In [221…

```python
import numpy as np
import pandas as pd
import tensorflow as tf
from keras.models import Sequential
from keras.layers import Dense, Dropout
from sklearn.metrics import mean_squared_error,mean_absolute_error,mean_squared_e
from sklearn.metrics import r2_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report,confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
```

In [169…

```python
train = pd.read_csv('/Users/ranood/Desktop/Financial/train_data.csv')
test = pd.read_csv('/Users/ranood/Desktop/Financial/test_data_hidden.csv')
```

In [94]:

```python
test
```

Out[94]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8
0	113050.0	0.114697	0.796303	-0.149553	-0.823011	0.878763	-0.553152	0.939259	-0.108502
1	26667.0	-0.039318	0.495784	-0.810884	0.546693	1.986257	4.386342	-1.344891	-1.743736
2	159519.0	2.275706	-1.531508	-1.021969	-1.602152	-1.220329	-0.462376	-1.196485	-0.147058
3	137545.0	1.940137	-0.357671	-1.210551	0.382523	0.050823	-0.171322	-0.109124	-0.002115
4	63369.0	1.081395	-0.502615	1.075887	-0.543359	-1.472946	-1.065484	-0.443231	-0.143374
...	...	...	...	...	...	...	...	...	...
56957	136579.0	2.030797	-0.825073	-0.729555	-0.519187	-0.639893	-0.169482	-0.619049	-0.017902
56958	150070.0	-0.263947	1.119700	-0.639394	-0.880567	1.194120	-0.310693	0.962087	-0.088880
56959	138634.0	2.206867	-0.748559	-1.443015	-1.101542	-0.332197	-0.646931	-0.536272	-0.129437
56960	53907.0	1.430579	-0.842354	0.415998	-1.328439	-1.284654	-0.888110	-0.653237	-0.238164
56961	66373.0	-7.792712	5.599937	0.258943	0.061360	-2.586555	4.770837	-8.221863	-20.298380

56962 rows × 31 columns

In [95]:

```python
#Perform an EDA on the Dataset
train.describe()
```

Out[95]:

	Time	V1	V2	V3	V4	V5	
count	227845.000000	227845.000000	227845.000000	227845.000000	227845.000000	227845.000000	227
mean	94752.853076	-0.003321	-0.001652	0.001066	-0.000374	0.000877	
std	47500.410602	1.963028	1.661178	1.516107	1.415061	1.367074	
min	0.000000	-56.407510	-72.715728	-32.965346	-5.683171	-42.147898	
25%	54182.000000	-0.922851	-0.598040	-0.889246	-0.848884	-0.690811	
50%	84607.000000	0.012663	0.066665	0.182170	-0.019309	-0.055243	
75%	139340.000000	1.314821	0.804401	1.029449	0.744822	0.610852	
max	172792.000000	2.454930	22.057729	9.382558	16.875344	34.801666	

8 rows × 31 columns

In [96]:

```python
#Find if there is any connection between Time, Amount, and the transaction being
corr = train.corr()
corr
```

Out[96]:

	Time	V1	V2	V3	V4	V5	V6	V7	V
Time	1.000000	0.117777	-0.010475	-0.419784	-0.105047	0.173334	-0.063703	0.084606	-0.03698
V1	0.117777	1.000000	0.008229	0.001239	-0.002554	-0.003554	0.003348	0.002552	-0.0018
V2	-0.010475	0.008229	1.000000	0.003076	-0.001860	-0.004229	0.001530	0.001980	-0.0003
V3	-0.419784	0.001239	0.003076	1.000000	-0.001147	-0.005848	0.004419	0.007959	0.0007
V4	-0.105047	-0.002554	-0.001860	-0.001147	1.000000	0.001981	-0.003068	-0.003616	-0.0013
V5	0.173334	-0.003554	-0.004229	-0.005848	0.001981	1.000000	0.014932	0.025536	-0.0046
V6	-0.063703	0.003348	0.001530	0.004419	-0.003068	0.014932	1.000000	-0.017351	0.0044
V7	0.084606	0.002552	0.001980	0.007959	-0.003616	0.025536	-0.017351	1.000000	0.0077
V8	-0.036989	-0.001899	-0.000321	0.000722	-0.001397	-0.004610	0.004478	0.007750	1.0000
V9	-0.009281	0.001410	-0.002448	0.000356	0.000070	-0.000297	0.000179	0.003864	-0.0001
V10	0.030671	0.001575	-0.004113	0.000990	0.001005	-0.003156	0.001427	0.007053	0.0021
V11	-0.249075	-0.000925	-0.000226	0.001510	0.003250	0.001762	-0.000244	-0.000213	-0.0024
V12	0.124282	-0.001735	-0.001595	-0.000445	0.001564	-0.003938	0.000958	0.006388	0.0036
V13	-0.066625	-0.000463	-0.001058	-0.000625	-0.000548	0.001445	-0.000971	-0.001361	0.0025
V14	-0.098673	-0.001540	-0.000624	-0.002743	0.001915	-0.001382	0.002442	0.002486	0.0048
V15	-0.183040	-0.000989	-0.000925	0.001516	-0.001500	0.002530	-0.002707	-0.002471	0.0016
V16	0.011148	-0.002626	-0.002382	0.001657	-0.001814	0.003845	-0.004873	-0.003515	0.0046
V17	-0.073465	-0.000568	-0.001191	-0.000760	-0.001052	-0.002759	-0.000066	0.005135	0.0053
V18	0.092087	0.000267	0.001879	0.001047	0.001431	0.000495	-0.000894	0.002213	0.0030
V19	0.028649	0.000583	0.002284	0.001547	-0.001817	0.001780	-0.000916	-0.003600	-0.0003
V20	-0.051160	-0.002032	-0.009700	-0.006893	0.003787	-0.016817	0.011611	0.021419	-0.0056
V21	0.044253	-0.001978	-0.003032	-0.002979	0.003349	-0.008510	0.005147	0.009986	-0.0002
V22	0.144670	0.002074	0.004273	0.001248	-0.002147	0.002784	-0.002171	-0.003077	-0.0010
V23	0.049683	0.010848	0.007231	0.008481	-0.001533	0.007772	-0.003562	-0.008366	0.0056
V24	-0.015547	-0.001089	0.002603	-0.000150	-0.001154	0.000823	-0.001478	-0.002986	0.0010
V25	-0.231976	0.003159	0.003127	0.003749	-0.000707	0.004833	-0.002102	-0.003375	0.0028
V26	-0.040369	0.002461	0.002701	0.001626	-0.000590	0.002322	-0.002203	-0.004025	0.0017
V27	-0.005736	0.014184	0.013555	0.013268	-0.007427	0.022597	-0.012878	-0.025285	0.0068
V28	-0.009289	-0.013830	-0.015893	-0.009067	0.004131	-0.010706	0.008897	0.017643	-0.0093
Amount	-0.011328	-0.228808	-0.537855	-0.207587	0.098695	-0.367801	0.201285	0.377417	-0.0979
Class	-0.012800	-0.103095	0.093795	-0.196385	0.134044	-0.099712	-0.042548	-0.196860	0.0133

31 rows × 31 columns

In [170…

```python
train=train.fillna(0.0)
test = test.fillna(0.0)
```

In [164…

```python
#Check the class count for each class. It's a class Imbalance problem
train.Class.value_counts()
```

Out[164…

```
0    227451
1       394
Name: Class, dtype: int64
```

In [208…

```python
from sklearn.preprocessing import StandardScaler
train['Amount'] = StandardScaler().fit_transform(train['Amount'].values.reshape(-
test['Amount'] = StandardScaler().fit_transform(test['Amount'].values.reshape(-1,
```

In [209…

```python
#resampling the data
from sklearn.utils import resample,shuffle
resampled_train = resample(train, random_state=0,replace=True)
resampled_train
```

Out[209…

	Time	V1	V2	V3	V4	V5	V6	V7	V8
199340	51106.0	0.912523	-0.659374	-0.185702	0.337227	-0.533990	-0.615183	0.241246	-0.100613
43567	149215.0	2.105219	-0.679972	-2.878544	-0.697267	0.689879	-0.768564	0.601819	-0.510514
173685	162070.0	1.920674	-0.502961	-0.386028	0.289099	-0.751019	-0.528680	-0.555433	0.010176
117952	11782.0	-2.380840	0.186371	1.494356	-0.944682	0.458592	-1.352140	0.683973	-0.505738
176963	62511.0	-1.636630	1.364288	0.692731	-0.013078	-0.350973	0.328026	-0.121401	1.027077
...	...	...	...	...	...	...	...	...	...
100439	8715.0	-0.447418	0.424726	3.214006	3.781284	-1.266217	1.173467	-0.798863	0.321206
132787	29316.0	-0.716717	1.073517	1.945507	1.414742	0.493326	-0.431706	1.741417	-0.373847
146523	140631.0	-0.976402	-0.033771	-1.161497	-1.089719	2.133701	-1.168664	1.325927	-0.252673
132742	150081.0	2.101029	-0.550179	-1.056237	0.832127	-0.590310	-0.811267	-0.278887	-0.239784
18751	133668.0	2.353997	-1.373840	-1.019958	-1.740606	-0.935964	-0.063832	-1.290681	-0.102815

227845 rows × 31 columns

In [210…

```python
resampled_test = resample(test, random_state=0,replace=True)
resampled_test
```

Out[210…

	Time	V1	V2	V3	V4	V5	V6	V7	V8
2732	125253.0	-3.322410	-5.000386	0.905887	-1.048763	2.868784	-1.772317	-2.768652	0.260680
43567	154560.0	-0.398460	1.317794	1.587745	3.095117	1.085044	0.265052	1.265179	-0.229037
42613	12836.0	-0.964034	-0.241693	3.664946	0.010621	-0.787341	1.166953	-0.328912	0.175454
52416	137560.0	2.078290	0.149326	-1.727627	0.406217	0.432470	-0.852913	0.178126	-0.227693
45891	24455.0	-0.666562	0.480840	3.402517	1.853352	-0.773867	1.311264	-0.686810	0.522063
...	...	...	...	...	...	...	...	...	...
32526	149957.0	-1.177417	-3.196110	-3.541038	-1.253830	1.161083	-0.954711	-0.320148	-3.294719
50658	118838.0	1.924982	-0.016883	-0.196387	1.347629	-0.392860	-0.478150	-0.251532	-0.113166
32216	35400.0	0.419528	0.908434	-1.353884	0.177473	2.750575	3.294971	-0.452299	-0.018172
4752	139042.0	2.116263	-0.010741	-1.559121	0.179385	0.362134	-0.803738	0.207659	-0.341591
16980	55006.0	-1.515695	-0.966944	2.214393	-1.623239	0.077217	-1.100484	0.554041	-0.297650

56962 rows × 31 columns

In [238…

```python
resampled_test = resampled_test.drop(columns = ['Amount', 'Time'], axis=1)
resampled_train = resampled_train.drop(columns = ['Amount', 'Time'], axis=1)
```

In [239…

```python
X_train= resampled_train.drop('Class',axis=1)
```

In [240…

```python
X_train.shape
```

Out[240…

```
(227845, 28)
```

In [241…

```python
y_train = resampled_train['Class']
y_test = resampled_test['Class']
X_test= resampled_test.drop('Class',axis=1)
```

In [242…

```python
X_train = np.array(X_train)
X_test = np.array(X_test)
y_train = np.array(y_train)
y_test = np.array(y_test)
```

Naïve Bayes

In [136…

```python
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
y_pred = gnb.fit(X_train, y_train).predict(X_test)
accuracy_score(y_test, y_pred)
```

Out[136…

```
0.9928197745865665
```

In [137…

```python
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.99      1.00     56869
           1       0.13      0.59      0.21        93

    accuracy                           0.99     56962
   macro avg       0.56      0.79      0.60     56962
weighted avg       1.00      0.99      1.00     56962
```

In [138…

```python
print(mean_absolute_error(y_test, y_pred))
print(mean_squared_error(y_test, y_pred))
print(np.sqrt(mean_squared_error(y_test, y_pred)))
```

```
0.007180225413433517
0.007180225413433517
0.08473621075687487
```

XGBoost

In [139…

```python
from sklearn.ensemble import GradientBoostingClassifier
clf = GradientBoostingClassifier().fit(X_train,y_train)
pred=clf.predict(X_test)
clf.score(X_test, y_test)
```

Out[139…

```
0.9992099996488887
```

In [140…

```python
print(accuracy_score(y_test,pred))
```

```
0.9992099996488887
```

In [141…

```python
print(classification_report(y_test, pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56869
           1       0.84      0.63      0.72        93

    accuracy                           1.00     56962
   macro avg       0.92      0.82      0.86     56962
weighted avg       1.00      1.00      1.00     56962
```

Random-Forest

In [142…

```python
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier()
rf.fit(X_train, y_train)
rf_pred = rf.predict(X_test)
rf.score(X_test, y_test)
```

Out[142…

```
0.9994382219725431
```

In [143…

```python
print(accuracy_score(y_test,rf_pred))
```

```
0.9994382219725431
```

In [144…

```python
print(classification_report(y_test, rf_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56869
           1       0.93      0.71      0.80        93

    accuracy                           1.00     56962
   macro avg       0.96      0.85      0.90     56962
weighted avg       1.00      1.00      1.00     56962
```

The XGBoost shows the highest accuracy

Artificial Neural Networks

In [243…

```python
model = Sequential([
    Dense(units=30, input_dim = X_train.shape[1], activation='relu'),
    Dense(units=20,activation='relu'),
    Dropout(0.25),
    Dense(units=20,activation='relu'),
    Dense(units=30,activation='relu'),
    Dense(1, activation='sigmoid')
])
model.summary()
```

```
Model: "sequential_17"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_58 (Dense)             (None, 30)                870
_________________________________________________________________
dense_59 (Dense)             (None, 20)                620
_________________________________________________________________
dropout_4 (Dropout)          (None, 20)                0
_________________________________________________________________
dense_60 (Dense)             (None, 20)                420
_________________________________________________________________
dense_61 (Dense)             (None, 30)                630
_________________________________________________________________
dense_62 (Dense)             (None, 1)                 31
=================================================================
Total params: 2,571
Trainable params: 2,571
Non-trainable params: 0
_________________________________________________________________
```

In [244…

```python
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.fit(X_train, y_train, batch_size=32, epochs=5)
```

```
Epoch 1/5
7121/7121 [==============================] - 16s 2ms/step - loss: 0.0201 - accura
cy: 0.9986 0s - loss: 0.0202 - accuracy
Epoch 2/5
7121/7121 [==============================] - 11s 2ms/step - loss: 0.0031 - accura
cy: 0.9994
Epoch 3/5
7121/7121 [==============================] - 12s 2ms/step - loss: 0.0028 - accura
cy: 0.9994
Epoch 4/5
7121/7121 [==============================] - 14s 2ms/step - loss: 0.0023 - accura
cy: 0.9995
Epoch 5/5
7121/7121 [==============================] - 13s 2ms/step - loss: 0.0022 - accura
cy: 0.9995
```

Out[244…

```
<keras.callbacks.History at 0x7fd9f9be8670>
```

In [247…

```python
Acc,Loss = model.evaluate(X_test, y_test)
print('The accuracy of the model for testing: ',Acc,'\n', 'The Loss of the model
```

```
1781/1781 [==============================] - 2s 1ms/step - loss: 0.0043 - accurac
y: 0.9992
The accuracy of the model for testing:  0.0043174875900149345
 The Loss of the model for testing:  0.9991573095321655
```

In [248…

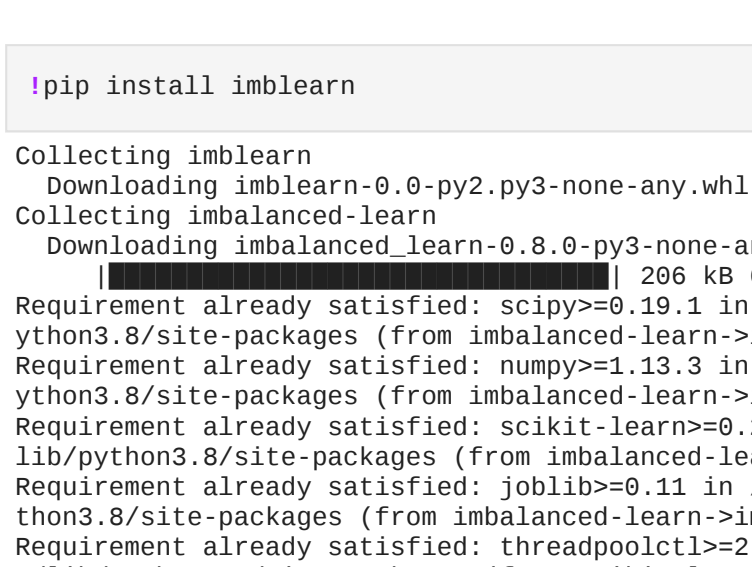
```python
y_test
```

Out[248…

```
array([0, 0, 0, ..., 0, 0, 0])
```

In [249…

```python
ANN_pred = model.predict(X_test)
y_test = pd.DataFrame(y_test)
CM = confusion_matrix(y_test,ANN_pred.round())
sns.heatmap(CM, annot=True,fmt='.0f',cmap='cividis_r')
plt.show()
```



In [250…

```python
print(classification_report(y_test, ANN_pred.round()))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56869
           1       0.74      0.74      0.74        93

    accuracy                           1.00     56962
   macro avg       0.87      0.87      0.87     56962
weighted avg       1.00      1.00      1.00     56962
```

In [246…

```python
!pip install imblearn
```

```
Collecting imblearn
  Downloading imblearn-0.0-py2.py3-none-any.whl (1.9 kB)
Collecting imbalanced-learn
  Downloading imbalanced_learn-0.8.0-py3-none-any.whl (206 kB)
     |████████████████████████████████| 206 kB 653 kB/s eta 0:00:01
Requirement already satisfied: scipy>=0.19.1 in /Users/ranood/opt/anaconda3/lib/p
ython3.8/site-packages (from imbalanced-learn->imblearn) (1.6.2)
Requirement already satisfied: numpy>=1.13.3 in /Users/ranood/opt/anaconda3/lib/p
ython3.8/site-packages (from imbalanced-learn->imblearn) (1.19.5)
Requirement already satisfied: scikit-learn>=0.24 in /Users/ranood/opt/anaconda3/
lib/python3.8/site-packages (from imbalanced-learn->imblearn) (0.24.1)
Requirement already satisfied: joblib>=0.11 in /Users/ranood/opt/anaconda3/lib/py
thon3.8/site-packages (from imbalanced-learn->imblearn) (1.0.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in /Users/ranood/opt/anaconda
3/lib/python3.8/site-packages (from scikit-learn>=0.24->imbalanced-learn->imblear
n) (2.1.0)
Installing collected packages: imbalanced-learn, imblearn
Successfully installed imbalanced-learn-0.8.0 imblearn-0.0
```

In [253…

```python
#balance the data with SMOTE Sampling
from imblearn.over_sampling import SMOTE
X_smote_train, y_smote_train = SMOTE().fit_resample(X_train, y_train)
X_smote_test, y_smote_test = SMOTE().fit_resample(X_test, y_test)

X_smote_train = pd.DataFrame(X_smote_train)
y_smote_train = pd.DataFrame(y_smote_train)

X_smote_test = pd.DataFrame(X_smote_test)
y_smote_test = pd.DataFrame(y_smote_test)
```

In [254…

```python
X_smote_train = np.array(X_smote_train)
X_smote_test = np.array(X_smote_test)
y_smote_train = np.array(y_smote_train)
y_smote_test = np.array(y_smote_test)
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.fit(X_train, y_train, batch_size = 32, epochs = 5)
```

```
Epoch 1/5
7121/7121 [==============================] - 16s 2ms/step - loss: 0.0021 - accura
cy: 0.9995
Epoch 2/5
7121/7121 [==============================] - 12s 2ms/step - loss: 0.0022 - accura
cy: 0.9995
Epoch 3/5
7121/7121 [==============================] - 14s 2ms/step - loss: 0.0018 - accura
cy: 0.9996
Epoch 4/5
7121/7121 [==============================] - 17s 2ms/step - loss: 0.0017 - accura
cy: 0.9995
Epoch 5/5
7121/7121 [==============================] - 15s 2ms/step - loss: 0.0020 - accura
cy: 0.9995
```

Out[254…

```
<keras.callbacks.History at 0x7fd9f8a4c460>
```

In [255…

```python
Acc,Loss = model.evaluate(X_smote_test, y_smote_test)
print('The accuracy of the model for testing: ',Acc,'\n', 'The Loss of the model
```

```
3555/3555 [==============================] - 5s 1ms/step - loss: 0.5120 - accurac
y: 0.8905
The accuracy of the model for testing:  0.512015700340271
 The Loss of the model for testing:  0.8904500007629395
```

In [257…

```python
ANN_pred2 = model.predict(X_smote_test)
y_test = pd.DataFrame(y_smote_test)
CM = confusion_matrix(y_smote_test,ANN_pred2.round())
sns.heatmap(CM, annot=True,fmt='.0f',cmap='cividis_r')
plt.show()
```


The ANN without SMOTE Sampling has higher accuracy

In [ ]: