Date: 17-10-2023

Project Title: Credit Card Fraudlent Detection

Team ID: 3890

Importing required libraries

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
from matplotlib import pyplot as plt
```

Set the jupyter notebook to show maximum number of columns

In [2]:
```
pd.options.display.max_columns = None
```

Loading the datasets

In [3]:
```
ccfd = pd.read_csv("C:\\Users\\jnave\\OneDrive\\Documents\\IBM Applied
```

Displaying top 5 rows

In [4]:
```
ccfd.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
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533

Displaying bottom 5 rows

In [5]:
```
ccfd.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
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006

Shows number of rows and columns

In [6]:
```
print("Number of rows in given dataset ",ccfd.shape[0])
print("Number of columns in the given dataset ",ccfd.shape[1])
```

```
Number of rows in given dataset  284807
Number of columns in the given dataset  31
```

Getting basis information

In [7]:
```
ccfd.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
```

Checking null values in the given data

In [8]:
```
ccfd.isnull().sum()
```

Out[8]:
```
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
```

Scaling the Amount features, removing the independent columns

In [9]:
```
#removing the column name Time, it is unnecessary to our training purpo
ccfd.head(2)
```

Out[9]:

	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
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102

In [ ]:
```
#time features is unnecessary here
ccfd.drop('Time',axis = 1,inplace=True).head()
```

In [11]:
```
ccfd.head()
```

Out[11]:

	V1	V2	V3	V4	V5	V6	V7	V8	
0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.36
1	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.25
2	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.51
3	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.38
4	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.81

Scaling the Amount column data

In [12]:
```
from sklearn.preprocessing import StandardScaler
```

In [13]:
```
ss = StandardScaler()
```

In [14]:
```
ccfd['Amounts'] = ss.fit_transform(pd.DataFrame(ccfd['Amount']))
```

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

Out[15]:

	V1	V2	V3	V4	V5	V6	V7	V8	
0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.36
1	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.25
2	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.51
3	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.38
4	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.81

In [16]:
```
ccfd.shape
```

Out[16]: (284807, 31)

In [17]:
```
ccfd.drop('Amount',axis=1,inplace=True)
```

In [18]:
```
ccfd.shape
```

Out[18]: (284807, 30)

Dropping the duplicate records

In [19]:
```
ccfd.duplicated().any()
```

Out[19]: True

In [20]:
```
ccfd.drop_duplicates(inplace=True)
```

In [21]:
```
ccfd.shape
```

Out[21]: (275663, 30)

In [22]:
```
284807 - 275663
```

Out[22]: 9144

Exploring Class columns

In [23]:
```
ccfd['Class'].unique()
```

Out[23]: array([0, 1], dtype=int64)

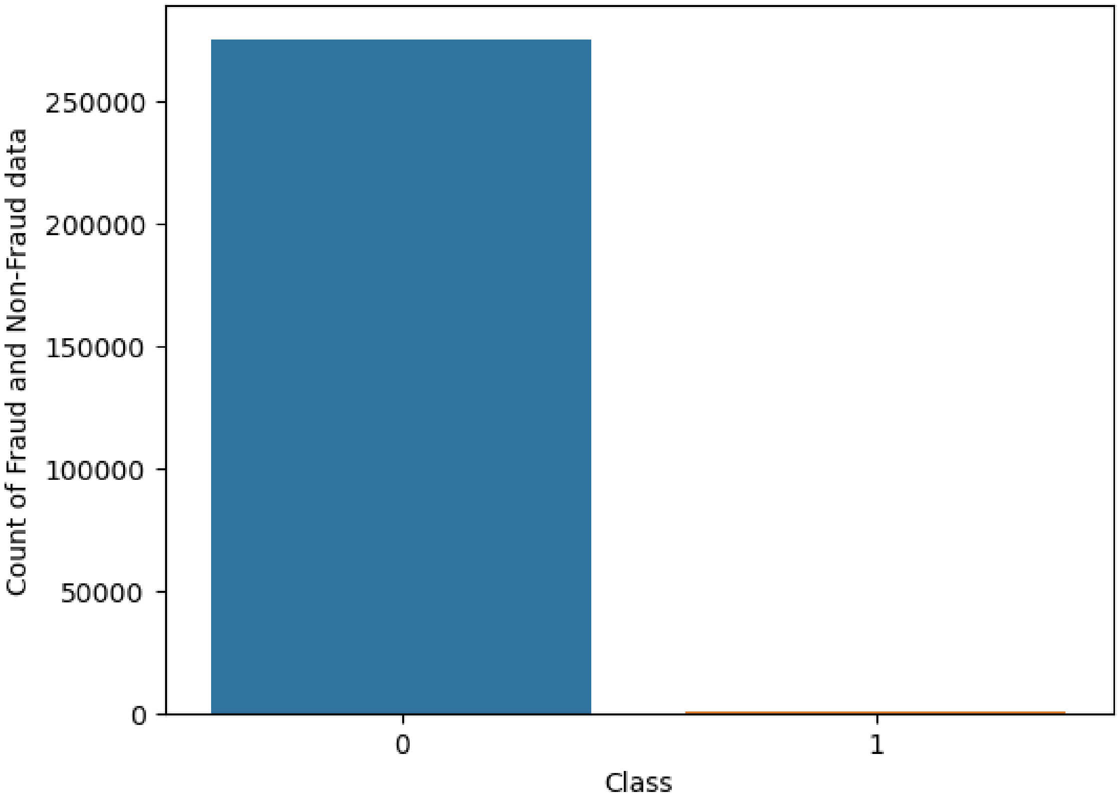
In [24]:
```
ccfd['Class'].nunique()
```

Out[24]: 2

In [25]:
```
ccfd['Class'].value_counts()
```

Out[25]:
```
0    275190
1       473
Name: Class, dtype: int64
```

In [26]:
```
#visualizing the distribution of 0 and 1 using seaborn countplot
sns.countplot(ccfd,x = ccfd['Class'])
plt.xlabel('Class')
plt.ylabel('Count of Fraud and Non-Fraud data')
plt.show()
```



From the above information, We can say that our data is high imbalanced, so need to apply oversampling and undersampling technique to train our model

Storing feature matrix in X and response (Target) in vector y

In [27]:
```
ccfd.head()
```

Out[27]:

	V1	V2	V3	V4	V5	V6	V7	V8	
0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.36
1	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.25
2	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.51
3	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.38
4	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.81

In [28]:
```
X = ccfd.drop('Class',axis = 1)
```

In [29]:
```
X
```

Out[29]:

	V1	V2	V3	V4	V5	V6	V7	V8
0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.09869
1	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.08510
2	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.24767
3	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.37743
4	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.27053
...	...	...	...	...	...	...	...	..
284802	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.30533
284803	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.29486
284804	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.70841
284805	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.67914
284806	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.41465

275663 rows × 29 columns

In [30]:
```
y = ccfd.Class
```

In [31]:
```
y
```

Out[31]:
```
0         0
1         0
2         0
3         0
4         0
         ..
284802    0
284803    0
284804    0
284805    0
284806    0
Name: Class, Length: 275663, dtype: int64
```

Splitting the dataset into the training set and test set

In [32]:
```
from sklearn.model_selection import train_test_split
```

In [33]:
```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size = 0.2,ra
```

In [34]:
```
X_train.shape
```

Out[34]:
```
(220530, 29)
```

Training into the Model

In [35]:
```
from sklearn.linear_model import LogisticRegression
```

In [36]:
```
LR = LogisticRegression()
```

In [37]:
```
LR.fit(X_train,y_train)
```

Out[37]:
```
LogisticRegression()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

Evaluating the accuracy_score, precision_score

In [38]:
```
from sklearn.metrics import precision_score,recall_score,f1_score,accur
```

In [39]:
```
y_pred = LR.predict(X_test)
```

In [40]:
```
accuracy_score(y_test,y_pred)
```

Out[40]:
```
0.9992200678359603
```

In [41]:
```
precision_score(y_test,y_pred)
```
Out[41]: 0.8870967741935484

In [42]:
```
recall_score(y_test,y_pred)
```
Out[42]: 0.6043956043956044

Here, precision_score is very low so we have to perform the oversampling and undersampling technique

Handling Imbalanced dataset

In [43]:
```
#undersampling
#oversampling
```

Undersampling

In [44]:
```
fraud = ccfd[ccfd['Class'] == 1]
normal = ccfd[ccfd['Class'] == 0]
```

In [45]:
```
fraud.shape
```
Out[45]: (473, 30)

In [46]:
```
normal.shape
```
Out[46]: (275190, 30)

In [47]:
```
#selecting the 473 necessary samples to balance the class feature
equal_sample = normal.sample(n=473)
```

In [48]:
```
equal_sample.shape
```
Out[48]: (473, 30)

In [49]:
```
new_ccfd = pd.concat([equal_sample,fraud],ignore_index = True)
```

In [50]:
```
new_ccfd['Class'].value_counts()
```
Out[50]:
```
0    473
1    473
Name: Class, dtype: int64
```

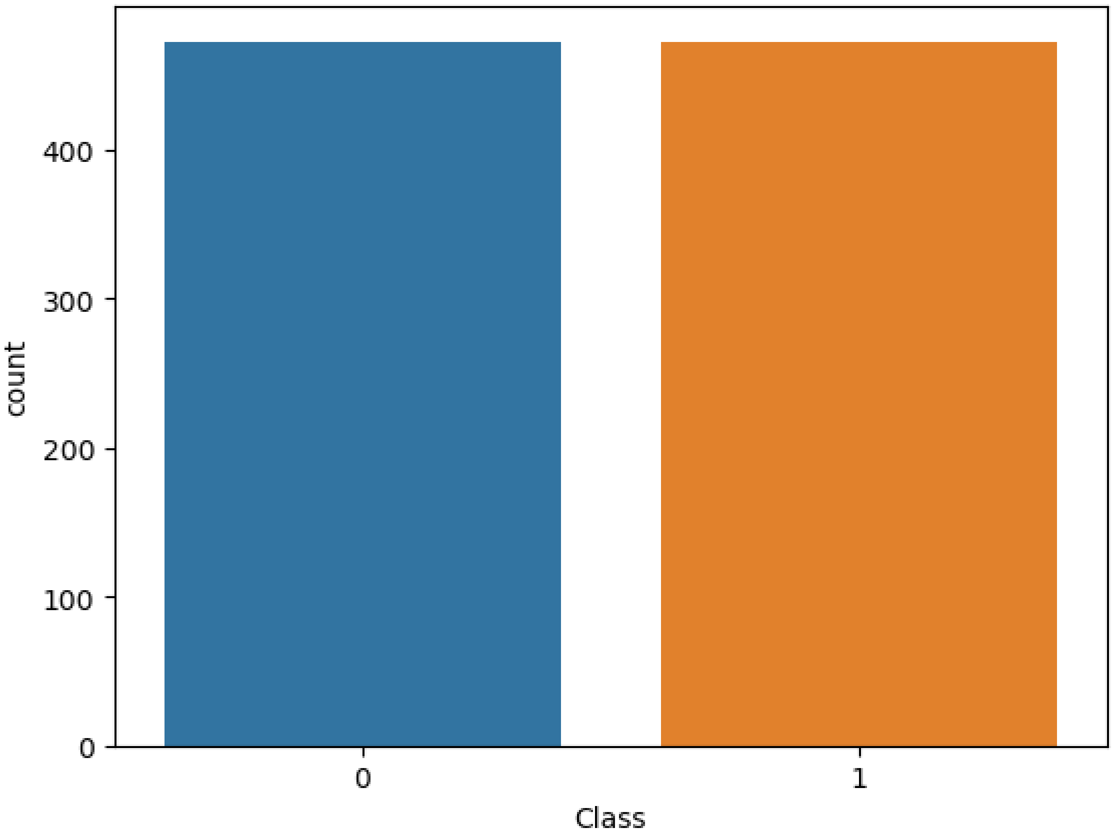
In [51]:
```
new_ccfd.head()
```

Out[51]:

	V1	V2	V3	V4	V5	V6	V7	V8	
0	-0.336788	1.163361	1.303065	0.057596	0.057744	-0.975195	0.735047	-0.093024	-0.44
1	-0.800695	0.799269	-0.744820	-1.097408	2.233199	3.195583	-0.096211	1.136893	-1.01
2	-0.641539	0.530215	1.518416	-0.893933	0.164667	0.391822	0.281905	0.086762	0.58
3	-0.118310	0.923913	-0.947681	-1.132053	1.470516	-1.236531	1.658472	-0.382232	-0.41
4	-0.783212	1.886366	1.434549	2.937871	-0.082150	-0.675020	0.894349	0.131387	-2.38

In [52]:
```
sns.countplot(x = new_ccfd['Class'],data=new_ccfd)
```

Out[52]: <Axes: xlabel='Class', ylabel='count'>



Now we equalized the Class feature

In [53]:
```
X = new_ccfd.drop('Class',axis = 1)
```

In [54]:
```
X
```

Out[54]:

	V1	V2	V3	V4	V5	V6	V7	V8	
0	-0.336788	1.163361	1.303065	0.057596	0.057744	-0.975195	0.735047	-0.093024	-0.
1	-0.800695	0.799269	-0.744820	-1.097408	2.233199	3.195583	-0.096211	1.136893	-1.
2	-0.641539	0.530215	1.518416	-0.893933	0.164667	0.391822	0.281905	0.086762	0.
3	-0.118310	0.923913	-0.947681	-1.132053	1.470516	-1.236531	1.658472	-0.382232	-0.
4	-0.783212	1.886366	1.434549	2.937871	-0.082150	-0.675020	0.894349	0.131387	-2.
...	...	...	...	...	...	...	...	...	
941	-1.927883	1.125653	-4.518331	1.749293	-1.566487	-2.010494	-0.882850	0.697211	-2.
942	1.378559	1.289381	-5.004247	1.411850	0.442581	-1.326536	-1.413170	0.248525	-1.
943	-0.676143	1.126366	-2.213700	0.468308	-1.120541	-0.003346	-2.234739	1.210158	-0.
944	-3.113832	0.585864	-5.399730	1.817092	-0.840618	-2.943548	-2.208002	1.058733	-1.
945	1.991976	0.158476	-2.583441	0.408670	1.151147	-0.096695	0.223050	-0.068384	0.

946 rows × 29 columns

In [55]:
```
y = new_ccfd.Class
```

In [56]:
```
y
```

Out[56]:
```
0      0
1      0
2      0
3      0
4      0
      ..
941    1
942    1
943    1
944    1
945    1
Name: Class, Length: 946, dtype: int64
```

Again Splitting the data for training and testing

In [57]:
```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size = 0.2,ra
```

In [58]:
```
X_train.shape
```

Out[58]:
```
(756, 29)
```

Logistis Regression

In [59]:
```
LR.fit(X_train,y_train)
```

Out[59]:
```
LogisticRegression()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [60]:
```
y_pred1 = LR.predict(X_test)
```

In [61]:
```
accuracy_score(y_test,y_pred1)
```

Out[61]:
```
0.9210526315789473
```

In [62]:
```
precision_score(y_test,y_pred1)
```

Out[62]:
```
0.9393939393939394
```

In [63]:
```
f1_score(y_test,y_pred1)
```

Out[63]:
```
0.9253731343283583
```

Decision Tree Classification

In [64]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [65]:
```
DTC = DecisionTreeClassifier()
```

In [66]:
```
DTC.fit(X_train,y_train)
```

Out[66]:
```
DecisionTreeClassifier()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [67]:
```
y_pred2 = DTC.predict(X_test)
```

Evaluating the precision_score, accuracy_score, f1_score

In [68]:
```
accuracy_score(y_test,y_pred2)
```

Out[68]:
```
0.8736842105263158
```

In [69]:
```
precision_score(y_test,y_pred2)
```
Out[69]: 0.8421052631578947

In [70]:
```
f1_score(y_test,y_pred2)
```
Out[70]: 0.8888888888888888

RandomForest Classifier

In [71]:
```
from sklearn.ensemble import RandomForestClassifier
```
In [72]:
```
RFC = RandomForestClassifier()
```
In [73]:
```
RFC.fit(X_train,y_train)
```
Out[73]: RandomForestClassifier()

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [74]:
```
y_pred3 = RFC.predict(X_test)
```

Evaluating the precision_Score, accuracy_score,f1_score

In [75]:
```
accuracy_score(y_test,y_pred3)
```
Out[75]: 0.9368421052631579

In [76]:
```
precision_score(y_test,y_pred3)
```
Out[76]: 0.9591836734693877

In [77]:
```
f1_score(y_test,y_pred3)
```
Out[77]: 0.9400000000000001

LightBGM

In [78]:
```
pip install lightgbm
```
```
Requirement already satisfied: lightgbm in c:\users\jnave\anaconda3\lib\si
te-packages (4.1.0)
Requirement already satisfied: numpy in c:\users\jnave\anaconda3\lib\site-
packages (from lightgbm) (1.24.3)
Requirement already satisfied: scipy in c:\users\jnave\anaconda3\lib\site-
packages (from lightgbm) (1.10.1)
Note: you may need to restart the kernel to use updated packages.
```

In [79]:
```
from lightgbm import LGBMClassifier
```

In [80]:
```
LGBM = LGBMClassifier()
```

In [81]:
```
LGBM.fit(X_train,y_train)
```

```
[LightGBM] [Info] Number of positive: 371, number of negative: 385
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of
testing was 0.000329 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 7317
[LightGBM] [Info] Number of data points in the train set: 756, number of u
sed features: 29
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.490741 -> initscore=-0.0
37041
[LightGBM] [Info] Start training from score -0.037041
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

Out[81]:
```
LGBMClassifier()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [82]:
```
y_pred4 = LGBM.predict(X_test)
```

Evaluating the precision_Score, accuracy_score,f1_score

In [83]:
```
accuracy_score(y_test,y_pred4)
```

Out[83]:
```
0.9315789473684211
```

In [84]:
```
precision_score(y_test,y_pred4)
```

Out[84]:
```
0.9494949494949495
```

In [85]:
```
f1_score(y_test,y_pred4)
```

Out[85]:
```
0.9353233830845771
```

Checking which model is performing better accuracy_score

In [86]:
```
stats = pd.DataFrame({'Model':['Logistic Regression','Decision Classifi
                      'Accuracy_score':[accuracy_score(y_test,y_pred1),ac
```

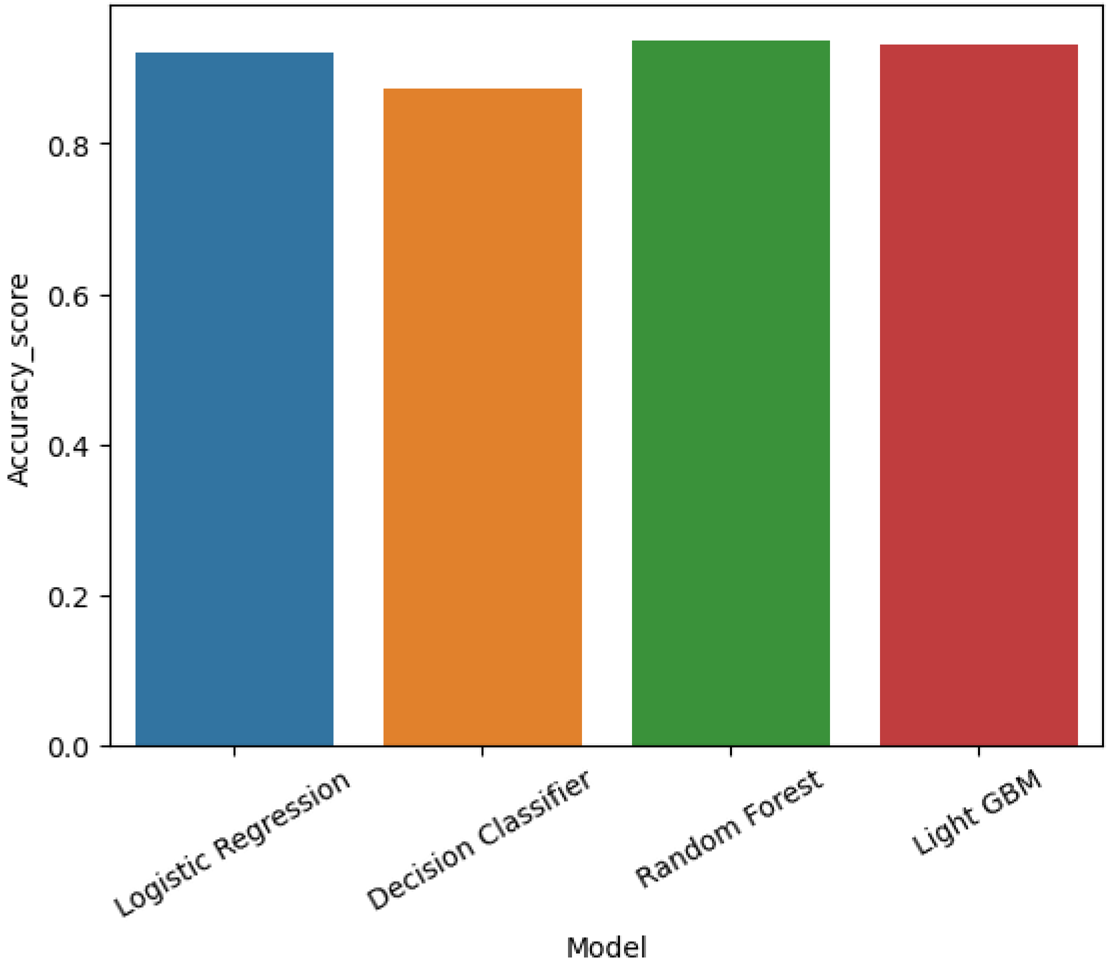
In [87]:
```
stats
```

Out[87]:

	Model	Accuracy_score
0	Logistic Regression	0.921053
1	Decision Classifier	0.873684
2	Random Forest	0.936842
3	Light GBM	0.931579

In [88]:
```

ax = sns.barplot(x = 'Model',y = 'Accuracy_score',data = stats)
plt.xticks(rotation=30)
plt.show()
```



As we are losting so much of feature information in undersampling, so move head to oversampling

In [ ]:
```

```

Oversampling

In [ ]:
```
pip install imbalanced-learn==0.10.1
```

In [89]:
```
pip install -U imbalanced-learn
```

```
Requirement already satisfied: imbalanced-learn in c:\users\jnave\anaconda
3\lib\site-packages (0.11.0)
Requirement already satisfied: numpy>=1.17.3 in c:\users\jnave\anaconda3\l
ib\site-packages (from imbalanced-learn) (1.24.3)
Requirement already satisfied: scipy>=1.5.0 in c:\users\jnave\anaconda3\li
b\site-packages (from imbalanced-learn) (1.10.1)
Requirement already satisfied: scikit-learn>=1.0.2 in c:\users\jnave\anaco
nda3\lib\site-packages (from imbalanced-learn) (1.2.2)
Requirement already satisfied: joblib>=1.1.1 in c:\users\jnave\anaconda3\l
ib\site-packages (from imbalanced-learn) (1.2.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\jnave\anac
onda3\lib\site-packages (from imbalanced-learn) (2.2.0)
Note: you may need to restart the kernel to use updated packages.
```

In [90]:
```
from imblearn.over_sampling import SMOTE
```

In [119]:
```
x2 = ccfd.drop('Class',axis=1)
```

In [122]:
```
x2.head()
```

Out[122]:

	V1	V2	V3	V4	V5	V6	V7	V8	
0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.36
1	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.25
2	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.51
3	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.38
4	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.81

In [120]:
```
y2 = ccfd.Class
```

In [121]:
```
y2
```

Out[121]:
```
0         0
1         0
2         0
3         0
4         0
         ..
284802    0
284803    0
284804    0
284805    0
284806    0
Name: Class, Length: 275663, dtype: int64
```

In [123]:
```
X_res,y_res = SMOTE().fit_resample(x2,y2)
```

In [124]:
```
y_res.value_counts()
```

Out[124]:
```
0    275190
1    275190
Name: Class, dtype: int64
```

Again split the training and testing data

In [125]:
```
X_train,X_test,y_train,y_test = train_test_split(X_res,y_res,test_size
```

Train the Model

Logistic Regression

In [130]:
```
#already imported
LR.fit(X_train,y_train)
```

Out[130]:
```
LogisticRegression()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

Evaluating accuracy_score,precision_score,f1_score

In [131]:
```
accuracy_score(y_test,LR.predict(X_test))
```

Out[131]:
```
0.9448926196446091
```

In [132]:
```
precision_score(y_test,LR.predict(X_test))
```

Out[132]:
```
0.9733975661191402
```

In [133]:
```
f1_score(y_test,LR.predict(X_test))
```

Out[133]:
```
0.9431436873183991
```

Decision Tree Classifier

In [134]:
```
DTC.fit(X_train,y_train)
```

Out[134]:
```
DecisionTreeClassifier()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

Evaluating accuracy_Score,precision_Score,f1_score

In [137]:
```
accuracy_score(y_test,DTC.predict(X_test))
```
Out[137]: 0.998128565718231

In [136]:
```
precision_score(y_test,DTC.predict(X_test))
```
Out[136]: 0.9974400406688575

In [135]:
```
f1_score(y_test,DTC.predict(X_test))
```
Out[135]: 0.9981286677204266

Random Forest Classifier

In [138]:
```
RFC.fit(X_train,y_train)
```
Out[138]: RandomForestClassifier()

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

Evaluating accuracy_Score,precision_Score,f1_score

In [139]:
```
accuracy_score(y_test,RFC.predict(X_test))
```
Out[139]: 0.999918238308078

In [140]:
```
precision_score(y_test,RFC.predict(X_test))
```
Out[140]: 0.9998363993310551

In [141]:
```
f1_score(y_test,RFC.predict(X_test))
```
Out[141]: 0.9999181929736854

LightGBM

In [142]:
```
LGBM.fit(X_train,y_train)
```

```
[LightGBM] [Info] Number of positive: 220187, number of negative: 220117
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of
testing was 0.036061 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 7395
[LightGBM] [Info] Number of data points in the train set: 440304, number o
f used features: 29
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500079 -> initscore=0.00
0318
[LightGBM] [Info] Start training from score 0.000318
```

Out[142]: LGBMClassifier()

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

Evaluating accuracy_Score,precision_Score,f1_score

In [143]:
```
accuracy_score(y_test,LGBM.predict(X_test))
```

Out[143]: 0.9991369599186017

In [144]:
```
precision_score(y_test,LGBM.predict(X_test))
```

Out[144]: 0.9984386347131445

In [145]:
```
f1_score(y_test,LGBM.predict(X_test))
```

Out[145]: 0.9991370147979253

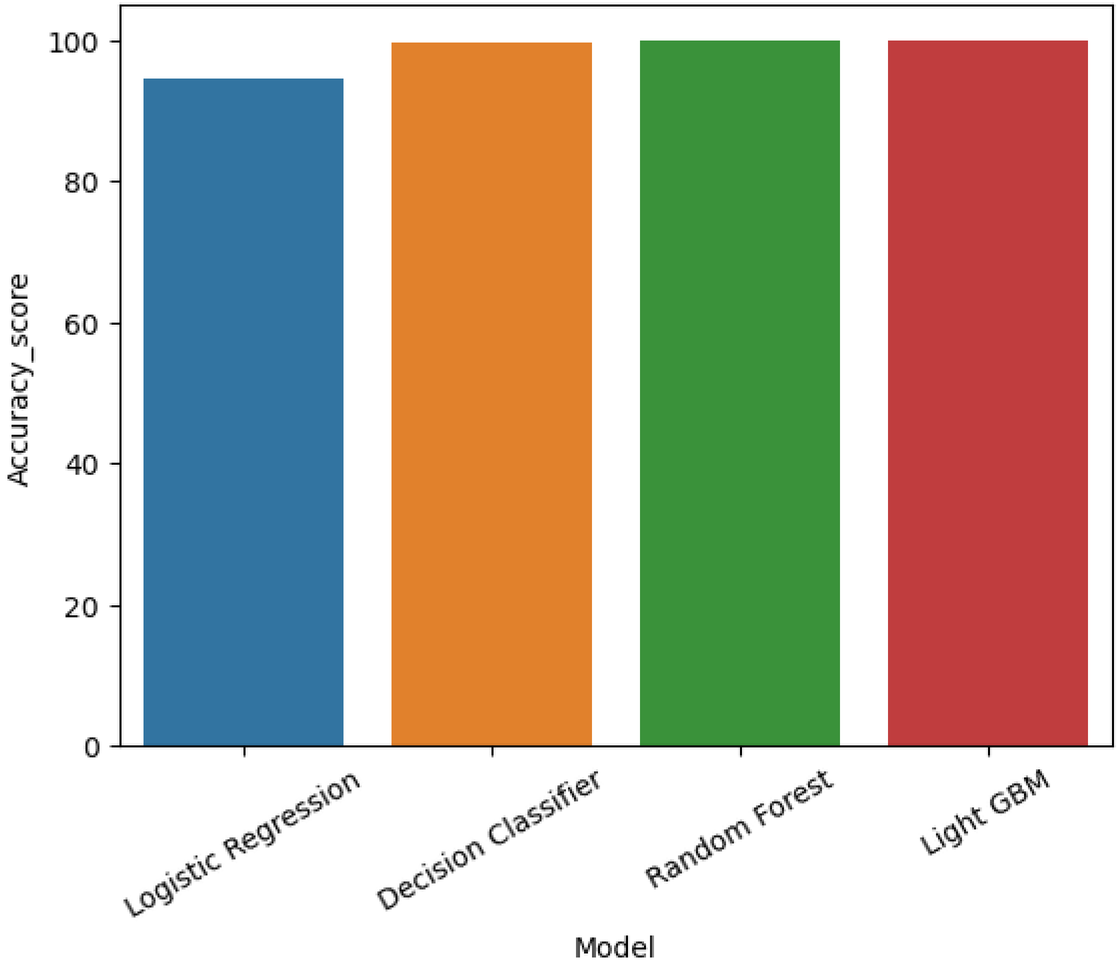
In [150]:
```
stats_oversampling = pd.DataFrame({'Model':['Logistic Regression','Deci
                                   'Accuracy_score':[accuracy_score(y_test,LR.predict(
```

In [151]:
```
stats_oversampling
```

Out[151]:

	Model	Accuracy_score
0	Logistic Regression	94.489262
1	Decision Classifier	99.812857
2	Random Forest	99.991824
3	Light GBM	99.913696

In [152]:
```
sns.barplot(x = 'Model',y = 'Accuracy_score',data = stats_oversampling)
plt.xticks(rotation=30)
plt.show()
```



Since Random Forest and Light Gradient Boosting Machine is performing better

In [153]:
```
import joblib
```

In [166]:
```
joblib.dump(RFC,"C:\\Users\\jnave\\OneDrive\\Documents\\IBM Applied Dat
```

Out[166]:
```
['C:\\Users\\jnave\\OneDrive\\Documents\\IBM Applied Data Science\\CCFD MO
DEL.txt']
```

In [157]:
```
model = joblib.load("C:\\Users\\jnave\\OneDrive\\Documents\\IBM Appliec
```

In [165]:
```
predicted = model.predict([[1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
if(predicted == 0):
    print("Normal Transaction")
else:
    print("Fraudlent Transaction")
```

```
Normal Transaction

C:\Users\jnave\anaconda3\Lib\site-packages\sklearn\base.py:439: UserWarnin
g: X does not have valid feature names, but RandomForestClassifier was fit
ted with feature names
  warnings.warn(
```

In [167]:
```
import pickle
```

In [172]:
```
pickle.dump(RFC,open("C:\\Users\\jnave\\OneDrive\\Documents\\IBM Applie
```

In [ ]:
```
```

In [ ]:
```
```