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 [0]:
```
#Drive mounted for reading files.
from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client
_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&
redirect_uri=urn%3Aietf%3Awg%3Aoauth%3A2.0%3Aoob&scope=email%20https%3A%2F%2F
www.googleapis.com%2Fauth%2Fdocs.test%20https%3A%2F%2Fwww.googleapis.com%2Fau
th%2Fdrive%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive.photos.readonly%
20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fpeopleapi.readonly&response_type=
code

Enter your authorization code:
··········
Mounted at /content/drive
```

In [0]:
```
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
```

In [0]:
```
#Reading files with pandas
train_transaction = pd.read_csv("/content/drive/My Drive/train_transaction.cs
v",index_col= 'TransactionID')
train_identity = pd.read_csv("/content/drive/My Drive/train_identity.csv",inde
x_col= 'TransactionID')
test_transaction = pd.read_csv("/content/drive/My Drive/test_transaction.csv",
index_col= 'TransactionID')
test_identity = pd.read_csv("/content/drive/My Drive/test_identity.csv",index_
col= 'TransactionID')

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

```
File Reading Successfull!!
```

In [0]:
```
#Checking how many transactions have associated identity information.

associated_train_data = np.sum(train_transaction.index.isin(train_identity.ind
ex.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 da
ta')
```

```
24.42391709283029 % have associated identity information in training data
28.006615471756945 % have associated identity information in testing data
```

In [0]:
```
#Merging transaction
#Merging with how = "left", guarantees that there are still 590540 training re
cords and 506691 testing records,
#even though not every transaction has associated identity information.
train_data = train_transaction.merge(train_identity, how = "left",left_index =
True, right_index = True)
test_data = test_transaction.merge(test_identity, how = "left", left_index = T
rue, right_index = True)


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

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

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 590540 entries, 2987000 to 3577539
Columns: 433 entries, isFraud to DeviceInfo
dtypes: float64(385), int64(17), object(31)
memory usage: 1.9+ GB
None
<class 'pandas.core.frame.DataFrame'>
Int64Index: 506691 entries, 3663549 to 4170239
Columns: 432 entries, TransactionDT to DeviceInfo
dtypes: float64(399), int64(2), object(31)
memory usage: 1.7+ GB
None
```

In [0]:
```
#Before we get rid of the NaN values, let's first find the percentage of missi
ng values in each column.
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 va
lues
percentage_miss_filter = percentage_miss > 50
#print(percentage_miss_filter)

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

```
isFraud            0.000000
TransactionDT      0.000000
TransactionAmt     0.000000
ProductCD          0.000000
card1              0.000000
card2              1.512683
card3              0.265012
card4              0.267044
card5              0.721204
card6              0.266028
addr1             11.126427
addr2             11.126427
dist1             59.652352
dist2             93.628374
P_emaildomain     15.994852
R_emaildomain     76.751617
C1                 0.000000
C2                 0.000000
C3                 0.000000
C4                 0.000000
C5                 0.000000
C6                 0.000000
C7                 0.000000
C8                 0.000000
C9                 0.000000
C10                0.000000
C11                0.000000
C12                0.000000
C13                0.000000
C14                0.000000
                    ...
id_11             76.127273
id_12             75.576083
id_13             78.440072
id_14             86.445626
id_15             76.126088
id_16             78.098012
id_17             76.399736
id_18             92.360721
id_19             76.408372
id_20             76.418024
id_21             99.126393
id_22             99.124699
id_23             99.124699
id_24             99.196159
id_25             99.130965
id_26             99.125715
id_27             99.124699
id_28             76.127273
id_29             76.127273
id_30             86.865411
id_31             76.245132
id_32             86.861855
id_33             87.589494
id_34             86.824771
id_35             76.126088
id_36             76.126088
```

```
id_37            76.126088
id_38            76.126088
DeviceType       76.155722
DeviceInfo       79.905510
Length: 433, dtype: float64
```

In [0]:
```
#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())
```

```
               isFraud  ...                      DeviceInfo
TransactionID           ...
2987000              0  ...                         Windows
2987001              0  ...                         Windows
2987002              0  ...                         Windows
2987003              0  ...                         Windows
2987004              0  ...  SAMSUNG SM-G892A Build/NRD90M

[5 rows x 433 columns]
               TransactionDT  TransactionAmt  ... DeviceType  DeviceInfo
TransactionID                                 ...
3663549             18403224           31.95  ...    desktop     Windows
3663550             18403263           49.00  ...    desktop     Windows
3663551             18403310          171.00  ...    desktop     Windows
3663552             18403310          284.95  ...    desktop     Windows
3663553             18403317           67.95  ...    desktop     Windows

[5 rows x 432 columns]
```

In [0]:
```
#These are the features we will consider in our prediction model
train_data_final = train_data.filter(['TransactionID','TransactionDT','Transac
tionAmt','ProductCD','DeviceType','DeviceInfo'
                      'addr1','addr2','card1','card2','card3','card4','card5','ca
rd6','P_emaildomain','C1','C2','C3','C4','C5','C6',
                      'C7','C8','C9','C10','C11','C12','C13','C14','isFraud'])

test_data_final = test_data.filter(['TransactionID','TransactionDT','Transacti
onAmt','ProductCD','DeviceType','DeviceInfo'
                      'addr1','addr2','card1','card2','card3','card4','card5','ca
rd6','P_emaildomain','C1','C2','C3','C4','C5','C6',
                      'C7','C8','C9','C10','C11','C12','C13','C14'])
```

In [0]:
```
#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()
```

In [0]:
```
#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))
```

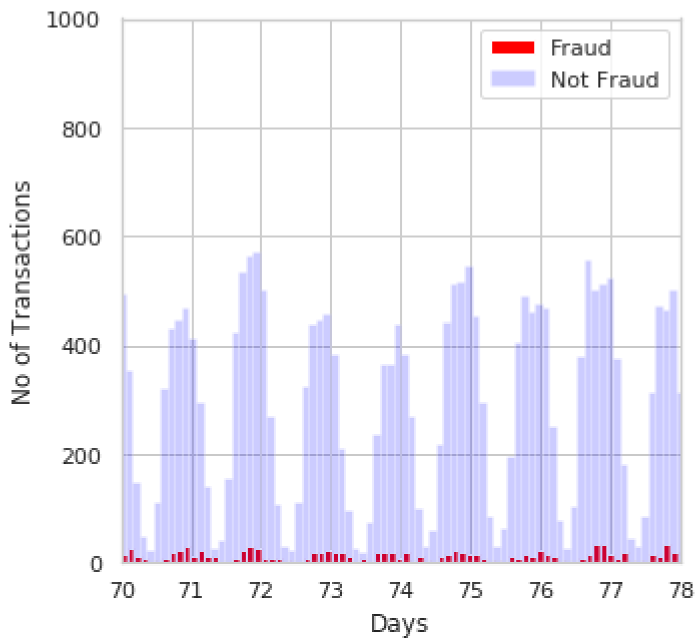
In [0]:
```
#Filtering data for parts 1 to 5.

train_data_filtered = train_data.filter(['TransactionID','DeviceType','DeviceI
nfo','TransactionDT','TransactionAmt','ProductCD',
                    'addr1','addr2','card4','card6','dist1','dist2','P_emaildo
main','R_emaildomain','isFraud'])
```

In [0]:
```
train_data_filtered.head()
```

Part 1 - Fraudulent vs Non-Fraudulent Transaction

In [0]:
```
# 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]
```

In [79]:
```
#
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', color = 'red')
plt.hist(train_transaction_notFraud['TransactionDT']/86400, bins = 1800, alpha
= 0.2, label = 'Not Fraud', color = 'blue')
plt.legend(loc='upper right')
plt.xlabel("Days")
plt.ylabel("No of Transactions")
plt.show()
```

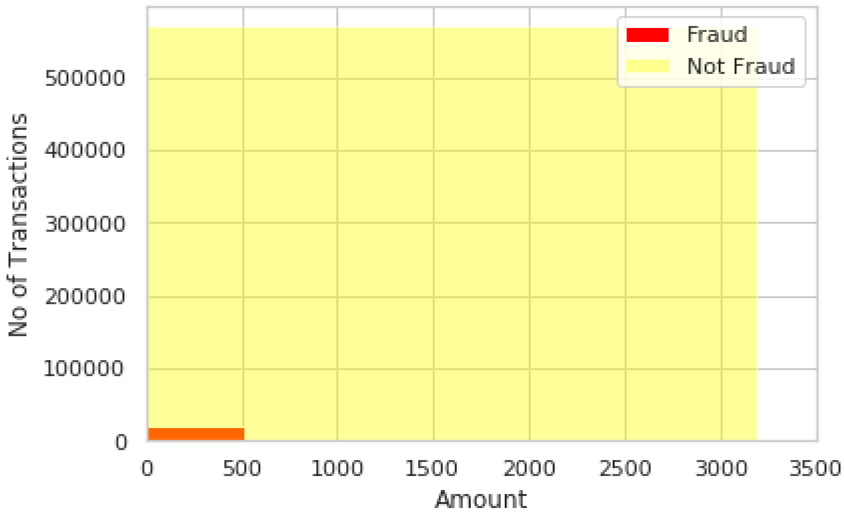


Answer:

In the above graph, I have plotted the number of fraudulent (red) and non-fraudulent (purple) transactions against day of the week.

There is no significant correlation between the day of the week and whether a transaction is fraudulent or not.

In [0]:
```
#
plt.hist(train_transaction_fraud['TransactionAmt'], alpha = 1, label = 'Fraud'
, color = 'red')
plt.hist(train_transaction_notFraud['TransactionAmt'], alpha = 0.4, label = 'N
ot Fraud', color = 'yellow')
plt.legend(loc='upper right')
plt.xlabel("Amount")
plt.ylabel("No of Transactions")
plt.xlim(0,3500,500)
plt.show()
```



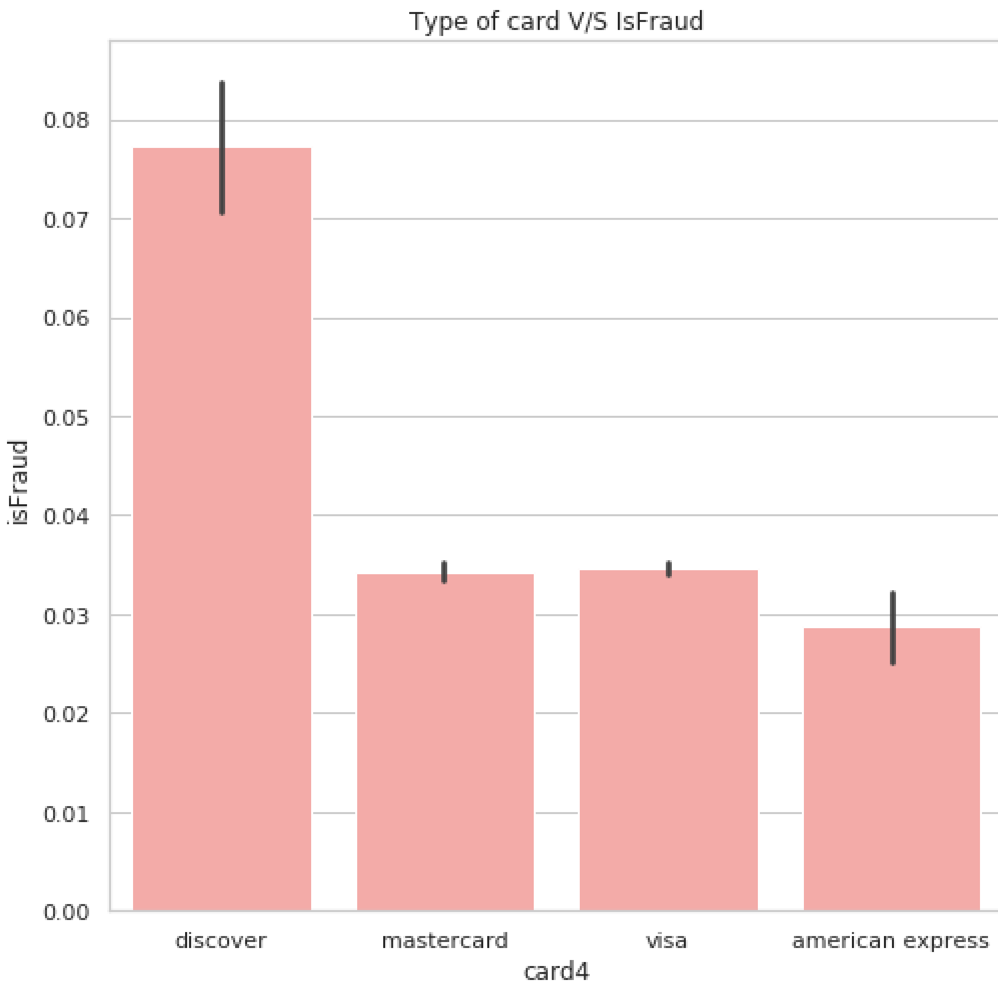
Answer:

In the above graph, I have plotted the Transaction amounts, number of transactions and whether they are fraudulent or not.

As we can see, the fradulent (red) transactions are mostly of lower amounts between 0-500.

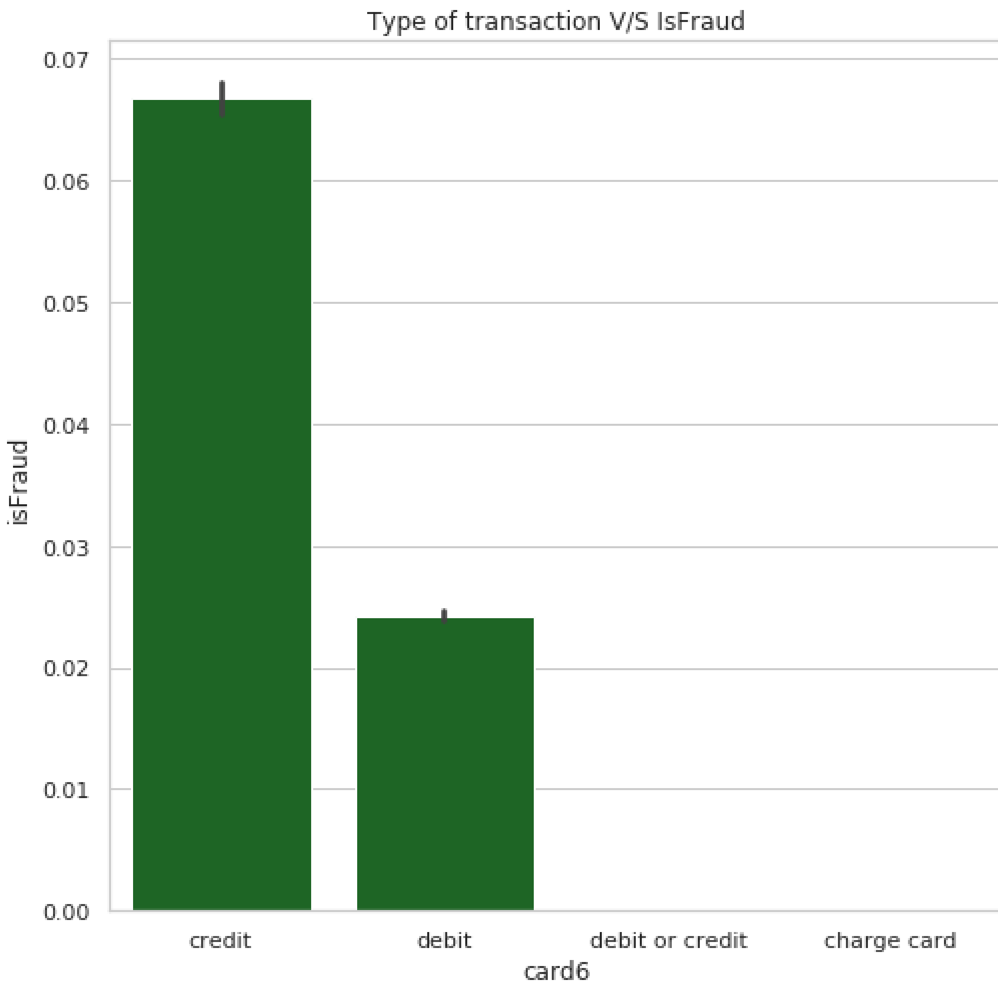
In [82]:
```
#Checking which type of card saw highest number of fraudulent transactions.
sns.set(style="whitegrid")
figure = plt.figure(figsize=(8,8))
sns.set_color_codes("pastel")
sns.barplot(x=train_data_filtered['card4'],y=train_data_filtered['isFraud'],
            data=train_data_filtered,label="Fraud or not",color = "r").set_tit
le("Type of card V/S IsFraud")
```

Out[82]:
```
Text(0.5, 1.0, 'Type of card V/S IsFraud')
```



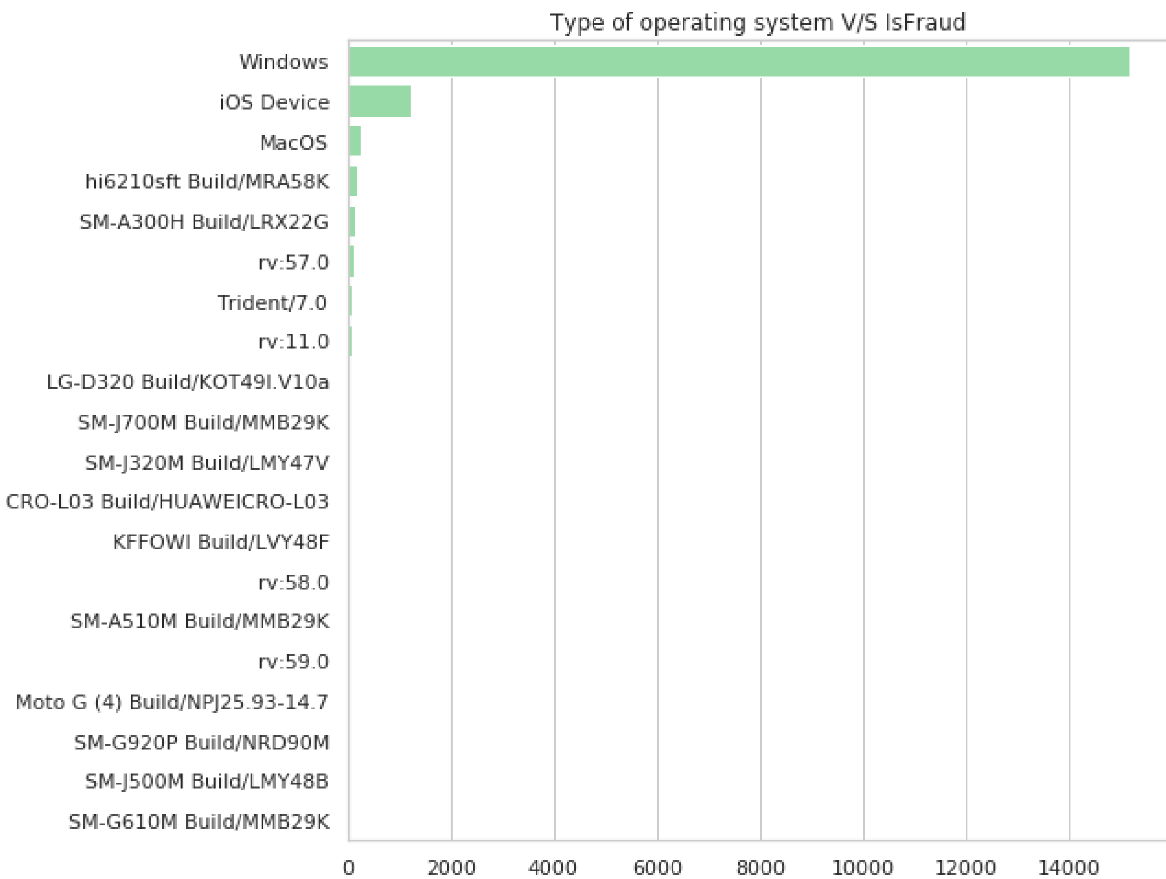
In [66]:
```
#Checking which type of card saw highest number of fraudulent transactions.
sns.set(style="whitegrid")
figure = plt.figure(figsize=(8,8))
sns.set_color_codes("dark")
sns.barplot(x=train_data_filtered['card6'],y=train_data_filtered['isFraud'],
            data=train_data_filtered,label="Fraud or not",color = "g").set_tit
le("Type of transaction V/S IsFraud")
```

Out[66]: Text(0.5, 1.0, 'Type of transaction V/S IsFraud')



In [85]:
```
#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").set_title("Type of operating system V/S IsFraud")
```

Out[85]: `Text(0.5, 1.0, 'Type of operating system V/S IsFraud')`



Answer:

The graph of "Type of Operating system v/s isFraud" helps us identify what number of transactions done on a particular operating systems were fraudulent.

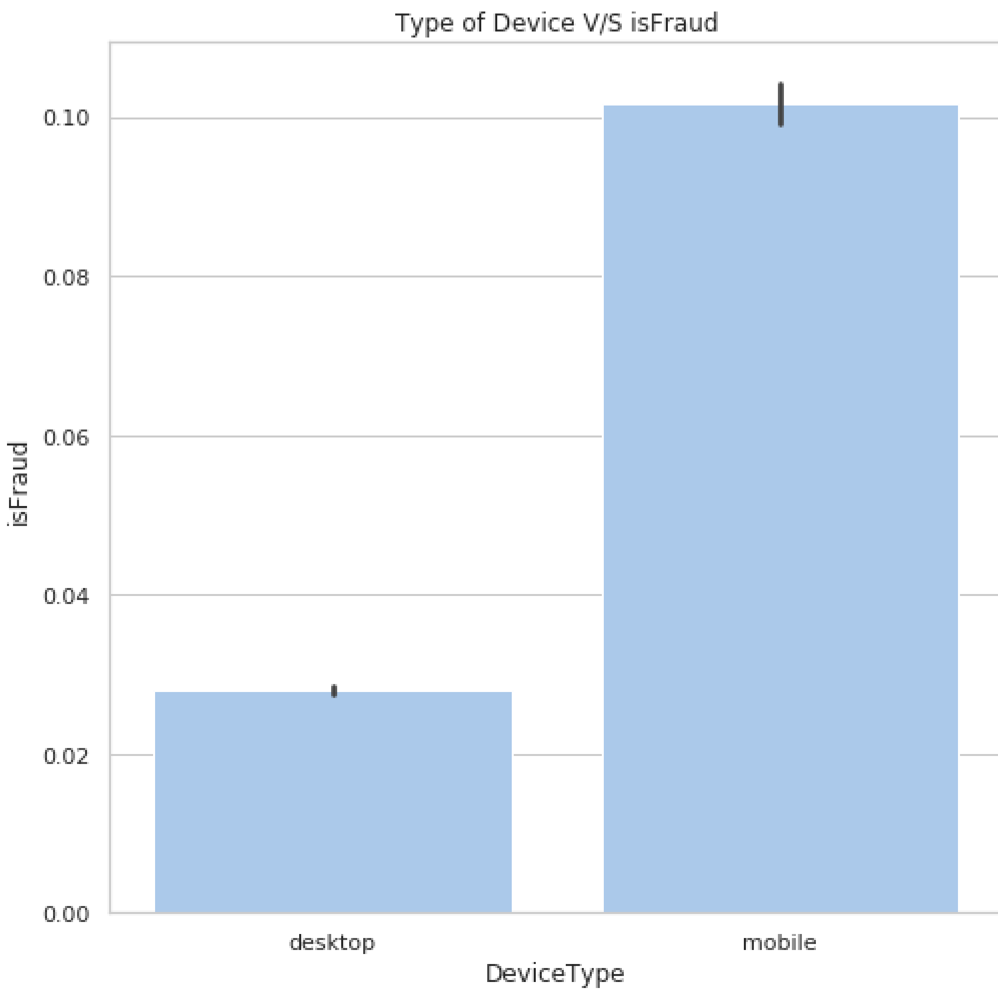
E.g.:-

Windows: >14000

MacOS: < 100

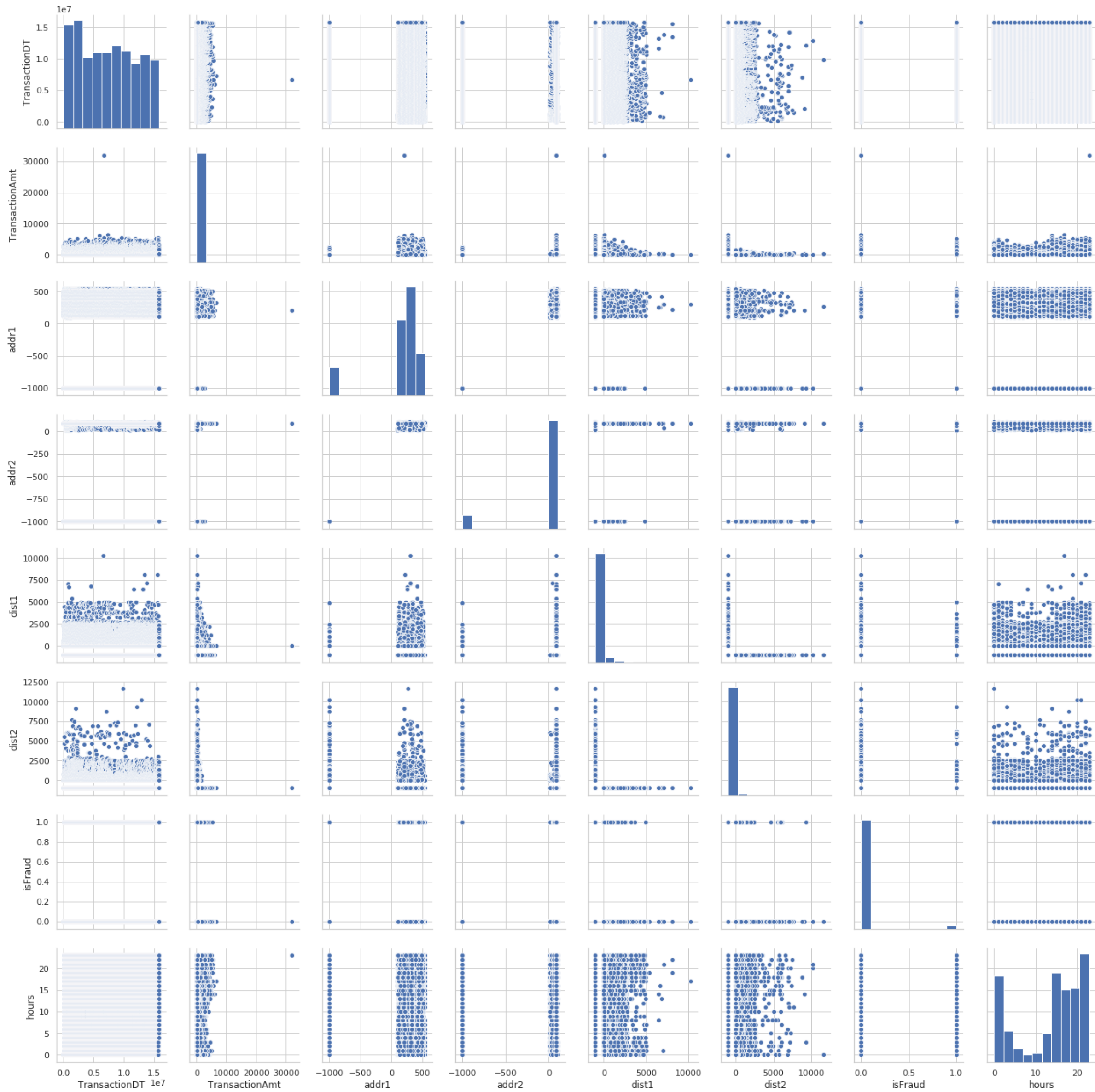
In [83]:
```
#Checking which type of device saw highest number of fraudulent transactions.
sns.set(style="whitegrid")
figure = plt.figure(figsize=(8,8))
sns.set_color_codes("pastel")
sns.barplot(x=train_data_filtered['DeviceType'],y=train_data_filtered['isFrau
d'],
            data=train_data_filtered,label="Fraud or not",color = "b").set_tit
le("Type of Device V/S isFraud")
```

Out[83]:
```
Text(0.5, 1.0, 'Type of Device V/S isFraud')
```

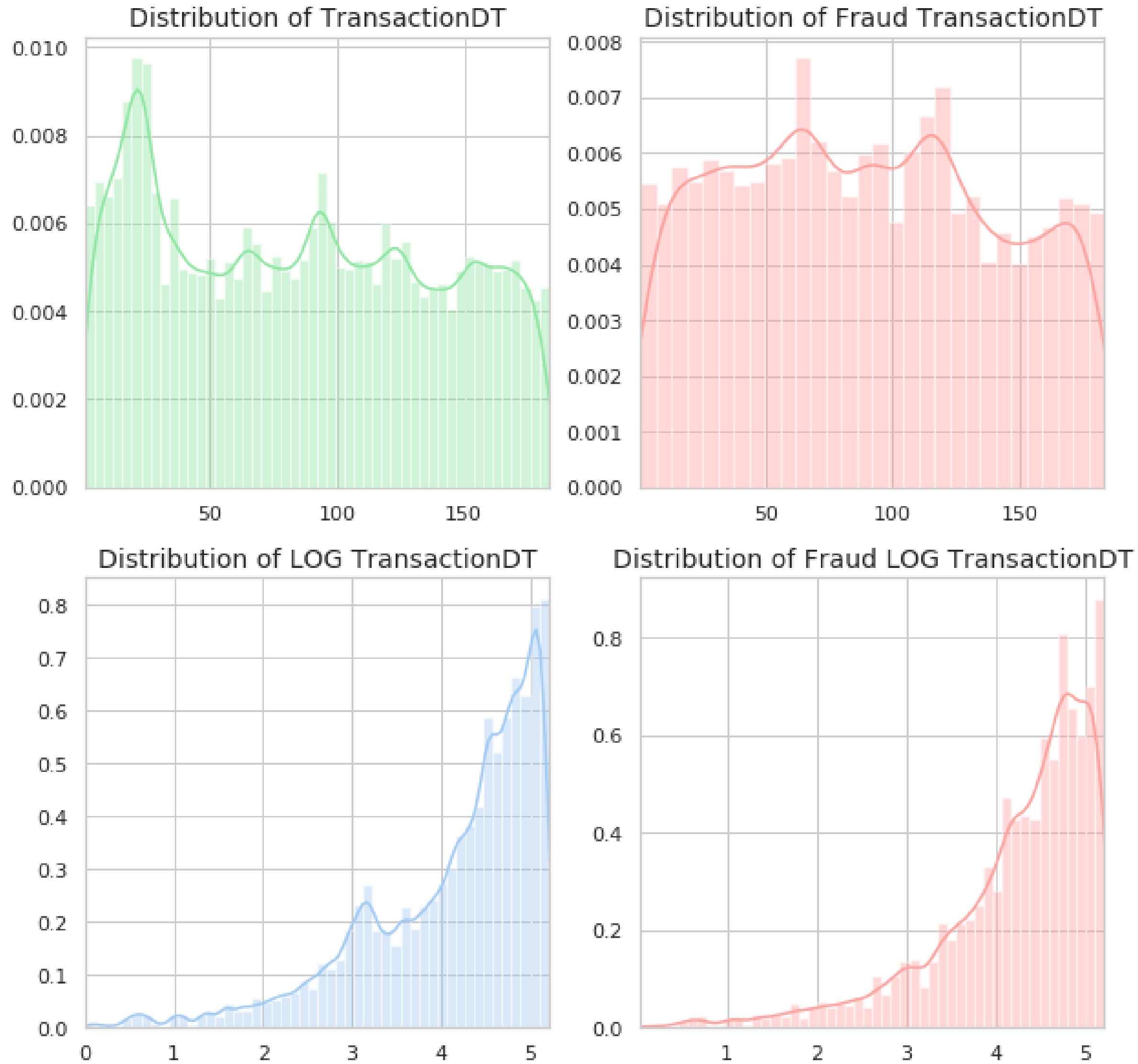


In [0]:
```
sns.pairplot(train_data_filtered)
```

Out[0]: <seaborn.axisgrid.PairGrid at 0x7ff16591d588>



In [87]:
```
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()
```



Answer:

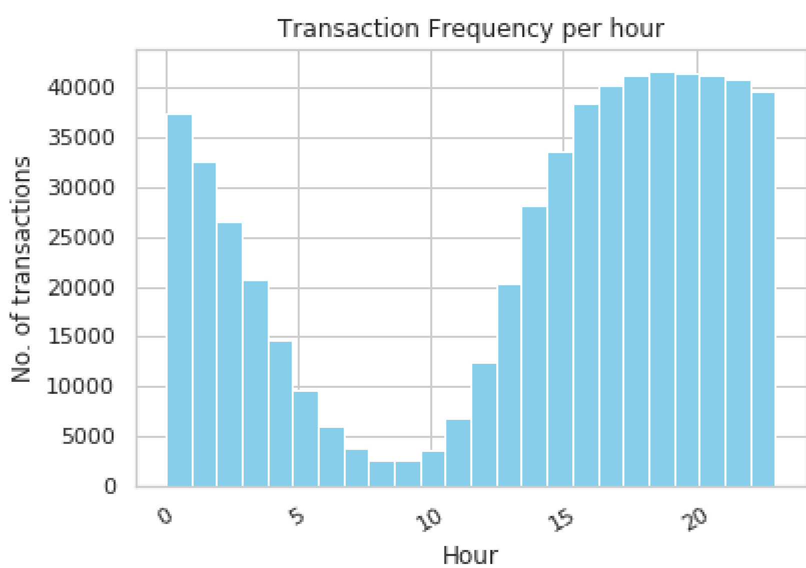
In the above graph, I have plotted the transaction date time V/S Fraud or Not.

Part 2 - Transaction Frequency

In [0]:
```
# 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")
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:19: SettingWithC
opyWarning:
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/st
able/indexing.html#indexing-view-versus-copy
```

Out[0]: Text(0.5, 0, 'Hour')



Answer:

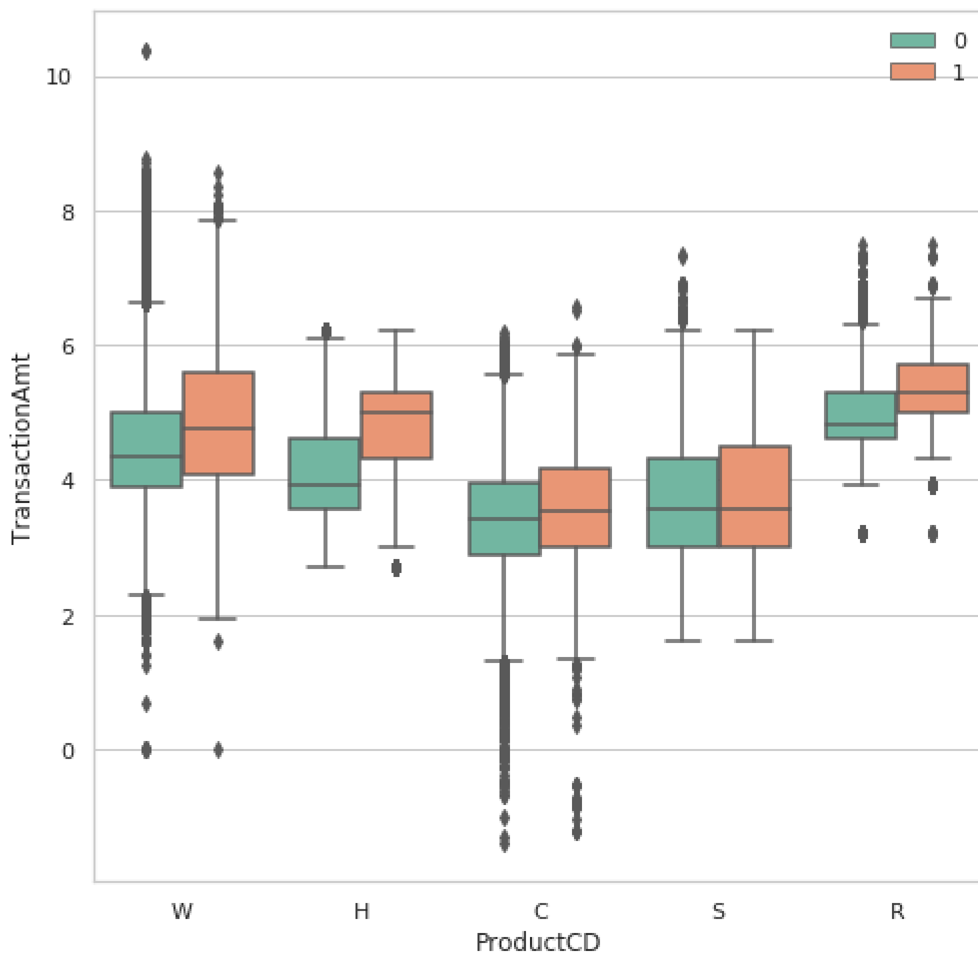
From the 24 hours in a day (shown as 0-23 in the graph above), one can clearly see the dip in the graph of number of transactions per hour. The dip represents the sleeping hours of the vast majority. An inverse bell curve graph helps determine the waking v/s sleeping hours for the most frequent country code which is 87.0

Part 3 - Product Code

In [0]:
```
# 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].Tran
sactionAmt.count())
  print("Mean: ",train_data_filtered[train_data_filtered.ProductCD == i].Trans
actionAmt.mean())
  count = train_data_filtered[train_data_filtered.ProductCD == i].TransactionA
mt.count()
  median = train_data_filtered[train_data_filtered.ProductCD == i].Transaction
Amt.median()
  mean = train_data_filtered[train_data_filtered.ProductCD == i].TransactionAm
t.mean()
  std_deviation = train_data_filtered[train_data_filtered.ProductCD == i].Tran
sactionAmt.std()
  if mean > maxCost:
    maxCost = mean
    product = i

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

for j in train_data_filtered.ProductCD.unique():
    mean = train_data_filtered[train_data_filtered.ProductCD == j].Transaction
Amt.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 = train_data_filtered['isFraud'], palette = "Set2")
ax = sns.boxplot( x=train_data_filtered['ProductCD'], y=np.log(train_data_filt
ered['TransactionAmt']), hue = train_data_filtered['isFraud'], palette = "Set
2")
ax.legend(frameon=False, loc='upper right', ncol=1)
```

```
W
Count:  439670
Mean:  153.15855385223293
H
Count:  33024
Mean:  73.17005813953489
C
Count:  68519
Mean:  42.872353113733446
S
Count:  11628
Mean:  60.269487444100434
R
Count:  37699
Mean:  168.30618849306347
The most expensive product is R and it's average cost is 168.30618849306347
The cheapest product is C and it's average cost is 42.872353113733446
```

Out[0]: <matplotlib.legend.Legend at 0x7f7471a96048>



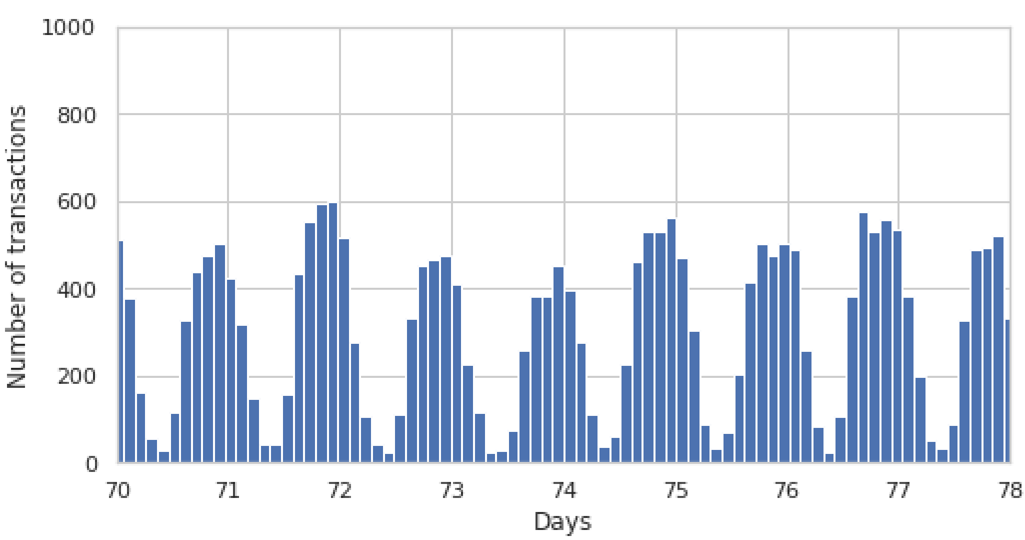
Answer:
The average cost of each product is calculated by the total cost of that product divided by the total number of product transactions. By calculating the above, we have found that Product **R** is the **most expensive** product with an average cost of **168.30** and Product **C** is the **cheapest** one with an average cost of **42.872**.

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

Part 4 - Correlation Coefficient

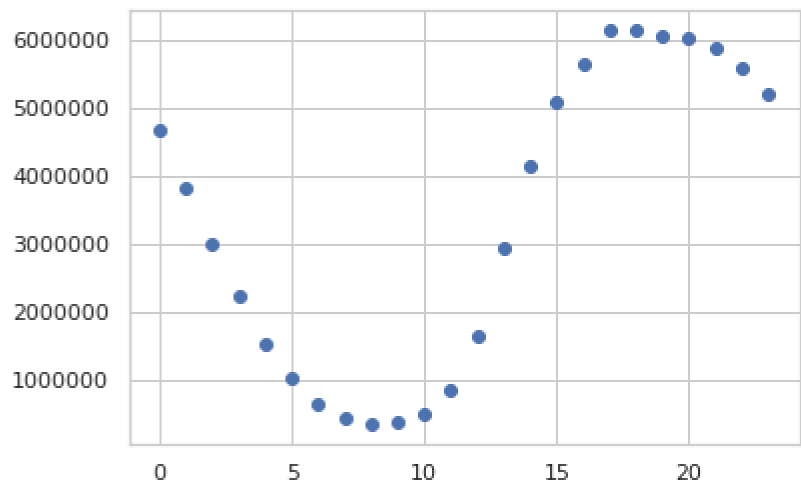
In [0]:
```
# TODO: code to calculate correlation coefficient
from datetime import datetime
train_data_filtered['TransactionTime'] = pd.to_datetime(train_data_filtered['T
ransactionDT'],unit = 's').dt.time


#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)
```

Out[0]: (0, 1000)



In [0]:
```
#By grouping them in hours and then summing the amount of transactions per hou
r gives below
train_data_filtered['hours'] = make_hour_feature(train_data_filtered)
dataGroupedByHours = train_data_filtered.groupby('hours')
plt.scatter(dataGroupedByHours['hours'].unique(),dataGroupedByHours.Transactio
nAmt.sum())
```

Out[0]: <matplotlib.collections.PathCollection at 0x7f746f497f28>



In [0]:
```
print("Spearman Correlation coeffecient with sum of transaction amounts: ",spe
armanr(dataGroupedByHours['hours'].unique(),dataGroupedByHours.TransactionAmt.
sum()))
```

```
Spearman Correlation coeffecient with sum of transaction amounts:  SpearmanrR
esult(correlation=0.6304347826086956, pvalue=0.0009590059599017164)
```

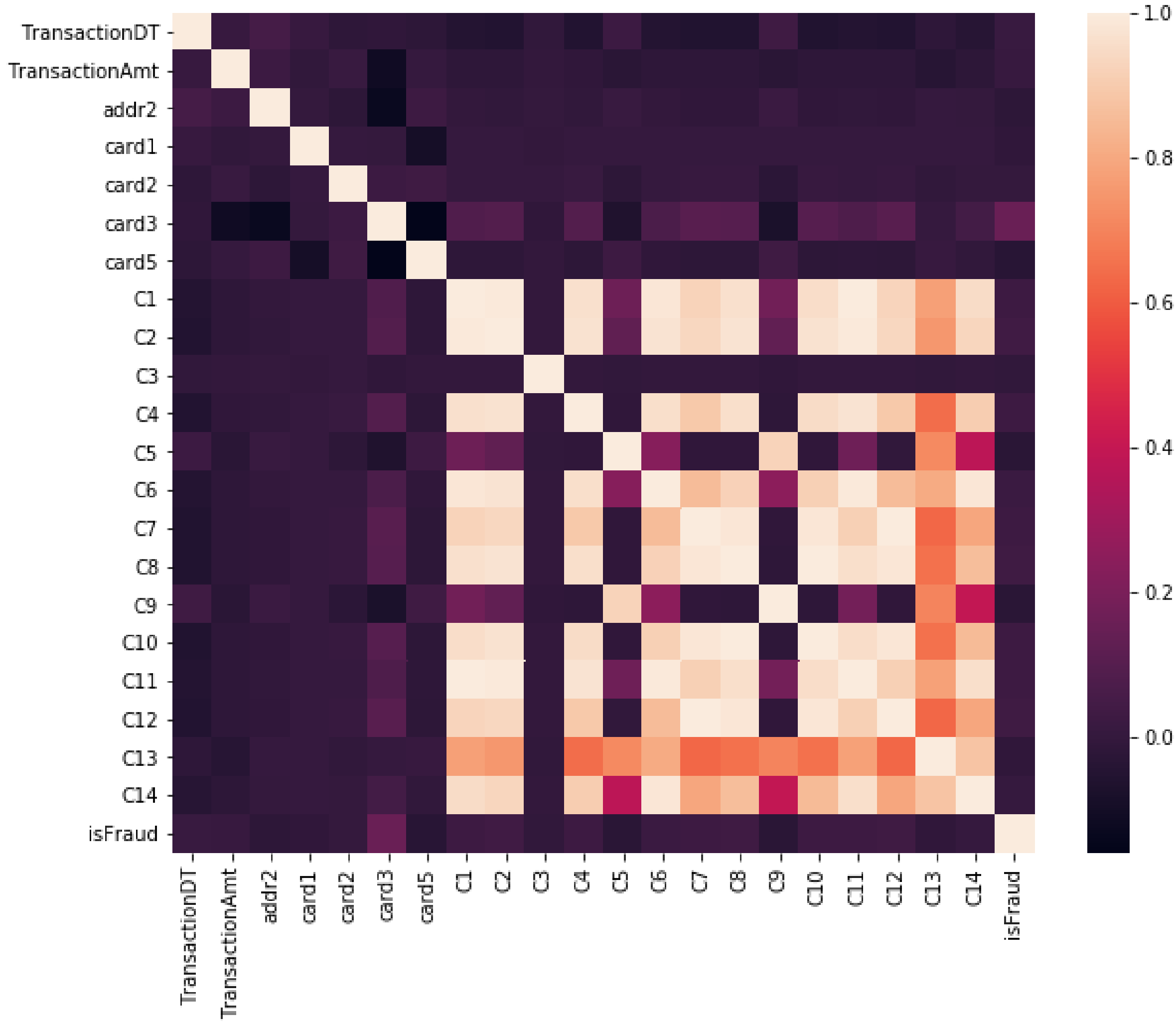
Answer:

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

Part 5 - Interesting Plot

In [0]:
```
plt.figure(figsize=(10,8))
sns.heatmap(train_data_final.corr())
```

Out[0]: <matplotlib.axes._subplots.AxesSubplot at 0x7fd87bb97898>

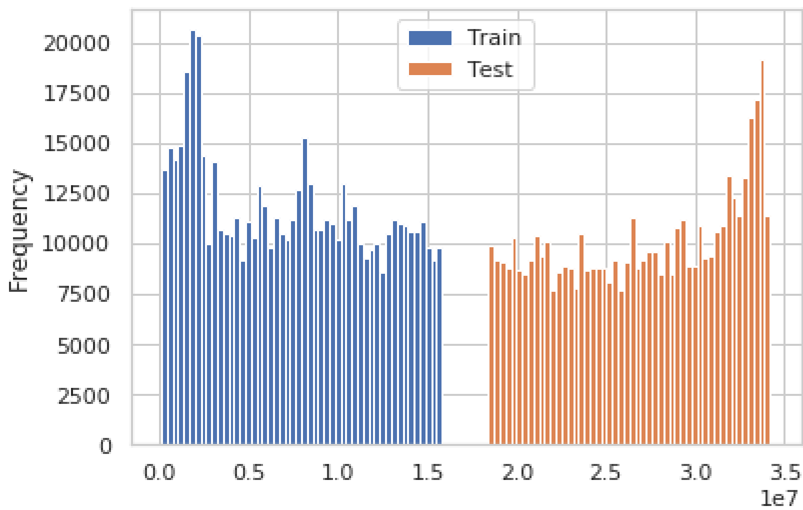


Answer:

The above heatmap helps us understand the correlation between the various columns of our dataset. However, a quick glance tells us that even though most of them depict very low correlation with each other, **the attribute 'card3' is highly correlated with most of the other attributes and it shows a correlation of around 0.2 with our target attribute 'isFraud'**. Some interesting insights can be found with this information and our model predicts better when this column is included.

In [88]:
```
#No overlap between train and test data dates.
train_data['TransactionDT'].plot(kind = "hist", label = "Train", bins = 50)
test_data['TransactionDT'].plot(kind = "hist", label = "Test", bins = 50)
plt.legend()
```

Out[88]: <matplotlib.legend.Legend at 0x7fd8755e8fd0>



Answer:

The above graph shows that there is no overlap between the training and testing data. This signifies that the transactions recorded for training and testing were not done during the same time interval but rather some time intervals apart.

Train: min = 86400, max = 15811131

Test: min = 18403224, max = 34214345

If we assume TransactionDT is in seconds, then:

Time span of the total dataset is 394.9993634259259 days.

Time span of Train dataset is 181.99920138888888 days.

Time span of Test dataset is 182.99908564814814 days.

The gap between train and test is 30.00107638888889 days.

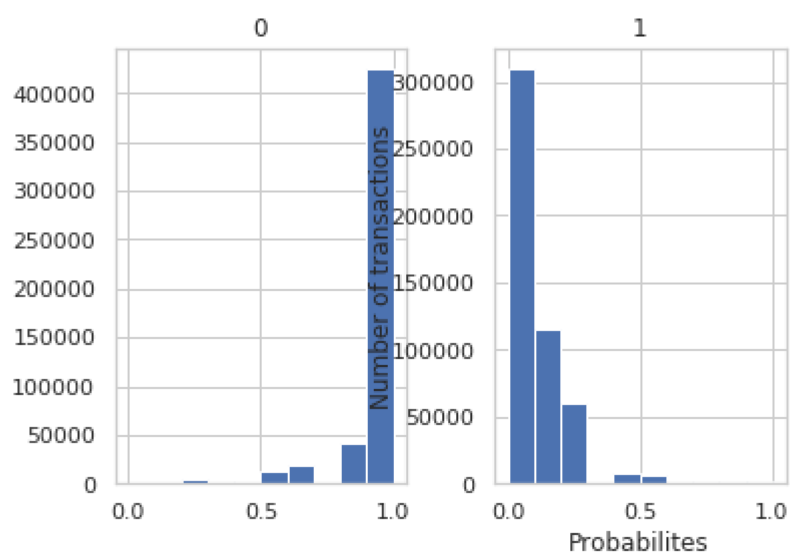
Part 6 - Prediction Model

In [0]:
```
#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 = 0.30, random_state = 10)
```

In [77]:
```
# 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 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()
plt.xlabel("Probabilites", ha = 'center')
plt.ylabel("Number of transactions", va = 'center')
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/ensemble/forest.py:245: Future
Warning: The default value of n_estimators will change from 10 in version 0.2
0 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
```

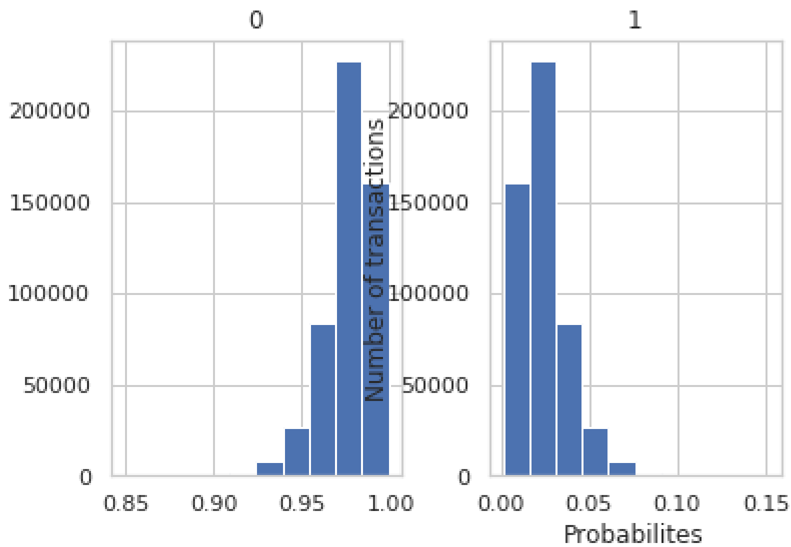
Out[77]:
```
Text(0, 0.5, 'Number of transactions')
```



In [78]:
```
#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 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()
plt.xlabel("Probabilites", ha = 'center')
plt.ylabel("Number of transactions", va = 'center')
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/logistic.py:432:
FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a s
olver to silence this warning.
  FutureWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/logistic.py:432:
FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a s
olver to silence this warning.
  FutureWarning)

Logistic Regression - Percentage accuracy:  96.44393267179193
```

Out[78]: `Text(0, 0.5, 'Number of transactions')`



Answer:

I have worked on two models for predicting whether a transaction is fraudulent or not - Random forests classifier and a logistic regression model. The random forests one works better as it uses a modified tree learning algorithm, where in at each step of the learning process, it selects a random subset of features. This way it tries out different combinations of the feature sets and finally ensembles the ones which give out the most accurate predictions.

Accuracy of this model: 98.45%

Kaggle score: 0.7089

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: PayalMehta_Kaggle_Link (https://www.kaggle.com/payal95/competitions)

Highest Rank: 5532

Score: 0.7089

Number of entries: 19

Credits and References:

1. Time and Day - Predictive feature (https://www.kaggle.com/fchmiel/day-and-time-powerful-predictive-feature)
2. Fraud_models (https://www.kaggle.com/jesucristo/fraud-complete-eda/notebook#Models)

In [71]:
```
#@title Part 8 - Kaggle Rank
%%html
<iframe src = "https://drive.google.com/uc?id=1OGNtjUoniouTm7r6nmVaeIlNv4RGzCBd" width = "840" height = "480"></iframe>
```