Machine Learning Project 🖥️

Loan Defaulter

By ⟹ PRINCE 👑💖

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
import pandas as pd
import numpy as np
import seaborn as sns
from matplotlib import pyplot as plt
import warnings
warnings.filterwarnings('ignore')
```

```
df = pd.read_csv('/content/drive/MyDrive/Dataset/Loan Project ')
pd.set_option('display.max_columns', None)
df.head()
```

	ID	Client_Income	Car_Owned	Bike_Owned	Active_Loan	House_
0	12142509	6750	0.0	0.0	1.0	
1	12138936	20250	1.0	0.0	1.0	
2	12181264	18000	0.0	0.0	1.0	
3	12188929	15750	0.0	0.0	1.0	
4	12133385	33750	1.0	0.0	1.0	

```
df.isna().sum()
```

```
ID                              0
Client_Income                3607
Car_Owned                    3581
```

```
Bike_Owned                   3624
Active_Loan                  3635
House_Own                    3661
Child_Count                  3638
Credit_Amount                3632
Loan_Annuity                 4812
Accompany_Client             1746
Client_Income_Type           3701
Client_Education             3645
Client_Marital_Status        3473
Client_Gender                2413
Loan_Contract_Type           3651
Client_Housing_Type          3687
Population_Region_Relative   4857
Age_Days                     3600
Employed_Days                3649
Registration_Days            3614
ID_Days                      5968
Own_House_Age               80095
Mobile_Tag                      0
Homephone_Tag                   0
Workphone_Working               0
Client_Occupation           41435
Client_Family_Members        2410
Cleint_City_Rating           2409
Application_Process_Day      2428
Application_Process_Hour     3663
Client_Permanent_Match_Tag      0
Client_Contact_Work_Tag         0
Type_Organization            3609
Score_Source_1              68835
Score_Source_2               5686
Score_Source_3              26921
Social_Circle_Default       61928
Phone_Change                 3664
Credit_Bureau               18540
Default                         0
dtype: int64
```

```
df.isna().sum().sum()
```

```
395817
```

Total Null Values : 395817

```
df.dtypes
```

```
ID                           int64
Client_Income               object
Car_Owned                  float64
Bike_Owned                 float64
Active_Loan                float64
House_Own                  float64
Child_Count                float64
Credit_Amount               object
Loan_Annuity                object
Accompany_Client            object
Client_Income_Type          object
Client_Education            object
Client_Marital_Status       object
Client_Gender               object
Loan_Contract_Type          object
Client_Housing_Type         object
Population_Region_Relative  object
Age_Days                    object
Employed_Days               object
Registration_Days           object
ID_Days                     object
Own_House_Age              float64
Mobile_Tag                   int64
```

```
Homephone_Tag                  int64
Workphone_Working              int64
Client_Occupation             object
Client_Family_Members        float64
Cleint_City_Rating           float64
Application_Process_Day      float64
Application_Process_Hour     float64
Client_Permanent_Match_Tag    object
Client_Contact_Work_Tag       object
Type_Organization             object
Score_Source_1               float64
Score_Source_2               float64
Score_Source_3                object
Social_Circle_Default        float64
Phone_Change                 float64
Credit_Bureau                float64
Default                        int64
dtype: object
```

```
df.isna().apply(pd.value_counts).T
```

	False	True
ID	121856.0	NaN
Client_Income	118249.0	3607.0
Car_Owned	118275.0	3581.0
Bike_Owned	118232.0	3624.0
Active_Loan	118221.0	3635.0
House_Own	118195.0	3661.0
Child_Count	118218.0	3638.0
Credit_Amount	118224.0	3632.0
Loan_Annuity	117044.0	4812.0
Accompany_Client	120110.0	1746.0
Client_Income_Type	118155.0	3701.0
Client_Education	118211.0	3645.0
Client_Gender	119443.0	2413.0

Percentage Of Missing Data

```
# Percentage of missing

def per(dataframe):
  a = dataframe.isna().sum()
  perc = (a / (len(dataframe))) *100
  perc = pd.DataFrame(perc,columns = ["%age of missing data"]) #Making DataFrame for better experience
  return perc
per(df)
```

	%age of missing data
ID	0.000000
Client_Income	2.960051
Car_Owned	2.938715
Bike_Owned	2.974002
Active_Loan	2.983029
House_Own	3.004366
Child_Count	2.985491
Credit_Amount	2.980567
Loan_Annuity	3.948923
Accompany_Client	1.432839
Client_Income_Type	3.037191
Client_Education	2.991236
Client_Marital_Status	2.850085
Client_Gender	1.980206
Loan_Contract_Type	2.996159
Client_Housing_Type	3.025702
Population_Region_Relative	3.985852
Age_Days	2.954307
Employed_Days	2.994518
Registration_Days	2.965796

Data Preprocessing

Mobile_Tag	0.000000

```
from sklearn.preprocessing import LabelEncoder
```

```
le = LabelEncoder()
```

```
df['Client_Education']=le.fit_transform(df['Client_Education'])
df['Accompany_Client']=le.fit_transform(df['Accompany_Client'])
df['Client_Income_Type']=le.fit_transform(df['Client_Income_Type'])
df['Client_Marital_Status']=le.fit_transform(df['Client_Marital_Status'])
df['Client_Gender']=le.fit_transform(df['Client_Gender'])
df['Loan_Contract_Type']=le.fit_transform(df['Loan_Contract_Type'])
df['Client_Housing_Type']=le.fit_transform(df['Client_Housing_Type'])
df['Client_Occupation']=le.fit_transform(df['Client_Occupation'])
df['Client_Permanent_Match_Tag']=le.fit_transform(df['Client_Permanent_Match_Tag'])
df['Client_Contact_Work_Tag']=le.fit_transform(df['Client_Contact_Work_Tag'])
df['Type_Organization']=le.fit_transform(df['Type_Organization'])
```

Score Source 1	56.488806

```
df.dtypes
```

```
ID                          int64
Client_Income              object
Car_Owned                 float64
Bike_Owned                float64
Active_Loan               float64
House_Own                 float64
Child_Count               float64
Credit_Amount              object
```

```
Loan_Annuity                 object
Accompany_Client              int64
Client_Income_Type            int64
Client_Education              int64
Client_Marital_Status         int64
Client_Gender                 int64
Loan_Contract_Type            int64
Client_Housing_Type           int64
Population_Region_Relative    object
Age_Days                      object
Employed_Days                 object
Registration_Days             object
ID_Days                       object
Own_House_Age                float64
Mobile_Tag                    int64
Homephone_Tag                 int64
Workphone_Working             int64
Client_Occupation             int64
Client_Family_Members        float64
Cleint_City_Rating           float64
Application_Process_Day      float64
Application_Process_Hour     float64
Client_Permanent_Match_Tag    int64
Client_Contact_Work_Tag       int64
Type_Organization             int64
Score_Source_1               float64
Score_Source_2               float64
Score_Source_3                object
Social_Circle_Default        float64
Phone_Change                 float64
Credit_Bureau                float64
Default                       int64
dtype: object
```

```
df.head()
```

	ID	Client_Income	Car_Owned	Bike_Owned	Active_Loan	House_Own	Child_Count	Credit_Amount	Loan_An
0	12142509	6750	0.0	0.0	1.0	0.0	0.0	61190.55	34
1	12138936	20250	1.0	0.0	1.0	NaN	0.0	15282	18
2	12181264	18000	0.0	0.0	1.0	0.0	1.0	59527.35	2
3	12188929	15750	0.0	0.0	1.0	1.0	0.0	53870.4	22
4	12133385	33750	1.0	0.0	1.0	0.0	2.0	133988.4	35

```
per(df)
```

	%age of missing data
ID	0.000000
Client_Income	2.960051
Car_Owned	2.938715
Bike_Owned	2.974002
Active_Loan	2.983029
House_Own	3.004366
Child_Count	2.985491
Credit_Amount	2.980567
Loan_Annuity	3.948923
Accompany_Client	0.000000
Client_Income_Type	0.000000
Client_Education	0.000000
Client_Marital_Status	0.000000
Client_Gender	0.000000
Loan_Contract_Type	0.000000
Client_Housing_Type	0.000000
Population_Region_Relative	3.985852
Age_Days	2.954307
Employed_Days	2.994518
Registration_Days	2.965796
ID_Days	4.897584
Own_House_Age	65.729221
Mobile_Tag	0.000000
Homephone_Tag	0.000000
Workphone_Working	0.000000
Client_Occupation	0.000000
Client_Family_Members	1.977744
Cleint_City_Rating	1.976924
Application_Process_Day	1.992516
Application_Process_Hour	3.006007

Filling Null Values of Object Datatype

```
df['Client_Income'].value_counts()
```

```
13500       11908
11250       10302
15750        8719
18000        8215
9000         7577
            ...
13005.0         1
11835.0         1
25026.3         1
12285.0         1
```

```
12840.75        1
Name: Client_Income, Length: 1516, dtype: int64
```

```
df['Client_Income'].isna().sum()
```

```
3607
```

```
df['Client_Income'] = df['Client_Income'].replace(['$'],np.nan).astype('float')
df['Client_Income'] = df['Client_Income'].astype('float')
df['Client_Income'].fillna(df['Client_Income'].median(), inplace = True)
```

```
df['Credit_Amount'] = df['Credit_Amount'].replace(['$'],np.nan).astype('float')
df['Credit_Amount'] = df['Credit_Amount'].astype('float')
df['Credit_Amount'].fillna(df['Credit_Amount'].median(), inplace = True)
```

```
df['Loan_Annuity'] = df['Loan_Annuity'].replace(['$','#VALUE!'],np.nan).astype('float')
df['Loan_Annuity'] = df['Loan_Annuity'].astype('float')
df['Loan_Annuity'].fillna(df['Loan_Annuity'].median(), inplace = True)
```

```
df['Population_Region_Relative'].value_counts()
```

```
0.035792    4159
0.04622     3444
0.030755    3116
0.025164    3017
0.026392    2967
            ...
@              6
#              5
0.000938       3
100.0          1
100            1
Name: Population_Region_Relative, Length: 164, dtype: int64
```

```
df['Population_Region_Relative'] = df['Population_Region_Relative'].replace(['@','#'],np.nan).astype('float')
df['Population_Region_Relative'] = df['Population_Region_Relative'].astype('float')
df['Population_Region_Relative'].fillna(df['Population_Region_Relative'].median(), inplace = True)
```

```
df['Population_Region_Relative'].isna().sum()
```

```
0
```

```
df['Age_Days'].isna().sum()
```

```
3600
```

```
df['Age_Days'].value_counts()
```

```
13740      20
10065      20
10936      20
13231      20
16987      20
           ..
18006.0     1
20614.0     1
16296.0     1
14241.0     1
10290.0     1
Name: Age_Days, Length: 22583, dtype: int64
```

```
df['Age_Days'] = df['Age_Days'].replace(['X','x'],np.nan).astype(float)
df['Age_Days'] = df['Age_Days'].astype('float')
```

```
df['Age_Days'].fillna(df['Age_Days'].median(), inplace = True)
```

```
df['Age_Days'].isna().sum()
```

```
0
```

```
df['Employed_Days'].value_counts()
```

```
365243      19848
365243.0     1250
381            65
212            62
231            59
             ...
7607            1
10547           1
10236           1
13360           1
2863.0          1
Name: Employed_Days, Length: 13220, dtype: int64
```

```
df['Employed_Days'].isna().sum()
```

```
3649
```

```
df['Employed_Days'] = df['Employed_Days'].replace(['x'],np.nan).astype(float)
df['Employed_Days'] = df['Employed_Days'].astype('float')
df['Employed_Days'].fillna(df['Employed_Days'].median(), inplace = True)
```

```
df['Registration_Days'].value_counts()
```

```
1          44
9          37
6          36
4          36
973        33
           ..
17619       1
16057.0     1
3581.0      1
5842.0      1
895.0       1
Name: Registration_Days, Length: 19254, dtype: int64
```

```
df['Registration_Days'].isna().sum()
```

```
3614
```

```
df['Registration_Days'] = df['Registration_Days'].replace(['x'],np.nan).astype(float)
df['Registration_Days'] = df['Registration_Days'].astype('float')
df['Registration_Days'].fillna(df['Registration_Days'].median(), inplace = True)
```

```
df['ID_Days'].isna().sum()
```

```
5968
```

```
df['ID_Days'].value_counts()
```

```
4053      73
4375      71
4032      70
4312      67
4144      66
          ..
1244.0     1
```

```
1090.0     1
1563.0     1
2009.0     1
5025.0     1
Name: ID_Days, Length: 9655, dtype: int64
```

```
df['ID_Days'] = df['ID_Days'].replace(['x'],np.nan).astype(float)
df['ID_Days'] = df['ID_Days'].astype('float')
df['ID_Days'].fillna(df['ID_Days'].median(), inplace = True)
```

```
df['Score_Source_3'].isna().sum()
```

```
26921
```

```
df['Score_Source_3'].value_counts()
```

```
0.746300213    484
0.694092643    461
0.7136314      449
0.554946769    414
0.670651753    402
               ...
0.059226501      1
0.051681767      1
0.823549312      1
0.053478087      1
0.030326003      1
Name: Score_Source_3, Length: 1430, dtype: int64
```

```
df['Score_Source_3'] = df['Score_Source_3'].replace(['&'],np.nan).astype(float)
df['Score_Source_3'] = df['Score_Source_3'].astype('float')
df['Score_Source_3'].fillna(df['Score_Source_3'].median(),inplace=True)
```

Filling Null Values of Float Datatype

```
df['Own_House_Age'].isna().sum()
```

```
80095
```

```
df['Own_House_Age'].value_counts()
```

```
7.0     3015
3.0     2555
6.0     2525
2.0     2321
8.0     2302
4.0     2175
9.0     2053
1.0     2041
10.0    1945
14.0    1855
13.0    1800
12.0    1692
11.0    1644
5.0     1433
15.0    1415
16.0    1347
17.0    1193
64.0     974
18.0     919
0.0      859
19.0     740
20.0     644
21.0     591
22.0     490
```

```
24.0     457
23.0     420
65.0     392
25.0     353
26.0     225
28.0     221
27.0     182
29.0     156
30.0     129
31.0     121
32.0      87
35.0      75
33.0      68
34.0      67
36.0      57
39.0      40
38.0      37
40.0      37
37.0      32
41.0      30
42.0      13
43.0      10
44.0       8
54.0       5
63.0       2
50.0       2
57.0       2
46.0       2
45.0       1
49.0       1
69.0       1
Name: Own_House_Age, dtype: int64
```

```
df['Own_House_Age'].fillna(df['Own_House_Age'].median(), inplace = True)
```

```
df['Client_Family_Members'].isna().sum()
```

```
2410
```

```
df['Client_Family_Members'].value_counts()
```

```
2.0     61652
1.0     26213
3.0     20434
4.0      9583
5.0      1349
6.0       157
7.0        32
8.0        11
9.0         4
10.0        3
12.0        3
16.0        2
13.0        1
14.0        1
15.0        1
Name: Client_Family_Members, dtype: int64
```

```
df['Car_Owned'].fillna(df['Car_Owned'].median(), inplace = True)
df['Bike_Owned'].fillna(df['Bike_Owned'].median(), inplace = True)
df['Active_Loan'].fillna(df['Active_Loan'].median(), inplace = True)
df['House_Own'].fillna(df['House_Own'].median(), inplace = True)
df['Child_Count'].fillna(df['Child_Count'].median(), inplace = True)
```

```
df['Client_Family_Members'].fillna(df['Client_Family_Members'].median(), inplace = True)
df['Cleint_City_Rating'].fillna(df['Cleint_City_Rating'].median(), inplace = True)
df['Application_Process_Day'].fillna(df['Application_Process_Day'].median(), inplace = True)
df['Application_Process_Hour'].fillna(df['Application_Process_Hour'].median(), inplace = True)
```

```
df['Score_Source_1'].fillna(df['Score_Source_1'].median(), inplace = True)
df['Score_Source_2'].fillna(df['Score_Source_2'].median(), inplace = True)
df['Social_Circle_Default'].fillna(df['Social_Circle_Default'].median(), inplace = True)
df['Phone_Change'].fillna(df['Phone_Change'].median(), inplace = True)
df['Credit_Bureau'].fillna(df['Credit_Bureau'].median(), inplace = True)
```

```
per(df)
```

	%age of missing data
ID	0.0
Client_Income	0.0
Car_Owned	0.0
Bike_Owned	0.0
Active_Loan	0.0
House_Own	0.0
Child_Count	0.0
Credit_Amount	0.0

```
df.isna().sum()
```

```
ID                          0
Client_Income               0
Car_Owned                   0
Bike_Owned                  0
Active_Loan                 0
House_Own                   0
Child_Count                 0
Credit_Amount               0
Loan_Annuity                0
Accompany_Client            0
Client_Income_Type          0
Client_Education            0
Client_Marital_Status       0
Client_Gender               0
Loan_Contract_Type          0
Client_Housing_Type         0
Population_Region_Relative  0
Age_Days                    0
Employed_Days               0
Registration_Days           0
ID_Days                     0
Own_House_Age               0
Mobile_Tag                  0
Homephone_Tag               0
Workphone_Working           0
Client_Occupation           0
Client_Family_Members       0
Cleint_City_Rating          0
Application_Process_Day     0
Application_Process_Hour    0
Client_Permanent_Match_Tag  0
Client_Contact_Work_Tag     0
Type_Organization           0
Score_Source_1              0
Score_Source_2              0
Score_Source_3              0
Social_Circle_Default       0
Phone_Change                0
Credit_Bureau               0
Default                     0
dtype: int64
```

```
df.isnull().sum().sum()
```

```
0
```

Score_Source_3 0.0

Now, Dataset doesn't contain any null values. So we can apply Machine Learning Models.

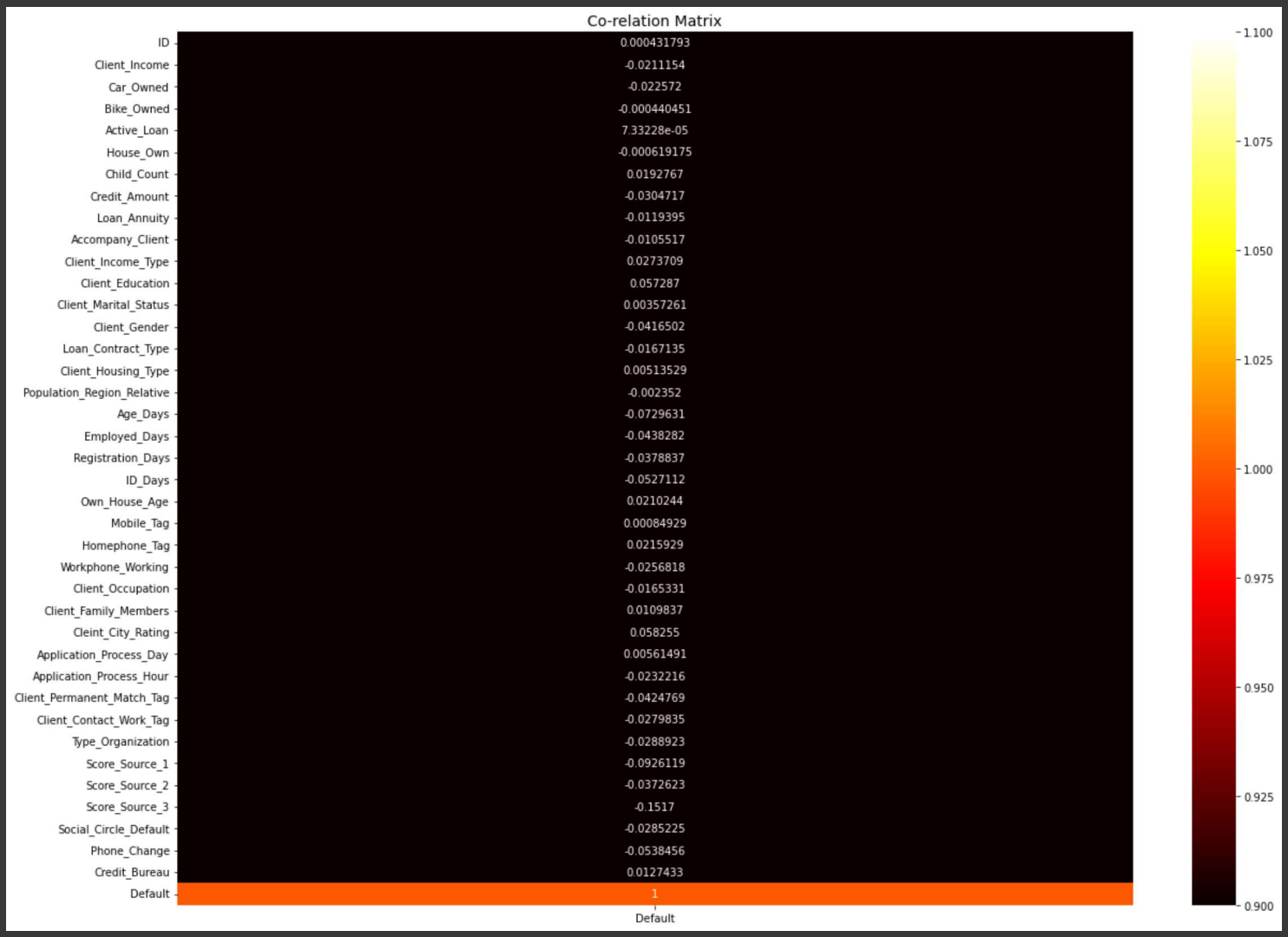
Phone_Change 0.0

Co-relation Matrix

Default 0.0

```
corr = df.corr()

plt.figure(figsize=(20,15))
plt.title('Co-relation Matrix', fontsize = 14)
sns.heatmap(corr[['Default']], vmax= 1.0, vmin = 1.0, fmt = 'g', annot = True, cmap ='hot')
plt.show()
```



Applying Machine Learning Models

1. Logistic Regression

```
# Defining Independent Variable
x = df.drop('Default', axis = 1)
# Defining Dependent Variable
y = df[['Default']]
```

Train Test Split

```
# Train Test Split
from sklearn.model_selection import train_test_split
xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size = 0.3, random_state = 1)
```

```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()

# Training the Data
lr.fit(xtrain, ytrain)
# Testing or Predicting
ypred = lr.predict(xtest)
```

```
print(ypred)
```

```
[0 0 0 ... 0 0 0]
```

```
# Training Accuracy
lr.score(xtrain, ytrain) * 100
```

```
91.96825285173331
```

Accuracy Score

```
# Accuracy Score
lr.score(xtest, ytest) * 100
```

```
91.81005005881227
```

```
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

```
accuracy_score(ytest, ypred) * 100
```

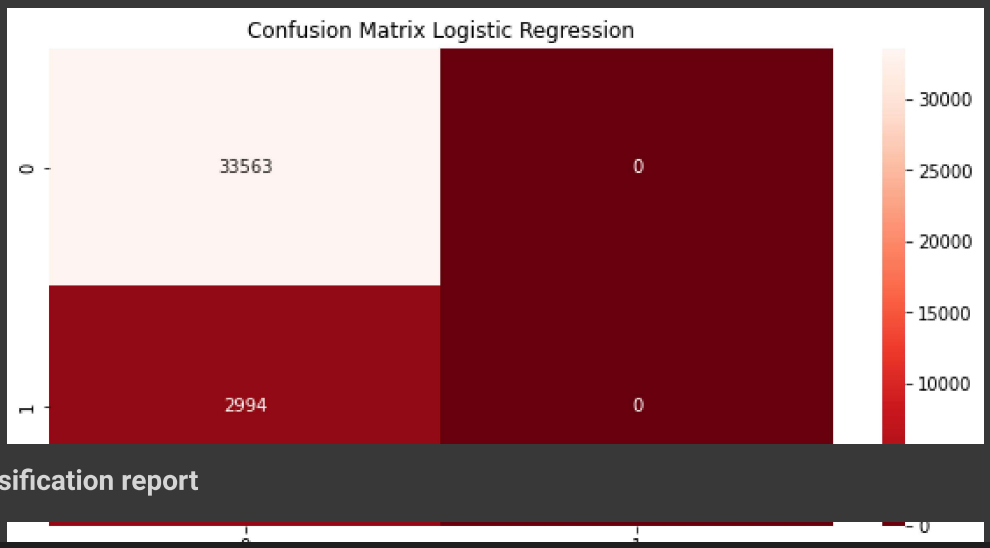
```
91.81005005881227
```

Confusion Matrix

```
# Confusion Matrix
cf = confusion_matrix(ytest, ypred)
print(cf)
```

```
[[33563     0]
 [ 2994     0]]
```

```
plt.figure(figsize=(10,5))
plt.title('Confusion Matrix Logistic Regression')
sns.heatmap(cf, annot = True, fmt = 'g', cmap = 'Reds_r')
plt.show()
```



Classification report

```
print(classification_report(ytest, ypred))
```

```
                  precision    recall  f1-score   support

               0       0.92      1.00      0.96     33563
               1       0.00      0.00      0.00      2994

        accuracy                           0.92     36557
       macro avg       0.46      0.50      0.48     36557
    weighted avg       0.84      0.92      0.88     36557
```

2. KNN : K-Nearest Neighbors

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier()
```

```
knn.fit(xtrain, ytrain)
```

```
KNeighborsClassifier()
```

```
knn_pred = knn.predict(xtest)
knn_pred
```

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

Accuracy Score using KNN Model

```
print(accuracy_score(ytest, knn_pred)*100)
```

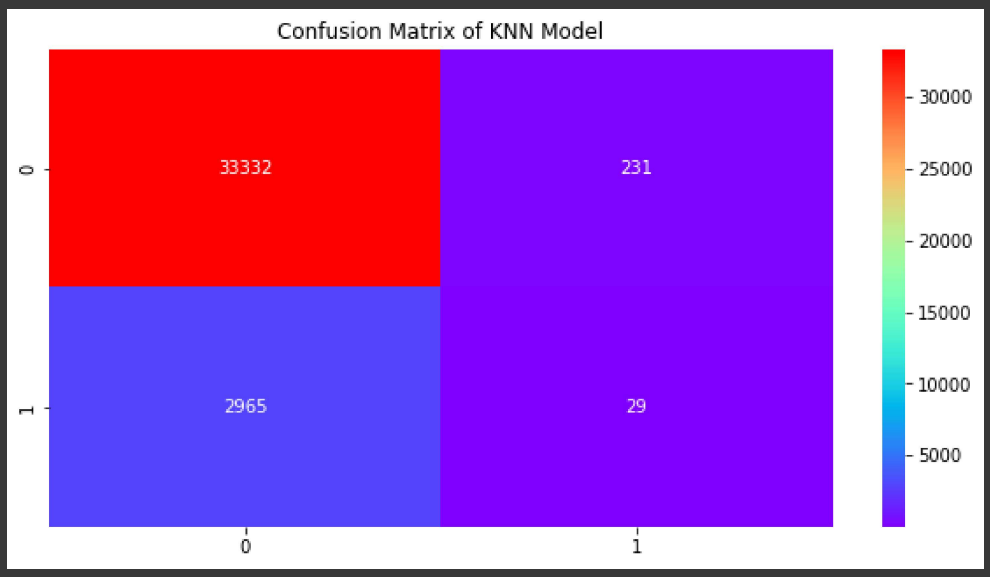
```
91.2574883059332
```

Confusion Matrix

```
cf_knn = confusion_matrix(ytest, knn_pred)
print(cf_knn)
```

```
[[33332   231]
 [ 2965    29]]
```

```
plt.figure(figsize=(10,5))
plt.title("Confusion Matrix of KNN Model")
sns.heatmap(cf_knn, annot = True, fmt = 'g', cmap = 'rainbow')
plt.show()
```



Classification Report

```
print(classification_report(ytest, knn_pred))
```

```
                  precision    recall  f1-score   support

               0       0.92      0.99      0.95     33563
               1       0.11      0.01      0.02      2994

        accuracy                           0.91     36557
       macro avg       0.51      0.50      0.49     36557
    weighted avg       0.85      0.91      0.88     36557
```

3. Naive Bayes

```
# Naive Bayes
from sklearn.naive_bayes import GaussianNB
nb = GaussianNB()
```

```
nb.fit(xtrain, ytrain)
```

```
GaussianNB()
```

```
nb_pred = nb.predict(xtest)
print(nb_pred)
```

```
[0 0 0 ... 0 0 0]
```

Accuracy Score of Naive Bayes Model

```
# Accuracy Score or Test Score
accuracy_score(ytest, nb_pred)*100
```

```
91.10703832371365
```

```
# Training Score
nb.score(xtrain, ytrain)*100
```

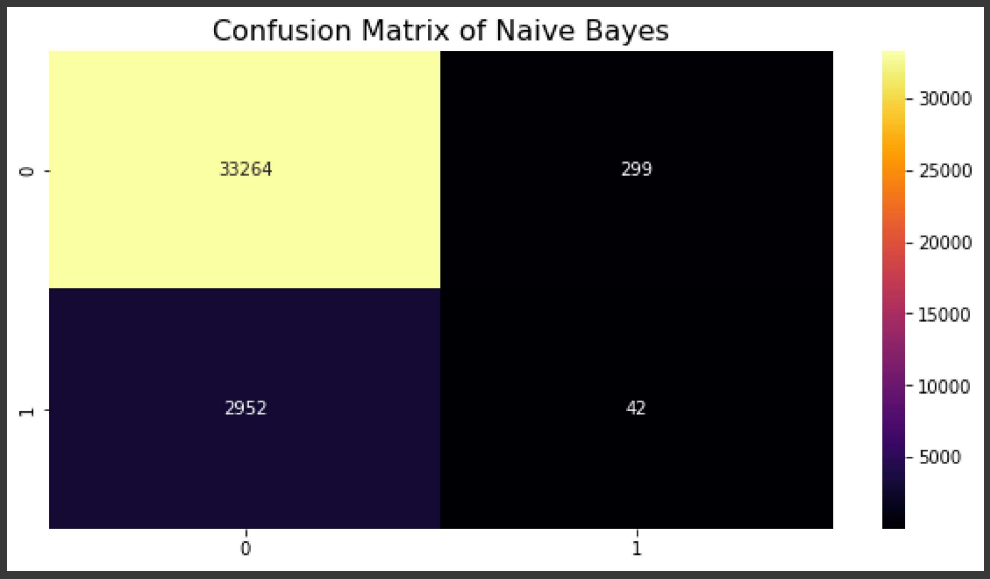
```
91.20622750559797
```

Confusion Matrix

```
cf_nb = confusion_matrix(ytest, nb_pred)
print(cf_nb)
```

```
[[33264   299]
 [ 2952    42]]
```

```
plt.figure(figsize=(10,5))
plt.title('Confusion Matrix of Naive Bayes', fontsize=16)
sns.heatmap(cf_nb, annot = True, fmt ='g', cmap = 'inferno')
plt.show()
```



Classification Report

```
print(classification_report(ytest, nb_pred))
```

```
                precision    recall  f1-score   support

             0       0.92      0.99      0.95     33563
             1       0.12      0.01      0.03      2994

      accuracy                           0.91     36557
     macro avg       0.52      0.50      0.49     36557
  weighted avg       0.85      0.91      0.88     36557
```

4. SVM : Support Vector Machine

```
# SVM : Support Vector Machine
from sklearn.svm import SVC
svm = SVC()
```

```
svm.fit(xtrain, ytrain)
```

```
SVC()
```

```
svm_pred = svm.predict(xtest)
svm_pred
```

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

Accuracy Score of SVM Model

```
# Accuracy Score or Test Accuracy
accuracy_score(ytest, svm_pred)*100
```

```
91.81005005881227
```

```
# Training Accuracy
svm.score(xtrain, ytrain)*100
```

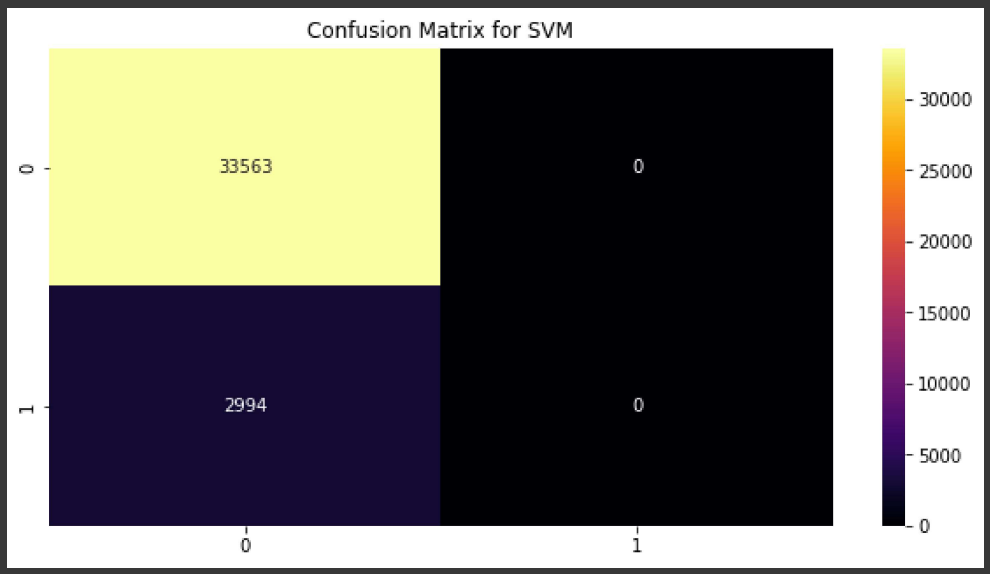
```
91.96825285173331
```

Confusion Matrix

```
cf_svm = confusion_matrix(ytest, svm_pred)
print(cf_svm)
```

```
[[33563     0]
 [ 2994     0]]
```

```
plt.figure(figsize=(10,5))
plt.title('Confusion Matrix for SVM')
sns.heatmap(cf_svm, fmt = 'g', annot = True, cmap = 'inferno')
plt.show()
```



Classification Report

```
print(classification_report(ytest, svm_pred))
```

```
              precision    recall  f1-score   support

           0       0.92      1.00      0.96     33563
           1       0.00      0.00      0.00      2994

    accuracy                           0.92     36557
   macro avg       0.46      0.50      0.48     36557
weighted avg       0.84      0.92      0.88     36557
```

CART : Classification And Regression Tree

5. Decision Tree Classifier

```
from sklearn.tree import DecisionTreeClassifier
dtc = DecisionTreeClassifier()
```

```
dtc.fit(xtrain, ytrain)
```

```
DecisionTreeClassifier()
```

```
dtc_pred = dtc.predict(xtest)
print(dtc_pred)
```

```
[0 0 0 ... 0 0 0]
```

Accuracy Score for Decision Tree Algorithm

```
# Accuracy Score or Test Accuracy
accuracy_score(ytest, dtc_pred)*100
```

```
86.65645430423722
```

```
# Training Accuracy
dtc.score(xtrain, ytrain)*100
```

```
100.0
```

Training Accuracy >> Test Accuracy

Training Accuracy is 13.5% higher than test accuracy, It means model is overfitted.

Pruning the model using parameter 'criterion'.

```
# Pruning using parameter 'criterion'
dtc1 = DecisionTreeClassifier(criterion = 'entropy', max_depth = 8)
dtc1.fit(xtrain, ytrain)
```

```
DecisionTreeClassifier(criterion='entropy', max_depth=8)
```

```
dtc1_pred = dtc1.predict(xtest)
print(dtc1_pred)
```

```
[0 0 0 ... 0 0 0]
```

Accuracy Score for Decision Tree

```
# Accuracy Score (Test Accuracy)
accuracy_score(ytest, dtc1_pred)*100
```

```
91.73892824903575
```

```
# Training Accuracy
dtc1.score(xtrain, ytrain)*100
```

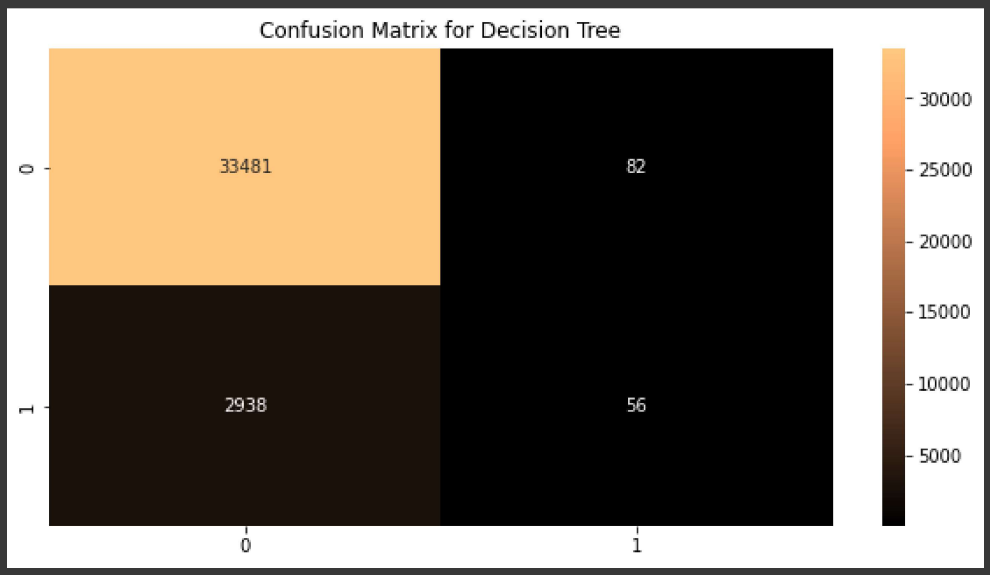
```
92.20858392243755
```

Confusion Matrix for Decision Tree

```
cf_dtc1 = confusion_matrix(ytest, dtc1_pred)
print(cf_dtc1)
```

```
[[33481    82]
 [ 2938    56]]
```

```
plt.figure(figsize=(10,5))
plt.title('Confusion Matrix for Decision Tree')
sns.heatmap(cf_dtc1, annot = True, fmt ='g', cmap = 'copper')
plt.show()
```



Classification Report for Decision Tree

```
print(classification_report(ytest, dtc1_pred))
```

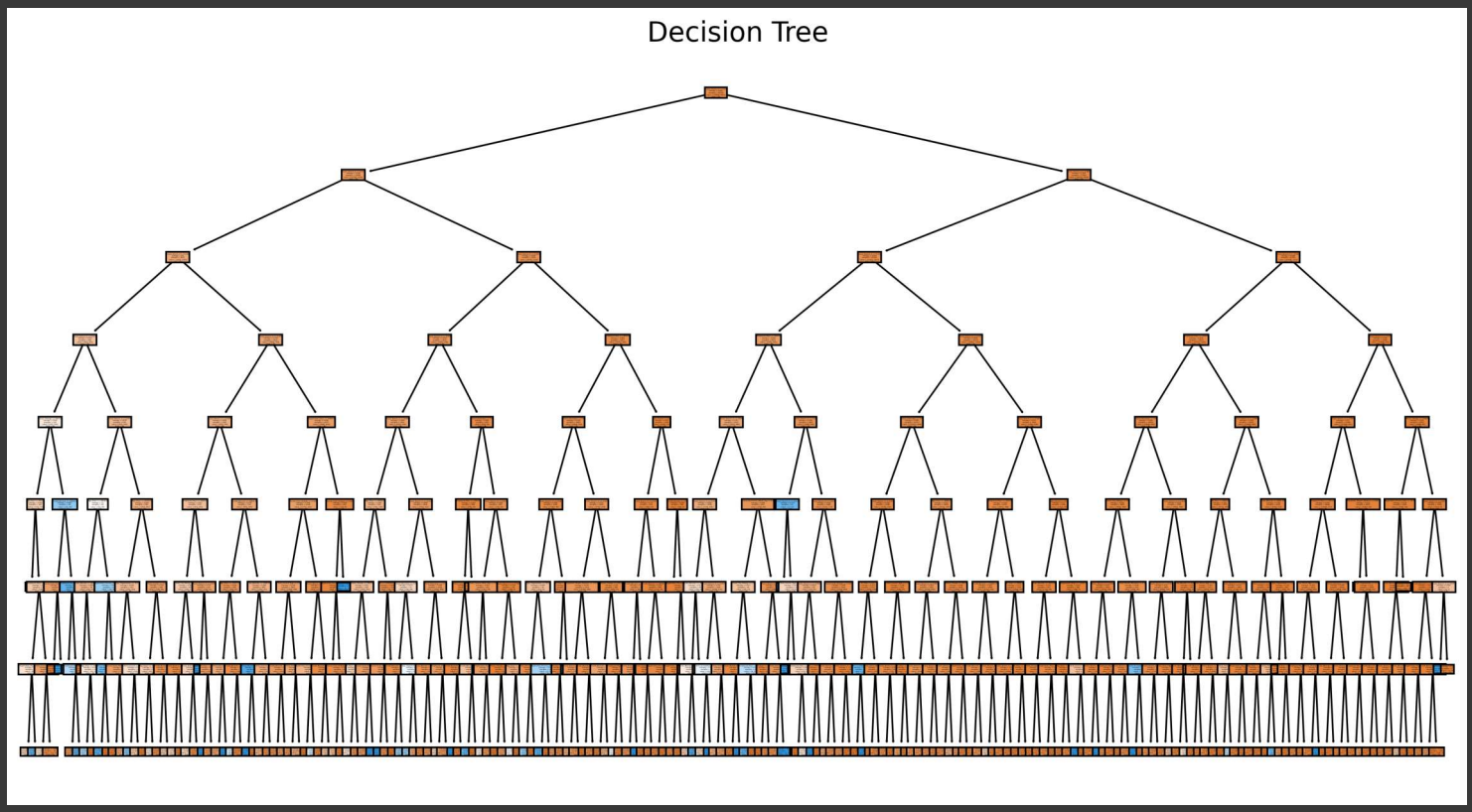
```
                  precision    recall  f1-score   support

               0       0.92      1.00      0.96     33563
               1       0.41      0.02      0.04      2994

        accuracy                           0.92     36557
       macro avg       0.66      0.51      0.50     36557
    weighted avg       0.88      0.92      0.88     36557
```

Plotting Decision Tree

```
from sklearn.tree import plot_tree
features = list(xtrain)
classes = ['No', 'Yes']
fig, axes = plt.subplots(nrows = 1, ncols = 1, figsize = (15,8), dpi = 300)
plot_tree(dtc1, feature_names = features, class_names = classes, filled = True)
plt.title('Decision Tree', fontsize = 16)
plt.show()
```



Ensemble Techniques

6. Random Forest Classifier

```
from sklearn.ensemble import RandomForestClassifier
# n_estimators = 500
rf = RandomForestClassifier(n_estimators = 500, criterion = 'entropy', max_depth = 8, random_state = 1)
```

```
rf.fit(xtrain, ytrain)
```

```
RandomForestClassifier(criterion='entropy', max_depth=8, n_estimators=500,
                       random_state=1)
```

```
rf_pred = rf.predict(xtest)
print(rf_pred)
```

```
[0 0 0 ... 0 0 0]
```

Accuracy Score for Random Forest

```
# Accuracy Score or Test Accuracy
accuracy_score(ytest, rf_pred)*100
```

```
91.81005005881227
```

```
# Training Accuracy
rf.score(xtrain, ytrain)*100
```

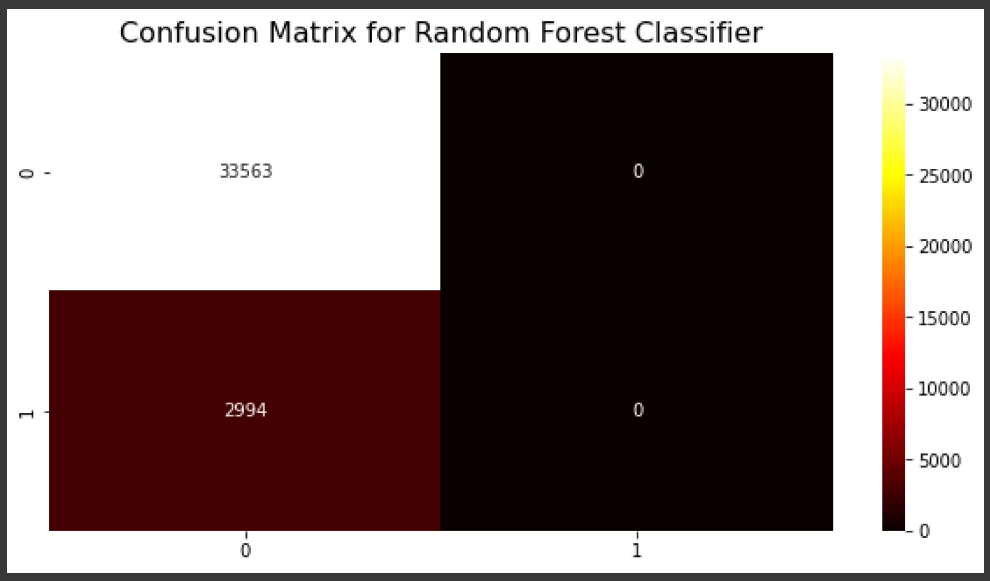
```
91.96825285173331
```

Confusion Matrix for Random Forest

```
cf_rf = confusion_matrix(ytest, rf_pred)
print(cf_rf)
```

```
[[33563     0]
 [ 2994     0]]
```

```
plt.figure(figsize=(10,5))
plt.title('Confusion Matrix for Random Forest Classifier', fontsize = 16)
sns.heatmap(cf_rf, annot = True, fmt = 'g', cmap = 'hot')
plt.show()
```



Classifiaction Report for Random Forest Classifier

```
print(classification_report(ytest, rf_pred))
```

```
                precision    recall  f1-score   support

             0       0.92      1.00      0.96     33563
             1       0.00      0.00      0.00      2994

      accuracy                           0.92     36557
     macro avg       0.46      0.50      0.48     36557
  weighted avg       0.84      0.92      0.88     36557
```

Ensemble Techniques

7. Bagging

```
from sklearn.ensemble import BaggingClassifier
bg = BaggingClassifier(base_estimator = dtc1, n_estimators = 100, random_state = 1)
```

```
bg.fit(xtrain, ytrain)
```

```
BaggingClassifier(base_estimator=DecisionTreeClassifier(criterion='entropy',
                                                        max_depth=8),
```

```
                  n_estimators=100, random_state=1)
```

```
bg_pred = bg.predict(xtest)
print(bg_pred)
```

```
[0 0 0 ... 0 0 0]
```

Accuracy Score for Bagging Classifier

```
accuracy_score(ytest, bg_pred)*100
```

```
91.82099187570097
```

```
# Training Accuracy
bg.score(xtrain, ytrain)*100
```

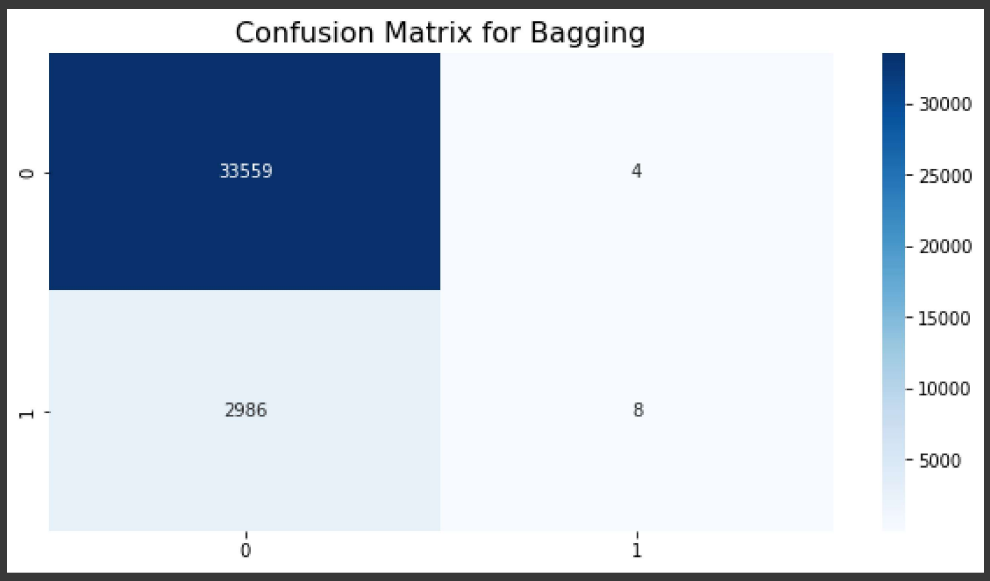
```
92.07845344025135
```

Confusion Matrix for Bagging

```
cf_bg = confusion_matrix(ytest, bg_pred)
print(cf_bg)
```

```
[[33559     4]
 [ 2986     8]]
```

```
plt.figure(figsize=(10,5))
plt.title('Confusion Matrix for Bagging', fontsize = 16)
sns.heatmap(cf_bg, annot = True, fmt = 'g', cmap = 'Blues')
plt.show()
```



Classification report for Bagging

```
print(classification_report(ytest, bg_pred))
```

```
              precision    recall  f1-score   support

           0       0.92      1.00      0.96     33563
           1       0.67      0.00      0.01      2994

    accuracy                           0.92     36557
   macro avg       0.79      0.50      0.48     36557
weighted avg       0.90      0.92      0.88     36557
```

Ensemble Techniques

8. Ada Boost

```
from sklearn.ensemble import AdaBoostClassifier
ada = AdaBoostClassifier()
```

```
ada.fit(xtrain, ytrain)
```

```
AdaBoostClassifier()
```

```
ada_pred = ada.predict(xtest)
print(ada_pred)
```

```
[0 0 0 ... 0 0 0]
```

Accuracy Score for AdaBoost Classifier

```
# Test Acuuracy or Accuracy Score
accuracy_score(ytest, ada_pred)*100
```

```
91.79910824192356
```

```
# Training Accuracy
ada.score(xtrain, ytrain)*100
```

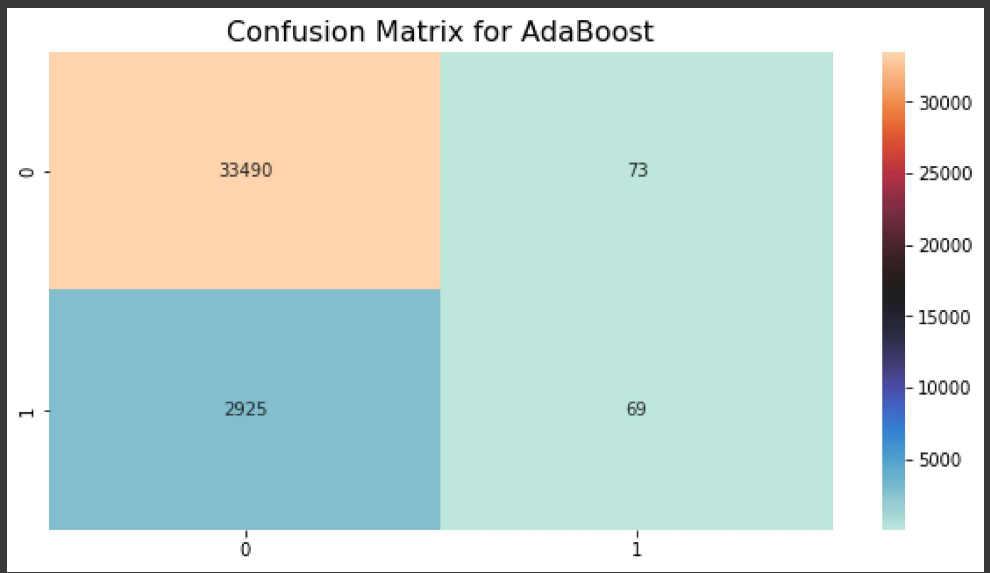
```
91.95887407824242
```

Confusion Matrix for AdaBoost

```
cf_ada = confusion_matrix(ytest, ada_pred)
print(cf_ada)
```

```
[[33490    73]
 [ 2925    69]]
```

```
plt.figure(figsize=(10,5))
plt.title('Confusion Matrix for AdaBoost', fontsize = 16)
sns.heatmap(cf_ada, annot = True, fmt = 'g', cmap = 'icefire')
plt.show()
```



Classification Report for AdaBoost

```
print(classification_report(ytest, ada_pred))
```

```
                  precision    recall  f1-score   support

               0       0.92      1.00      0.96     33563
               1       0.49      0.02      0.04      2994

        accuracy                           0.92     36557
       macro avg       0.70      0.51      0.50     36557
    weighted avg       0.88      0.92      0.88     36557
```

Ensemble Techniques

9. Gradient Boosting Classifier

```
from sklearn.ensemble import GradientBoostingClassifier
gdb = GradientBoostingClassifier()
```

```
gdb.fit(xtrain, ytrain)
```

```
GradientBoostingClassifier()
```

```
gdb_pred = gdb.predict(xtest)
print(gdb_pred)
```

```
[0 0 0 ... 0 0 0]
```

Accuracy Score for Gradient Boosting Classifier

```
# Accuracy Score or Test Accuracy
accuracy_score(ytest, gdb_pred)*100
```

```
91.79363733347923
```

```
# Training Accuracy
gdb.score(xtrain, ytrain)*100
```

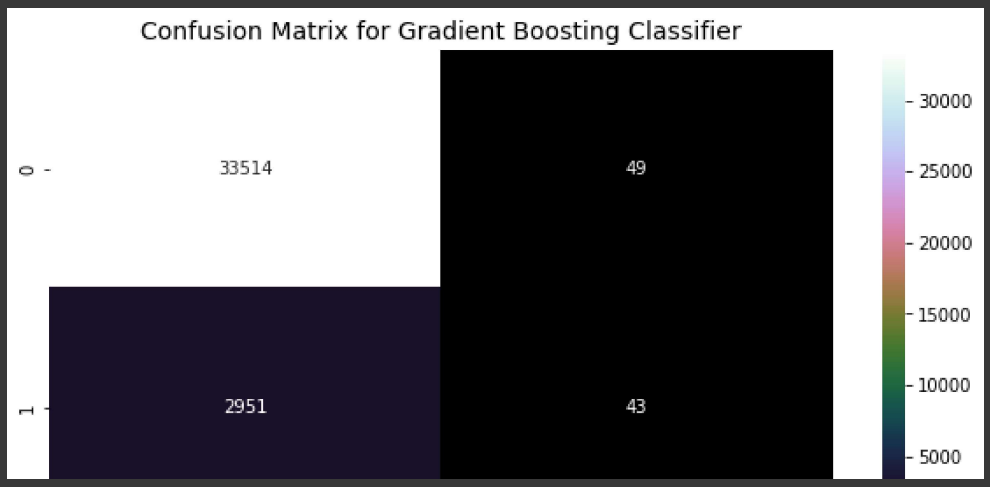
```
92.08079813362407
```

Confusion Matrix for GradientBoosting Classifier

```
cf_gdb = confusion_matrix(ytest, gdb_pred)
print(cf_gdb)
```

```
[[33514    49]
 [ 2951    43]]
```

```
plt.figure(figsize=(10,5))
plt.title('Confusion Matrix for Gradient Boosting Classifier', fontsize = 14)
sns.heatmap(cf_gdb, annot = True, fmt = 'g', cmap ='cubehelix')
plt.show()
```



Classification Report for Gradient Boosting Classifier

```
print(classification_report(ytest, gdb_pred))
```

```
                  precision    recall  f1-score   support

               0       0.92      1.00      0.96     33563
               1       0.47      0.01      0.03      2994

        accuracy                           0.92     36557
       macro avg       0.69      0.51      0.49     36557
    weighted avg       0.88      0.92      0.88     36557
```

XGBoost : eXtreme Gradient Boost

```
import xgboost as xgb
xgb = xgb.XGBClassifier()
```

```
xgb.fit(xtrain, ytrain)
```

```
XGBClassifier()
```

```
xgb_pred = xgb.predict(xtest)
print(xgb_pred)
```

```
[0 0 0 ... 0 0 0]
```

Accuracy Score for XGBoost Classifier

```
# Accuracy Score or Test Accuracy
accuracy_score(ytest, xgb_pred)*100
```

```
91.8264627841453
```

```
# Training Accuracy
xgb.score(xtrain, ytrain)*100
```

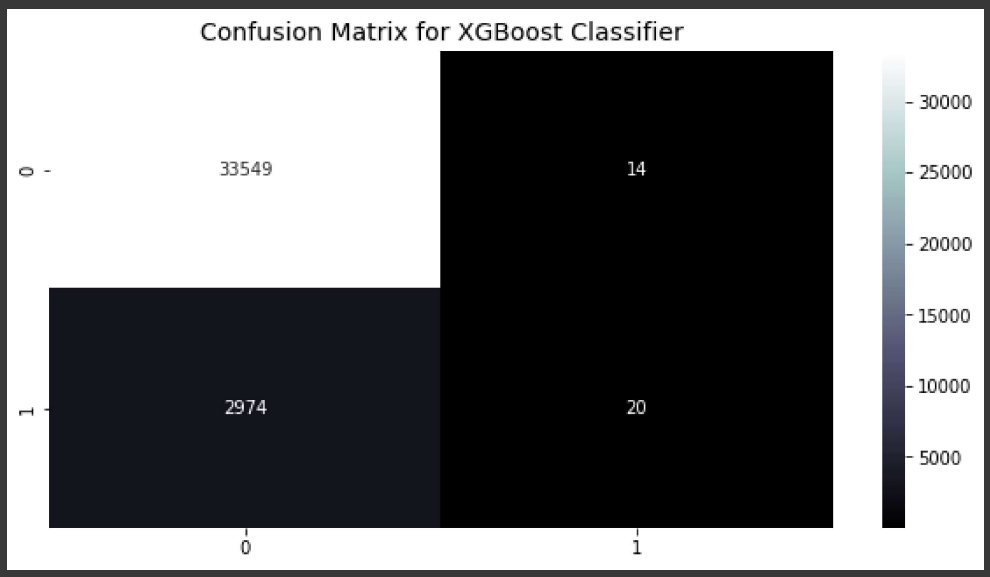
```
92.009284985756
```

Confusion Matrix for XGBoost Classifier

```
cf_xgb = confusion_matrix(ytest, xgb_pred)
print(cf_xgb)
```

```
[[33549    14]
 [ 2974    20]]
```

```
plt.figure(figsize=(10,5))
plt.title('Confusion Matrix for XGBoost Classifier', fontsize = 14)
sns.heatmap(cf_xgb, annot = True, fmt = 'g', cmap ='bone')
plt.show()
```



Classification Report XGBoost Classifier

```
print(classification_report(ytest, xgb_pred))
```

```
                  precision    recall  f1-score   support

               0       0.92      1.00      0.96     33563
               1       0.59      0.01      0.01      2994

        accuracy                           0.92     36557
       macro avg       0.75      0.50      0.49     36557
    weighted avg       0.89      0.92      0.88     36557
```

11. LGBM : Light Gradient Boosting Machine (LightGBM)

```
import lightgbm as lgb
lgb = lgb.LGBMClassifier()
```

```
lgb.fit(xtrain, ytrain)
```

```
LGBMClassifier()
```

```
lgb_pred = lgb.predict(xtest)
print(lgb_pred)
```

```
[0 0 0 ... 0 0 0]
```

Accuracy Score for LightGBM Classifier

```
# Test Accuracy
accuracy_score(ytest, lgb_pred)*100
```

```
91.87023005170009
```

```
# Training Accuracy
lgb.score(xtrain, ytrain)*100
```

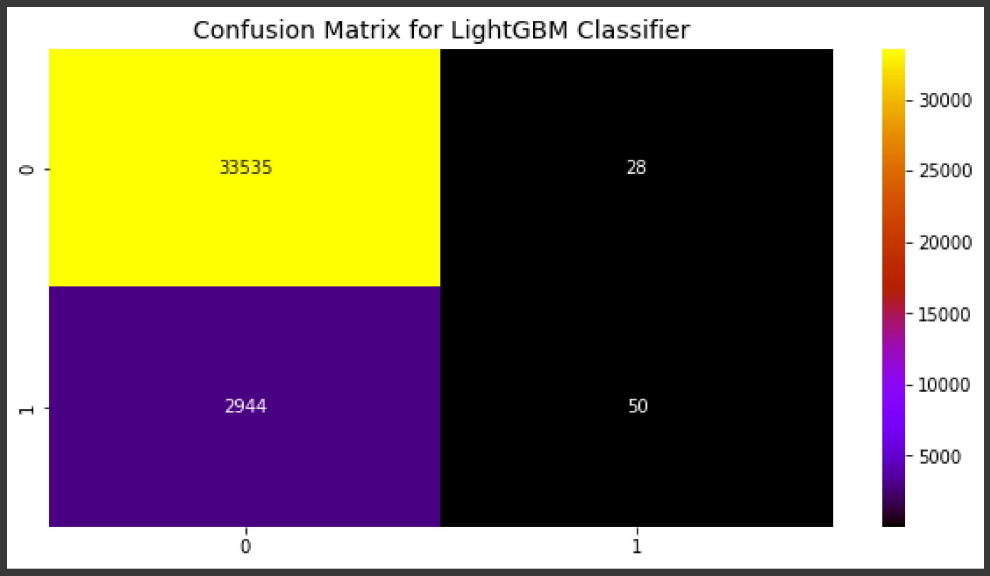
```
92.24844370977385
```

Confusion Matrix for LightGBM Classifier

```
cf_lgb = confusion_matrix(ytest, lgb_pred)
print(cf_lgb)
```

```
[[33535    28]
 [ 2944    50]]
```

```
plt.figure(figsize=(10,5))
plt.title('Confusion Matrix for LightGBM Classifier', fontsize = 14)
sns.heatmap(cf_lgb, annot = True, fmt = 'g', cmap ='gnuplot')
plt.show()
```



12. CatBoost Classifier

```
!pip install catboost
```

```
import catboost as ctb
ctb = ctb.CatBoostClassifier()
```

```
ctb.fit(xtrain, ytrain)
```

```
ctb_pred = ctb.predict(xtest)
print(ctb_pred)
```

```
[0 0 0 ... 0 0 0]
```

Accuracy Score for CatBoost Classifier

```
# Accuracy Score (Test Score)
accuracy_score(ytest,ctb_pred)*100
```

```
91.9304100445879
```

```
# Training Score
ctb.score(xtrain, ytrain)*100
```

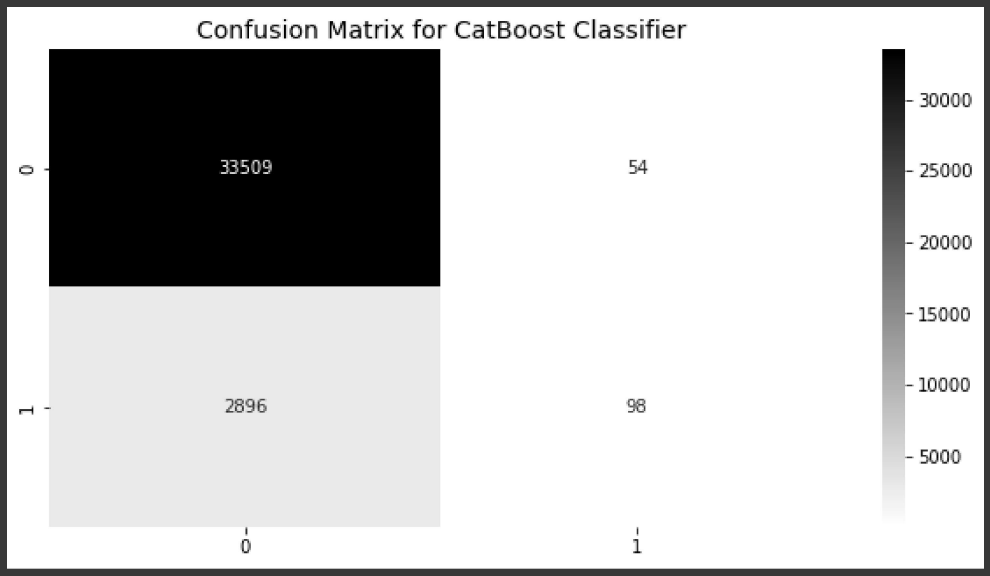
```
93.18045932543171
```

Confusion Matrix for CatBoost Classifier

```
cf_ctb = confusion_matrix(ytest, ctb_pred)
print(cf_ctb)
```

```
[[33509    54]
 [ 2896    98]]
```

```
plt.figure(figsize=(10,5))
plt.title('Confusion Matrix for CatBoost Classifier', fontsize = 14)
sns.heatmap(cf_ctb, annot = True, fmt = 'g', cmap ='binary')
plt.show()
```



Classification Report for CatBoost Classifier

```
print(classification_report(ytest, ctb_pred))
```

```
                precision    recall  f1-score   support

           0       0.92      1.00      0.96     33563
           1       0.64      0.03      0.06      2994

    accuracy                           0.92     36557
   macro avg       0.78      0.52      0.51     36557
weighted avg       0.90      0.92      0.88     36557
```

Approx all the Supervised Machine Learning Algorithms are applied to train the model for the given 'Loan Defaulter' dataset.

Project submitted by : Prince Kumar

Thank You

Colab paid products - Cancel contracts here

0s completed at 4:18 AM