CREDIT CARD DEFAULT PREDICTION

BY - Machine learning

Domain - Banking

Dataset link:

https://www.kaggle.com/datasets/uciml/default-of-credit-card-clients-dataset

```
import pandas as pd

data = pd.read_csv('Credit_Card.csv')
```

Display Top 5 Rows of Dataset

```
data.head()
```

```
   ID  LIMIT_BAL  SEX  EDUCATION  MARRIAGE  AGE  PAY_0  PAY_2  PAY_3
PAY_4  \
0   1    20000.0    2          2         1   24      2      2     -1
-1
1   2   120000.0    2          2         2   26     -1      2      0
0
2   3    90000.0    2          2         2   34      0      0      0
0
3   4    50000.0    2          2         1   37      0      0      0
0
4   5    50000.0    1          2         1   57     -1      0     -1
0

   ...  BILL_AMT4  BILL_AMT5  BILL_AMT6  PAY_AMT1  PAY_AMT2  PAY_AMT3
\
0  ...        0.0        0.0        0.0       0.0     689.0       0.0

1  ...     3272.0     3455.0     3261.0       0.0    1000.0    1000.0

2  ...    14331.0    14948.0    15549.0    1518.0    1500.0    1000.0

3  ...    28314.0    28959.0    29547.0    2000.0    2019.0    1200.0

4  ...    20940.0    19146.0    19131.0    2000.0   36681.0   10000.0


   PAY_AMT4  PAY_AMT5  PAY_AMT6  payment
0       0.0       0.0       0.0        1
1    1000.0       0.0    2000.0        1
2    1000.0    1000.0    5000.0        0
3    1100.0    1069.0    1000.0        0
4    9000.0     689.0     679.0        0
```

```
[5 rows x 25 columns]
```

Check Last 5 Rows of the Dataset

```
data.tail()
```

```
          ID  LIMIT_BAL  SEX  EDUCATION  MARRIAGE  AGE  PAY_0  PAY_2
PAY_3  \
29995  29996   220000.0    1          3         1   39      0      0
0
29996  29997   150000.0    1          3         2   43     -1     -1
-1
29997  29998    30000.0    1          2         2   37      4      3
2
29998  29999    80000.0    1          3         1   41      1     -1
0
29999  30000    50000.0    1          2         1   46      0      0
0

       PAY_4  ...  BILL_AMT4  BILL_AMT5  BILL_AMT6  PAY_AMT1  PAY_AMT2
\
29995      0  ...    88004.0    31237.0    15980.0    8500.0   20000.0

29996     -1  ...     8979.0     5190.0        0.0    1837.0    3526.0

29997     -1  ...    20878.0    20582.0    19357.0       0.0       0.0

29998      0  ...    52774.0    11855.0    48944.0   85900.0    3409.0

29999      0  ...    36535.0    32428.0    15313.0    2078.0    1800.0


       PAY_AMT3  PAY_AMT4  PAY_AMT5  PAY_AMT6  payment
29995    5003.0    3047.0    5000.0    1000.0        0
29996    8998.0     129.0       0.0       0.0        0
29997   22000.0    4200.0    2000.0    3100.0        1
29998    1178.0    1926.0   52964.0    1804.0        1
29999    1430.0    1000.0    1000.0    1000.0        1

[5 rows x 25 columns]
```

Find Shape of our Dataset (Number of Rows And Number of Columns)

```
data.shape
```

```
(30000, 25)
```

```
print("Number of Rows",data.shape[0])
print("Number of Columns",data.shape[1])
```

```
Number of Rows 30000
Number of Columns 25
```

Get informaiton About our datset like total Number Rows, Total number of Columns, Datatypes of each column and Memory Requirement

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30000 entries, 0 to 29999
Data columns (total 25 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   ID         30000 non-null  int64  
 1   LIMIT_BAL  30000 non-null  float64
 2   SEX        30000 non-null  int64  
 3   EDUCATION  30000 non-null  int64  
 4   MARRIAGE   30000 non-null  int64  
 5   AGE        30000 non-null  int64  
 6   PAY_0      30000 non-null  int64  
 7   PAY_2      30000 non-null  int64  
 8   PAY_3      30000 non-null  int64  
 9   PAY_4      30000 non-null  int64  
 10  PAY_5      30000 non-null  int64  
 11  PAY_6      30000 non-null  int64  
 12  BILL_AMT1  30000 non-null  float64
 13  BILL_AMT2  30000 non-null  float64
 14  BILL_AMT3  30000 non-null  float64
 15  BILL_AMT4  30000 non-null  float64
 16  BILL_AMT5  30000 non-null  float64
 17  BILL_AMT6  30000 non-null  float64
 18  PAY_AMT1   30000 non-null  float64
 19  PAY_AMT2   30000 non-null  float64
 20  PAY_AMT3   30000 non-null  float64
 21  PAY_AMT4   30000 non-null  float64
 22  PAY_AMT5   30000 non-null  float64
 23  PAY_AMT6   30000 non-null  float64
 24  payment    30000 non-null  int64  
dtypes: float64(13), int64(12)
memory usage: 5.7 MB
```

Check Null Values in the Dataset

```
data.isnull().sum()
```

```
ID           0
LIMIT_BAL    0
SEX          0
```

```
EDUCATION    0
MARRIAGE     0
AGE          0
PAY_0        0
PAY_2        0
PAY_3        0
PAY_4        0
PAY_5        0
PAY_6        0
BILL_AMT1    0
BILL_AMT2    0
BILL_AMT3    0
BILL_AMT4    0
BILL_AMT5    0
BILL_AMT6    0
PAY_AMT1     0
PAY_AMT2     0
PAY_AMT3     0
PAY_AMT4     0
PAY_AMT5     0
PAY_AMT6     0
payment      0
dtype: int64
```

```
data.head()
```

```
   ID  LIMIT_BAL  SEX  EDUCATION  MARRIAGE  AGE  PAY_0  PAY_2  PAY_3
PAY_4  \
0   1    20000.0    2          2         1   24      2      2     -1
-1
1   2   120000.0    2          2         2   26     -1      2      0
0
2   3    90000.0    2          2         2   34      0      0      0
0
3   4    50000.0    2          2         1   37      0      0      0
0
4   5    50000.0    1          2         1   57     -1      0     -1
0

   ...  BILL_AMT4  BILL_AMT5  BILL_AMT6  PAY_AMT1  PAY_AMT2  PAY_AMT3
\
0  ...        0.0        0.0        0.0       0.0     689.0       0.0

1  ...     3272.0     3455.0     3261.0       0.0    1000.0    1000.0

2  ...    14331.0    14948.0    15549.0    1518.0    1500.0    1000.0

3  ...    28314.0    28959.0    29547.0    2000.0    2019.0    1200.0

4  ...    20940.0    19146.0    19131.0    2000.0   36681.0   10000.0
```

```
   PAY_AMT4  PAY_AMT5  PAY_AMT6  payment
0       0.0       0.0       0.0        1
1    1000.0       0.0    2000.0        1
2    1000.0    1000.0    5000.0        0
3    1100.0    1069.0    1000.0        0
4    9000.0     689.0     679.0        0

[5 rows x 25 columns]
```

```
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
data['LIMIT_BAL']=sc.fit_transform(pd.DataFrame(data['LIMIT_BAL']))

data.head()
```

```
   ID  LIMIT_BAL  SEX  EDUCATION  MARRIAGE  AGE  PAY_0  PAY_2  PAY_3
PAY_4  \
0   1  -1.136720    2          2         1   24      2      2     -1
-1
1   2  -0.365981    2          2         2   26     -1      2      0
0
2   3  -0.597202    2          2         2   34      0      0      0
0
3   4  -0.905498    2          2         1   37      0      0      0
0
4   5  -0.905498    1          2         1   57     -1      0     -1
0

   ...  BILL_AMT4  BILL_AMT5  BILL_AMT6  PAY_AMT1  PAY_AMT2  PAY_AMT3
\
0  ...        0.0        0.0        0.0       0.0     689.0       0.0

1  ...     3272.0     3455.0     3261.0       0.0    1000.0    1000.0

2  ...    14331.0    14948.0    15549.0    1518.0    1500.0    1000.0

3  ...    28314.0    28959.0    29547.0    2000.0    2019.0    1200.0

4  ...    20940.0    19146.0    19131.0    2000.0   36681.0   10000.0


   PAY_AMT4  PAY_AMT5  PAY_AMT6  payment
0       0.0       0.0       0.0        1
1    1000.0       0.0    2000.0        1
2    1000.0    1000.0    5000.0        0
3    1100.0    1069.0    1000.0        0
4    9000.0     689.0     679.0        0
```

```
[5 rows x 25 columns]
```

```
data = data.drop(['ID'],axis=1)
```

```
data.head()
```

```
   LIMIT_BAL  SEX  EDUCATION  MARRIAGE  AGE  PAY_0  PAY_2  PAY_3
PAY_4  \
0  -1.136720    2          2         1   24      2      2     -1     -
1
1  -0.365981    2          2         2   26     -1      2      0
0
2  -0.597202    2          2         2   34      0      0      0
0
3  -0.905498    2          2         1   37      0      0      0
0
4  -0.905498    1          2         1   57     -1      0     -1
0

   PAY_5  ...  BILL_AMT4  BILL_AMT5  BILL_AMT6  PAY_AMT1  PAY_AMT2
PAY_AMT3  \
0     -2  ...        0.0        0.0        0.0       0.0     689.0
0.0
1      0  ...     3272.0     3455.0     3261.0       0.0    1000.0
1000.0
2      0  ...    14331.0    14948.0    15549.0    1518.0    1500.0
1000.0
3      0  ...    28314.0    28959.0    29547.0    2000.0    2019.0
1200.0
4      0  ...    20940.0    19146.0    19131.0    2000.0   36681.0
10000.0

   PAY_AMT4  PAY_AMT5  PAY_AMT6  payment
0       0.0       0.0       0.0        1
1    1000.0       0.0    2000.0        1
2    1000.0    1000.0    5000.0        0
3    1100.0    1069.0    1000.0        0
4    9000.0     689.0     679.0        0

[5 rows x 24 columns]
```

```
data.shape
```

```
(30000, 24)
```

```
data.duplicated().any()
```

```
True
```

```
data = data.drop_duplicates()
```

```
data.shape
```

```
(29965, 24)
```

```
30000-29965
```

```
35
```

Not Handling Imbalanced

```
data['payment'].value_counts()
```

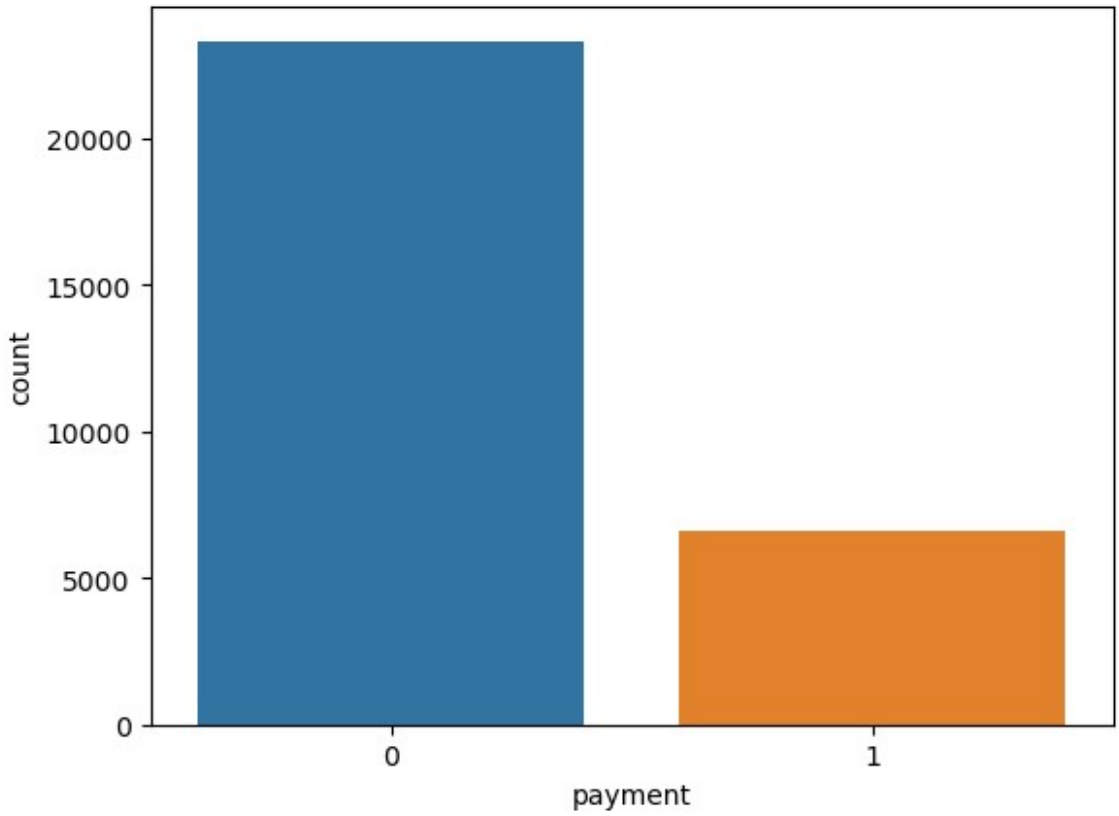
```
0    23335
1     6630
Name: payment, dtype: int64
```

```
import seaborn as sns
```

```
sns.countplot(data['payment'])
```

```
C:\Users\Hp\anaconda3\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variable as a keyword arg: x. From
version 0.12, the only valid positional argument will be `data`, and
passing other arguments without an explicit keyword will result in an
error or misinterpretation.
  warnings.warn(
```

```
<AxesSubplot:xlabel='payment', ylabel='count'>
```



Store Feature Matrix in X and Response (target) in Vector Y

```
X = data.drop('payment',axis=1)
Y = data['payment']
```

Splitting the dataset into training set and Test set

```
from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test =
train_test_split(X,Y,test_size=0.20,random_state=42)
```

Handling imbalanced Dataset

- Undersampling
- Oversampling

Undersampling

```
normal = data[data['payment']==0]
fraud = data[data['payment']==1]

normal.shape
```

```
(23335, 24)
```

```
fraud.shape
```

```
(6630, 24)
```

```
normal_sample=normal.sample(n=6630)
```

```
normal_sample.shape
```

```
(6630, 24)
```

```
new_data = pd.concat([normal_sample,fraud],ignore_index=True)
```

```
new_data['payment'].value_counts()
```

```
0    6630
1    6630
Name: payment, dtype: int64
```

```
new_data.head()
```

```
        LIMIT_BAL  SEX  EDUCATION  MARRIAGE  AGE  PAY_0  PAY_2  PAY_3
PAY_4  \
808     0.173537    2          1         2   26      0      0      0
0
14355   0.250611    1          2         1   40      1      2      2
0
4634   -0.057685    1          2         2   27      0      0      2
2
11731  -1.136720    1          2         1   51      0      0      0
0
922    -0.443054    1          1         1   40      0      0      0
0

        PAY_5  ...  BILL_AMT4  BILL_AMT5  BILL_AMT6  PAY_AMT1  PAY_AMT2
\
808         0  ...    33726.0    35758.0    35171.0    1750.0    1720.0

14355      -1  ...   234177.0   201490.0   190721.0    5133.0       0.0

4634        0  ...   116284.0   119248.0   122228.0   12000.0    5000.0

11731       0  ...    18235.0    16259.0    10640.0    1355.0    1136.0

922         0  ...    22204.0    26784.0    31352.0    4980.0    5124.0


        PAY_AMT3  PAY_AMT4  PAY_AMT5  PAY_AMT6  payment
808       1300.0    2600.0       0.0    1400.0        0
14355     5857.0  202076.0    6726.0    5346.0        0
4634         0.0    5000.0    4958.0   36000.0        0
11731      646.0     381.0     337.0    1251.0        0
922       5256.0    5416.0    5552.0    5742.0        0

[5 rows x 24 columns]
```

```
X = data.drop('payment',axis=1)
Y = data['payment']

from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test =
train_test_split(X,Y,test_size=0.20,random_state=42)
```

Logistic Regression

```
from sklearn.linear_model import LogisticRegression
log = LogisticRegression()
log.fit(X_train,Y_train)
```

```
C:\Users\Hp\anaconda3\lib\site-packages\sklearn\linear_model\
_logistic.py:814: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

```
LogisticRegression()
```

```
y_pred1 = log.predict(X_test)
```

```
from sklearn.metrics import accuracy_score
```

```
accuracy_score(Y_test,y_pred1)
```

```
0.779743033539129
```

```
from sklearn.metrics import precision_score,recall_score,f1_score
```

```
precision_score(Y_test,y_pred1)
```

```
0.5
```

```
recall_score(Y_test,y_pred1)
```

```
0.0007575757575757576
```

```
f1_score(Y_test,y_pred1)
```

```
0.0015128593040847202
```

Decisison Tree Classifer

```
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier()
dt.fit(X_train,Y_train)
```

```
DecisionTreeClassifier()
```

```
y_pred2 = dt.predict(X_test)
```

```
accuracy_score(Y_test,y_pred2)
```

```
0.7313532454530285
```

```
precision_score(Y_test,y_pred2)
```

```
0.39146706586826346
```

```
recall_score(Y_test,y_pred2)
```

```
0.39621212121212124
```

```
f1_score(Y_test,y_pred2)
```

```
0.3938253012048193
```

Random Forest Classifier

```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier()
rf.fit(X_train,Y_train)
```

```
RandomForestClassifier()
```

```
y_pred3 = rf.predict(X_test)
```

```
accuracy_score(Y_test,y_pred3)
```

```
0.8147839145669948
```

```
precision_score(Y_test,y_pred3)
```

```
0.6450276243093923
```

```
recall_score(Y_test,y_pred3)
```

```
0.35378787878787876
```

```
f1_score(Y_test,y_pred3)
```

```
0.4569471624266145
```

```
final_data = pd.DataFrame({'Models':['LR','DT','RF'],
              "AC":[accuracy_score(Y_test,y_pred1)*100,
```

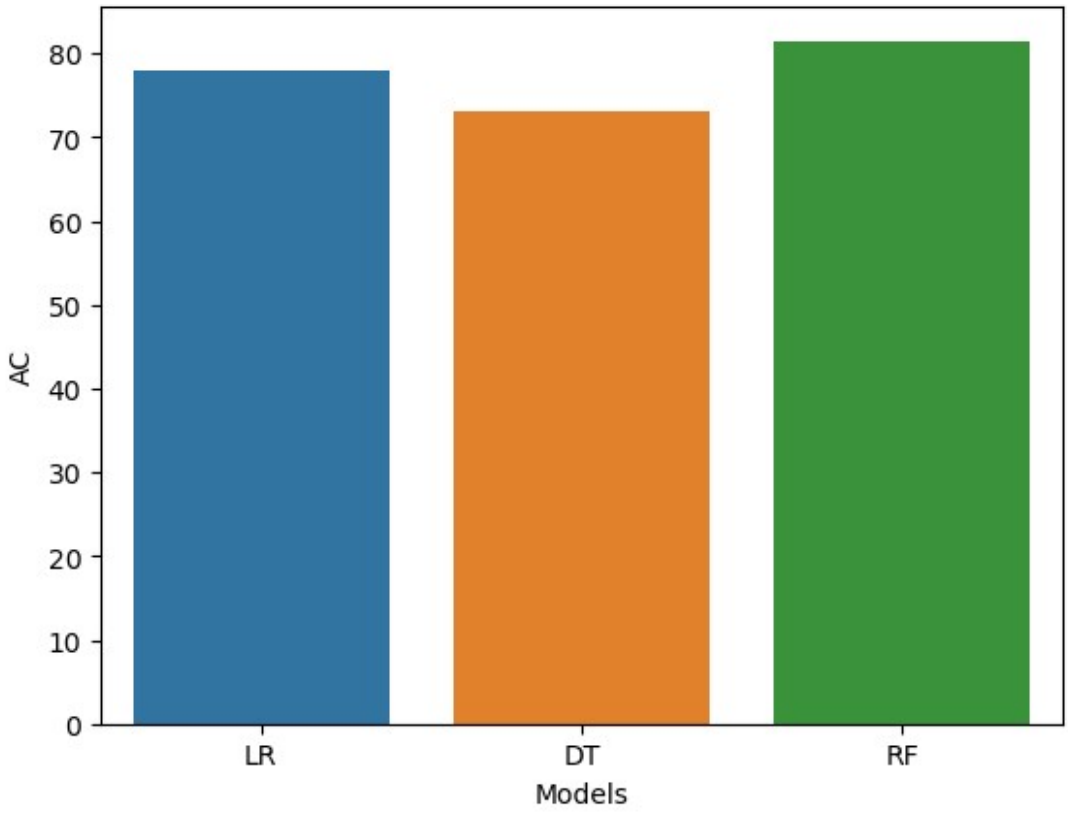
```
                    accuracy_score(Y_test,y_pred2)*100,
                    accuracy_score(Y_test,y_pred3)*100
                   ]})

final_data
```

```
  Models         AC
0     LR  77.974303
1     DT  73.135325
2     RF  81.478391
```

```
sns.barplot(final_data['Models'],final_data['AC'])
```

```
C:\Users\Hp\anaconda3\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variables as keyword args: x, y.
From version 0.12, the only valid positional argument will be `data`,
and passing other arguments without an explicit keyword will result in
an error or misinterpretation.
  warnings.warn(
```

```
<AxesSubplot:xlabel='Models', ylabel='AC'>
```



Oversampling

```
X = data.drop('payment',axis=1)
Y = data['payment']

X.shape

(29965, 23)

Y.shape

(29965,)

from imblearn.over_sampling import SMOTE

X_res,y_res = SMOTE().fit_resample(X,Y)

y_res.value_counts()

1    23335
0    23335
Name: payment, dtype: int64

from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test =
train_test_split(X_res,y_res,test_size=0.20,random_state=42)
```

Logistic Regression

```
log = LogisticRegression()
log.fit(X_train,Y_train)

LogisticRegression()

y_pred1 = log.predict(X_test)

accuracy_score(Y_test,y_pred1)

0.6079922862652668

precision_score(Y_test,y_pred1)

0.573664009447889

recall_score(Y_test,y_pred1)

0.8344427743182307

f1_score(Y_test,y_pred1)

0.6799055200769837
```

Decision Tree Classifier

```
dt=DecisionTreeClassifier()
dt.fit(X_train,Y_train)
```

```
DecisionTreeClassifier()
```

```
y_pred2 = dt.predict(X_test)
```

```
accuracy_score(Y_test,y_pred2)
```

```
0.7557317334476109
```

```
precision_score(Y_test,y_pred2)
```

```
0.7427987742594484
```

```
recall_score(Y_test,y_pred2)
```

```
0.780760146016749
```

```
f1_score(Y_test,y_pred2)
```

```
0.7613065326633165
```

Random Forest Classifier

```
rf = RandomForestClassifier()
rf.fit(X_train,Y_train)
```

```
RandomForestClassifier()
```

```
y_pred3 = rf.predict(X_test)
```

```
accuracy_score(Y_test,y_pred3)
```

```
0.8458324405399614
```

```
precision_score(Y_test,y_pred3)
```

```
0.8502394427514149
```

```
recall_score(Y_test,y_pred2)
```

```
0.780760146016749
```

```
f1_score(Y_test,y_pred2)
```

```
0.7613065326633165
```

```
final_data = pd.DataFrame({'Models':['LR','DT','RF'],
              "AC":[accuracy_score(Y_test,y_pred1)*100,
                    accuracy_score(Y_test,y_pred2)*100,
```

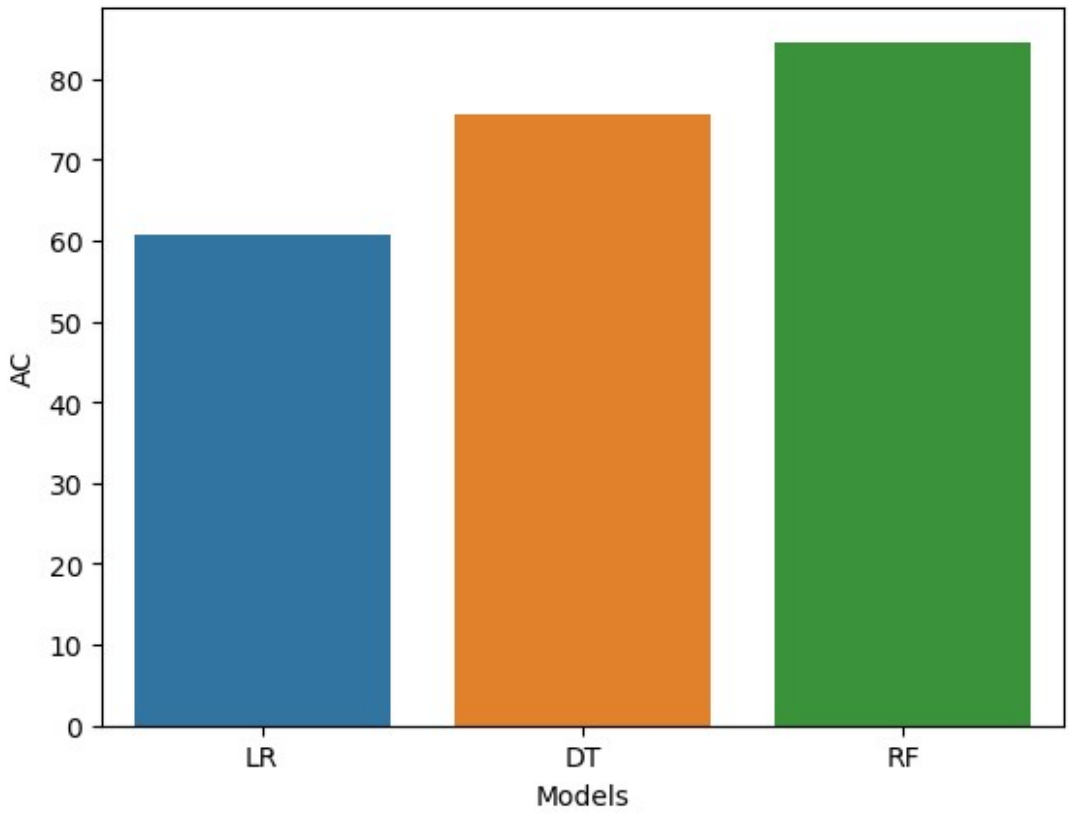
```
                    accuracy_score(Y_test,y_pred3)*100
                   ]})

final_data
```

```
  Models         AC
0     LR  60.799229
1     DT  75.573173
2     RF  84.583244
```

```
sns.barplot(final_data['Models'],final_data['AC'])
```

```
C:\Users\Hp\anaconda3\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variables as keyword args: x, y.
From version 0.12, the only valid positional argument will be `data`,
and passing other arguments without an explicit keyword will result in
an error or misinterpretation.
  warnings.warn(
```

```
<AxesSubplot:xlabel='Models', ylabel='AC'>
```



Save the Model

```
rf1 = RandomForestClassifier()
rf1.fit(X_res,y_res)
```

```
RandomForestClassifier()
```

```
import joblib
```

```
joblib.dump(rf1,"credit_card_model")
```

```
['credit_card_model']
```

```
model = joblib.load("credit_card_model")
```

```
model
```

```
RandomForestClassifier()
```

```
pred = model.predict(X_test)
```

```
if (pred == 0).all():
    print("All transactions are normal")
else:
    print("At least one transaction is fraudulent")
```

```
At least one transaction is fraudulent
```

```
if (pred == 0).any():
    print("transaction is normal")
else:
    print("All transactions are fraudulent")
```

```
transaction is normal
```