Final Project [BDM 2053 - Big Data Algorithms and Statistic 01]

Submitted by: Aadarsha Chapagain

Student Id: C0825975

Select the data set

For this project I have Selected the credit card dataset available at Kaggle in the following link.
Link:https://www.kaggle.com/code/janiobachmann/credit-fraud-dealing-with-imbalanced-datasets/data

In [1]:

```
import sys
!{sys.executable} -m pip install imblearn
```

```
Requirement already satisfied: imblearn in s:\anaconda\lib\site-packages (0.0)
Requirement already satisfied: imbalanced-learn in s:\anaconda\lib\site-packages (from imblearn) (0.9.0)
Requirement already satisfied: scipy>=1.1.0 in s:\anaconda\lib\site-packages (from imbalanced-learn->imblearn) (1.7.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in s:\anaconda\lib\site-packages (from imbalanced-learn->imblearn)
(2.2.0)
Requirement already satisfied: scikit-learn>=1.0.1 in s:\anaconda\lib\site-packages (from imbalanced-learn->imblearn) (1.
0.2)
Requirement already satisfied: numpy>=1.14.6 in s:\anaconda\lib\site-packages (from imbalanced-learn->imblearn) (1.20.3)
Requirement already satisfied: joblib>=0.11 in s:\anaconda\lib\site-packages (from imbalanced-learn->imblearn) (1.1.0)
```

In [2]:

```
import pandas as pd
import warnings
warnings.filterwarnings("ignore")
```

In [3]:

```
df =pd.read_csv('./creditcard.csv')
df.head()
```

Out[3]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0

5 rows × 31 columns

Here we can see the features v1, v2, v3 ... V28,Time,Amount,Class. Because of the confidentilaity issue the original features v1, v2, v3,...V28 are transformed with pca.Only feature which has not been transformed with PCA are Time, Amount and Class. Class is the classification of whether the transaction is fraud or not. Here 0 represents no fraud and 1 represents fraud.

In [4]:

```
# Check out the summary
df.describe()
print("Shape:",df.shape)
```

```
Shape: (284807, 31)
```

Cleaning Data and Exploratory Data Analysis

In [5]:

```
# Check for null and missing value
df.isnull().sum().max()
# There are no Null values
```

Out[5]:

```
0
```

In [6]:

```
df.columns
```

Out[6]:

```
Index(['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10',
       'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20',
       'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount',
       'Class'],
      dtype='object')
```

In [7]:

```
#  The data are heavily skewed
df['Class'].value_counts()
```

Out[7]:
```
0    284315
1       492
Name: Class, dtype: int64
```

In [8]:
```
print("NO Fraud:", round(df['Class'].value_counts()[0]/len(df) * 100,2), '%')
print("Fraud", round(df['Class'].value_counts()[1]/len(df) * 100,2), '%')
```

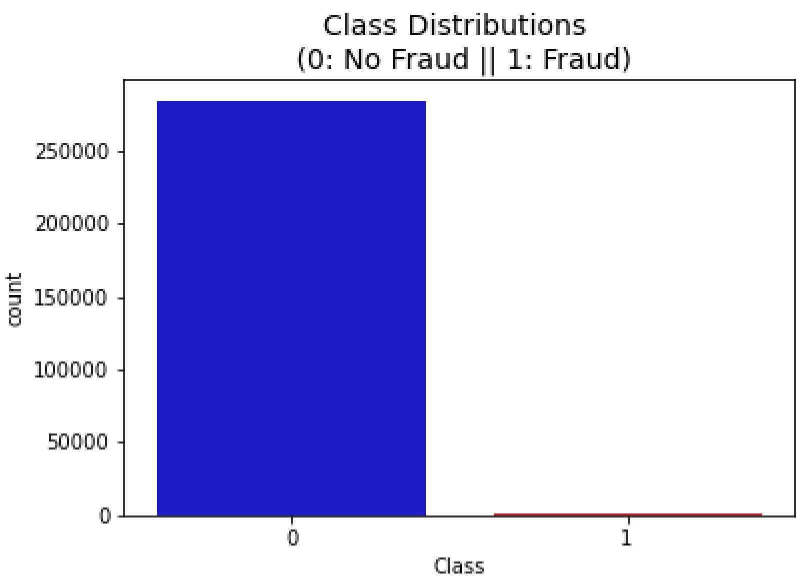
```
NO Fraud: 99.83 %
Fraud 0.17 %
```

We can see that our data set is imblanced and training on such datasets might give us faulty result. Since the most of our datasets represent no fraud we will have model that will overfit and predict the fraud data as no fraud

In [9]:
```
import matplotlib.pyplot as plt
import seaborn as sns

colors = ["#0101DF", "#DF0101"]
sns.countplot('Class', data=df, palette=colors)
plt.title('Class Distributions \n (0: No Fraud || 1: Fraud)', fontsize=14)
```

Out[9]:
```
Text(0.5, 1.0, 'Class Distributions \n (0: No Fraud || 1: Fraud)')
```



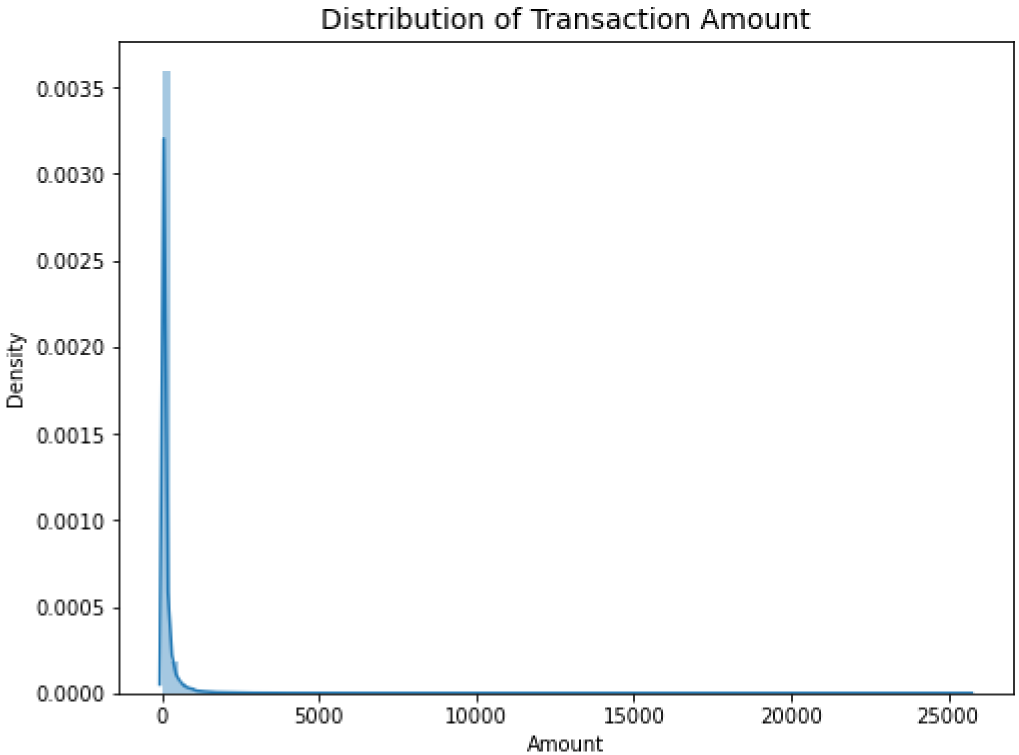
In our dataset, it is seen that most of our transaction around 99% of the transactions are not fraud and rest of the transaction are fraud.

In [10]:
```
df[['Amount','Time']].describe()
```

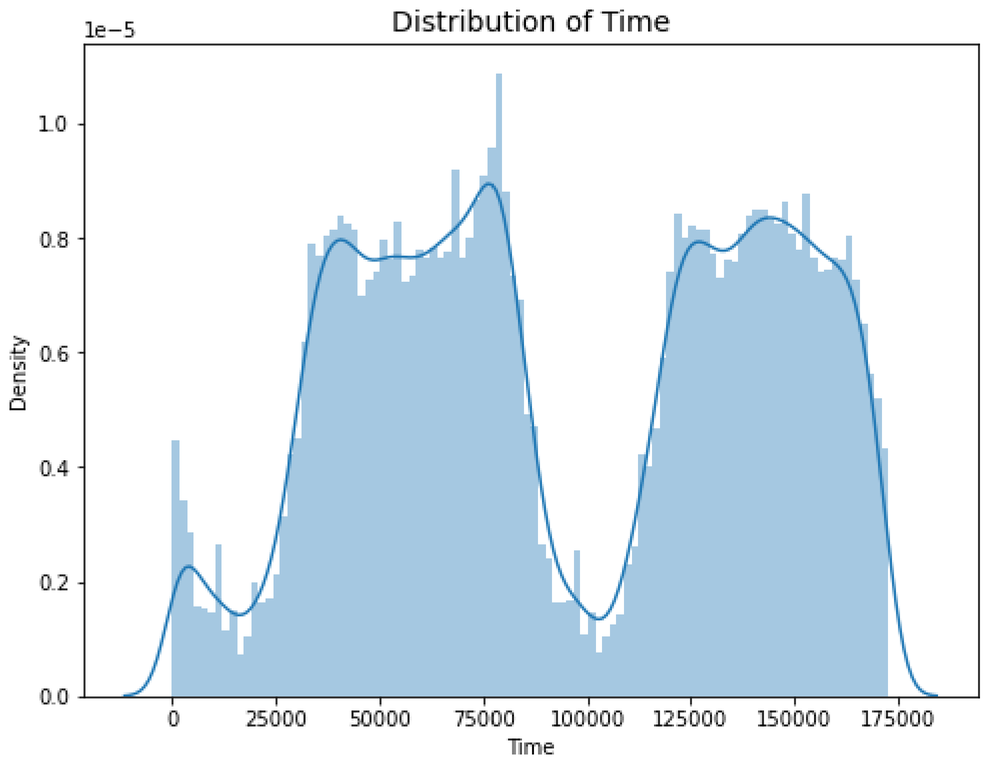
Out[10]:

	Amount	Time
count	284807.000000	284807.000000
mean	88.349619	94813.859575
std	250.120109	47488.145955
min	0.000000	0.000000
25%	5.600000	54201.500000
50%	22.000000	84692.000000
75%	77.165000	139320.500000
max	25691.160000	172792.000000

In [11]:
```
# Lets see the distribution of Transaction
plt.figure(figsize=(8,6))
plt.title('Distribution of Transaction Amount', fontsize=14)
sns.distplot(df['Amount'], bins=100)
plt.show()
```

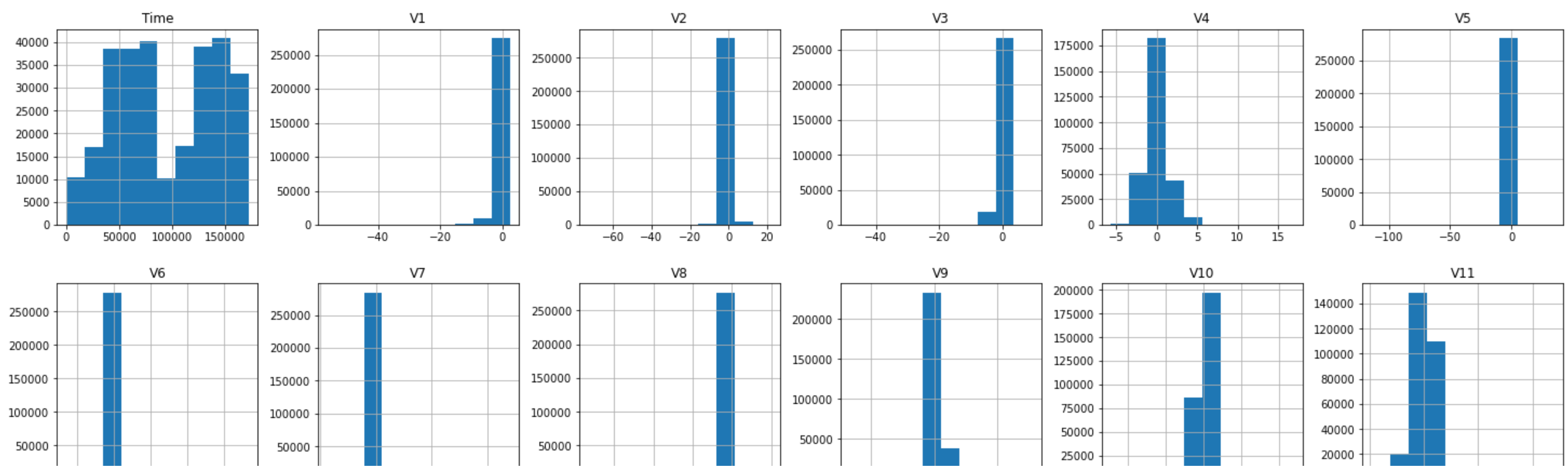


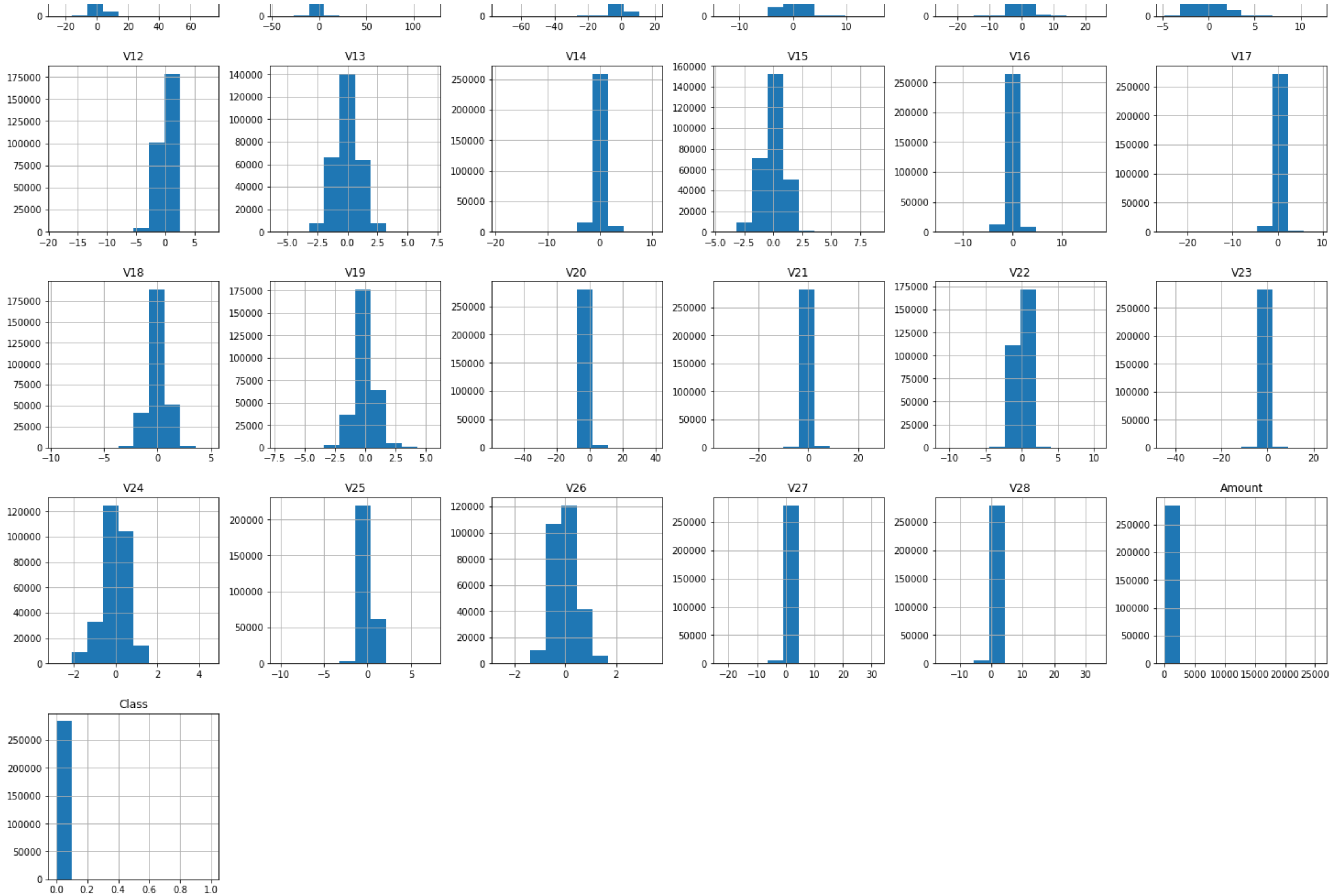
In [12]:
```
# Distribution of Time
plt.figure(figsize=(8,6))
plt.title('Distribution of Time', fontsize=14)
sns.distplot(df['Time'], bins=100)
plt.show()
```



In [13]:

```
# Anamoly detection
df.hist(figsize = (25,25))
plt.show()
```





Scaling

It is good idea to scale the features so that less significant features do not end up dominating the important features.Since all columns except amount and time are transformed we will be scaling these two features only.

In [14]:

```python
# Lets scale the amount feature by Mean-Max scaling
# RobustScaler is less prone to outliers.
from sklearn.preprocessing import StandardScaler, RobustScaler
rob_scaler = RobustScaler()

df['scaled_amount'] = rob_scaler.fit_transform(df['Amount'].values.reshape(-1,1))
df['scaled_time'] = rob_scaler.fit_transform(df['Time'].values.reshape(-1,1))

df.head()
```

Out[14]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V23	V24	V25	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.110474	0.066928	0.128539	-0
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	0.101288	-0.339846	0.167170	0
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.909412	-0.689281	-0.327642	-0
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.190321	-1.175575	0.647376	-0
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.137458	0.141267	-0.206010	0

5 rows × 33 columns

In [15]:

```python
# Seperating the target and predictor variables.
X = df.drop(['Time','Class','Amount'],axis=1)
y = df['Class']
X
```

Out[15]:

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	...	V21	V22	
0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	0.090794	...	-0.018307	0.277838	-
1	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.166974	...	-0.225775	-0.638672	
2	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.207643	...	0.247998	0.771679	
3	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	-0.054952	...	-0.108300	0.005274	-
4	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.753074	...	-0.009431	0.798278	-
...	...	...	...	...	...	...	...	...	...	...	...	...	...	

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	...	V21	V22
284802	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.305334	1.914428	4.356170	...	0.213454	0.111864
284803	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.294869	0.584800	-0.975926	...	0.214205	0.924384
284804	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.708417	0.432454	-0.484782	...	0.232045	0.578229
284805	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.679145	0.392087	-0.399126	...	0.265245	0.800049
284806	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.414650	0.486180	-0.915427	...	0.261057	0.643078

284807 rows × 30 columns

Training, Testing and splitting.

Lets split the data into training and testing, here we have splitted the data into 70% for the training and 30% for the testing.

In [16]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, shuffle=True, random_state=101)
print("X_train: ",X_train.shape)
print("y_train:- ",y_train.shape)
print("X_test: ",X_test.shape)
print("y_test: ",y_test.shape)
type(y_test)
```

```
X_train:  (199364, 30)
y_train:-  (199364,)
X_test:  (85443, 30)
y_test:  (85443,)
```

Out[16]:

```
pandas.core.series.Series
```

Different Classification Algorithm

In [17]:

```
#  lets use different classifier to train our model
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics
```

```
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsClassifier
```

Training And Predicting

We use Scikit Learn fit method to train the model and predict method the predict using that model.

In [18]:

```
# Logistic Regression
from sklearn.metrics import confusion_matrix
lgreg = LogisticRegression()
lgreg.fit(X_train, y_train)
y_pred = lgreg.predict(X_test)
print("Values Count of y_test:\n",y_test.value_counts())
print("---------------")
print("-------------------------Confusion Matrix-------------------------------")
print(confusion_matrix(y_test, y_pred))

print("-------------------------------------------------------------------------------------")
print("Classification report for Logical Regression on test data:\n",metrics.classification_report(y_test, y_pred))
print("Accuracy score:",metrics.accuracy_score(y_test, y_pred))
print('F1 : {0:0.5f}'.format(metrics.f1_score(y_test , y_pred)))
print("-------------------------------------------------------------------------------------")
```

```
Values Count of y_test:
 0    85299
1      144
Name: Class, dtype: int64
---------------
-------------------------Confusion Matrix-------------------------------
[[85287    12]
 [   56    88]]
-------------------------------------------------------------------------------------
Classification report for Logical Regression on test data:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00     85299
           1       0.88      0.61      0.72       144

    accuracy                           1.00     85443
   macro avg       0.94      0.81      0.86     85443
weighted avg       1.00      1.00      1.00     85443

Accuracy score: 0.999204147794436
```

```
F1 : 0.72131
--------------------------------------------------------------------------------------
```

Here we can see the algorithm is doing pretty well in case of non fradulent data that is predicting non-fradulent transaction as not fraud but it is relatively doing poor in case of predicting fraudelent data. Among 144 transaction only 88 were identified as fraud that is true positives.

In [19]:

```
# Decision Tree Classifier
Dt_model = DecisionTreeClassifier()
Dt_model.fit(X_train, y_train)
y_pred = Dt_model.predict(X_test)
print("Values Count of y_test:\n",y_test.value_counts())
print("---------------")
print("-------------------------Confusion Matrix-------------------------------")
print(confusion_matrix(y_test, y_pred))
print("--------------------------------------------------------------------------------------")
print("Classification report for Dt on test data:\n",metrics.classification_report(y_test, y_pred))
print("Accuracy score:",metrics.accuracy_score(y_test, y_pred))
print('F1 : {0:0.5f}'.format(metrics.f1_score(y_test , y_pred)))
print("--------------------------------------------------------------------------------------")
```

```
Values Count of y_test:
 0    85299
1      144
Name: Class, dtype: int64
---------------
-------------------------Confusion Matrix-------------------------------
[[85260    39]
 [   34   110]]
--------------------------------------------------------------------------------------
Classification report for Dt on test data:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00     85299
           1       0.74      0.76      0.75       144

    accuracy                           1.00     85443
   macro avg       0.87      0.88      0.88     85443
weighted avg       1.00      1.00      1.00     85443

Accuracy score: 0.9991456292499094
F1 : 0.75085
--------------------------------------------------------------------------------------
```

As compared to Logistic regression, decision tree classifier is performing well in terms of true positive, that is predicting fraudenlt transaction as fraud. 109 out of 144 fradulent transaction are identified as fraud.

Undersampling

The model is performing well but Since we have only few fradulent transaction lets try decreasing the majority with Random Under Sampling and check the performance of the model.

In [ ]:

```

```

In [20]:

```
from imblearn.over_sampling import SMOTE, ADASYN

from collections import Counter
from imblearn.under_sampling import RandomUnderSampler
```

In [21]:

```
print('Original dataset shape %s' % Counter(y_train))

# Undersampling only on train
rus = RandomUnderSampler(random_state=42)
X_train_rus, y_train_rus = rus.fit_resample(X_train, y_train)
print('Resampled dataset shape %s' % Counter(y_train_rus))
```

```
Original dataset shape Counter({0: 199016, 1: 348})
Resampled dataset shape Counter({0: 348, 1: 348})
```

In [22]:

```
# Undersampling with Logistic Regression
logreg = LogisticRegression()
logreg.fit(X_train_rus, y_train_rus)
y_pred_rus = logreg.predict(X_test)
print("Values Count of y_train_rus:\n",y_train_rus.value_counts())
print("Values Count of y_test:\n",y_test.value_counts())
print("---------------")
print("-------------------------Confusion Matrix-------------------------------")
print(confusion_matrix(y_test, y_pred_rus))
print("-----------------------------------------------------------------------------------")
print("Classification report for Logical regression on test data:\n",metrics.classification_report(y_test, y_pred_rus))
print("Accuracy score:",metrics.accuracy_score(y_test, y_pred))
print('F1 : {0:0.5f}'.format(metrics.f1_score(y_test , y_pred_rus)))
print("-----------------------------------------------------------------------------------")
```

```
Values Count of y_train_rus:
 0    348
1    348
Name: Class, dtype: int64
Values Count of y_test:
 0    85299
1      144
Name: Class, dtype: int64
---------------
-------------------------Confusion Matrix------------------------------
[[83193  2106]
 [   12   132]]
-------------------------------------------------------------------------------------
Classification report for Logical regression on test data:
               precision    recall  f1-score   support

           0       1.00      0.98      0.99     85299
           1       0.06      0.92      0.11       144

    accuracy                           0.98     85443
   macro avg       0.53      0.95      0.55     85443
weighted avg       1.00      0.98      0.99     85443

Accuracy score: 0.9991456292499094
F1 : 0.11083
-------------------------------------------------------------------------------------
```

Here we can see that the number of true positive (that is fraudelent transaction identified as fraud) has increased but the false postive (Non Fradulent Transaction identified as Fraud has also increased)

lets try increasing the minority with Random Over Sampling and check the performance of the model.

In [23]:

```
from imblearn.over_sampling import RandomOverSampler
print('Original dataset shape %s' % Counter(y_train))

ros = RandomOverSampler(random_state=42)
X_train_ros, y_train_ros = ros.fit_resample(X_train, y_train)

print('Resampled dataset shape %s' % Counter(y_train_ros))
```

```
Original dataset shape Counter({0: 199016, 1: 348})
Resampled dataset shape Counter({0: 199016, 1: 199016})
```

we can see in resampled dataset that the number of fradulent transaction has been randomly increased to 199016

Random Over Sampling with logistic Regression

In [24]:

```
# Oversampling with Logistic Regression
logreg = LogisticRegression()
logreg.fit(X_train_ros, y_train_ros)

y_pred_ros = logreg.predict(X_test)
print("Values Count of y_test:\n",y_test.value_counts())
print("---------------")
print("------------------------Confusion Matrix-------------------------------")
print(confusion_matrix(y_test, y_pred_ros))
print("-------------------------------------------------------------------------------------")
print("Classification report for Logical regression with oversampling:\n",metrics.classification_report(y_test, y_pred_ro
print("Accuracy score:",metrics.accuracy_score(y_test, y_pred_ros))
print("-------------------------------------------------------------------------------------")
print('F1 : {0:0.5f}'.format(metrics.f1_score(y_test , y_pred_ros)))
```

```
Values Count of y_test:
 0    85299
1      144
Name: Class, dtype: int64
---------------
-------------------------Confusion Matrix-------------------------------
[[83443  1856]
 [   14   130]]
-------------------------------------------------------------------------------------
Classification report for Logical regression with oversampling:
               precision    recall  f1-score   support

           0       1.00      0.98      0.99     85299
           1       0.07      0.90      0.12       144

    accuracy                           0.98     85443
   macro avg       0.53      0.94      0.56     85443
weighted avg       1.00      0.98      0.99     85443

Accuracy score: 0.9781140643469916
-------------------------------------------------------------------------------------
F1 : 0.12207
```

We can see that the number of True Positive(that is fradulent transaction identified as fraud is 130 which is better than using original dataset) has increase but the false positive has increased as well.

Undersampling with Random Forest Classifier

In [25]:

```
# Undersampling with Random Forest Classifier
from sklearn.ensemble import RandomForestClassifier

RFC_model = RandomForestClassifier()
RFC_model.fit(X_train_rus, y_train_rus)
y_pred_rus = RFC_model.predict(X_test)
print("Values Count of y_test:\n",y_test.value_counts())
print("---------------")
print("-------------------------Confusion Matrix-------------------------------")
print(confusion_matrix(y_test, y_pred_rus))
print("----------------------------------------------------------------------------------------")
print("Classification report for Logical regression on test data:\n",metrics.classification_report(y_test, y_pred_rus))
print("Accuracy score:",metrics.accuracy_score(y_test, y_pred))
print('F1 : {0:0.5f}'.format(metrics.f1_score(y_test , y_pred_rus)))
print("----------------------------------------------------------------------------------------")
```

```
Values Count of y_test:
 0    85299
1      144
Name: Class, dtype: int64
---------------
-------------------------Confusion Matrix-------------------------------
[[82635  2664]
 [   16   128]]
----------------------------------------------------------------------------------------
Classification report for Logical regression on test data:
               precision    recall  f1-score   support

           0       1.00      0.97      0.98     85299
           1       0.05      0.89      0.09       144

    accuracy                           0.97     85443
   macro avg       0.52      0.93      0.54     85443
weighted avg       1.00      0.97      0.98     85443

Accuracy score: 0.9991456292499094
F1 : 0.08719
----------------------------------------------------------------------------------------
```

Using RandomForest Classifier also we get the same kind of result as logistic regression for undersampling majority

Oversampling with Random Forest Classifier

In [26]:
```
# Oversampling with Random forest Classifier
RFC_model = RandomForestClassifier()
RFC_model.fit(X_train_ros, y_train_ros)

y_pred_ros = RFC_model.predict(X_test)
print("Values Count of y_test:\n",y_test.value_counts())
print("-------------------------Confusion Matrix-------------------------------")
print(confusion_matrix(y_test, y_pred_rus))
print("-------------------------------------------------------------------------------------")
print("Classification report for Random forest Classifier with oversampling:\n",metrics.classification_report(y_test, y_p
print("Accuracy score:",metrics.accuracy_score(y_test, y_pred_ros))
print("-------------------------------------------------------------------------------------")
print('F1 : {0:0.5f}'.format(metrics.f1_score(y_test , y_pred_ros)))
```

```
Values Count of y_test:
 0    85299
1      144
Name: Class, dtype: int64
-------------------------Confusion Matrix-------------------------------
[[82635  2664]
 [   16   128]]
-------------------------------------------------------------------------------------
Classification report for Random forest Classifier with oversampling:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00     85299
           1       0.95      0.78      0.85       144

    accuracy                           1.00     85443
   macro avg       0.97      0.89      0.93     85443
weighted avg       1.00      1.00      1.00     85443

Accuracy score: 0.9995552590615966
-------------------------------------------------------------------------------------
F1 : 0.85496
```

Conclusion

Conclusion : For this sceanario model for credit card Fraud Transaction Analysis I reccommend Using Decision Tree Classifier model since it has accuracy of 0.99 and F1 score of 0.72. Further more We can use Area Under of Curve (AUC) to compare the models.

Accuracy score: 0.9990754069964772 F1 : 0.72664

References:

https://www.kaggle.com/code/dktalaicha/credit-card-fraud-detection-using-smote-adasyn
https://www.kaggle.com/code/janiobachmann/credit-fraud-dealing-with-imbalanced-datasets/notebook
https://www.kaggle.com/code/vitorgamalemos/credit-card-fraud-analysis/notebook