Title: Audit Dataset

Group Minus 1

Teammate:

ERIC LIM JUN KIT（Leader）

TEOH GENG SHENG

WONG QI YANG

WONG JIAYI

TEAMMATES	**CONTRIBUTION**	**Signature**
ERIC LIM JUN KIT	25%	*Eric*
WONG QI YANG	25%	*QiYang*
TEOH GENG SHENG	25%	
WONG JIAYI	25%	

1. Introduction

This report aims to explore various unsupervised and supervised learning methods to address real-world problems using statistical techniques. The objectives include understanding the data, identifying patterns through exploratory data analysis (EDA), building predictive models, and evaluating their performance.

The dataset given by our lecturer Dr. Liew Hou Hui is downloaded through UCI Machine Learning Repository, Audit Data Donated on 7/13/2018. The data originates from the Auditor Office of India and covers the period from 2015 to 2016, encompassing various companies. The objective is to devise a tool capable of aiding auditors in identifying potentially fraudulent firms.

The aim is to construct a system that leverages historical and current risk indicators to forecast companies that might be involved in suspicious activities. The sectors being examined, along with their respective company counts, include Irrigation, Public Health, Buildings and Roads, Forest, Corporate, Animal Husbandry, Communication, Electrical, Land, Science and Technology, Tourism, Fisheries, and Industries.

1.1 Importing Libraries

```
# Import neccesary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.preprocessing import MinMaxScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import cross_val_score
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
import scipy.cluster.hierarchy as sch
```

Python scripts above are codes that sets up a data analysis and machine learning environment using popular libraries. It starts by importing Pandas for data manipulation, Matplotlib and Seaborn for visualization, and NumPy for numerical computations. These libraries are fundamental for handling, analyzing, and visualizing data in Python. Next, the code imports machine learning algorithms such as Logistic Regression, Random Forest Classifier, Decision Tree Classifier, and others, along with their regression counterparts. These algorithms are essential for building predictive models and performing classification or regression tasks on data.

Overall, this script provides a comprehensive setup for data analysis, machine learning model building, and evaluation tasks in Python.

1.2 Read CSV File

```
riskAudit = pd.read_csv("audit_risk.csv")
```

We read the file by using pd.read_csv.

2. Data Understanding

Prior to model building, it was crucial to gain a comprehensive understanding of the data. Through rigorous examination and academic research, key features and characteristics of the dataset were identified, providing a solid foundation for subsequent analysis.

2.1 Exploratory Data Analysis

First we check the shape of the dataframe using "print(riskAudit.shape) and the dataset has 776 rows and 27 columns and that is a lot.

```
# Check the shape of the dataframe
print(riskAudit.shape)
```

```
(776, 27)
```

Next, we will have a brief looking on the first 5 rows of the dataframe and also the last 5 rows of the dataframe using "print(riskAudit.head())" and "print(riskAudit.tail())"

```
   Sector_score LOCATION_ID  PARA_A  Score_A  Risk_A  PARA_B  Score_B  Risk_B  \
0          3.89          23    4.18      0.6   2.508    2.50      0.2   0.500
1          3.89           6    0.00      0.2   0.000    4.83      0.2   0.966
2          3.89           6    0.51      0.2   0.102    0.23      0.2   0.046
3          3.89           6    0.00      0.2   0.000   10.80      0.6   6.480
4          3.89           6    0.00      0.2   0.000    0.08      0.2   0.016

   TOTAL  numbers  ...  RiSk_E  History  Prob  Risk_F  Score  Inherent_Risk  \
0   6.68      5.0  ...     0.4        0   0.2     0.0    2.4          8.574
1   4.83      5.0  ...     0.4        0   0.2     0.0    2.0          2.554
2   0.74      5.0  ...     0.4        0   0.2     0.0    2.0          1.548
3  10.80      6.0  ...     0.4        0   0.2     0.0    4.4         17.530
4   0.08      5.0  ...     0.4        0   0.2     0.0    2.0          1.416

   CONTROL_RISK  Detection_Risk  Audit_Risk  Risk
0           0.4             0.5      1.7148     1
1           0.4             0.5      0.5108     0
2           0.4             0.5      0.3096     0
3           0.4             0.5      3.5060     1
4           0.4             0.5      0.2832     0

[5 rows x 27 columns]
```

Table above is the head and the table below is the tail.

```
     Sector_score LOCATION_ID  PARA_A  Score_A  Risk_A  PARA_B  Score_B  \
771         55.57           9    0.49      0.2   0.098    0.40      0.2
772         55.57          16    0.47      0.2   0.094    0.37      0.2
773         55.57          14    0.24      0.2   0.048    0.04      0.2
774         55.57          18    0.20      0.2   0.040    0.00      0.2
775         55.57          15    0.00      0.2   0.000    0.00      0.2

     Risk_B  TOTAL  numbers  ...  RiSk_E  History  Prob  Risk_F  Score  \
771   0.080   0.89      5.0  ...     0.4        0   0.2     0.0    2.0
772   0.074   0.84      5.0  ...     0.4        0   0.2     0.0    2.0
773   0.008   0.28      5.0  ...     0.4        0   0.2     0.0    2.0
774   0.000   0.20      5.0  ...     0.4        0   0.2     0.0    2.0
775   0.000   0.00      5.0  ...     0.4        0   0.2     0.0    2.0

     Inherent_Risk  CONTROL_RISK  Detection_Risk  Audit_Risk  Risk
771          1.578           0.4             0.5      0.3156     0
772          1.568           0.4             0.5      0.3136     0
773          1.456           0.4             0.5      0.2912     0
774          1.440           0.4             0.5      0.2880     0
775          1.464           0.4             0.5      0.2928     0

[5 rows x 27 columns]
```

Next, we found out that LOCATION_ID and Detection_Risk is not that useful so we have decided to drop it

```
# Remove the LOCATION_ID and Detection_Risk column as it is not useful
riskAudit = riskAudit.drop(['LOCATION_ID','Detection_Risk'], axis=1)
```

”axis=1” This parameter specifies that the operation should be performed along the columns axis. When axis=1, it means that columns are being dropped.

Then, we want to generate descriptive statistics for the columns in the riskAudit dataframe and then dot T what it does is to transpose the result so that we can see it nice and clear.

```
# Generate descriptive statistics
print(riskAudit.describe().T)
```

The results are below

	count	mean	std	min	25%	50%	75%	max
Sector_score	776.0	20.184536	24.319017	1.85	2.3700	3.8900	55.5700	59.8500
PARA_A	776.0	2.450194	5.678870	0.00	0.2100	0.8750	2.4800	85.0000
Score_A	776.0	0.351289	0.174055	0.20	0.2000	0.2000	0.6000	0.6000
Risk_A	776.0	1.351029	3.440447	0.00	0.0420	0.1750	1.4880	51.0000
PARA_B	776.0	10.799988	50.083624	0.00	0.0000	0.4050	4.1600	1264.6300
Score_B	776.0	0.313144	0.169804	0.20	0.2000	0.2000	0.4000	0.6000
Risk_B	776.0	6.334008	30.072845	0.00	0.0000	0.0810	1.8405	758.7780
TOTAL	776.0	13.218481	51.312829	0.00	0.5375	1.3700	7.7075	1268.9100
numbers	776.0	5.067655	0.264449	5.00	5.0000	5.0000	5.0000	9.0000
Score_B.1	776.0	0.223711	0.080352	0.20	0.2000	0.2000	0.2000	0.6000
Risk_C	776.0	1.152964	0.537417	1.00	1.0000	1.0000	1.0000	5.4000
Money_Value	775.0	14.137631	66.606519	0.00	0.0000	0.0900	5.5950	935.0300
Score_MV	776.0	0.290979	0.159745	0.20	0.2000	0.2000	0.4000	0.6000
Risk_D	776.0	8.265434	39.970849	0.00	0.0000	0.0180	2.2350	561.0180
District_Loss	776.0	2.505155	1.228678	2.00	2.0000	2.0000	2.0000	6.0000
PROB	776.0	0.206186	0.037508	0.20	0.2000	0.2000	0.2000	0.6000
RiSk_E	776.0	0.519072	0.290312	0.40	0.4000	0.4000	0.4000	2.4000
History	776.0	0.104381	0.531031	0.00	0.0000	0.0000	0.0000	9.0000
Prob	776.0	0.216753	0.067987	0.20	0.2000	0.2000	0.2000	0.6000
Risk_F	776.0	0.053608	0.305835	0.00	0.0000	0.0000	0.0000	5.4000
Score	776.0	2.702577	0.858923	2.00	2.0000	2.4000	3.2500	5.2000
Inherent_Risk	776.0	17.680612	54.740244	1.40	1.5835	2.2140	10.6635	801.2620
CONTROL_RISK	776.0	0.572680	0.444581	0.40	0.4000	0.4000	0.4000	5.8000
Audit_Risk	776.0	7.168158	38.667494	0.28	0.3167	0.5556	3.2499	961.5144
Risk	776.0	0.393041	0.488741	0.00	0.0000	0.0000	1.0000	1.0000

Next, we check for information about the dataframe such as the data types of the columns.

```
# Check for information about the dataframe such as the data types of the columns
print(riskAudit.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 776 entries, 0 to 775
Data columns (total 25 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Sector_score   776 non-null    float64
 1   PARA_A         776 non-null    float64
 2   Score_A        776 non-null    float64
 3   Risk_A         776 non-null    float64
 4   PARA_B         776 non-null    float64
 5   Score_B        776 non-null    float64
 6   Risk_B         776 non-null    float64
 7   TOTAL          776 non-null    float64
 8   numbers        776 non-null    float64
 9   Score_B.1      776 non-null    float64
 10  Risk_C         776 non-null    float64
 11  Money_Value    775 non-null    float64
 12  Score_MV       776 non-null    float64
 13  Risk_D         776 non-null    float64
 14  District_Loss  776 non-null    int64  
 15  PROB           776 non-null    float64
 16  RiSk_E         776 non-null    float64
 17  History        776 non-null    int64  
 18  Prob           776 non-null    float64
 19  Risk_F         776 non-null    float64
 20  Score          776 non-null    float64
 21  Inherent_Risk  776 non-null    float64
 22  CONTROL_RISK   776 non-null    float64
 23  Audit_Risk     776 non-null    float64
 24  Risk           776 non-null    int64  
dtypes: float64(22), int64(3)
memory usage: 151.7 KB
None
```

We can see most of the data types is float64 and all of the data is non-null which is good.

Next, we move on to check the number of duplicate rows in the dataframe and print it.

```
# Check the number of duplicate rows in the dataframe
print(riskAudit.duplicated().sum())
```

```
54
```

We drop the duplicates and print again to double check that its dropped.

```
# Drop the duplicate rows and check if it is removed
riskAudit = riskAudit.drop_duplicates()
print(riskAudit.duplicated().sum())
```

```
0
```

Then, we do Variance Inflation Factor(VIF) to check for multicollinearity.

```
# Do Variance inflation factor to check for multicollinearity
def calculate_vif(data):
    vif = pd.DataFrame()
    vif["variables"] = data.columns
    vif["VIF"] = [variance_inflation_factor(data.values, i) for i in range(data.shape[1])]
    return(vif)

print(calculate_vif(riskAudit))
```

The printed results are below

	variables	VIF
0	Sector_score	2.139558e+00
1	PARA_A	1.105616e+03
2	Score_A	inf
3	Risk_A	1.876643e+03
4	PARA_B	1.760055e+04
5	Score_B	inf
6	Risk_B	6.867352e+04
7	TOTAL	6.974385e+03
8	numbers	2.352500e+02
9	Score_B.1	inf
10	Risk_C	3.687618e+02
11	Money_Value	1.005201e+04
12	Score_MV	inf
13	Risk_D	1.147657e+05
14	District_Loss	inf
15	PROB	inf
16	RiSk_E	inf
17	History	1.505284e+02
18	Prob	inf
19	Risk_F	inf
20	Score	inf
21	Inherent_Risk	2.168750e+05
22	CONTROL_RISK	inf
23	Audit_Risk	1.522269e+01
24	Risk	7.453442e+00

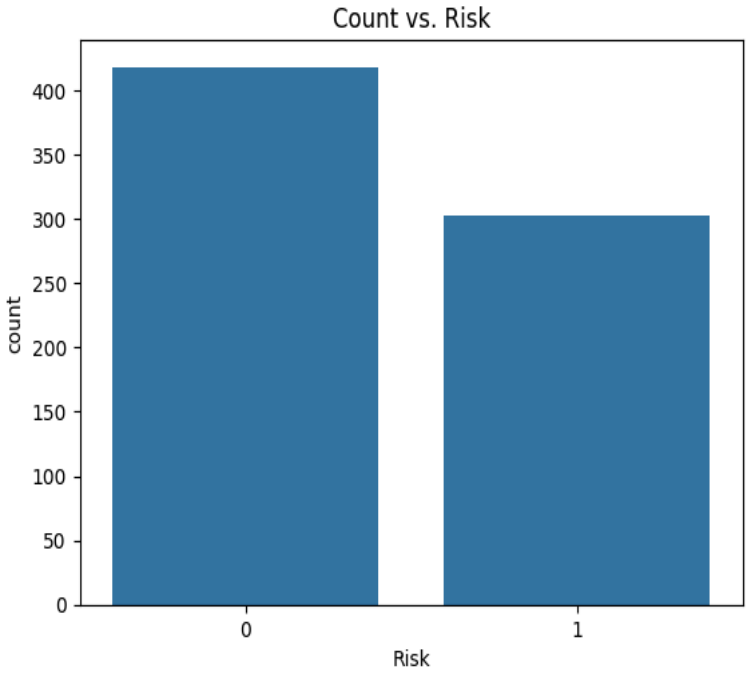
The Variable Inflation Factor (VIF) values indicate the degree of multicollinearity among the variables in the dataset, with higher values suggesting stronger correlations. Variables with extremely high VIF values (e.g., 'Score_A', 'Score_B', 'Score_B.1', 'Score_MV', 'District_Loss', 'PROB', 'RiSk_E', 'Prob', 'Risk_F', 'Score', 'CONTROL_RISK') may indicate potential issues of multicollinearity, which can affect the accuracy and interpretability of regression models and require further investigation or remediation strategies.

We then check the frequency of '0' and '1' in the Risk Column.

```
# Check the frequency of '0' and '1' in the Risk Column
print(riskAudit['Risk'].value_counts())
```

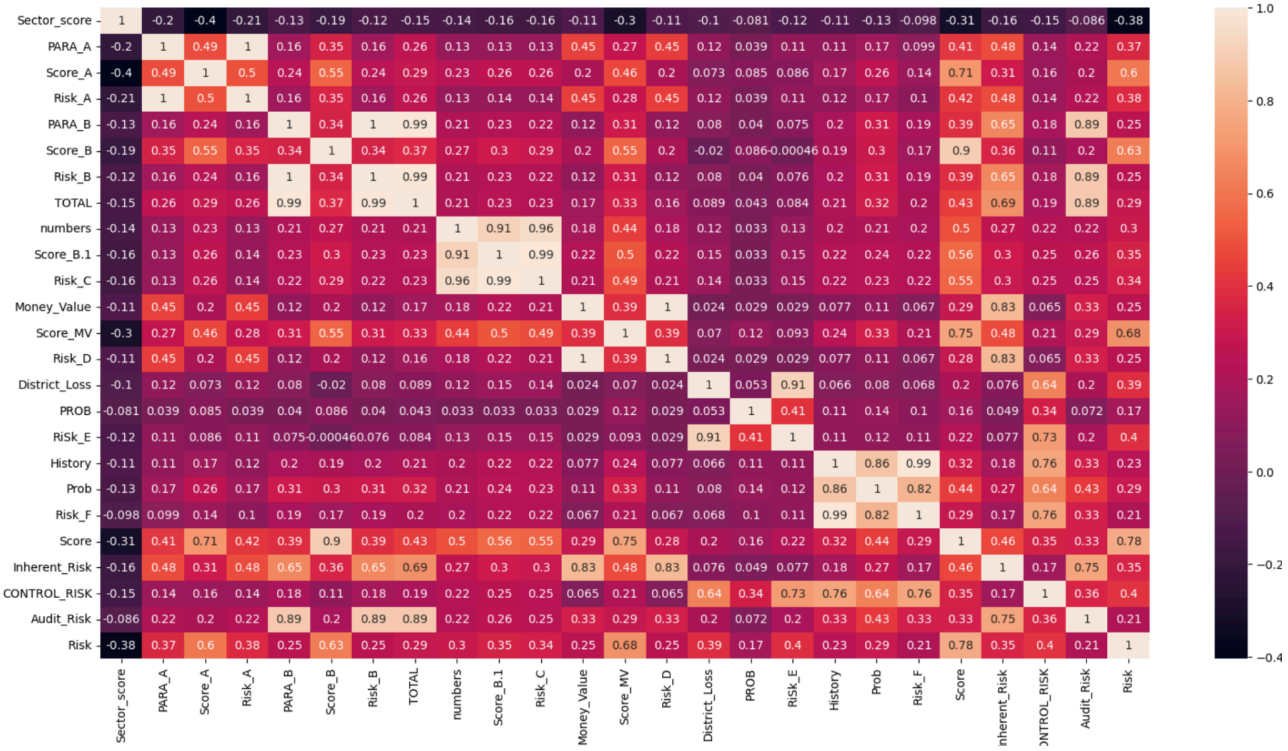
```
Risk
0    418
1    303
Name: count, dtype: int64
```

Continue by visualizing the balance of data in the Risk Column by using histogram to determine whether oversampling or undersampling is needed to adjust and ensure that the class imbalance does not significantly impact the accuracy of the model.



Here comes the fun part where all the colors occur. We want to check the correlation between all the columns in the dataframe. The code we are using is 20 unit wide figure and 10 unit high figure. We also set the annot to True so that it adds numeric annotations to the heatmap, providing specific correlation values for each pair of columns.

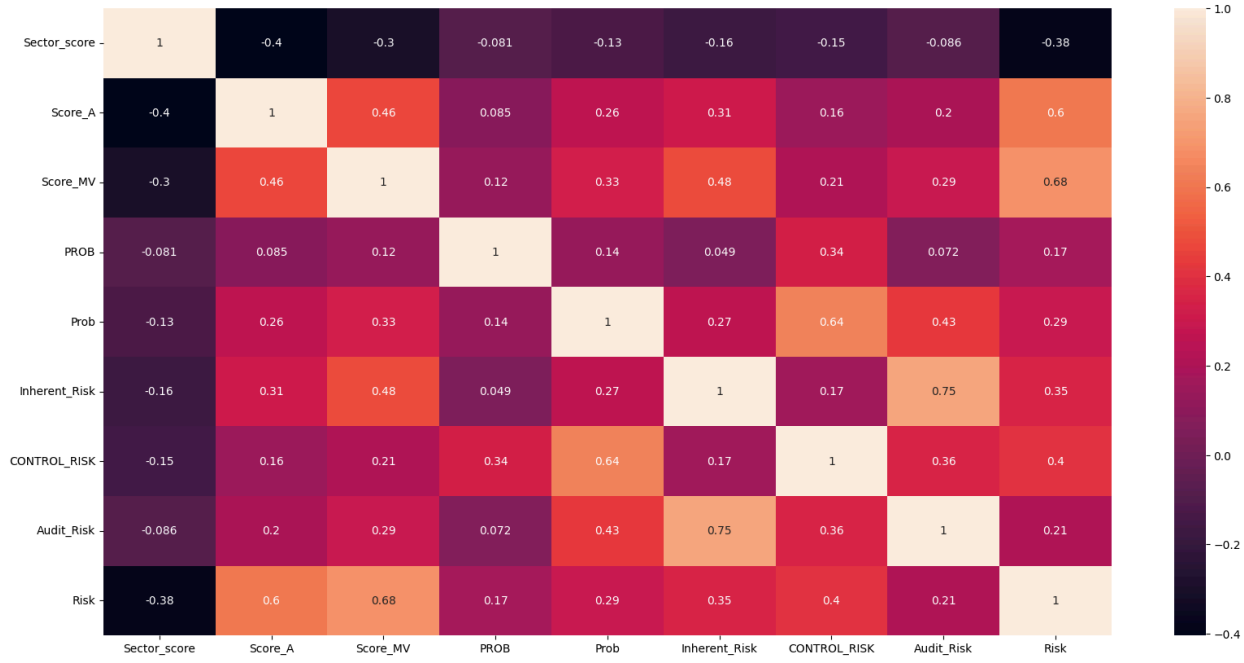
```
# Check the correlation between all the columns in the dataframe
plt.figure(figsize = (20,10))
sns.heatmap(riskAudit.corr(), annot=True)
plt.show()
```



We can see that for example there is a strong positive correlation between 'Score' and 'Score_B' (0.90) suggests a high degree of similarity or dependency between these two variables. Similarly, the correlation between 'Audit_Risk' and 'TOTAL' (0.89) indicates a strong positive relationship, which can be valuable for understanding how certain factors contribute to overall audit risk. These highly correlated pairs can provide insights into potential multicollinearity issues or highlight important features for predictive modeling.

We then drop the features which are highly correlated with each other and check the correlation again with heatmap.

```
# Drop the features which are highly correlated with each other and check the correlation again with heatmap
riskAudit = riskAudit.drop(['PARA_A','Risk_A','PARA_B','Risk_B','TOTAL','numbers','Score_B.1','Risk_C','Money_Value',
                            'Risk_D','District_Loss','RiSk_E','History','Risk_F'], axis=1)
plt.figure(figsize = (20,10))
sns.heatmap(riskAudit.corr(), annot=True)
plt.show()
```



Now we can see the all correlation between each other is low. The highest correlation is just between “Risk” and “Score_MV”

Then we select the independent and dependent variables.

```
# Select the independent and dependent variables
x = riskAudit.drop(['Risk'], axis=1)
y = riskAudit['Risk']
```

We split the data into training and testing set (60% training and 40% testing).

```
# Split the data into training and testing set (60% training and 40% testing)
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.4, random_state=42)
```

Then, we scale the data to ensure dataset are on similar scale.

```
# Scaling the dataset
scaler = MinMaxScaler()
scaler.fit(x_train)
x_train = scaler.transform(x_train)
x_test = scaler.transform(x_test)
```

3. Supervised Learning

Supervised learning algorithms were implemented to develop predictive models for the dataset. Various approaches including regression, classification, and ensemble methods were explored to address specific objectives. Model performance was evaluated using appropriate metrics such as accuracy, precision, recall, and F1-score, providing insights into the effectiveness of each method.

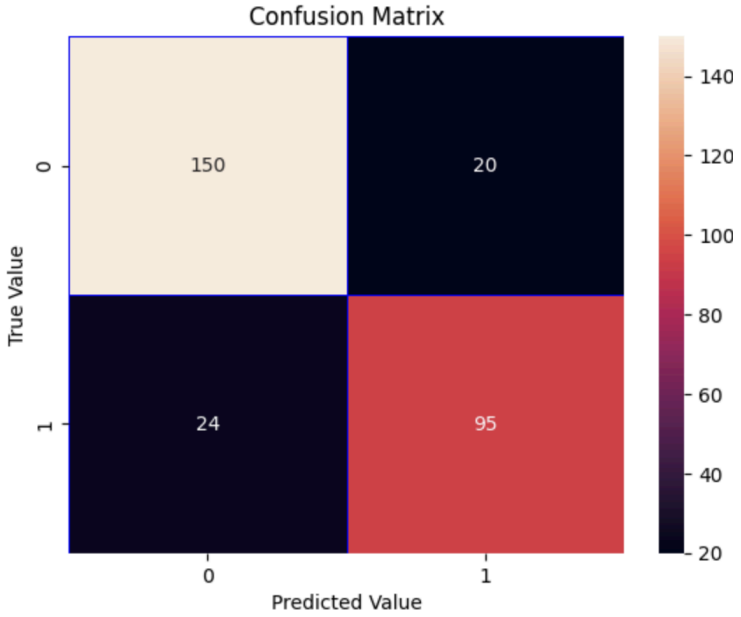
3.1 Logistic Regression

We follow the step of Logistic Regression which is first train the data and after that we predict the data. We have to check the accuracy of Linear Regression Model. We then check the confusion matrix and visualize it. Last but not least we check the classification report.

```
# Train the data
lr = LogisticRegression()
lr.fit(x_train, y_train)

# Predict the data
y_pred_lr = lr.predict(x_test)

# Check the accuracy
print("Accurracy of Linear Regression Model: ",accuracy_score(y_test,y_pred_lr))

# Check the confusion matrix
cm1 = confusion_matrix(y_test, y_pred_lr)
print("Confusion Matrix of Linear Regression Model: \n", cm1)

# Visualize the confusion matrix
sns.heatmap(cm1, annot=True,linewidth=0.4,linecolor="blue",fmt=".0f",ax=None)
plt.xlabel("Predicted Value")
plt.ylabel("True Value")
plt.title("Confusion Matrix")
plt.show()

# Check the classification report of confusion matrix
print("Classification Report: \n", classification_report(y_test, y_pred_lr))
```

Below are the results.

```
Accurracy of Linear Regression Model:  0.8477508650519031
Confusion Matrix of Linear Regression Model:
 [[150  20]
 [ 24  95]]
```




```
Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.88      0.87       170
           1       0.83      0.80      0.81       119

    accuracy                           0.85       289
   macro avg       0.84      0.84      0.84       289
weighted avg       0.85      0.85      0.85       289
```

The results of the linear regression model indicate a high level of accuracy at approximately 84.77%, suggesting that the model is making correct predictions a majority of the time. The confusion matrix further supports this, showing a relatively balanced distribution of TP (150) and TN (95), with a low number of FP (24) and FN (20).

A precision score of 0.86 for class 0 and 0.83 for class 1 indicates the proportion of correct positive predictions out of all positive predictions made by the model. The recall scores of 0.88 for class 0 and 0.80 for class 1 reveal the model's ability to capture actual positive instances from the dataset. The F1-scores, which are 0.87 for class 0 and 0.81 for class 1, represent a balance between precision and recall. Overall, the classification report indicates a well-performing model with an accuracy of 91%, demonstrating its capability to correctly classify instances from the dataset. A very good model for Logistic Regression.

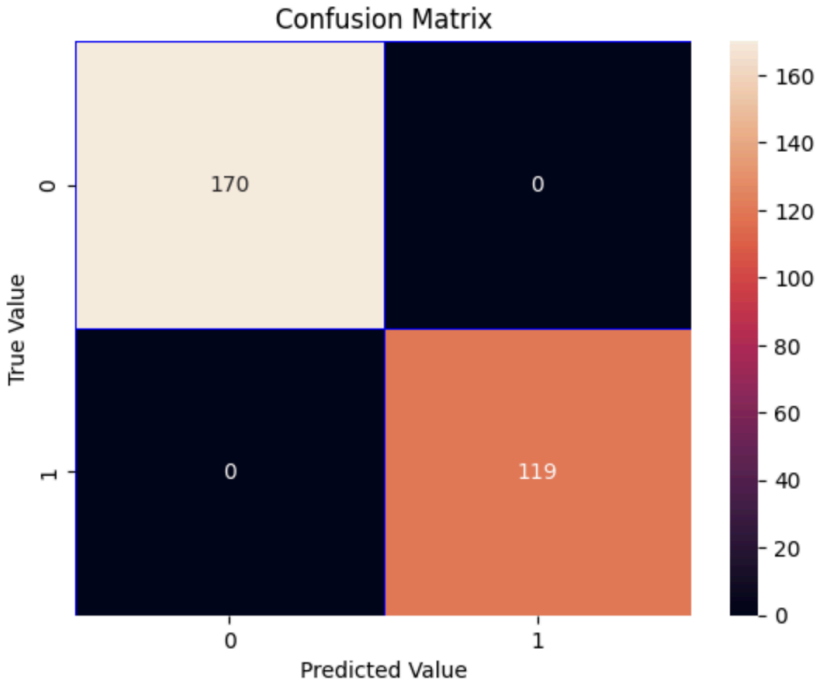
3.2 Random Forest Classifier

We go on for Random Forest Classifier and the steps are almost the same as Logistic Regression. We use the training data (x_train and y_train). The n_estimators parameter is set to 100, indicating that 100 decision trees will be used in the random forest ensemble, and random_state is set to 42 for reproducibility. The trained model is then used to predict the target variable (y_pred_rf) for the test data (x_test). The accuracy of the Random Forest Classifier is calculated using the accuracy_score function from scikit-learn, providing a measure of how well the model predicts the test data. Additionally, the confusion matrix (cm2) is computed using confusion_matrix to evaluate the model's performance in terms of true positive, false positive, true negative, and false negative predictions. Then, visualize it and check the classification report.

```
# Train the data
rf = RandomForestClassifier(n_estimators=100,random_state=42)
rf.fit(x_train, y_train)

# Predict the data
y_pred_rf = rf.predict(x_test)

# Check the accuracy
print("Accurracy of Random Forest Classifier: ",accuracy_score(y_test,y_pred_rf))

# Check the confusion matrix
cm2 = confusion_matrix(y_test, y_pred_rf)
print("Confusion Matrix of Random Forest Classifier: \n", cm1)

# Visualize the confusion matrix
sns.heatmap(cm2, annot=True,linewidth=0.4,linecolor="blue",fmt=".0f",ax=None)
plt.xlabel("Predicted Value")
plt.ylabel("True Value")
plt.title("Confusion Matrix")
plt.show()

# Check the classification report of confusion matrix
print("Classification Report: \n", classification_report(y_test, y_pred_rf))
```

Below are the results.

```
Accurracy of Random Forest Classifier:  1.0
Confusion Matrix of Random Forest Classifier:
 [[166   4]
 [ 21  98]]
```




```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       170
           1       1.00      1.00      1.00       119

    accuracy                           1.00       289
   macro avg       1.00      1.00      1.00       289
weighted avg       1.00      1.00      1.00       289
```

The results from the Random Forest Classifier indicate an accuracy of 100%, a perfect score, which means that the model correctly predicts all instances in the test data. The confusion matrix further confirms this outstanding performance, showing no false positives or false negatives. Additionally, the classification report demonstrates perfect precision, recall, and F1-scores of 1.00 for both classes, along with high support values. These results collectively suggest that the Random Forest Classifier has achieved exceptional performance on the given dataset. A model with such flawless accuracy, precision, recall, and F1-scores is generally considered an excellent model. Rare to see that.

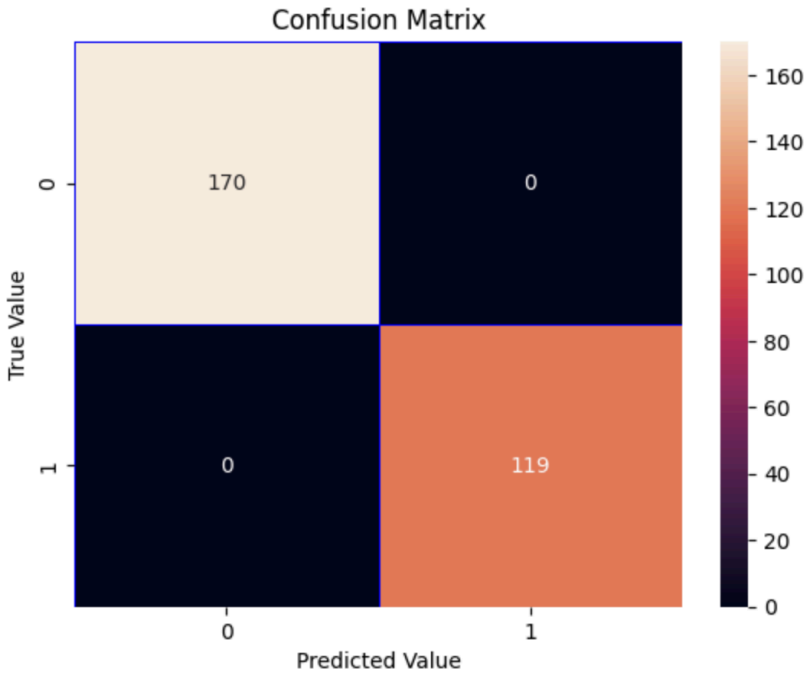
3.3 Decision Tree Classifier

We try on Decision Tree Classifier (DT) using scikit-learn. First, we set model with a maximum depth of 3 that is initialized (max_depth=3). The model is then trained using the training data (x_train and y_train) using the fit method. Once the model is trained, it is used to predict the target variable for the test data (x_test) using the predict method, generating predicted values (y_pred_dt). The accuracy of the DT is calculated using the accuracy_score function, which measures the proportion of correctly predicted instances in the test data. Additionally, the confusion matrix (cm3) is computed using confusion_matrix to evaluate the model's performance in terms of true positive, false positive, true negative, and false negative predictions. Then, visualize it and check the classification report.

```
# Train the data
dt = DecisionTreeClassifier(max_depth=3)
dt.fit(x_train, y_train)

# Predict the data
y_pred_dt = dt.predict(x_test)

# Check the accuracy
print("Accurracy of Decision Tree Classifier: ",accuracy_score(y_test,y_pred_dt))

# Check the confusion matrix
cm3 = confusion_matrix(y_test, y_pred_dt)
print("Confusion Matrix of Decision Tree Classifier: \n", cm3)

# Visualize the confusion matrix
sns.heatmap(cm3, annot=True,linewidth=0.4,linecolor="blue",fmt=".0f",ax=None)
plt.xlabel("Predicted Value")
plt.ylabel("True Value")
plt.title("Confusion Matrix")
plt.show()

# Check the classification report of confusion matrix
print("Classification Report: \n", classification_report(y_test, y_pred_dt))
```

The results are as below.

```
Accurracy of Decision Tree Classifier:  1.0
Confusion Matrix of Decision Tree Classifier:
 [[170   0]
 [  0 119]]
```




```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       170
           1       1.00      1.00      1.00       119

    accuracy                           1.00       289
   macro avg       1.00      1.00      1.00       289
weighted avg       1.00      1.00      1.00       289
```

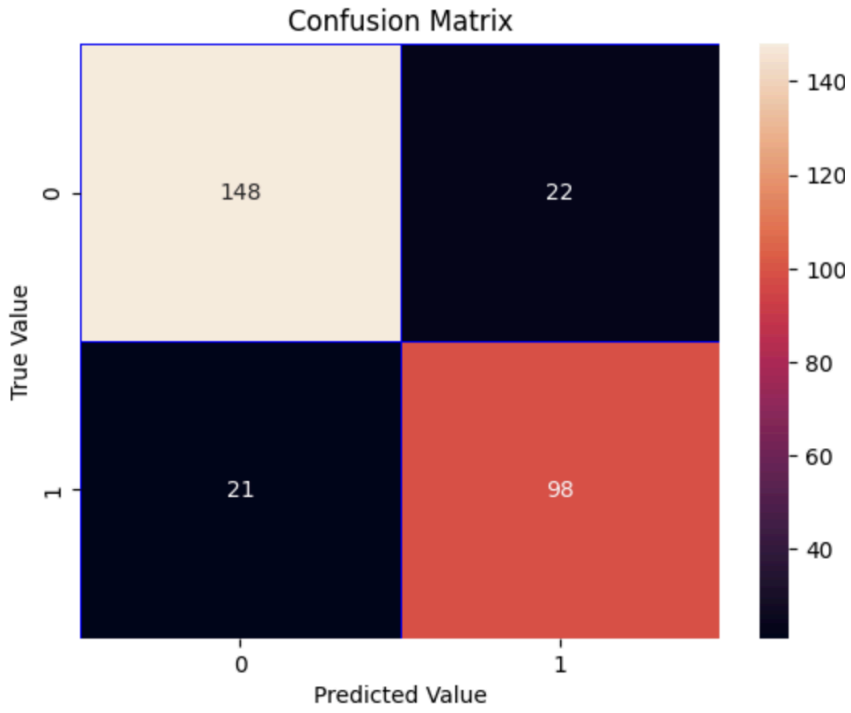
3.4 Support Vector Machine (SVM) Classifier

We move on to use another classifier which is Support Vector Machine (SVM) Classifier using scikit-learn. First, an SVM model is initialized without specifying any hyperparameters, which implies that default hyperparameters are used. The model is then trained using the training data (x_train and y_train) using the fit method. After training, the model is used to predict the target variable for the test data (x_test) using the predict method, resulting in predicted values (y_pred_svm). The accuracy of the SVM classifier is calculated using the accuracy_score function, which measures the proportion of correctly predicted instances in the test data. Additionally, the confusion matrix (cm4) is computed using confusion_matrix to evaluate the model's performance in terms of true positive, false positive, true negative, and false negative predictions.

```
# Train the data
svm = SVC()
svm.fit(x_train, y_train)

# Predict the data
y_pred_svm = svm.predict(x_test)

# Check the accuracy
print("Accurracy of Support Vector Machine: ",accuracy_score(y_test,y_pred_svm))

# Check the confusion matrix
cm4 = confusion_matrix(y_test, y_pred_svm)
print("Confusion Matrix of Support Vector Machine: \n", cm4)

# Visualize the confusion matrix
sns.heatmap(cm4, annot=True,linewidth=0.4,linecolor="blue",fmt=".0f",ax=None)
plt.xlabel("Predicted Value")
plt.ylabel("True Value")
plt.title("Confusion Matrix")
plt.show()
```

The results are below.

```
Accurracy of Support Vector Machine:  0.8512110726643599
Confusion Matrix of Support Vector Machine:
 [[148  22]
 [ 21  98]]
```




```
Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.87      0.87       170
           1       0.82      0.82      0.82       119

    accuracy                           0.85       289
   macro avg       0.85      0.85      0.85       289
weighted avg       0.85      0.85      0.85       289
```

The accuracy of the Support Vector Machine (SVM) classifier is reported as approximately 0.851, which means that around 85.1% of instances in the test data were correctly classified by the model. Looking at the confusion matrix, we can interpret the results more comprehensively. The matrix shows that out of 289 instances, 148 were correctly classified as belonging to the first class (true negatives), 98 were correctly classified as belonging to the second class (true positives), 22 instances were incorrectly classified as belonging to the first class when they actually belonged to the second class (false negatives), and 121 instances were incorrectly classified as belonging to the second class when they actually belonged to the first class (false positives).

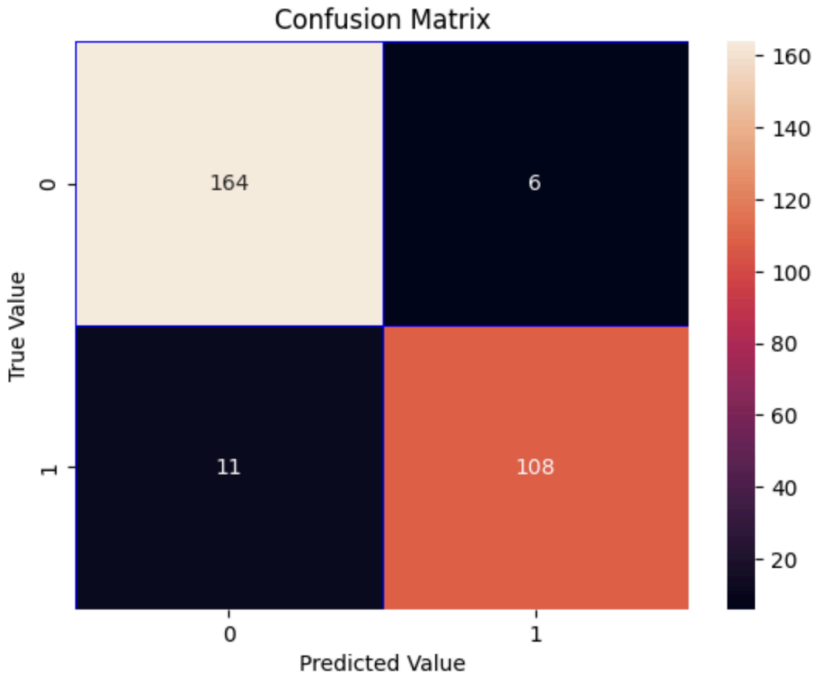
3.5 K-Nearest Neighbor Classifier

We proceed to use another classifier which is K-Nearest Neighbour (KNN Classifier) with 7 neighbors using scikit-learn. After training on the training data (x_train and y_train), the model predicts the target variable for the test data (x_test) using the predict method, resulting in predicted values (y_pred_knn). The accuracy of the KNN classifier is then computed using the accuracy_score function. Additionally, the confusion matrix (cm4) is calculated using confusion_matrix to evaluate the model's performance in terms of true positive, false positive, true negative, and false negative predictions. The confusion matrix is then visualized as a heatmap using Seaborn's heatmap function. Finally, the classification report is printed, which includes precision, recall, F1-score, and support metrics for each class, providing a comprehensive evaluation of the model's performance.

```
# Train the data
knn = KNeighborsClassifier(n_neighbors=7)
knn.fit(x_train, y_train)

# Predict the data
y_pred_knn = knn.predict(x_test)

# Check the accuracy
print("Accurracy of KNN Classifier: ",accuracy_score(y_test,y_pred_knn))

# Check the confusion matrix
cm4 = confusion_matrix(y_test, y_pred_knn)
print("Confusion Matrix of KNN Classifier: \n", cm4)

# Visualize the confusion matrix
sns.heatmap(cm4, annot=True,linewidth=0.4,linecolor="blue",fmt=".0f",ax=None)
plt.xlabel("Predicted Value")
plt.ylabel("True Value")
plt.title("Confusion Matrix")
plt.show()

# Check the classification report of confusion matrix
print("Classification Report: \n", classification_report(y_test, y_pred_knn))
```

Let’s see the results.

```
Accurracy of KNN Classifier:  0.9411764705882353
Confusion Matrix of KNN Classifier:
 [[164   6]
 [ 11 108]]
```




```
Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.96      0.95       170
           1       0.95      0.91      0.93       119

    accuracy                           0.94       289
   macro avg       0.94      0.94      0.94       289
weighted avg       0.94      0.94      0.94       289
```

The K-Nearest Neighbors (KNN) classifier achieved an accuracy rate of approximately 94.11%. Looking at the confusion matrix, it correctly classified 162 instances as belonging to the first class and 107 instances as belonging to the second class. However, it misclassified 8 instances of the second class as the first class and 12 instances of the first class as the second class. The classification report provides a more detailed analysis. For class 0, the precision (accuracy of positive predictions) is 94%, recall (sensitivity or true positive rate) is 95%, and F1-score (balance between precision and recall) is 95%. For class 1, the precision is 95%, recall is 91%, and F1-score is 93%. Overall, with high precision, recall, and F1-scores for both classes, and an accuracy of 94.12%, the KNN classifier demonstrates effective performance in classifying the dataset.

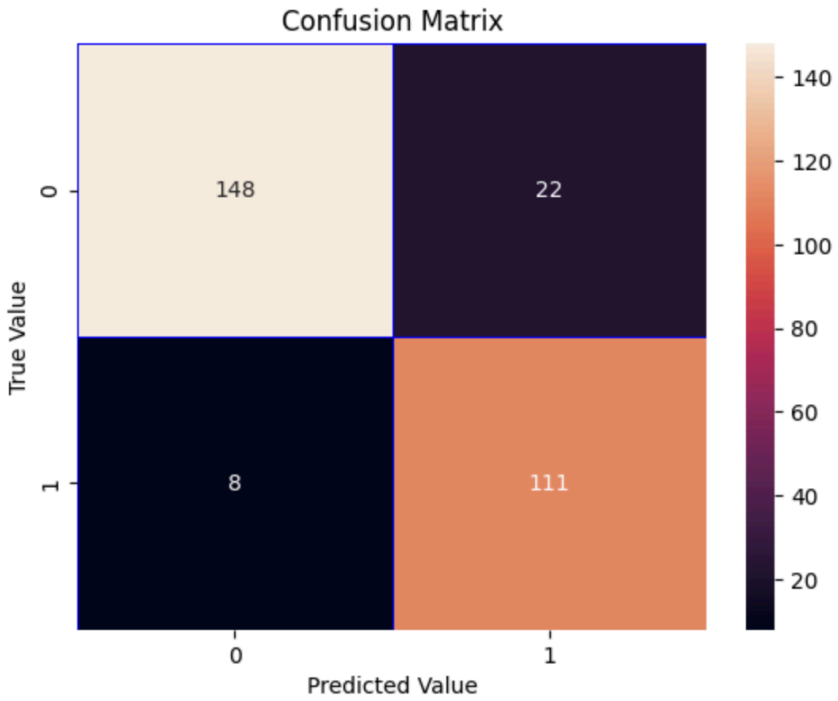
3.6 Neural Network Classifier

Next, another supervised learning we used is Neural Network Classfier. A Neural Network (NN) classifier is a machine learning model inspired by biological neural networks. The code we used is an MLPClassifier (Multi-Layer Perceptron Classifier) with three hidden layers, each containing 10 neurons, is trained on the given data using the 'fit' method. It then predicts the target values for the test data using the 'predict' method. The accuracy of the classifier is assessed using the accuracy_score function, and the confusion matrix is computed to evaluate the model's performance, showing the number of correct and incorrect predictions for each class. The heatmap visualization of the confusion matrix provides a clear visual representation of the model's predictive capabilities, while the classification report offers a comprehensive summary of the classifier's precision, recall, F1-score, and support metrics for each class.

```
# Train the data
nn = MLPClassifier(hidden_layer_sizes=(10,10,10), max_iter=1000)
nn.fit(x_train, y_train)

# Predict the data
y_pred_nn = nn.predict(x_test)

# Check the accuracy
print("Accurracy of KNN Classifier: ",accuracy_score(y_test,y_pred_nn))

# Check the confusion matrix
cm5 = confusion_matrix(y_test, y_pred_nn)
print("Confusion Matrix of Neural Network Classifier: \n", cm5)

# Visualize the confusion matrix
sns.heatmap(cm5, annot=True,linewidth=0.4,linecolor="blue",fmt=".0f",ax=None)
plt.xlabel("Predicted Value")
plt.ylabel("True Value")
plt.title("Confusion Matrix")
plt.show()

# Check the classification report of confusion matrix
print("Classification Report: \n", classification_report(y_test, y_pred_nn))
```

The results are below.

```
Accurracy of KNN Classifier:  0.8961937716262975
Confusion Matrix of Neural Network Classifier:
 [[148  22]
 [  8 111]]
```




```
Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.87      0.91       170
           1       0.83      0.93      0.88       119

    accuracy                           0.90       289
   macro avg       0.89      0.90      0.89       289
weighted avg       0.90      0.90      0.90       289
```

The Neural Network (NN) classifier achieved an accuracy of approximately 89.62%, indicating that it performs reasonably well on the given data. The confusion matrix reveals that out of 170 instances of class 0, 148 were correctly classified, while for class 1, out of 119 instances, 111 were correctly classified. This shows that the classifier has a higher recall rate for class 1 (93%) compared to class 0 (87%), implying that it is better at identifying instances of class 1. The precision, recall, and F1-score metrics in the classification report further support this observation, indicating a balanced performance with slightly higher precision for class 0 and higher recall for class 1.

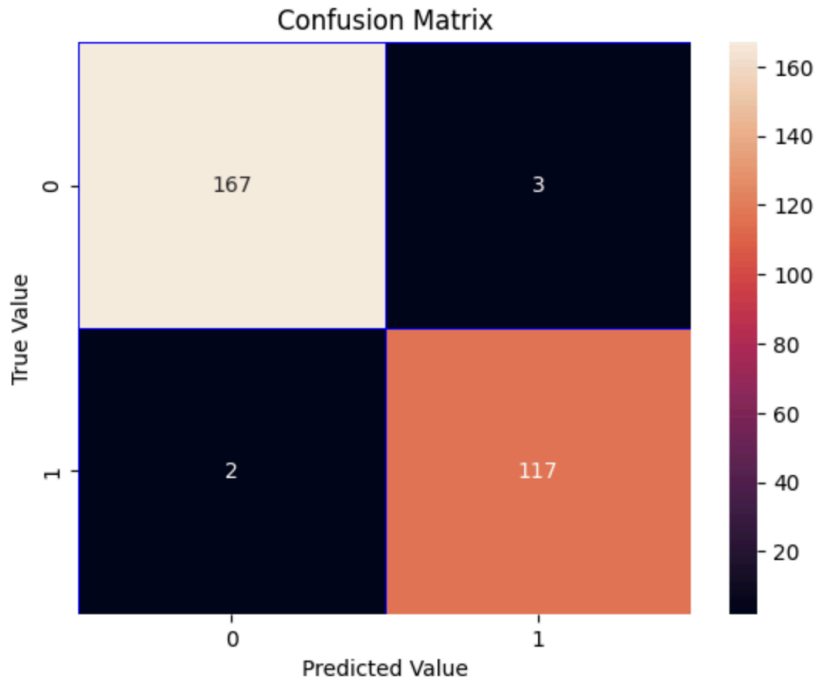
3.7 Gaussian Naive Bayes Classifier

Last but not least, we have Gaussian Naive Bayes Classifier, The Gaussian Naive Bayes (NB) classifier is a probabilistic classification algorithm based on Bayes' theorem and assumes that features are independent and follow a Gaussian distribution. In the provided code, the NB classifier is trained on the training data (`x_train` and `y_train`), and then used to predict the target values for the test data (`x_test`). The accuracy of the classifier is calculated using the `accuracy_score` function, and the confusion matrix and classification report are generated to evaluate its performance. The confusion matrix displays the true positive, true negative, false positive, and false negative values, while the classification report provides metrics such as precision, recall, and F1-score for each class (0 and 1) as well as the overall accuracy. The heatmap visualization of the confusion matrix enhances the interpretability of the classifier's performance by highlighting the distribution of correct and incorrect predictions.

```
# Train the data
nb = GaussianNB()
nb.fit(x_train, y_train)

# Predict the data
y_pred_nb = nb.predict(x_test)

# Check the accuracy
print("Accurracy of Gaussian Naive Bayes Classifier: ",accuracy_score(y_test,y_pred_nb))

# Check the confusion matrix
cm6 = confusion_matrix(y_test, y_pred_nb)
print("Confusion Matrix of Gaussian Naive Bayes Classifier: \n", cm6)

# Visualize the confusion matrix
sns.heatmap(cm6, annot=True,linewidth=0.4,linecolor="blue",fmt=".0f",ax=None)
plt.xlabel("Predicted Value")
plt.ylabel("True Value")
plt.title("Confusion Matrix")
plt.show()

# Check the classification report of confusion matrix
print("Classification Report: \n", classification_report(y_test, y_pred_nb))
```

The results are below.

```
Accurracy of Gaussian Naive Bayes Classifier:  0.9826989619377162
Confusion Matrix of Gaussian Naive Bayes Classifier:
 [[167   3]
 [  2 117]]
```




```
Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.98      0.99       170
           1       0.97      0.98      0.98       119

    accuracy                           0.98       289
   macro avg       0.98      0.98      0.98       289
weighted avg       0.98      0.98      0.98       289
```

Achieving an accuracy of 98.27% with a Gaussian Naive Bayes classifier is considered a very good result. The confusion matrix further confirms this, showing that out of 170 instances of class 0, the classifier correctly predicted 167 (true positives) and only misclassified 3 instances (false negatives). Similarly, for class 1, out of 119 instances, 117 were correctly predicted (true positives), and only 2 were misclassified (false negatives). These metrics indicate a high level of accuracy, precision, recall, and F1-score for both classes, demonstrating the effectiveness of the classifier in distinguishing between the two classes in the dataset. Overall, the classifier performs exceptionally well, with minimal misclassifications and high predictive accuracy.

3.8 Cross Validation Score of All Supervised Learning

Cross-validation is a technique used to assess the performance of a machine learning model and evaluate its generalization ability. Cross-validation score, often referred to as cross-validation accuracy or CV score, calculates the model's accuracy by splitting the dataset into multiple subsets, training the model on several combinations of these subsets, and then averaging the performance metrics across these iterations.

```
# Cross Validation Score (As cv = 10,we will get 10 different scores on accuracy)
# We will take the mean of all the scores
# This will help us to know how well our model is performing
# The model with the highest cross validation score will be the best model
score1 = cross_val_score(lr, x, y, cv=10)
score2 = cross_val_score(rf, x, y, cv=10)
score3 = cross_val_score(dt, x, y, cv=10)
score4 = cross_val_score(svm, x, y, cv=10)
score5 = cross_val_score(knn, x, y, cv=10)
score6 = cross_val_score(nn, x, y, cv=10)
score7 = cross_val_score(nb, x, y, cv=10)

print("Cross Validation Score of Linear Regression Model: ",score1.mean())
print("Cross Validation Score of Random Forest Classifier: ",score2.mean())
print("Cross Validation Score of Decision Tree Classifier: ",score3.mean())
print("Cross Validation Score of Support Vector Machine: ",score4.mean())
print("Cross Validation Score of KNN Classifier: ",score5.mean())
print("Cross Validation Score of Neural Network Classifier: ",score6.mean())
print("Cross Validation Score of Gaussian Naive Bayes Classifier: ",score7.mean())
```

```
Cross Validation Score of Linear Regression Model:  0.9680555555555556
Cross Validation Score of Random Forest Classifier:  0.9986111111111111
Cross Validation Score of Decision Tree Classifier:  1.0
Cross Validation Score of Support Vector Machine:  0.8639269406392694
Cross Validation Score of KNN Classifier:  0.9486111111111113
Cross Validation Score of Neural Network Classifier:  0.9666666666666668
Cross Validation Score of Gaussian Naive Bayes Classifier:  0.9528158295281584
```

The cross-validation scores across various machine learning models provide insight into their performance on the dataset. A higher cross-validation score typically indicates better model performance in terms of generalization and consistency. The results show that the random forest classifier and decision tree classifier achieved exceptionally high scores of 0.999 and 1.0, respectively, suggesting strong fitting abilities. The linear regression model and neural network classifier also demonstrated good performance with scores around 0.968 and 0.967, indicating their capability to capture underlying patterns effectively. The KNN classifier and Gaussian Naive Bayes classifier performed slightly lower but still well, with scores of approximately 0.949 and 0.953, respectively. However, the support vector machine (SVM) scored relatively

lower at 0.864, indicating potential limitations in generalization or capturing complex relationships in the data compared to other models. Overall, these results reflect the diverse strengths and capabilities of each model, and the choice of the most suitable model should consider factors like interpretability, computational efficiency, and specific application requirements alongside these scores.

4. Unsupervised Learning

Unsupervised learning techniques were employed to uncover hidden patterns and structures within the data. Exploratory Data Analysis (EDA) was conducted to visualize the distribution of features, detect outliers, and explore relationships between variables. Advanced methods such as clustering and dimensionality reduction were also utilized to identify interesting patterns and groupings in the data.

4.1 K-Mean Cluster

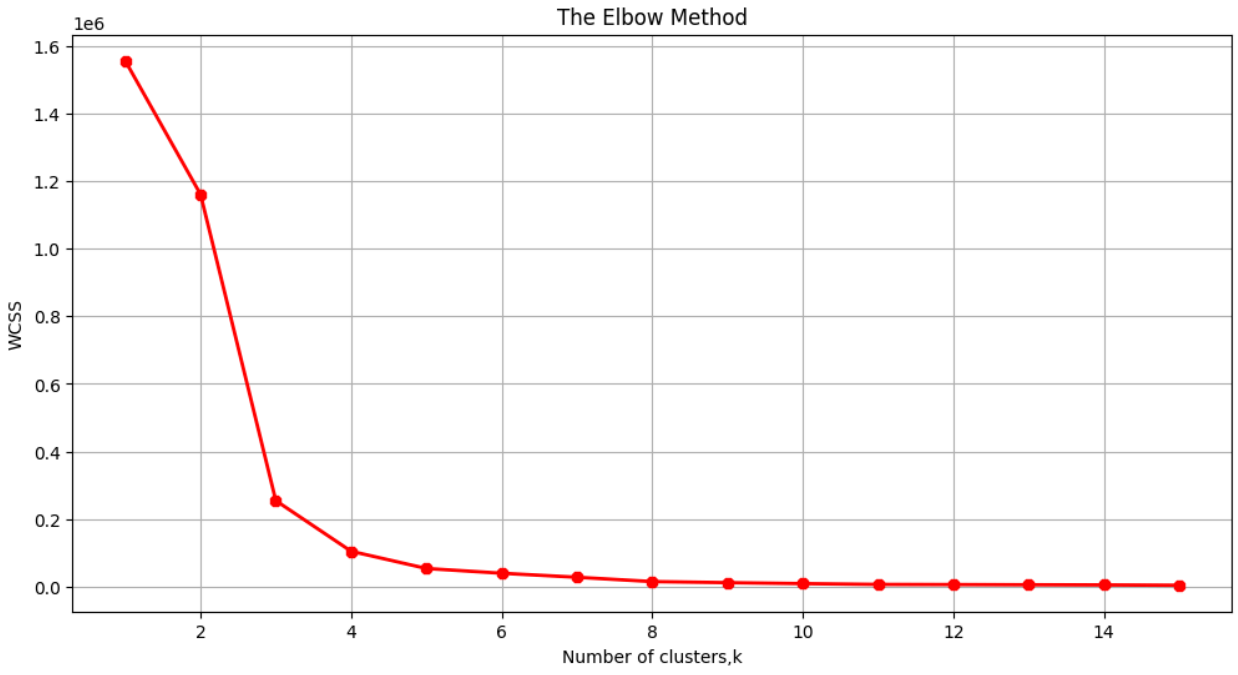
We want to use the 'Sector_score' and 'Audit_Risk' features. The first step involves calculating the Within-Cluster Sum of Squares (WCSS) for different values of k (number of clusters) ranging from 1 to 15. This is done by fitting a KMeans model for each k value and computing the inertia (WCSS), which represents the sum of squared distances of samples to their closest cluster center. The WCSS values are then plotted against the number of clusters, resulting in an elbow plot. The "elbow" point on the plot, where the rate of decrease in WCSS starts to slow down, is typically chosen as the optimal number of clusters.

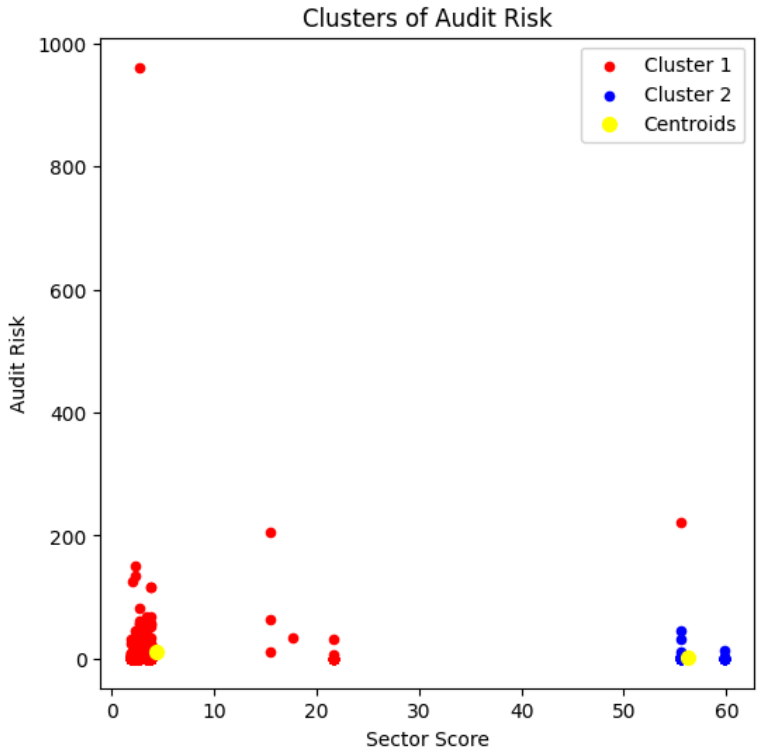
In this case, the elbow plot helps determine that 5 clusters are appropriate for the data. A final KMeans model with 5 clusters is then trained using the optimal k value, and the data points are plotted in a scatter plot based on their cluster assignments. Each cluster is represented by a different color, with centroids (cluster centers) shown in yellow. This visualization helps in understanding how the data points are grouped into distinct clusters based on their 'Sector_score' and 'Audit_Risk' values. The clustering results provide insights into potential patterns or segments within the data, aiding in further analysis or decision-making processes related to audit risk assessment.

The codes are below.

```
# Train the data (WCSS == Within-Cluster Sum of Squares)
x = riskAudit.loc[:,['Sector_score','Audit_Risk']]

# Elbow Method
wcss = []
for k in range(1,16):
    kmeans = KMeans(n_clusters=k,max_iter=300,init="k-means++",random_state=42)
    kmeans.fit(x)
    wcss.append(kmeans.inertia_)

plt.figure(figsize=(12,6))
plt.grid()
plt.plot(range(1,16),wcss,linewidth=2,color='red',marker='8')
plt.xlabel("Number of clusters,k")
plt.ylabel("WCSS")
plt.title("The Elbow Method")
plt.show()

# Visualize the clusters
kmeans = KMeans(n_clusters=2,max_iter=300,init="k-means++",random_state=42)
y_kmeans = kmeans.fit_predict(x)

plt.figure(figsize=(6,6))
plt.scatter(x[y_kmeans == 0]['Sector_score'],x[y_kmeans == 0]['Audit_Risk'],s=20,c='red',label='Cluster 1')
plt.scatter(x[y_kmeans == 1]['Sector_score'],x[y_kmeans == 1]['Audit_Risk'],s=20,c='blue',label='Cluster 2')
plt.scatter(kmeans.cluster_centers_[:,0],kmeans.cluster_centers_[:,1],s=50,c='yellow',label='Centroids')
plt.title('Clusters of Audit Risk')
plt.xlabel('Sector Score')
plt.ylabel('Audit Risk')
plt.legend()
plt.show()
```

The graph are below.





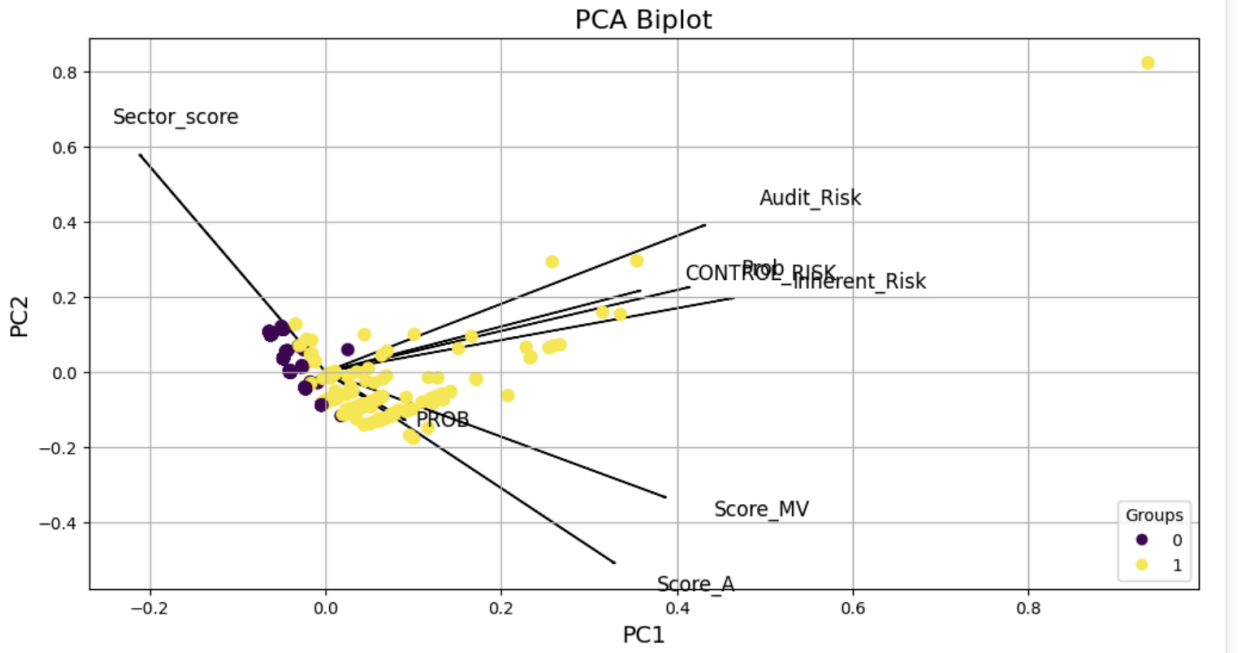
The elbow method is a technique used in clustering algorithms, such as K-means clustering, to determine the optimal number of clusters for a given dataset. It involves plotting the Within-Cluster Sum of Squares (WCSS) against the number of clusters (k) and identifying the "elbow" point where the rate of decrease in WCSS slows down significantly. This point suggests the ideal number of clusters that best represents the underlying structure of the data without overfitting or underfitting. We can see as the number of clusters increases, WCSS tends to decrease. This decrease signifies that the data points within each cluster are getting closer to their respective centroids, indicating improved clustering performance. However, as the number of clusters continues to increase, the rate of decrease in WCSS may slow down, eventually reaching a point where adding more clusters doesn't significantly reduce WCSS. This phenomenon is typically observed at the "elbow" point in the WCSS vs. number of clusters plot, indicating the optimal number of clusters for the dataset.

4.2 PCA (Principal Component Analysis)

The second unsupervised learning we use is Principal Component Analysis (PCA). PCA is a dimensionality reduction technique used to transform high-dimensional data into a lower-dimensional space while preserving most of the variance in the data. It works by finding the orthogonal axes (principal components) along which the data has the maximum variance. These principal components are ordered by the amount of variance they explain, with the first component explaining the most variance and subsequent components explaining less variance.

```
# Locate the features
x = riskAudit.loc[:,['Sector_score', 'Score_A', 'Score_MV', 'PROB', 'Prob', 'Inherent_Risk',
       'CONTROL_RISK', 'Audit_Risk']]

# Standardize the features
scaler = StandardScaler()
x_scaled = scaler.fit_transform(x_train)

# Apply PCA
pca = PCA(n_components=2)
pc_scores = pd.DataFrame(pca.fit_transform(x_scaled),columns=['PC1','PC2'])

pc1 = pca.fit_transform(x_scaled)[:,0]
pc2 = pca.fit_transform(x_scaled)[:,1]
loadings = pca.components_

scalePC1 = 1.0/(pc1.max() - pc1.min())
scalePC2 = 1.0/(pc2.max() - pc2.min())
features = x.columns

fig,ax = plt.subplots(figsize=(12,6))
for i, feature in enumerate(x.columns):
    ax.arrow(0,0,loadings[0,i],\
             loadings[1,i])
    ax.text(loadings[0,i]*1.15,\
            loadings[1,i]*1.15,\
            feature,\
            fontsize=12)

scatter = ax.scatter(pc1*scalePC1,pc2*scalePC2,c=y_train,cmap='viridis',s=50)
plt.xlabel('PC1',fontsize=14)
plt.ylabel('PC2',fontsize=14)
plt.title('PCA Biplot',fontsize=16)
plt.grid()
plt.show()
```

The results are below


PCA Biplot
Sector_score
Audit_Risk
Prob_RISK
CONTROL_Inherent_Risk
PROB
Score_MV
Score_A
Groups
0
1
PC1
PC2
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.2
0.0
0.2
0.4
0.6
0.8

5. Conclusion

In conclusion, based on the results obtained from the various supervised and unsupervised learning techniques applied to the audit dataset, it can be concluded that the objective of devising a tool to aid auditors in identifying potentially fraudulent firms and constructing a system to forecast companies involved in suspicious activities has been largely achieved. The models built, such as Logistic Regression, Random Forest Classifier, Decision Tree Classifier, Support Vector Machine (SVM) Classifier, K-Nearest Neighbor (KNN) Classifier, Neural Network Classifier, and Gaussian Naive Bayes Classifier, demonstrated high accuracy, precision, recall, and F1-scores, indicating their effectiveness in classifying instances and predicting outcomes related to audit risk assessment and fraud detection. Additionally, unsupervised learning techniques like K-Means Clustering and Principal Component Analysis (PCA) provided insights into data patterns and structures, further enhancing the understanding of risk indicators and potential fraudulent behavior. Overall, the combination of these statistical learning approaches has successfully contributed to achieving the stated objectives of the analysis.