Importing the required Libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import datetime as dt
```

Importing the Data.

In [2]:
```
train_df = pd.read_csv('fraudTrain.csv')
test_df = pd.read_csv('fraudTest.csv')
```

In [3]:
```
train_df.head()
```

Out[3]:

	Unnamed: 0	trans_date_trans_time	cc_num	merchant	category	amt	
0	0	2019-01-01 00:00:18	2703186189652095	fraud_Rippin, Kub and Mann	misc_net	4.97	Je
1	1	2019-01-01 00:00:44	630423337322	fraud_Heller, Gutmann and Zieme	grocery_pos	107.23	Stej
2	2	2019-01-01 00:00:51	38859492057661	fraud_Lind-Buckridge	entertainment	220.11	E
3	3	2019-01-01 00:01:16	3534093764340240	fraud_Kutch, Hermiston and Farrell	gas_transport	45.00	J
4	4	2019-01-01 00:03:06	375534208663984	fraud_Keeling-Crist	misc_pos	41.96	

5 rows × 23 columns

In [4]:
```
test_df.head()
```

Out[4]:

	Unnamed: 0	trans_date_trans_time	cc_num	merchant	category	amt
0	0	2020-06-21 12:14:25	2291163933867244	fraud_Kirlin and Sons	personal_care	2.86
1	1	2020-06-21 12:14:33	3573030041201292	fraud_Sporer-Keebler	personal_care	29.84
2	2	2020-06-21 12:14:53	3598215285024754	fraud_Swaniawski, Nitzsche and Welch	health_fitness	41.28
3	3	2020-06-21 12:15:15	3591919803438423	fraud_Haley Group	misc_pos	60.05
4	4	2020-06-21 12:15:17	3526826139003947	fraud_Johnston-Casper	travel	3.19

5 rows × 22 columns

In [5]:
```
train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1296675 entries, 0 to 1296674
Data columns (total 23 columns):
 #   Column                 Non-Null Count    Dtype
---  ------                 --------------    -----
 0   Unnamed: 0             1296675 non-null  int64
 1   trans_date_trans_time  1296675 non-null  object
 2   cc_num                 1296675 non-null  int64
 3   merchant               1296675 non-null  object
 4   category               1296675 non-null  object
 5   amt                    1296675 non-null  float64
 6   first                  1296675 non-null  object
 7   last                   1296675 non-null  object
 8   gender                 1296675 non-null  object
 9   street                 1296675 non-null  object
 10  city                   1296675 non-null  object
 11  state                  1296675 non-null  object
 12  zip                    1296675 non-null  int64
 13  lat                    1296675 non-null  float64
 14  long                   1296675 non-null  float64
 15  city_pop               1296675 non-null  int64
 16  job                    1296675 non-null  object
 17  dob                    1296675 non-null  object
 18  trans_num              1296675 non-null  object
 19  unix_time              1296675 non-null  int64
 20  merch_lat              1296675 non-null  float64
 21  merch_long             1296675 non-null  float64
 22  is_fraud               1296675 non-null  int64
```

In [5]:
```
train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1296675 entries, 0 to 1296674
Data columns (total 23 columns):
 #   Column                 Non-Null Count    Dtype
---  ------                 --------------    -----
 0   Unnamed: 0             1296675 non-null  int64
 1   trans_date_trans_time  1296675 non-null  object
 2   cc_num                 1296675 non-null  int64
 3   merchant               1296675 non-null  object
 4   category               1296675 non-null  object
 5   amt                    1296675 non-null  float64
 6   first                  1296675 non-null  object
 7   last                   1296675 non-null  object
 8   gender                 1296675 non-null  object
 9   street                 1296675 non-null  object
 10  city                   1296675 non-null  object
 11  state                  1296675 non-null  object
 12  zip                    1296675 non-null  int64
 13  lat                    1296675 non-null  float64
 14  long                   1296675 non-null  float64
 15  city_pop               1296675 non-null  int64
 16  job                    1296675 non-null  object
 17  dob                    1296675 non-null  object
 18  trans_num              1296675 non-null  object
 19  unix_time              1296675 non-null  int64
 20  merch_lat              1296675 non-null  float64
 21  merch_long             1296675 non-null  float64
 22  is_fraud               1296675 non-null  int64
dtypes: float64(5), int64(6), object(12)
memory usage: 227.5+ MB
```

In [6]:
```
test_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 555719 entries, 0 to 555718
Data columns (total 23 columns):
 #   Column                 Non-Null Count   Dtype
---  ------                 --------------   -----
 0   Unnamed: 0             555719 non-null  int64
 1   trans_date_trans_time  555719 non-null  object
 2   cc_num                 555719 non-null  int64
 3   merchant               555719 non-null  object
 4   category               555719 non-null  object
 5   amt                    555719 non-null  float64
 6   first                  555719 non-null  object
 7   last                   555719 non-null  object
 8   gender                 555719 non-null  object
 9   street                 555719 non-null  object
 10  city                   555719 non-null  object
 11  state                  555719 non-null  object
 12  zip                    555719 non-null  int64
 13  lat                    555719 non-null  float64
 14  long                   555719 non-null  float64
 15  city_pop               555719 non-null  int64
 16  job                    555719 non-null  object
 17  dob                    555719 non-null  object
 18  trans_num              555719 non-null  object
 19  unix_time              555719 non-null  int64
 20  merch_lat              555719 non-null  float64
 21  merch_long             555719 non-null  float64
 22  is_fraud               555719 non-null  int64
dtypes: float64(5), int64(6), object(12)
memory usage: 97.7+ MB
```

In [9]:
```
test_data.info()
```

In [7]:
```
## We have established that both train and test data have same columns wi
```

Conducting EDA

In [8]:
```
train_data = train_df
test_data = test_df
```

In [9]:
```
test_data.info()
```

```
dtypes: float64(5), int64(6), object(12)
memory usage: 555719+ MB
RangeIndex: 555719 entries, 0 to 555718
Data columns (total 23 columns):
 #   Column                 Non-Null Count   Dtype
---  ------                 --------------   -----
 0   Unnamed: 0             555719 non-null  int64
 1   trans_date_trans_time  555719 non-null  object
 2   cc_num                 555719 non-null  int64
 3   merchant               555719 non-null  object
 4   category               555719 non-null  object
 5   amt                    555719 non-null  float64
 6   first                  555719 non-null  object
 7   last                   555719 non-null  object
 8   gender                 555719 non-null  object
 9   street                 555719 non-null  object
 10  city                   555719 non-null  object
 11  state                  555719 non-null  object
 12  zip                    555719 non-null  int64
 13  lat                    555719 non-null  float64
 14  long                   555719 non-null  float64
 15  city_pop               555719 non-null  int64
 16  job                    555719 non-null  object
 17  dob                    555719 non-null  object
 18  trans_num              555719 non-null  object
 19  unix_time              555719 non-null  int64
 20  merch_lat              555719 non-null  float64
 21  merch_long             555719 non-null  float64
 22  is_fraud               555719 non-null  int64
dtypes: float64(5), int64(6), object(12)
memory usage: 97.5+ MB
```

In [7]:
```
## WE have established that both train and test data have same columns wi
```

Conducting EDA

In [8]:
```
train_data = train_df
test_data = test_df
```

In [10]:
```
train_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1296675 entries, 0 to 1296674
Data columns (total 23 columns):
 #   Column                 Non-Null Count    Dtype
---  ------                 --------------    -----
 0   Unnamed: 0             1296675 non-null  int64
 1   trans_date_trans_time  1296675 non-null  object
 2   cc_num                 1296675 non-null  int64
 3   merchant               1296675 non-null  object
 4   category               1296675 non-null  object
 5   amt                    1296675 non-null  float64
 6   first                  1296675 non-null  object
 7   last                   1296675 non-null  object
 8   gender                 1296675 non-null  object
 9   street                 1296675 non-null  object
 10  city                   1296675 non-null  object
 11  state                  1296675 non-null  object
 12  zip                    1296675 non-null  int64
 13  lat                    1296675 non-null  float64
 14  long                   1296675 non-null  float64
 15  city_pop               1296675 non-null  int64
 16  job                    1296675 non-null  object
 17  dob                    1296675 non-null  object
 18  trans_num              1296675 non-null  object
 19  unix_time              1296675 non-null  int64
 20  merch_lat              1296675 non-null  float64
 21  merch_long             1296675 non-null  float64
 22  is_fraud               1296675 non-null  int64
dtypes: float64(5), int64(6), object(12)
memory usage: 227.5+ MB
```

Creating new columns

In [13]:
```
#Transaction time seemigly plays a important role in respect to fraud.
#So we will make a new column of hour.
train_data['hour'] = pd.to_datetime(train_data['trans_date_trans_time']).
test_data['hour'] = pd.to_datetime(test_data['trans_date_trans_time']).dt

#Card usage or transaction frequency also might also play a role in relat
#So we will make a new column of CC_frequency.

freq_enc = (train_data.groupby("cc_num").size())
freq_enc.sort_values(ascending=True)
train_data["cc_frequency"] = train_data["cc_num"].apply(lambda x: freq_en

freq_enc = (test_data.groupby("cc_num").size())
freq_enc.sort_values(ascending=True)
test_data["cc_frequency"] = test_data["cc_num"].apply(lambda x: freq_enc[

# Age group might also play an important role in fraud transaction
# So we will make a new column of age.

train_data['age'] = dt.date.today().year-pd.to_datetime(train_data['dob']
test_data['age'] = dt.date.today().year-pd.to_datetime(test_data['dob']).
```

In [11]:
```
train_data.shape
```

Out[11]:
```
(1296675, 23)
```

In [12]:
```
test_data.shape
```

Out[12]:
```
(555719, 23)
```

```
100%|██████████| 100/100 [00:10<00:00,  9.84it/s]
```

```
 20  merch_lat              1296675 non-null  float64
 21  merch_long             1296675 non-null  float64
 22  is_fraud               1296675 non-null  int64
dtypes: float64(5), int64(6), object(12)
memory usage: 227.5+ MB
```

Creating new columns

In [13]:
```
#Transaction time seemigly plays a important role in respect to fraud.
#So we will make a new column of hour.

train_data['hour'] = pd.to_datetime(train_data['trans_date_trans_time']).
test_data['hour'] = pd.to_datetime(test_data['trans_date_trans_time']).dt

#Card usage or transaction frequency also might also play a role in relat
#So we will make a new column of CC_frequency.

freq_enc = (train_data.groupby("cc_num").size())
freq_enc.sort_values(ascending=True)
train_data["cc_frequency"] = train_data["cc_num"].apply(lambda x: freq_en

freq_enc = (test_data.groupby("cc_num").size())
freq_enc.sort_values(ascending=True)
test_data["cc_frequency"] = test_data["cc_num"].apply(lambda x: freq_enc[

# Age group might also play an important role in fraud transaction
# So we will make a new column of age.

train_data['age'] = dt.date.today().year-pd.to_datetime(train_data['dob']
test_data['age'] = dt.date.today().year-pd.to_datetime(test_data['dob']).
```

```
100%|██████████| 100/100 [00:10<00:00,  9.84it/s]
```

In [14]:
```
#Calculate distance between merchant and home location
train_data['latitudinal_distance'] = abs(round(train_data['merch_lat']-tr
train_df['longitudinal_distance'] = abs(round(train_data['merch_long']-tr

test_data['latitudinal_distance'] = abs(round(test_data['merch_lat']-test_
test_data['longitudinal_distance'] = abs(round(test_data['merch_long']-te
```

In [15]:
```
# Converting Male/ Female into 0/1 for the machine to read easily.
gender_mapping = {"F": 0, "M": 1}

train_data["gender_binary"] = train_data["gender"].map(gender_mapping)
test_data["gender_binary"] = test_data["gender"].map(gender_mapping)
```

In [16]:
```
#Drop Columns that are not relevant to predicy fraud transaction
drop_columns = ['cc_num','Unnamed: 0','street','zip','city_pop','trans_nu
train_data.drop(columns=drop_columns,inplace=True)
test_data.drop(columns=drop_columns,inplace=True)
```

In [17]:
```
# Making dummie columns for category variable.

test_data = pd.get_dummies(test_data,columns=['category'],drop_first=True
train_data = pd.get_dummies(train_data,columns=['category'],drop_first=Tr
```

In [18]:
```
# Scaling the amount and frequency variables.
from sklearn.preprocessing import StandardScaler

# Creating function for scaling
def Standard_Scaler (df, col_names):
    features = df[col_names]
    scaler = StandardScaler().fit(features.values)
    features = scaler.transform(features.values)
    df[col_names] = features
    return df

col_names = ['amt', 'cc_frequency']
train_data_std = Standard_Scaler (train_data, col_names)
test_data_std  = Standard_Scaler (test_data, col_names)
```

In [21]:
```
test_data.head()
```

Out[21]:

	amt	is_fraud	hour	cc_frequency	age	latitudinal_distance	longitudinal_distance	gend
0	-0.424483	0	12	-0.437280	55	0.020	0.265	
1	-0.252387	0	12	0.171606	33	0.870	0.476	
2	-0.179353	0	12	0.901033	53	0.177	0.660	
3	-0.052605	0	12	0.366192	36	0.243	0.064	
4	-0.422358	0	12	0.338509	66	0.706	0.868	

5 rows × 21 columns

In [19]:
```
train_data.shape
```

Out[19]:
```
(1296675, 21)
```

In [22]:
```
train_data.head()
```

Out[22]:

	amt	is_fraud	hour	cc_frequency	age	latitudinal_distance	longitudinal_distance	gend
0	0.407826	0	0	0.281849	35	0.068	0.870	
1	0.230039	0	0	1.631335	45	0.271	0.024	
2	0.934149	0	0	-1.772008	61	0.970	0.108	
3	-0.158132	0	0	-1.785476	56	0.804	0.447	
4	-0.177094	0	0	0.267035	37	0.254	0.830	

In [20]:
```
test_data.shape
```

Out[20]:
```
(555719, 21)
```

In [21]:
```
test_data.head()
```

Out[21]:

	amt	is_fraud	hour	cc_frequency	age	latitudinal_distance	longitudinal_distance	gend
0	-0.424483	0	12	0.437280	55	0.020	0.265	
1	-0.252337	0	12	0.171606	33	0.870	0.476	
2	-0.179353	0	12	0.901033	53	0.177	0.660	
3	-0.269685	0	12	0.366192	36	0.243	0.064	
4	-0.422358	0	12	0.338509	68	0.766	0.868	

5 rows × 21 columns

In [19]:
```
train_data.shape
```

Out[19]: (1296675, 21)

In [22]:
```
train_data.head()
```

Out[22]:

	amt	is_fraud	hour	cc_frequency	age	latitudinal_distance	longitudinal_distance	gend
0	0.407826	0	0	0.281849	35	0.068	0.870	
1	0.230039	0	0	1.631335	45	0.271	0.024	
2	0.934149	0	0	-1.772008	61	0.970	0.108	
3	-0.158132	0	0	-1.785476	56	0.804	0.447	
4	-0.177094	0	0	0.267035	37	0.254	0.830	

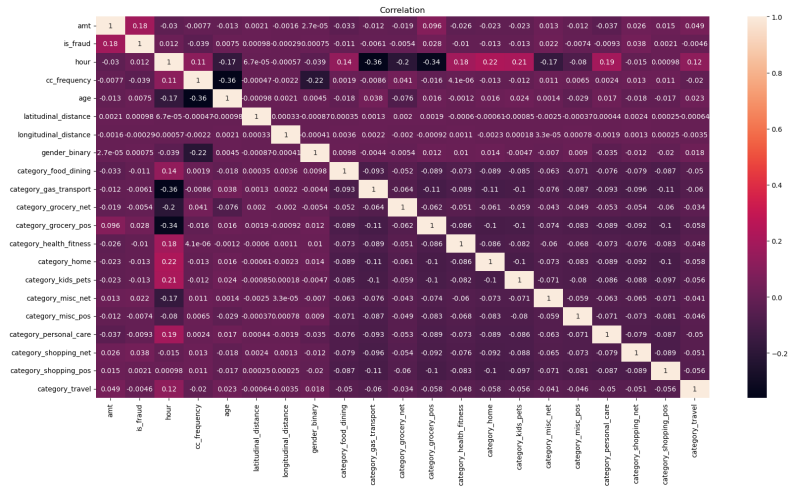
5 rows × 21 columns

In [20]:
```
test_data.shape
```

Out[20]: (555719, 21)

In [23]:
```
# creating a correlation heatmap using this we will check for any multico
# multicollinearity is when two variables have a correlation >0.7 with ea
import seaborn as sns
fig, ax = plt.subplots(figsize=(20,10))
sns.heatmap(test_data.corr(), annot=True).set_title('Correlation')
```

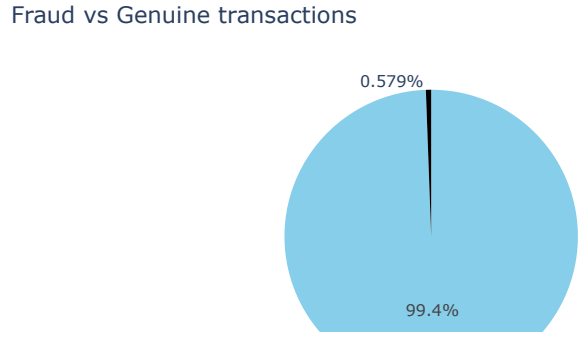
Out[23]: Text(0.5, 1.0, 'Correlation')



Checking the data balance in target vector: is_fraud

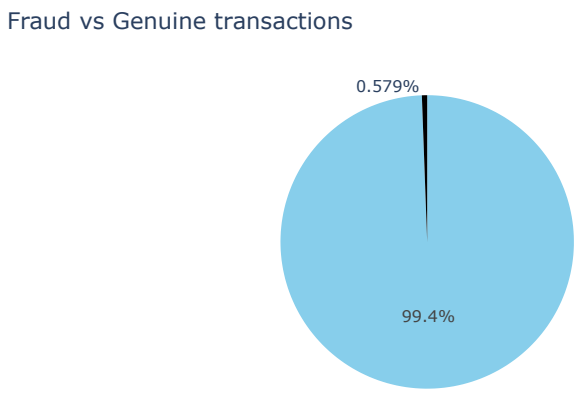
In [24]:
```
# As we know fraud transactions are very less than the genuine transactio
# We explore how imbalanced the data is.

import plotly.express as px
fig = px.pie(values=train_data['is_fraud'].value_counts(), names=["Genuin
             ,title="Fraud vs Genuine transactions")
fig.show()
```



In [24]:
```
# As we know fraud transactions are very less than the genuine transactio
# We explore how imbalanced the data is.

import plotly.express as px
fig = px.pie(values=train_data['is_fraud'].value_counts(), names=["Genuin
             ,title="Fraud vs Genuine transactions")
fig.show()
```



In [25]:
```
#Spliting the data into X = train data, y = target column.
```

In [26]:
```
X_train_data = train_data.drop('is_fraud', axis = 1)
y_train_data = train_data['is_fraud']

X_test_data = test_data.drop('is_fraud', axis = 1)
y_test_data = test_data['is_fraud']
```

We can see that our data set is unbalanced as only 0.58% data contains frudulent transactions which we have to label.

Thus lets explore how we can manage this by using SMOTE (Synthetic Minority Oversampeling Technique).

In [29]:
```
smote_data.head()
```

In [27]:
```
from imblearn.over_sampling import SMOTE

# Initialize the SMOTE object
smote = SMOTE(random_state=42)

# Apply SMOTE to the training data
X_train_data, y_train_data = smote.fit_resample(X_train_data, y_train_dat
```

Out[29]:

	amt	hour	cc_frequency	age	latitudinal_distance	longitudinal_distance	gender_binary
0	-0.407826	0	0.281649	35	0.068	0.870	0
1	0.230939	0	1.681335	45	0.271	0.024	0
2	0.934149	0	-1.772088	61	0.970	0.108	1
3	-0.158132	0	-1.785476	56	0.804	0.447	1
4	-0.177094	0	0.267035	37	0.254	0.830	1

5 rows × 21 columns

In [28]:
```
L = [X_train_data, y_train_data]
smote_data = pd.concat(L, ignore_index=False, axis=1)
```

In [30]:
```
y_train_data.info()
```

```
<class 'pandas.core.series.Series'>
RangeIndex: 2578338 entries, 0 to 2578337
Series name: is_fraud
Non-Null Count    Dtype
--------------    -----
2578338 non-null  int64
dtypes: int64(1)
memory usage: 19.7 MB
```

In [31]:
```
# Checking the change in train data after SMOTE
```

Oversampling Technique):

In [29]:
```
smote_data.head()
```

In [27]:
```
from imblearn.over_sampling import SMOTE

# Initialize the SMOTE object
smote = SMOTE(random_state=42)

# Apply SMOTE to the training data
X_train_data, y_train_data = smote.fit_resample(X_train_data, y_train_dat
```

Out[29]:

	amt	hour	cc_frequency	age	latitudinal_distance	longitudinal_distance	gender_binary
0	-0.407826	0	0.281849	35	0.068	0.870	0
1	0.230039	0	1.684335	45	0.271	0.024	0
2	0.934149	0	-1.772068	61	0.970	0.108	1
3	-0.158132	0	-1.785476	56	0.804	0.447	1
4	-0.177094	0	0.267035	37	0.254	0.830	1

5 rows × 21 columns

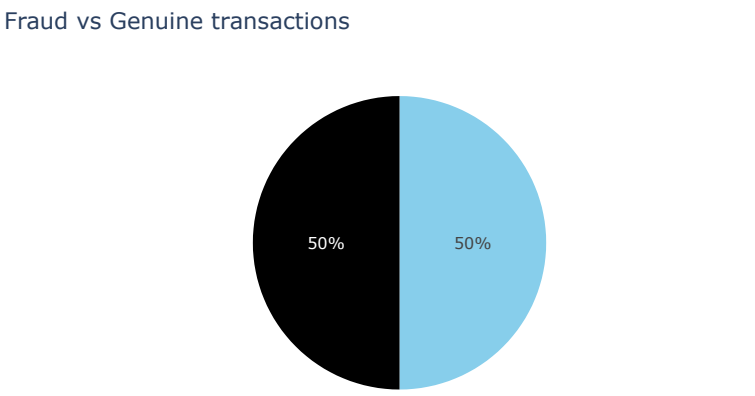
In [28]:
```
L = [X_train_data, y_train_data]
smote_data = pd.concat(L, ignore_index=False, axis=1)
```

In [30]:
```
y_train_data.info()
```

```
<class 'pandas.core.series.Series'>
RangeIndex: 2578338 entries, 0 to 2578337
Series name: is_fraud
Non-Null Count    Dtype
--------------    -----
2578338 non-null  int64
dtypes: int64(1)
memory usage: 19.7 MB
```

In [31]:
```
# Checking the change in train data after SMOTE

fig = px.pie(values=smote_data['is_fraud'].value_counts(), names=["Genuin
             ,title="Fraud vs Genuine transactions")
fig.show()
```



In [35]:
```
from sklearn.metrics import confusion_matrix, recall_score, precision_sco
cm = confusion_matrix(y_test_data, y_pred)
rf_Recall = recall_score(y_test_data, y_pred)
rf_Precision = precision_score(y_test_data, y_pred)
rf_f1 = f1_score(y_test_data, y_pred)
rf_accuracy = accuracy_score(y_test_data, y_pred)
score = [rf_Recall, rf_Precision, rf_f1, rf_accuracy)]
rf_score = pd.DataFrame(data=score, columns=['Recall','Precision','F1 S
rf_score.insert(0, 'Model', 'RandomeForestClassifier')
rf_score
```

In [32]:
```
# Importing following Models for testing scores.

from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import GradientBoostingClassifier
from lightgbm import LGBMClassifier
rf = RandomForestClassifier(random_state=23)
knn = KNeighborsClassifier()
gboost = GradientBoostingClassifier(random_state=23)
lgbm = LGBMClassifier(random_state=23)
```

In [33]:
```
# Model Fitting RamdomForestClassifier

rf.fit(X_train_data, y_train_data)
rf_score
```

Out[33]:
```
RandomForestClassifier(random_state=23)
```

Out[35]:

	Model	Recall	Precision	F1 Score	Accuracy
0	RandomeForestClassifier	0.861072	0.241343	0.377016	0.989016

In [34]:
```
# Making prediction using the Model

y_pred = rf.predict(X_test_data)
```

In [36]:
```
print(cm)
```

```
[[547768   5806]
 [   298   1847]]
```

In [37]:
```
# Model Fitting KNeighborsClassifier

```

In [35]:
```
from sklearn.metrics import confusion_matrix, recall_score, precision_sco
# Importing following Models for testing scores.

from sklearn.ensemble import RandomForestClassifier
cm = confusion_matrix(y_test_data, y_pred)
from sklearn.neighbors import KNeighborsClassifier

from sklearn.ensemble import GradientBoostingClassifier
rf_Recall = recall_score(y_test_data, y_pred)
from lightgbm import LGBMClassifier
rf_Precision = precision_score(y_test_data, y_pred)

rf_f1 = f1_score(y_test_data, y_pred)

rf_accuracy = accuracy_score(y_test_data, y_pred)
rf = RandomForestClassifier(random_state=23)

knn = KNeighborsClassifier()
score = [(rf_Recall, rf_Precision, rf_f1, rf_accuracy)]
gboost = GradientBoostingClassifier(random_state=23)

lgbm = LGBMClassifier(random_state=23)
rf_score = pd.DataFrame(data = score, columns=['Recall','Precision','F1 S
rf_score.insert(0, 'Model', 'RandomeForestClassifier')
rf_score
```

In [33]:
```
# Model Fitting RamdomForestClassifier
rf.fit(X_train_data, y_train_data)
rf_score
```

Out[33]:
```
RandomForestClassifier(random_state=23)
```

Out[35]:

	Model	Recall	Precision	F1 Score	Accuracy
0	RandomeForestClassifier	0.861072	0.241343	0.377016	0.989016

In [34]:
```
# Making prediction using the Model
print(cm)

y_pred = rf.predict(X_test_data)
```

In [36]:
```
[[547768   5806]
 [   298   1847]]
```

In [37]:
```
# Model Fitting KNeighborsClassifier

knn.fit(X_train_data, y_train_data)
```

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

In [38]:
```
# Making prediction using the Model KNeighborsClassifier

y_pred = knn.predict(X_test_data)
```

In [39]:
```
cm = confusion_matrix(y_test_data, y_pred)

knn_Recall = recall_score(y_test_data, y_pred)
knn_Precision = precision_score(y_test_data, y_pred)
knn_f1 = f1_score(y_test_data, y_pred)
knn_accuracy = accuracy_score(y_test_data, y_pred)

score = [(knn_Recall, knn_Precision, knn_f1, knn_accuracy)]

knn_score = pd.DataFrame(data = score, columns=['Recall','Precision','F1 S
knn_score.insert(0, 'Model', 'KNeighborsClassifier')
knn_score

knn_score
```

Out[39]:

	Model	Recall	Precision	F1 Score	Accuracy
0	KNeighborsClassifier	0.735664	0.190557	0.302705	0.986918

In [43]:
```
cm = confusion_matrix(y_test_data, y_pred)
print(cm)
gboost_Recall = recall_score(y_test_data, y_pred)
gboost_Precision = precision_score(y_test_data, y_pred)
gboost_f1 = f1_score(y_test_data, y_pred)
gboost_accuracy = accuracy_score(y_test_data, y_pred)

score = [(gboost_Recall, gboost_Precision, gboost_f1, gboost_accuracy)]
```

In [40]:
```
[[546877   6703]
 [   307   1838]]
```

In [41]:
```
# Model Fitting GradientBoostingClassifier.
gboost.fit(X_train_data, y_train_data)
gboost_score = pd.DataFrame(data = score, columns=['Recall','Precision','
gboost_score.insert(0, 'Model', 'GradientBoostingClassifier')
gboost_score

gboost_score
```

Out[41]:
```
GradientBoostingClassifier(random_state=23)
```

Out[43]:

	Model	Recall	Precision	F1 Score	Accuracy
0	GradientBoostingClassifier	0.940793	0.080057	0.147558	0.958044

In [42]:
```
# Making prediction using the Model
y_pred = gboost.predict(X_test_data)
```

In [44]:
```
print (cm)
```

```
[[530385  23189]
 [   127   2018]]
```

In [45]:
```
# Making prediction using the Model LGBMClassifier
lgbm.fit(X_train_data, y_train_data)
```

```
[LightGBM] [Info] Number of positive: 1289169, number of negative: 1289169
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of te
```

In [43]:
```
cm = confusion_matrix(y_test_data, y_pred)
print(cm)
```

```
[[546871   6703]
 [   567   1578]]
```

In [40]:
```
gboost_Recall = recall_score(y_test_data, y_pred)
gboost_Precision = precision_score(y_test_data, y_pred)
gboost_f1 = f1_score(y_test_data, y_pred)
gboost_accuracy = accuracy_score(y_test_data, y_pred)

score = [(gboost_Recall, gboost_Precision, gboost_f1, gboost_accuracy)]

gboost_score = pd.DataFrame(data = score, columns=['Recall','Precision','
gboost_score.insert(0, 'Model', 'GradientBoostingClassifier')
gboost_score

gboost_score
```

In [41]:
```
# Model Fitting GradientBoostingClassifier.
gboost.fit(X_train_data, y_train_data)
```

Out[41]:
```
GradientBoostingClassifier
GradientBoostingClassifier(random_state=23)
```

Out[43]:

	Model	Recall	Precision	F1 Score	Accuracy
0	GradientBoostingClassifier	0.940793	0.080057	0.147558	0.958044

In [42]:
```
# Making prediction using the Model
y_pred = gboost.predict(X_test_data)
```

In [44]:
```
print (cm)
```

```
[[530385  23189]
 [   127   2018]]
```

In [45]:
```
# Making prediction using the Model LGBMClassifier
lgbm.fit(X_train_data, y_train_data)
```

```
[LightGBM] [Info] Number of positive: 1289169, number of negative: 1289169
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of te
sting was 0.014545 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1154
[LightGBM] [Info] Number of data points in the train set: 2578338, number of
used features: 20
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=0.0000
00
```

Out[45]:
```
LGBMClassifier
LGBMClassifier(random_state=23)
```

In [46]:
```
# Making prediction using the Model LGBMClassifier

y_pred = lgbm.predict(X_test_data)
```

In [47]:
```
cm = confusion_matrix(y_test_data, y_pred)

lgbm_Recall = recall_score(y_test_data, y_pred)
lgbm_Precision = precision_score(y_test_data, y_pred)
lgbm_f1 = f1_score(y_test_data, y_pred)
lgbm_accuracy = accuracy_score(y_test_data, y_pred)

score = [(lgbm_Recall, lgbm_Precision, lgbm_f1, lgbm_accuracy)]

lgbm_score = pd.DataFrame(data = score, columns=['Recall','Precision','F1
lgbm_score.insert(0, 'Model', 'LGBMClassifier')
lgbm_score

lgbm_score
```

Out[47]:

	Model	Recall	Precision	F1 Score	Accuracy
0	LGBMClassifier	0.951981	0.10072	0.182167	0.967007

In [48]:
```
print(cm)
```

```
[[535342  18232]
 [   103   2042]]
```

Comparing the performance of different Models.

In [49]:
```
predictions = pd.concat([rf_score, knn_score, lgbm_score, gboost_score],
predictions.sort_values(by=['Recall'], ascending=False)
```

Out[49]:

	Model	Recall	Precision	F1 Score	Accuracy
2	LGBMClassifier	0.951981	0.100720	0.182167	0.967007
3	GradientBoostingClassifier	0.940793	0.080057	0.147558	0.958044
0	RandomeForestClassifier	0.861072	0.241343	0.377016	0.989016
1	KNeighborsClassifier	0.735664	0.190557	0.302705	0.986918

```
13
14 lgbm_score
```

Out[47]:

	Model	Recall	Precision	F1 Score	Accuracy
0	LGBMClassifier	0.951981	0.10072	0.182167	0.967007

In [48]:

```
1 print(cm)
```

```
[[535342  18232]
 [   103   2042]]
```

Comparing the performance of different Models.

In [49]:

```
1 predictions = pd.concat([rf_score, knn_score, lgbm_score, gboost_score],
2 predictions.sort_values(by=['Recall'], ascending=False)
```

Out[49]:

	Model	Recall	Precision	F1 Score	Accuracy
2	LGBMClassifier	0.951981	0.100720	0.182167	0.967007
3	GradientBoostingClassifier	0.940793	0.080057	0.147558	0.958044
0	RandomeForestClassifier	0.861072	0.241343	0.377016	0.989016
1	KNeighborsClassifier	0.735664	0.190557	0.302705	0.986918