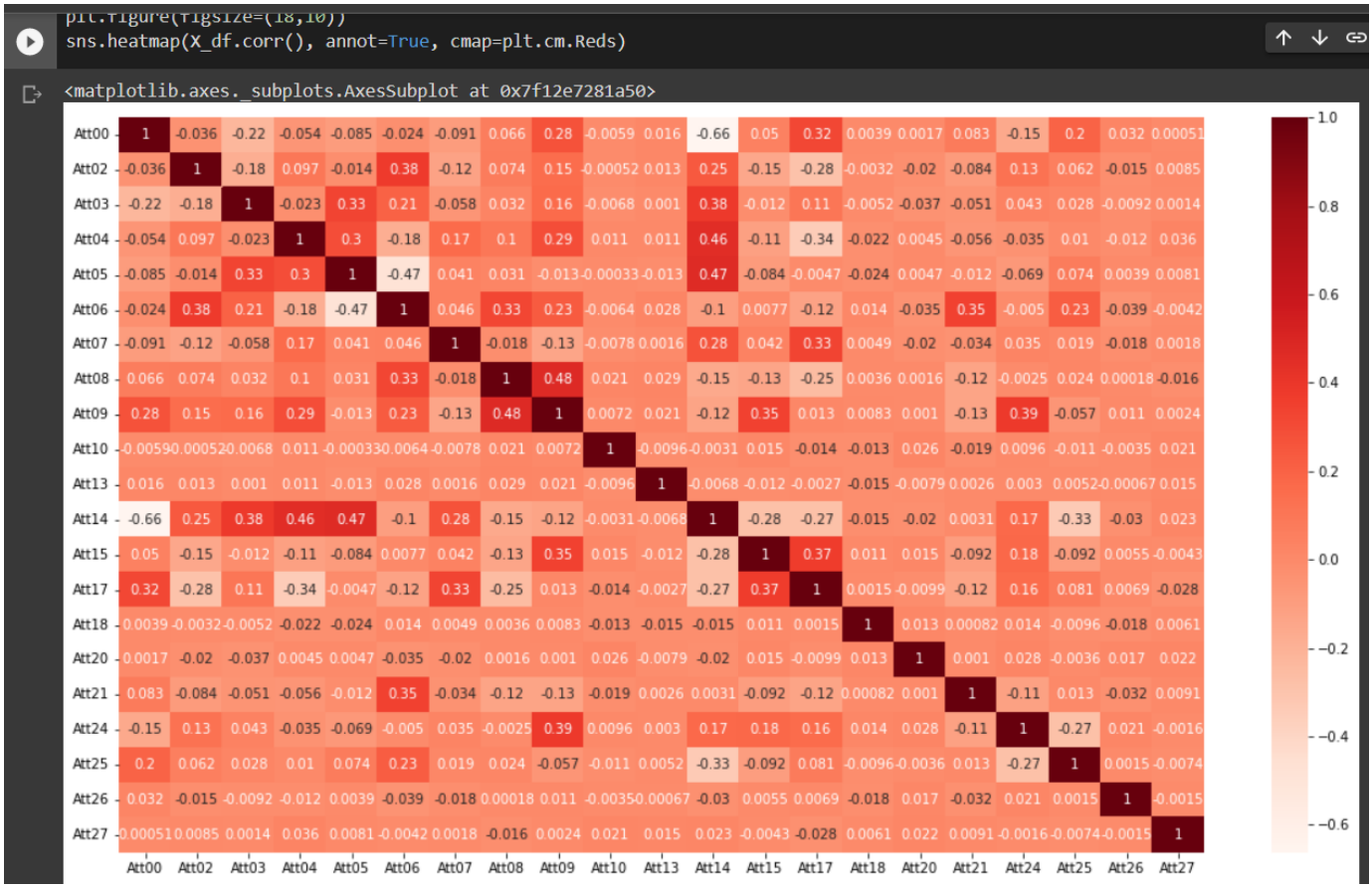

Figure 3a (Confusion Matrix against k-NN model)

```
fig, ax = plt.subplots(1,1, figsize=(6, 6))

ConfusionMatrixDisplay.from_estimator(best_knn,
                                      X_test, y_test,
                                      display_labels=dfTrain['class'],
                                      ax=ax)
plt.tight_layout()
plt.show()
```

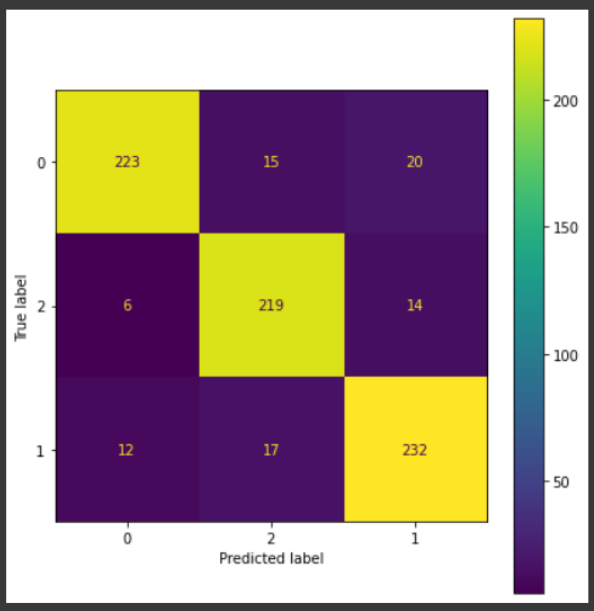

Figure 3b (Confusion Matrix against naive-Bayes model)

```
from sklearn import naive_bayes

# no parameters to adjust so no need to optimise, just train
fig, ax = plt.subplots(1,1)
nb = naive_bayes.GaussianNB()
nb.fit(X_train, y_train)
ConfusionMatrixDisplay.from_estimator(nb,
                                      X_train, y_train,
                                      display_labels=dfTrain['class'],
                                      ax=ax)
plt.tight_layout()
plt.title('Confusion Matrix for Naive Bayes Model')
plt.show()
```