Capstone_Project Prediction of Credit Card fraud By Rishabh Yadav

In [1]:
```
#Importing libraries
import pandas as pd
```

In [2]:
```
# Reading the dataset into DataFrame object data
data = pd.read_csv('creditcard.csv')
```

In [3]:
```
# Using pandas library to view the whole set of columns in the dataset
pd.options.display.max_columns = None
```

In [4]:
```
# Displaying the first few rows (5) of the dataset
data.head()
```

Out[4]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.3
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.2
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.5
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.3
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.8

In [5]:
```
# Displaying the last few rows of the dataset
data.tail()
```

Out[5]:

	Time	V1	V2	V3	V4	V5	V6	V7	
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.3
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.2
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.7
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.6
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.4

In [6]:
```
# A look at the number of rows and columns in the dataset
data.shape
```

Out[6]:
```
(284807, 31)
```

In [7]:
```
# Printing the number of rows in the dataset
print("Numbers of Rows", data.shape[0])

# Printing the number of columns in the dataset
print("Numbers of Columns", data.shape[1])
```

```
Numbers of Rows 284807
Numbers of Columns 31
```

In [8]:
```
# Displaying information about the dataset, such as data types, memory usage, and n
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
 9   V9      284807 non-null  float64
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
 13  V13     284807 non-null  float64
 14  V14     284807 non-null  float64
 15  V15     284807 non-null  float64
 16  V16     284807 non-null  float64
 17  V17     284807 non-null  float64
 18  V18     284807 non-null  float64
 19  V19     284807 non-null  float64
 20  V20     284807 non-null  float64
 21  V21     284807 non-null  float64
 22  V22     284807 non-null  float64
 23  V23     284807 non-null  float64
 24  V24     284807 non-null  float64
 25  V25     284807 non-null  float64
 26  V26     284807 non-null  float64
 27  V27     284807 non-null  float64
 28  V28     284807 non-null  float64
 29  Amount  284807 non-null  float64
 30  Class   284807 non-null  int64  
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```

In [9]:
```
# Calculating the number of missing values in each column of the dataset
data.isnull().sum()
```

Out[9]:
```
Time      0
V1        0
V2        0
V3        0
V4        0
V5        0
V6        0
V7        0
V8        0
V9        0
V10       0
V11       0
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       0
V19       0
V20       0
V21       0
V22       0
V23       0
V24       0
V25       0
V26       0
V27       0
V28       0
Amount    0
Class     0
dtype: int64
```

In [10]:
```
# Displaying the first few rows (5) of the dataset
data.head()
```

Out[10]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.3
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.2
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.5
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.3
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.8

In [11]:
```
from sklearn.preprocessing import StandardScaler
```

In [12]:
```
sc = StandardScaler()
data['Amount'] = sc.fit_transform(pd.DataFrame(data['Amount']))
```

In [13]:
```
data.head()
```

Out[13]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.3
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.2
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.5
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.3
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.8

In [14]:
```
# Dropping the 'Time' column from the dataset
data = data.drop(['Time'],axis=1)
```

In [15]:
```
data.head()
```

Out[15]:

	V1	V2	V3	V4	V5	V6	V7	V8	V9
0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787
1	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425
2	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654
3	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024
4	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739

In [16]:
```
# Shape of the dataset after dropping the columns
data.shape
```

Out[16]:
```
(284807, 30)
```

In [17]:
```
# Checking if there are any duplicated rows in the dataset
data.duplicated().any()
```

Out[17]:
```
True
```

In [18]:
```
# Removing duplicated rows from the dataset
data = data.drop_duplicates()
```

In [19]:
```
# Getting the dimensions of the dataset after removing duplicates
data.shape
```

Out[19]:
```
(275663, 30)
```

In [20]:
```
284807 - 275663
```

Out[20]:
```
9144
```

Checking Data is Balance or not

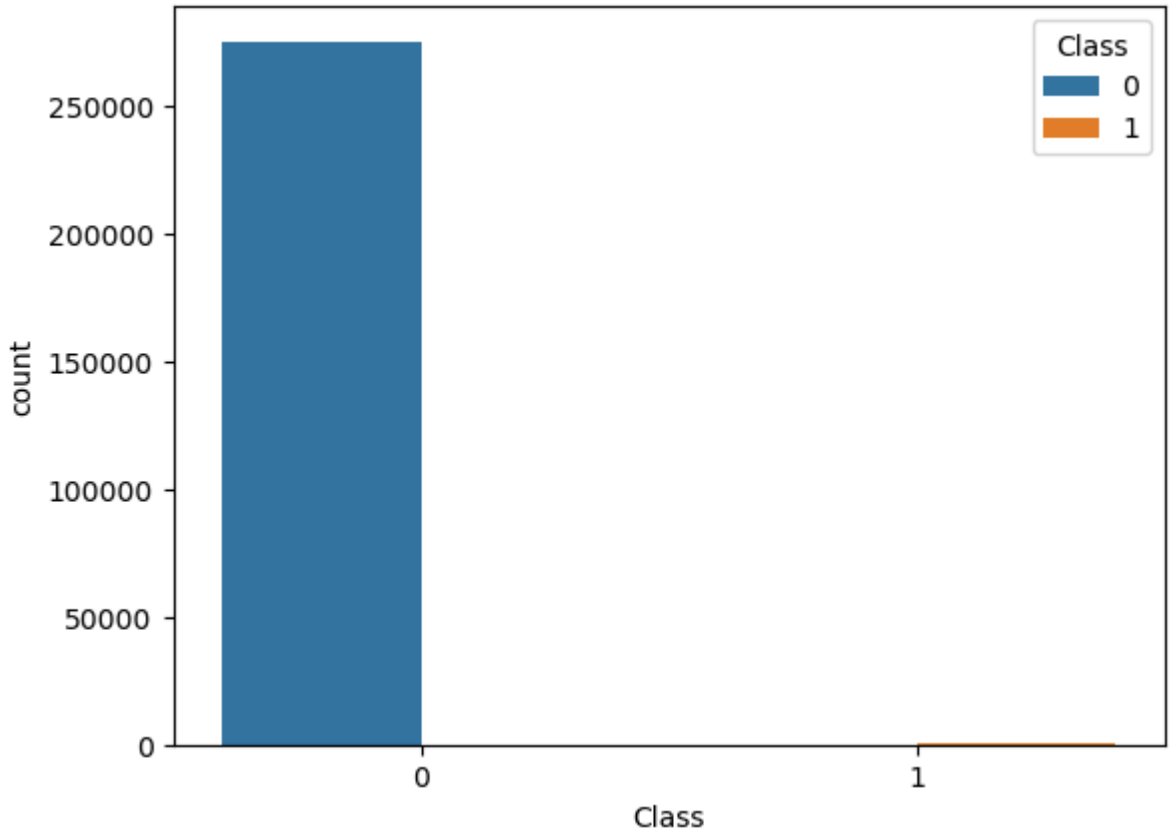
In [21]:
```
data['Class'].value_counts()
```

Out[21]:
```
0    275190
1       473
Name: Class, dtype: int64
```

In [22]:
```
import seaborn as sns
```

In [23]:
```
# Creating a count plot using Seaborn to visualize the distribution of the 'Class'
sns.countplot(data=data, x= 'Class',hue ='Class')
```

Out[23]:
```
<Axes: xlabel='Class', ylabel='count'>
```



In [24]:
```
# Store Feature Matrix In X And Response (Target) In Vector y
X = data.drop('Class',axis=1)
y = data['Class']
```

In [25]:
```
# Importing train_test_split function from sklearn.model_selection
# Splitting The Dataset Into The 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

In [26]:
```
# Creating a subset of the dataset containing only non-fraudulent transactions
unfraud = data[data['Class']==0]

# Creating a subset of the dataset containing only fraudulent transactions
fraud = data[data['Class']==1]
```

In [27]:
```
# Getting the dimensions of the subset containing only non-fraudulent transactions
unfraud.shape
```

Out[27]:
```
(275190, 30)
```

In [28]:
```
# Getting the dimensions of the subset containing only fraudulent transactions
fraud.shape
```

Out[28]:
```
(473, 30)
```

In [29]:
```
unfraud_sample = unfraud.sample(n=473)
```

In [30]:
```
unfraud_sample.shape
```

Out[30]:
```
(473, 30)
```

In [31]:
```
# Concatenating the sample subset of non-fraudulent transactions and the subset of
new_data = pd.concat([unfraud_sample,fraud],ignore_index=True)
```

In [32]:
```
new_data['Class'].value_counts()
```

Out[32]:
```
0    473
1    473
Name: Class, dtype: int64
```

In [33]:
```
new_data.head()
```

Out[33]:

	V1	V2	V3	V4	V5	V6	V7	V8	V9
0	1.384797	-0.811723	0.998531	-0.806756	-1.359144	-0.042123	-1.304257	0.071747	-0.159205
1	1.251877	-0.533420	0.031715	-0.708695	-0.647884	-0.156149	-0.760031	0.137998	-0.916860
2	-0.444882	0.947597	2.164558	-0.517446	0.222028	-1.235420	1.350962	-0.689905	-0.263424
3	-0.598605	1.282699	-0.308352	-0.562917	0.733795	-0.632448	1.074496	-0.579998	0.143414
4	-15.602753	-4.979618	-7.058110	-1.381307	-3.383137	1.248084	-0.008572	1.953405	3.410060

In [34]:
```
# Creating feature matrix X by dropping the 'Class' column from the concatenated da
# Creating target vector y containing only the 'Class' column from the concatenated
X = new_data.drop('Class',axis=1)
y = new_data['Class']
```

In [35]:
```
# Splitting the concatenated dataset into training and testing sets
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 Model

In [36]:
```
# Importing Logistic Regression model from sklearn.linear_model
# Creating an instance of Logistic Regression model
# Fitting the model on the training data
from sklearn.linear_model import LogisticRegression
log1 = LogisticRegression()
log1.fit(X_train,y_train)
```

Out[36]:
```
LogisticRegression()
```

In [37]:
```
# Generating predictions on the testing data using the trained Logistic Regression
y_predict1 = log1.predict(X_test)
```

In [38]:
```
# Importing the accuracy_score function from sklearn.metrics
from sklearn.metrics import accuracy_score
```

In [39]:
```
# Calculating the accuracy score by comparing the actual target values (y_test) wit
accuracy_score(y_test,y_predict1)
```

Out[39]:
```
0.9526315789473684
```

In [40]:
```
# Importing precision_score, recall_score, and f1_score functions from sklearn.metr
from sklearn.metrics import precision_score,recall_score,f1_score
```

In [41]:
```
# Calculating the precision score by comparing the actual target values (y_test) wi
precision_score(y_test,y_predict1)
```

Out[41]:
```
0.9894736842105263
```

In [42]:
```
# Calculating the recall score by comparing the actual target values (y_test) with
recall_score(y_test,y_predict1)
```

Out[42]:
```
0.9215686274509803
```

In [43]:
```
# Calculating the F1 score by comparing the actual target values (y_test) with the
f1_score(y_test,y_predict1)
```

Out[43]:
```
0.9543147208121827
```

Decision Tree Classifier Model

In [44]:
```
# Importing Decision Tree Classifier from sklearn.tree
# Creating an instance of Decision Tree Classifier
# Fitting the model on the training data
from sklearn.tree import DecisionTreeClassifier
dtc1 = DecisionTreeClassifier()
dtc1.fit(X_train,y_train)
```

Out[44]:
```
DecisionTreeClassifier()
```

In [45]:
```
# Generating predictions on the testing data using the trained Decision Tree Classi
y_predict2 = dtc1.predict(X_test)
```

In [46]:
```
# Calculating the accuracy score by comparing the actual target values (y_test) wit
accuracy_score(y_test,y_predict2)
```

Out[46]:
```
0.9052631578947369
```

In [47]:
```
# Calculating the precision score by comparing the actual target values (y_test) wi
precision_score(y_test,y_predict2)
```

Out[47]:
```
0.9285714285714286
```

In [48]:
```
# Calculating the recall score by comparing the actual target values (y_test) with
recall_score(y_test,y_predict2)
```

Out[48]:
```
0.8921568627450981
```

In [49]:
```
# Calculating the F1 score by comparing the actual target values (y_test) with the
f1_score(y_test,y_predict2)
```

Out[49]:
```
0.9099999999999999
```

Random Forest Classifier Model

In [50]:
```
# Importing RandomForestClassifier from sklearn.ensemble
# Creating an instance of RandomForestClassifier
# Fitting the model on the training data
from sklearn.ensemble import RandomForestClassifier
rf1 = RandomForestClassifier()
rf1.fit(X_train,y_train)
```

Out[50]:
```
▾ RandomForestClassifier
RandomForestClassifier()
```

In [51]:
```
# Generating predictions on the testing data using the trained RandomForestClassifi
y_predict3 = rf1.predict(X_test)
```

In [52]:
```
# Calculating the accuracy score by comparing the actual target values (y_test) wit
accuracy_score(y_test,y_predict3)
```

Out[52]:
```
0.9421052631578948
```

In [53]:
```
# Calculating the precision score by comparing the actual target values (y_test) wi
precision_score(y_test,y_predict3)
```

Out[53]:
```
0.989247311827957
```

In [54]:
```
# Calculating the recall score by comparing the actual target values (y_test) with
recall_score(y_test,y_predict3)
```

Out[54]:
```
0.9019607843137255
```

In [55]:
```
# Calculating the F1 score by comparing the actual target values (y_test) with the
f1_score(y_test,y_predict3)
```

Out[55]:
```
0.9435897435897437
```

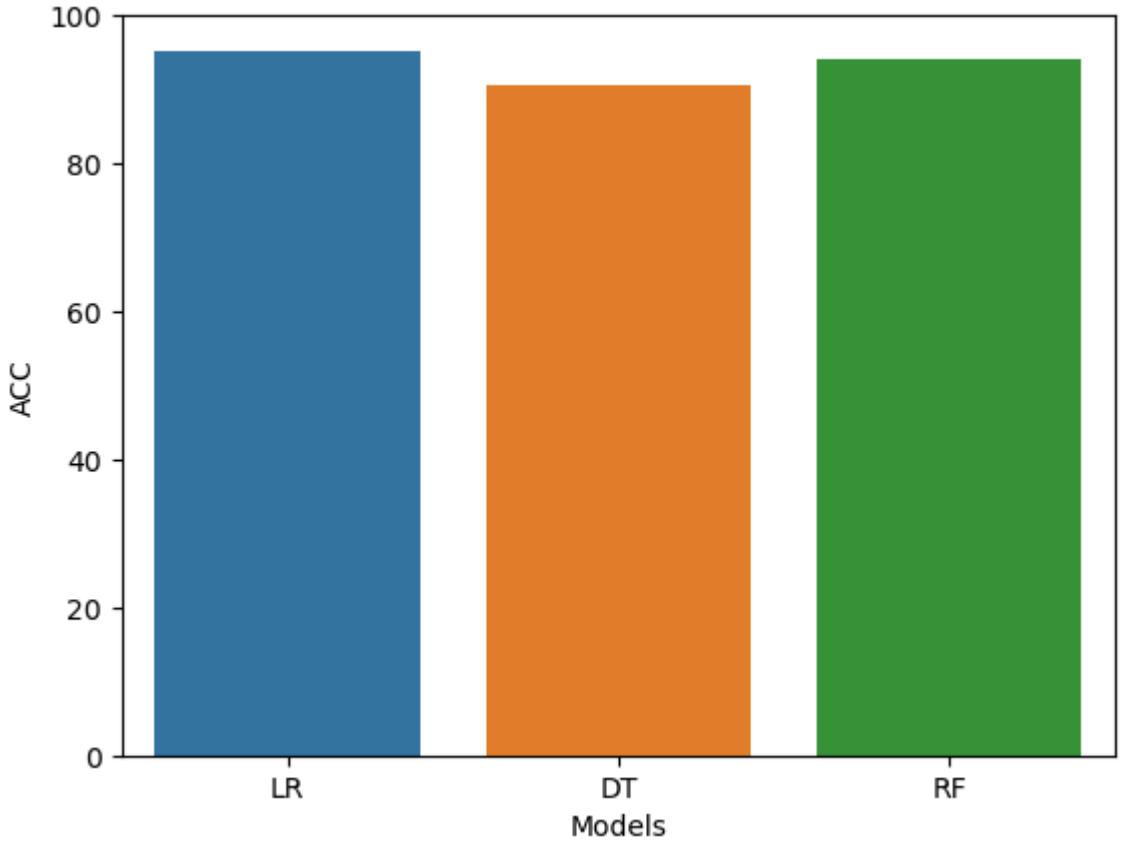
In [56]:
```
# Creating a DataFrame to store the evaluation results of different models
final_data = pd.DataFrame({'Models':['LR','DT','RF'],
              "ACC":[accuracy_score(y_test,y_predict1)*100,
                     accuracy_score(y_test,y_predict2)*100,
                     accuracy_score(y_test,y_predict3)*100
                    ]})
final_data
```

Out[56]:

	Models	ACC
0	LR	95.263158
1	DT	90.526316
2	RF	94.210526

In [57]:
```
sns.barplot(data=final_data, x='Models',y='ACC')
```

Out[57]:
```
<Axes: xlabel='Models', ylabel='ACC'>
```



Oversampling

In [58]:
```
X = data.drop('Class',axis=1)
Y = data['Class']
```

In [59]:
```
# Getting the dimensions of the feature matrix X (number of rows, number of columns
X.shape
```

Out[59]:
```
(275663, 29)
```

In [60]:
```
# Getting the dimensions of the target vector Y (number of elements)
Y.shape
```

Out[60]:
```
(275663,)
```

In [61]:
```
from imblearn.over_sampling import SMOTE
```

In [62]:
```
X_res,Y_res = SMOTE().fit_resample(X,Y)
```

In [63]:
```
Y_res.value_counts()
```

Out[63]:
```
0    275190
1    275190
Name: Class, dtype: int64
```

In [64]:
```
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 Model

In [65]:
```
# Creating an instance of Logistic Regression model# Fitting the model on the train
# Fitting the model on the training data
log2 = LogisticRegression()
log2.fit(X_train,Y_train)
```

Out[65]:
```
LogisticRegression
LogisticRegression()
```

In [66]:
```
Y_predict1 = log2.predict(X_test)
```

In [67]:
```
accuracy_score(y_test,y_predict1)
```

Out[67]:
```
0.9526315789473684
```

In [68]:
```
precision_score(y_test,y_predict1)
```

Out[68]:
```
0.9894736842105263
```

In [69]:
```
recall_score(y_test,y_predict1)
```

Out[69]:
```
0.9215686274509803
```

In [70]:
```
f1_score(y_test,y_predict1)
```

Out[70]:
```
0.9543147208121827
```

Decision Tree Classifier Model

In [71]:
```
# Creating an instance of Decision Tree Classifier model
# Fitting the model on the training data (resampled data)
dtc2=DecisionTreeClassifier()
dtc2.fit(X_train,Y_train)
```

Out[71]:
```
DecisionTreeClassifier
DecisionTreeClassifier()
```

In [72]:
```
Y_predict2 = dtc2.predict(X_test)
```

In [73]:
```
accuracy_score(Y_test,Y_predict2)
```

Out[73]:
```
0.9982920891020749
```

In [74]:
```
precision_score(Y_test,Y_predict2)
```

Out[74]:
```
0.9976395823876532
```

In [75]:
```
recall_score(Y_test,Y_predict2)
```

Out[75]:
```
0.9989455120629784
```

In [76]:
```
f1_score(Y_test,Y_predict2)
```

Out[76]:
```
0.9982921201329966
```

Random Forest Classifier Model

In [77]:
```
# Creating an instance of RandomForestClassifier model
# Fitting the model on the training data (resampled data)
rf2 = RandomForestClassifier()
rf2.fit(X_train,Y_train)
```

Out[77]:
```
RandomForestClassifier
RandomForestClassifier()
```

In [78]:
```
Y_predict3 = rf2.predict(X_test)
```

In [79]:
```
accuracy_score(Y_test,Y_predict3)
```

Out[79]:
```
0.9999091536756423
```

In [80]:
```
precision_score(Y_test,Y_predict3)
```

Out[80]:
```
0.999818224783233
```

In [81]:
```
recall_score(Y_test,Y_predict3)
```

Out[81]:
```
1.0
```

In [82]:
```
f1_score(Y_test,Y_predict3)
```

Out[82]:
```
0.9999091041303083
```

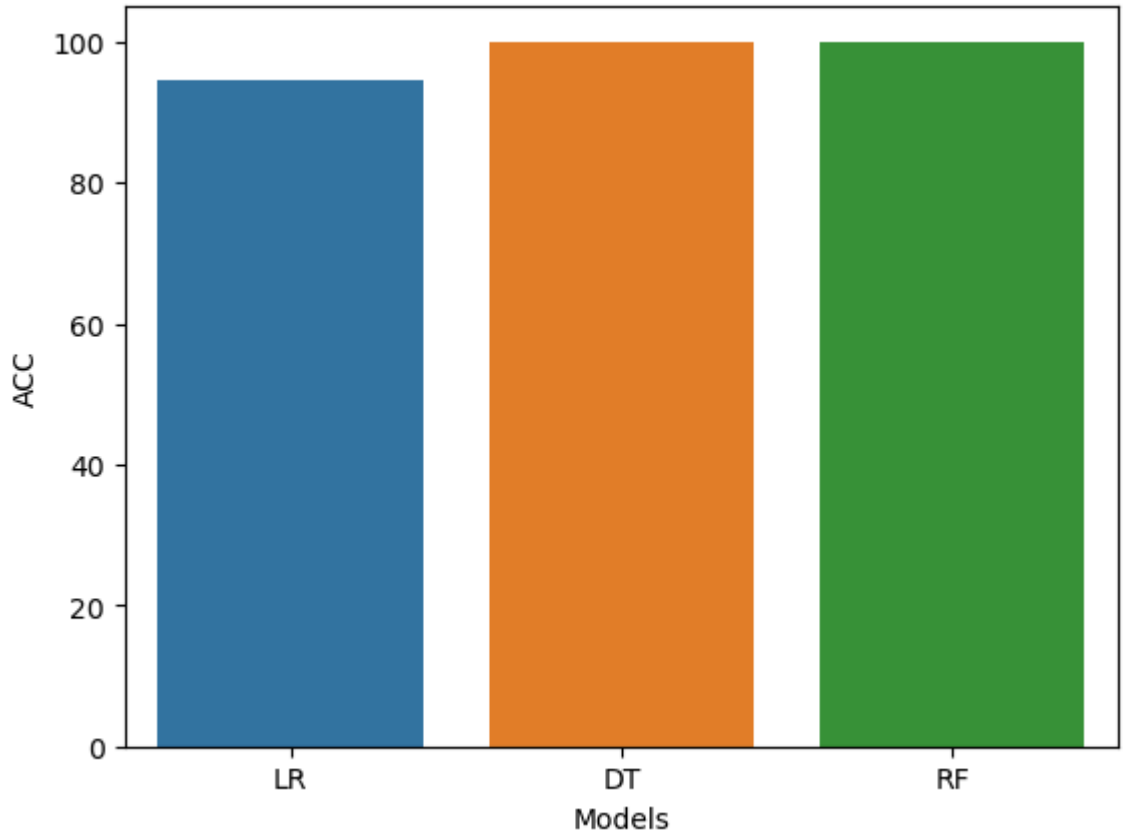
In [83]:
```
final_data = pd.DataFrame({'Models':['LR','DT','RF'],
              "ACC":[accuracy_score(Y_test,Y_predict1)*100,
                     accuracy_score(Y_test,Y_predict2)*100,
                     accuracy_score(Y_test,Y_predict3)*100
                    ]})
final_data
```

Out[83]:

	Models	ACC
0	LR	94.470184
1	DT	99.829209
2	RF	99.990915

In [84]:
```
sns.barplot(data=final_data, x='Models',y='ACC')
```

Out[84]:
```
<Axes: xlabel='Models', ylabel='ACC'>
```



Save The Model

In [85]:
```
import joblib
```

In [86]:
```
# Save the Logistic Regression Model
joblib.dump(log1,"LogisticRegression_CC_model1")
```

Out[86]:
```
['LogisticRegression_CC_model1']
```

In [87]:
```
# Save the Decision Tree Classifier Model
joblib.dump(dtc1,"Decision_Tree_CC_model1")
```

Out[87]:
```
['Decision_Tree_CC_model1']
```

In [88]:
```
# Save the Random Forest Classifier Model
joblib.dump(rf1,"Random_Forest_CC_model1")
```

Out[88]:
```
['Random_Forest_CC_model1']
```

In [89]:
```
joblib.dump(log2,"LogisticRegression_CC_model2")
```

Out[89]:
```
['LogisticRegression_CC_model2']
```

In [90]:
```
joblib.dump(dtc2,"Decision_Tree_CC_model2")
```

Out[90]:
```
['Decision_Tree_CC_model2']
```

In [91]:
```
joblib.dump(rf2,"Random_Forest_CC_model2")
```

Out[91]:
```
['Random_Forest_CC_model2']
```

In [92]:
```
model = joblib.load("Random_Forest_CC_model2")
```

In [93]:
```
predict = model.predict([[1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
```

```
C:\Users\risha\anaconda3\Lib\site-packages\sklearn\base.py:439: UserWarning: X doe
s not have valid feature names, but RandomForestClassifier was fitted with feature
names
  warnings.warn(
```

In [94]:
```
if predict == 0:
    print("Normal Transcation")
else:
    print("Fraudulent Transcation")
```

```
Normal Transcation
```