Homework 2 - IEEE Fraud Detection

For all parts below, answer all parts as shown in the Google document for Homework 2. Be sure to include both code that justifies your answer as well as text to answer the questions. We also ask that code be commented to make it easier to follow.

In [26]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.feature_extraction import FeatureHasher

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression, LinearRegression
from sklearn.feature_extraction.text import CountVectorizer
from sklearn import metrics
```

Part 1 - Fraudulent vs Non-Fraudulent Transaction

Creating a train_transaction Dataframe which consists data from train_transaction.csv

Creating a train_identity Dataframe which consists data from train_identity.csv

Left join the two tables and naming it train_merge

In [39]:

```
# TODO: code and runtime results
train_transaction = pd.read_csv(r'C:\Users\preet\Desktop\ieee-fraud-detection
\train_transaction.csv')
train_identity = pd.read_csv(r'C:\Users\preet\Desktop\ieee-fraud-detection\tra
in_identity.csv')
train_merge = pd.merge(train_transaction, train_identity, on='TransactionID',
how='left')
train_transaction.TransactionDT = (train_transaction.TransactionDT%86400)/3600
```

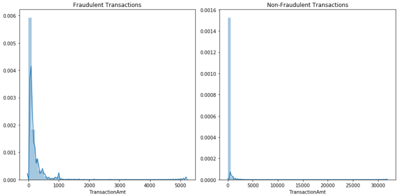
In [65]:

```
is_1 = train_transaction['isFraud'] == 1
tt1 = train_transaction[is_1]     #creating tt1 dataframe that has only those r
ows in which isFraud coloumn is == 1

is_0 = train_transaction['isFraud'] == 0
tt0 = train_transaction[is_0]   #creating tt0 dataframe that has only those row
s in which isFraud coloumn is == 0
```

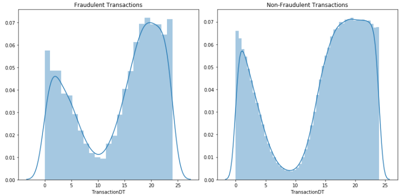
In [63]:

```
fig, axes = plt.subplots(1, 2,figsize=(12, 6))
sns.distplot(tt1['TransactionAmt'], ax=axes[0]).set_title('Fraudulent Transact
ions')
sns.distplot(tt0['TransactionAmt'], ax=axes[1]).set_title('Non-Fraudulent Tran
sactions')
plt.tight_layout()
```



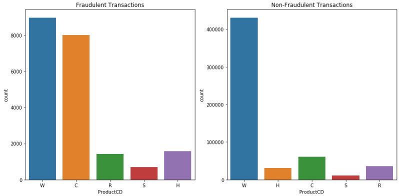
In [66]:

```
fig, axes = plt.subplots(1, 2,figsize=(12, 6))
sns.distplot(tt1['TransactionDT'], ax=axes[0]).set_title('Fraudulent Transacti
ons')
sns.distplot(tt0['TransactionDT'], ax=axes[1]).set_title('Non-Fraudulent Trans
actions')
plt.tight_layout()
```



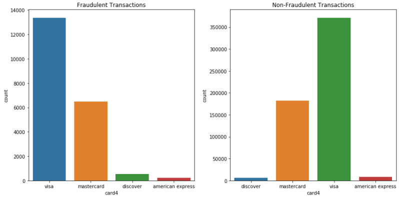
In [67]:

```
fig, axes = plt.subplots(1, 2,figsize=(12, 6))
sns.countplot(tt1['ProductCD'], ax=axes[0]).set_title('Fraudulent Transactions')
sns.countplot(tt0['ProductCD'], ax=axes[1]).set_title('Non-Fraudulent Transactions')
plt.tight_layout()
```



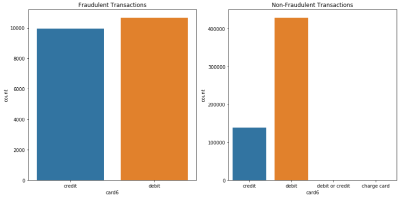
In [68]:

```
fig, axes = plt.subplots(1, 2,figsize=(12, 6))
sns.countplot(tt1['card4'], ax=axes[0]).set_title('Fraudulent Transactions')
sns.countplot(tt0['card4'], ax=axes[1]).set_title('Non-Fraudulent Transactions')
plt.tight_layout()
```



In [69]:

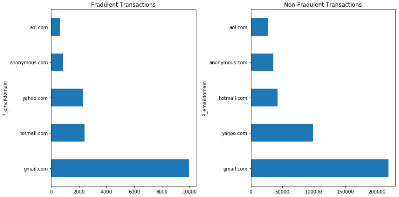
```
fig, axes = plt.subplots(1, 2,figsize=(12, 6))
sns.countplot(tt1['card6'], ax=axes[0]).set_title('Fraudulent Transactions')
sns.countplot(tt0['card6'], ax=axes[1]).set_title('Non-Fraudulent Transactions')
plt.tight_layout()
```



In [71]:

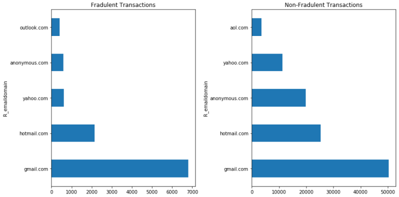
```
fig, axes = plt.subplots(1, 2,figsize=(12, 6))
#sns.countplot(tt1['P_emaildomain'], ax=axes[0]).set_title('Fraudulent Transactions')
#sns.countplot(tt0['P_emaildomain'], ax=axes[1]).set_title('Non-Fraudulent Transactions')

tt1.groupby('P_emaildomain')['P_emaildomain'].count().nlargest(5).plot(kind='barh', ax=axes[0], title='Fradulent Transactions')
tt0.groupby('P_emaildomain')['P_emaildomain'].count().nlargest(5).plot(kind='barh', ax=axes[1], title='Non-Fradulent Transactions')
plt.tight_layout()
```



In [72]:

```
fig, axes = plt.subplots(1, 2,figsize=(12, 6))
#sns.countplot(tt1['R_emaildomain'], ax=axes[0]).set_title('Fraudulent Transactions')
#sns.countplot(tt0['R_emaildomain'], ax=axes[1]).set_title('Non-Fraudulent Transactions')
tt1.groupby('R_emaildomain')['R_emaildomain'].count().nlargest(5).plot(kind='barh', ax=axes[0], title='Fradulent Transactions')
tt0.groupby('R_emaildomain')['R_emaildomain'].count().nlargest(5).plot(kind='barh', ax=axes[1], title='Non-Fradulent Transactions')
plt.tight_layout()
```



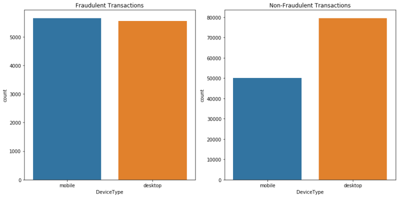
In [11]:

```
is_1 = train_merge['isFraud'] == 1
tt3 = train_merge[is_1]     #creating tt3 dataframe that has only those rows in which isFraud coloumn is == 1

is_0 = train_merge['isFraud'] == 0
tt4 = train_merge[is_0]   #creating tt4 dataframe that has only those rows in which isFraud coloumn is == 0
```

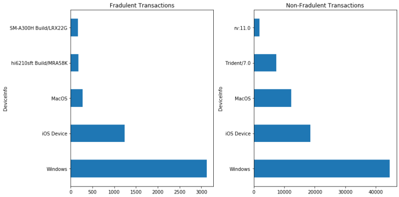
In [12]:

```
fig, axes = plt.subplots(1, 2,figsize=(12, 6))
sns.countplot(tt3['DeviceType'], ax=axes[0]).set_title('Fraudulent Transactions')
sns.countplot(tt4['DeviceType'], ax=axes[1]).set_title('Non-Fraudulent Transactions')
plt.tight_layout()
```



In [75]:

```
fig, axes = plt.subplots(1, 2,figsize=(12, 6))
#sns.countplot(tt3['DeviceInfo'], ax=axes[0]).set_title('Fraudulent Transactio
ns')
#sns.countplot(tt4['DeviceInfo'], ax=axes[1]).set_title('Non-Fraudulent Transa
ctions')
tt3.groupby('DeviceInfo')['DeviceInfo'].count().nlargest(5).plot(kind='barh',
ax=axes[0], title='Fradulent Transactions')
tt4.groupby('DeviceInfo')['DeviceInfo'].count().nlargest(5).plot(kind='barh',
ax=axes[1], title='Non-Fradulent Transactions')
plt.tight_layout()
```



Analysis for Q1:

- In Transaction Amt, for fradulent transactions we can observe that almost all of the amount are in the range of 0-500 while for non-fradulent transactions the transactions are not as skewed.
- In TransactionDT, from the distplot we can observe that both the fradulent and non-fradulent transactions are occuring at similar times.
- In ProductCD, for both fradulent & non fradulent transactions "W" has the highest frequency.
- In card4, for both fradulent & non fradulent transactions "VISA" card has the highest frequency.
- In card6, for fradulent transactions debit card has slightly higher frequency than credit card. But for non-fradulent transactions debit card has significantly much higher frequency than credit cards.
- In P_emaildomain, for both fradulent & non fradulent transactions "gmail" has the highest frequency than any other email domains.
- In R_emaildomain, for both fradulent & non fradulent transactions "gmail" has the highest frequency than any other email domains.
- In DeviceType, for fradulent transactions Mobile has slightly higher frequency than Desktop. but, for non- fradulent Desktop has significantly much higher frequency than mobile devices.
- In Device Info, for both fradulent & non fradulent transactions "Windows" has the highest frequency than any other DeviceInfo

Part 2 - Transaction Frequency

In [29]:

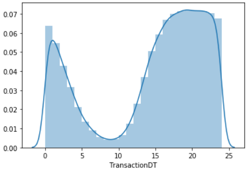
```
# TODO: code to generate the frequency graph
#train_transaction.addr2.mode()  #finding the most frequent "addr2"
mostfrequent = train_transaction['addr2'] == 87.0
mfc = train_transaction[mostfrequent]  #creating mostfrequentcountry data fram
e that has only those rows which has addr2== most frequent country
mfc.TransactionDT = (mfc.TransactionDT%86400)/3600
sns.distplot(mfc['TransactionDT'], bins =24)#.set_title('Non-Fraudulent Transa
ctions')
```

```
C:\Users\preet\Anaconda3\lib\site-packages\pandas\core\generic.py:5096: Set
tingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/
stable/indexing.html#indexing-view-versus-copy
  self[name] = value
```

Out[29]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x24597631da0>
```



Analysis for Q2:

From the addr2 column I found out that the most frequent country code in the data is **"87"**. Using that country code I created a new data frame called **"mfc"** using the main dataframe "train_transaction". In the "TransactionDT" column of the mfc I have converted the given time reference into hours my doing modulo 86400 and then dividing by 3600. After that I plotted a Dist plot for "TransactionDT". From the distribution we can observe that there is a dip in the no. of transactions for some hours of the day. We can deduce that mostly that dip is due to non-awake times.

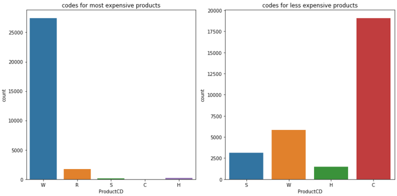
Part 3 - Product Code

In [15]:

```
# TODO: code to analyze prices for different product codes

costly = train_transaction.copy() #creating a new dataframe called costly
costly = costly.sort_values('TransactionAmt',ascending=False)
import math
res1 = costly.head(math.ceil(0.05*590540))  # res1 contains top 5 percent
res2 = costly.tail(math.ceil(0.05*590540))  # res2 least 5 percent

fig, axes = plt.subplots(1, 2,figsize=(12, 6))
sns.countplot(res1['ProductCD'], ax=axes[0]).set_title('codes for most expensive products')
sns.countplot(res2['ProductCD'], ax=axes[1]).set_title('codes for less expensive products')
plt.tight_layout()
```



Analysis for Q3

- Educated Guess:

```
a) "W" corresponds to the most expensive products.
b) "C" corresponds to the least expensive products.
```

- Procedure:- Create a dataframe "Costly" that has all the fields of "train_transaction" but in the costly data frame the "TransactionAmt" is in descending order. After that create two dataframes res1, res2 that contain the top 5 and least 5 percentile of rows respectively. After that I drew countplots for the most expensive and least expensive products.

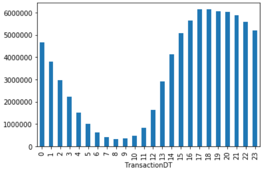
Part 4 - Correlation Coefficient

In [31]:

```
# TODO: code to calculate correlation coefficient
q5 = train_transaction.copy()
q5.TransactionDT = (q5.TransactionDT%86400)/3600
q5.TransactionDT = q5.TransactionDT.astype(int)
q5.groupby(['TransactionDT']).sum()['TransactionAmt'].plot(kind="bar")
```

Out[31]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2455489d898>
```



In [73]:

```
q5['TransactionDT'].corr(q5['TransactionAmt'])
```

Out[73]:

```
0.0445323560306529
```

In [32]:

```
d4 = q5.groupby(['TransactionDT'], as_index=False)['TransactionAmt'].sum()
d4['TransactionDT'].corr(d4['TransactionAmt'])
```

Out[32]:

```
0.6421174943084446
```

Analysis for Q4:

- Correlation Co-efficient : 0.64211 - if we group transaction amounts corresponding to the time.
- Correlatopn Co-efficient : 0.044 - if we find correlation for time and amounts individually.
- Procedure : Create a dataframe "q5" using the train_transaction DataFrame. Then in the q5 dataframe convert the time reference into hours using modulo 86400 and then dividing by 3600. after tha convert the float times into into using asType(int). Finally draw a bar graph by grouping TransactionDT and sum of the TransactionAmt.

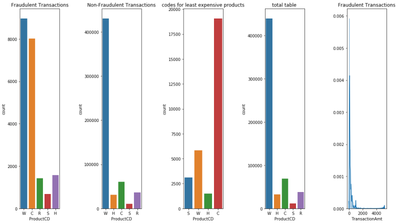
Part 5 - Interesting Plot

In [87]:

```
# TODO: code to generate the plot here.

fig, axes = plt.subplots(1, 5,figsize=(14, 8))
sns.countplot(tt1['ProductCD'], ax=axes[0]).set_title('Fraudulent Transactions')
sns.countplot(tt0['ProductCD'], ax=axes[1]).set_title('Non-Fraudulent Transactions')
#plt.tight_layout()


#fig, axes = plt.subplots(1, 2,figsize=(12, 6))
sns.countplot(res2['ProductCD'], ax=axes[2]).set_title('codes for least expensive products')
sns.countplot(train_transaction['ProductCD'], ax=axes[3]).set_title('total table ')
sns.distplot(tt1['TransactionAmt'], ax=axes[4]).set_title('Fraudulent Transactions')
plt.tight_layout()
```



Interesting insight that we can observe is that altough C has very less frequency count in the total table it has very high frequency count in the fradulent transactions data frame.And we also know that fradulent transactions are skewed toward low money range i.e from 0 to 1000 and from the third graph(least expensive products) above we see that "C" has the highest frequency count. So, we can deduce that C is having more number of fradulent transactions.

Part 6 - Prediction Model

In [6]:

```
# TODO: code for your final model
```

Read train transaction and train identity and left merge tham into a new dataframe "train_merge". Convert the time reference of the train merge into hours using modulo 86400 and them divide by 3600.

In [76]:

```
train_merge.TransactionDT = (train_merge.TransactionDT%86400)/3600
train_merge.TransactionDT = train_merge.TransactionDT.astype(int)
```

Filter only those rows from train_merge which I think are useful for the modelling.

In [41]:

```
train_merge = train_merge.filter(['TransactionID', 'DeviceType', 'DeviceInfo', 'TransactionDT', 'TransactionAmt', 'ProductCD', 'card4', 'card6', 'P_emaildomain', 'R_emaildomain', 'addr1', 'addr2', 'dist1', 'dist2'])
```

Impute the missing/nan/null vaues using median. For categorical data fill the missing/nan/null values with any random string

In [42]:

```
train_merge['DeviceType'] = train_merge['DeviceType'].fillna('not available')
train_merge['DeviceInfo'] = train_merge['DeviceInfo'].fillna('not available')
train_merge['card4'] = train_merge['card4'].fillna('not available')
train_merge['card6'] = train_merge['card6'].fillna('not available')
train_merge['P_emaildomain'] = train_merge['P_emaildomain'].fillna('not available')
train_merge['R_emaildomain'] = train_merge['R_emaildomain'].fillna('not available')
```

In [43]:

```
train_merge=train_merge.fillna(train_merge.median())
```

Now I did oneHotEncoding for DeviceType, ProductCD, card4, card6.

In [44]:

```
train_merge = pd.concat([train_merge,pd.get_dummies(train_merge['DeviceType'], prefix='DeviceType')],axis=1).drop(['DeviceType'],axis=1)
train_merge = pd.concat([train_merge,pd.get_dummies(train_merge['ProductCD'],prefix='ProductCD')],axis=1).drop(['ProductCD'],axis=1)
train_merge = pd.concat([train_merge,pd.get_dummies(train_merge['card4'],prefix='card4')],axis=1).drop(['card4'],axis=1)
train_merge = pd.concat([train_merge,pd.get_dummies(train_merge['card6'],prefix='card6')],axis=1).drop(['card6'],axis=1)
```

I did FeatureHashing for DeviceInfo, R_emaildomain, P_emaildomain.

In [45]:

```
fh = FeatureHasher(n_features=5, input_type='string')
sp = fh.fit_transform(train_merge['DeviceInfo'])
dev_0 = pd.DataFrame(sp.toarray(), columns=['DeviceInfo1', 'DeviceInfo2', 'DeviceInfo3', 'DeviceInfo4', 'DeviceInfo5'])
train_merge = pd.concat([train_merge, dev_0], axis=1)
```

In [46]:

```
fh = FeatureHasher(n_features=5, input_type='string')
sp = fh.fit_transform(train_merge['R_emaildomain'])
dev_1 = pd.DataFrame(sp.toarray(), columns=['R_emaildomain1', 'R_emaildomain2', 'R_emaildomain3', 'R_emaildomain4', 'R_emaildomain5'])
train_merge = pd.concat([train_merge, dev_1], axis=1)
```

In [47]:

```
fh = FeatureHasher(n_features=5, input_type='string')
sp = fh.fit_transform(train_merge['P_emaildomain'])
dev_2 = pd.DataFrame(sp.toarray(), columns=['P_emaildomain1', 'P_emaildomain2', 'P_emaildomain3', 'P_emaildomain4', 'P_emaildomain5'])
train_merge = pd.concat([train_merge, dev_2], axis=1)
```

In [49]:

```
train_merge_copy = train_merge.copy()
```

In [50]:

```
train_merge = train_merge.drop('TransactionID', 1)
train_merge = train_merge.drop('DeviceInfo', 1)
train_merge = train_merge.drop('P_emaildomain', 1)
train_merge = train_merge.drop('R_emaildomain', 1)
```

In [51]:

```
train_merge = train_merge.drop('card6_debit or credit', 1)
```

create a new data frame Y with only one colums 'isFraud'

In [52]:

```
Y = train_transaction.filter(['isFraud'])
Y.shape
```

Out[52]:

```
(590540, 1)
```

feature contains all the colums from train_merge, result contains columns from Y, splitting the train_merge into 70-30 ratio for training and testing, fitting the train variables into linear regression model, predicting the result on test variables.

In [53]:

```
train_merge = pd.concat([train_merge, Y], axis=1)

feature,result = train_merge.loc[:,train_merge.columns != 'isFraud'], Y.loc[:,'isFraud']
X_train, X_test, y_train, y_test = train_test_split(feature,result, test_size=0.3)

lm = LinearRegression().fit(X_train,y_train)

y_pred = lm.predict(X_test)
```

Calculating MAE, MSQ, RSME

In [80]:

```
mae = metrics.mean_absolute_error(y_test, y_pred)
msq = metrics.mean_squared_error(y_test, y_pred)
rmse = np.sqrt(metrics.mean_squared_error(y_test, y_pred))

print('Mean Absolute Error:'+str(mae))
print('Mean Squared Error:'+str(msq))
print('Root Mean Squared Error:'+str(rmse))
```

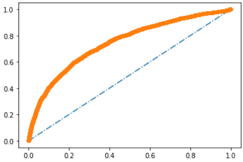
```
Mean Absolute Error:0.06566647349801807
Mean Squared Error:0.033323700491939605
Root Mean Squared Error:0.1825478033062562
```

In [82]:

```
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve

area_under_curve = roc_auc_score(y_test,y_pred)
print('area_under_curve: %.2f' % area_under_curve)
fpr, tpr, _ = roc_curve(y_test,y_pred)
plt.plot([0, 1], [0, 1], linestyle='dashdot')
plt.plot(fpr, tpr, marker='*')
plt.show()
```

```
area_under_curve: 0.74
```



Read test transaction and test identity and left merge tham into a new dataframe "test_merge". Convert the time reference of the train merge into hours using modulo 86400 and them divide by 3600. Filter only those rows from train_merge which I think are useful for the modelling.

In [55]:

```python
test_transaction = pd.read_csv(r'C:\Users\preet\Desktop\ieee-fraud-detection\test_transaction.csv')
test_identity = pd.read_csv(r'C:\Users\preet\Desktop\ieee-fraud-detection\test_identity.csv')

test_merge = pd.merge(test_transaction, test_identity, on='TransactionID', how='left')

test_merge.TransactionDT = (test_merge.TransactionDT%86400)/3600
test_merge.TransactionDT = test_merge.TransactionDT.astype(int)

test_merge = test_merge.filter(['TransactionID', 'DeviceType', 'DeviceInfo', 'TransactionDT', 'TransactionAmt', 'ProductCD', 'card4', 'card6', 'P_emaildomain', 'R_emaildomain', 'addr1', 'addr2', 'dist1', 'dist2'])
```

Impute the missing/nan/null vaues using median. For categorical data fill the missing/nan/null values with any random string

In [56]:

```python
test_merge['DeviceType'] = test_merge['DeviceType'].fillna('not available')
test_merge['DeviceInfo'] = test_merge['DeviceInfo'].fillna('not available')
test_merge['card4'] = test_merge['card4'].fillna('not available')
test_merge['card6'] = test_merge['card6'].fillna('not available')
test_merge['P_emaildomain'] = test_merge['P_emaildomain'].fillna('not available')
test_merge['R_emaildomain'] = test_merge['R_emaildomain'].fillna('not available')
```

Now I did oneHotEncoding for DeviceType, ProductCD, card4, card6.

In [57]:

```python
test_merge = test_merge.fillna(test_merge.median())

test_merge = pd.concat([test_merge,pd.get_dummies(test_merge['DeviceType'],prefix='DeviceType')],axis=1).drop(['DeviceType'],axis=1)
test_merge = pd.concat([test_merge,pd.get_dummies(test_merge['ProductCD'],prefix='ProductCD')],axis=1).drop(['ProductCD'],axis=1)
test_merge = pd.concat([test_merge,pd.get_dummies(test_merge['card4'],prefix='card4')],axis=1).drop(['card4'],axis=1)
test_merge = pd.concat([test_merge,pd.get_dummies(test_merge['card6'],prefix='card6')],axis=1).drop(['card6'],axis=1)
```

I did FeatureHashing for DeviceInfo, R_emaildomain, P_emaildomain.

In [58]:

```python
fh = FeatureHasher(n_features=5, input_type='string')
sp = fh.fit_transform(test_merge['DeviceInfo'])
dev_3 = pd.DataFrame(sp.toarray(), columns=['DeviceInfo1', 'DeviceInfo2', 'DeviceInfo3', 'DeviceInfo4', 'DeviceInfo5'])
test_merge = pd.concat([test_merge, dev_3], axis=1)

fh = FeatureHasher(n_features=5, input_type='string')
sp = fh.fit_transform(test_merge['R_emaildomain'])
dev_4 = pd.DataFrame(sp.toarray(), columns=['R_emaildomain1', 'R_emaildomain2', 'R_emaildomain3', 'R_emaildomain4', 'R_emaildomain5'])
test_merge = pd.concat([test_merge, dev_4], axis=1)

fh = FeatureHasher(n_features=5, input_type='string')
sp = fh.fit_transform(test_merge['P_emaildomain'])
dev_5 = pd.DataFrame(sp.toarray(), columns=['P_emaildomain1', 'P_emaildomain2', 'P_emaildomain3', 'P_emaildomain4', 'P_emaildomain5'])
test_merge = pd.concat([test_merge, dev_5], axis=1)
```

predicting on test_merge and storing the result in res_pred and then creating a data frame with 2 colums in which the first one is TransactionId and the second one is res_pred. This data frame is the final output that is uploaded as a CSV file on KAGGLE!

In [59]:

```python
test_merge_copy = test_merge.copy()

test_merge = test_merge.drop('TransactionID', 1)
test_merge = test_merge.drop('DeviceInfo', 1)
test_merge = test_merge.drop('P_emaildomain', 1)
test_merge = test_merge.drop('R_emaildomain', 1)

res_pred = lm.predict(test_merge)

res = pd.DataFrame()
res['TransactionID'] = test_merge_copy['TransactionID']
res['isFraud'] = res_pred
```

In [61]:

```python
res.to_csv('LinearRegression.csv', index=False)
```

Accuracy of the model

- Accuracy of the model(Linear Regression):- 0.74
- kaggle Score : 0.8023

Part 7 - Final Result

Report the rank, score, number of entries, for your highest rank. Include a snapshot of your best score on the leaderboard as confirmation. Be sure to provide a link to your Kaggle profile. Make sure to include a screenshot of your ranking. Make sure your profile includes your face and affiliation with SBU.

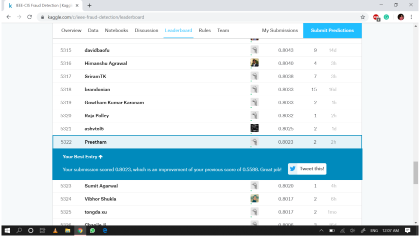
Kaggle Link: https://www.kaggle.com/preetham17

Highest Rank: 5322

Score: 0.8023

Number of entries: 2

INCLUDE IMAGE OF YOUR KAGGLE RANKING



5315	davidbaofu	0.8043	9	14d
5316	Himanshu Agrawal	0.8040	4	3h
5317	SriramTK	0.8038	7	3h
5318	brandonian	0.8033	15	16d
5319	Gowtham Kumar Karanam	0.8033	2	1h
5320	Raja Palley	0.8032	1	2h
5321	ashvtol5	0.8025	2	1d
5322	Preetham	0.8023	2	2h

Your Best Entry ↑

Your submission scored 0.8023, which is an improvement of your previous score of 0.5588. Great job!

5323	Sumit Agarwal	0.8020	1	4h
5324	Vibhor Shukla	0.8017	2	6h
5325	tongda xu	0.8017	2	1mo