Business Understanding

Credit Card Fraud Detection is a classic class-imbalance problem where the number of fraud transactions is much lesser than the number of legitimate transaction for any bank. Most of the approaches involve building model on such imbalanced data, and thus fails to produce results on real-time new data because of overfitting on training data and a bias towards the majoritarian class of legitimate transactions. Thus, we can see this as an anomaly detection problem.

In [69]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from pylab import rcParams
import warnings
from sklearn.model_selection import GridSearchCV
warnings.filterwarnings('ignore')
```

In [63]:
```
pip install xgboost
```

```
Defaulting to user installation because normal site-packages is not wri
teableNote: you may need to restart the kernel to use updated packages.

Collecting xgboost
  Obtaining dependency information for xgboost from https://files.pytho
nhosted.org/packages/e2/7b/8c1b410cd0604cee9a167a19f7e1746f5b92ae7d02ad
574ab560b73c5a48/xgboost-2.1.1-py3-none-win_amd64.whl.metadata (http
s://files.pythonhosted.org/packages/e2/7b/8c1b410cd0604cee9a167a19f7e17
46f5b92ae7d02ad574ab560b73c5a48/xgboost-2.1.1-py3-none-win_amd64.whl.me
tadata)
  Downloading xgboost-2.1.1-py3-none-win_amd64.whl.metadata (2.1 kB)
Requirement already satisfied: numpy in c:\users\zisha\appdata\roaming
\python\python311\site-packages (from xgboost) (1.24.3)
Requirement already satisfied: scipy in c:\users\zisha\appdata\roaming
\python\python311\site-packages (from xgboost) (1.11.3)
Downloading xgboost-2.1.1-py3-none-win_amd64.whl (124.9 MB)
   ---------------------------------------- 0.0/124.9 MB ? eta -:--:--
   ---------------------------------------- 0.0/124.9 MB 1.4 MB/s eta
0:01:32
```

In [2]:
```
#READING DATASET :
```

In [3]:
```
df = pd.read_csv(r'C:\Users\zisha\Downloads\creditcard.csv')
```

In [4]:
```
df.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

5 rows × 31 columns

In [5]:
```
#Null Values
```

In [6]:
```
df.isnull().sum()
```

Out[6]:
```
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 [7]:
```
#Thus there are no null values in the dataset.

#INFORMATION
```

In [8]:
```
df.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]:
```
#DESCRIPTIVE STATISTICS
```

In [10]:
```
df.describe().T.head()
```

Out[10]:

	count	mean	std	min	25%	50%	
Time	284807.0	9.481386e+04	47488.145955	0.000000	54201.500000	84692.000000	139320
V1	284807.0	1.759061e-12	1.958696	-56.407510	-0.920373	0.018109	1
V2	284807.0	-8.251130e-13	1.651309	-72.715728	-0.598550	0.065486	0
V3	284807.0	-9.654937e-13	1.516255	-48.325589	-0.890365	0.179846	1
V4	284807.0	8.321385e-13	1.415869	-5.683171	-0.848640	-0.019847	0

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

Out[11]:
```
(284807, 31)
```

In [12]:
```
#Thus there are 284807 rows and 31 columns.
```

In [13]:
```
df.columns
```

Out[13]:
```
Index(['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V1
0',
       'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V2
0',
       'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount',
       'Class'],
      dtype='object')
```

In [14]:
```
#FRAUD CASES AND GENUINE CASES
```

In [15]:
```
fraud_cases=len(df[df['Class']==1])
```

In [16]:
```
print(' Number of Fraud Cases:',fraud_cases)
```

```
 Number of Fraud Cases: 492
```

In [17]:
```
non_fraud_cases=len(df[df['Class']==0])
```

In [18]:
```
print('Number of Non Fraud Cases:',non_fraud_cases)
```

```
Number of Non Fraud Cases: 284315
```

In [19]:
```
fraud=df[df['Class']==1]
```

In [20]:
```
genuine=df[df['Class']==0]
```

In [21]:
```
fraud.Amount.describe()
```

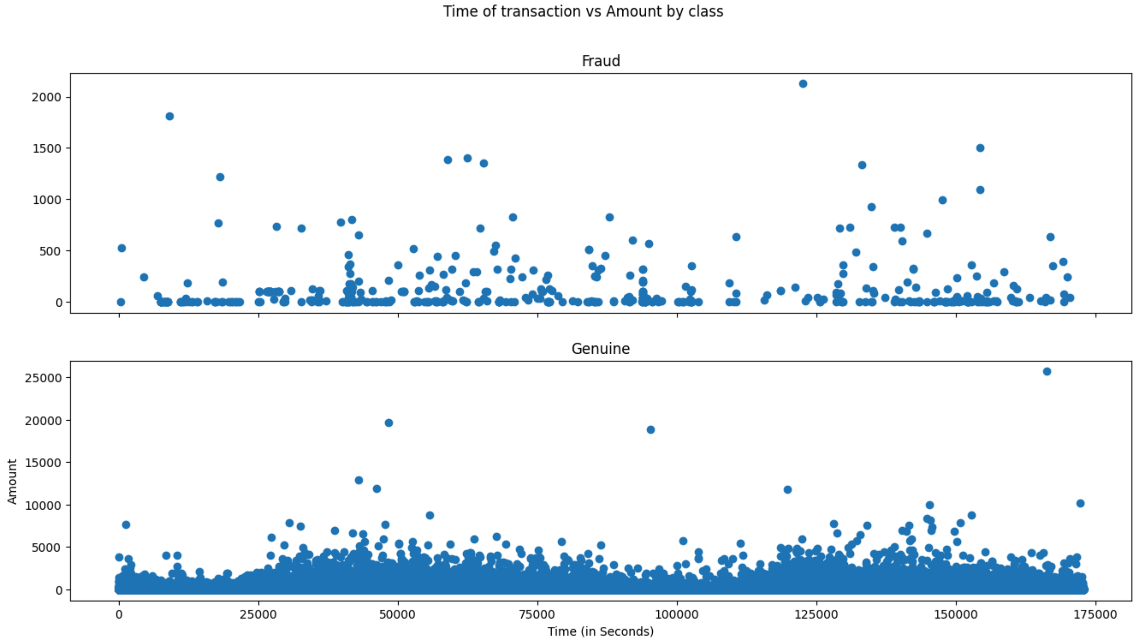
Out[21]:
```
count     492.000000
mean      122.211321
std       256.683288
min         0.000000
25%         1.000000
50%         9.250000
75%       105.890000
max      2125.870000
Name: Amount, dtype: float64
```

In [22]:
```
genuine.Amount.describe()
```

Out[22]:
```
count    284315.000000
mean         88.291022
std         250.105092
min           0.000000
25%           5.650000
50%          22.000000
75%          77.050000
max       25691.160000
Name: Amount, dtype: float64
```

In [23]:
```
#EDA
```

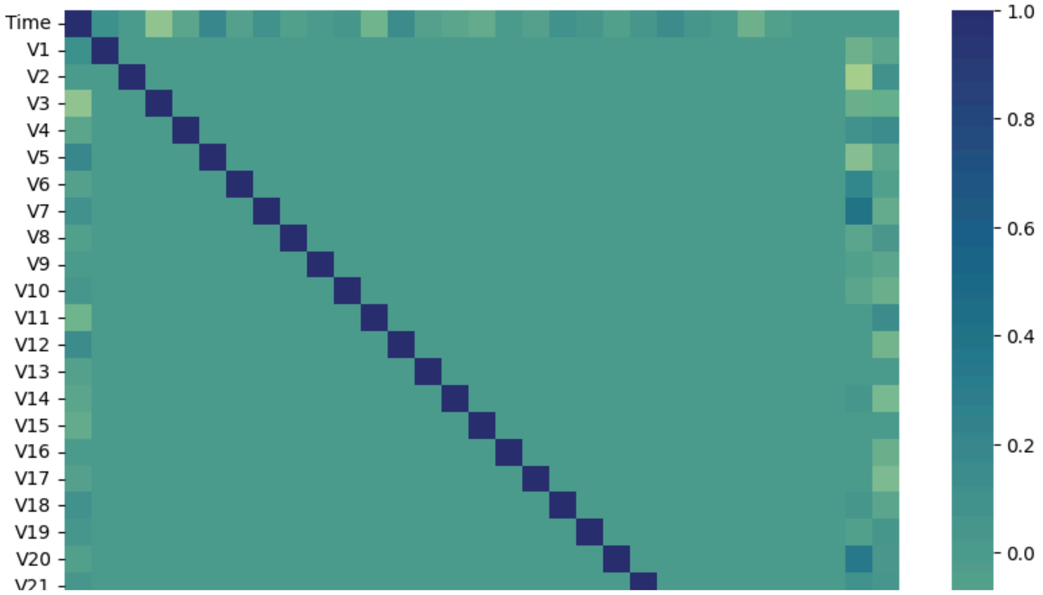
In [24]:
```
rcParams['figure.figsize'] = 16, 8
f,(ax1, ax2) = plt.subplots(2, 1, sharex=True)
f.suptitle('Time of transaction vs Amount by class')
ax1.scatter(fraud.Time, fraud.Amount)
ax1.set_title('Fraud')
ax2.scatter(genuine.Time, genuine.Amount)
ax2.set_title('Genuine')
plt.xlabel('Time (in Seconds)')
plt.ylabel('Amount')
plt.show()
```



In [25]:
```
#CORRELATION
```

In [26]:
```
plt.figure(figsize=(10,8))
corr=df.corr()
sns.heatmap(corr,cmap='crest')
```

Out[26]: <Axes: >



In [27]:
```
from sklearn.model_selection import train_test_split
```

In [28]:
```
#Model 1:
```

In [29]:
```
X=df.drop(['Class'],axis=1)
```

In [30]:
```
y=df['Class']
```

In [31]:
```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.20,random_st
```

In [32]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [33]:
```
rfc=RandomForestClassifier()
```

In [34]:
```
model=rfc.fit(X_train,y_train)
```

In [35]:
```
yprediction=model.predict(X_test)
```

In [36]:
```
from sklearn.metrics import accuracy_score
```

In [37]:
```
accuracy_score(y_test,yprediction)
```

Out[37]: 0.9994908886626171

In [38]:
```
#Model 2:
```

In [39]:
```
from sklearn.linear_model import LogisticRegression
```

In [40]:
```
X1=df.drop(['Class'],axis=1)
```

In [41]:
```
y1=df['Class']
```

In [42]:
```
X1_train,X1_test,y1_train,y1_test=train_test_split(X1,y1,test_size=0.2,rand
```

In [43]:
```
lr=LogisticRegression()
```

In [44]:
```
model2=lr.fit(X1_train,y1_train)
```

In [45]:
```
prediction2=model2.predict(X1_test)
```

In [46]:
```
x_train_prediction2=model2.predict(X1_train)
```

In [47]:
```
accuracy_score(y1_test,prediction2)
```

Out[47]:
```
0.9989291106351603
```

In [48]:
```
#Model 3:
```

In [49]:
```
from sklearn.tree import DecisionTreeRegressor
```

In [50]:
```
X2=df.drop(['Class'],axis=1)
```

In [51]:
```
y2=df['Class']
```

In [52]:
```
dt=DecisionTreeRegressor()
```

In [53]:
```
X2_train,X2_test,y2_train,y2_test=train_test_split(X2,y2,test_size=0.2,rand
```

In [54]:
```
model3=dt.fit(X2_train,y2_train)
```

In [55]:
```
prediction3=model3.predict(X2_test)
```

In [56]:
```
accuracy_score(y2_test,prediction3)
```

Out[56]:
```
0.9990695551420246
```

Hence Accurcy of model is greater 75%

In [57]:
```
x_train_prediction2 = model2.predict(X1_train)
```

In [58]:
```
x_test_prediction2=model2.predict(X1_test)
```

In [59]:
```
 from sklearn.metrics import f1_score
```

In [60]:
```
# F1 Score for traning data predictions
f1_score_train = f1_score(y1_train, x_train_prediction2)
print ('Training data F1 Score =', f1_score_train)
```

```
Training data F1 Score = 0.686030428769018
```

In [61]:
```
# F1 Scor for test data predictions
f1_score_train = f1_score(y1_test, x_test_prediction2)
print ('Training data F1 Score =', f1_score_train)
```

```
Training data F1 Score = 0.7214611872146118
```

In [64]:
```
from xgboost import XGBRegressor
xgb_model = XGBRegressor(random_state = 123)
```

In [65]:
```
# make a dictionary of hyperparameter values to search
search_space = {
    "n_estimators" : [100, 200, 500],
    "max_depth" : [3, 6, 9],
    "gamma" : [0.01, 0.1],
    "learning_rate" : [0.001, 0.01, 0.1, 1]
}
```

In [71]:
```
from sklearn.model_selection import GridSearchCV
# make a GridSearchCV object
GS = GridSearchCV(estimator = xgb_model,
                  param_grid = search_space,
                  scoring = ["r2", "neg_root_mean_squared_error"], #sklearn
                  refit = "r2",
                  cv = 5,
                  verbose = 4)
```

In [74]:
```
GS.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 72 candidates, totalling 360 fits
[CV 1/5] END gamma=0.01, learning_rate=0.001, max_depth=3, n_estimators
=100; neg_root_mean_squared_error: (test=-0.039) r2: (test=0.119) total
time=   3.1s
[CV 2/5] END gamma=0.01, learning_rate=0.001, max_depth=3, n_estimators
=100; neg_root_mean_squared_error: (test=-0.038) r2: (test=0.124) total
time=   2.9s
[CV 3/5] END gamma=0.01, learning_rate=0.001, max_depth=3, n_estimators
=100; neg_root_mean_squared_error: (test=-0.038) r2: (test=0.121) total
time=   3.2s
[CV 4/5] END gamma=0.01, learning_rate=0.001, max_depth=3, n_estimators
=100; neg_root_mean_squared_error: (test=-0.040) r2: (test=0.112) total
time=   2.7s
[CV 5/5] END gamma=0.01, learning_rate=0.001, max_depth=3, n_estimators
=100; neg_root_mean_squared_error: (test=-0.036) r2: (test=0.126) total
time=   3.2s
[CV 1/5] END gamma=0.01, learning_rate=0.001, max_depth=3, n_estimators
=200; neg_root_mean_squared_error: (test=-0.036) r2: (test=0.218) total
time=   4.8s
```

In [75]:
```
print(GS.best_estimator_) # to get the complete details of the best modelb
```

```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=Non
e,
             enable_categorical=False, eval_metric=None, feature_types=Non
e,
             gamma=0.01, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=0.1, max_bin=Non
e,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=6, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=Non
e,
             multi_strategy=None, n_estimators=500, n_jobs=None,
             num_parallel_tree=None, random_state=123, ...)
```

In [76]:
```
print(GS.best_params_) # to get only the best hyperparameter values that we
```

```
{'gamma': 0.01, 'learning_rate': 0.1, 'max_depth': 6, 'n_estimators': 500}
```

In [77]:
```
print(GS.best_score_) # score according to the metric we passed in refit
```

```
0.7449093400036638
```