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 coc
answer the questions. We also ask that code be commented to make it easier to follow.

```
#Drive mounted for reading files.
from google.colab import drive
drive.mount('/content/drive')
```

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn import preprocessing
from scipy.stats import spearmanr
from scipy.stats import pearsonr
```

```
#Reading files with pandas
train_transaction = pd.read_csv("/content/drive/My Drive/train_transaction.csv",index_col= 'Transact
train_identity = pd.read_csv("/content/drive/My Drive/train_identity.csv",index_col= 'TransactionID
test_transaction = pd.read_csv("/content/drive/My Drive/test_transaction.csv",index_col= 'Transactio
test_identity = pd.read_csv("/content/drive/My Drive/test_identity.csv",index_col= 'TransactionID')

print("File Reading Successfull!!")
```

```
#Checking how many transactions have associated identity information.

associated_train_data = np.sum(train_transaction.index.isin(train_identity.index.unique()))
associated_test_data = np.sum(test_transaction.index.isin(test_identity.index.unique()))

#Percentage of associated transaction-identity information.
train_association = associated_train_data/len(train_transaction.index)*100
test_association = associated_test_data/len(test_transaction.index)*100

print(train_association,'%','have associated identity information in training data')
print(test_association,'%','have associated identity information in testing data')
```

```
#Merging transaction
#Merging with how = "left", guarantees that there are still 590540 training records and 506691 testi
#even though not every transaction has associated identity information.
train_data = train_transaction.merge(train_identity, how = "left",left_index = True, right_index = 
test_data = test_transaction.merge(test_identity, how = "left", left_index = True, right_index = Tru


print(train_data.info())
print(test_data.info())

#For the next few cells, I will be cleaning and transforming the data.
```

```
#Before we get rid of the NaN values, let's first find the percentage of missing values in each colu
train_data_missing = train_data.isna()
percentage_miss = train_data_missing.sum()/len(train_data)*100

print(percentage_miss)
#Now we identify which columns have more than 50% of data missing. i.e. NaN values
percentage_miss_filter = percentage_miss > 50
#print(percentage_miss_filter)

#We will use this info later to filter the features for our prediction model.
```

```
#Replacing NaN values with the most frequent value in the column.
train_data = train_data.fillna(train_data.mode().iloc[0])
test_data = test_data.fillna(test_data.mode().iloc[0])
print(train_data.head())
print(test_data.head())
```

```
#These are the features we will consider in our prediction model
train_data_final = train_data.filter(['TransactionID','TransactionDT','TransactionAmt','ProductCD',
                   'addr1','addr2','card1','card2','card3','card4','card5','card6','P_emaildomain',
                   'C7','C8','C9','C10','C11','C12','C13','C14','isFraud'])

test_data_final = test_data.filter(['TransactionID','TransactionDT','TransactionAmt','ProductCD','De
                   'addr1','addr2','card1','card2','card3','card4','card5','card6','P_emaildomain',
                   'C7','C8','C9','C10','C11','C12','C13','C14'])
```

```
#Separting the features and the labels
features_train = train_data_final.drop('isFraud', axis = 1)
label_train = train_data_final['isFraud']
features_test = test_data_final.copy()
```

```
#Mapping categorical data to numeric values
for feature in features_train:
  if features_train[feature].dtype == 'object' or features_test[feature].dtype == 'object':
    le = preprocessing.LabelEncoder()
    le.fit(list(features_train[feature].values) + list(features_test[feature].values))
    features_train[feature] = le.transform(list(features_train[feature].values))
    features_test[feature] = le.transform(list(features_test[feature].values))
```

```
#Filtering data for parts 1 to 5.

train_data_filtered = train_data.filter(['TransactionID','DeviceType','DeviceInfo','TransactionDT',
                    'addr1','addr2','card4','card6','dist1','dist2','P_emaildomain','R_emaildomain',
```

```
train_data_filtered.head()
```

Part 1 - Fraudulent vs Non-Fraudulent Transaction

```
# TODO: code and runtime results

#Separating the fraudulent and Non-fraudulent transactions
train_transaction_fraud = train_data_filtered[train_data_filtered['isFraud'] == 1]
train_transaction_notFraud = train_data_filtered[train_data_filtered['isFraud'] == 0]
```

```
#
figure = plt.figure(figsize=(5,5))
plt.xlim(70,78)
plt.ylim(0,1000)
plt.hist(train_transaction_fraud['TransactionDT']/86400, bins = 1800, alpha = 1, label = 'Fraud', co
plt.hist(train_transaction_notFraud['TransactionDT']/86400, bins = 1800, alpha = 0.2, label = 'Not F
plt.legend(loc='upper right')
plt.xlabel("Every second")
plt.ylabel("No of Transactions")
plt.show()
```

```
#
plt.hist(train_transaction_fraud['TransactionAmt'], alpha = 1, label = 'Fraud', color = 'red')
plt.hist(train_transaction_notFraud['TransactionAmt'], alpha = 0.4, label = 'Not Fraud', color = 'ye
plt.legend(loc='upper right')
plt.xlabel("Amount")
plt.ylabel("No of Transactions")
plt.xlim(0,3500,500)
plt.show()
```

```
#Checking which type of card saw highest number of fraudulent transactions.
sns.set(style="whitegrid")
figure = plt.figure(figsize=(8,8))
sns.set_color_codes("pastel")
sns.barplot(x=train_data_filtered['card4'],y=train_data_filtered['isFraud'],data=train_data_filtere
```

```
#Checking which type of operating system saw highest number of fraudulent transactions.
sns.set(style="whitegrid")
figure = plt.figure(figsize=(8,8))
sns.set_color_codes("pastel")
list = train_transaction_fraud['DeviceInfo'].value_counts().nlargest(20)
sns.barplot(x=list.values,y=list.index,data=train_transaction_fraud,label="Fraud or not",color = "g'
#Checking whic type of device saw highest number of fraudulent transactions.
sns.set(style="whitegrid")
figure = plt.figure(figsize=(8,8))
sns.set_color_codes("pastel")
sns.barplot(x=train_data_filtered['DeviceType'],y=train_data_filtered['isFraud'],data=train_data_fi
```

```
sns.pairplot(train_data_filtered)
```

```
fig, ax = plt.subplots(2, 2, figsize = (10,10))
```

```
time = train_transaction_fraud['TransactionDT']
time2 = train_transaction_notFraud['TransactionDT']

log_time = np.log(time.values/86400)
log_time2 = np.log(time2.values/86400)

sns.distplot(time2.values/86400, ax = ax[0,0], color = 'g')
ax[0,0].set_title('Distribution of TransactionDT', fontsize=14)
ax[0,0].set_xlim([min(time2.values/86400), max(time2.values/86400)])

sns.distplot(time.values/86400, ax=ax[0,1], color='r')
ax[0,1].set_title('Distribution of Fraud TransactionDT', fontsize=14)
ax[0,1].set_xlim([min(time.values/86400), max(time.values/86400)])

sns.distplot(log_time2, ax=ax[1,0], color='b')
ax[1,0].set_title('Distribution of LOG TransactionDT', fontsize=14)
ax[1,0].set_xlim([min(log_time2), max(log_time2)])

sns.distplot(log_time, ax=ax[1,1], color='r')
ax[1,1].set_title('Distribution of Fraud LOG TransactionDT', fontsize=14)
ax[1,1].set_xlim([min(log_time), max(log_time)])

plt.show()
```

Write your answer here

Part 2 - Transaction Frequency

```
# TODO: code to generate the frequency graph

def make_hour_feature(df, tname='TransactionDT'):
    """
    Creates an hour of the day feature, encoded as 0-23.

    Parameters:
    -----------
    df : pd.DataFrame
        df to manipulate.
    tname : str
        Name of the time column in df.
    """
    hours = df[tname] / (3600)
    encoded_hours = np.floor(hours) % 24
    return encoded_hours

mode = np.max(train_data_filtered.addr2.mode().iloc[0])
data_addr = train_data_filtered.loc[train_data_filtered['addr2'] == mode]
data_addr['hours'] = make_hour_feature(data_addr)
data_addr['hours'].hist(bins = 24 , xrot = 30, color = "skyblue", lw = 1)
plt.title("Transaction Frequency per hour")
plt.ylabel("No. of transactions")
plt.xlabel("Hour")
```

Answer: From the 24 hours in a day (shown as 0-23 in the graph above), one can clearly see the dip in the graph of r
represents the sleeping hours of the vast majority. An inverse bell curve graph helps determine the waking v/s sleep
which is 87.0

Part 3 - Product Code

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

maxCost = 0
minCost = 100
for i in train_data_filtered.ProductCD.unique():
  print(i)
  print("Count: ",train_data_filtered[train_data_filtered.ProductCD == i].TransactionAmt.count())
  print("Mean: ",train_data_filtered[train_data_filtered.ProductCD == i].TransactionAmt.mean())
  count = train_data_filtered[train_data_filtered.ProductCD == i].TransactionAmt.count()
  median = train_data_filtered[train_data_filtered.ProductCD == i].TransactionAmt.median()
  mean = train_data_filtered[train_data_filtered.ProductCD == i].TransactionAmt.mean()
  std_deviation = train_data_filtered[train_data_filtered.ProductCD == i].TransactionAmt.std()
  if mean > maxCost:
    maxCost = mean
    product = i

print("The most expensive product is",product,"and it's average cost is",maxCost)

for j in train_data_filtered.ProductCD.unique():
    mean = train_data_filtered[train_data_filtered.ProductCD == j].TransactionAmt.mean()
    if mean < minCost:
      minCost = mean
      product = j

print("The cheapest product is",product,"and it's average cost is",minCost)

figure = plt.figure(figsize=(8,8))
#ax = sns.boxplot( x=train_data_filtered['ProductCD'], y=train_data_filtered['TransactionAmt'], hue
ax = sns.boxplot( x=train_data_filtered['ProductCD'], y=np.log(train_data_filtered['TransactionAmt']
ax.legend(frameon=False, loc='upper right', ncol=1)
```

Answer:

The average cost of each product is calculated by the total cost of that product divided by the total number of prod
have found that Product **R** is the **most expensive** product with an average cost of **168.30** and Product **C** is the **chea**

The graph above shows product W as the most expensive, however, that is just because it has more number of tran
this, I have used mean average cost as the metric.

Part 4 - Correlation Coefficient

```
# TODO: code to calculate correlation coefficient
from datetime import datetime
train_data_filtered['TransactionTime'] = pd.to_datetime(train_data_filtered['TransactionDT'],unit =


#Plot to visualize date time v/s number of transactions
figure = plt.figure(figsize=(8,4))
vals = plt.hist(train_data_filtered['TransactionDT'] / (3600*24), bins=1800)
plt.xlim(70, 78)
plt.xlabel('Days')
plt.ylabel('Number of transactions')
plt.ylim(0,1000)
```

```
#By grouping them in hours and then summing the amount of transactions per hour gives below
train_data_filtered['hours'] = make_hour_feature(train_data_filtered)
dataGroupedByHours = train_data_filtered.groupby('hours')
plt.scatter(dataGroupedByHours['hours'].unique(),dataGroupedByHours.TransactionAmt.sum())
```

```
print("Spearman Correlation coeffecient with sum of transaction amounts: ",spearmanr(dataGroupedByH
```

Answer: The correlation coeffecient between hours of a day and the sum of transaction amount per hour is 0.63. Tl
how hours (waking v/s sleeping) can affect the frequency of large transactions made.

Part 5 - Interesting Plot

```
plt.figure(figsize=(10,8))
sns.heatmap(train_data_final.corr())
```

Answer:

The above heatmap helps us understand the correlation between the various columns of our dataset. However, a qu
them depict very low correlation with each other, **the attribute 'card3' is highly correlated with most of the other att**
with our target attribute 'isFraud'. Some interesting insights can be found with this information and our model pred

```
#No overlap between train and test data dates.
train_data['TransactionDT'].plot(kind = "hist", label = "Train", bins = 100)
test_data['TransactionDT'].plot(kind = "hist", label = "Test", bins = 100)
plt.legend()
```

Answer:

The above graph shows that there is no overlap between the training and testing data. This signifies that the transa
done during the same time interval but rather some time intervals apart.

Part 6 - Prediction Model

```
#Definig parameters for my model.
x_train = features_train
y_train = label_train
x_test = features_test
from sklearn.model_selection import train_test_split
data_train, data_test, target_train, target_test = train_test_split(x_train,y_train, test_size = 0.
```

```
# TODO: code for your final model

#Random Forests Classifier
from sklearn.ensemble import RandomForestClassifier

clf = RandomForestClassifier()
clf.fit(x_train, y_train)
p_prob = clf.predict_proba(x_test)
preds = clf.predict(x_test)
sub = pd.read_csv('/content/drive/My Drive/sample_submission.csv', index_col='TransactionID')
sub['isFraud'] = preds
sub.to_csv('myPrediction17.csv')


#Measuring the accuracy of my model.
from sklearn.metrics import accuracy_score
#Using the object of Random Forests classifier
#train the algorithm on training data and predict using the testing data
pred = clf.fit(data_train, target_train).predict(data_test)
#print(pred.tolist())
#print the accuracy score of the model
accuracy_score = accuracy_score(target_test, pred, normalize = True)

print("Random Forests - Percentage accuracy: ",accuracy_score * 100)

#Plotting the probabilities
pd.DataFrame(p_prob).hist()
```

```
#Model2
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression()
logreg.fit(x_train, y_train)
p_prob = logreg.predict_proba(x_test)
y_pred = logreg.predict(x_test)
sub = pd.read_csv('/content/drive/My Drive/sample_submission.csv', index_col='TransactionID')
sub['isFraud'] = y_pred
sub.to_csv('myPrediction18.csv')


#Measuring the accuracy of my model.
from sklearn.metrics import accuracy_score
#Using the object of Random Forests classifier
#train the algorithm on training data and predict using the testing data
pred2 = logreg.fit(data_train, target_train).predict(data_test)
#print(pred.tolist())
#print the accuracy score of the model
accuracy_score = accuracy_score(target_test, pred2, normalize = True)

print("Logistic Regression - Percentage accuracy: ",accuracy_score * 100)



#Plotting the probabilities
pd.DataFrame(p_prob).hist()
```

Answer:

I have worked on two models for predicting whether a transaction is fraudulent or not - Random forests classifier ar
forests one works better as it uses a modified tree learning algorithm, where in at each step of the learning process
it tries out different combinations of the feature sets and finally ensembles the ones which give out the most accur

Accuracy of this model: 98.45%

Kaggle score: 0.708

Part 7 - Final Result

Report the rank, score, number of entries, for your highest rank. Include a snapshot of your best score on the leader
to your Kaggle profile. Make sure to include a screenshot of your ranking. Make sure your profile includes your face

Kaggle Link: PayalMehta_Kaggle_Link

Highest Rank: 5532

Score: 0.7089

Number of entries: 19

@title kaggle screenshot

%%html