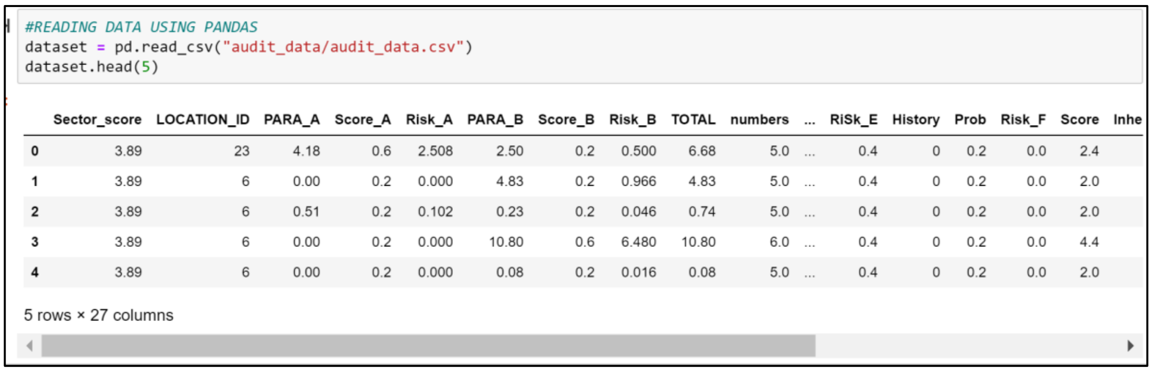
Description of the problem

The purpose of this report is to critically analyse an audit dataset consisting of **776** instances from (*Fig 1*) six firms, with the rationale being to build a classification model that will classify whether an organisation has a very high likelihood to be involved in fraud or not based on the historical and present factors. The data consists of **26** attributes containing continuous values with the **27th** (Fig 1) being the class attribute containing a discrete value of **0** and **1**, with 0 being that the firm is *not* likely at all to be engaged in any fraudulent activity, and 1 being the firm *is very likely to be* involved in a fraudulent activity.

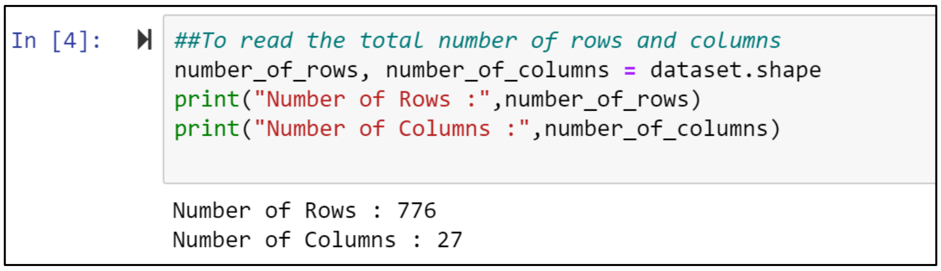
Out of the **776** instances, **471** were classified as **not** fraudulent (0) and **305** as fraudulent (1), representing **60.70%** and **39.30%** respectively as shown in Fig 2.



```
#READING DATA USING PANDAS
dataset = pd.read_csv("audit_data/audit_data.csv")
dataset.head(5)
```

	Sector_score	LOCATION_ID	PARA_A	Score_A	Risk_A	PARA_B	Score_B	Risk_B	TOTAL	numbers	...	RiSk_E	History	Prob	Risk_F	Score	Inhe
0	3.89	23	4.18	0.6	2.508	2.50	0.2	0.500	6.68	5.0	...	0.4	0	0.2	0.0	2.4	
1	3.89	6	0.00	0.2	0.000	4.83	0.2	0.966	4.83	5.0	...	0.4	0	0.2	0.0	2.0	
2	3.89	6	0.51	0.2	0.102	0.23	0.2	0.046	0.74	5.0	...	0.4	0	0.2	0.0	2.0	
3	3.89	6	0.00	0.2	0.000	10.80	0.6	6.480	10.80	6.0	...	0.4	0	0.2	0.0	4.4	
4	3.89	6	0.00	0.2	0.000	0.08	0.2	0.016	0.08	5.0	...	0.4	0	0.2	0.0	2.0	

5 rows × 27 columns



```
In [4]:  ##To read the total number of rows and columns
         number_of_rows, number_of_columns = dataset.shape
         print("Number of Rows :",number_of_rows)
         print("Number of Columns :",number_of_columns)

         Number of Rows : 776
         Number of Columns : 27
```

Figure 1: Number of Rows and Columns

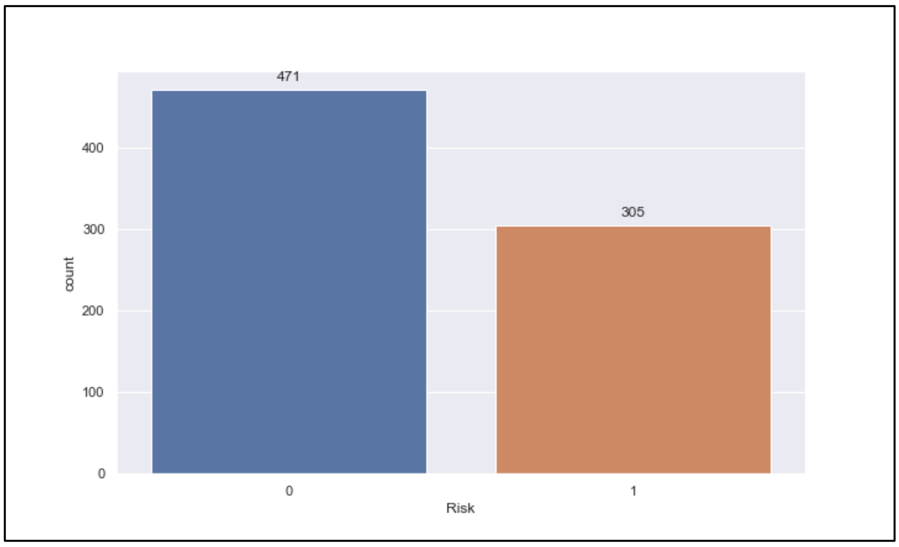
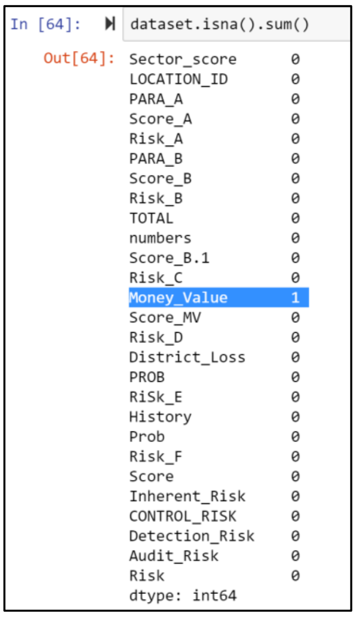


Figure 2: Count of values in Dependent Variable

By running the command below, the output showed that the dataset has only **1** missing value in the **Money_Value (**As shown in Figure 3**)** attribute which is just 0.1% of the whole dataset. As this is very low (1 out of 776) it can either be imputed with the mean of that attribute to have it filled up or the particular row can just be omitted from the data. However, I will fill it with the mean of the Money_Value attribute.



```
In [64]:  dataset.isna().sum()

Out[64]:  Sector_score      0
          LOCATION_ID       0
          PARA_A            0
          Score_A           0
          Risk_A            0
          PARA_B            0
          Score_B           0
          Risk_B            0
          TOTAL             0
          numbers           0
          Score_B.1         0
          Risk_C            0
          Money_Value       1
          Score_MV          0
          Risk_D            0
          District_Loss     0
          PROB              0
          RiSk_E            0
          History           0
          Prob              0
          Risk_F            0
          Score             0
          Inherent_Risk     0
          CONTROL_RISK      0
          Detection_Risk    0
          Audit_Risk        0
          Risk              0
          dtype: int64
```

Figure 3: Missing Values

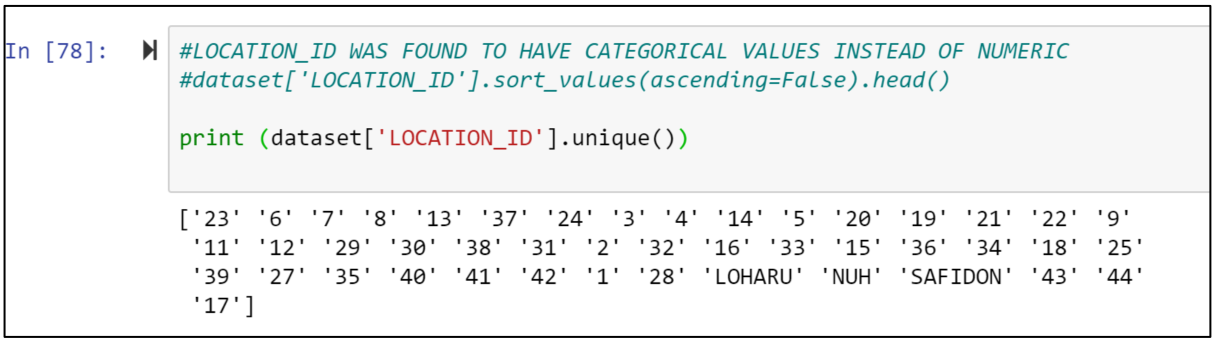
A **Support Vector Machine** algorithm will be used to classify whether an organization is involved in a fraudulent activity or not. This is chosen because it is good for classifying in such situations and will draw a line, referred to as a hyperplane to distinguish between the two, which will be shown at the end of this report.

Construction and tuning the classifier

DATA PRE-PROCESSING

- CHECKING AND HANDLING MISSING VALUES AND UNEXPECTED VALUES

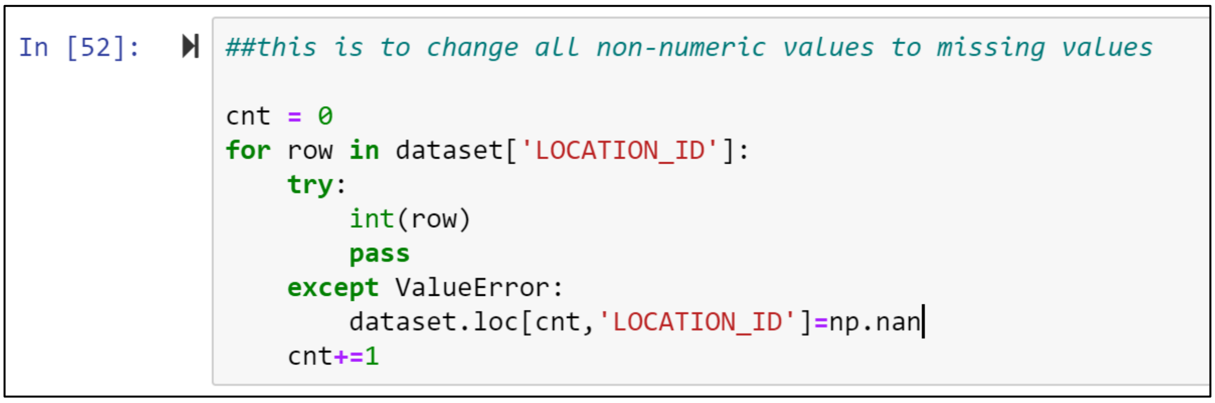
Firstly, all missing and incorrect data had to be found and dealt with, during this it was found out that one attribute '**LOCATION_ID**' had 3 instances having non-numeric values as shown in Figure 4. As mentioned earlier, the data had only one missing in the '**Money_value**' attribute.



```
In [78]:  #LOCATION_ID WAS FOUND TO HAVE CATEGORICAL VALUES INSTEAD OF NUMERIC
          #dataset['LOCATION_ID'].sort_values(ascending=False).head()

          print (dataset['LOCATION_ID'].unique())

          ['23' '6' '7' '8' '13' '37' '24' '3' '4' '14' '5' '20' '19' '21' '22' '9'
           '11' '12' '29' '30' '38' '31' '2' '32' '16' '33' '15' '36' '34' '18' '25'
           '39' '27' '35' '40' '41' '42' '1' '28' 'LOHARU' 'NUH' 'SAFIDON' '43' '44'
           '17']
```

Figure 4: Non-Numeric Values found

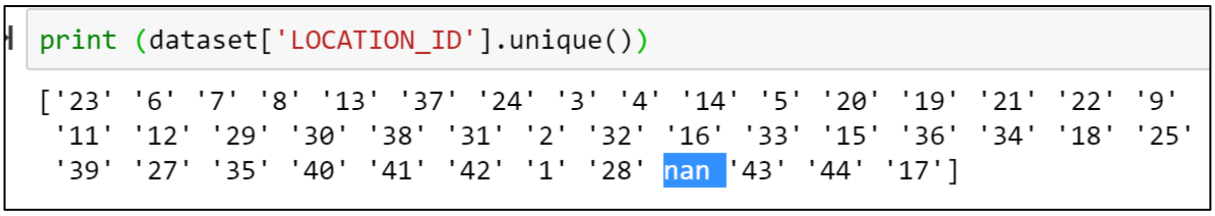
To have this treated, a *for loop* was created to change all non-numeric values and replace them as missing values as shown in **Figure 5.**



```
In [52]:  ##this is to change all non-numeric values to missing values

          cnt = 0
          for row in dataset['LOCATION_ID']:
              try:
                  int(row)
                  pass
              except ValueError:
                  dataset.loc[cnt,'LOCATION_ID']=np.nan
              cnt+=1
```

Figure 5: Converting Non-Numeric to NaN's

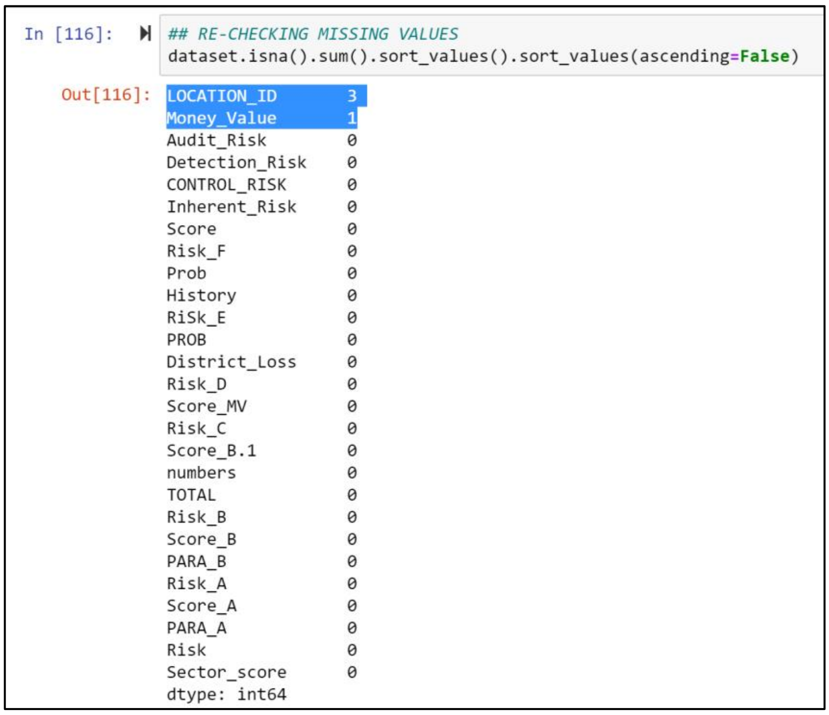
Below is confirmation of the non-numeric values being treated as missing values.



```
print (dataset['LOCATION_ID'].unique())

['23' '6' '7' '8' '13' '37' '24' '3' '4' '14' '5' '20' '19' '21' '22' '9'
 '11' '12' '29' '30' '38' '31' '2' '32' '16' '33' '15' '36' '34' '18' '25'
 '39' '27' '35' '40' '41' '42' '1' '28' nan '43' '44' '17']
```

Figure 6: Non-Numeric values treated as NaN's

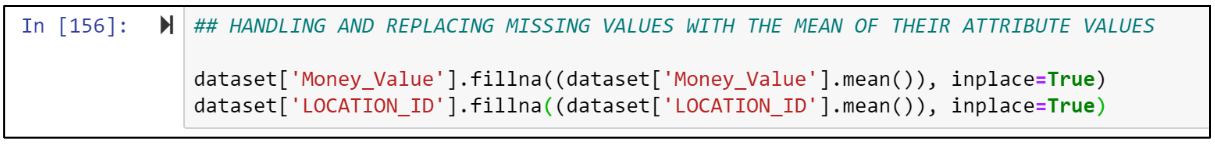
Once this was done, the number of missing values then increased to **4**, one in 'Money_Value' and 3 in 'LOCATION_ID' as shown in **Figure 7.**



```
In [116]:   ## RE-CHECKING MISSING VALUES
            dataset.isna().sum().sort_values().sort_values(ascending=False)

Out[116]:   LOCATION_ID       3
            Money_Value       1
            Audit_Risk        0
            Detection_Risk    0
            CONTROL_RISK      0
            Inherent_Risk     0
            Score             0
            Risk_F            0
            Prob              0
            History           0
            RiSk_E            0
            PROB              0
            District_Loss     0
            Risk_D            0
            Score_MV          0
            Risk_C            0
            Score_B.1         0
            numbers           0
            TOTAL             0
            Risk_B            0
            Score_B           0
            PARA_B            0
            Risk_A            0
            Score_A           0
            PARA_A            0
            Risk              0
            Sector_score      0
            dtype: int64
```

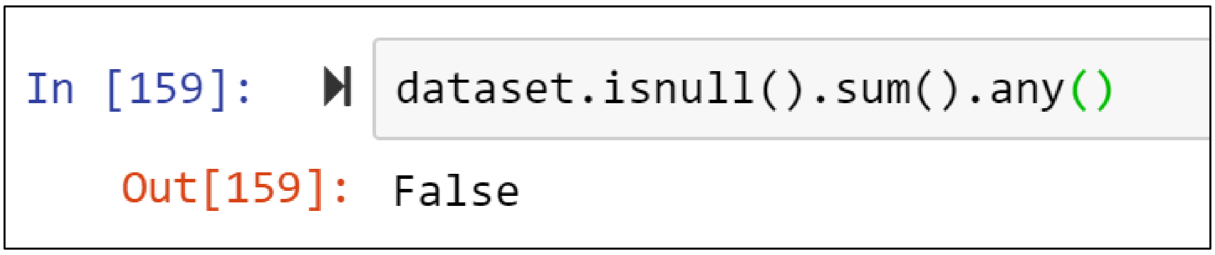
Figure 7:Count of Missing Values in Dataset

These missing values were then replaced with their mean of their various attribute values respectively **(Fig 8),** therefore leaving no missing values in the data, as shown by the command in Fig 9, which shows *TRUE* if there are missing values and *FALSE* if there are no missing values.



```
In [156]:   ## HANDLING AND REPLACING MISSING VALUES WITH THE MEAN OF THEIR ATTRIBUTE VALUES

            dataset['Money_Value'].fillna((dataset['Money_Value'].mean()), inplace=True)
            dataset['LOCATION_ID'].fillna((dataset['LOCATION_ID'].mean()), inplace=True)
```

Figure 8: Replacing Missing values with their mean



```
In [159]:   dataset.isnull().sum().any()

Out[159]:   False
```

Figure 9: Proof of no missing values

As all invalid and missing data has been handled now, I then move on to visually exploring my data and the features to see which attributes and matter and which do not so I can use the features that matter for the model building.

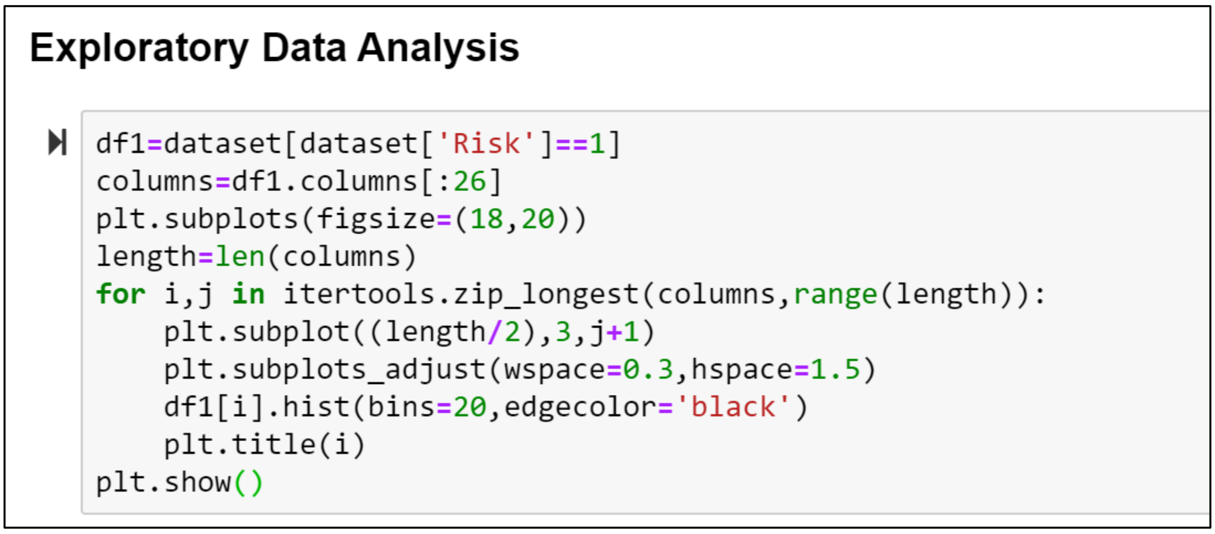
EXPLORATORY DATA ANALYSIS

Before feature selection can be well performed, a basic understanding of the domain and of some of the variables and an Exploratory Data Analysis will help inform the inclusion and exclusion of the features. Diagram below gives a summary of the major features in the dataset.

FEATURE	SUMMARY
LOCATION_ID	Unique ID of the City/Province
Sector_Score	Historical risk score value of the Firm Target Unit
PARA_A	Discrepancy found in the planned expenditure of inspection and summary report A in Rs (in crore)
PARA_B	Discrepancy found in the planned expenditure of inspection and summary report B in Rs (in crore)
TOTAL	Total amount of discrepancy found in PARA_A and PARA_B
NUMBER	Historical Discrepancy score
Money_Value	Amount of money involved in misstatement in the past audits
Loss	Amount of loss suffered by the firm last year
History	Average Historical Loss suffered by the firm in the last 10 years
Score	Historical Risk Score of a district in the last 10 years
Risk	Risk class assigned to an audit-case. (Target class Feature)

Table 1:Summary of Major Features

By running command below which is to visualise all the features in the dataset with a bar graph and from the output (Fig 10), it can clearly be seen that the attribute '**LOCATION_ID**' (1) is very insignificant for our prediction process, and therefore can be dropped from our dataset.



Exploratory Data Analysis

```
df1=dataset[dataset['Risk']==1]
columns=df1.columns[:26]
plt.subplots(figsize=(18,20))
length=len(columns)
for i,j in itertools.zip_longest(columns,range(length)):
    plt.subplot((length/2),3,j+1)
    plt.subplots_adjust(wspace=0.3,hspace=1.5)
    df1[i].hist(bins=20,edgecolor='black')
    plt.title(i)
plt.show()
```

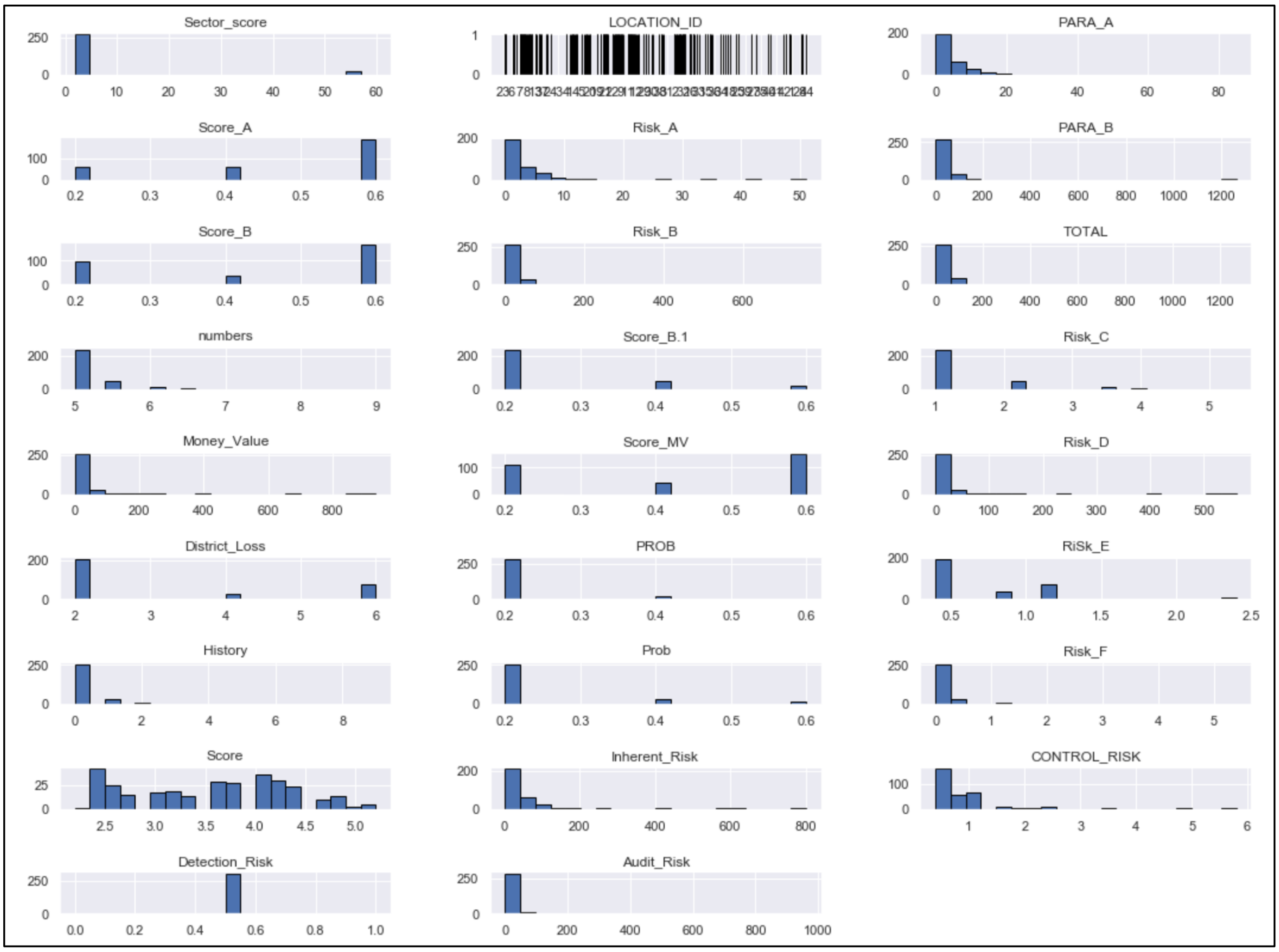


Figure 10: Visual Exploration of features

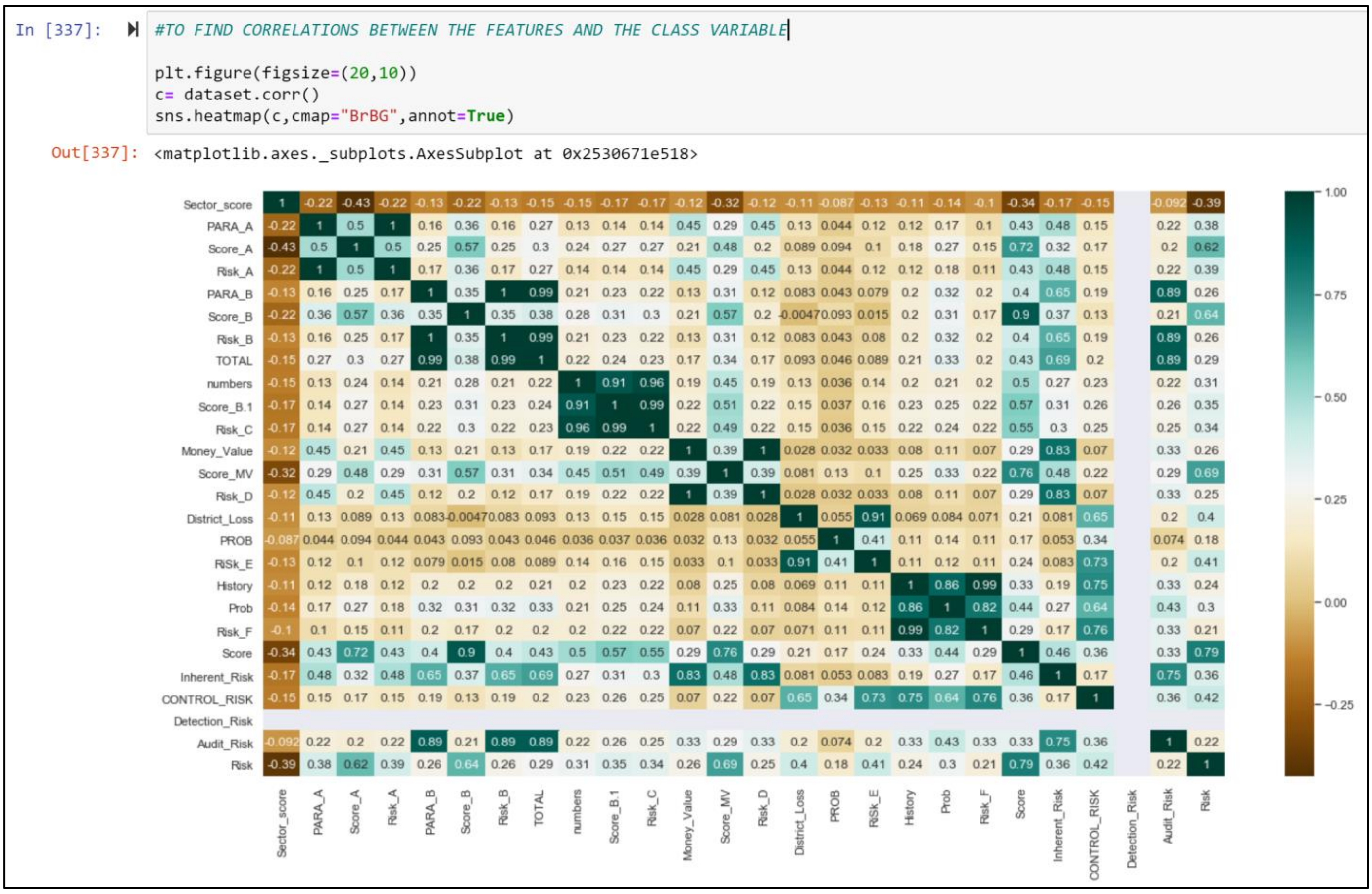
Secondly, the attribute "**TOTAL**" (2) will also be taken out as it is just is the total of "***PARA_A***" and "***PARA_B***" features (as mentioned in Table 1) and therefore will be redundant if kept in the data.

CORRELATION MATRIX WITH HEATMAP

A correlation matrix is a table which shows the correlation coefficients between variables, most especially between the predictors and the dependent variable. Each cell has a figure which represents the correlation between two variables. Its used to summarize the data.

By creating a heatmap to draw and show the important features having correlations with the Class Variable ("Risk"), it can be seen that "**Detection_Risk**" (3) in Fig 11 had no relation and therefore showing as '*BLANK*', therefore has to be taken out. This is due to the fact that there is no variation in that attribute. All the values are the same, 0.5, and therefore won't be a good feature for training the model.

The command used to run the correlation and the output are shown in Fig 11 below.



```
In [337]:  #TO FIND CORRELATIONS BETWEEN THE FEATURES AND THE CLASS VARIABLE

           plt.figure(figsize=(20,10))
           c= dataset.corr()
           sns.heatmap(c,cmap="BrBG",annot=True)

Out[337]:  <matplotlib.axes._subplots.AxesSubplot at 0x2530671e518>
```

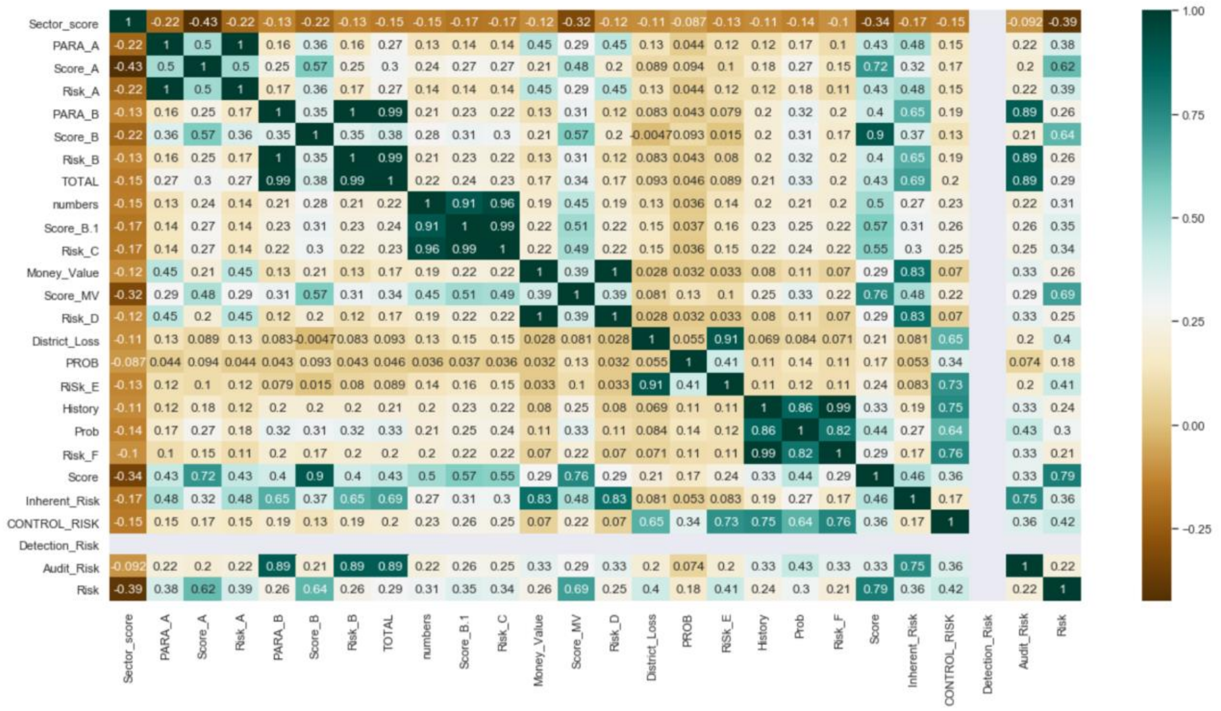
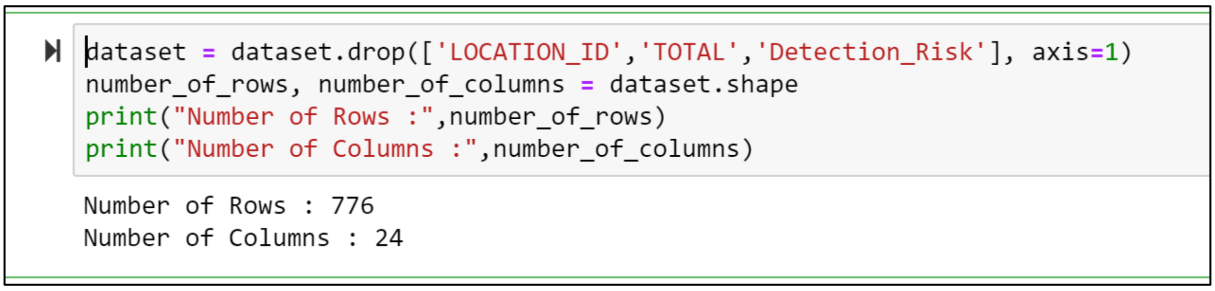


Figure 11: Correlation Matrix with Heatmap

By doing all these checks and exploration, three features can now be confidently dropped from our data(*LOCATION_ID, Total and Detection_Risk*). This is to enable us only have the significant features to build the model, reduce noise in the data and increase the accuracy of our classification.

Below is the command being used to drop the three features which is insignificant for building our model, and therefore reducing the number of features from 27 to 24. (Fig 12)



```
dataset = dataset.drop(['LOCATION_ID','TOTAL','Detection_Risk'], axis=1)
number_of_rows, number_of_columns = dataset.shape
print("Number of Rows :",number_of_rows)
print("Number of Columns :",number_of_columns)

Number of Rows : 776
Number of Columns : 24
```

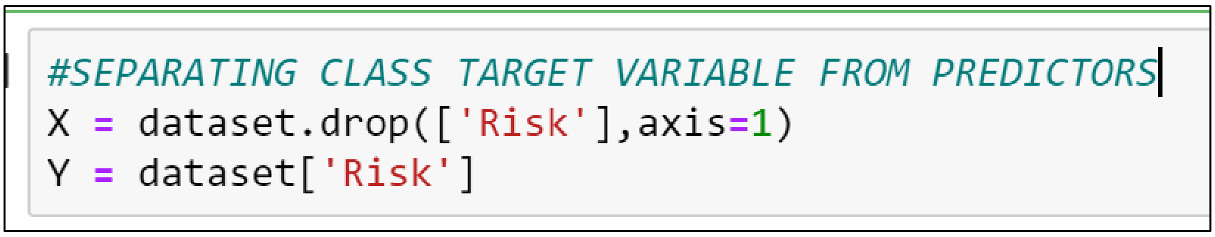
Figure 12: Dropping the three insignificant features

CHI SQUARE STATISTICAL TEST

To finalise on the best features to select for the model, a statistical test will be applied to select those that have the strongest relationship with the output variable, "**Risk**". The chi-squared(chi2) statistical test will be used to select the best features and the library to be used is the "SelectKBest" which will give us the best number of attributes that will be specified. This will give us the score of each of the 23 variables in relation to the output variable. The higher a score for a variable, the more important the feature is in our model, the lower a score, the less important it is in the model. For this to be done, the data will be divided into two, X representing the predictors, and Y representing the class variable. It is very necessary to separate the Target from the predictor attributes since X is used as input variables for a Machine Learning algorithm, while Y is the output variable that the algorithm is attempting to predict. This is often denoted as y = f(X) where f is the function used to calculate y from the input X.

- *SEPARATING INDEPENDENT ATTRIBUTES FROM DEPENDENT ATTRIBUTES(X AND Y)*

Below is command being used to separate predictors from the target variable, represented by X and Y respectively.

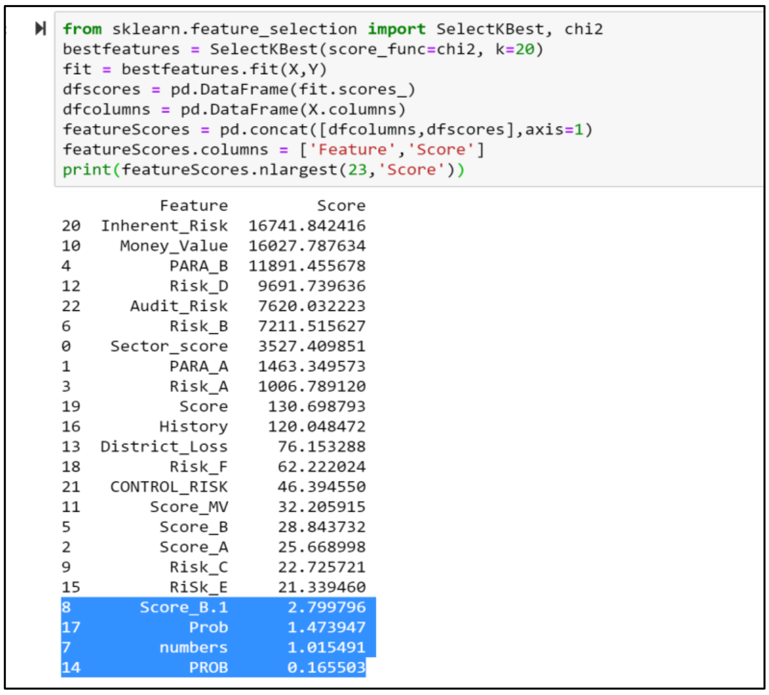


```
#SEPARATING CLASS TARGET VARIABLE FROM PREDICTORS
X = dataset.drop(['Risk'],axis=1)
Y = dataset['Risk']
```

Figure 13: Separating Independent variables from Dependent Variable

- *Scoring Variables using SelectKBest and Chisquare*

The below command is then run to calculate the scores of each of the features in X with the class variable in Y, with the output displayed in descending order to show the most important variables to the least important. (Fig 14).



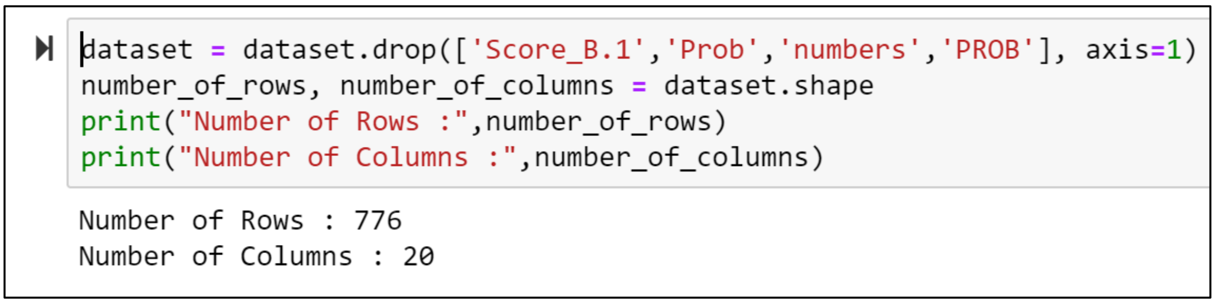
```
from sklearn.feature_selection import SelectKBest, chi2
bestfeatures = SelectKBest(score_func=chi2, k=20)
fit = bestfeatures.fit(X,Y)
dfscores = pd.DataFrame(fit.scores_)
dfcolumns = pd.DataFrame(X.columns)
featureScores = pd.concat([dfcolumns,dfscores],axis=1)
featureScores.columns = ['Feature','Score']
print(featureScores.nlargest(23,'Score'))
```

```
          Feature         Score
20  Inherent_Risk  16741.842416
10    Money_Value  16027.787634
4          PARA_B  11891.455678
12         Risk_D   9691.739636
22     Audit_Risk   7620.032223
6          Risk_B   7211.515627
0    Sector_score   3527.409851
1          PARA_A   1463.349573
3          Risk_A   1006.789120
19          Score    130.698793
16        History    120.048472
13  District_Loss     76.153288
18         Risk_F     62.222024
21   CONTROL_RISK     46.394550
11       Score_MV     32.205915
5         Score_B     28.843732
2         Score_A     25.668998
9          Risk_C     22.725721
15         RiSk_E     21.339460
8       Score_B.1      2.799796
17           Prob      1.473947
7         numbers      1.015491
14           PROB      0.165503
```

Figure 14: Scoring of Features

From the above screenshot provided, it can be seen the most important Feature is “**Inherent_Risk**” as it has the highest score of *16741.842416* and the least important is “PROB” with a score of *0.165503*. The last four Features *(“Score_B.1”,”Prob”,”numbers” and “PROB”)* can therefore be taken out of the dataset as it has scores less than the value of 10 and have no relation with the class variable. The reason for dropping less important features is because (1) The training time for training the model increases exponentially with the number of features and (2) The models have an increasing risk of overfitting with increasing number of features.

After dropping the last 4 features, the total number of features in the data have finally been reduced from 27 to 20 (19 predictors, 1 class variable) which can then be used to build our Support Vector Machine.



```
dataset = dataset.drop(['Score_B.1','Prob','numbers','PROB'], axis=1)
number_of_rows, number_of_columns = dataset.shape
print("Number of Rows :",number_of_rows)
print("Number of Columns :",number_of_columns)
```

```
Number of Rows : 776
Number of Columns : 20
```

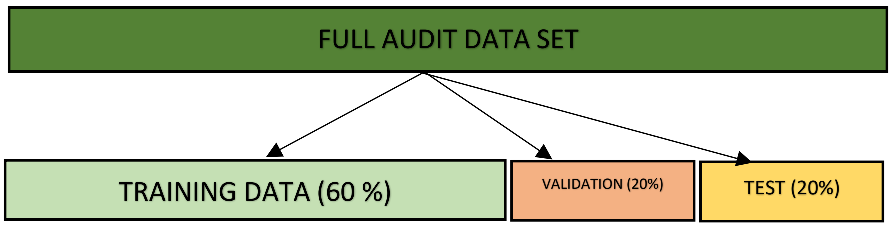
Figure 15: Total number of features left after feature Selection

Now our data contains only the most important features and can be used to build the model. However, the data will be split up first into ***training data***, ***validation data*** and ***test data***.

SPLITTING OF DATA

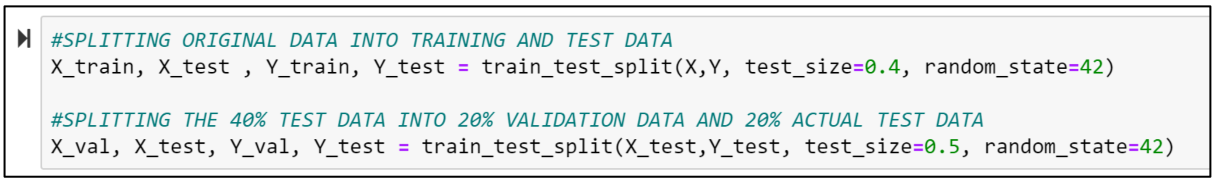
In the building of a model, two things can happen, (1) ***Overfitting*** – where the model fits too closely and acts extremely well on training, but extremely poor on test and unseen data, (2) ***Underfitting*** – where the model does not fit the training data and misses the trends in the data. To avoid such scenarios, it is always best to divide the data into parts to have the model fit and work extremely well on both seen and unseen data. To have this perfectly done, the data should be divided into three (3) parts:

1. **Training Data**: This is the actual data that is used to build the model. It contains a known output and the model learns on this data in order to be generalized to other data later on
2. **Validation Data**: This is used to evaluate the model that is built. It is used to provide an unbiased evaluation of a model fit on the training dataset while tuning model hyperparameters.
3. **Test Data**: This is used to test the final model’s prediction efficiency. It provides the gold standard that is used to evaluate the model and only used when the model is completely trained.



To have this done in python, I will use the “***train_test_split***” function from the scikit library, however, this function only divides the data set into two (training and testing data). So, to have all three datasets as mentioned earlier:

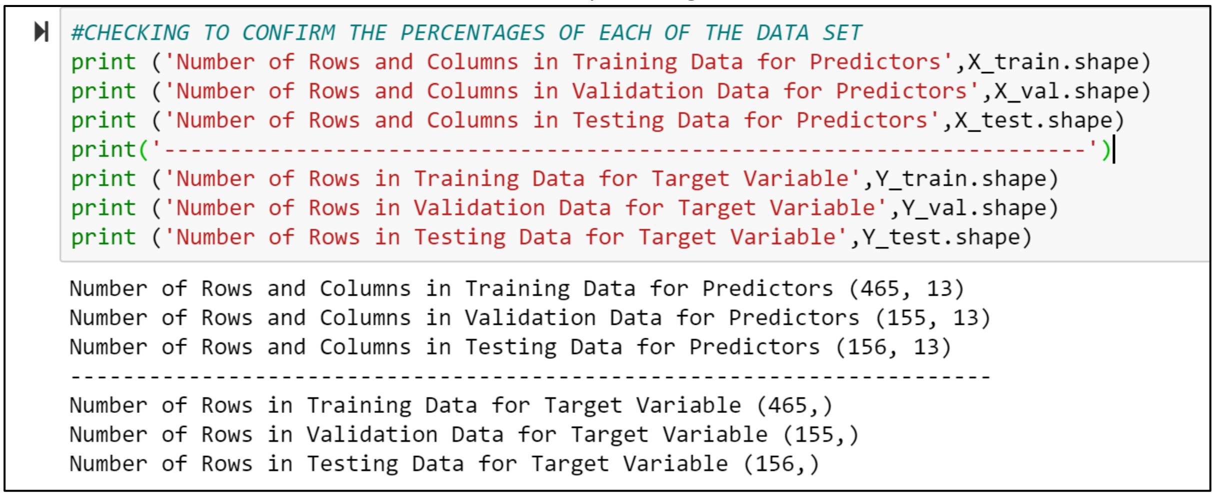
- The original dataset will first be divided into training data containing 60% of the data and the testing data will have the other 40%.
- The testing dataset which contains 40% of the data will then be divided into two parts again, with the first part being the validation data and will contain 20% of the data, and the other half being the actual test data, also having 20%. Command and output for this is shown below (Fig 16 and 17)



```
#SPLITTING ORIGINAL DATA INTO TRAINING AND TEST DATA
X_train, X_test , Y_train, Y_test = train_test_split(X,Y, test_size=0.4, random_state=42)

#SPLITTING THE 40% TEST DATA INTO 20% VALIDATION DATA AND 20% ACTUAL TEST DATA
X_val, X_test, Y_val, Y_test = train_test_split(X_test,Y_test, test_size=0.5, random_state=42)
```

Figure 16: Splitting of Original Dataset into training,validation and Test Data

60% of the data is split translating to 465 values for the training data and 20% translating to 155 ~ 156 each for validation and test data, as shown in output in Fig 17.



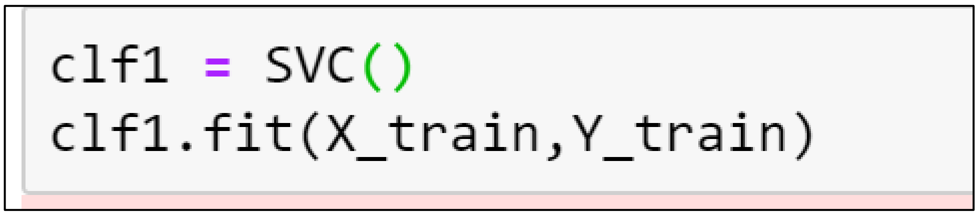
```
#CHECKING TO CONFIRM THE PERCENTAGES OF EACH OF THE DATA SET
print ('Number of Rows and Columns in Training Data for Predictors',X_train.shape)
print ('Number of Rows and Columns in Validation Data for Predictors',X_val.shape)
print ('Number of Rows and Columns in Testing Data for Predictors',X_test.shape)
print('-----------------------------------------------------------------------')
print ('Number of Rows in Training Data for Target Variable',Y_train.shape)
print ('Number of Rows in Validation Data for Target Variable',Y_val.shape)
print ('Number of Rows in Testing Data for Target Variable',Y_test.shape)
```

```
Number of Rows and Columns in Training Data for Predictors (465, 13)
Number of Rows and Columns in Validation Data for Predictors (155, 13)
Number of Rows and Columns in Testing Data for Predictors (156, 13)
-----------------------------------------------------------------------
Number of Rows in Training Data for Target Variable (465,)
Number of Rows in Validation Data for Target Variable (155,)
Number of Rows in Testing Data for Target Variable (156,)
```

Figure 17: Confirmation of all three data with the number of values

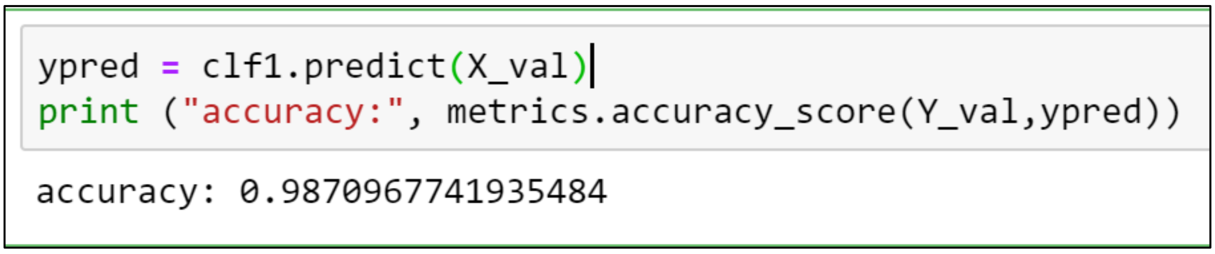
TRAINING/BUILDING THE SVM MODEL

Now, the SVM model is being built and trained using the training data. This is done by using SVC from scikit learn and running command below (Fig 18). By default, the kernel being used is 'rbf' which produced the accuracy measure of *0.9870967741935484* , approximately 98.7% when tested on the validation data, which is a better accuracy. (Fig 19)



```
clf1 = SVC()
clf1.fit(X_train,Y_train)
```

Figure 18: Building SVC model with training data

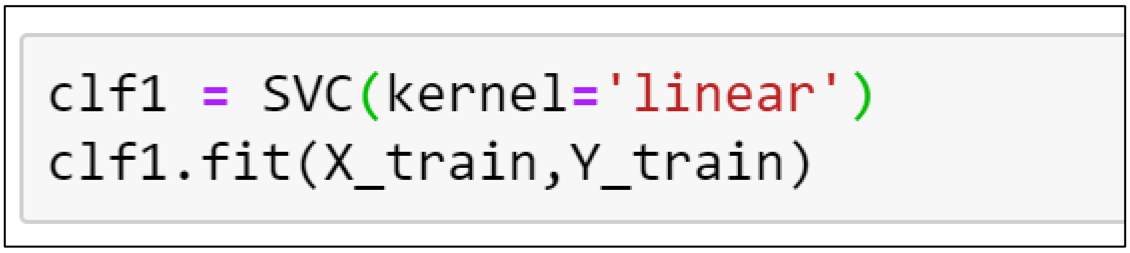


```
ypred = clf1.predict(X_val)
print ("accuracy:", metrics.accuracy_score(Y_val,ypred))

accuracy: 0.9870967741935484
```

Figure 19: Testing with Validation Data

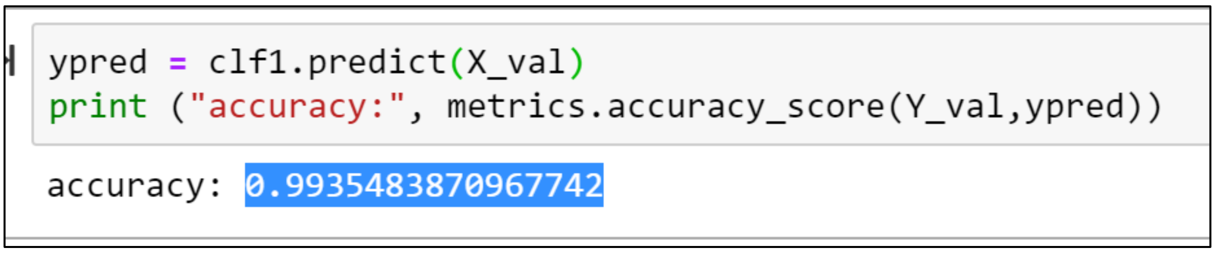
TUNING HYPERPARAMETERS

Upon tuning the parameters of the SVC classifier by making the kernel='linear', this produced a better accuracy of 0.9935483870967742, which is approximately 99.35% and much better than the previous classifier with rbf as the kernel. This explains that using linear makes a better model for our classification. (Fig 20,21)



```
clf1 = SVC(kernel='linear')
clf1.fit(X_train,Y_train)
```

Figure 20:Changing Kernel to Linear



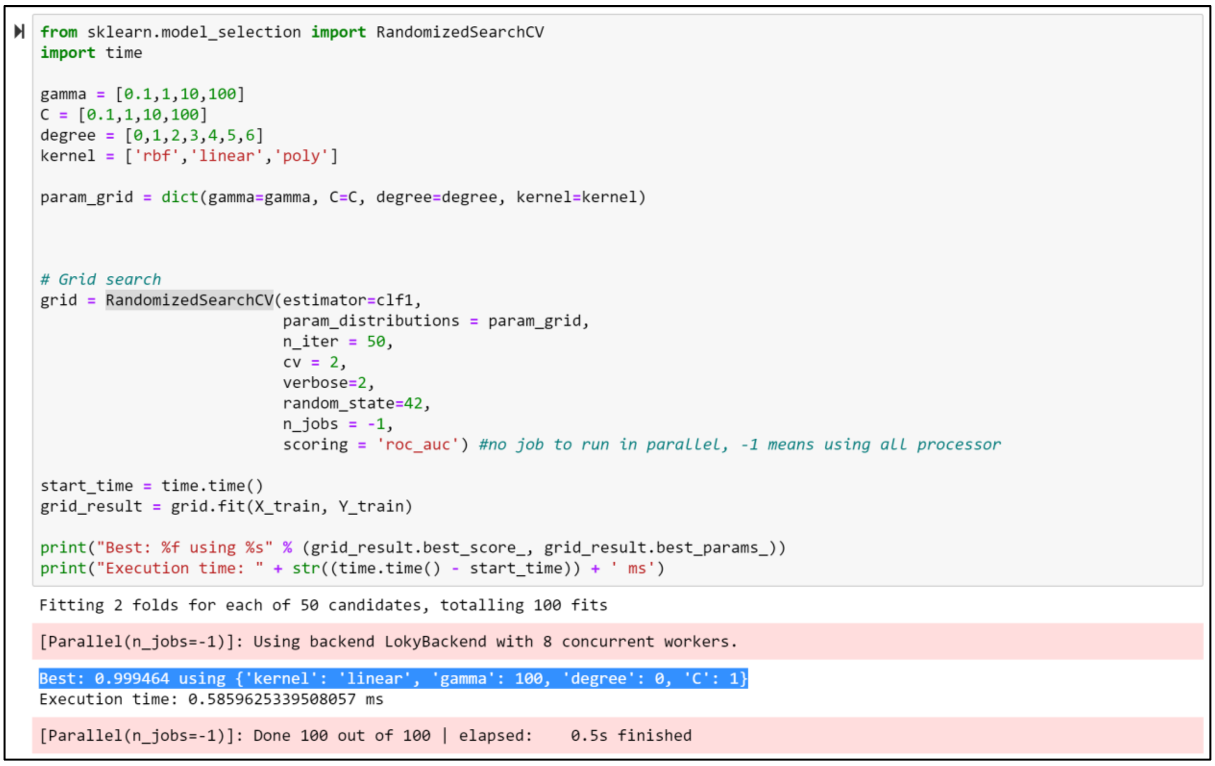
```
ypred = clf1.predict(X_val)
print ("accuracy:", metrics.accuracy_score(Y_val,ypred))

accuracy: 0.9935483870967742
```

Figure 21:Testing with Validation Data

KERNEL	ACCURACY
Rbf	0.9870967741935484 $\underset{-}{\sim}$ 98.7%
Linear	0.9935483870967742 $\underset{-}{\sim}$ 99.35%

Table 2: Accuracy of both kernels

However, by using code below, the best set of parameters for a potential of ***three different kernel types***(rbf,linear,poly), ***C to be either 0.1,1,10,100, gamma to be either 0.1,1,10,100*** and ***degree to be either 0,1,2,3,4,5,6***. It autonomously decided by the function of ***RandomizedSearchCV*** which helps determine best fit parameters. The outcome of this allowed an accuracy of 0.999464 which is shown in fig 22. The best parameter selected is gamma of 10, degree of 1 and C of 100. These settings allow the SVC to be non-linear at fitting the training data set, penalizes error term by 100 which may lead to overfitting and uses 1 degree of polynomial used to find the hyperplane to split the data. This then allowed for manual manipulation of the parameters to attempt to slightly increase the potential overall accuracy. The improvement and finalization of the tuning allowed the results to be examined.



```
from sklearn.model_selection import RandomizedSearchCV
import time

gamma = [0.1,1,10,100]
C = [0.1,1,10,100]
degree = [0,1,2,3,4,5,6]
kernel = ['rbf','linear','poly']

param_grid = dict(gamma=gamma, C=C, degree=degree, kernel=kernel)



# Grid search
grid = RandomizedSearchCV(estimator=clf1,
                          param_distributions = param_grid,
                          n_iter = 50,
                          cv = 2,
                          verbose=2,
                          random_state=42,
                          n_jobs = -1,
                          scoring = 'roc_auc') #no job to run in parallel, -1 means using all processor

start_time = time.time()
grid_result = grid.fit(X_train, Y_train)

print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
print("Execution time: " + str((time.time() - start_time)) + ' ms')
```

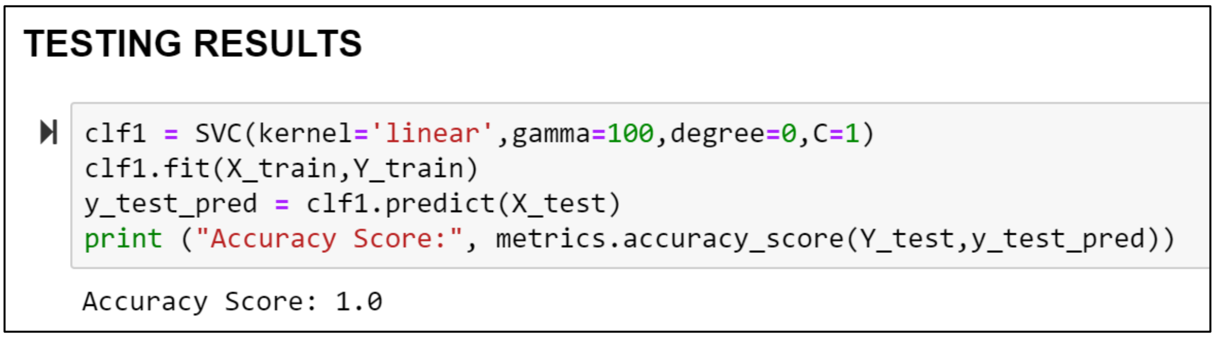
```
Fitting 2 folds for each of 50 candidates, totalling 100 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
Best: 0.999464 using {'kernel': 'linear', 'gamma': 100, 'degree': 0, 'C': 1}
Execution time: 0.5859625339508057 ms
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:    0.5s finished
```

Figure 22: Tuning of Hyperparameters

TESTING RESULTS

Now the model has been built successfully and the kernel to be used will be linear, C to be 1 and gamma to be 100, as it has a higher accuracy. To finally validate and conclude on our model, it will lastly be tested with the testing data which was separated earlier. As mentioned earlier, the testing data is used to test the final model's prediction efficiency. It provides the gold standard that is used to evaluate the model and only used when the model is completely trained. Below is the script being used to test the test data, the accuracy and the confusion matrix to show the differences in the predicted values and the actual values.

By testing our model with the parameters stated above with the test data, the model gave an accuracy of 1.0 ~ 100% which is very good and very accurate.



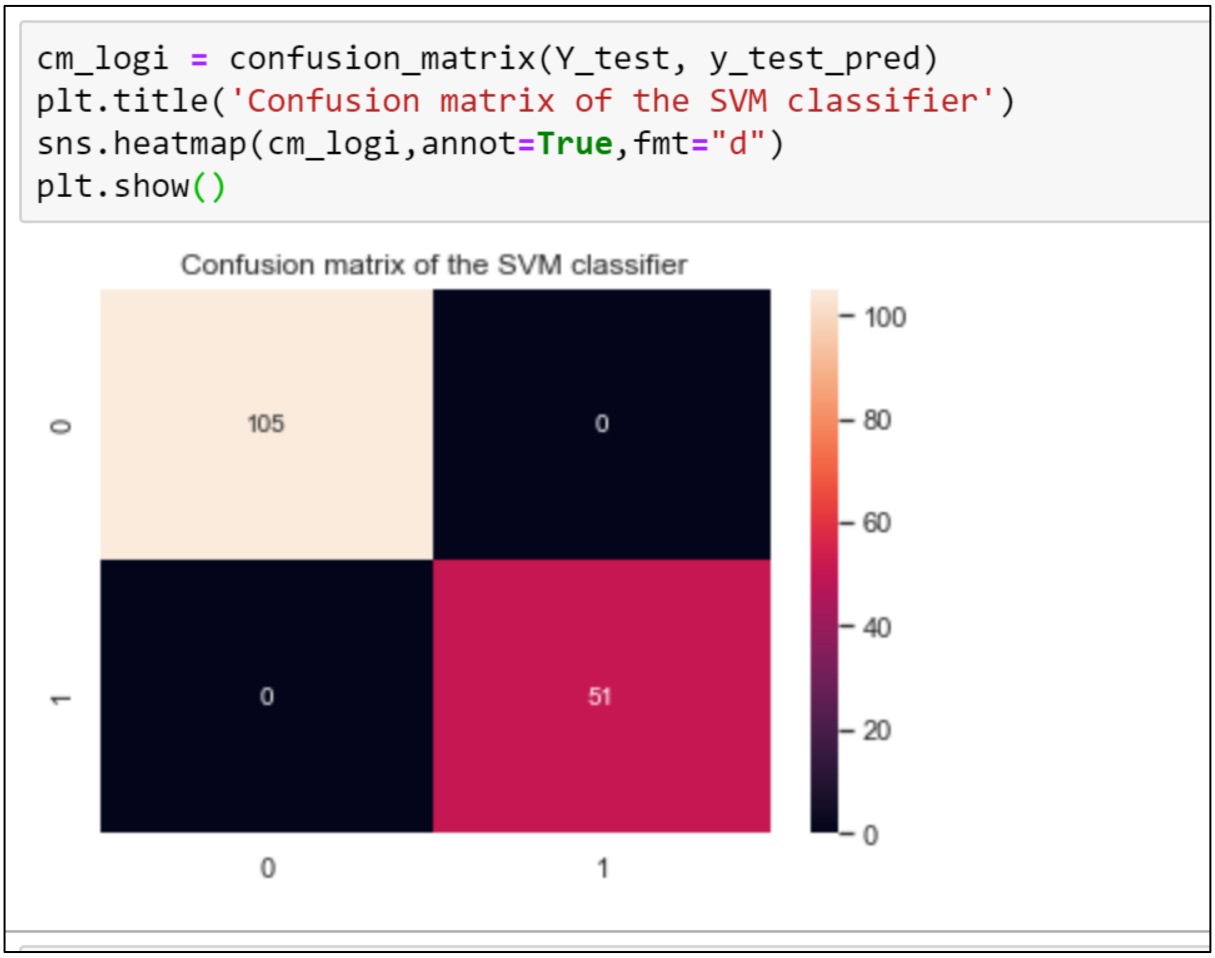
TESTING RESULTS

```
clf1 = SVC(kernel='linear',gamma=100,degree=0,C=1)
clf1.fit(X_train,Y_train)
y_test_pred = clf1.predict(X_test)
print ("Accuracy Score:", metrics.accuracy_score(Y_test,y_test_pred))
```

```
Accuracy Score: 1.0
```

Figure 23:Testing with Test Data

By using the confusion matrix below, it tells us that the model was able to predict accurately and correctly 105 out of the 105 that were stated to be 0 and 51 correctly out of the 51 that were stated to be 1. In conclusion, this confirms why the test of our model with the test data gave a 100% accuracy.



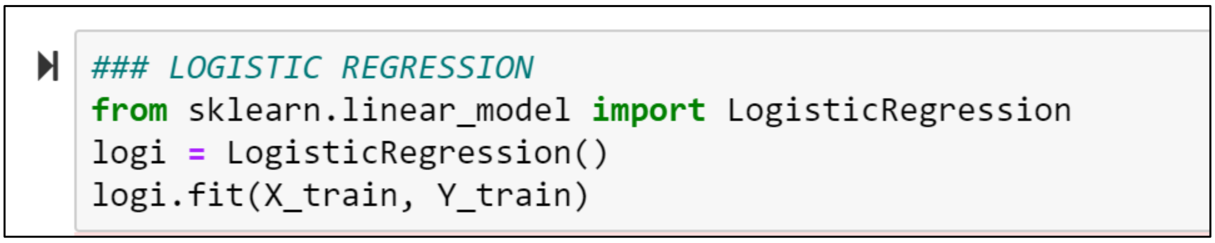
```
cm_logi = confusion_matrix(Y_test, y_test_pred)
plt.title('Confusion matrix of the SVM classifier')
sns.heatmap(cm_logi,annot=True,fmt="d")
plt.show()
```

Figure 24: Confusion matrix of SVM using Test Data

<u>DISCUSSION</u>

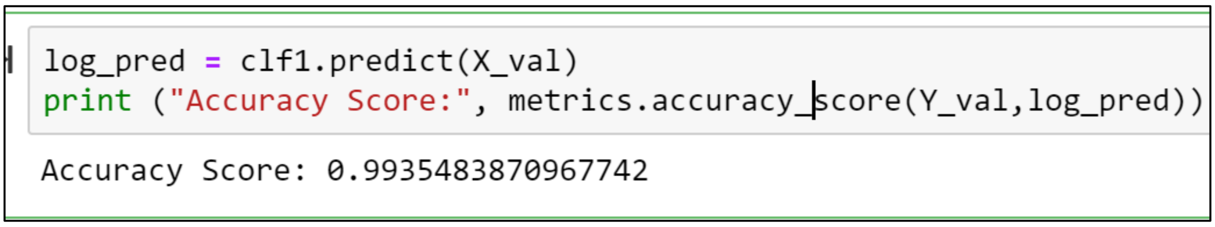
The next model to be built is a logistic regression model,

Firstly, the model is trained using the training data and tested with the validation data as shown below which gives an accuracy score of 0.99 ~ 99% which also shows the Logistic regression model is a good classifier for the Audit Data set. (Fig 25,26)



```
### LOGISTIC REGRESSION
from sklearn.linear_model import LogisticRegression
logi = LogisticRegression()
logi.fit(X_train, Y_train)
```

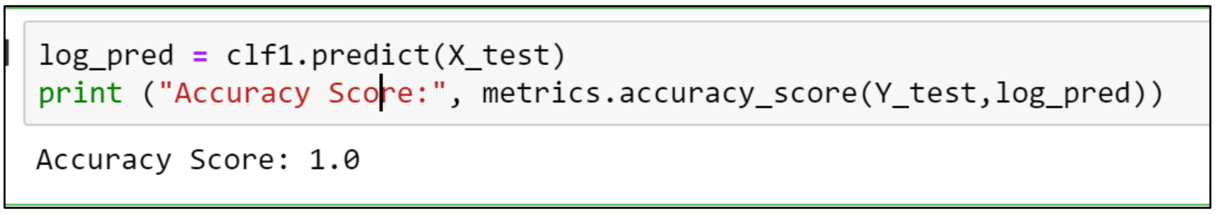
Figure 25:Logistic Regression Model



```
log_pred = clf1.predict(X_val)
print ("Accuracy Score:", metrics.accuracy_score(Y_val,log_pred))

Accuracy Score: 0.9935483870967742
```

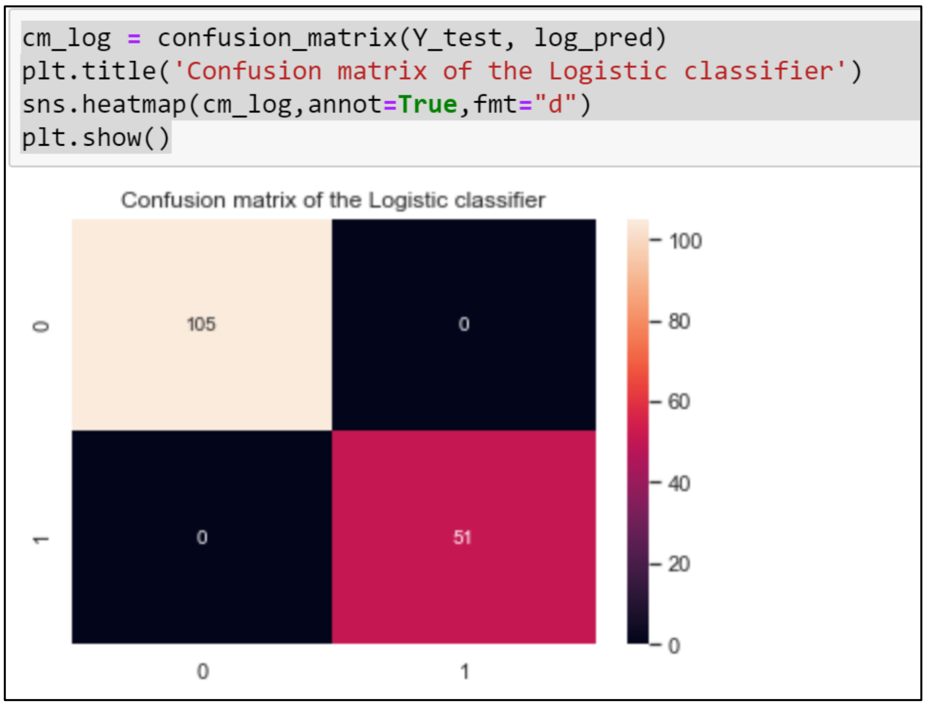
Figure 26: Testing With Validation Data

By running the model using the test data, it also produced an accuracy of 1.0 ~ meaning that Logistic regression performs equally well as the SVM classifier and could be used to predict well whether an organisation has a likelihood of committing fraud or not. (Fig 27)



```
log_pred = clf1.predict(X_test)
print ("Accuracy Score:", metrics.accuracy_score(Y_test,log_pred))

Accuracy Score: 1.0
```

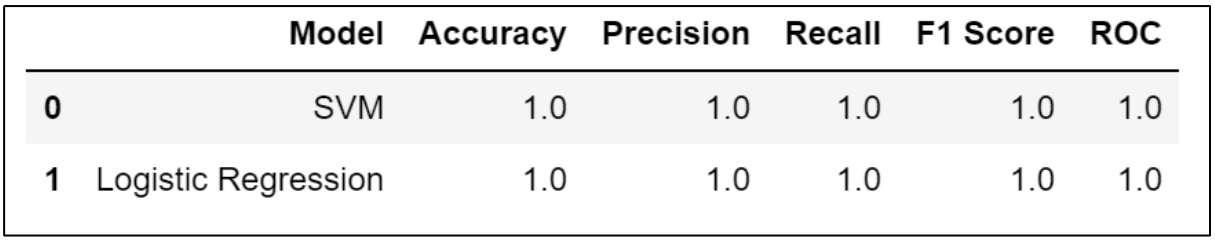
Figure 27:Testing With Test Data



```
cm_log = confusion_matrix(Y_test, log_pred)
plt.title('Confusion matrix of the Logistic classifier')
sns.heatmap(cm_log,annot=True,fmt="d")
plt.show()
```

Figure 28:Confusion matrix of Logistic Regression using Test Data

COMPARISON



	Model	Accuracy	Precision	Recall	F1 Score	ROC
0	SVM	1.0	1.0	1.0	1.0	1.0
1	Logistic Regression	1.0	1.0	1.0	1.0	1.0

Table 3: Metrics Score of Both Models

The performance of the proposed models is assessed on the basis of the confusion matrix as shown in Figure 24 and 28. Table 3 presents the different evaluation metrics calculated. When comparing the Support Vector Machines and the Logistic Regression, it is obvious that both models perform equally well and have a 100% accuracy in determining whether an organisation may or may not be participating in a fraud based on values of certain features. The goal of both SVM and logistic regression is to find the best line with the maximum likelihood of accurately classifying unknown points.

The Support Vector Machines algorithm is much more geometrically motivated. Instead of following a probabilistic method, it tries to find a particular optimum hyperplane dividing organizations that are likely to be involved in fraud and those that are not. It finds the hyperplane with the maximum margin in an N-dimensional space(N — the number of features) which categorises the data points separately

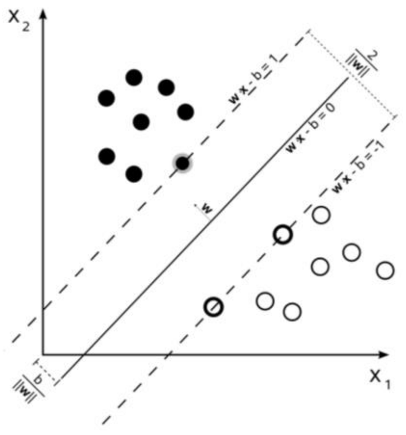


Figure 29: SVM

The focus of logistic regression is to maximise the data's probability. The further the data lies (on the correcy side) from the separating hyperplane, the happier the model is. The output of the linear function is taken in logistic regression and the result is squashed with the sigmoid feature within the scope of [0,1]. In our scenario, the output is either a 0 or 1.

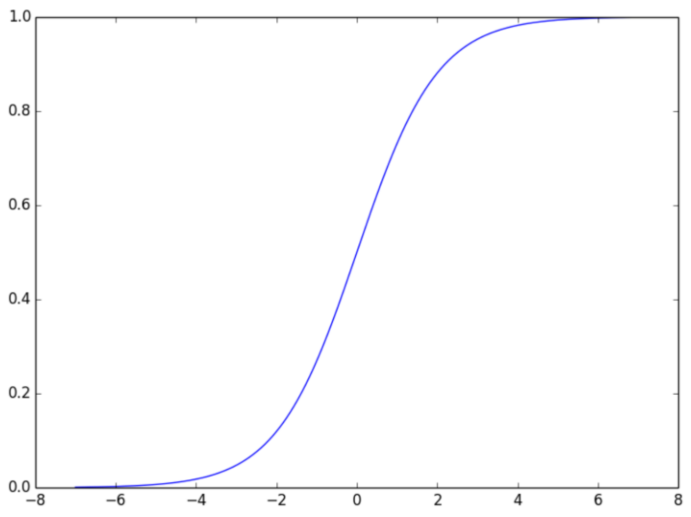


Figure 30: Logistic Regression

SVM is trying to maximise the distance between the closest support vectors while Logistic Regression is trying to maximise the probability of the posterior group. Thus, SVM finds a solution which is as fare as possible for the two categories while Logistic Regression does not have this property. Logistic Regression yields probabilistic values while SVM yields a 1 or 0. In a few terms, then, Logistic Regression does not make an absolute assumption and it does not presume the evidence are adequate to make a final decision. This may be a reasonable property if we want an approximation or if we don't have a strong level of data trust. In our case, where we are just classifying the organizations by either a 0 or a 1 to indicate the likeliness of fraud, the SVM will be a better model as it fits the business question we are trying to answer, as it gets the best separating hyperplane.

DATASOURCE. : https://archive.ics.uci.edu/ml/datasets/Audit+Data