IDS Project Report

Audit-Data

Submitted By:

Neeraj Appujani (18ucc091)

Submitted to :

Dr. Sakthi Balan, Dr. Subrat Kumar Dash and Dr. Sudheer Sharma

1.Data Set Used :

Audit-Data (click for more info)

No. of instances - 775
No. of attributes - 27
No. of independent variables - 26
No. of dependent variables - 1

2.Source :

UCI Machine Learning Repository

3.Aim :

This data set helps us to build a classification model that can predict the fraudulent firm on the basis of present and historical risk factors.

4.Data set Specifications :

Data Set Characteristics:	Multivariate	**Number of Instances:**	775	**Area:**	Industry
Attribute Characteristics:	Real	**Number of Attributes:**	27	**Date Donated**	2018-07-14
Associated Tasks:	Classification	**Missing Values?**	No		

5.Data Attributes :

- Sector_score

- LOCATION_ID
- PARA_A
- Score_A
- Risk_A
- PARA_B
- Score_B
- Risk_B
- TOTAL
- numbers
- Score_B.1
- Risc_C
- Money_Value
- Score_MV
- Risk_D
- District_Loss
- PROB
- Risk_E
- History
- Prob
- Risk_F
- Score
- Inherent_Risk
- Control_Risk
- Detection_Risk
- Audit_Risk
- Risk

Implementation :

1.Importing Libraries and Loading Dataset :

(i) Importing Libraries :

```
# importing......
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn import metrics
%matplotlib inline
```

(ii) Reading Data Set :

```
[ ] # data reading.....
    dataset = pd.read_csv("/content/audit_risk.csv")
    dataset.shape
```

```
(775, 27)
```

2. Data Visualization :

(i) Initial Rows :

There are 27 columns in this table which represents the 27 attributes. Also there is 1 class label.

```
# initial rows....
dataset.head(10)
```

	Sector_score	LOCATION_ID	PARA_A	Score_A	Risk_A	PARA_B	Score_B	Risk_B	TOTAL	numbers	Score_B.1	Risk_C	Money_Value	Score_MV	Risk_D	District_Loss	PF
0	3.89	23	4.18	0.6	2.508	2.50	0.2	0.500	6.68	5.0	0.2	1.0	3.38	0.2	0.676	2	
1	3.89	6	0.00	0.2	0.000	4.83	0.2	0.966	4.83	5.0	0.2	1.0	0.94	0.2	0.188	2	
2	3.89	6	0.51	0.2	0.102	0.23	0.2	0.046	0.74	5.0	0.2	1.0	0.00	0.2	0.000	2	
3	3.89	6	0.00	0.2	0.000	10.80	0.6	6.480	10.80	6.0	0.6	3.6	11.75	0.6	7.050	2	
4	3.89	6	0.00	0.2	0.000	0.08	0.2	0.016	0.08	5.0	0.2	1.0	0.00	0.2	0.000	2	
5	3.89	6	0.00	0.2	0.000	0.83	0.2	0.166	0.83	5.0	0.2	1.0	2.95	0.2	0.590	2	
6	3.89	7	1.10	0.4	0.440	7.41	0.4	2.964	8.51	5.0	0.2	1.0	44.95	0.6	26.970	2	
7	3.89	8	8.50	0.6	5.100	12.03	0.6	7.218	20.53	5.5	0.4	2.2	7.79	0.4	3.116	2	
8	3.89	8	8.40	0.6	5.040	11.05	0.6	6.630	19.45	5.5	0.4	2.2	7.34	0.4	2.936	2	
9	3.89	8	3.98	0.6	2.388	0.99	0.2	0.198	4.97	5.0	0.2	1.0	1.93	0.2	0.386	2	

(ii) Checking for null values :

```
[ ] dataset.isnull().sum()
    # no NULL values in dataset

    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       0
    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
```

Since there are non-null values, data cleaning is not required.

(iii) Data Types of all attributes :

The 4 attributes take integer value while the other 23 take float value .

```python
# discription of coloumns,,,
dataset.info()
#(no null values, so no need to data cleaning).......
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 775 entries, 0 to 774
Data columns (total 27 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Sector_score    775 non-null    float64
 1   LOCATION_ID     775 non-null    int64  
 2   PARA_A          775 non-null    float64
 3   Score_A         775 non-null    float64
 4   Risk_A          775 non-null    float64
 5   PARA_B          775 non-null    float64
 6   Score_B         775 non-null    float64
 7   Risk_B          775 non-null    float64
 8   TOTAL           775 non-null    float64
 9   numbers         775 non-null    float64
 10  Score_B.1       775 non-null    float64
 11  Risk_C          775 non-null    float64
 12  Money_Value     775 non-null    float64
 13  Score_MV        775 non-null    float64
 14  Risk_D          775 non-null    float64
 15  District_Loss   775 non-null    int64  
 16  PROB            775 non-null    float64
 17  RiSk_E          775 non-null    float64
 18  History         775 non-null    int64  
 19  Prob            775 non-null    float64
 20  Risk_F          775 non-null    float64
 21  Score           775 non-null    float64
 22  Inherent_Risk   775 non-null    float64
 23  CONTROL_RISK    775 non-null    float64
 24  Detection_Risk  775 non-null    float64
 25  Audit_Risk      775 non-null    float64
 26  Risk            775 non-null    int64  
dtypes: float64(23), int64(4)
memory usage: 163.6 KB
```

(iv) Unique Values in Class Label :

There are 2 different values for Class Label i.e. 0 and 1.

```python
# different class label.....
dataset['Risk'].unique()
```

```
array([1, 0])
```

(v) Description of dataset :

- Following is the statical description of the complete dataset.

- The features are not on the same scale.
  For example - Sector_score has mean 20.13 and Score B.1 is 0.223. Features should be on the same scale for an algorithm such as logistic regression to converge fast.

```
# description....
dataset.describe()
```

	Sector_score	LOCATION_ID	PARA_A	Score_A	Risk_A	PARA_B	Score_B	Risk_B	TOTAL	numbers	Score_B.1	Risk_C	Money_Value
count	775.000000	775.000000	775.000000	775.000000	775.000000	775.000000	775.000000	775.000000	775.000000	775.000000	775.000000	775.000000	775.000000
mean	20.138877	14.843871	2.453059	0.351484	1.352712	10.813924	0.313290	6.342181	13.235241	5.067742	0.223742	1.153161	14.137631
std	24.301417	9.881214	5.681977	0.174082	3.442348	50.114461	0.169865	30.091403	51.343841	0.264608	0.080399	0.537736	66.606519
min	1.850000	1.000000	0.000000	0.200000	0.000000	0.000000	0.200000	0.000000	0.000000	5.000000	0.200000	1.000000	0.000000
25%	2.370000	8.000000	0.210000	0.200000	0.042000	0.000000	0.200000	0.000000	0.540000	5.000000	0.200000	1.000000	0.000000
50%	3.890000	13.000000	0.880000	0.200000	0.176000	0.410000	0.200000	0.082000	1.370000	5.000000	0.200000	1.000000	0.090000
75%	55.570000	19.000000	2.480000	0.600000	1.488000	4.160000	0.400000	1.887000	7.725000	5.000000	0.200000	1.000000	5.595000

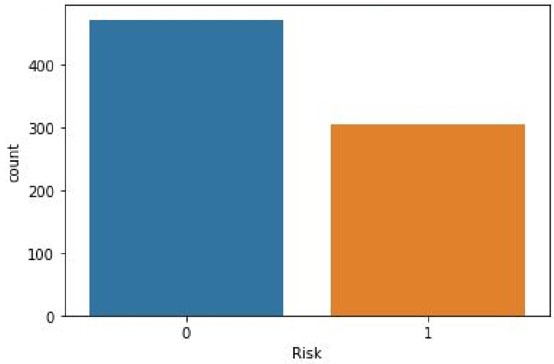
(vi) Frequency of Class labels :

- The frequency distribution of class labels is pretty balanced.

- The instances of type 1 and type 2 constitute almost equally.

```
# graph plot.....
sns.countplot(x='Risk', data=dataset)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0b741b2550>
```



We can see plots of both the values (Risk=0) & (Risk=1) are pretty much balanced .

(vii) Checking and handling outliers :

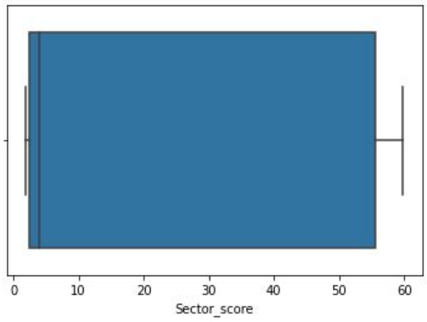
With the help of box plots, we try to visualize the outliers present in the dataset for each attribute and handle it accordingly. On X axis we have our class label **'Risk'** which is plotted against each attribute.

```
# frequency of class label.....
print(dataset.groupby('Risk').size())
```

```
Risk
0    471
1    305
dtype: int64
```

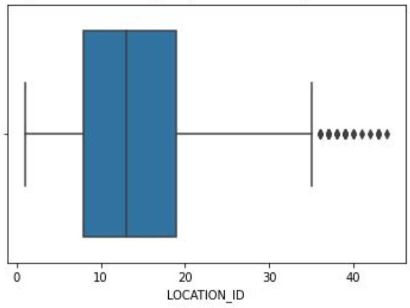
```
# box plot.....
sns.boxplot(x=dataset['Sector_score'])
# no outlier,,,
```

<matplotlib.axes._subplots.AxesSubplot at 0x7fa5c9cf5588>

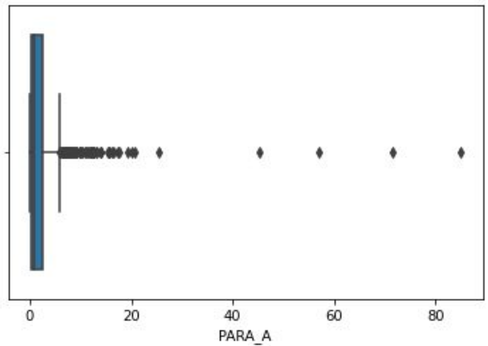



```
# box plot.....
sns.boxplot(x=dataset['LOCATION_ID'])
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f80e93b8400>

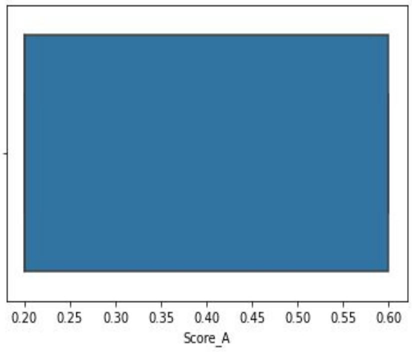



```
# box plot.....
sns.boxplot(x=dataset['PARA_A'])
# outlier handling,,,
a=dataset['PARA_A'][dataset['PARA_A']>80].index
dataset.drop(a,inplace=True)
```

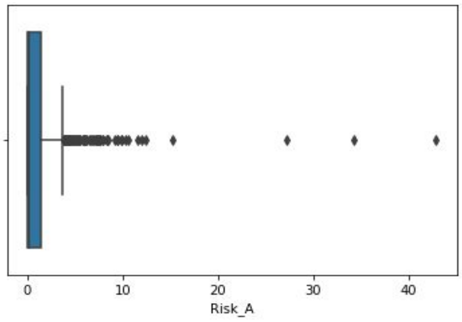


```python
# box plot.....
sns.boxplot(x=dataset['Score_A'])
```

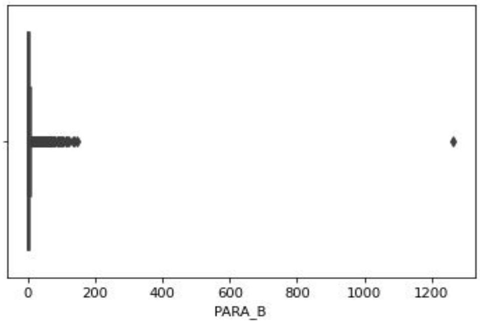
<matplotlib.axes._subplots.AxesSubplot at 0x7f80ebc5c4e0>




```python
# box plot.....
sns.boxplot(x=dataset['Risk_A'])
# outlier handling,,,
b=dataset['Risk_A'][dataset['Risk_A']>40].index
dataset.drop(b,inplace=True)
```

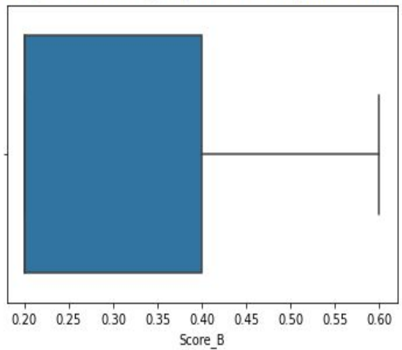



```python
# box plot.....
sns.boxplot(x=dataset['PARA_B'])
# outlier handling,,,
c=dataset['PARA_B'][dataset['PARA_B']>1200].index
dataset.drop(c,inplace=True)
```

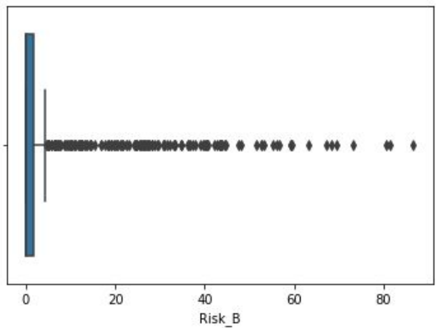



```python
# box plot.....
sns.boxplot(x=dataset['Score_B'])
```

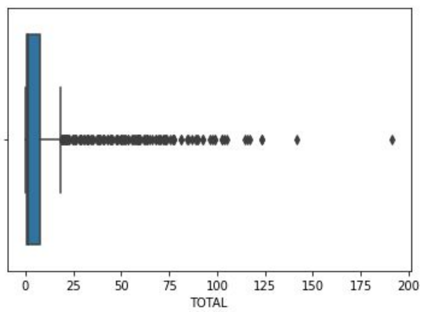
<matplotlib.axes._subplots.AxesSubplot at 0x7f80eb9052b0>



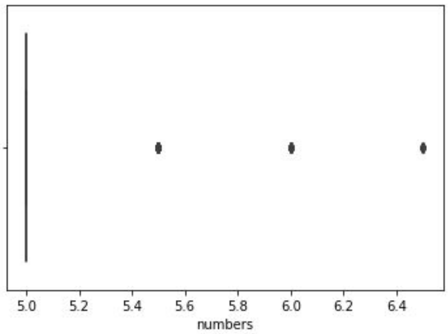

```python
# box plot.....
sns.boxplot(x=dataset['Risk_B'])
# outlier handling,,,
d=dataset['Risk_B'][dataset['Risk_B']>85].index
dataset.drop(d,inplace=True)
```



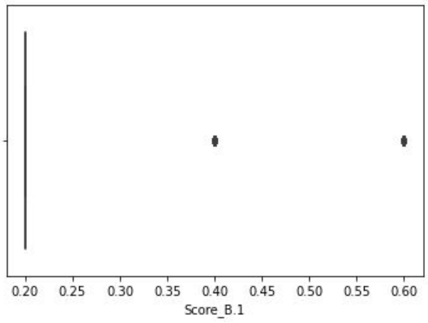

```python
# box plot.....
sns.boxplot(x=dataset['TOTAL'])
# outlier handling,,,
e=dataset['TOTAL'][dataset['TOTAL']>175].index
dataset.drop(e,inplace=True)
```



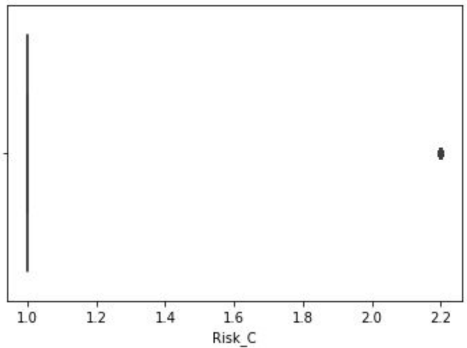
```python
# box plot.....
sns.boxplot(x=dataset['numbers'])
# outlier handling,,,
f=dataset['numbers'][dataset['numbers']>8.5].index
dataset.drop(f,inplace=True)
```



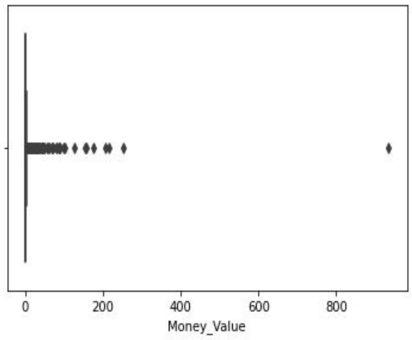

```python
# box plot.....
sns.boxplot(x=dataset['Score_B.1'])
# outlier handling,,,
g=dataset['Score_B.1'][dataset['Score_B.1']>0.55].index
dataset.drop(g,inplace=True)
```



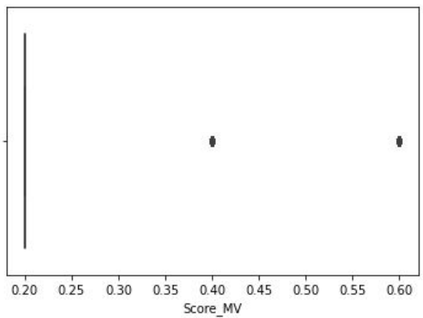

```python
# box plot.....
sns.boxplot(x=dataset['Risk_C'])
# outlier handling,,,
g1=dataset['Risk_C'][dataset['Risk_C']>2.0].index
dataset.drop(g1,inplace=True)
```




```python
# box plot.....
sns.boxplot(x=dataset['Money_Value'])
# outlier handling,,,
h=dataset['Money_Value'][dataset['Money_Value']>800].index
dataset.drop(h,inplace=True)
```

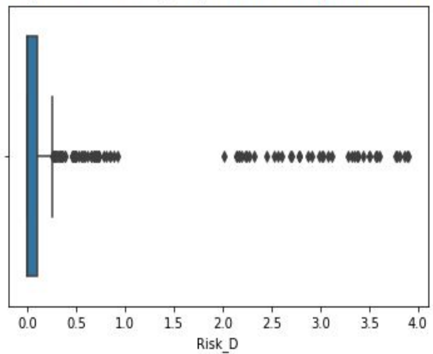



```python
# box plot.....
sns.boxplot(x=dataset['Score_MV'])
# outlier handling,,,
i=dataset['Score_MV'][dataset['Score_MV']>0.55].index
dataset.drop(i,inplace=True)
```

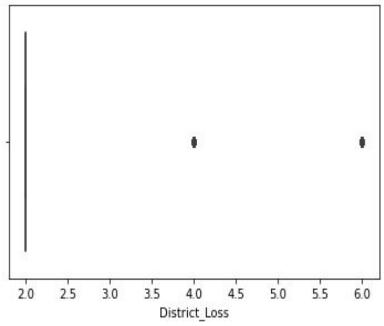



```python
# box plot.....
sns.boxplot(x=dataset['Risk_D'])
```

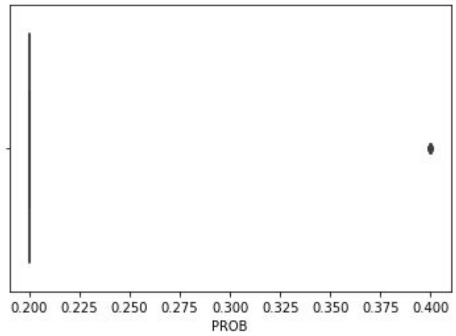
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe3617fae48>
```



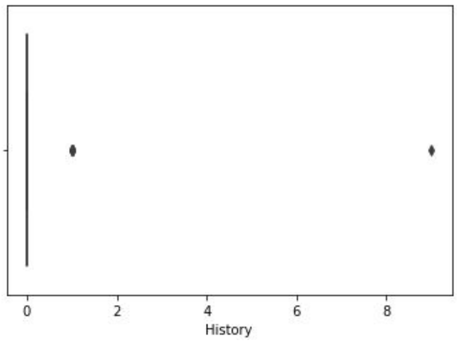
```python
# box plot.....
sns.boxplot(x=dataset['District_Loss'])
# outlier handling,,,
j=dataset['District_Loss'][dataset['District_Loss']>5.5].index
dataset.drop(j,inplace=True)
```



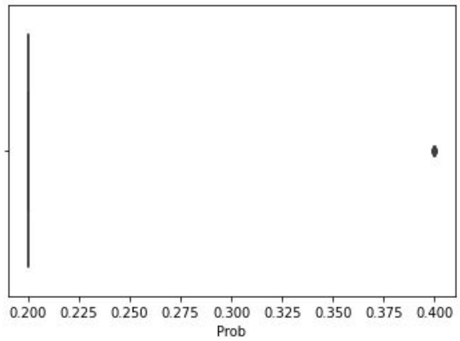

```python
# box plot.....
sns.boxplot(x=dataset['PROB'])
# outlier handling,,,
k=dataset['PROB'][dataset['PROB']>0.375].index
dataset.drop(k,inplace=True)
```




```python
# box plot.....
sns.boxplot(x=dataset['History'])
# outlier handling,,,
l=dataset['History'][dataset['History']>8].index
dataset.drop(l,inplace=True)
```

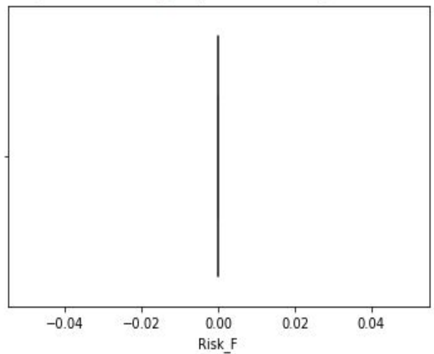



```python
# box plot.....
sns.boxplot(x=dataset['Prob'])
# outlier handling,,,
m=dataset['Prob'][dataset['Prob']>0.375].index
dataset.drop(m,inplace=True)
```

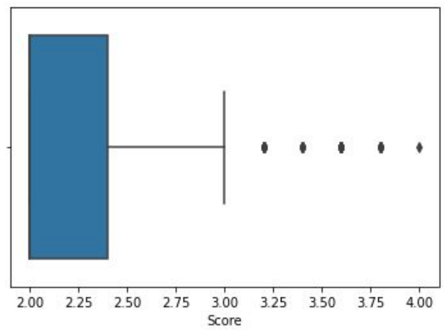



```python
# box plot.....
sns.boxplot(x=dataset['Risk_F'])
```

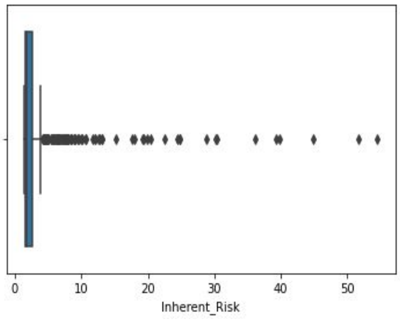
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe361764cc0>
```



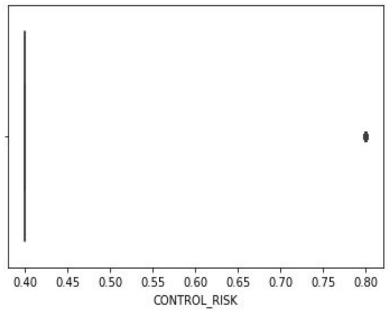

```python
# box plot.....
sns.boxplot(x=dataset['Score'])
# outlier handling,,,
o=dataset['Score'][dataset['Score']>3.85].index
dataset.drop(o,inplace=True)
```



```
# box plot.....
sns.boxplot(x=dataset['Inherent_Risk'])
# outlier handling,,,
p=dataset['Inherent_Risk'][dataset['Inherent_Risk']>52].index
dataset.drop(p,inplace=True)
```

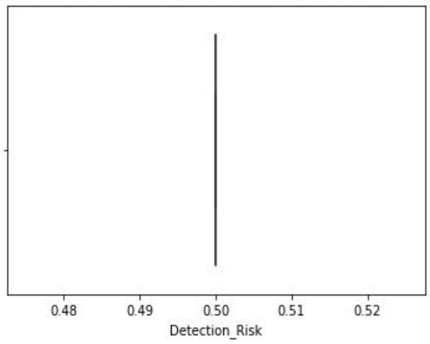



```
# box plot.....
sns.boxplot(x=dataset['CONTROL_RISK'])
# outlier handling,,,
q=dataset['CONTROL_RISK'][dataset['CONTROL_RISK']>0.75].index
dataset.drop(q,inplace=True)
```

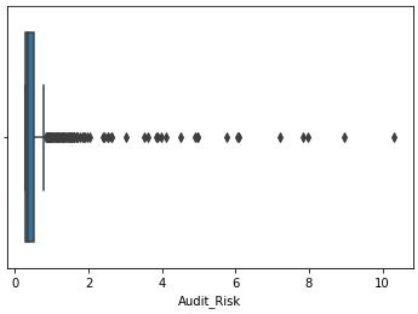



```
# box plot.....
sns.boxplot(x=dataset['Detection_Risk'])
```

<matplotlib.axes._subplots.AxesSubplot at 0x7fe361839400>




```
# box plot.....
sns.boxplot(x=dataset['Audit_Risk'])
# outlier handling,,,
r=dataset['Audit_Risk'][dataset['Audit_Risk']>10].index
dataset.drop(r,inplace=True)
```




```
[ ] dataset.shape

(506, 27)
```

NOTE:- The outliers present in each attribute have been handled successfully and the corresponding values have been dropped accordingly . Therefore we can see the number of instances have been decreased to 506.

3. Data Pre-processing :

(i) Splitting Attributes and Class label :

X will now contain Attributes and y will contain

Class label. This is helpful as we have to study the relationship between Attributes and Class label

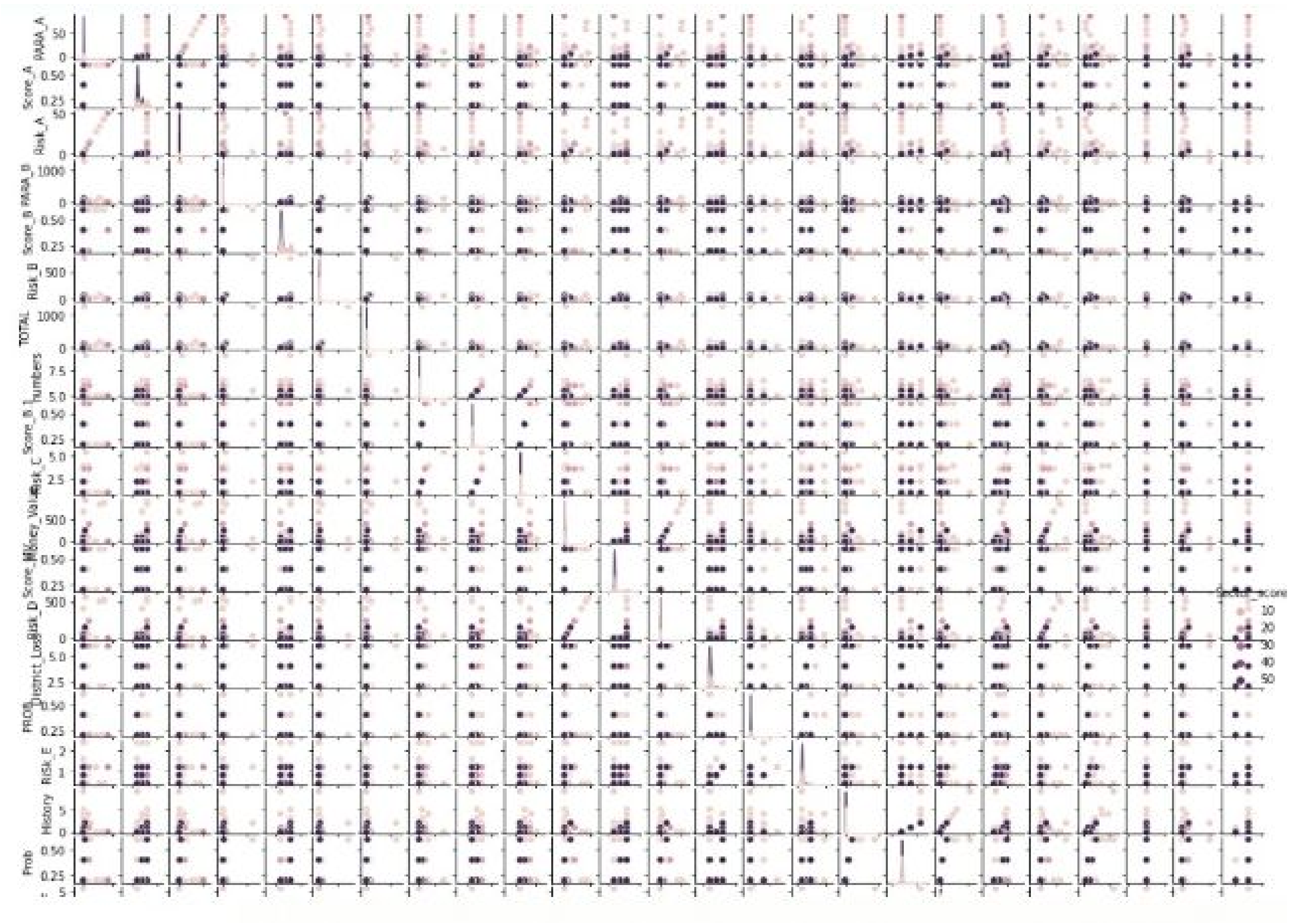
```
[ ]  X=dataset.drop(['Risk'], axis=1)
     y=dataset['Risk']
```

(ii) Analysis of Processed Data :

Pairwise Plot - It allows us to see both distribution of single variables and relationships between two variables. Pair plots are a great method to identify trends for follow-up analysis.

```
plt.figure(figsize=(30,30))
g=sns.pairplot(dataset,hue='Sector_score')
plt.show()
```

(iii) Output :



Above diagram shows that our dataset is skewed either positive side or negative side and data is not normalized

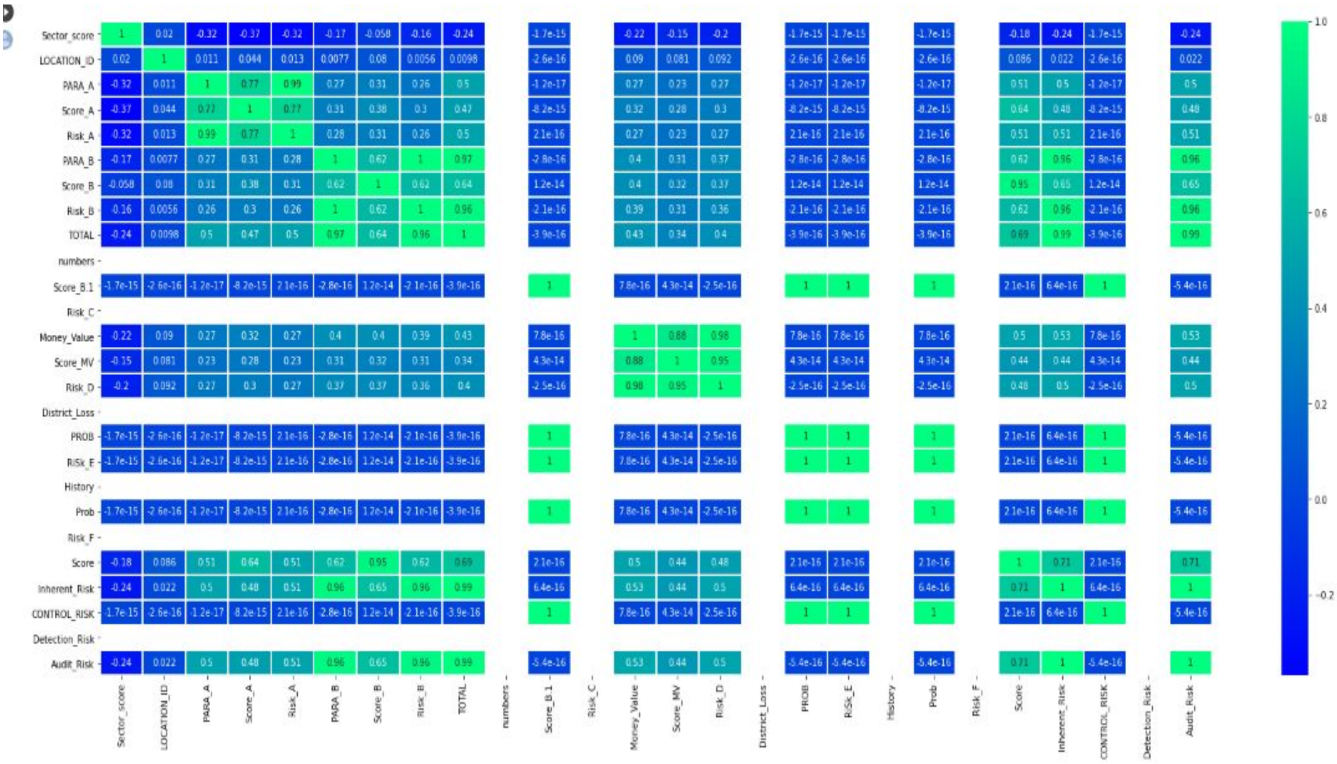
(iv) Correlation Matrix - A correlation matrix is a table showing correlation coefficients between variables. A correlation matrix is used to summarize data, as an input into a more advanced analysis.

```
plt.figure(figsize=(30,12))
sns.heatmap(X.corr(),annot=True,linecolor="white",linewidths=(1,1),cmap="winter")
plt.show()
```

These attributes are highly correlated:

Risk_A and PARA_A & TOTAL and PARA_B
Money_Value and Risk_D & Inherent_Risk and Total

Output :



(iv) Scaling the features of data :

- We scale the values of features and standardize the different features to bring them on the same scale.

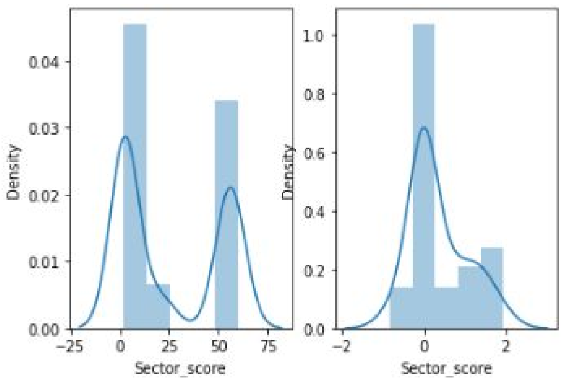
```
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
X=sc.fit_transform(X)
```

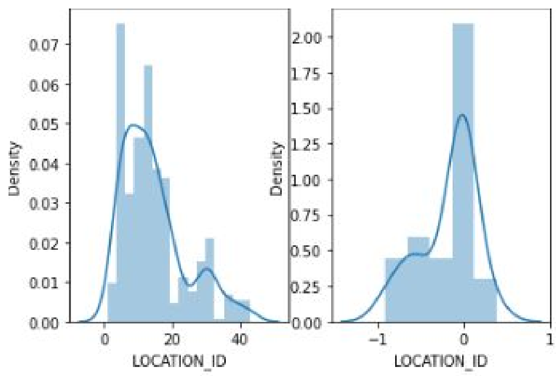

```
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
X=sc.fit_transform(X)
```

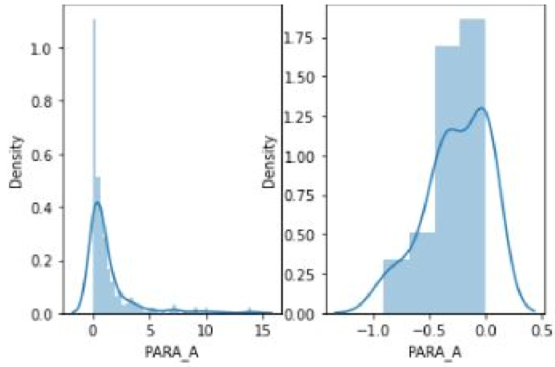
(ii) Visualizing Data after scaling using distplots :

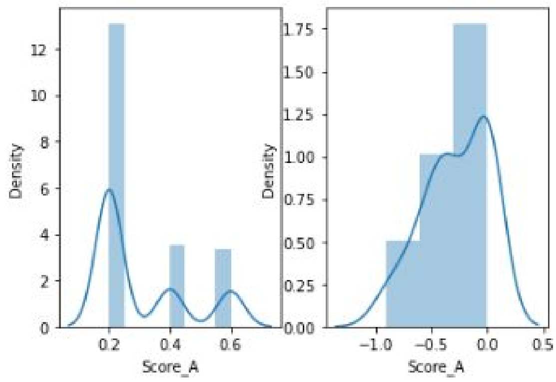
These plots show values of each attribute before and after scaling respectively.

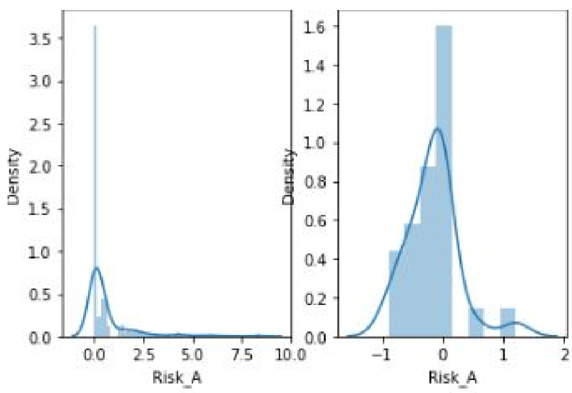
```
features=dataset.columns[:-1].tolist()
for i in range(26) :
  fig,ax=plt.subplots(1,2)
  skew=dataset[features[i]].skew()
  sns.distplot(dataset[features[i]], label='Skew= %.3f' %(skew), ax=ax[0])
  sns.distplot(X[i],ax=ax[1])
  plt.xlabel(features[i])
  plt.show()
  fig.show()
```

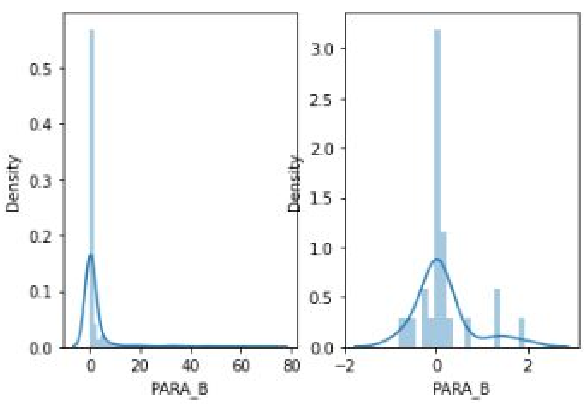


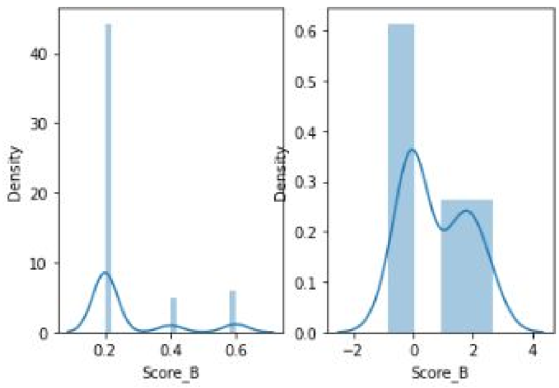


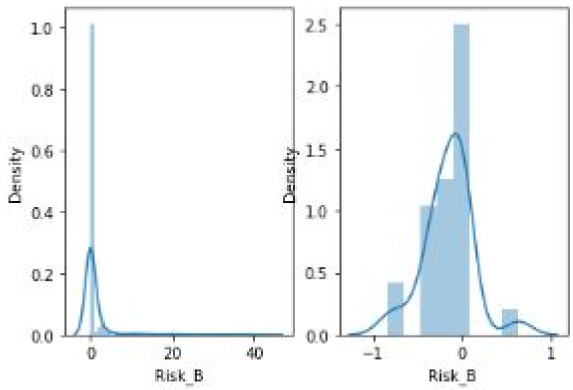

Density
PARA_A
Density
PARA_A

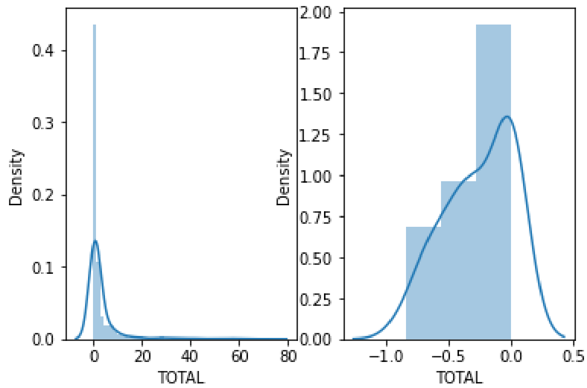

Density
Score_A
Density
Score_A

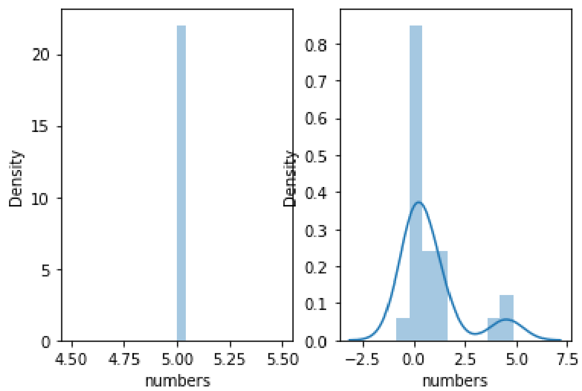

Density
Risk_A
Density
Risk_A

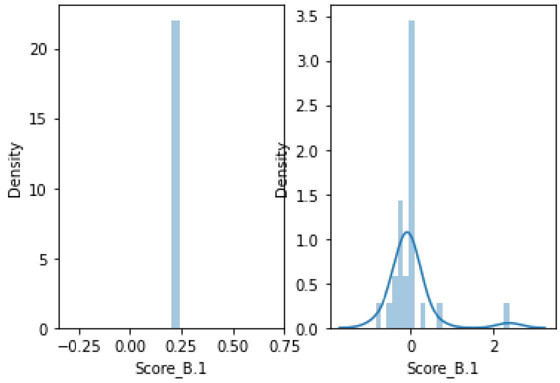

Density
PARA_B
Density
PARA_B

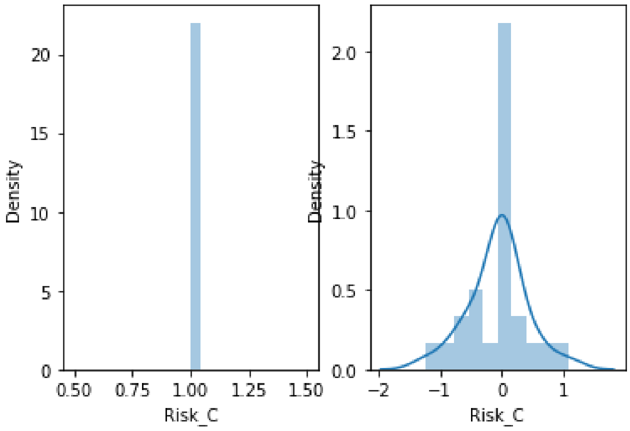

Density
Score_B
Density
Score_B

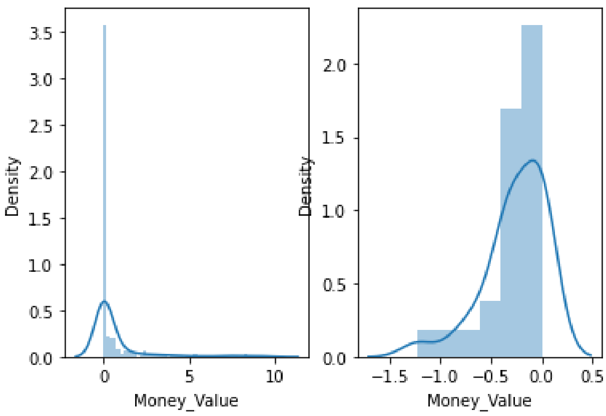

Density
Risk_B
Density
Risk_B

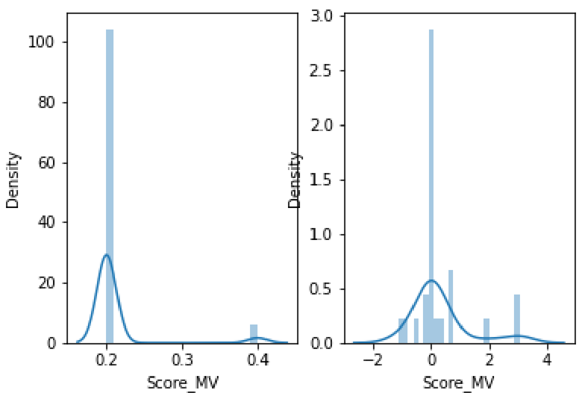

Density
TOTAL
Density
TOTAL

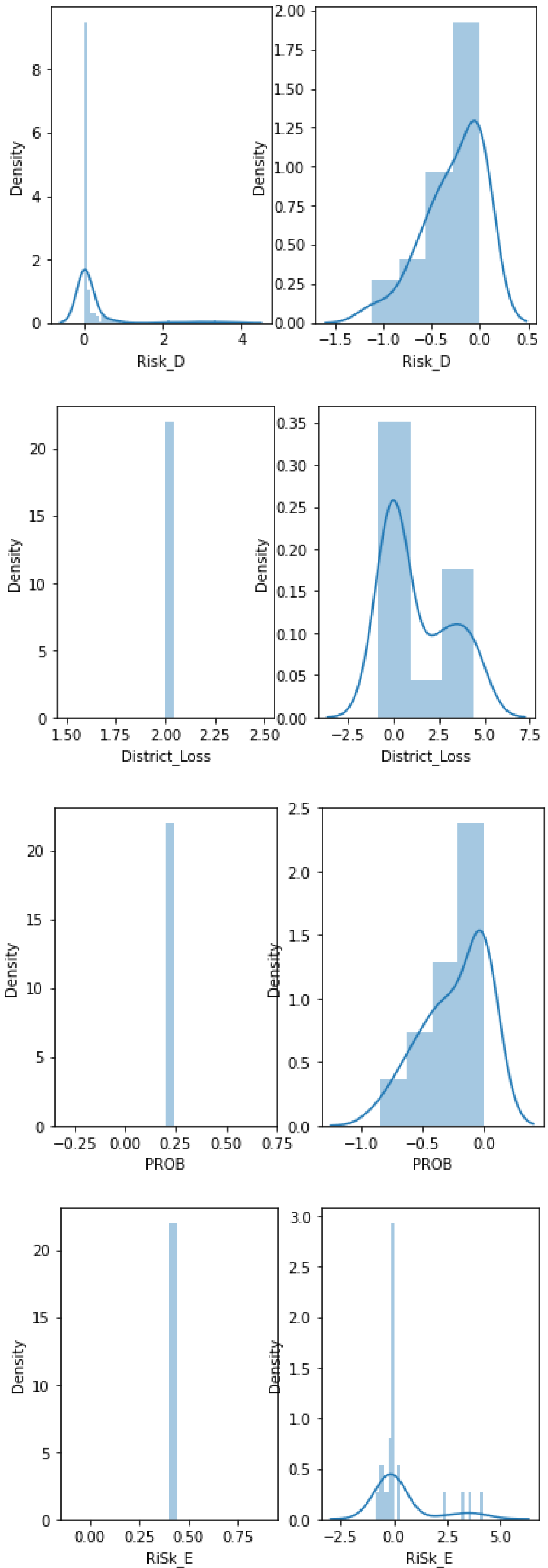

Density
numbers
Density
numbers


Density
Score_B.1
Density
Score_B.1

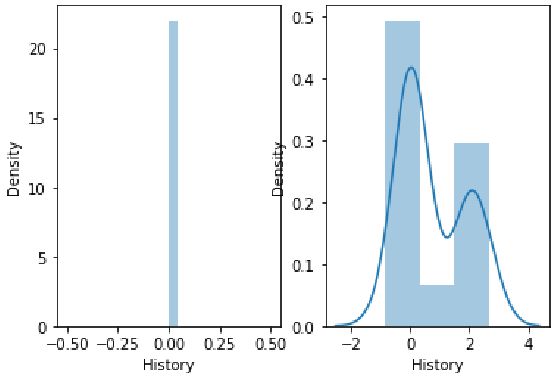

Density
Risk_C
Density
Risk_C

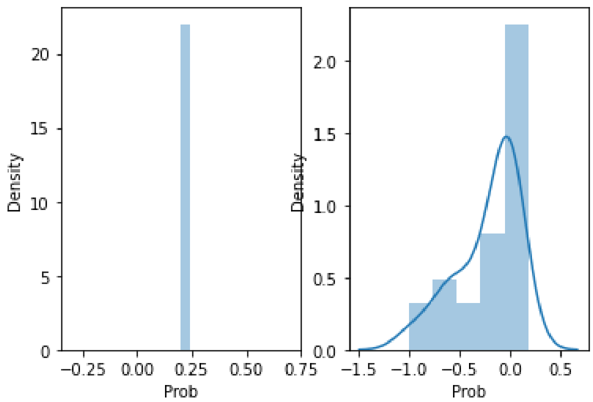

Density
Money_Value
Density
Money_Value

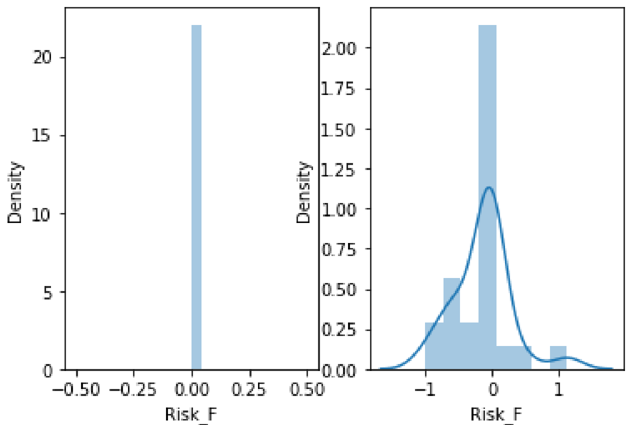

Density
Score_MV
Density
Score_MV

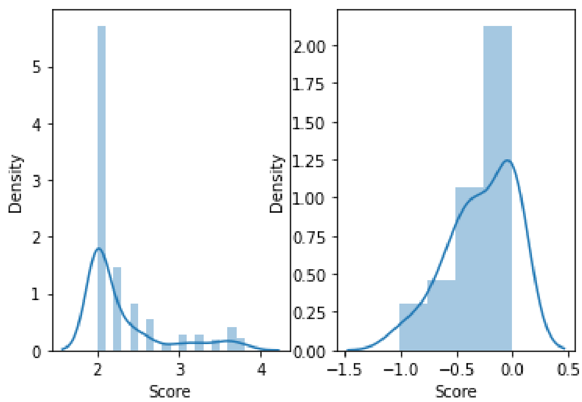


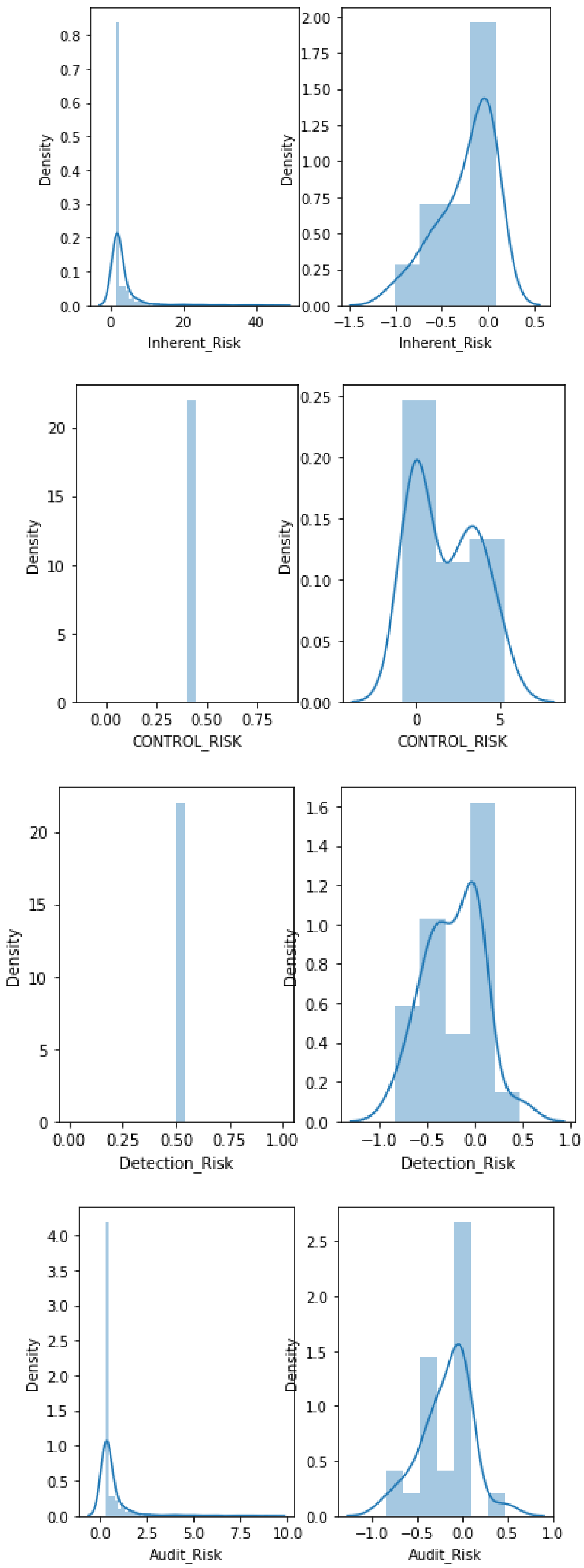
Density
Risk_D
Density
Risk_D
Density
District_Loss
Density
District_Loss
Density
PROB
Density
PROB
Density
RiSk_E
Density
RiSk_E


Density
History
Density
History


Density
Prob
Density
Prob


Density
Risk_F
Density
Risk_F


Density
Score
Density
Score


Density
Inherent_Risk
Density
Inherent_Risk
Density
CONTROL_RISK
Density
CONTROL_RISK
Density
Detection_Risk
Density
Detection_Risk
Density
Audit_Risk
Density
Audit_Risk

4.Machine Learning Models :

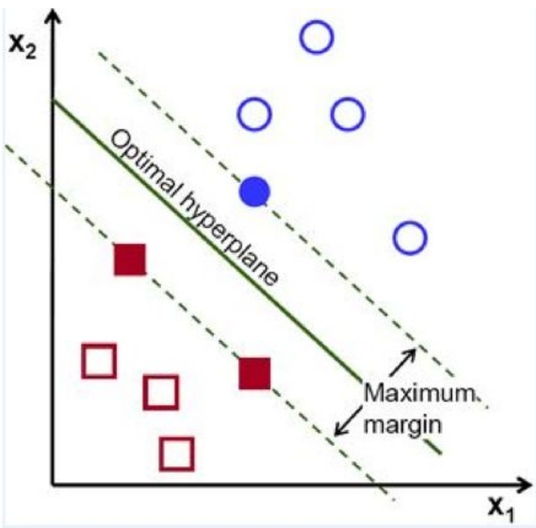
1. Splitting data into Training and Test Data :

```
X_train, X_test, y_train ,y_test = train_test_split(X,y,test_size=0.2,random_state=0,stratify=y)
y_train=y_train.values.ravel()
y_test=y_test.values.ravel()
```

2.The Support Vector Machine (SVM) Classifier :

- SVM is a supervised machine learning model that uses classification algorithms for two-group classification problems.
- After giving an SVM model sets of labeled training data for each category, we are able to categorize new text.

- The objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space(N — the number of features) that distinctly classifies the data points.



(i) Applying the classifier :

The model was defined using SVC class of sklearn.svm

```
[ ] # APPLYING SVM CLASSIFIER
    from sklearn.svm import SVC
    classifier=SVC(kernel='rbf', random_state=0)
    classifier.fit(X_train,y_train)

    SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
        decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
        max_iter=-1, probability=False, random_state=0, shrinking=True, tol=0.001,
        verbose=False)
```

```
[ ] y_pred=classifier.predict(X_test)
```

(ii) Checking its accuracy :

```
[ ] #CHECKING THE ACCURACY OF SVM
    from sklearn import metrics
    metrics.accuracy_score(y_test,y_pred)

    0.9901960784313726
```

Therefore accuracy = 99.01%

(iii) Result :

```
[ ] confusion=metrics.confusion_matrix(y_test,y_pred)
    print(confusion)
```

```
[[89  0]
 [ 1 12]]
```

```
[ ] print(metrics.classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.99      1.00      0.99        89
           1       1.00      0.92      0.96        13

    accuracy                           0.99       102
   macro avg       0.99      0.96      0.98       102
weighted avg       0.99      0.99      0.99       102
```

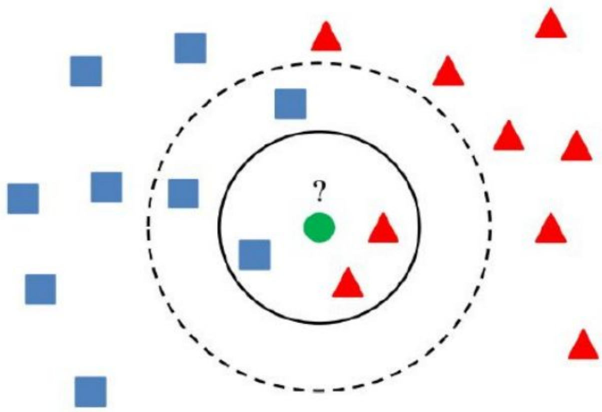
3. K-Nearest Neighbor (KNN) :

- K-Nearest Neighbors is one of the most basic yet essential classification algorithms in Machine Learning. It belongs to the supervised learning domain and finds intense application in pattern recognition, data mining and intrusion detection.

- The KNN algorithm assumes that similar things exist in close proximity. In other words, similar things are near to each other. For e.g. Birds of a feather flock together.

- This algorithm captures the idea of closeness and classify an unknown record using distance metric(Euclidean distance, in our case) and to determine the class label it takes the majority vote of k-nearest neighbors.

- Following diagram shows the example of K-nearest neighbor :



(i) Applying KNN :

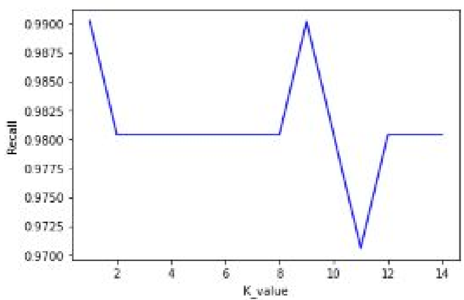
```
[ ]  #APPLYING K- NEAREST NEIGHBOUR
     from sklearn.neighbors import KNeighborsClassifier
     ranges=[]
     scores=[]
     for i in range(1,15) :
       KNN_Model=KNeighborsClassifier(n_neighbors=i,p=1)
       KNN_Model.fit(X_train,y_train)
       y_pred=KNN_Model.predict(X_test)

       cm=metrics.confusion_matrix(y_test,y_pred)
       print("The Score of "),
       print(i)
       print("Nearest Neighbors : "),
       ranges.append(i)
       scores.append(KNN_Model.score(X_test,y_test))
       print(scores[i-1])
       print("\n")

     from matplotlib import pyplot as plt
     plt.plot(ranges,scores,color='blue')
     plt.xlabel('K_value')
     plt.ylabel('Recall')
     plt.show()
```

Output :

```
The Score of
7
Nearest Neighbors :
0.9803921568627451

The Score of
8
Nearest Neighbors :
0.9803921568627451

The Score of
9
Nearest Neighbors :
0.9901960784313726

The Score of
10
Nearest Neighbors :
0.9803921568627451

The Score of
11
Nearest Neighbors :
0.9705882352941176

The Score of
12
Nearest Neighbors :
0.9803921568627451
```

```
The Score of
1
Nearest Neighbors :
0.9901960784313726

The Score of
2
Nearest Neighbors :
0.9803921568627451

The Score of
3
Nearest Neighbors :
0.9803921568627451

The Score of
4
Nearest Neighbors :
0.9803921568627451

The Score of
5
Nearest Neighbors :
0.9803921568627451

The Score of
6
Nearest Neighbors :
0.9803921568627451
```

```
The Score of
13
Nearest Neighbors :
0.9803921568627451

The Score of
14
Nearest Neighbors :
0.9803921568627451
```



5. Conclusion:

- The accuracy of K-Nearest Neighbour(K=8) : 98.03%
- The accuracy of SVM : 99.01%