Fraud Detection in Credit Card Transactions

In [60]:

```
import pandas as pd
pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 1000)

import scipy.stats
import numpy as np
import datetime as dt
import seaborn as sns
import matplotlib.pyplot as plt
from copy import deepcopy
import itertools
from typing import List
import time
from scipy import stats
from scipy.stats import zscore
import random
import math
from math import log10
import statistics
from pandas.plotting import register_matplotlib_converters
import matplotlib.dates as mdates
import psutil

%matplotlib inline
start_time = pd.datetime.now()
```

Data Preprocessing

Import the Dataset

In [2]:

```
df_original = pd.read_excel('card transactions.xlsx', converters={'Merchnum': lambda x: st
r(x)})
```

In [3]:

```
# Convert contents of the Merch description dta field to uppercase
df_original['Merch description'] = df_original['Merch description'].str.upper()
```

In [4]:

```
df_original.dtypes
```

Out[4]:

```
Recnum                        int64
Cardnum                       int64
Date                 datetime64[ns]
Merchnum                     object
Merch description            object
Merch state                  object
Merch zip                   float64
Transtype                    object
Amount                      float64
Fraud                         int64
dtype: object
```

Remove all records that do not have a value of "P" ("purchase") in the Transtype data field.

In [5]:

```
df_P = df_original.loc[df_original['Transtype'] == 'P']
print(len(df_P))
df_P.head()
```

```
96398
```

Out[5]:

	Recnum	Cardnum	Date	Merchnum	Merch description	Merch state	Merch zip	Transtype	Amc
0	1	5142190439	2010-01-01	5509006296254	FEDEX SHP 12/23/09 AB#	TN	38118.0	P	
1	2	5142183973	2010-01-01	61003026333	SERVICE MERCHANDISE #81	MA	1803.0	P	3
2	3	5142131721	2010-01-01	4503082993600	OFFICE DEPOT #191	MD	20706.0	P	17
3	4	5142148452	2010-01-01	5509006296254	FEDEX SHP 12/28/09 AB#	TN	38118.0	P	
4	5	5142190439	2010-01-01	5509006296254	FEDEX SHP 12/23/09 AB#	TN	38118.0	P	

Data Imputation

In this section we fill missing values of the dataset.

CSV File of US Zip Codes

This portion of the code requires an additional downnload of a CSV file found at the reference URL below. This file is available for free.
Reference: https://simplemaps.com/data/us-zips (https://simplemaps.com/data/us-zips)

In [6]:

```
# Import the CSV file of US Zip Codes
# Reference:  https://simplemaps.com/data/us-zips
df_zipCodes = pd.read_csv('uszips.csv')

# Create a new column of the zip codes (zip2) and convert it to a float since our datafram
e uses a float for the Zip data field
df_zipCodes['zip2'] = df_zipCodes.zip.astype('float64')
df_zipCodes.head()
```

Out[6]:

	zip	lat	lng	city	state_id	state_name	zcta	parent_zcta	population	dens
0	601	18.18004	-66.75218	Adjuntas	PR	Puerto Rico	True	NaN	17242	11
1	602	18.36073	-67.17517	Aguada	PR	Puerto Rico	True	NaN	38442	52
2	603	18.45439	-67.12202	Aguadilla	PR	Puerto Rico	True	NaN	48814	66
3	606	18.16724	-66.93828	Maricao	PR	Puerto Rico	True	NaN	6437	6(
4	610	18.29032	-67.12243	Anasco	PR	Puerto Rico	True	NaN	27073	31

In [7]:

```
# Make a dictionary to map the zip code to the state from the uszips.csv file
# zip2 is the dictionary key; state_id is the dictionary value
zip_dict = pd.Series(df_zipCodes.state_id.values,index=df_zipCodes.zip2.values).to_dict()
```

Merch state

The “Merch state” data field had 1,021 records with a missing value. Since this data field had the fewest number of records that required cleaning, we opted to start our data imputation with this data field.

In [8]:

```
df_impute = deepcopy(df_P)
print(len(df_impute.loc[df_impute['Merch state'].isna()]))

# STEP 1: map the zip_dict contents to the "Merch state" data field
df_impute.loc[df_impute['Merch state'].isnull(),'Merch state'] = df_impute['Merch zip'].ma
p(zip_dict) # reduced to 976
print(len(df_impute.loc[df_impute['Merch state'].isna()]))
```

```
1021
976
```

In [9]:

```
## STEP 2:
#### These lines of code enable us to use data fields with NaN values when using groupby #
###
# Get rid of the records with NaN values in the Merch zip data field
zip_notnull = df_impute.loc[df_impute['Merch zip'].notnull(), ['Merch zip', 'Merch state'
]]

# Groupby zip, then find the mode of the state
zip_notnull_state = zip_notnull.groupby(['Merch zip'])['Merch state'].transform(lambda x:
x.fillna(x.mode()[0] if not x.mode().empty else np.nan))

# Make a dictionary to match the Recnum to the mode of the zip from the groupby done above
zip_notnull_state_dict = pd.Series(zip_notnull_state.values,index=zip_notnull_state.index+
1).to_dict()

# Map the dictionary back to the "Merch state" data field
df_impute.loc[df_impute['Merch state'].isnull(),'Merch state'] = df_impute['Recnum'].map(z
ip_notnull_state_dict)
print(len(df_impute.loc[df_impute['Merch state'].isna()]))
```

```
959
```

In [10]:

```
# STEP 3: Groupby Merch description, then Cardnum, then find the mode of the state
f_zip = lambda x: x.mode()[0] if not x.mode().empty else np.nan

df_impute['state1'] = df_impute['Merch state'].fillna(df_impute.groupby(['Merch descriptio
n','Cardnum'])['Merch state'].transform(f_zip))
len(df_impute.loc[df_impute['state1'].isna()])
```

Out[10]:

```
904
```

In [11]:

```
# STEP 4: Groupby Cardnm, then date, then find the mode of the state
f_zip = lambda x: x.mode()[0] if not x.mode().empty else np.nan
df_impute['state2'] = df_impute['state1'].fillna(df_impute.groupby(['Cardnum','Date'])['st
ate1'].transform(f_zip))
len(df_impute.loc[df_impute['state2'].isna()])
```

Out[11]:

```
597
```

In [12]:

```
# STEP 5: Groupby Merch description, then find the mode of the state
f_zip = lambda x: x.mode()[0] if not x.mode().empty else np.nan
df_impute['state3'] = df_impute['state2'].fillna(df_impute.groupby(['Merch description'])[
'state2'].transform(f_zip))
len(df_impute.loc[df_impute['state3'].isna()])
```

Out[12]:

```
129
```

In [13]:

```
# STEP 6: Groupby Cardnm, then date, then find the mode of the state
f_zip = lambda x: x.mode()[0] if not x.mode().empty else np.nan
df_impute['state4'] = df_impute['state3'].fillna(df_impute.groupby(['Cardnum','Date'])['st
ate3'].apply(lambda x: x.ffill().bfill()))
len(df_impute.loc[df_impute['state4'].isna()])
```

Out[13]:

```
126
```

In [14]:

```
# STEP 7: Groupby Cardnm, then find the mode of the state
f_zip = lambda x: x.mode()[0] if not x.mode().empty else np.nan
df_impute['state5'] = df_impute['state4'].fillna(df_impute.groupby(['Cardnum'])['state4'].
apply(lambda x: x.ffill().bfill()))
len(df_impute.loc[df_impute['state5'].isna()])
```

Out[14]:

```
31
```

In [15]:

```
# STEP 8: Fill remaining missing values with the Recnum to avoid linkages
df_impute['state6'] = df_impute['state5'].fillna(df_impute.Recnum)
len(df_impute.loc[df_impute['state6'].isna()])
```

Out[15]:

```
0
```

Merch zip

The “Merch zip” data field had 4,301 records with a missing value.

In [16]:

```
# Look at how many NaN values for "Merch zip"
len(df_impute.loc[df_impute['Merch zip'].isna()])
```

Out[16]:

```
4301
```

In [17]:

```
# Look at how many NaN values for "Merchnum"
len(df_impute.loc[df_impute['Merchnum'].isna()])
```

Out[17]:

```
3199
```

In [18]:

```
f_mode = lambda x: x.mode()[0] if not x.mode().empty else np.nan
print(len(df_impute.loc[df_impute['Merch zip'].isna()]))

## STEP 1:
#### These lines of code enables us to use data fields with NaN values when using groupby
 ####
# Get rid of the records with NaN values in the Merchnum
merchnum_notnull = df_impute.loc[df_impute['Merchnum'].notnull(), ['Merchnum', 'state6','M
erch zip']]

# Groupby Merchnum, Merch state, then find the mode of the zip
merchnum_notnull_zip = merchnum_notnull.groupby(['Merchnum','state6'])['Merch zip'].transf
orm(lambda x: x.fillna(x.mode()[0] if not x.mode().empty else np.nan))

# Make a dictionary to match the Recnum to the mode of the zip from the groupby done above
merchnum_notnull_zip_dict = pd.Series(merchnum_notnull_zip.values,index=merchnum_notnull_z
ip.index+1).to_dict()

# Map the dictionary back to the "Merch zip" data field
df_impute.loc[df_impute['Merch zip'].isnull(),'Merch zip'] = df_impute['Recnum'].map(merch
num_notnull_zip_dict)
print(len(df_impute.loc[df_impute['Merch zip'].isna()]))
```

```
4301
2474
```

In [19]:

```
# STEP 2: Groupby Merch description, then the fully filled state (state6), then find the m
ode of the Merch zip
f_mode = lambda x: x.mode()[0] if not x.mode().empty else np.nan

df_impute['zip1'] = df_impute['Merch zip'].fillna(df_impute.groupby(['Merch description',
'state6'])['Merch zip'].transform(f_mode))
len(df_impute.loc[df_impute['zip1'].isna()])
```

Out[19]:

```
2151
```

In [20]:

```
# STEP 3: Groupby Merch description, then find the mode of the zip
f_mode = lambda x: x.mode()[0] if not x.mode().empty else np.nan

df_impute['zip2'] = df_impute['zip1'].fillna(df_impute.groupby(['Merch description'])['zip
1'].transform(f_mode))
len(df_impute.loc[df_impute['zip2'].isna()])
```

Out[20]:

```
2030
```

In [21]:

```
# STEP 4: Groupby fully filled state (state6), then find the mode of the zip
f_mode = lambda x: x.mode()[0] if not x.mode().empty else np.nan

df_impute['zip3'] = df_impute['zip2'].fillna(df_impute.groupby(['state6'])['zip2'].transfo
rm(f_mode))
len(df_impute.loc[df_impute['zip3'].isna()])
```

Out[21]:

```
71
```

In [22]:

```
# STEP 5: Groupby Cardnm, then date, then find the mode of the zip
f_mode = lambda x: x.mode()[0] if not x.mode().empty else np.nan

df_impute['zip4'] = df_impute['zip3'].fillna(df_impute.groupby(['Cardnum','Date'])['zip3']
.transform(f_mode))
len(df_impute.loc[df_impute['zip4'].isna()])
```

Out[22]:

```
54
```

In [23]:

```
# STEP 6: Groupby Merch description, then the fully filled state (state6), then find the m
ode of the zip
f_mode = lambda x: x.mode()[0] if not x.mode().empty else np.nan

df_impute['zip5'] = df_impute['zip4'].fillna(df_impute.groupby(['Merch description','state
6'])['zip4'].transform(f_mode))
len(df_impute.loc[df_impute['zip5'].isna()])
```

Out[23]:

46

In [24]:

```
# STEP 7: Groupby Merch description, then find the mode of the zip (PRODUCED NO ADVANCEMEN
T IN RESULTS)
f_mode = lambda x: x.mode()[0] if not x.mode().empty else np.nan

df_impute['zip6'] = df_impute['zip5'].fillna(df_impute.groupby(['Merch description'])['zip
5'].transform(f_mode))
len(df_impute.loc[df_impute['zip6'].isna()])
```

Out[24]:

46

In [25]:

```
# STEP 7: Groupby fully filled state (state6), then find the mode of the zip
f_mode = lambda x: x.mode()[0] if not x.mode().empty else np.nan

df_impute['zip7'] = df_impute['zip6'].fillna(df_impute.groupby(['state6'])['zip6'].transfo
rm(f_mode))
len(df_impute.loc[df_impute['zip7'].isna()])
```

Out[25]:

31

In [26]:

```
# STEP 8: Groupby Cardnm, then date, then find the mode of the zip (PRODUCED NO ADVANCEMEN
T IN RESULTS)
f_mode = lambda x: x.mode()[0] if not x.mode().empty else np.nan

df_impute['zip8'] = df_impute['zip7'].fillna(df_impute.groupby(['Cardnum','Date'])['zip7']
.transform(f_mode))
len(df_impute.loc[df_impute['zip8'].isna()])
```

Out[26]:

31

In [27]:

```
# STEP 8: Fill remaining missing values with the Recnum to avoid linkages
f_mode = lambda x: x.mode()[0] if not x.mode().empty else np.nan

df_impute['zip9'] = df_impute['zip8'].fillna(df_impute.Recnum)
len(df_impute.loc[df_impute['zip9'].isna()])
```

Out[27]:

```
0
```

Merchnum

The “Merchnum” data field had 3,199 records with a missing value and 53 records with an erroneous value of “0” as its merchant number.

In [28]:

```
f_mode = lambda x: x.mode()[0] if not x.mode().empty else np.nan

# View the number of records with "NaN" values for Merchnum
print(len(df_impute.loc[df_impute['Merchnum'].isna()]))

# STEP 1: Convert values of "0" in the Merchnum data field to "NaN" values
df_impute['Merchnum0'] = np.where(df_impute['Merchnum'] == '0', np.nan, df_impute['Merchnu
m'])
print(len(df_impute.loc[df_impute['Merchnum0'].isna()])) # Look at the new number of recor
ds with "NaN" values

# STEP 2: Groupby Merch description, then the fully filled zip (zip9), then find the mode
 of the Merchnum
df_impute['Merchnum1'] = df_impute['Merchnum0'].fillna(df_impute.groupby(['Merch descripti
on','zip9'])['Merchnum0'].transform(f_mode))
len(df_impute.loc[df_impute['Merchnum1'].isna()])
```

```
3199
3252
```

Out[28]:

```
2169
```

In [29]:

```
# STEP 3: Groupby Merch description, then the fully filled state (state6), then find the m
ode of the Merchnum
df_impute['Merchnum2'] = df_impute['Merchnum1'].fillna(df_impute.groupby(['Merch descripti
on','state6'])['Merchnum1'].transform(f_mode))
len(df_impute.loc[df_impute['Merchnum2'].isna()])
```

Out[29]:

```
2110
```

In [30]:

```
# STEP 4: Groupby Merch description, then find the mode of the Merchnum
df_impute['Merchnum3'] = df_impute['Merchnum2'].fillna(df_impute.groupby(['Merch descripti
on'])['Merchnum2'].transform(f_mode))
len(df_impute.loc[df_impute['Merchnum3'].isna()])
```

Out[30]:

```
2095
```

In [31]:

```
# STEP 5: Fill remaining missing values with the Recnum to avoid linkages
df_impute['Merchnum4'] = df_impute['Merchnum3'].fillna(df_impute.Recnum)
len(df_impute.loc[df_impute['Merchnum4'].isna()])
```

Out[31]:

```
0
```

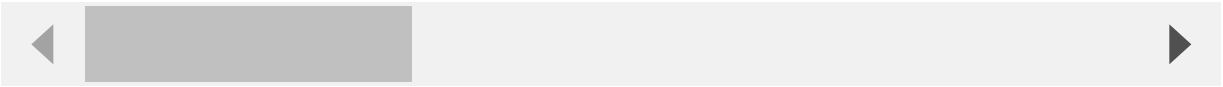
Save the fully imputed dataframe (raw form)

In [32]:

```
df_impute.head()
```

Out[32]:

	Recnum	Cardnum	Date	Merchnum	Merch description	Merch state	Merch zip	Transtype	Amo
0	1	5142190439	2010-01-01	5509006296254	FEDEX SHP 12/23/09 AB#	TN	38118.0	P	3
1	2	5142183973	2010-01-01	61003026333	SERVICE MERCHANDISE #81	MA	1803.0	P	3
2	3	5142131721	2010-01-01	4503082993600	OFFICE DEPOT #191	MD	20706.0	P	178
3	4	5142148452	2010-01-01	5509006296254	FEDEX SHP 12/28/09 AB#	TN	38118.0	P	3
4	5	5142190439	2010-01-01	5509006296254	FEDEX SHP 12/23/09 AB#	TN	38118.0	P	3



Save the fully imputed dataframe (clean form)

In [33]:

```
df_impute_final = df_P.copy()
df_impute_final.Merchnum = df_impute.Merchnum4
print(len(df_impute_final.loc[df_impute_final['Merchnum'].isna()]))
df_impute_final['Merch zip'] = df_impute['zip9'].astype(int)
print(len(df_impute_final.loc[df_impute_final['Merch zip'].isna()]))
df_impute_final['Merch state'] = df_impute['state6']
print(len(df_impute_final.loc[df_impute_final['Merch state'].isna()]))
df_impute_final['Merch description'] = df_impute_final['Merch description'].str.upper()
```

```
0
0
0
```

In [34]:

```
df_impute_final.head()
```

Out[34]:

	Recnum	Cardnum	Date	Merchnum	Merch description	Merch state	Merch zip	Transtype	Amou
0	1	5142190439	2010-01-01	5509006296254	FEDEX SHP 12/23/09 AB#	TN	38118	P	3.
1	2	5142183973	2010-01-01	61003026333	SERVICE MERCHANDISE #81	MA	1803	P	31.
2	3	5142131721	2010-01-01	4503082993600	OFFICE DEPOT #191	MD	20706	P	178.
3	4	5142148452	2010-01-01	5509006296254	FEDEX SHP 12/28/09 AB#	TN	38118	P	3.
4	5	5142190439	2010-01-01	5509006296254	FEDEX SHP 12/23/09 AB#	TN	38118	P	3.

In [35]:

```
df_impute_final.dtypes
```

Out[35]:

```
Recnum                         int64
Cardnum                        int64
Date                  datetime64[ns]
Merchnum                      object
Merch description             object
Merch state                   object
Merch zip                      int64
Transtype                     object
Amount                       float64
Fraud                          int64
dtype: object
```

Remove the HIGH AMOUNT value

This was a required exclusion outlined as a part of our project. We forgot to do it earlier, so we ended up doing it here.

In [36]:

```
df_impute_final.loc[df_impute_final['Recnum']==52715]
```

Out[36]:

	Recnum	Cardnum	Date	Merchnum	Merch description	Merch state	Merch zip	Transtype	Am
52714	52715	5142189135	2010-07-13	52715	INTERMEXICO	CA	92656	P	31020

In [37]:

```
df_impute_final.drop(index=52714,inplace=True,axis=0)
```

In [38]:

```
df_impute_final.loc[df_impute_final['Recnum']==52715]
```

Out[38]:

	Recnum	Cardnum	Date	Merchnum	Merch description	Merch state	Merch zip	Transtype	Amount	Fraud

In [39]:

```
df_impute_final.tail()
```

Out[39]:

	Recnum	Cardnum	Date	Merchnum	Merch description	Merch state	Merch zip	Transtype	Amou
96748	96749	5142276053	2010-12-31	3500000006160	BEST BUY 00001610	KY	41042	P	84
96749	96750	5142225701	2010-12-31	8090710030950	MARKUS OFFICE SUPPLIES	OH	45248	P	118
96750	96751	5142226486	2010-12-31	4503057341100	TECH PAC, INC	OH	45150	P	363
96751	96752	5142244619	2010-12-31	8834000695412	BUY.COM	CA	92656	P	2202
96752	96753	5142243247	2010-12-31	9108347680006	STAPLES NATIONAL #471	NJ	7606	P	554

Feature Engineering

In this section we create new features based on the original ones in the dataset.

Create the Categorical Candidate Variables

In [40]:

```
start_convert = pd.datetime.now()
df_var = deepcopy(df_impute_final)

# Convert values to string type so that we can concatenate some of them together to make v
ariables
cols_convert = [df_var.columns.drop(['Date','Fraud','Amount'])] # Don't convert these data
fields
for item in cols_convert:
    df_var[item] = df_var[item].astype(str)

print('convert time', pd.datetime.now()-start_convert)
df_var.dtypes
```

```
convert time 0:00:00.281777
```

Out[40]:

```
Recnum                        object
Cardnum                       object
Date                  datetime64[ns]
Merchnum                      object
Merch description             object
Merch state                   object
Merch zip                     object
Transtype                     object
Amount                       float64
Fraud                          int64
dtype: object
```

In [41]:

```
# Make variable combos from the Lesson 10 slides (slide 40)
df_var['card-merch'] = df_var['Cardnum'] + df_var['Merchnum'] # Class Q&A said not to use
 Merch description
df_var['card-zip'] = df_var['Cardnum'] + df_var['Merch zip']
df_var['card-state'] = df_var['Cardnum'] + df_var['Merch state']
```

Convert appropriate data fields to numbers

In [42]:

```
# Convert appropriate data fields back to integers for faster processing later in the code
start_convert=pd.datetime.now()

cols_int = ['Recnum','Cardnum','Merchnum','Merch zip']
cols_var = df_var.columns

for item in cols_var:
    temp = set(item.split('-'))
#     print(temp)
    if temp.issubset(cols_int):
        try:
            df_var[item] = df_var[item].astype(int)
        except:
            try:
                df_var[item] = df_var[item].astype('int64')
            except:
                print('The values are probably too big:', item)
                continue

print('convert time', pd.datetime.now()-start_convert)
df_var.dtypes
```

```
The values are probably too big: Merchnum
convert time 0:00:00.035558
```

Out[42]:

```
Recnum                         int64
Cardnum                        int64
Date                  datetime64[ns]
Merchnum                      object
Merch description             object
Merch state                   object
Merch zip                      int64
Transtype                     object
Amount                       float64
Fraud                          int64
card-merch                    object
card-zip                      object
card-state                    object
dtype: object
```

Days-Since Variables

It is number of days since the last time a categorical data field or candidate variable was seen for a particular transaction record.

In [43]:

```
def ds(dataframe, g1, g2, name):
#     'Helps with calculating the day since variables'
    day_since = dataframe.groupby(g1)[g1].first()
    day_since = day_since.rename_axis(['None' for i in range(len(g1))]).groupby(g2).diff()
    day_since.columns = [name]
    day_since = day_since.rename_axis(g1)
    day_since[name] = day_since[name].dt.days.fillna(0)
    day_since = day_since.reset_index()
    return day_since
```

In [44]:

```
# Example from Prof's code
# ssn - finds the numbers of "days since" for the "ssn"
start_daySince = pd.datetime.now()
day_card = ds(df_var, ['Cardnum', 'Date'], 'Cardnum', 'card_daysSince')
print("done!", pd.datetime.now()-start_daySince)
day_card
```

```
done! 0:00:01.315008
```

Out[44]:

	Cardnum	Date	card_daysSince
0	5142110002	2010-10-12	0.0
1	5142110081	2010-03-08	0.0
2	5142110081	2010-11-26	263.0
3	5142110081	2010-12-27	31.0
4	5142110313	2010-10-07	0.0
...	...	...	...
61900	5142847398	2010-03-21	3.0
61901	5142847398	2010-03-22	1.0
61902	5142847398	2010-03-24	2.0
61903	5142847398	2010-03-28	4.0
61904	5142847398	2010-03-29	1.0

61905 rows × 3 columns

In [45]:

```
time_ds_all=pd.datetime.now()
# Calculate the Days Since variables for the required columns
ds_cols = ['Cardnum','Merchnum','card-merch','card-state','card-zip']

ds_dict={}
for col in ds_cols:
    curr_time=pd.datetime.now()
    curr_name = 'daysSince_' + col

    # Calculate the days-since variable (ds) and assign it to a global variable (curr_nam
e)
    vars()[curr_name] = ds(df_var, [col, 'Date'], col, col+'_daysSince')
    ds_dict[curr_name] = vars()[curr_name] # Save results to a dictionary

    print("Done with:", col, "; Time:", pd.datetime.now()-curr_time)

print("DONE!", pd.datetime.now()-time_ds_all)
```

```
Done with: Cardnum ; Time: 0:00:01.265444
Done with: Merchnum ; Time: 0:00:10.781707
Done with: card-merch ; Time: 0:00:28.874647
Done with: card-state ; Time: 0:00:11.128392
Done with: card-zip ; Time: 0:00:23.553132
DONE! 0:01:15.604240
```

In [46]:

```
ds_dict.keys()
```

Out[46]:

```
dict_keys(['daysSince_Cardnum', 'daysSince_Merchnum', 'daysSince_card-merch',
'daysSince_card-state', 'daysSince_card-zip'])
```

In [47]:

```
df_ds = df_var.copy()

# Merge the days-since variables with the main dataset
for item in ds_dict.keys():
    col_variable = item.split('_')[1]
    df_ds = pd.merge(df_ds, vars()[item], how='left', left_on=[col_variable,'Date'], right
_on=[col_variable,'Date'])
```

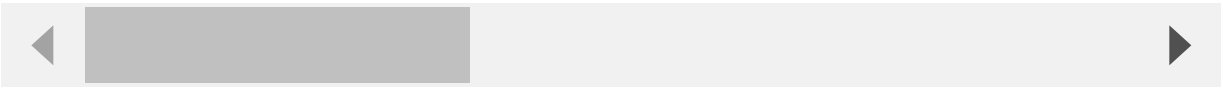
In [48]:

```
df_ds
```

Out[48]:

	Recnum	Cardnum	Date	Merchnum	Merch description	Merch state	Merch zip	Transtype	A
0	1	5142190439	2010-01-01	5509006296254	FEDEX SHP 12/23/09 AB#	TN	38118	P	
1	2	5142183973	2010-01-01	61003026333	SERVICE MERCHANDISE #81	MA	1803	P	
2	3	5142131721	2010-01-01	4503082993600	OFFICE DEPOT #191	MD	20706	P	
3	4	5142148452	2010-01-01	5509006296254	FEDEX SHP 12/28/09 AB#	TN	38118	P	
4	5	5142190439	2010-01-01	5509006296254	FEDEX SHP 12/23/09 AB#	TN	38118	P	
...	...	...	...	...	...	...	...	...	
96392	96749	5142276053	2010-12-31	3500000006160	BEST BUY 00001610	KY	41042	P	
96393	96750	5142225701	2010-12-31	8090710030950	MARKUS OFFICE SUPPLIES	OH	45248	P	
96394	96751	5142226486	2010-12-31	4503057341100	TECH PAC, INC	OH	45150	P	
96395	96752	5142244619	2010-12-31	8834000695412	BUY.COM	CA	92656	P	2
96396	96753	5142243247	2010-12-31	9108347680006	STAPLES NATIONAL #471	NJ	7606	P	

96397 rows × 18 columns



In [ ]:

```
# Save the days-since variables
# df_ds.to_csv('df_daysSince.csv')
```

Frequency Variables

They indicate the speed at which categorical data fields and candidate variables were seen in a dataset for a particular transaction record.

Create Columns for the Necessary Time Periods

This makes new columns for the various time periods.

In [49]:

```
# Make a list of variable combinations to iterate through and create time-related variable
s
cols_drop = ['Recnum','Date','Merch description','Merch state','Merch zip','Transtype','Am
ount','Fraud']
var_combos = df_var.drop(cols_drop,axis=1).columns

# Create column names
time_list = [0,1,3,7,14,30]
time_joined =[]
for time in time_list:
    time_joined.append('join_ts_'+str(time))

start_copy = pd.datetime.now()
df_var1 = deepcopy(df_var)
df_var2 = deepcopy(df_var)
print('copy time', pd.datetime.now()-start_copy)

# Creating columns for time
start_loop=pd.datetime.now()
for time in time_list:
    temp_endTime = 'join_ts_' + str(time)
    df_var2[temp_endTime] = df_var2['Date'] + dt.timedelta(time)
print('first loop', pd.datetime.now()-start_loop)
```

```
copy time 0:00:00.013889
first loop 0:00:00.016109
```

Create the Frequency Candidate Variables

This makes the frequency variables. It's counting the number of records it sees based on the time period. For instance, it counts all the records for a Cardnum it sees over the last 3 days based on the date and Recnum. It only counts the current record and those records in the past for those last 3 days.

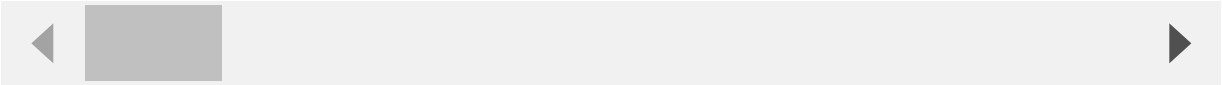
In [50]:

```
start_loop2=pd.datetime.now()
df_final = deepcopy(df_var.set_index('Recnum'))

for item in var_combos:
    df_var3 = df_var1[['Recnum','Date',item]]
    temp_list = time_joined + [item]
    df_var4 = df_var2[temp_list + ['Recnum']].copy()
    df_var4.rename(columns={'Recnum':'Recnum2'},inplace=True) # this causes a warning to a
rise

    df_temp = pd.merge(df_var3, df_var4, left_on=[item], right_on=[item])

    for time in time_list:
        temp_endTime = 'join_ts_' + str(time)
        df2_temp = df_temp[(df_temp['Date'] <= df_temp[temp_endTime]) & (df_temp['Recnum2'
] <= df_temp['Recnum'])]

        temp_groupby = df2_temp[['Recnum','Date']].groupby('Recnum')

        temp_name = item + '_' + 'freq' + str(time) + '_'
        df_final = pd.merge(df_final, getattr(temp_groupby,'count')().add_prefix(temp_name
), left_index=True, right_index=True, how='left')

print('second loop', pd.datetime.now()-start_loop2)
print(len(df_final.columns))
df_final.head()
```

```
second loop 0:00:23.094071
42
```

Out[50]:

Recnum	Cardnum	Date	Merchnum	Merch description	Merch state	Merch zip	Transtype	Amount
1	5142190439	2010-01-01	5509006296254	FEDEX SHP 12/23/09 AB#	TN	38118	P	3.62
2	5142183973	2010-01-01	61003026333	SERVICE MERCHANDISE #81	MA	1803	P	31.42
3	5142131721	2010-01-01	4503082993600	OFFICE DEPOT #191	MD	20706	P	178.49
4	5142148452	2010-01-01	5509006296254	FEDEX SHP 12/28/09 AB#	TN	38118	P	3.62
5	5142190439	2010-01-01	5509006296254	FEDEX SHP 12/23/09 AB#	TN	38118	P	3.62



In [ ]:

```
# Save the frequency variables
# df_final.to_csv('df_frequency_vars.csv')
```

Velocity Change Variables

They indicate the number of appearences with that group seen in the recent past / number of appearences with that same group seen in the past 1, 3, 7, 14, 30, etc. days / number days

In [51]:

```
start_loop3=pd.datetime.now()
groupbyvar_denom = deepcopy(var_combos)
days_numer = ['0','1']
days_denom = ['7','14','30']

for b in groupbyvar_denom:
    for c in days_numer:
        for d in days_denom:
            temp = d
            df_final[b + '_' + c + '_dayfreq' + '_div_' + d + '_dayfreq' + '_velchange'] =
\
                df_final[b + '_freq' + c + '_Date'] / \
                df_final[b + '_freq' + d + '_Date'] / float(temp)
print('third loop', pd.datetime.now() - start_loop3)
```

```
third loop 0:00:00.059850
```

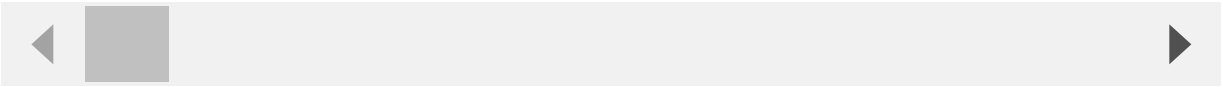
In [52]:

```
print(len(df_final.columns))
df_final.head()
```

72

Out[52]:

Recnum	Cardnum	Date	Merchnum	Merch description	Merch state	Merch zip	Transtype	Amount
1	5142190439	2010-01-01	5509006296254	FEDEX SHP 12/23/09 AB#	TN	38118	P	3.62
2	5142183973	2010-01-01	61003026333	SERVICE MERCHANDISE #81	MA	1803	P	31.42
3	5142131721	2010-01-01	4503082993600	OFFICE DEPOT #191	MD	20706	P	178.49
4	5142148452	2010-01-01	5509006296254	FEDEX SHP 12/28/09 AB#	TN	38118	P	3.62
5	5142190439	2010-01-01	5509006296254	FEDEX SHP 12/23/09 AB#	TN	38118	P	3.62



In [ ]:

```
# Save the frequency and velocity change variables
# df_final.to_csv('df_freq_velchange_vars.csv')
```

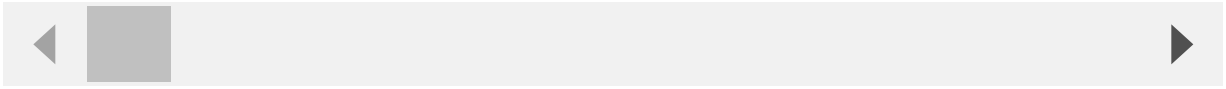
Merge dataframes for all variables

In [53]:

```
# Reset the index so that the Recnum returns to being a normal column
df_final.reset_index(inplace=True)
df_final.head()
```

Out[53]:

	Recnum	Cardnum	Date	Merchnum	Merch description	Merch state	Merch zip	Transtype	Amou
0	1	5142190439	2010-01-01	5509006296254	FEDEX SHP 12/23/09 AB#	TN	38118	P	3.
1	2	5142183973	2010-01-01	61003026333	SERVICE MERCHANDISE #81	MA	1803	P	31.
2	3	5142131721	2010-01-01	4503082993600	OFFICE DEPOT #191	MD	20706	P	178.
3	4	5142148452	2010-01-01	5509006296254	FEDEX SHP 12/28/09 AB#	TN	38118	P	3.
4	5	5142190439	2010-01-01	5509006296254	FEDEX SHP 12/23/09 AB#	TN	38118	P	3.



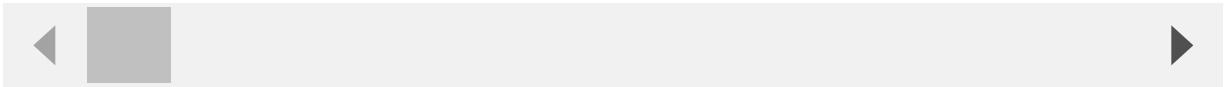
In [54]:

```
# Merge df_final (frequency and velocity change) and df_ds (days-since)
df_all_vars = pd.merge(df_final, df_ds, on=list(df_var.columns))
df_all_vars
```

Out[54]:

	Recnum	Cardnum	Date	Merchnum	Merch description	Merch state	Merch zip	Transtype	A
0	1	5142190439	2010-01-01	5509006296254	FEDEX SHP 12/23/09 AB#	TN	38118	P	
1	2	5142183973	2010-01-01	61003026333	SERVICE MERCHANDISE #81	MA	1803	P	
2	3	5142131721	2010-01-01	4503082993600	OFFICE DEPOT #191	MD	20706	P	
3	4	5142148452	2010-01-01	5509006296254	FEDEX SHP 12/28/09 AB#	TN	38118	P	
4	5	5142190439	2010-01-01	5509006296254	FEDEX SHP 12/23/09 AB#	TN	38118	P	
...	...	...	...	...	...	...	...	...	
96392	96749	5142276053	2010-12-31	3500000006160	BEST BUY 00001610	KY	41042	P	
96393	96750	5142225701	2010-12-31	8090710030950	MARKUS OFFICE SUPPLIES	OH	45248	P	
96394	96751	5142226486	2010-12-31	4503057341100	TECH PAC, INC	OH	45150	P	
96395	96752	5142244619	2010-12-31	8834000695412	BUY.COM	CA	92656	P	2
96396	96753	5142243247	2010-12-31	9108347680006	STAPLES NATIONAL #471	NJ	7606	P	

96397 rows × 78 columns



In [ ]:

```
# Save the merged dataframe (this is missing the amount variables)
# df_all_vars.to_csv('df_noamount_vars.csv')
```

In [55]:

```
# Optimize Function
def optimize_floats(df: pd.DataFrame) -> pd.DataFrame:
    floats = df.select_dtypes(include=['float64']).columns.tolist()
    df[floats] = df[floats].apply(pd.to_numeric, downcast='float')
    return df


def optimize_ints(df: pd.DataFrame) -> pd.DataFrame:
    ints = df.select_dtypes(include=['int64']).columns.tolist()
    df[ints] = df[ints].apply(pd.to_numeric, downcast='integer')
    return df


def optimize_objects(df: pd.DataFrame, datetime_features: List[str]) -> pd.DataFrame:
    for col in df.select_dtypes(include=['object']):
        if col not in datetime_features:
            num_unique_values = len(df[col].unique())
            num_total_values = len(df[col])
            if float(num_unique_values) / num_total_values < 0.5:
                df[col] = df[col].astype('category')
        else:
            df[col] = pd.to_datetime(df[col])
    return df



def optimize(df: pd.DataFrame, datetime_features: List[str] = []):
    return optimize_floats(optimize_ints(optimize_objects(df, datetime_features)))
```

Amount Velocity Variables

The number of records (transactions) in a given time period

In [56]:

```
# Make variable combos from the Lesson 10 slides (slide 40)
main_dataset_filled = deepcopy(df_impute_final)
main_dataset_filled['Cardnum_|_merchnum'] = main_dataset_filled['Cardnum'].astype('str') +
main_dataset_filled['Merchnum'].astype('str') # Class Q&A said not to use Merch descriptio
n
main_dataset_filled['Cardnum_|_zip'] = main_dataset_filled['Cardnum'].astype('str') + main
_dataset_filled['Merch zip'].astype('str')
main_dataset_filled['Cardnum_|_state'] = main_dataset_filled['Cardnum'].astype('str') + ma
in_dataset_filled['Merch state'].astype('str')
```

In [57]:

```
var_combos = ['Cardnum','Merchnum','Cardnum_|_zip','Cardnum_|_state','Cardnum_|_merchnum']
```

In [61]:

```
print('Memory percentage used',psutil.virtual_memory().percent)
```

```
Memory percentage used 18.0
```

In [64]:

```
time_joined =['join_ts1']
for num in time_list:
    time_joined.append('join_ts2_'+str(num))
```

In [71]:

```
# Making time variables
start=pd.datetime.now()

time_list = [0,1,3,7,14,30]
main_dataset_filled['join_ts1']=main_dataset_filled['Date']
# dt_i
for dt_i in time_list:
    time = 'join_ts2_'+str(dt_i)
    main_dataset_filled[time]=main_dataset_filled['Date'] + dt.timedelta(dt_i)
print('Completed ',pd.datetime.now()-start)
```

```
Completed  0:00:00.015444
```

In [73]:

```
df_var1 = main_dataset_filled.copy()
df_var2 = main_dataset_filled.copy()
```

In [ ]:

```
# 'Cardnum_|_zip_totalamount14_Amount2' = total amount for the past 14 days
# for the same cardnumber and zip
```

Total Amount Variables

The total amount of the number of records (transactions) in a given time period

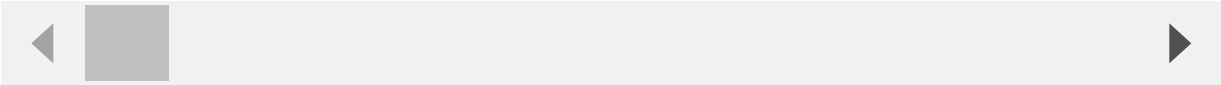
In [74]:

```
start_loop2=pd.datetime.now()
df_final = deepcopy(main_dataset_filled.set_index('Recnum'))

for item in var_combos:
    df_var3 = df_var1[['Recnum','Date','Amount',item]]
    temp_list = time_joined + [item]
    df_var4 = df_var2[temp_list + ['Recnum','Amount']].copy()
    df_var4.rename(columns={'Recnum':'Recnum2','Amount':'Amount2'},inplace=True) # this ca
uses a warning to arise
#     df_var4['record2'] = df_var2['record'] # this causes a warning to arise

    df_temp = pd.merge(df_var3, df_var4, left_on=[item], right_on=[item])

    for time in time_list:
        temp_endTime = 'join_ts2_' + str(time)
#         df2_temp = df_temp[(df_temp['date'] >= df_temp['join_ts1']) & (df_temp['date'] <
= df_temp[temp_endTime])] # Original from TA
        df2_temp = df_temp[(df_temp['Date'] <= df_temp[temp_endTime]) & (df_temp['Recnum2'
] <= df_temp['Recnum'])]

        temp_groupby = df2_temp[['Recnum','Amount2']].groupby('Recnum')

        temp_name = item + '_' + 'totalamount' + str(time) + '_'
        df_final = pd.merge(df_final, getattr(temp_groupby,'sum')().add_prefix(temp_name),
left_index=True, right_index=True, how='left')
#         break
#     break
print('second loop', pd.datetime.now()-start_loop2)
print(len(df_final.columns))
df_final.head()
```

```
second loop 0:00:28.784288
49
```

Out[74]:

Recnum	Cardnum	Date	Merchnum	Merch description	Merch state	Merch zip	Transtype	Amount
1	5142190439	2010-01-01	5509006296254	FEDEX SHP 12/23/09 AB#	TN	38118	P	3.62
2	5142183973	2010-01-01	61003026333	SERVICE MERCHANDISE #81	MA	1803	P	31.42
3	5142131721	2010-01-01	4503082993600	OFFICE DEPOT #191	MD	20706	P	178.49
4	5142148452	2010-01-01	5509006296254	FEDEX SHP 12/28/09 AB#	TN	38118	P	3.62
5	5142190439	2010-01-01	5509006296254	FEDEX SHP 12/23/09 AB#	TN	38118	P	3.62



In [75]:

```
print('Memory percentage used',psutil.virtual_memory().percent)
```

```
Memory percentage used 16.9
```

Median Amount variables

The median amount of the number of records (transactions) in a given time period

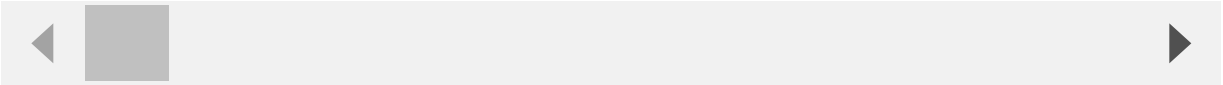
In [76]:

```
start_loop2=pd.datetime.now()
# df_final = deepcopy(main_dataset_filled.set_index('Recnum'))

for item in var_combos:
    df_var3 = df_var1[['Recnum','Date','Amount',item]]
    temp_list = time_joined + [item]
    df_var4 = df_var2[temp_list + ['Recnum','Amount']].copy()
    df_var4.rename(columns={'Recnum':'Recnum2','Amount':'Amount2'},inplace=True) # this ca
uses a warning to arise
#     df_var4['record2'] = df_var2['record'] # this causes a warning to arise

    df_temp = pd.merge(df_var3, df_var4, left_on=[item], right_on=[item])

    for time in time_list:
        temp_endTime = 'join_ts2_' + str(time)
#         df2_temp = df_temp[(df_temp['date'] >= df_temp['join_ts1']) & (df_temp['date'] <
= df_temp[temp_endTime])] # Original from TA
        df2_temp = df_temp[(df_temp['Date'] <= df_temp[temp_endTime]) & (df_temp['Recnum2'
] <= df_temp['Recnum'])]

        temp_groupby = df2_temp[['Recnum','Amount2']].groupby('Recnum')

        temp_name = item + '_' + 'median' + str(time) + '_'
        df_final = pd.merge(df_final, getattr(temp_groupby,'median')().add_prefix(temp_nam
e), left_index=True, right_index=True, how='left')
#         break
#     break
print('second loop', pd.datetime.now()-start_loop2)
print(len(df_final.columns))
df_final.head()
```

```
second loop 0:00:28.613825
79
```

Out[76]:

Recnum	Cardnum	Date	Merchnum	Merch description	Merch state	Merch zip	Transtype	Amount
1	5142190439	2010-01-01	5509006296254	FEDEX SHP 12/23/09 AB#	TN	38118	P	3.62
2	5142183973	2010-01-01	61003026333	SERVICE MERCHANDISE #81	MA	1803	P	31.42
3	5142131721	2010-01-01	4503082993600	OFFICE DEPOT #191	MD	20706	P	178.49
4	5142148452	2010-01-01	5509006296254	FEDEX SHP 12/28/09 AB#	TN	38118	P	3.62
5	5142190439	2010-01-01	5509006296254	FEDEX SHP 12/23/09 AB#	TN	38118	P	3.62



In [77]:

```
print('Memory percentage used',psutil.virtual_memory().percent)
```

```
Memory percentage used 17.0
```

In [ ]:

```
# df_final = optimize(df_final,['Date'])
```

Mean Amount Variables

The average amount of the number of records (transactions) in a given time period

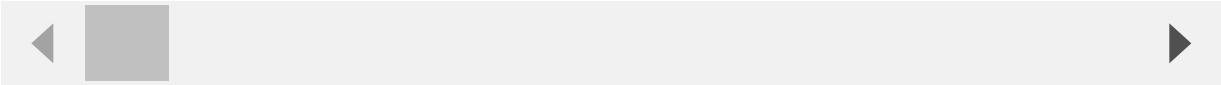
In [78]:

```
start_loop2=pd.datetime.now()
# df_final = deepcopy(main_dataset_filled.set_index('Recnum'))

for item in var_combos:
    df_var3 = df_var1[['Recnum','Date','Amount',item]]
    temp_list = time_joined + [item]
    df_var4 = df_var2[temp_list + ['Recnum','Amount']].copy()
    df_var4.rename(columns={'Recnum':'Recnum2','Amount':'Amount2'},inplace=True) # this ca
uses a warning to arise
#     df_var4['record2'] = df_var2['record'] # this causes a warning to arise

    df_temp = pd.merge(df_var3, df_var4, left_on=[item], right_on=[item])

    for time in time_list:
        temp_endTime = 'join_ts2_' + str(time)
#         df2_temp = df_temp[(df_temp['date'] >= df_temp['join_ts1']) & (df_temp['date'] <
= df_temp[temp_endTime])] # Original from TA
        df2_temp = df_temp[(df_temp['Date'] <= df_temp[temp_endTime]) & (df_temp['Recnum2'
] <= df_temp['Recnum'])]

        temp_groupby = df2_temp[['Recnum','Amount2']].groupby('Recnum')

        temp_name = item + '_' + 'mean' + str(time) + '_'
        df_final = pd.merge(df_final, getattr(temp_groupby,'mean')().add_prefix(temp_name
), left_index=True, right_index=True, how='left')
#         break
#     break
print('second loop', pd.datetime.now()-start_loop2)
print(len(df_final.columns))
df_final.head()
```

```
second loop 0:00:28.191120
109
```

Out[78]:

Recnum	Cardnum	Date	Merchnum	Merch description	Merch state	Merch zip	Transtype	Amount
1	5142190439	2010-01-01	5509006296254	FEDEX SHP 12/23/09 AB#	TN	38118	P	3.62
2	5142183973	2010-01-01	61003026333	SERVICE MERCHANDISE #81	MA	1803	P	31.42
3	5142131721	2010-01-01	4503082993600	OFFICE DEPOT #191	MD	20706	P	178.49
4	5142148452	2010-01-01	5509006296254	FEDEX SHP 12/28/09 AB#	TN	38118	P	3.62
5	5142190439	2010-01-01	5509006296254	FEDEX SHP 12/23/09 AB#	TN	38118	P	3.62



In [79]:

```
print('Memory percentage used',psutil.virtual_memory().percent)
```

```
Memory percentage used 17.1
```

MAX Amount Variables

The maximum amount of the number of records (transactions) in a given time period

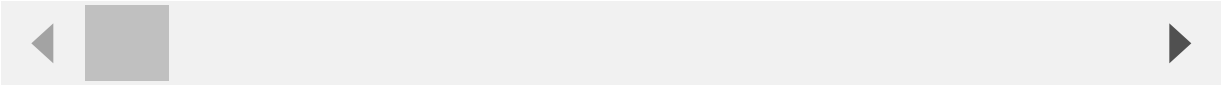
In [80]:

```
start_loop2=pd.datetime.now()
# df_final = deepcopy(main_dataset_filled.set_index('Recnum'))

for item in var_combos:
    df_var3 = df_var1[['Recnum','Date','Amount',item]]
    temp_list = time_joined + [item]
    df_var4 = df_var2[temp_list + ['Recnum','Amount']].copy()
    df_var4.rename(columns={'Recnum':'Recnum2','Amount':'Amount2'},inplace=True) # this ca
uses a warning to arise
#     df_var4['record2'] = df_var2['record'] # this causes a warning to arise

    df_temp = pd.merge(df_var3, df_var4, left_on=[item], right_on=[item])

    for time in time_list:
        temp_endTime = 'join_ts2_' + str(time)
#         df2_temp = df_temp[(df_temp['date'] >= df_temp['join_ts1']) & (df_temp['date'] <
= df_temp[temp_endTime])] # Original from TA
        df2_temp = df_temp[(df_temp['Date'] <= df_temp[temp_endTime]) & (df_temp['Recnum2'
] <= df_temp['Recnum'])]

        temp_groupby = df2_temp[['Recnum','Amount2']].groupby('Recnum')

        temp_name = item + '_' + 'max' + str(time) + '_'
        df_final = pd.merge(df_final, getattr(temp_groupby,'max')().add_prefix(temp_name),
left_index=True, right_index=True, how='left')
#         break
#     break
print('second loop', pd.datetime.now()-start_loop2)
print(len(df_final.columns))
df_final.head()
```

```
second loop 0:00:27.696463
139
```

Out[80]:

Recnum	Cardnum	Date	Merchnum	Merch description	Merch state	Merch zip	Transtype	Amount
1	5142190439	2010-01-01	5509006296254	FEDEX SHP 12/23/09 AB#	TN	38118	P	3.62
2	5142183973	2010-01-01	61003026333	SERVICE MERCHANDISE #81	MA	1803	P	31.42
3	5142131721	2010-01-01	4503082993600	OFFICE DEPOT #191	MD	20706	P	178.49
4	5142148452	2010-01-01	5509006296254	FEDEX SHP 12/28/09 AB#	TN	38118	P	3.62
5	5142190439	2010-01-01	5509006296254	FEDEX SHP 12/23/09 AB#	TN	38118	P	3.62



In [81]:

```
print('Memory percentage used',psutil.virtual_memory().percent)
```

```
Memory percentage used 17.2
```

In [82]:

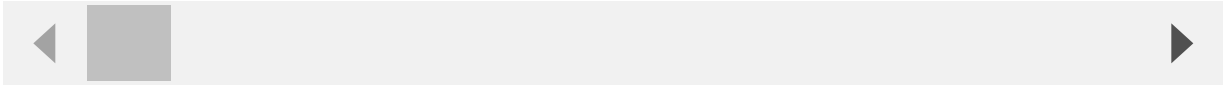
```
df_final = optimize(df_final,['Date'])
```

In [83]:

```
df_final.head()
```

Out[83]:

Recnum	Cardnum	Date	Merchnum	Merch description	Merch state	Merch zip	Transtype	Amou
1	5142190439	2010-01-01	5509006296254	FEDEX SHP 12/23/09 AB#	TN	38118	P	3.6200
2	5142183973	2010-01-01	61003026333	SERVICE MERCHANDISE #81	MA	1803	P	31.4200
3	5142131721	2010-01-01	4503082993600	OFFICE DEPOT #191	MD	20706	P	178.4900
4	5142148452	2010-01-01	5509006296254	FEDEX SHP 12/28/09 AB#	TN	38118	P	3.6200
5	5142190439	2010-01-01	5509006296254	FEDEX SHP 12/23/09 AB#	TN	38118	P	3.6200



In [87]:

```
start_column = df_final.columns.get_loc('Cardnum_totalamount0_Amount2')
for col_names in df_final.columns[start_column:]:
#     print(col_names)
#  the number of days in each column is the column index mod length of time array
    number_index = time_list[(df_final.columns.get_loc(col_names)%len(time_list))-1]
#     we split the name of the array to keep the first part and use it to create the new c
olumn names
    new_col_name = col_names.split(str(number_index))[0]+'_'+str(number_index)+'_actual'
#     print(new_col_name)
    df_final[new_col_name] = df_final["Amount"]/df_final[col_names]
```

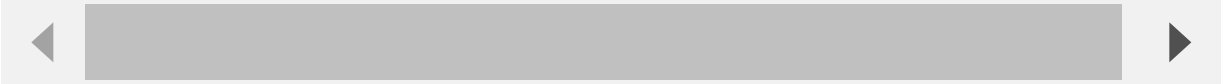
verification of amount variables

In [89]:

```
# df_final['Merchnum'][8]
df_final[['Merchnum','Date','Amount','Merchnum_max7_Amount2','Merchnum_max_7_actual']].loc
[df_final['Merchnum']=='6098208200062']
# np.array([df_final.columns == 'Merchnum_max_7_actual']).sum()
# df_final['Merch']
```

Out[89]:

Recnum	Merchnum	Date	Amount	Merchnum_max7_Amount2	Merchnum_max_7_actua
8	6098208200062	2010-01-01	230.320007	230.320007	1.000000
521	6098208200062	2010-01-05	267.220001	267.220001	1.000000
5052	6098208200062	2010-01-24	140.149994	140.149994	1.000000
25240	6098208200062	2010-04-06	685.619995	685.619995	1.000000
28338	6098208200062	2010-04-18	735.369995	735.369995	1.000000
33190	6098208200062	2010-05-04	10.850000	10.850000	1.000000
33926	6098208200062	2010-05-08	9.300000	10.850000	0.857143
54055	6098208200062	2010-07-18	4749.169922	4749.169922	1.000000
56826	6098208200062	2010-07-26	554.940002	554.940002	1.000000
62561	6098208200062	2010-08-11	24.170000	24.170000	1.000000
74143	6098208200062	2010-09-15	200.000000	200.000000	1.000000
80885	6098208200062	2010-10-11	1367.380005	1367.380005	1.000000
82219	6098208200062	2010-10-19	1395.000000	1395.000000	1.000000
82321	6098208200062	2010-10-20	669.330017	1395.000000	0.479806
84902	6098208200062	2010-11-03	284.089996	284.089996	1.000000
94433	6098208200062	2010-12-17	667.179993	667.179993	1.000000
96194	6098208200062	2010-12-28	4950.000000	4950.000000	1.000000
96403	6098208200062	2010-12-29	263.320007	4950.000000	0.053196



Explanation of column name:

Cardnum_|_state_max_30*Amount2 = max amount at this card number in this state over the past 30 days*
Cardnum|_state_max_30_actual = actual amount/ max at this card number in this state over the past 30 days

In [81]:

```
print('Memory percentage used',psutil.virtual_memory().percent)
```

```
Memory percentage used 17.2
```

Amount Velocity Change Variables

In [90]:

```
# iterates through  the variables created with the var_combos columns and finds the relati
ve
# velocity variables. Divides the number of days of days_numer array with the days_denom o
ne
start_loop3=pd.datetime.now()
groupbyvar_denom = deepcopy(var_combos)
days_numer = ['0','1']
days_denom = ['7','14','30']
# Cardnum_totalamount0_Amount2_/_Cardnum_totalamount3_Amount2
for b in groupbyvar_denom:
    for c in days_numer:
        for d in days_denom:
            temp = d
            df_final[b + '_' + c + '_dayamount' + '_div_' + d + '_dayamount' + '_velchang
e'] = \
                df_final[b+'_totalamount'+c+'_Amount2'] / \
                df_final[b+'_totalamount'+d+'_Amount2'] / float(temp)
print('third loop', pd.datetime.now() - start_loop3)
```

```
third loop 0:00:00.041360
```

Create Benford's Law variables

In [ ]:

```
# remove FEDEX first only for this part
```

In [91]:

```
# main_dataset_filled.head()
applic_ds_df = main_dataset_filled[['Recnum','Cardnum','Date','Merchnum','Merch descriptio
n','Amount','Fraud']]
# add column with all ones
applic_ds_df["Cardnum_U*"] = np.ones(applic_ds_df.shape[0])
applic_ds_df["Merchnum_U*"] = np.ones(applic_ds_df.shape[0])
applic_ds_df
```

```
/home/thanos/miniconda3/envs/dev/lib/python3.6/site-packages/ipykernel_launch
er.py:4: SettingWithCopyWarning:
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/user_guide/indexing.html#returning-a-view-versus-a-copy
  after removing the cwd from sys.path.
/home/thanos/miniconda3/envs/dev/lib/python3.6/site-packages/ipykernel_launch
er.py:5: SettingWithCopyWarning:
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/user_guide/indexing.html#returning-a-view-versus-a-copy
  """
```

Out[91]:

	Recnum	Cardnum	Date	Merchnum	Merch description	Amount	Fraud	Cardnum_U
0	1	5142190439	2010-01-01	5509006296254	FEDEX SHP 12/23/09 AB#	3.62	0	1.
1	2	5142183973	2010-01-01	61003026333	SERVICE MERCHANDISE #81	31.42	0	1.
2	3	5142131721	2010-01-01	4503082993600	OFFICE DEPOT #191	178.49	0	1.
3	4	5142148452	2010-01-01	5509006296254	FEDEX SHP 12/28/09 AB#	3.62	0	1.
4	5	5142190439	2010-01-01	5509006296254	FEDEX SHP 12/23/09 AB#	3.62	0	1.
...	...	...	...	...	...	...	...	.
96748	96749	5142276053	2010-12-31	3500000006160	BEST BUY 00001610	84.79	0	1.
96749	96750	5142225701	2010-12-31	8090710030950	MARKUS OFFICE SUPPLIES	118.75	0	1.
96750	96751	5142226486	2010-12-31	4503057341100	TECH PAC, INC	363.56	0	1.
96751	96752	5142244619	2010-12-31	8834000695412	BUY.COM	2202.03	0	1.
96752	96753	5142243247	2010-12-31	9108347680006	STAPLES NATIONAL #471	554.64	0	1.

96397 rows × 9 columns

In [92]:

```
# remove FEDEX first only for this part
# drop fedex values
fed_ex_index = applic_ds_df['Merch description'].str.contains('FEDEX SHP')
print(fed_ex_index.sum())

applic_ds_df = applic_ds_df.drop(applic_ds_df.index[fed_ex_index])
applic_ds_df.shape
```

```
11767
```

Out[92]:

```
(84630, 9)
```

In [93]:

```
def get_N_high_arr(df):
    x = np.abs(df)
    e = np.floor(np.log10(x))
    msg_tst_arr = np.floor(x*10**-e).astype('int')
    hig_dig = np.sum(msg_tst_arr > 2)
    return np.maximum(hig_dig,1)
```

In [94]:

```
def get_Nlow_arr(df):
    x = np.abs(df)
    e = np.floor(np.log10(x))
    msg_tst_arr = np.floor(x*10**-e).astype('int')
    low_dig =   np.sum(msg_tst_arr <= 2)
    return np.maximum(low_dig,1)
```

In [95]:

```
def find_Benford_vars(df_name, col_name):
# this function creates benford's law variable for each different col_name entry, taking i
n
# consideration all the entries BEFORE time_index. Time index is ONLY the record number
# returns each variable in an new dataframe
# requires get_Nlow_arr & get_N_high_arr functions
#  example card_benford_vars_df = find_Benford_vars(card_df,'Cardnum')

    uniq_cards = df_name[col_name].unique()
    new_df = pd.DataFrame({col_name:[],'n_low':[],'n_high':[]})

    # find n high and n low for each entry
    start_time = pd.datetime.now()
    for card_no in uniq_cards:
        n_low_count = get_Nlow_arr(df_name[df_name[col_name] == card_no]['Amount'].values)
        n_high_count = get_N_high_arr(df_name[df_name[col_name] == card_no]['Amount'].valu
es)
        new_df = new_df.append(pd.DataFrame.from_records([{col_name:card_no,'n_low':n_low_
count,'n_high':n_high_count}]), ignore_index=True)
    print('duration: ', pd.datetime.now() - start_time)

    # set any values with n_high or n_low = 0 to 1
#     new_df['n_low'] = np.where(new_df['n_low'] == 0,1,new_df['n_low'])
#     new_df['n_high'] = np.where(new_df['n_high'] == 0,1,new_df['n_high'])

    # find R and 1/R
    new_df['R'] = (new_df['n_low']*1.096)/new_df['n_high']
    new_df['1/R'] = 1/new_df['R']

    # find U = max(R,1/R)
    new_df['U'] = np.maximum(new_df['R'],new_df['1/R'])
    # find column n
    new_df['n'] = new_df['n_high'] + new_df['n_low']

#     smoothing formula
    n_mid = 15
    c = 3
    new_df['t'] = (new_df['n']-n_mid)/c
    new_df[col_name+'_U*'] = 1 + (new_df['U']-1)/(1+np.exp(-new_df['t']))
    new_df[['Recnum','Cardnum','n_low','n_high']] = new_df[['Recnum','Cardnum','n_low','n_
high']].astype('int')
    return new_df
```

In [96]:

```
main_dataset_filled.head()
applic_ds_df = main_dataset_filled[['Recnum','Cardnum','Date','Merchnum','Merch descriptio
n','Amount','Fraud']]
# add column with all ones
applic_ds_df["Cardnum_U*"] = np.ones(applic_ds_df.shape[0])
applic_ds_df["Merchnum_U*"] = np.ones(applic_ds_df.shape[0])
applic_ds_df
```

```
/home/thanos/miniconda3/envs/dev/lib/python3.6/site-packages/ipykernel_launch
er.py:4: SettingWithCopyWarning:
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/user_guide/indexing.html#returning-a-view-versus-a-copy
  after removing the cwd from sys.path.
/home/thanos/miniconda3/envs/dev/lib/python3.6/site-packages/ipykernel_launch
er.py:5: SettingWithCopyWarning:
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/user_guide/indexing.html#returning-a-view-versus-a-copy
  """
```

Out[96]:

	Recnum	Cardnum	Date	Merchnum	Merch description	Amount	Fraud	Cardnum_U
0	1	5142190439	2010-01-01	5509006296254	FEDEX SHP 12/23/09 AB#	3.62	0	1.
1	2	5142183973	2010-01-01	61003026333	SERVICE MERCHANDISE #81	31.42	0	1.
2	3	5142131721	2010-01-01	4503082993600	OFFICE DEPOT #191	178.49	0	1.
3	4	5142148452	2010-01-01	5509006296254	FEDEX SHP 12/28/09 AB#	3.62	0	1.
4	5	5142190439	2010-01-01	5509006296254	FEDEX SHP 12/23/09 AB#	3.62	0	1.
...	...	...	...	...	...	...	...	...
96748	96749	5142276053	2010-12-31	3500000006160	BEST BUY 00001610	84.79	0	1.
96749	96750	5142225701	2010-12-31	8090710030950	MARKUS OFFICE SUPPLIES	118.75	0	1.
96750	96751	5142226486	2010-12-31	4503057341100	TECH PAC, INC	363.56	0	1.
96751	96752	5142244619	2010-12-31	8834000695412	BUY.COM	2202.03	0	1.
96752	96753	5142243247	2010-12-31	9108347680006	STAPLES NATIONAL #471	554.64	0	1.

96397 rows × 9 columns

In [97]:

```
# drop fedex values
fed_ex_index = applic_ds_df['Merch description'].str.contains('FEDEX SHP')
print(fed_ex_index.sum())

applic_ds_df = applic_ds_df.drop(applic_ds_df.index[fed_ex_index])
applic_ds_df.shape
```

```
11767
```

Out[97]:

```
(84630, 9)
```

In [98]:

```
applic_ds_df
```

Out[98]:

	Recnum	Cardnum	Date	Merchnum	Merch description	Amount	Fraud	Cardn
1	2	5142183973	2010-01-01	61003026333	SERVICE MERCHANDISE #81	31.42	0	
2	3	5142131721	2010-01-01	4503082993600	OFFICE DEPOT #191	178.49	0	
7	8	5142191182	2010-01-01	6098208200062	MIAMI COMPUTER SUPPLY	230.32	0	
8	9	5142258629	2010-01-01	602608969534	FISHER SCI ATL	62.11	0	
13	14	5142124791	2010-01-01	5725000466504	CDW*GOVERNMENT INC	106.89	0	
...	...	...	...	...	...	...	...	
96748	96749	5142276053	2010-12-31	3500000006160	BEST BUY 00001610	84.79	0	
96749	96750	5142225701	2010-12-31	8090710030950	MARKUS OFFICE SUPPLIES	118.75	0	
96750	96751	5142226486	2010-12-31	4503057341100	TECH PAC, INC	363.56	0	
96751	96752	5142244619	2010-12-31	8834000695412	BUY.COM	2202.03	0	
96752	96753	5142243247	2010-12-31	9108347680006	STAPLES NATIONAL #471	554.64	0	

84630 rows × 9 columns

a) for Cardnum

In [99]:

```
start_loop2=pd.datetime.now()

for current_record_no in applic_ds_df['Recnum'].values:
#     print(current_record_no)
    current_df = applic_ds_df[applic_ds_df['Recnum'] <= current_record_no]
    # all credit card numbers untill that record number
#     current_df['Cardnum'].values
    # credit card number of current record number
    current_card_no = current_df['Cardnum'].values[-1]
#     smoothing formula
    n_mid = 15
    c = 3
    n_low_count = get_Nlow_arr(current_df[current_df['Cardnum'] == current_card_no]['Amoun
t'].values)
    n_high_count = get_N_high_arr(current_df[current_df['Cardnum'] == current_card_no]['Am
ount'].values)
#     print(n_low_count)
#     print(n_high_count)
    R = n_low_count*1.096/n_high_count
#     print('R:',R)
    U = np.maximum(R,(1/R))
#     print('U:',U)
    t = ((n_high_count+n_low_count)-n_mid)/c
    U_star = 1+(U-1)/(1+np.exp(-t))
#     print('U*:',U_star)
    set_index_row = current_df[current_df['Recnum'] == current_record_no].index[0]
    set_index_col = applic_ds_df.columns.get_loc('Cardnum_U*')
    applic_ds_df.loc[set_index_row,'Cardnum_U*'] = U_star
print('Cardnumber U*:', pd.datetime.now()-start_loop2)
```

```
Cardnumber U*: 0:09:09.313227
```

b) for Merchnum

In [100]:

```
start_loop2=pd.datetime.now()

for current_record_no in applic_ds_df['Recnum'].values:
#     print(current_record_no)
    current_df = applic_ds_df[applic_ds_df['Recnum'] <= current_record_no]
    current_card_no = current_df['Merchnum'].values[-1]
#     smoothing formula
    n_mid = 15
    c = 3
    n_low_count = get_Nlow_arr(current_df[current_df['Merchnum'] == current_card_no]['Amou
nt'].values)
    n_high_count = get_N_high_arr(current_df[current_df['Merchnum'] == current_card_no]['A
mount'].values)
#     print(n_low_count)
#     print(n_high_count)
    R = n_low_count*1.096/n_high_count
#     print('R:',R)
    U = np.maximum(R,(1/R))
#     print('U:',U)
    t = ((n_high_count+n_low_count)-n_mid)/c
    U_star = 1+(U-1)/(1+np.exp(-t))
#     print('U*:',U_star)
    set_index_row = current_df[current_df['Recnum'] == current_record_no].index[0]
    set_index_col = applic_ds_df.columns.get_loc('Merchnum_U*')
    applic_ds_df.loc[set_index_row,'Merchnum_U*'] = U_star
print('Merchnumber U*:', pd.datetime.now()-start_loop2)
```

```
Merchnumber U*: 0:14:34.893326
```

In [101]:

```
applic_ds_df
```

Out[101]:

	Recnum	Cardnum	Date	Merchnum	Merch description	Amount	Fraud	Cardn
1	2	5142183973	2010-01-01	61003026333	SERVICE MERCHANDISE #81	31.42	0	1.(
2	3	5142131721	2010-01-01	4503082993600	OFFICE DEPOT #191	178.49	0	1.(
7	8	5142191182	2010-01-01	6098208200062	MIAMI COMPUTER SUPPLY	230.32	0	1.(
8	9	5142258629	2010-01-01	602608969534	FISHER SCI ATL	62.11	0	1.(
13	14	5142124791	2010-01-01	5725000466504	CDW*GOVERNMENT INC	106.89	0	1.(
...	...	...	...	...	...	...	...	
96748	96749	5142276053	2010-12-31	3500000006160	BEST BUY 00001610	84.79	0	1.(
96749	96750	5142225701	2010-12-31	8090710030950	MARKUS OFFICE SUPPLIES	118.75	0	1.1
96750	96751	5142226486	2010-12-31	4503057341100	TECH PAC, INC	363.56	0	1.2
96751	96752	5142244619	2010-12-31	8834000695412	BUY.COM	2202.03	0	1.4
96752	96753	5142243247	2010-12-31	9108347680006	STAPLES NATIONAL #471	554.64	0	1.2

84630 rows × 9 columns

In [103]:

```
print('Memory percentage used',psutil.virtual_memory().percent)
```

```
Memory percentage used 17.1
```

Merge Benford's law with rest of the dataset

In [124]:

```
merge1_df = pd.DataFrame(df_final.reset_index())
merge2 = applic_ds_df[['Recnum','Cardnum_U*','Merchnum_U*']]
```

In [125]:

```
merge1_all = pd.merge(merge1_df,merge2,how='left',left_on='Recnum',right_on='Recnum')
```

In [126]:

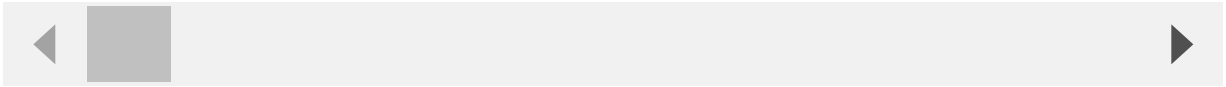
```
merge1_all['Cardnum_U*'] = merge1_all['Cardnum_U*'].fillna(1)
merge1_all['Merchnum_U*'] = merge1_all['Merchnum_U*'].fillna(1)
```

In [163]:

```
merge1_all.head()
```

Out[163]:

	Recnum	Cardnum	Date	Merchnum	Merch description	Merch state	Merch zip	Transtype	An
0	1	5142190439	2010-01-01	5509006296254	FEDEX SHP 12/23/09 AB#	TN	38118	P	3.6
1	2	5142183973	2010-01-01	61003026333	SERVICE MERCHANDISE #81	MA	1803	P	31.4
2	3	5142131721	2010-01-01	4503082993600	OFFICE DEPOT #191	MD	20706	P	178.4
3	4	5142148452	2010-01-01	5509006296254	FEDEX SHP 12/28/09 AB#	TN	38118	P	3.6
4	5	5142190439	2010-01-01	5509006296254	FEDEX SHP 12/23/09 AB#	TN	38118	P	3.6



In [180]:

```
df_all_vars = optimize(df_all_vars)
```

Merge all variables

In oder to merge properly make sure all the feature data types are consistent

In [192]:

```
# df_all_vars contains frequency variables, velocity change variables, and days since vari
ables
# merge1_df contains amount variables, velocity change amount variables, and benford's law
variables
df_all_vars_final = pd.merge(df_all_vars,merge1_all,how='left',left_on=df_original.columns
.to_list(),right_on=df_original.columns.to_list())
```

In [193]:

```
df_all_vars_final = df_all_vars_final.drop(columns=['Cardnum_|_merchnum',
       'Cardnum_|_zip', 'Cardnum_|_state', 'join_ts1', 'join_ts2_0',
       'join_ts2_1', 'join_ts2_3', 'join_ts2_7', 'join_ts2_14',
       'join_ts2_30'])
df_all_vars_final.head()
```

Out[193]:

	Recnum	Cardnum	Date	Merchnum	Merch description	Merch state	Merch zip	Transtype	An
0	1	5142190439	2010-01-01	5509006296254	FEDEX SHP 12/23/09 AB#	TN	38118	P	3.6
1	2	5142183973	2010-01-01	61003026333	SERVICE MERCHANDISE #81	MA	1803	P	31.4
2	3	5142131721	2010-01-01	4503082993600	OFFICE DEPOT #191	MD	20706	P	178.4
3	4	5142148452	2010-01-01	5509006296254	FEDEX SHP 12/28/09 AB#	TN	38118	P	3.6
4	5	5142190439	2010-01-01	5509006296254	FEDEX SHP 12/23/09 AB#	TN	38118	P	3.6

In [186]:

```
df_all_vars_final.shape
```

Out[186]:

```
(96397, 350)
```

Risk Table

In [143]:

```
#convert 'Date' to day of week and fill that into a newly created column called 'Day_of_we
ek'
mydata = deepcopy(df_original)
mydata['Day_of_week'] = mydata['Date'].dt.weekday_name
mydata.head(20)
```

Out[143]:

	Recnum	Cardnum	Date	Merchnum	Merch description	Merch state	Merch zip	Transtype
0	1	5142190439	2010-01-01	5509006296254	FEDEX SHP 12/23/09 AB#	TN	38118.0	P
1	2	5142183973	2010-01-01	61003026333	SERVICE MERCHANDISE #81	MA	1803.0	P
2	3	5142131721	2010-01-01	4503082993600	OFFICE DEPOT #191	MD	20706.0	P
3	4	5142148452	2010-01-01	5509006296254	FEDEX SHP 12/28/09 AB#	TN	38118.0	P
4	5	5142190439	2010-01-01	5509006296254	FEDEX SHP 12/23/09 AB#	TN	38118.0	P
5	6	5142149874	2010-01-01	5509006296254	FEDEX SHP 12/22/09 AB#	TN	38118.0	P
6	7	5142189277	2010-01-01	5509006296254	FEDEX SHP 12/28/09 AB#	TN	38118.0	P
7	8	5142191182	2010-01-01	6098208200062	MIAMI COMPUTER SUPPLY	OH	45429.0	P
8	9	5142258629	2010-01-01	602608969534	FISHER SCI ATL	GA	30091.0	P
9	10	5142190439	2010-01-01	5509006296254	FEDEX SHP 12/23/09 AB#	TN	38118.0	P
10	11	5142283088	2010-01-01	5509006296254	FEDEX SHP 12/22/09 AB#	TN	38118.0	P
11	12	5142190439	2010-01-01	5509006296254	FEDEX SHP 12/23/09 AB#	TN	38118.0	P
12	13	5142146134	2010-01-01	5509006296254	FEDEX SHP 12/21/09 AB#	TN	38118.0	P
13	14	5142124791	2010-01-01	5725000466504	CDW*GOVERNMENT INC	IL	60061.0	P
14	15	5142151402	2010-01-01	5725000466504	CDW*GOVERNMENT INC	IL	60061.0	P
15	16	5142148452	2010-01-01	5509006296254	FEDEX SHP 12/22/09 AB#	TN	38118.0	P
16	17	5142148452	2010-01-01	5509006296254	FEDEX SHP 12/28/09 AB#	TN	38118.0	P
17	18	5142125684	2010-01-01	5509006296254	FEDEX SHP 12/22/09 AB#	TN	38118.0	P
18	19	5142139349	2010-01-01	5509006296254	FEDEX SHP 12/23/09 AB#	TN	38118.0	P
19	20	5142190596	2010-01-01	602608969284	FISHER SCI HUS	TX	77251.0	P

In [144]:

```
#take out the last two months data before buidling risk table
mydata_10= mydata[mydata['Date'] < '2010-11-01']
```

In [145]:

```
#calculate the average fraud rate for each day of week
mydata_10.groupby(['Day_of_week']).mean()
```

Out[145]:

Day_of_week	Recnum	Cardnum	Merch zip	Amount	Fraud
Friday	52961.678147	5.142199e+09	44059.295347	372.752805	0.025986
Monday	41448.131426	5.142203e+09	45057.737461	388.056109	0.008680
Saturday	45853.643038	5.142204e+09	44115.570858	387.049779	0.010040
Sunday	39681.191029	5.142204e+09	43751.027994	388.774515	0.009588
Thursday	38880.644673	5.142200e+09	44978.431283	346.161248	0.018614
Tuesday	42454.089730	5.142202e+09	44968.096435	580.388743	0.007095
Wednesday	41982.279746	5.142203e+09	44991.314593	456.200272	0.009743

In [146]:

```
#count how many instances are there for each day of week
num_instances_day_of_week=mydata_10.groupby('Day_of_week').size()
num_instances_day_of_week
```

Out[146]:

```
Day_of_week
Friday        3194
Monday       16359
Saturday      9861
Sunday       14914
Thursday      7790
Tuesday      17196
Wednesday    14985
dtype: int64
```

In [147]:

```
#use smoothing method if necessary (here actually not necessary)
c=4
nmid=20
fraud_rate_avg=mydata_10['Fraud'].mean()
fraud_rate_day_of_week=mydata_10.groupby('Day_of_week')['Fraud'].mean()
```

In [148]:

```
#smoothing method
fraud_rate_day_of_week_smooth=fraud_rate_avg+(fraud_rate_day_of_week-fraud_rate_avg)/(1+np
.exp(-(num_instances_day_of_week-nmid)/c))
fraud_rate_day_of_week_smooth
```

Out[148]:

```
Day_of_week
Friday       0.025986
Monday       0.008680
Saturday     0.010040
Sunday       0.009588
Thursday     0.018614
Tuesday      0.007095
Wednesday    0.009743
dtype: float64
```

In [149]:

```
#build a column called 'Day_of_week_smooth' and fill in it with the smoothed result
mydata_10['Day_of_week_smooth']=mydata_10['Day_of_week'].map(fraud_rate_day_of_week_smooth
)
```

```
/home/thanos/miniconda3/envs/dev/lib/python3.6/site-packages/ipykernel_launch
er.py:2: SettingWithCopyWarning:
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/user_guide/indexing.html#returning-a-view-versus-a-copy
```

In [150]:

```
mydata_10
```

Out[150]:

	Recnum	Cardnum	Date	Merchnum	Merch description	Merch state	Merch zip	Transtype
0	1	5142190439	2010-01-01	5509006296254	FEDEX SHP 12/23/09 AB#	TN	38118.0	P
1	2	5142183973	2010-01-01	61003026333	SERVICE MERCHANDISE #81	MA	1803.0	P
2	3	5142131721	2010-01-01	4503082993600	OFFICE DEPOT #191	MD	20706.0	P
3	4	5142148452	2010-01-01	5509006296254	FEDEX SHP 12/28/09 AB#	TN	38118.0	P
4	5	5142190439	2010-01-01	5509006296254	FEDEX SHP 12/23/09 AB#	TN	38118.0	P
...	...	...	...	...	...	...	...	...
84294	84295	5142130739	2010-10-31	9108347680000	STAPLES NATIONAL #471	NJ	7606.0	P
84295	84296	5142219772	2010-10-31	6855293370648	PRESTIGE OFFICE PRODUCTS	NC	27705.0	P
84296	84297	5142257707	2010-10-31	300025852	AMERICAN SOCIETY OF AG	WI	53711.0	P
84297	84298	5142168022	2010-10-31	607900047334	MYSTIC LAKE CASINO	MN	55372.0	P
84298	84299	5142137416	2010-10-31	9108347680000	STAPLES NATIONAL #471	NJ	7606.0	P

84299 rows × 12 columns

In [151]:

```
#drop 'Day_of_week'
mydata_10.drop(['Day_of_week'], axis=1, inplace=True)
```

```
/home/thanos/miniconda3/envs/dev/lib/python3.6/site-packages/pandas/core/fram
e.py:4117: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  errors=errors,
```

In [159]:

```
#rename 'Day_of_week' with 'Day_of_week_smooth'
mydata_10.rename(columns={'Day_of_week_smooth': 'Day_of_week'}, inplace=True)
```

In [154]:

```
#drop index column
# mydata_10a=mydata_10.drop(mydata_10.columns[0], axis = 1)
```

In [189]:

```
mydata_10 =  optimize(mydata_10)
```

```
/home/thanos/miniconda3/envs/dev/lib/python3.6/site-packages/ipykernel_launch
er.py:20: SettingWithCopyWarning:
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/user_guide/indexing.html#returning-a-view-versus-a-copy
/home/thanos/miniconda3/envs/dev/lib/python3.6/site-packages/pandas/core/fram
e.py:3509: SettingWithCopyWarning:
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/user_guide/indexing.html#returning-a-view-versus-a-copy
  self[k1] = value[k2]
```

Merge Day of week risk variable with all the variables

In [194]:

```
df_all_vars_final = pd.merge(df_all_vars_final,mydata_10,how='left',left_on=df_original.co
lumns.to_list(),right_on=df_original.columns.to_list())
```

In [195]:

```
df_all_vars_final.head()
```

Out[195]:

	Recnum	Cardnum	Date	Merchnum	Merch description	Merch state	Merch zip	Transtype	An
0	1	5142190439	2010-01-01	5509006296254	FEDEX SHP 12/23/09 AB#	TN	38118	P	3.6
1	2	5142183973	2010-01-01	61003026333	SERVICE MERCHANDISE #81	MA	1803	P	31.4
2	3	5142131721	2010-01-01	4503082993600	OFFICE DEPOT #191	MD	20706	P	178.4
3	4	5142148452	2010-01-01	5509006296254	FEDEX SHP 12/28/09 AB#	TN	38118	P	3.6
4	5	5142190439	2010-01-01	5509006296254	FEDEX SHP 12/23/09 AB#	TN	38118	P	3.6

Feature Selection

Filter Method

Import the consolidated dataset from the share drive and then find the univariate FDR @ 3% rank and univariate KS rank. Calculate the average rank and keep about 80 variables.

In [ ]:

```
# df_filter = pd.read_csv('.csv', parse_dates=['Date'])
# df_filter.info()
```

In [ ]:

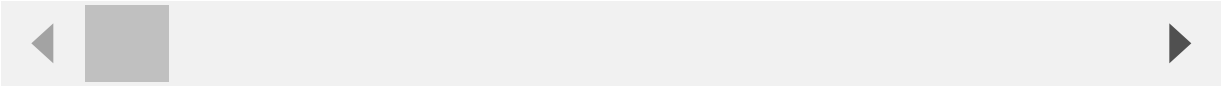
```
df_filter = df_all_vars.copy()
```

In [ ]:

```
df_filter.head()
```

Out[ ]:

	Recnum	Cardnum	Date	Merchnum	Merch description	Merch state	Merch zip	Transtype	Amou
0	1	5142190439	2010-01-01	5509006296254	FEDEX SHP 12/23/09 AB#	TN	38118	P	3.
1	2	5142183973	2010-01-01	61003026333	SERVICE MERCHANDISE #81	MA	1803	P	31.
2	3	5142131721	2010-01-01	4503082993600	OFFICE DEPOT #191	MD	20706	P	178.
3	4	5142148452	2010-01-01	5509006296254	FEDEX SHP 12/28/09 AB#	TN	38118	P	3.
4	5	5142190439	2010-01-01	5509006296254	FEDEX SHP 12/23/09 AB#	TN	38118	P	3.



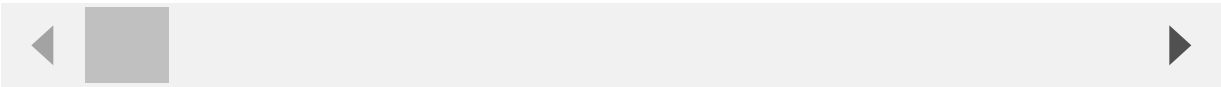
Z-Scale All Variables

In [ ]:

```
df_filter.iloc[:,13:].head()
```

Out[ ]:

	Cardnum_freq0_Date	Cardnum_freq1_Date	Cardnum_freq3_Date	Cardnum_freq7_Date	Cardnu
0	1	1	1	1	
1	1	1	1	1	
2	1	1	1	1	
3	1	1	1	1	
4	2	2	2	2	



In [ ]:

```
# Get only the numerical variables
df_numerical = pd.concat([df_filter[['Recnum','Date','Fraud']],df_filter.iloc[:,13:]], axis=1)
```

In [ ]:

```
# Add a variable full of random numbers
df_numerical.insert(3, 'rand_num', random.sample(range(1, 4000000), len(df_numerical)))
df_numerical.head()
```

Out[ ]:

	Recnum	Date	Fraud	rand_num	Cardnum_freq0_Date	Cardnum_freq1_Date	Cardnum_freq3_
0	1	2010-01-01	0	553074	1	1	
1	2	2010-01-01	0	1142716	1	1	
2	3	2010-01-01	0	111835	1	1	
3	4	2010-01-01	0	233030	1	1	
4	5	2010-01-01	0	1442850	2	2	

In [ ]:

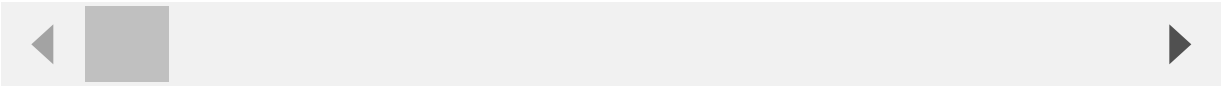
```
# z-scale the numerical variables
df_numerical_zscale = df_numerical.iloc[:,2:].apply(zscore)
df_numerical_zscale['Fraud'] = df_numerical['Fraud']
df_numerical_zscale.head(100)
```

Out[ ]:

	Fraud	rand_num	Cardnum_freq0_Date	Cardnum_freq1_Date	Cardnum_freq3_Date	Cardnum
0	0	-1.255202	-0.245524	-0.297938	-0.331377	
1	0	-0.745087	-0.245524	-0.297938	-0.331377	
2	0	-1.636929	-0.245524	-0.297938	-0.331377	
3	0	-1.532080	-0.245524	-0.297938	-0.331377	
4	0	-0.485434	-0.078915	-0.172072	-0.244041	
5	0	1.485242	-0.245524	-0.297938	-0.331377	
6	0	-0.275929	-0.245524	-0.297938	-0.331377	
7	0	-1.236067	-0.245524	-0.297938	-0.331377	
8	0	-1.089684	-0.245524	-0.297938	-0.331377	
9	0	0.982509	0.087694	-0.046206	-0.156705	
10	0	1.425656	-0.245524	-0.297938	-0.331377	
11	0	1.624651	0.254302	0.079660	-0.069368	
12	0	-0.074081	-0.245524	-0.297938	-0.331377	
13	0	-0.895060	-0.245524	-0.297938	-0.331377	
14	0	0.814900	-0.245524	-0.297938	-0.331377	
15	0	-1.010837	-0.078915	-0.172072	-0.244041	
16	0	0.579951	0.087694	-0.046206	-0.156705	
17	0	-0.450979	-0.245524	-0.297938	-0.331377	
18	0	0.746452	-0.245524	-0.297938	-0.331377	
19	0	0.620782	-0.245524	-0.297938	-0.331377	
20	0	-0.879577	0.420911	0.205526	0.017968	
21	0	-1.371653	0.587520	0.331392	0.105304	
22	0	0.499779	0.754129	0.457258	0.192640	
23	0	0.314506	-0.245524	-0.297938	-0.331377	
24	0	0.862887	-0.245524	-0.297938	-0.331377	
25	0	0.240237	-0.245524	-0.297938	-0.331377	
26	0	1.241962	-0.245524	-0.297938	-0.331377	
27	0	-0.340364	0.920737	0.583124	0.279977	
28	0	0.087606	-0.245524	-0.297938	-0.331377	
29	0	-1.523548	0.254302	0.079660	-0.069368	
30	0	0.270337	-0.245524	-0.297938	-0.331377	
31	1	0.845203	-0.245524	-0.297938	-0.331377	
32	0	0.308359	-0.245524	-0.297938	-0.331377	
33	0	-0.759675	1.087346	0.708990	0.367313	

	Fraud	rand_num	Cardnum_freq0_Date	Cardnum_freq1_Date	Cardnum_freq3_Date	Cardnum
34	0	0.147474	-0.245524	-0.297938	-0.331377	
35	0	-0.498469	-0.245524	-0.297938	-0.331377	
36	0	-0.396444	0.420911	0.205526	0.017968	
37	0	-1.500709	-0.245524	-0.297938	-0.331377	
38	0	1.371066	-0.245524	-0.297938	-0.331377	
39	0	-0.252334	1.253955	0.834856	0.454649	
40	0	1.049972	-0.078915	-0.172072	-0.244041	
41	0	-0.917500	0.587520	0.331392	0.105304	
42	0	-1.471813	-0.245524	-0.297938	-0.331377	
43	0	0.061125	0.754129	0.457258	0.192640	
44	0	-1.326576	-0.245524	-0.297938	-0.331377	
45	0	-1.339589	-0.245524	-0.297938	-0.331377	
46	0	-1.028273	-0.245524	-0.297938	-0.331377	
47	0	-0.174878	1.420564	0.960722	0.541985	
48	0	-0.930082	-0.245524	-0.297938	-0.331377	
49	0	-0.274081	-0.245524	-0.297938	-0.331377	
50	0	-1.347616	-0.245524	-0.297938	-0.331377	
51	0	-0.150967	-0.245524	-0.297938	-0.331377	
52	0	-0.629785	-0.245524	-0.297938	-0.331377	
53	0	-0.790526	-0.245524	-0.297938	-0.331377	
54	0	1.072222	-0.245524	-0.297938	-0.331377	
55	0	-0.403738	-0.245524	-0.297938	-0.331377	
56	0	0.154736	-0.245524	-0.297938	-0.331377	
57	0	0.355142	-0.245524	-0.297938	-0.331377	
58	0	0.851846	-0.078915	-0.172072	-0.244041	
59	0	-0.950397	0.087694	-0.046206	-0.156705	
60	0	-0.939698	0.254302	0.079660	-0.069368	
61	0	-0.792213	0.420911	0.205526	0.017968	
62	0	-0.779299	-0.245524	-0.297938	-0.331377	
63	0	0.703074	-0.245524	-0.297938	-0.331377	
64	0	0.484584	0.587520	0.331392	0.105304	
65	0	0.976030	0.754129	0.457258	0.192640	
66	0	-0.851231	-0.078915	-0.172072	-0.244041	
67	0	-0.748617	-0.245524	-0.297938	-0.331377	
68	0	-0.236289	0.087694	-0.046206	-0.156705	
69	0	1.510265	0.254302	0.079660	-0.069368	

	Fraud	rand_num	Cardnum_freq0_Date	Cardnum_freq1_Date	Cardnum_freq3_Date	Cardnum
70	0	-0.896606	-0.245524	-0.297938	-0.331377	
71	0	1.109031	0.920737	0.583124	0.279977	
72	0	0.694546	1.087346	0.708990	0.367313	
73	0	0.609145	-0.245524	-0.297938	-0.331377	
74	0	1.111719	-0.245524	-0.297938	-0.331377	
75	0	-1.695392	-0.078915	-0.172072	-0.244041	
76	0	0.981434	-0.245524	-0.297938	-0.331377	
77	0	-1.553873	1.253955	0.834856	0.454649	
78	0	0.650607	-0.245524	-0.297938	-0.331377	
79	0	1.244777	-0.245524	-0.297938	-0.331377	
80	0	-0.199622	-0.245524	-0.297938	-0.331377	
81	0	-1.073885	-0.245524	-0.297938	-0.331377	
82	0	-1.429826	-0.245524	-0.297938	-0.331377	
83	0	-1.610303	-0.245524	-0.297938	-0.331377	
84	0	-1.090855	-0.245524	-0.297938	-0.331377	
85	0	-0.184484	-0.245524	-0.297938	-0.331377	
86	0	0.203515	-0.245524	-0.297938	-0.331377	
87	0	-0.039475	-0.245524	-0.297938	-0.244041	
88	0	1.209699	-0.245524	-0.297938	-0.331377	
89	0	-0.855170	-0.245524	-0.297938	-0.331377	
90	0	0.481902	-0.078915	-0.172072	-0.244041	
91	0	0.214364	-0.245524	-0.297938	-0.331377	
92	0	-0.215779	-0.245524	-0.297938	-0.331377	
93	0	-1.311893	-0.245524	-0.297938	-0.331377	
94	0	1.208730	-0.245524	-0.297938	-0.331377	
95	0	0.955098	-0.245524	-0.172072	-0.244041	
96	0	0.827928	-0.245524	-0.297938	-0.331377	
97	0	1.152058	-0.245524	-0.297938	-0.331377	
98	0	-0.154466	-0.078915	-0.172072	-0.244041	
99	0	-1.008235	-0.245524	-0.297938	-0.331377	



Separate data into modeling/Out of Time (OOT) (validation set)

In [ ]:

```
df_oot_mine = df_numerical_zscale.loc[(df_filter['Date'] >= '2010-11-1')]
df_KSFDR_mine = df_numerical_zscale.loc[((df_filter['Date'] < '2010-11-1') & (df_filter['Date'] > '2010-1-14'))]
```

In [ ]:

```
# Mrinal combined the variables, z-scaled the data, and separated the data into modeling and OOT
df_oot = pd.read_csv('df_scaled_oot_data.csv', index_col='Unnamed: 0') # OOT data
df_KSFDR = pd.read_csv('df_scaled_train_test_data.csv', index_col='Unnamed: 0') # Modeling data
```

In [ ]:

```
df_KSFDR.head()
```

Out[ ]:

	Cardnum_totalamount0_Amount2	Cardnum_totalamount1_Amount2	Cardnum_totalamount3_A
3338	-0.118952	0.077781	0
3339	-0.214860	-0.079228	-0
3340	-0.136886	0.316441	1
3341	-0.213805	-0.078589	-0
3342	-0.211389	-0.192924	-0

In [ ]:

```
oot_fraud = df_oot['Fraud']
oot_recnum = df_oot['Recnum']
oot_date = df_oot['Date']
df_oot.drop(labels=['Fraud','Recnum','Date'], axis=1, inplace = True)
df_oot.insert(0, 'Fraud', oot_fraud)
```

In [ ]:

```
df_oot.tail()
```

Out[ ]:

	Fraud	Cardnum_totalamount0_Amount2	Cardnum_totalamount1_Amount2	Cardnum_totalar
96392	0	-0.191423	-0.180840	
96393	0	0.018099	-0.054026	
96394	0	-0.110183	0.068983	
96395	0	0.425591	0.192610	
96396	0	-0.054498	-0.097965	

In [ ]:

```
KSFDR_remove = df_KSFDR[['Fraud','Random_label','Recnum','Date']]
df_KSFDR.drop(labels=['Fraud','Random_label','Recnum','Date'], axis=1, inplace = True)
df_KSFDR.insert(0, 'Fraud', KSFDR_remove['Fraud'])
df_KSFDR.insert(1, 'Random_label', KSFDR_remove['Random_label'])
# df_KSFDR.rename(columns={'Unnamed: 0':'Recnum'}, inplace=True)
# df_KSFDR.Recnum = df_KSFDR.Recnum + 1
```

In [ ]:

```
df_KSFDR.tail()
```

Out[ ]:

	Fraud	Random_label	Cardnum_totalamount0_Amount2	Cardnum_totalamount1_Amount2
83965	0	283223	-0.128773	-0.142921
83966	0	358348	0.013893	-0.056571
83967	0	222953	-0.147649	-0.154345
83968	0	189590	-0.041279	-0.089965
83969	0	285001	-0.207390	-0.190504

Make "goods" and "bads" Dataframes

In [ ]:

```
goods = df_KSFDR.loc[df_KSFDR['Fraud'] == 0]
bads = df_KSFDR.loc[df_KSFDR['Fraud'] == 1]
```

In [ ]:

```
print(len(goods))
print(len(bads))
print(len(goods)+len(bads))
```

```
79764
868
80632
```

Calculate Univariate KS Scores

In [ ]:

```
# Calculate the KS
#https://towardsdatascience.com/how-to-compare-two-distributions-in-practice-8c676904a285
start_ks=pd.datetime.now()

KS_dict={}
p_value_dict={}
stats_cols = df_KSFDR.columns

for column in stats_cols:
    KS_dict[column], p_value_dict[column] = stats.ks_2samp(goods[column],bads[column])

print("DONE!", pd.datetime.now()-start_ks)
```

```
DONE! 0:00:03.204411
```

Calculate Univariate FDR Scores

In [ ]:

```
print(df_KSFDR.shape[0])
print(int(df_KSFDR.shape[0]*0.03))
print(int(np.floor(df_KSFDR.shape[0]*0.03)))
print(int(round(len(df_KSFDR)*0.03)))
```

```
80632
2418
2418
2419
```

In [ ]:

```
start_fdr=pd.datetime.now()

topRows = int(round(len(df_KSFDR)*0.03))
FDR_dict={}
stats_cols = df_KSFDR.columns
numbads = len(bads)
# count=0

for column in stats_cols:

    # Get the X% of the records at both the top and bottom of a sorted column
    df_sorted = df_KSFDR.sort_values(column,ascending=False)
    temp1 = df_sorted.head(topRows)
    temp2 = df_sorted.tail(topRows)

    # Get the values of the fraud labels for the records at X%
    needed1 = temp1.loc[:,'Fraud']
    needed2 = temp2.loc[:,'Fraud']

    # Calculate the FDR
    FDR1 = sum(needed1)/numbads
    FDR2 = sum(needed2)/numbads

    # Take the higher of the two calculated FDRs and save in a dictionary
    FDRate = np.maximum(FDR1,FDR2)
    FDR_dict[column] = FDRate
#     count+=1
#     if count==10:
#         break

print("DONE!", pd.datetime.now()-start_fdr)
```

```
DONE! 0:01:10.414153
```

Rank Order Univariate KS and FDR Scores

In [ ]:

```
# Convert the KS and FDR dictionaries to dataframes and concatenate them
df_KS_scores = pd.DataFrame.from_dict(KS_dict, orient='index', columns=['KS_Scores'])
df_FDR_scores = pd.DataFrame.from_dict(FDR_dict, orient='index', columns=['FDR_Scores'])
df_KSFDR_scores = pd.concat([df_KS_scores,df_FDR_scores], axis=1)
```

In [ ]:

```
# Determine the ranks of each record based on the KS and FDR scores, then find the avg of
 those ranks
df_KSFDR_scores['KS_Rank'] = df_KSFDR_scores['KS_Scores'].rank(method='max', ascending=Fal
se)
df_KSFDR_scores['FDR_Rank'] = df_KSFDR_scores['FDR_Scores'].rank(method='max', ascending=F
alse)
df_KSFDR_scores['Average_Rank'] = df_KSFDR_scores[['KS_Rank', 'FDR_Rank']].mean(axis=1)
```

In [ ]:

```
# Save the dataframe for HW9
df_KSFDR_scores.sort_values(['Average_Rank']).to_csv("hw9_var_ranking_chrissy.csv")
```

In [ ]:

```
df_KSFDR_scores.sort_values(['Average_Rank'])
```

Out[ ]:

	KS_Scores	FDR_Scores	KS_R
Fraud	1.000000	1.000000	
Cardnum_|_zip_totalamount7_Amount2	0.686397	0.640553	
Cardnum_|_zip_totalamount3_Amount2	0.678991	0.642857	
Cardnum_|_merchnum_totalamount7_Amount2	0.683319	0.633641	
Cardnum_|_merchnum_totalamount14_Amount2	0.678014	0.632488	
Cardnum_|_merchnum_totalamount3_Amount2	0.676973	0.631336	
Cardnum_|_zip_totalamount14_Amount2	0.673669	0.627880	
Cardnum_|_state_totalamount3_Amount2	0.673558	0.629032	
Cardnum_|_state_totalamount7_Amount2	0.668370	0.594470	
Cardnum_|_zip_totalamount1_Amount2	0.660731	0.597926	
Cardnum_|_state_totalamount1_Amount2	0.658662	0.601382	
Cardnum_|_merchnum_totalamount1_Amount2	0.659891	0.599078	
Cardnum_|_merchnum_totalamount30_Amount2	0.661936	0.562212	
Cardnum_|_state_totalamount14_Amount2	0.667886	0.521889	
Cardnum_|_zip_totalamount30_Amount2	0.657079	0.548387	
Cardnum_|_merchnum_totalamount0_Amount2	0.612589	0.561060	
Cardnum_|_zip_max14_Amount2	0.657030	0.476959	
Cardnum_|_state_totalamount0_Amount2	0.610209	0.562212	
Cardnum_|_zip_max30_Amount2	0.650586	0.481567	
Cardnum_|_zip_totalamount0_Amount2	0.612411	0.557604	
Cardnum_|_state_max7_Amount2	0.646085	0.489631	
Cardnum_|_merchnum_max14_Amount2	0.655723	0.474654	
Cardnum_|_zip_max3_Amount2	0.648767	0.476959	
Cardnum_|_zip_max7_Amount2	0.656644	0.466590	
Cardnum_|_state_max14_Amount2	0.629058	0.488479	
Cardnum_|_merchnum_max30_Amount2	0.652089	0.473502	
Cardnum_totalamount3_Amount2	0.602209	0.552995	
Cardnum_|_merchnum_max3_Amount2	0.645477	0.475806	
Cardnum_|_merchnum_max7_Amount2	0.651565	0.465438	
Merchnum_totalamount1_Amount2	0.609657	0.490783	
Cardnum_|_state_max3_Amount2	0.647579	0.473502	
Cardnum_totalamount7_Amount2	0.600245	0.518433	
Merchnum_totalamount0_Amount2	0.583273	0.566820	
Cardnum_|_merchnum_max1_Amount2	0.621315	0.457373	

	KS_Scores	FDR_Scores	KS_R
Cardnum_|_zip_max1_Amount2	0.624030	0.456221	2
Cardnum_|_state_totalamount30_Amount2	0.633921	0.444700	2
Cardnum_|_state_max30_Amount2	0.596037	0.478111	4
Cardnum_|_state_max1_Amount2	0.625468	0.442396	2
Merchnum_totalamount3_Amount2	0.616930	0.421659	3
Merchnum_max0_Amount2	0.608987	0.444700	3
Cardnum_totalamount1_Amount2	0.577029	0.544931	6
Cardnum_totalamount0_Amount2	0.570880	0.551843	6
Cardnum_|_state_max0_Amount2	0.602307	0.419355	3
Cardnum_|_zip_max0_Amount2	0.604075	0.415899	3
Merchnum_max1_Amount2	0.595033	0.441244	4
Cardnum_|_merchnum_max0_Amount2	0.601165	0.415899	4
Merchnum_max3_Amount2	0.583431	0.445853	5
Cardnum_max0_Amount2	0.585192	0.425115	5
Merchnum_totalamount7_Amount2	0.589256	0.345622	5
Cardnum_max7_Amount2	0.558813	0.489631	8
Cardnum_max1_Amount2	0.570395	0.430876	7
Cardnum_max3_Amount2	0.561316	0.456221	8
Cardnum_max14_Amount2	0.525391	0.498848	10
Cardnum_|_state_mean7_Amount2	0.586851	0.308756	5
Cardnum_mean1_Amount2	0.571881	0.354839	6
Cardnum_|_state_mean3_Amount2	0.588905	0.305300	5
Cardnum_totalamount14_Amount2	0.547572	0.475806	9
Merchnum_max7_Amount2	0.551014	0.463134	9
Cardnum_mean3_Amount2	0.569964	0.360599	7
Merchnum_mean0_Amount2	0.583053	0.308756	5
Cardnum_|_merchnum_mean0_Amount2	0.573724	0.319124	6
Cardnum_|_state_mean0_Amount2	0.572609	0.323733	6
Cardnum_|_zip_mean30_Amount2	0.600041	0.293779	4
Cardnum_|_merchnum_mean3_Amount2	0.590112	0.297235	4
Cardnum_|_zip_mean3_Amount2	0.589485	0.298387	5
Cardnum_|_zip_mean0_Amount2	0.573385	0.319124	6
Cardnum_|_zip_mean7_Amount2	0.590632	0.294931	4
Cardnum_|_state_mean1_Amount2	0.582529	0.305300	5
Cardnum_|_state_mean14_Amount2	0.572467	0.315668	6
Cardnum_mean0_Amount2	0.570021	0.328341	7

	KS_Scores	FDR_Scores	KS_R
Cardnum_|_zip_mean1_Amount2	0.579440	0.301843	6
Cardnum_|_state_mean30_Amount2	0.566152	0.329493	7
Cardnum_|_merchnum_mean14_Amount2	0.589531	0.292627	4
Cardnum_|_merchnum_mean30_Amount2	0.594145	0.291475	4
Cardnum_|_zip_mean14_Amount2	0.592836	0.291475	4
Cardnum_|_merchnum_mean7_Amount2	0.588476	0.293779	5
Cardnum_|_merchnum_mean1_Amount2	0.576940	0.299539	6
Cardnum_median1_Amount2	0.557158	0.331797	9
Cardnum_mean7_Amount2	0.547133	0.403226	9
Merchnum_mean1_Amount2	0.577429	0.291475	6
Cardnum_mean14_Amount2	0.531209	0.404378	10
Merchnum_max14_Amount2	0.524519	0.407834	10
Cardnum_|_state_median0_Amount2	0.561077	0.302995	8
Cardnum_|_merchnum_median0_Amount2	0.563600	0.299539	7
Cardnum_|_zip_median0_Amount2	0.562624	0.299539	8
Cardnum_mean30_Amount2	0.508856	0.384793	10
Cardnum_median0_Amount2	0.557773	0.308756	8
Cardnum_median3_Amount2	0.544729	0.323733	9
Cardnum_max30_Amount2	0.504592	0.380184	10
Cardnum_totalamount30_Amount2	0.487345	0.404378	11
Cardnum_|_zip_median1_Amount2	0.561281	0.293779	8
Cardnum_|_zip_median3_Amount2	0.566730	0.288018	7
Cardnum_|_merchnum_median1_Amount2	0.564129	0.291475	7
Cardnum_|_state_median1_Amount2	0.560197	0.296083	8
Cardnum_|_merchnum_median3_Amount2	0.565663	0.286866	7
Cardnum_|_merchnum_median30_Amount2	0.570472	0.277650	7
Cardnum_|_state_median30_Amount2	0.550525	0.299539	9
Cardnum_median14_Amount2	0.482923	0.328341	11
Cardnum_|_state_median3_Amount2	0.560409	0.286866	8
Merchnum_mean3_Amount2	0.565893	0.274194	7
Cardnum_median7_Amount2	0.483354	0.308756	11
Cardnum_|_state_median14_Amount2	0.537267	0.296083	10
Cardnum_|_state_median7_Amount2	0.551875	0.290323	9
Cardnum_|_merchnum_median14_Amount2	0.560931	0.278802	8
Cardnum_|_merchnum_median7_Amount2	0.555145	0.279954	9
Cardnum_|_zip_median7_Amount2	0.555049	0.281106	9

	KS_Scores	FDR_Scores	KS_R
Cardnum_|_zip_median30_Amount2	0.560472	0.276498	8
Cardnum_median30_Amount2	0.453773	0.300691	1
Merchnum_median0_Amount2	0.541140	0.288018	1
Cardnum_|_zip_median14_Amount2	0.550917	0.277650	9
Merchnum_mean14_Amount2	0.496421	0.281106	1
Merchnum_mean7_Amount2	0.531377	0.276498	1
Merchnum_median1_Amount2	0.504750	0.278802	1
Merchnum_median3_Amount2	0.490531	0.269585	1
Merchnum_totalamount14_Amount2	0.489803	0.248848	1
Merchnum_max30_Amount2	0.484361	0.250000	1
Merchnum_mean30_Amount2	0.455996	0.260369	1
Merchnum_median7_Amount2	0.449833	0.252304	1
Cardnum_velocity1_Date	0.380536	0.263825	1
Merchnum_median14_Amount2	0.420604	0.233871	1
Merchnum_median30_Amount2	0.414852	0.198157	1
Merchnum_mean_30_actual	0.380517	0.210829	1
Cardnum_velocity3_Date	0.406945	0.197005	1
Cardnum_U*	0.369879	0.202765	1
Merchnum_median_30_actual	0.355859	0.244240	1
Merchnum_totalamount30_Amount2	0.443081	0.130184	1
Cardnum_|_state_velocity1_Date	0.356120	0.183180	1
Cardnum_|_state_velocity3_Date	0.358175	0.141705	1
Merchnum_mean_14_actual	0.345149	0.157834	1
Cardnum_velocity0_Date	0.339447	0.171659	1
Cardnum_|_state_totalamount_1_actual	0.356245	0.122120	1
Cardnum_|_merchnum_velocity1_Date	0.322559	0.191244	1
Merchnum_median_14_actual	0.317564	0.208525	1
Merchnum_mean_7_actual	0.344317	0.120968	1
Cardnum_|_zip_velocity1_Date	0.321233	0.186636	1
Cardnum_|_merchnum_velocity3_Date	0.330937	0.144009	1
Cardnum_|_merchnum_totalamount_1_actual	0.322646	0.154378	1
Merchnum_U*	0.310067	0.198157	1
Cardnum_|_zip_velocity3_Date	0.328043	0.141705	1
Cardnum_|_zip_totalamount_1_actual	0.321359	0.145161	1
Cardnum_|_merchnum_totalamount_3_actual	0.330975	0.116359	1
Merchnum_median_7_actual	0.308119	0.168203	1

	KS_Scores	FDR_Scores	KS_R
Cardnum_velocity7_Date	0.400797	0.086406	12
Cardnum_|_zip_totalamount_3_actual	0.322367	0.102535	15
Cardnum_mean_30_actual	0.357841	0.084101	13
Cardnum_1_dayamount_div_30_dayamount_velchange	0.418193	0.077189	12
Cardnum_|_state_velocity0_Date	0.288048	0.110599	17
Cardnum_median_30_actual	0.281541	0.111751	17
Merchnum_mean_3_actual	0.293718	0.107143	17
Cardnum_|_merchnum_max_1_actual	0.294163	0.101382	16
Cardnum_|_state_totalamount_3_actual	0.349428	0.074885	13
Cardnum_|_zip_max_1_actual	0.296855	0.088710	16
Cardnum_|_state_totalamount_0_actual	0.288224	0.091014	17
Cardnum_|_merchnum_velocity0_Date	0.270659	0.114055	18
Cardnum_|_merchnum_max_3_actual	0.301737	0.084101	16
Cardnum_|_merchnum_totalamount_0_actual	0.270721	0.107143	18
Cardnum_|_zip_velocity0_Date	0.267500	0.110599	19
Cardnum_|_zip_totalamount_0_actual	0.267588	0.104839	19
Merchnum_median_3_actual	0.250209	0.117512	20
Cardnum_|_zip_max_3_actual	0.299690	0.077189	16
Cardnum_|_state_max_1_actual	0.317534	0.070276	15
Cardnum_|_merchnum_max_0_actual	0.230403	0.110599	21
Cardnum_|_zip_max_0_actual	0.230683	0.096774	21
Cardnum_mean_14_actual	0.284988	0.073733	17
Cardnum_|_merchnum_totalamount_7_actual	0.303401	0.064516	16
Merchnum_mean_1_actual	0.255435	0.084101	20
Cardnum_|_merchnum_velocity7_Date	0.345526	0.049539	13
Cardnum_|_state_velocity7_Date	0.361848	0.047235	12
Cardnum_1_dayamount_div_14_dayamount_velchange	0.357973	0.047235	13
Cardnum_|_state_max_0_actual	0.242007	0.081797	20
Cardnum_mean_7_actual	0.273638	0.072581	18
Cardnum_median_14_actual	0.252580	0.078341	20
Cardnum_1_dayvel_div_30_dayvel_velchange	0.281437	0.064516	17
Cardnum_median_7_actual	0.234785	0.079493	20
Cardnum_|_state_max_3_actual	0.309086	0.054147	15
Cardnum_|_merchnum_median_7_actual	0.206237	0.092166	23
Cardnum_|_zip_totalamount_7_actual	0.300989	0.055300	16
Cardnum_|_merchnum_max_7_actual	0.280365	0.063364	17

	KS_Scores	FDR_Scores	KS_R
Cardnum_|_zip_velocity7_Date	0.339417	0.043779	1
Cardnum_|_zip_median_7_actual	0.201160	0.094470	2
Merchnum_median_1_actual	0.215696	0.081797	2
Cardnum_0_dayamount_div_30_dayamount_velchange	0.320645	0.047235	1
Cardnum_max_30_actual	0.337203	0.042627	1
Cardnum_totalamount_0_actual	0.316540	0.046083	1
Cardnum_|_merchnum_median_14_actual	0.190218	0.092166	2
Cardnum_|_zip_max_7_actual	0.276706	0.055300	1
Cardnum_|_merchnum_median_3_actual	0.198526	0.085253	2
Cardnum_|_merchnum_median_1_actual	0.190030	0.089862	2
Cardnum_|_merchnum_0_dayvel_div_7_dayvel_velchange	0.192893	0.085253	2
Cardnum_|_state_totalamount_7_actual	0.295211	0.046083	1
Cardnum_|_zip_median_1_actual	0.188769	0.087558	2
Cardnum_|_state_median_3_actual	0.206427	0.074885	2
Cardnum_|_merchnum_mean_1_actual	0.186460	0.086406	2
Cardnum_|_merchnum_totalamount_14_actual	0.271272	0.048387	1
Cardnum_|_state_0_dayvel_div_7_dayvel_velchange	0.194624	0.079493	2
Cardnum_|_state_1_dayvel_div_30_dayvel_velchange	0.186118	0.087558	2
Cardnum_|_state_mean_30_actual	0.215655	0.069124	2
Cardnum_|_merchnum_mean_3_actual	0.186299	0.085253	2
Cardnum_velocity14_Date	0.333995	0.027650	1
Cardnum_|_state_mean_14_actual	0.210584	0.066820	2
Cardnum_|_zip_0_dayvel_div_7_dayvel_velchange	0.180596	0.088710	2
Cardnum_1_dayvel_div_14_dayvel_velchange	0.268813	0.047235	1
Cardnum_|_state_median_7_actual	0.218141	0.063364	2
Cardnum_|_merchnum_mean_0_actual	0.159201	0.105991	2
Cardnum_|_zip_median_3_actual	0.192265	0.076037	2
Cardnum_|_state_mean_7_actual	0.205537	0.065668	2
Cardnum_totalamount_1_actual	0.303048	0.033410	1
Cardnum_|_merchnum_mean_7_actual	0.183900	0.080645	2
Cardnum_|_state_mean_3_actual	0.203157	0.066820	2
card-state_daysSince	0.261428	0.047235	1
Cardnum_|_state_median_1_actual	0.197437	0.071429	2
Cardnum_|_merchnum_median_0_actual	0.153330	0.101382	2
Cardnum_mean_3_actual	0.219932	0.056452	2
Cardnum_|_state_velocity14_Date	0.332170	0.021889	1

	KS_Scores	FDR_Scores	KS_R
Cardnum_|_zip_mean_1_actual	0.186402	0.076037	2(
Merchnum_mean_0_actual	0.181325	0.078341	2(
Cardnum_|_zip_mean_0_actual	0.159857	0.089862	2{
Cardnum_|_state_mean_1_actual	0.201457	0.064516	2-
Cardnum_|_merchnum_max_14_actual	0.248877	0.047235	2(
Cardnum_|_merchnum_mean_14_actual	0.185179	0.076037	2(
card-zip_daysSince	0.258469	0.044931	1!
Cardnum_|_zip_median_14_actual	0.187278	0.073733	2!
Cardnum_|_state_median_14_actual	0.205996	0.057604	2:
Cardnum_median_3_actual	0.204461	0.055300	2:
Cardnum_|_zip_mean_7_actual	0.180840	0.073733	2(
Cardnum_|_merchnum_median_30_actual	0.167717	0.077189	2{
Cardnum_|_state_median_30_actual	0.196598	0.058756	2-
Cardnum_|_zip_mean_3_actual	0.181516	0.072581	2(
Cardnum_|_zip_totalamount_14_actual	0.255002	0.040323	2(
Cardnum_|_zip_mean_14_actual	0.180764	0.073733	2;
Cardnum_|_state_max_7_actual	0.265351	0.035714	1!
Cardnum_|_zip_max_14_actual	0.236306	0.041475	2(
Cardnum_|_zip_median_0_actual	0.152337	0.080645	2!
Cardnum_max_0_actual	0.279596	0.026498	1;
Cardnum_|_merchnum_mean_30_actual	0.169684	0.073733	2;
Cardnum_1_dayamount_div_7_dayamount_velchange	0.234873	0.039171	2(
Cardnum_|_merchnum_totalamount_30_actual	0.228827	0.042627	2:
Cardnum_|_state_mean_0_actual	0.168161	0.071429	2{
Cardnum_mean_1_actual	0.203875	0.047235	2:
Cardnum_|_merchnum_velocity14_Date	0.327702	0.004608	1-
Cardnum_|_zip_velocity14_Date	0.319360	0.006912	1!
Merchnum_1_dayamount_div_30_dayamount_velchange	0.281379	0.017281	1;
Cardnum_|_merchnum_0_dayamount_div_7_dayamount_velchange	0.192830	0.048387	2-
Merchnum_0_dayamount_div_30_dayamount_velchange	0.222855	0.038018	2;
Merchnum_velocity14_Date	0.265048	0.025346	1!
Cardnum_|_state_totalamount_14_actual	0.225614	0.036866	2:
Cardnum_|_state_1_dayamount_div_30_dayamount_velchange	0.228315	0.035714	2:
Cardnum_|_zip_mean_30_actual	0.163899	0.064516	2{
Cardnum_velocity30_Date	0.265946	0.024194	1!
Cardnum_|_state_median_0_actual	0.159132	0.071429	2!

	KS_Scores	FDR_Scores	KS_R
Cardnum_max_1_actual	0.278441	0.017281	18
Cardnum_|_zip_velocity30_Date	0.282641	0.013825	17
Cardnum_0_dayamount_div_7_dayamount_velchange	0.131258	0.076037	30
Merchnum_velocity1_Date	0.263919	0.024194	19
Merchnum_median_0_actual	0.151704	0.071429	29
Cardnum_|_state_velocity30_Date	0.279428	0.014977	18
Cardnum_median_0_actual	0.163282	0.063364	28
Cardnum_totalamount_3_actual	0.271152	0.016129	18
Cardnum_median_1_actual	0.192278	0.046083	25
Cardnum_|_merchnum_velocity30_Date	0.296707	0.005760	16
Cardnum_max_14_actual	0.244646	0.025346	20
Cardnum_0_dayamount_div_14_dayamount_velchange	0.225718	0.028802	21
Cardnum_mean_0_actual	0.171128	0.051843	27
Cardnum_|_zip_median_30_actual	0.161466	0.056452	28
Cardnum_|_zip_0_dayamount_div_7_dayamount_velchange	0.179230	0.048387	27
Cardnum_|_merchnum_0_dayvel_div_14_dayvel_velchange	0.149117	0.061060	29
Cardnum_0_dayvel_div_7_dayvel_velchange	0.078253	0.076037	32
Cardnum_max_3_actual	0.213301	0.027650	22
Cardnum_|_merchnum_max_30_actual	0.203523	0.031106	23
Merchnum_velocity3_Date	0.278636	0.004608	18
Day_of_week	0.134586	0.063364	30
Cardnum_0_dayvel_div_30_dayvel_velchange	0.191982	0.038018	25
Merchnum_velocity0_Date	0.226099	0.018433	21
Cardnum_|_zip_0_dayvel_div_14_dayvel_velchange	0.127854	0.062212	3
Merchnum_totalamount_3_actual	0.232457	0.014977	2
Merchnum_1_dayamount_div_14_dayamount_velchange	0.259775	0.009217	19
Cardnum_|_zip_totalamount_30_actual	0.212268	0.021889	22
Cardnum_|_state_max_14_actual	0.201562	0.026498	23
Merchnum_velocity7_Date	0.264440	0.004608	19
Merchnum_daysSince	0.126833	0.057604	31
Cardnum_|_state_0_dayamount_div_7_dayamount_velchange	0.173193	0.042627	27
Cardnum_|_state_0_dayvel_div_14_dayvel_velchange	0.126811	0.056452	31
Merchnum_1_dayamount_div_7_dayamount_velchange	0.186156	0.034562	26
Merchnum_velocity30_Date	0.222846	0.014977	22
card-merch_daysSince	0.233280	0.008065	21
Cardnum_1_dayvel_div_7_dayvel_velchange	0.177662	0.039171	27

	KS_Scores	FDR_Scores	KS_R
Merchnum_totalamount_1_actual	0.231745	0.009217	2
Cardnum_totalamount_30_actual	0.199594	0.023041	2
Cardnum_|_zip_1_dayvel_div_30_dayvel_velchange	0.154076	0.043779	2
Cardnum_|_zip_max_30_actual	0.187096	0.027650	2
Merchnum_max_30_actual	0.216785	0.011521	2
Cardnum_|_state_1_dayamount_div_14_dayamount_velchange	0.127162	0.049539	3
Cardnum_|_merchnum_1_dayamount_div_14_dayamount_velchange	0.058815	0.058756	3
Cardnum_|_zip_0_dayvel_div_30_dayvel_velchange	0.079935	0.055300	3
Cardnum_0_dayvel_div_14_dayvel_velchange	0.144191	0.044931	3
Cardnum_|_merchnum_1_dayvel_div_14_dayvel_velchange	0.057627	0.058756	3
Merchnum_max_14_actual	0.184635	0.027650	2
Merchnum_0_dayvel_div_14_dayvel_velchange	0.141456	0.044931	3
Cardnum_|_merchnum_0_dayamount_div_14_dayamount_velchange	0.149142	0.042627	2
Merchnum_totalamount_0_actual	0.208689	0.009217	2
Cardnum_|_state_1_dayvel_div_14_dayvel_velchange	0.096098	0.049539	3
Cardnum_daysSince	0.140105	0.042627	3
Cardnum_|_merchnum_1_dayvel_div_7_dayvel_velchange	0.071504	0.050691	3
Cardnum_|_merchnum_1_dayamount_div_7_dayamount_velchange	0.071441	0.050691	3
Cardnum_totalamount_7_actual	0.189594	0.016129	2
Cardnum_|_zip_1_dayvel_div_7_dayvel_velchange	0.055155	0.051843	3
Cardnum_|_zip_1_dayamount_div_7_dayamount_velchange	0.054578	0.051843	3
Merchnum_1_dayvel_div_7_dayvel_velchange	0.131952	0.040323	3
Merchnum_0_dayamount_div_7_dayamount_velchange	0.171305	0.026498	2
Merchnum_totalamount_7_actual	0.195399	0.010369	2
Merchnum_0_dayvel_div_7_dayvel_velchange	0.172133	0.024194	2
Merchnum_max_7_actual	0.152249	0.028802	2
Cardnum_|_merchnum_0_dayvel_div_30_dayvel_velchange	0.107625	0.042627	3
Cardnum_|_merchnum_1_dayvel_div_30_dayvel_velchange	0.131178	0.035714	3
Cardnum_|_zip_0_dayamount_div_14_dayamount_velchange	0.126562	0.036866	3
Cardnum_|_state_max_30_actual	0.116862	0.038018	3
Cardnum_|_merchnum_0_dayamount_div_30_dayamount_velchange	0.107524	0.033410	3
Cardnum_|_state_totalamount_30_actual	0.147967	0.021889	2
Merchnum_0_dayamount_div_14_dayamount_velchange	0.180792	0.005760	2
Cardnum_|_zip_1_dayamount_div_30_dayamount_velchange	0.146958	0.020737	3
Random_label	0.031640	0.039171	3
Cardnum_|_zip_0_dayamount_div_30_dayamount_velchange	0.078493	0.031106	3

	KS_Scores	FDR_Scores	KS_R
Merchnum_max_1_actual	0.161662	0.012673	28
Cardnum_|_zip_1_dayvel_div_14_dayvel_velchange	0.075024	0.029954	33
Cardnum_max_7_actual	0.160589	0.012673	28
Merchnum_totalamount_14_actual	0.163617	0.009217	28
Cardnum_|_zip_1_dayamount_div_14_dayamount_velchange	0.075402	0.028802	32
Merchnum_max_0_actual	0.129699	0.020737	30
Cardnum_|_state_0_dayamount_div_14_dayamount_velchange	0.102214	0.025346	32
Cardnum_|_state_0_dayamount_div_30_dayamount_velchange	0.094675	0.025346	32
Merchnum_totalamount_30_actual	0.163386	0.004608	28
Merchnum_max_3_actual	0.140639	0.013825	30
Cardnum_|_merchnum_1_dayamount_div_30_dayamount_velchange	0.123121	0.019585	31
Cardnum_|_state_1_dayvel_div_7_dayvel_velchange	0.038649	0.028802	33
Cardnum_|_state_0_dayvel_div_30_dayvel_velchange	0.074700	0.026498	33
Merchnum_1_dayvel_div_30_dayvel_velchange	0.119312	0.016129	31
Cardnum_totalamount_14_actual	0.128733	0.011521	31
Merchnum_0_dayvel_div_30_dayvel_velchange	0.119053	0.013825	31
Merchnum_1_dayvel_div_14_dayvel_velchange	0.121940	0.012673	31
Cardnum_|_state_1_dayamount_div_7_dayamount_velchange	0.047746	0.014977	33

In [ ]:

```
df_KSFDR_scores.sort_values(['Average_Rank']).tail(50)
```

Out[ ]:

	KS_Scores	FDR_Scores	KS_R
Merchnum_max_30_actual	0.216785	0.011521	2
Cardnum_|_state_1_dayamount_div_14_dayamount_velchange	0.127162	0.049539	3
Cardnum_|_merchnum_1_dayamount_div_14_dayamount_velchange	0.058815	0.058756	3
Cardnum_|_zip_0_dayvel_div_30_dayvel_velchange	0.079935	0.055300	3
Cardnum_0_dayvel_div_14_dayvel_velchange	0.144191	0.044931	3
Cardnum_|_merchnum_1_dayvel_div_14_dayvel_velchange	0.057627	0.058756	3
Merchnum_max_14_actual	0.184635	0.027650	2
Merchnum_0_dayvel_div_14_dayvel_velchange	0.141456	0.044931	3
Cardnum_|_merchnum_0_dayamount_div_14_dayamount_velchange	0.149142	0.042627	2
Merchnum_totalamount_0_actual	0.208689	0.009217	2
Cardnum_|_state_1_dayvel_div_14_dayvel_velchange	0.096098	0.049539	3
Cardnum_daysSince	0.140105	0.042627	3
Cardnum_|_merchnum_1_dayvel_div_7_dayvel_velchange	0.071504	0.050691	3
Cardnum_|_merchnum_1_dayamount_div_7_dayamount_velchange	0.071441	0.050691	3
Cardnum_totalamount_7_actual	0.189594	0.016129	2
Cardnum_|_zip_1_dayvel_div_7_dayvel_velchange	0.055155	0.051843	3
Cardnum_|_zip_1_dayamount_div_7_dayamount_velchange	0.054578	0.051843	3
Merchnum_1_dayvel_div_7_dayvel_velchange	0.131952	0.040323	3
Merchnum_0_dayamount_div_7_dayamount_velchange	0.171305	0.026498	2
Merchnum_totalamount_7_actual	0.195399	0.010369	2
Merchnum_0_dayvel_div_7_dayvel_velchange	0.172133	0.024194	2
Merchnum_max_7_actual	0.152249	0.028802	2
Cardnum_|_merchnum_0_dayvel_div_30_dayvel_velchange	0.107625	0.042627	3
Cardnum_|_merchnum_1_dayvel_div_30_dayvel_velchange	0.131178	0.035714	3
Cardnum_|_zip_0_dayamount_div_14_dayamount_velchange	0.126562	0.036866	3
Cardnum_|_state_max_30_actual	0.116862	0.038018	3
Cardnum_|_merchnum_0_dayamount_div_30_dayamount_velchange	0.107524	0.033410	3
Cardnum_|_state_totalamount_30_actual	0.147967	0.021889	2
Merchnum_0_dayamount_div_14_dayamount_velchange	0.180792	0.005760	2
Cardnum_|_zip_1_dayamount_div_30_dayamount_velchange	0.146958	0.020737	3
Random_label	0.031640	0.039171	3
Cardnum_|_zip_0_dayamount_div_30_dayamount_velchange	0.078493	0.031106	3
Merchnum_max_1_actual	0.161662	0.012673	2
Cardnum_|_zip_1_dayvel_div_14_dayvel_velchange	0.075024	0.029954	3

	KS_Scores	FDR_Scores	KS_R
Cardnum_max_7_actual	0.160589	0.012673	2
Merchnum_totalamount_14_actual	0.163617	0.009217	2
Cardnum_|_zip_1_dayamount_div_14_dayamount_velchange	0.075402	0.028802	3
Merchnum_max_0_actual	0.129699	0.020737	3
Cardnum_|_state_0_dayamount_div_14_dayamount_velchange	0.102214	0.025346	3
Cardnum_|_state_0_dayamount_div_30_dayamount_velchange	0.094675	0.025346	3
Merchnum_totalamount_30_actual	0.163386	0.004608	2
Merchnum_max_3_actual	0.140639	0.013825	3
Cardnum_|_merchnum_1_dayamount_div_30_dayamount_velchange	0.123121	0.019585	3
Cardnum_|_state_1_dayvel_div_7_dayvel_velchange	0.038649	0.028802	3
Cardnum_|_state_0_dayvel_div_30_dayvel_velchange	0.074700	0.026498	3
Merchnum_1_dayvel_div_30_dayvel_velchange	0.119312	0.016129	3
Cardnum_totalamount_14_actual	0.128733	0.011521	3
Merchnum_0_dayvel_div_30_dayvel_velchange	0.119053	0.013825	3
Merchnum_1_dayvel_div_14_dayvel_velchange	0.121940	0.012673	3
Cardnum_|_state_1_dayamount_div_7_dayamount_velchange	0.047746	0.014977	3

FILTER Top 80 Candidate Variables

In [ ]:

```
# Import the agreed upon ranking from the group
df_KSFDR_fixed = pd.read_csv('KSFDR_final.csv',index_col='Unnamed: 0')
df_KSFDR_fixed.head()
```

Out[ ]:

	FDR	FDR_rank	KS Score	rank_ks	average_
Fraud	1.000000	1.0	1.000000	1.0	
Cardnum_|_zip_totalamount7_Amount2	0.640553	3.0	0.686397	2.0	
Cardnum_|_zip_totalamount3_Amount2	0.642857	2.0	0.678991	4.0	
Cardnum_|_merchnum_totalamount7_Amount2	0.633641	4.0	0.683319	3.0	
Cardnum_|_merchnum_totalamount14_Amount2	0.632488	5.0	0.678014	5.0	

In [ ]:

```
# Remove all but the top 80 variables
df_KSFDR_scores_top80 = df_KSFDR_fixed.sort_values(['average_rank']).head(81) # Need 81 be
cause "Fraud" is ranked 1st
top80_vars = df_KSFDR_scores_top80.index
for item in top80_vars:
    print(item)
```

Fraud
Cardnum_|_zip_totalamount7_Amount2
Cardnum_|_zip_totalamount3_Amount2
Cardnum_|_merchnum_totalamount7_Amount2
Cardnum_|_merchnum_totalamount14_Amount2
Cardnum_|_merchnum_totalamount3_Amount2
Cardnum_|_state_totalamount3_Amount2
Cardnum_|_zip_totalamount14_Amount2
Cardnum_|_state_totalamount7_Amount2
Cardnum_|_state_totalamount1_Amount2
Cardnum_|_zip_totalamount1_Amount2
Cardnum_|_merchnum_totalamount1_Amount2
Cardnum_|_merchnum_totalamount30_Amount2
Cardnum_|_state_totalamount14_Amount2
Cardnum_|_zip_totalamount30_Amount2
Cardnum_|_zip_max14_Amount2
Cardnum_|_merchnum_totalamount0_Amount2
Cardnum_|_state_totalamount0_Amount2
Cardnum_|_zip_max30_Amount2
Cardnum_|_zip_totalamount0_Amount2
Cardnum_|_state_max7_Amount2
Cardnum_|_merchnum_max14_Amount2
Cardnum_|_zip_max3_Amount2
Cardnum_|_state_max14_Amount2
Cardnum_|_zip_max7_Amount2
Cardnum_|_merchnum_max30_Amount2
Cardnum_totalamount3_Amount2
Cardnum_|_merchnum_max3_Amount2
Cardnum_|_merchnum_max7_Amount2
Cardnum_|_state_max3_Amount2
Merchnum_totalamount1_Amount2
Cardnum_totalamount7_Amount2
Merchnum_totalamount0_Amount2
Cardnum_|_merchnum_max1_Amount2
Cardnum_|_zip_max1_Amount2
Cardnum_|_state_totalamount30_Amount2
Cardnum_|_state_max30_Amount2
Cardnum_|_state_max1_Amount2
Merchnum_max0_Amount2
Merchnum_totalamount3_Amount2
Cardnum_totalamount1_Amount2
Cardnum_totalamount0_Amount2
Cardnum_|_state_max0_Amount2
Cardnum_|_zip_max0_Amount2
Merchnum_max1_Amount2
Cardnum_|_merchnum_max0_Amount2
Merchnum_max3_Amount2
Cardnum_max0_Amount2
Merchnum_totalamount7_Amount2
Cardnum_max7_Amount2
Cardnum_max1_Amount2
Cardnum_max3_Amount2
Cardnum_max14_Amount2
Cardnum_|_state_mean7_Amount2
Cardnum_|_state_mean3_Amount2
Cardnum_mean1_Amount2
Cardnum_totalamount14_Amount2

```
Merchnum_mean0_Amount2
Cardnum_mean3_Amount2
Cardnum_|_zip_mean30_Amount2
Merchnum_max7_Amount2
Cardnum_|_merchnum_mean0_Amount2
Cardnum_|_state_mean0_Amount2
Cardnum_|_merchnum_mean3_Amount2
Cardnum_|_zip_mean0_Amount2
Cardnum_|_zip_mean3_Amount2
Cardnum_|_state_mean1_Amount2
Cardnum_|_zip_mean7_Amount2
Cardnum_mean0_Amount2
Cardnum_|_state_mean14_Amount2
Cardnum_|_zip_mean1_Amount2
Cardnum_|_state_mean30_Amount2
Cardnum_|_merchnum_mean30_Amount2
Cardnum_|_zip_mean14_Amount2
Cardnum_|_merchnum_mean14_Amount2
Cardnum_|_merchnum_mean7_Amount2
Cardnum_|_merchnum_mean1_Amount2
Cardnum_median1_Amount2
Cardnum_mean7_Amount2
Merchnum_mean1_Amount2
Cardnum_mean14_Amount2
```

Wrapper Method

In [ ]:

```
from sklearn import datasets, metrics, preprocessing
from sklearn.feature_selection import RFE, RFECV
from sklearn.linear_model import LogisticRegression, LogisticRegressionCV, Lasso, LassoCV,
LinearRegression, Ridge, RidgeCV
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split, cross_validate, GridSearchCV, KFold,
StratifiedKFold, cross_val_score, cross_val_predict
from sklearn.multiclass import OneVsRestClassifier
from sklearn.preprocessing import StandardScaler, label_binarize,scale
from sklearn.metrics import confusion_matrix, roc_auc_score, mean_squared_error
from sklearn.ensemble import GradientBoostingClassifier
import statsmodels.api as sm
import statsmodels.formula.api as smf
import xgboost as xgb
from numpy import loadtxt
from xgboost import XGBClassifier
from sklearn.utils import resample

# Use for feature importance on a classification problem
from sklearn.datasets import make_classification
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.inspection import permutation_importance
from matplotlib import pyplot
```

In [ ]:

```
df_80 = df_KSFDR[top80_vars]
print(df_80.shape)
df_80.head()
```

```
(80632, 81)
```

Out[ ]:

	Fraud	Cardnum_|_zip_totalamount7_Amount2	Cardnum_|_zip_totalamount3_Amount2	Cardn
3338	0	0.303879	-0.075880	
3339	0	-0.089808	-0.114007	
3340	0	-0.031034	-0.014539	
3341	0	-0.088928	-0.113117	
3342	0	-0.168823	-0.153873	

In [ ]:

```
# Create the independent and dependent variables to train the model (X_train and y_train)
X_train = df_80.iloc[:,1:]
y_train = df_80['Fraud']
```

In [ ]:

```
# Create the independent and dependent variables to validate the model (oot_X, oot_y)
oot_X = df_oot[top80_vars].drop(['Fraud'], axis=1)
oot_y = df_oot['Fraud']
print(oot_X.shape)
print(oot_y.shape)
oot_X.head()
```

```
(12427, 80)
(12427,)
```

Out[ ]:

	Cardnum_|_zip_totalamount7_Amount2	Cardnum_|_zip_totalamount3_Amount2	Cardnum_|_r
83970	-0.128527	-0.113124	
83971	-0.130324	-0.114941	
83972	-0.159895	-0.144844	
83973	-0.166703	-0.151729	
83974	-0.149154	-0.133983	

Determine the Level of Importance for each Candidate Variable with a Score

We decided to use three tree-based algorithms to determine the level of importance of the candidate variables (scores). We used a decision tree, a random forest, and a boosted tree. For each tree-based algorithm, we extracted the ".feature*importance*" method from the resultant model as the scores and then ranked those scores from 1 to n. We then averaged the three ranks and used the average to give a final rank ordering of the candidate variables from 1 to n.
References:
https://machinelearningmastery.com/calculate-feature-importance-with-python/ (https://machinelearningmastery.com/calculate-feature-importance-with-python/)
https://www.scikit-yb.org/en/latest/api/model_selection/importances.html (https://www.scikit-yb.org/en/latest/api/model_selection/importances.html)

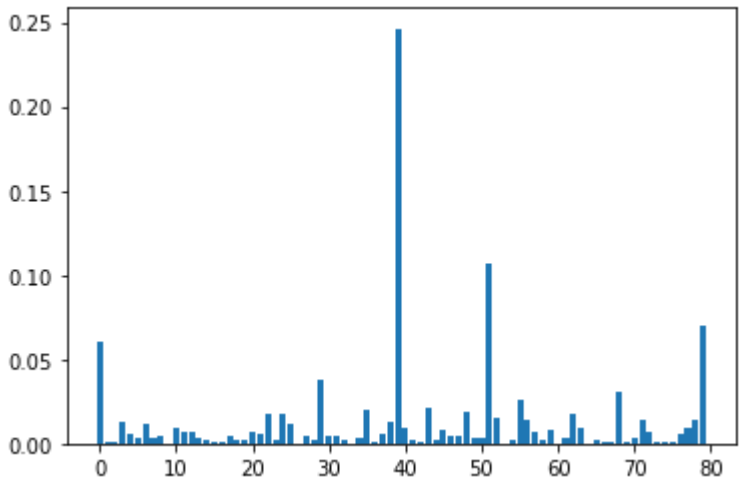
Decision Tree Feature Importance

In [ ]:

```
# define the model
dtc = DecisionTreeClassifier()

# fit the model
dtc.fit(X_train, y_train)

# # perform permutation importance
# results_dtc = permutation_importance(dtc, X_train, y_train, scoring='accuracy', n_repeat
s=5)

# # get importance
# importance_dtc = results_dtc.importances_mean

# get importance
importance_dtc = dtc.feature_importances_

# summarize feature importance
for i,v in enumerate(importance_dtc):
    print('Feature: %0d, Score: %.5f' % (i,v))

# plot feature importance
pyplot.bar([x for x in range(len(importance_dtc))], importance_dtc)
pyplot.show()
```

Feature: 0, Score: 0.06071
Feature: 1, Score: 0.00151
Feature: 2, Score: 0.00078
Feature: 3, Score: 0.01230
Feature: 4, Score: 0.00593
Feature: 5, Score: 0.00277
Feature: 6, Score: 0.01201
Feature: 7, Score: 0.00274
Feature: 8, Score: 0.00490
Feature: 9, Score: 0.00000
Feature: 10, Score: 0.00880
Feature: 11, Score: 0.00677
Feature: 12, Score: 0.00723
Feature: 13, Score: 0.00341
Feature: 14, Score: 0.00194
Feature: 15, Score: 0.00098
Feature: 16, Score: 0.00112
Feature: 17, Score: 0.00418
Feature: 18, Score: 0.00229
Feature: 19, Score: 0.00260
Feature: 20, Score: 0.00648
Feature: 21, Score: 0.00568
Feature: 22, Score: 0.01819
Feature: 23, Score: 0.00201
Feature: 24, Score: 0.01805
Feature: 25, Score: 0.01118
Feature: 26, Score: 0.00011
Feature: 27, Score: 0.00470
Feature: 28, Score: 0.00165
Feature: 29, Score: 0.03728
Feature: 30, Score: 0.00487
Feature: 31, Score: 0.00498
Feature: 32, Score: 0.00201
Feature: 33, Score: 0.00000
Feature: 34, Score: 0.00354
Feature: 35, Score: 0.02048
Feature: 36, Score: 0.00112
Feature: 37, Score: 0.00598
Feature: 38, Score: 0.01281
Feature: 39, Score: 0.24626
Feature: 40, Score: 0.00932
Feature: 41, Score: 0.00166
Feature: 42, Score: 0.00057
Feature: 43, Score: 0.02131
Feature: 44, Score: 0.00205
Feature: 45, Score: 0.00865
Feature: 46, Score: 0.00413
Feature: 47, Score: 0.00425
Feature: 48, Score: 0.01868
Feature: 49, Score: 0.00344
Feature: 50, Score: 0.00377
Feature: 51, Score: 0.10690
Feature: 52, Score: 0.01529
Feature: 53, Score: 0.00000
Feature: 54, Score: 0.00156
Feature: 55, Score: 0.02611
Feature: 56, Score: 0.01356

```
Feature: 57, Score: 0.00725
Feature: 58, Score: 0.00215
Feature: 59, Score: 0.00843
Feature: 60, Score: 0.00000
Feature: 61, Score: 0.00371
Feature: 62, Score: 0.01712
Feature: 63, Score: 0.00892
Feature: 64, Score: 0.00000
Feature: 65, Score: 0.00186
Feature: 66, Score: 0.00116
Feature: 67, Score: 0.00105
Feature: 68, Score: 0.03086
Feature: 69, Score: 0.00058
Feature: 70, Score: 0.00363
Feature: 71, Score: 0.01356
Feature: 72, Score: 0.00643
Feature: 73, Score: 0.00087
Feature: 74, Score: 0.00058
Feature: 75, Score: 0.00078
Feature: 76, Score: 0.00615
Feature: 77, Score: 0.00890
Feature: 78, Score: 0.01450
Feature: 79, Score: 0.06987
```



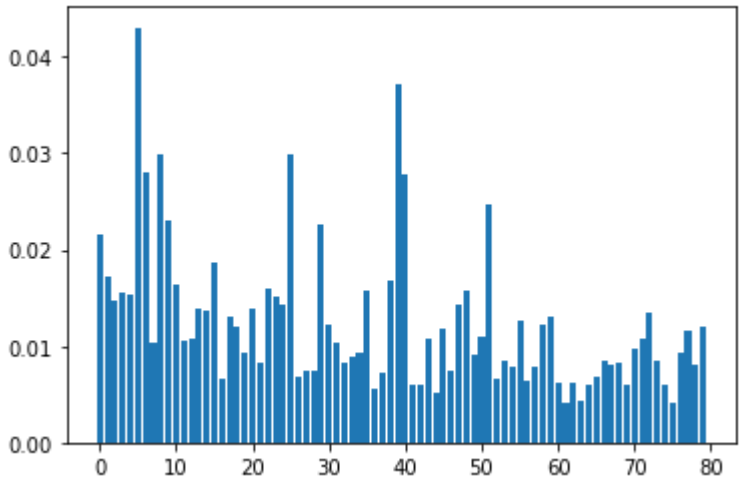
Random Forest Feature Importance

In [ ]:

```
# define the model
rfc = RandomForestClassifier()

# fit the model
rfc.fit(X_train, y_train)

# # perform permutation importance
# results_rfc = permutation_importance(rfc, X_train, y_train, scoring='accuracy', n_repeat
s=5)

# # get importance
# importance_rfc = results_rfc.importances_mean

# get importance
importance_rfc = rfc.feature_importances_

# summarize feature importance
for i,v in enumerate(importance_rfc):
    print('Feature: %0d, Score: %.5f' % (i,v))

# plot feature importance
pyplot.bar([x for x in range(len(importance_rfc))], importance_rfc)
pyplot.show()
```

Feature: 0, Score: 0.02162
Feature: 1, Score: 0.01727
Feature: 2, Score: 0.01482
Feature: 3, Score: 0.01554
Feature: 4, Score: 0.01536
Feature: 5, Score: 0.04305
Feature: 6, Score: 0.02800
Feature: 7, Score: 0.01026
Feature: 8, Score: 0.02997
Feature: 9, Score: 0.02307
Feature: 10, Score: 0.01648
Feature: 11, Score: 0.01066
Feature: 12, Score: 0.01079
Feature: 13, Score: 0.01393
Feature: 14, Score: 0.01360
Feature: 15, Score: 0.01862
Feature: 16, Score: 0.00666
Feature: 17, Score: 0.01305
Feature: 18, Score: 0.01204
Feature: 19, Score: 0.00924
Feature: 20, Score: 0.01386
Feature: 21, Score: 0.00819
Feature: 22, Score: 0.01592
Feature: 23, Score: 0.01520
Feature: 24, Score: 0.01439
Feature: 25, Score: 0.02986
Feature: 26, Score: 0.00672
Feature: 27, Score: 0.00736
Feature: 28, Score: 0.00750
Feature: 29, Score: 0.02262
Feature: 30, Score: 0.01226
Feature: 31, Score: 0.01030
Feature: 32, Score: 0.00825
Feature: 33, Score: 0.00893
Feature: 34, Score: 0.00941
Feature: 35, Score: 0.01581
Feature: 36, Score: 0.00566
Feature: 37, Score: 0.00713
Feature: 38, Score: 0.01673
Feature: 39, Score: 0.03719
Feature: 40, Score: 0.02782
Feature: 41, Score: 0.00592
Feature: 42, Score: 0.00602
Feature: 43, Score: 0.01069
Feature: 44, Score: 0.00508
Feature: 45, Score: 0.01182
Feature: 46, Score: 0.00739
Feature: 47, Score: 0.01425
Feature: 48, Score: 0.01570
Feature: 49, Score: 0.00917
Feature: 50, Score: 0.01091
Feature: 51, Score: 0.02462
Feature: 52, Score: 0.00670
Feature: 53, Score: 0.00841
Feature: 54, Score: 0.00777
Feature: 55, Score: 0.01273
Feature: 56, Score: 0.00649

```
Feature: 57, Score: 0.00783
Feature: 58, Score: 0.01221
Feature: 59, Score: 0.01310
Feature: 60, Score: 0.00628
Feature: 61, Score: 0.00409
Feature: 62, Score: 0.00629
Feature: 63, Score: 0.00439
Feature: 64, Score: 0.00596
Feature: 65, Score: 0.00678
Feature: 66, Score: 0.00838
Feature: 67, Score: 0.00802
Feature: 68, Score: 0.00818
Feature: 69, Score: 0.00592
Feature: 70, Score: 0.00964
Feature: 71, Score: 0.01079
Feature: 72, Score: 0.01354
Feature: 73, Score: 0.00847
Feature: 74, Score: 0.00601
Feature: 75, Score: 0.00419
Feature: 76, Score: 0.00933
Feature: 77, Score: 0.01157
Feature: 78, Score: 0.00814
Feature: 79, Score: 0.01200
```



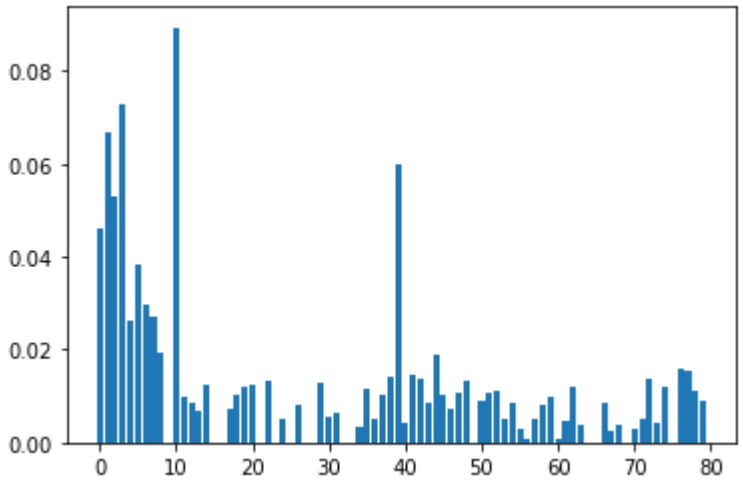
Boosted Tree Feature Importance

In [ ]:

```
# define the model
xgbClass =  XGBClassifier()

# fit the model
xgbClass.fit(X_train, y_train)

# # perform permutation importance
# results_xgb = permutation_importance(xgbClass, X_train, y_train, scoring='accuracy', n_r
epeats=5)

# # get importance
# importance_xgb = results_xgb.importances_mean

# get importance
importance_xgb = xgbClass.feature_importances_

# summarize feature importance
for i,v in enumerate(importance_xgb):
    print('Feature: %0d, Score: %.5f' % (i,v))

# plot feature importance
pyplot.bar([x for x in range(len(importance_xgb))], importance_xgb)
pyplot.show()
```

Feature: 0, Score: 0.04597
Feature: 1, Score: 0.06666
Feature: 2, Score: 0.05285
Feature: 3, Score: 0.07304
Feature: 4, Score: 0.02631
Feature: 5, Score: 0.03831
Feature: 6, Score: 0.02981
Feature: 7, Score: 0.02690
Feature: 8, Score: 0.01949
Feature: 9, Score: 0.00000
Feature: 10, Score: 0.08939
Feature: 11, Score: 0.00988
Feature: 12, Score: 0.00851
Feature: 13, Score: 0.00669
Feature: 14, Score: 0.01262
Feature: 15, Score: 0.00000
Feature: 16, Score: 0.00000
Feature: 17, Score: 0.00734
Feature: 18, Score: 0.01016
Feature: 19, Score: 0.01207
Feature: 20, Score: 0.01232
Feature: 21, Score: 0.00000
Feature: 22, Score: 0.01329
Feature: 23, Score: 0.00000
Feature: 24, Score: 0.00516
Feature: 25, Score: 0.00000
Feature: 26, Score: 0.00796
Feature: 27, Score: 0.00000
Feature: 28, Score: 0.00000
Feature: 29, Score: 0.01271
Feature: 30, Score: 0.00555
Feature: 31, Score: 0.00655
Feature: 32, Score: 0.00000
Feature: 33, Score: 0.00000
Feature: 34, Score: 0.00332
Feature: 35, Score: 0.01164
Feature: 36, Score: 0.00498
Feature: 37, Score: 0.01036
Feature: 38, Score: 0.01416
Feature: 39, Score: 0.05978
Feature: 40, Score: 0.00409
Feature: 41, Score: 0.01453
Feature: 42, Score: 0.01362
Feature: 43, Score: 0.00849
Feature: 44, Score: 0.01869
Feature: 45, Score: 0.01013
Feature: 46, Score: 0.00724
Feature: 47, Score: 0.01053
Feature: 48, Score: 0.01319
Feature: 49, Score: 0.00000
Feature: 50, Score: 0.00883
Feature: 51, Score: 0.01054
Feature: 52, Score: 0.01120
Feature: 53, Score: 0.00488
Feature: 54, Score: 0.00834
Feature: 55, Score: 0.00300
Feature: 56, Score: 0.00098

```
Feature: 57, Score: 0.00510
Feature: 58, Score: 0.00824
Feature: 59, Score: 0.00983
Feature: 60, Score: 0.00066
Feature: 61, Score: 0.00458
Feature: 62, Score: 0.01215
Feature: 63, Score: 0.00359
Feature: 64, Score: 0.00000
Feature: 65, Score: 0.00000
Feature: 66, Score: 0.00855
Feature: 67, Score: 0.00270
Feature: 68, Score: 0.00399
Feature: 69, Score: 0.00000
Feature: 70, Score: 0.00281
Feature: 71, Score: 0.00509
Feature: 72, Score: 0.01363
Feature: 73, Score: 0.00416
Feature: 74, Score: 0.01180
Feature: 75, Score: 0.00000
Feature: 76, Score: 0.01596
Feature: 77, Score: 0.01534
Feature: 78, Score: 0.01096
Feature: 79, Score: 0.00878
```



Rank Ordered Feature Importance (All 3 Tree Based Algorithms)

In [ ]:

```
tree_scores = {'Feat_Num':list(range(1,81)),
               'Decision_Tree':importance_dtc,
               'Random_Forest':importance_rfc,
               'Boosted_Tree':importance_xgb}
df_tree_scores = pd.DataFrame(tree_scores)
```

In [ ]:

```
# Determine the ranks of each record based on the tree scores, then find the avg of those
 ranks
df_tree_scores['DT_Rank'] = df_tree_scores['Decision_Tree'].rank(method='max', ascending=F
alse)
df_tree_scores['RFC_Rank'] = df_tree_scores['Random_Forest'].rank(method='max', ascending=
False)
df_tree_scores['BT_Rank'] = df_tree_scores['Boosted_Tree'].rank(method='max', ascending=Fa
lse)
df_tree_scores['AVG_Rank'] = df_tree_scores[['DT_Rank', 'RFC_Rank','BT_Rank']].mean(axis=1
)
df_tree_scores['Feat_Name'] = X_train.columns
```

In [ ]:

```
df_tree_scores_ordered = df_tree_scores.sort_values(['AVG_Rank'])['Feat_Name']
```

In [ ]:

```
df_tree_scores.sort_values(['AVG_Rank'])
```

Out[ ]:

	Feat_Num	Decision_Tree	Random_Forest	Boosted_Tree	DT_Rank	RFC_Rank	BT_Rank	A
39	40	0.246261	0.037190	0.059777	1.0	2.0	4.0	
0	1	0.060712	0.021623	0.045968	4.0	10.0	6.0	
6	7	0.012013	0.027998	0.029807	20.0	5.0	8.0	1
29	30	0.037276	0.022622	0.012706	5.0	9.0	21.0	1
3	4	0.012296	0.015536	0.073042	19.0	18.0	2.0	1
51	52	0.106899	0.024624	0.010543	2.0	7.0	30.0	1
10	11	0.008796	0.016482	0.089390	25.0	14.0	1.0	1
22	23	0.018189	0.015924	0.013289	11.0	15.0	19.0	1
48	49	0.018684	0.015705	0.013187	10.0	17.0	20.0	1
38	39	0.012814	0.016735	0.014157	18.0	13.0	16.0	1
8	9	0.004902	0.029969	0.019489	38.0	3.0	11.0	1
35	36	0.020482	0.015812	0.011643	9.0	16.0	27.0	1
5	6	0.002774	0.043051	0.038314	50.0	1.0	7.0	1
4	5	0.005932	0.015362	0.026312	35.0	19.0	10.0	2
77	78	0.008902	0.011567	0.015343	24.0	36.0	14.0	2
79	80	0.069867	0.012001	0.008776	3.0	34.0	38.0	2
72	73	0.006433	0.013535	0.013626	32.0	27.0	17.0	2
1	2	0.001515	0.017273	0.066665	63.0	12.0	3.0	2
20	21	0.006480	0.013857	0.012316	31.0	25.0	23.0	2
24	25	0.018055	0.014393	0.005163	12.0	22.0	50.0	2
40	41	0.009319	0.027822	0.004094	22.0	6.0	57.0	2
43	44	0.021310	0.010692	0.008490	8.0	40.0	41.0	2
59	60	0.008431	0.013097	0.009834	27.0	28.0	36.0	3
76	77	0.006149	0.009333	0.015958	33.0	46.0	13.0	3
45	46	0.008648	0.011816	0.010128	26.0	35.0	34.0	3
47	48	0.004247	0.014254	0.010533	41.0	23.0	31.0	3
2	3	0.000776	0.014819	0.052855	71.0	21.0	5.0	3
55	56	0.026115	0.012730	0.003002	7.0	30.0	61.0	3
78	79	0.014498	0.008140	0.010961	15.0	56.0	29.0	3
7	8	0.002742	0.010259	0.026896	51.0	43.0	9.0	3
25	26	0.011185	0.029861	0.000000	21.0	4.0	80.0	3
11	12	0.006772	0.010660	0.009881	30.0	41.0	35.0	3
62	63	0.017124	0.006293	0.012152	13.0	69.0	24.0	3
14	15	0.001941	0.013603	0.012616	58.0	26.0	22.0	3

	Feat_Num	Decision_Tree	Random_Forest	Boosted_Tree	DT_Rank	RFC_Rank	BT_Rank	A
71	72	0.013562	0.010794	0.005091	17.0	38.0	52.0	3
52	53	0.015285	0.006704	0.011204	14.0	66.0	28.0	3
12	13	0.007232	0.010790	0.008512	29.0	39.0	40.0	3
17	18	0.004183	0.013046	0.007341	42.0	29.0	45.0	3
50	51	0.003771	0.010908	0.008832	44.0	37.0	37.0	3
68	69	0.030856	0.008184	0.003991	6.0	55.0	58.0	3
18	19	0.002290	0.012043	0.010159	53.0	33.0	33.0	3
30	31	0.004869	0.012265	0.005549	39.0	31.0	49.0	3
13	14	0.003415	0.013934	0.006689	49.0	24.0	47.0	4
19	20	0.002595	0.009240	0.012071	52.0	47.0	25.0	4
31	32	0.004978	0.010301	0.006550	37.0	42.0	48.0	4
58	59	0.002148	0.012205	0.008243	54.0	32.0	43.0	4
37	38	0.005982	0.007135	0.010363	34.0	63.0	32.0	4
57	58	0.007247	0.007833	0.005098	28.0	58.0	51.0	4
44	45	0.002046	0.005078	0.018694	55.0	77.0	12.0	4
56	57	0.013565	0.006488	0.000983	16.0	68.0	64.0	4
41	42	0.001664	0.005925	0.014533	60.0	74.0	15.0	4
46	47	0.004131	0.007388	0.007239	43.0	61.0	46.0	5
70	71	0.003629	0.009642	0.002810	46.0	44.0	62.0	5
34	35	0.003543	0.009414	0.003322	47.0	45.0	60.0	5
66	67	0.001165	0.008381	0.008554	64.0	52.0	39.0	5
23	24	0.002006	0.015204	0.000000	57.0	20.0	80.0	5
15	16	0.000977	0.018620	0.000000	68.0	11.0	80.0	5
63	64	0.008925	0.004394	0.003585	23.0	78.0	59.0	5
54	55	0.001558	0.007772	0.008339	62.0	59.0	42.0	5
42	43	0.000574	0.006023	0.013616	74.0	71.0	18.0	5
9	10	0.000000	0.023071	0.000000	80.0	8.0	80.0	5
21	22	0.005678	0.008192	0.000000	36.0	54.0	80.0	5
74	75	0.000582	0.006010	0.011800	73.0	72.0	26.0	5
73	74	0.000873	0.008473	0.004163	69.0	50.0	56.0	5
49	50	0.003435	0.009167	0.000000	48.0	48.0	80.0	5
61	62	0.003708	0.004095	0.004577	45.0	80.0	55.0	6
27	28	0.004704	0.007359	0.000000	40.0	62.0	80.0	6
26	27	0.000113	0.006718	0.007958	75.0	65.0	44.0	6
53	54	0.000000	0.008408	0.004880	80.0	51.0	54.0	6
67	68	0.001048	0.008020	0.002702	67.0	57.0	63.0	6

	Feat_Num	Decision_Tree	Random_Forest	Boosted_Tree	DT_Rank	RFC_Rank	BT_Rank	A
32	33	0.002014	0.008253	0.000000	56.0	53.0	80.0	6
36	37	0.001120	0.005665	0.004983	65.0	76.0	53.0	6
28	29	0.001650	0.007501	0.000000	61.0	60.0	80.0	6
65	66	0.001863	0.006785	0.000000	59.0	64.0	80.0	6
33	34	0.000000	0.008931	0.000000	80.0	49.0	80.0	6
16	17	0.001118	0.006661	0.000000	66.0	67.0	80.0	7
60	61	0.000000	0.006282	0.000655	80.0	70.0	65.0	7
69	70	0.000582	0.005920	0.000000	73.0	75.0	80.0	7
75	76	0.000776	0.004186	0.000000	71.0	79.0	80.0	7
64	65	0.000000	0.005958	0.000000	80.0	73.0	80.0	7

In [ ]:

```
df_tree_scores.sort_values(['AVG_Rank']).to_csv('df_tree_scores_results.csv')
```

Model Algorithms

Boosted Trees

Train a boosted tree with the selected variables

In [ ]:

```
top25vars = df_tree_scores.sort_values(['AVG_Rank']).Feat_Name[:25]
top20vars = df_tree_scores.sort_values(['AVG_Rank']).Feat_Name[:20]
top30vars = df_tree_scores.sort_values(['AVG_Rank']).Feat_Name[:30]
df_top30vars = df_KSFDR[top30vars]
df_top21vars = df_KSFDR[top30vars]
df_top25vars = df_KSFDR[top25vars]
df_top20vars = df_KSFDR[top20vars]
```

In [ ]:

```
# Create Train and Test sets
X_train, X_test, y_train, y_test = train_test_split(df_top30vars, df_KSFDR['Fraud'], test_
size=0.3, train_size=0.7)
```

In [ ]:

```
# Get the OOT data
X_oot = df_oot[consolidated_dict['top30'].Feat_Name]
y_oot = df_oot['Fraud']

# X_oot = df_oot[top20vars]
# y_oot = df_oot['Fraud']
```

In [ ]:

```
sum_pos = sum(y_train== 1.0)
print(sum_pos)
sum_neg = sum(y_train== 0.0)
print(sum_neg)
ratio = sum_neg / sum_pos
print(ratio)
```

```
587
55855
95.15332197614991
```

Initial Results with GridSearchCV

Function Definition Area

In [ ]:

```
from sklearn.metrics import make_scorer
def custom_FDR(y_true, y_scores):
    res_df = pd.DataFrame({'score':y_scores,'label': y_true}).sort_values(by='score',ascen
ding=False)
    top3_res1 = res_df.head(round(y_true.shape[0]*0.03))
    top3_res2 = res_df.tail(round(y_true.shape[0]*0.03))

#     return np.maximum((top3_res1['label'].sum()/sum(y_true)),(top3_res1['label'].sum()/s
um(y_true)))
    return (top3_res1['label'].sum()/sum(y_true))
my_fdr_metric = make_scorer(custom_FDR, greater_is_better=True,needs_proba = True)
```

In [ ]:

```
def make_data(model, X_data, y_data):
    fraud_proba = model.predict_proba(X_data)[:, 1]
    fraud_pred = model.predict(X_data)
    curr_data = X_data.copy()
    curr_data.insert(0, 'Fraud', y_data)
    curr_data.insert(1, 'fraud_proba', fraud_proba)
    curr_data.insert(2, 'frad_pred', fraud_pred)
    curr_data = curr_data.sort_values(['fraud_proba'], ascending=False)
    return curr_data
```

In [ ]:

```
def fdr(data, topRows):
    topRows_fs = int(round(len(data)*topRows))
    data_topRows = data.head(topRows_fs)
    frauds_current = data_topRows.loc[:,'Fraud']
    bads_all = data.loc[data['Fraud'] == 1]
    FDR = sum(frauds_current) / len(bads_all)
    return FDR
```

GridSearchCV caused overfitting.

In [ ]:

```
# xgbClass =  xgb.XGBClassifier(objective='binary:logistic', learning_rate = 0.01, n_estim
ators=800, max_depth=6, scale_pos_weight=ratio)
xgbClass =  XGBClassifier(objective='binary:logistic')
```

In [ ]:

```
# Define a set of parameters to test using GridSearchCV
# NOTE: a variety of different parameters were tested (some not shown)
parameters_xgb = {
    'max_depth': range (3, 7, 1),
    'n_estimators': range(600, 900, 200),
    'eta': [0.001, 0.0001, 0.00001],
    'scale_pos_weight':[ratio],
    'scoring':[my_fdr_metric]}

# parameters_xgb = {
#     'max_depth': range (3, 7, 1),
#     'n_estimators': range(400, 1500, 200),
#     'eta': [0.1, 0.01, 0.001, 0.002, 0.2, 0.3],
#     'scale_pos_weight':[ratio, 1],
#     'scoring':['accuracy','roc_auc']
# }
```

In [ ]:

```
# Instantiate the model with GridSearchCV
xgb_gsCV = GridSearchCV(
    estimator=xgbClass,
    param_grid=parameters_xgb,
    n_jobs = -1,
    cv = 3,
    verbose=True
)
```

In [ ]:

```
# Fit GridSearchCV
start_xgbfit=pd.datetime.now()
xgb_gsCV.fit(X_train,y_train)
print("DONE!", pd.datetime.now()-start_xgbfit)
```

```
Fitting 3 folds for each of 24 candidates, totalling 72 fits


[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed: 16.9min
[Parallel(n_jobs=-1)]: Done  72 out of  72 | elapsed: 35.1min finished

DONE! 0:37:06.987029
```

In [ ]:

```
# View the best estimator from GridSearchCV
xgb_gsCV.best_estimator_
```

Out[ ]:

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, eta=0.001, gamma=0,
              learning_rate=0.1, max_delta_step=0, max_depth=5,
              min_child_weight=1, missing=None, n_estimators=600, n_jobs=1,
              nthread=None, objective='binary:logistic', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=92.75747508305648,
              scoring=make_scorer(custom_FDR, needs_proba=True), seed=None,
              silent=None, subsample=1, verbosity=1)
```

In [ ]:

```
# View the best parameters from GridSearchCV
xgb_gsCV.best_params_
```

Out[ ]:

```
{'eta': 0.001,
 'max_depth': 5,
 'n_estimators': 600,
 'scale_pos_weight': 92.75747508305648,
 'scoring': make_scorer(custom_FDR, needs_proba=True)}
```

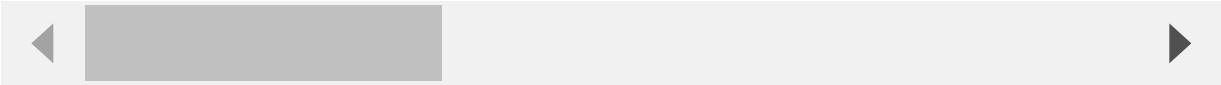
In [ ]:

```
# View the results of GridSearchCV
df_xgb_gsCV = pd.DataFrame.from_dict(xgb_gsCV.cv_results_)
# df_xgb_gsCV.to_csv("df_xgb_gsCV.csv")
df_xgb_gsCV.sort_values('rank_test_score')
```

Out[ ]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_eta	param_max_depth
21	282.051802	1.679179	1.189662	0.121395	1e-05	5
4	209.488939	0.589999	1.006986	0.083744	0.001	5
5	280.793649	1.896140	1.266117	0.069475	0.001	5
20	223.349348	0.119128	0.936688	0.008936	1e-05	5
13	285.930664	2.925059	1.224047	0.017197	0.0001	5
12	217.616284	2.475429	1.090873	0.115589	0.0001	5
11	237.803846	3.959255	0.977739	0.032849	0.0001	4
3	235.609316	1.642838	0.978220	0.081575	0.001	4
19	251.816939	3.371222	0.986578	0.032717	1e-05	4
2	181.050254	3.284566	0.728718	0.019138	0.001	4
18	192.449793	0.841711	0.709227	0.032050	1e-05	4
10	177.102899	1.060720	0.753004	0.056833	0.0001	4
15	308.133928	3.694003	1.393442	0.052395	0.0001	6
23	233.726858	22.464724	0.826473	0.154591	1e-05	6
7	305.386951	5.331737	1.373480	0.032354	0.001	6
14	244.812217	2.482369	1.146194	0.034451	0.0001	6

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_eta	param_max_depth
6	243.603689	2.822373	1.087757	0.012332	0.001	6
22	237.476179	3.802629	0.915300	0.064436	1e-05	6
9	186.822788	0.401500	0.690448	0.033818	0.0001	3
17	195.598373	4.200792	0.738298	0.050170	1e-05	3
1	189.428870	2.296054	0.692990	0.030821	0.001	3
16	142.383669	1.174513	0.536757	0.029157	1e-05	3
8	141.918293	2.504744	0.569905	0.050617	0.0001	3
0	142.767306	1.474759	0.556844	0.020118	0.001	3



In [ ]:

```
# Calculate the accuracy scores of the model
train_score = xgb_gsCV.score(X_train, y_train)
print("Training Score:", train_score)
test_score = xgb_gsCV.score(X_test, y_test)
print("Test Score:", test_score)
oot_score = xgb_gsCV.score(X_oot, y_oot)
print("OOT Score:", oot_score)
print("")

# Calculate the ".predict_proba" and make dataframes for all datasets
train_data = make_data(xgb_gsCV, X_train, y_train)
test_data = make_data(xgb_gsCV, X_test, y_test)
oot_data = make_data(xgb_gsCV, X_oot, y_oot)

# Calculate the FDRs
fdr_rate = 0.03
train_FDR = fdr(train_data, fdr_rate)
print("Training FDR:", train_FDR)
test_FDR = fdr(test_data, fdr_rate)
print("Test FDR:", test_FDR)
oot_FDR = fdr(oot_data, fdr_rate)
print("OOT FDR:", oot_FDR)
```

```
Training Score: 1.0
Test Score: 0.9970649028524183
OOT Score: 0.9884927979399695

Training FDR: 1.0
Test FDR: 0.9511278195488722
OOT FDR: 0.48044692737430167
```

Initial Results via Manual Processing

As a part of the project, we were required to run a model 10 times with each run using a different train_test_split to see the variance in the results.

In [ ]:

```
top25vars = consolidated_dict['top30'].Feat_Name.iloc[:-5]
top20vars = consolidated_dict['top30'].Feat_Name.iloc[:-10]

df_top30vars = df_KSFDR[consolidated_dict['top30'].Feat_Name]
df_top21vars = df_KSFDR[consolidated_dict['top21'].Feat_Name]
df_top25vars = df_KSFDR[top25vars]
df_top20vars = df_KSFDR[top20vars]

goods = df_KSFDR.loc[df_KSFDR['Fraud'] == 0]
bads = df_KSFDR.loc[df_KSFDR['Fraud'] == 1]
```

In [ ]:

```
consolidated_dict['top30'].Feat_Name
```

Out[ ]:

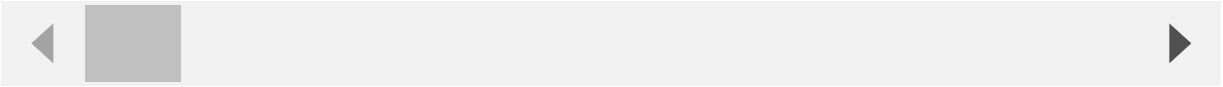
```
0                 Cardnum_totalamount1_Amount2
2                Merchnum_totalamount1_Amount2
3                        Cardnum_max14_Amount2
4                         Cardnum_max7_Amount2
10         Cardnum_|_zip_totalamount14_Amount2
5     Cardnum_|_merchnum_totalamount14_Amount2
11     Cardnum_|_merchnum_totalamount1_Amount2
7      Cardnum_|_merchnum_totalamount3_Amount2
8                Merchnum_totalamount3_Amount2
13               Cardnum_|_state_max14_Amount2
16     Cardnum_|_merchnum_totalamount7_Amount2
14                      Cardnum_mean14_Amount2
9                Cardnum_|_state_max30_Amount2
24          Cardnum_|_zip_totalamount3_Amount2
17                       Merchnum_max7_Amount2
12        Cardnum_|_state_totalamount3_Amount2
21                Cardnum_totalamount0_Amount2
22               Merchnum_totalamount7_Amount2
18        Cardnum_|_state_totalamount1_Amount2
37                 Cardnum_|_zip_max30_Amount2
27            Cardnum_|_merchnum_max14_Amount2
30                       Merchnum_max3_Amount2
25                      Merchnum_mean1_Amount2
26                       Merchnum_max0_Amount2
35               Cardnum_totalamount14_Amount2
23                     Cardnum_median1_Amount2
31    Cardnum_|_merchnum_totalamount30_Amount2
28                Cardnum_totalamount3_Amount2
36                Cardnum_|_state_max7_Amount2
32          Cardnum_|_zip_totalamount0_Amount2
Name: Feat_Name, dtype: object
```

In [ ]:

```
goods_xgb = df_top30vars.loc[df_KSFDR['Fraud'] == 0]
bads_xgb = df_top30vars.loc[df_KSFDR['Fraud'] == 1]
goods_xgb.head()
```

Out[ ]:

	Cardnum_totalamount1_Amount2	Merchnum_totalamount1_Amount2	Cardnum_max14_Amou
3338	0.077781	-0.176272	0.2212
3339	-0.079228	-0.251945	-0.2908
3340	0.316441	-0.118910	0.7014
3341	-0.078589	-0.251113	-0.2908
3342	-0.192924	-0.249207	-0.6408



Run a Boosted Tree 10 Times

The purpose of this loop was to run a boosted tree 10 times with a differen train_test_split for each iteration. The results are saved in a dictionary for further review.

During the project, we manually changed the parameters to see how they performed. We then manually saved the set of 10 runs in a different dictionary.

In [ ]:

```
time_results=pd.datetime.now()
X_goods_xgb = df_top30vars.loc[df_KSFDR['Fraud'] == 0]
y_goods_xgb = df_KSFDR.loc[df_KSFDR['Fraud'] == 0, 'Fraud']
X_bads_xgb = df_top30vars.loc[df_KSFDR['Fraud'] == 1]
y_bads_xgb = df_KSFDR.loc[df_KSFDR['Fraud'] == 1, 'Fraud']
down_goods_size = int(len(goods_xgb)*.75)
# down_goods_size = int(len(bads_xgb))


results_dict={}
oot_FDR_list=[]
train_FDR_list=[]
test_FDR_list=[]
for num in range(1,11):
    curr_time=pd.datetime.now()
    curr_model_name = "xgb_" + str(num)

    results_dict[curr_model_name]={'scores':{},
                                   'data':{},
                                   'FDR':{}
                                  }

    # Downsample majority class
    X_goods_temp = resample(X_goods_xgb, replace=False, n_samples=down_goods_size)
    y_goods_temp = y_goods_xgb.loc[X_goods_temp.index]

    # Create Train and Test sets
    X_train_goods, X_test_goods, y_train_goods, y_test_goods = train_test_split(X_goods_te
mp, y_goods_temp, test_size=0.3, train_size=0.7)
    X_train_bads, X_test_bads, y_train_bads, y_test_bads = train_test_split(X_bads_xgb, y_
bads_xgb, test_size=0.3, train_size=0.7)
    X_train = pd.concat([X_train_goods, X_train_bads])
    X_test = pd.concat([X_test_goods, X_test_bads])
    y_train = pd.concat([y_train_goods, y_train_bads])
    y_test = pd.concat([y_test_goods, y_test_bads])

    # Get the OOT data
    X_oot = df_oot[consolidated_dict['top30'].Feat_Name]
    y_oot = df_oot['Fraud']

    # Calculate ratio for scale_pos_weight parameter
    sum_pos = sum(y_train== 1.0)
    sum_neg = sum(y_train== 0.0)
    ratio = sum_neg / sum_pos

    ### NOTE: The parameters were manually changed for the project since we were manually
 testing different ideas (GridSearchCV caused overfitting)
    # Train a model
    start_xgbfit=pd.datetime.now()
    vars()[curr_model_name] =  xgb.XGBClassifier(objective='binary:logistic', learning_rat
e = 0.01,
                               n_estimators=1500, max_depth=2, scale_pos_weight=ratio,
                               booster='gbtree')
    vars()[curr_model_name].fit(X_train, y_train)
    print("Done training!", pd.datetime.now()-start_xgbfit)
```

```
    # Calculate the accuracy scores of the model
    train_score = vars()[curr_model_name].score(X_train, y_train)
    test_score = vars()[curr_model_name].score(X_test, y_test)
    oot_score = vars()[curr_model_name].score(X_oot, y_oot)

    # Save the accuracy scores of the model
    results_dict[curr_model_name]['scores']['train_score'] = train_score
    results_dict[curr_model_name]['scores']['test_score'] = test_score
    results_dict[curr_model_name]['scores']['oot_score'] = oot_score

    # Calculate the ".predict_proba" and make dataframes for all datasets
    train_data = make_data(vars()[curr_model_name], X_train, y_train)
    test_data = make_data(vars()[curr_model_name], X_test, y_test)
    oot_data = make_data(vars()[curr_model_name], X_oot, y_oot)

    # Save all the dataframes for the model
    results_dict[curr_model_name]['data']['train_data'] = train_data
    results_dict[curr_model_name]['data']['test_data'] = test_data
    results_dict[curr_model_name]['data']['oot_data'] = oot_data

    # Calculate the FDRs
    train_FDR = fdr(train_data, 0.03)
    test_FDR = fdr(test_data, 0.03)
    oot_FDR = fdr(oot_data, 0.03)
    train_FDR_list.append(train_FDR)
    test_FDR_list.append(test_FDR)
    oot_FDR_list.append(oot_FDR)
    print('OOT FDR', oot_FDR)

    # Save the FDRs
    results_dict[curr_model_name]['FDR']['train_FDR'] = train_FDR
    results_dict[curr_model_name]['FDR']['test_FDR'] = test_FDR
    results_dict[curr_model_name]['FDR']['oot_FDR'] = oot_FDR

    print("Done with:",curr_model_name, ";  time:",pd.datetime.now()-curr_time,'\n')

print("DONE WITH ALL!", pd.datetime.now()-time_results)
print('Avg OOT FDR', statistics.mean(oot_FDR_list))
```

```
Done training! 0:01:50.971035
OOT FDR 0.5418994413407822
Done with: xgb_371 ;  time: 0:01:56.452756

Done training! 0:01:58.798694
OOT FDR 0.5195530726256983
Done with: xgb_372 ;  time: 0:02:04.699607

Done training! 0:01:56.468470
OOT FDR 0.5083798882681564
Done with: xgb_373 ;  time: 0:02:01.717486

Done training! 0:02:02.828474
OOT FDR 0.553072625698324
Done with: xgb_374 ;  time: 0:02:08.460225

Done training! 0:02:02.270879
OOT FDR 0.5251396648044693
Done with: xgb_375 ;  time: 0:02:08.094378

Done training! 0:02:07.612490
OOT FDR 0.5195530726256983
Done with: xgb_376 ;  time: 0:02:14.330837

Done training! 0:02:02.578860
OOT FDR 0.5418994413407822
Done with: xgb_377 ;  time: 0:02:08.184658

Done training! 0:02:07.604587
OOT FDR 0.5195530726256983
Done with: xgb_378 ;  time: 0:02:13.381394

Done training! 0:02:05.455949
OOT FDR 0.5307262569832403
Done with: xgb_379 ;  time: 0:02:11.055967

Done training! 0:02:06.827699
OOT FDR 0.5195530726256983
Done with: xgb_380 ;  time: 0:02:12.700923

DONE WITH ALL! 0:21:19.278915
Avg OOT FDR 0.5279329608938548
```

We manually saved each of the 10 iterations that were run.

In [ ]:

```
# # During the first set of 10 runs, this dictionary needs to be instantiated. Comment it
 out after the first set.
# results_report={}
```

In [ ]:

```
# This is where we manually save the set of 10 runs. Simply change the number to whatever
 you need (e.g. 1 to infinity)
results_report[37]={'results_dict':results_dict.copy(),
                    'avg_train_fdr':statistics.mean(train_FDR_list),
                    'avg_test_fdr':statistics.mean(test_FDR_list),
                    'avg_oot_fdr':statistics.mean(oot_FDR_list)}
```

In [ ]:

```
# Have a quick look at the averaged FDR results
print(results_report[37]['avg_train_fdr'])
print(results_report[37]['avg_test_fdr'])
print(results_report[37]['avg_oot_fdr'])
```

```
0.9031301482701812
0.8651340996168582
0.5279329608938548
```

In [ ]:

```
# Have a quick look at the results of the 10 runs
for model in results_dict.keys():
    print(model)
    print(results_dict[model]['scores'])
    print(results_dict[model]['FDR'])
    print("")
```

xgb_1
{'train_score': 0.9631146576277569, 'test_score': 0.9578207381370826, 'oot_sc
ore': 0.9593626780397522}
{'train_FDR': 0.8731466227347611, 'test_FDR': 0.8045977011494253, 'oot_FDR':
0.5865921787709497}

xgb_2
{'train_score': 0.9597250664971871, 'test_score': 0.9588093145869947, 'oot_sc
ore': 0.953246962259596}
{'train_FDR': 0.8879736408566722, 'test_FDR': 0.8390804597701149, 'oot_FDR':
0.5698324022346368}

xgb_3
{'train_score': 0.9599604547701434, 'test_score': 0.9590839191564148, 'oot_sc
ore': 0.952925082481693}
{'train_FDR': 0.8896210873146623, 'test_FDR': 0.8160919540229885, 'oot_FDR':
0.5642458100558659}

xgb_4
{'train_score': 0.9572770284584422, 'test_score': 0.9553492970123023, 'oot_sc
ore': 0.9508328639253238}
{'train_FDR': 0.8747940691927513, 'test_FDR': 0.8275862068965517, 'oot_FDR':
0.5642458100558659}

xgb_5
{'train_score': 0.9581008874137891, 'test_score': 0.9567772407732865, 'oot_sc
ore': 0.9489015852579061}
{'train_FDR': 0.8813838550247117, 'test_FDR': 0.8314176245210728, 'oot_FDR':
0.5698324022346368}

xgb_6
{'train_score': 0.9581008874137891, 'test_score': 0.9611159929701231, 'oot_sc
ore': 0.9555805906493925}
{'train_FDR': 0.8813838550247117, 'test_FDR': 0.8697318007662835, 'oot_FDR':
0.5642458100558659}

xgb_7
{'train_score': 0.9581950427229715, 'test_score': 0.9571067662565905, 'oot_sc
ore': 0.9525227327593144}
{'train_FDR': 0.8879736408566722, 'test_FDR': 0.7931034482758621, 'oot_FDR':
0.553072625698324}

xgb_8
{'train_score': 0.9650683802932938, 'test_score': 0.9645760105448155, 'oot_sc
ore': 0.9614548965961214}
{'train_FDR': 0.9028006589785832, 'test_FDR': 0.789272030651341, 'oot_FDR':
0.5698324022346368}

xgb_9
{'train_score': 0.9615846338535414, 'test_score': 0.9591388400702988, 'oot_sc
ore': 0.952925082481693}
{'train_FDR': 0.8813838550247117, 'test_FDR': 0.789272030651341, 'oot_FDR':
0.5865921787709497}

xgb_10
{'train_score': 0.9590189016783184, 'test_score': 0.9595232864674869, 'oot_sc
ore': 0.9505914540918967}

{'train_FDR': 0.8780889621087314, 'test_FDR': 0.8659003831417624, 'oot_FDR': 0.5754189944134078}

In [ ]:

```
# View the average FDR results for the results stored in the results_report dictionary
train_fdr_result=[]
test_fdr_result=[]
oot_fdr_result=[]
for key in results_report.keys():
    train_fdr_result.append(results_report[key]['avg_train_fdr'])
    test_fdr_result.append(results_report[key]['avg_test_fdr'])
    oot_fdr_result.append(results_report[key]['avg_oot_fdr'])

    print(key)
    print(results_report[key]['avg_train_fdr'])
    print(results_report[key]['avg_test_fdr'])
    print(results_report[key]['avg_oot_fdr'])
    print("")
```

1
0.8838550247116969
0.8226053639846743
0.570391061452514

2
0.9377265238879736
0.8662835249042146
0.5273743016759777

3
0.9752883031301482
0.8842911877394636
0.47039106145251397

4
0.9181219110378913
0.8494252873563218
0.5536312849162011

5
0.956177924217463
0.8954022988505748
0.5078212290502794

6
0.9927512355848435
0.8931034482758621
0.42402234636871505

7
0.9342668863261944
0.8735632183908046
0.5430167597765363

8
0.9789126853377266
0.8931034482758621
0.5150837988826815

9
1.0
0.9103448275862069
0.4435754189944134

10
0.7008237232289951
0.6743295019157088
0.4983240223463687

11
0.7700164744645799
0.717624521072797
0.46033519553072627

12
0.8400329489291598

0.7233716475095785
0.4145251396648045

13
0.7082372322899506
0.667816091954023
0.4983240223463687

14
0.7952224052718286
0.728735632183908
0.4513966480446927

15
0.8622734761120263
0.7459770114942529
0.39608938547486033

16
0.7237232289950577
0.6812260536398468
0.5083798882681564

17
0.8059308072487644
0.7295019157088123
0.46256983240223465

18
0.8616144975288303
0.7590038314176245
0.4134078212290503

19
0.6441515650741351
0.6275862068965518
0.40670391061452515

20
0.7092257001647446
0.6777777777777778
0.4849162011173184

21
0.6632619439868204
0.6383141762452107
0.4776536312849162

22
0.7228995057660627
0.6862068965517242
0.4273743016759777

23
0.6594728171334432
0.6467432950191571
0.4664804469273743

24
0.713838550247117
0.6827586206896552
0.5173184357541899

25
0.9891268533772652
0.8992337164750958
0.45754189944134077

26
1.0
0.9145593869731801
0.4441340782122905

27
0.9
0.8272030651340996
0.4206703910614525

28
1.0
0.9145593869731801
0.42178770949720673

29
0.900164744645799
0.8180076628352491
0.42681564245810055

30
1.0
0.9126436781609195
0.4569832402234637

31
1.0
0.9264367816091954
0.48379888268156424

32
1.0
0.9371647509578545
0.46871508379888266

33
1.0
0.9421455938697318
0.4553072625698324

34
1.0
0.9344827586206896
0.4977653631284916

35

1.0
0.9375478927203065
0.4430167597765363

36
1.0
0.946360153256705
0.4743016759776536

37
0.9031301482701812
0.8651340996168582
0.5279329608938548

In [ ]:

```
# This was a very manual way to create the necessary table for the project report
num_trees=[600,600,600,800,800,800,1000,1000,1000,600,600,600,800,800,800,1000,1000,1000,6
00,600,800,800,1000,1000,600,600,800,800,1000,1000,600,600,800,800,1000,1000,1500]
max_depth=[3,4,5,3,4,5,3,4,5,3,4,5,3,4,5,3,4,3,4,3,4,3,4,3,4,3,4,3,4,3,4,3,4,2]
learning_rate=[0.01,0.01,0.01,0.01,0.01,0.01,0.01,0.01,0.01,0.001,0.001,0.001,0.001,0.001,
0.001,0.001,0.001,0.001,0.0001,0.0001,0.0001,0.0001,0.0001,0.0001,1.0,1.0,1.0,1.0,1.0,1.0,
0.1,0.1,0.1,0.1,0.1,0.1,0.01]

df_results_report = pd.DataFrame.from_dict({'Num of Trees':num_trees,
                                            'Max Depth':max_depth,
                                            'Learning Rate':learning_rate,
                                            'AVG TRAIN FDR':train_fdr_result,
                                            'AVG TEST FDR':test_fdr_result,
                                            'AVG OOT FDR':oot_fdr_result})
df_results_report
```

Out[ ]:

	Num of Trees	Max Depth	Learning Rate	AVG TRAIN FDR	AVG TEST FDR	AVG OOT FDR
0	600	3	0.0100	0.883855	0.822605	0.570391
1	600	4	0.0100	0.937727	0.866284	0.527374
2	600	5	0.0100	0.975288	0.884291	0.470391
3	800	3	0.0100	0.918122	0.849425	0.553631
4	800	4	0.0100	0.956178	0.895402	0.507821
5	800	5	0.0100	0.992751	0.893103	0.424022
6	1000	3	0.0100	0.934267	0.873563	0.543017
7	1000	4	0.0100	0.978913	0.893103	0.515084
8	1000	5	0.0100	1.000000	0.910345	0.443575
9	600	3	0.0010	0.700824	0.674330	0.498324
10	600	4	0.0010	0.770016	0.717625	0.460335
11	600	5	0.0010	0.840033	0.723372	0.414525
12	800	3	0.0010	0.708237	0.667816	0.498324
13	800	4	0.0010	0.795222	0.728736	0.451397
14	800	5	0.0010	0.862273	0.745977	0.396089
15	1000	3	0.0010	0.723723	0.681226	0.508380
16	1000	4	0.0010	0.805931	0.729502	0.462570
17	1000	5	0.0010	0.861614	0.759004	0.413408
18	600	3	0.0001	0.644152	0.627586	0.406704
19	600	4	0.0001	0.709226	0.677778	0.484916
20	800	3	0.0001	0.663262	0.638314	0.477654
21	800	4	0.0001	0.722900	0.686207	0.427374
22	1000	3	0.0001	0.659473	0.646743	0.466480
23	1000	4	0.0001	0.713839	0.682759	0.517318
24	600	3	1.0000	0.989127	0.899234	0.457542
25	600	4	1.0000	1.000000	0.914559	0.444134
26	800	3	1.0000	0.900000	0.827203	0.420670
27	800	4	1.0000	1.000000	0.914559	0.421788
28	1000	3	1.0000	0.900165	0.818008	0.426816
29	1000	4	1.0000	1.000000	0.912644	0.456983
30	600	3	0.1000	1.000000	0.926437	0.483799
31	600	4	0.1000	1.000000	0.937165	0.468715
32	800	3	0.1000	1.000000	0.942146	0.455307
33	800	4	0.1000	1.000000	0.934483	0.497765

	Num of Trees	Max Depth	Learning Rate	AVG TRAIN FDR	AVG TEST FDR	AVG OOT FDR
34	1000	3	0.1000	1.000000	0.937548	0.443017
35	1000	4	0.1000	1.000000	0.946360	0.474302
36	1500	2	0.0100	0.903130	0.865134	0.527933

In [ ]:

```
# # Save the table for the report
# df_results_report.to_csv('xgb_results_report.csv')
```

Neural Networks

Cross Validation results for Keras neural net

dataset choice

In [ ]:

```
max_no_features = 30
current_columns = col_KS_FDR
# current_columns = cols_WD_ED_FS
# current_columns = svc_L1_cols
# current_columns = rfe_cols80
# current_columns = DT_RF_XGB_cols_80

# split into input (X) and output (Y) variables
X = feature_select_df[current_columns[0:max_no_features]].values
Y = feature_select_df['Fraud'].values
X_oot = OOT_df[current_columns[0:max_no_features]].values
y_oot = OOT_df["Fraud"].values
```

In [ ]:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X = scaler.fit_transform(X)

X_oot = scaler.fit_transform(X_oot)
# verify results
X_oot.std(axis=0)
```

Out[ ]:

```
array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])
```

In [ ]:

```
X = np.clip(X, -5, 5)
X_oot = np.clip(X_oot, -5, 5)
```

Out[ ]:

```
array([-0.16414568, -0.14985067, -0.15899912, -0.1750719 , -0.14671968,
       -0.17080219, -0.1832474 , -0.20344669, -0.15475512, -0.1406809 ,
       -0.14282392, -0.20497747, -0.24629319, -0.21879987, -0.4616672 ,
       -0.18981744, -0.199151  , -0.19177528, -0.4871884 , -0.48009136,
       -0.45253676, -0.43079224, -0.50895226, -0.44652182, -0.47449493,
       -0.23871565, -0.42685482, -0.44010672, -0.45556587, -0.24424374],
      dtype=float32)
```

In [ ]:

```
import tensorflow as tf
from tensorflow import keras

METRICS = [
      keras.metrics.TruePositives(name='tp'),
      keras.metrics.FalsePositives(name='fp'),
      keras.metrics.TrueNegatives(name='tn'),
      keras.metrics.FalseNegatives(name='fn'),
      keras.metrics.BinaryAccuracy(name='accuracy'),
      keras.metrics.Precision(name='precision'),
      keras.metrics.Recall(name='recall'),
      keras.metrics.AUC(name='auc'),
]
```

In [ ]:

```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import StratifiedShuffleSplit
from keras.optimizers import SGD
from keras.optimizers import Adam

from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler
from imblearn.pipeline import Pipeline

# import regularizer
from keras.regularizers import l1 ,l2

# ----------- sampling minority class ----------------
# neg0, pos0 = np.bincount(Y)
# print('Or count of neg:',neg0,'of pos',pos0)
# # define pipeline
# over = SMOTE(sampling_strategy=0.1)
# under = RandomUnderSampler(sampling_strategy=0.5)
# steps = [('o', over), ('u', under)]
# pipeline = Pipeline(steps=steps)
# # transform the dataset
# X, Y = pipeline.fit_resample(X, Y)
# ------------------------------------------------------

neg, pos = np.bincount(Y)
total = neg + pos

print('New count of neg:',neg,'of pos',pos)

# define opt function----------------------------------
# sgd = SGD(lr=0.001)#, decay=1e-6, momentum=0.9, nesterov=True)
# lr = [1, 0.1, 0.0.1, 0.001, 0.0001] 0.001 is default
adam_par = Adam(learning_rate= 0.0005)#, beta_1=0.9, beta_2=0.999, amsgrad=False)
# # ------------------------------------------------------

# instantiate regularizer----------------------------------
reg_l1 = l1(0.1)
reg_l2 = l2(5e-2)
# # ------------------------------------------------------

class_weights = {0: 0.5, 1: 8.5}
# # optimal class weights-----------------------------------
# weight_for_0 = (1 / neg)*(total)/2.0
# weight_for_1 = (1 / pos)*(total)/2.0
# class_weights_opt = {0: weight_for_0, 1: weight_for_1}
# # ------------------------------------------------------
DROPOUT_RATE = 0.25

initial_bias = tf.keras.initializers.Constant(np.log([pos/neg]))

# define 10-fold cross validation test harness
kfold = StratifiedKFold(n_splits=8, shuffle=True)
```

```
# sskfold = StratifiedShuffleSplit(n_splits=5, test_size=0.5)

cvscores_FDR_train = []
cvscores_FDR_test = []
cvscores_FDR_oot = []

start_time = pd.datetime.now()
for train, test in kfold.split(X, Y):
# for train, test in sskfold.split(X, Y):

    model = Sequential()
    model.add(Dense(20, input_dim=X.shape[1], activation='relu'))#, activity_regularizer=r
eg_l2))
#     model.add(Dropout(DROPOUT_RATE))
#     ---------------added hidden layers---------------------
#     model.add(Dense(30, activation='relu'))#, activity_regularizer=reg_l2))
#     model.add(Dropout(DROPOUT_RATE))
#     model.add(Dense(10, activation='relu'))#, activity_regularizer=reg_l1))
#     model.add(Dropout(DROPOUT_RATE))
#     model.add(Dense(2, activation='tanh'))#, activity_regularizer=reg_l1))
#     model.add(Dropout(DROPOUT_RATE))
#     model.add(Dense(5, activation='tanh'))#, activity_regularizer=reg_l1))
#     model.add(Dropout(DROPOUT_RATE))
# -------------------------------------------------
    model.add(Dense(1, activation='sigmoid'))#,
#                           bias_initializer=initial_bias))
    # Compile model
    model.compile(loss='binary_crossentropy', optimizer=adam_par, metrics=[METRICS])
    # Fit the model
    model.fit(X[train], Y[train], epochs=300, batch_size=2048, verbose=0,class_weight=clas
s_weights)

    y_train_pred = np.squeeze(model.predict(X[train], verbose=0))
    y_test_pred = np.squeeze(model.predict(X[test], verbose=0))
    y_oot_pred = np.squeeze(model.predict(X_oot, verbose=0))

    FDR_train = 100*get_FDR(y_train_pred, Y[train])
    FDR_test = 100*get_FDR(y_test_pred, Y[test])
    FDR_OOT = 100*get_FDR(y_oot_pred,y_oot)

    cvscores_FDR_train.append(FDR_train)
    cvscores_FDR_test.append(FDR_test)
    cvscores_FDR_oot.append(FDR_OOT)
    print('Train FDR:',FDR_train,'Test FDR:',FDR_test, 'OOT FDR',FDR_OOT)

print("Train scores: %.2f%% (+/- %.2f%%)" % (np.mean(cvscores_FDR_train), np.std(cvscores_
FDR_train)))
print("Test scores: %.2f%% (+/- %.2f%%)" % (np.mean(cvscores_FDR_test), np.std(cvscores_FD
R_test)))
print("OOT scores: %.2f%% (+/- %.2f%%)" % (np.mean(cvscores_FDR_oot), np.std(cvscores_FDR_
oot)))

print('duration: ', pd.datetime.now() - start_time)
```

```
New count of neg: 79764 of pos 868
Train FDR: 80.65789473684211 Test FDR: 73.14814814814815 OOT FDR 56.424581005
586596
Train FDR: 81.05263157894737 Test FDR: 76.85185185185185 OOT FDR 54.189944134
07822
Train FDR: 80.52631578947368 Test FDR: 81.48148148148148 OOT FDR 51.396648044
69274
Train FDR: 80.13157894736842 Test FDR: 75.92592592592592 OOT FDR 52.513966480
446925
Train FDR: 80.50065876152833 Test FDR: 77.06422018348624 OOT FDR 56.424581005
586596
Train FDR: 79.97364953886694 Test FDR: 81.65137614678899 OOT FDR 53.072625698
324025
Train FDR: 80.50065876152833 Test FDR: 77.98165137614679 OOT FDR 56.983240223
46368
Train FDR: 79.31488801054019 Test FDR: 85.3211009174312 OOT FDR 54.7486033519
553
Train scores: 80.33% (+/- 0.49%)
Test scores: 78.68% (+/- 3.63%)
OOT scores: 54.47% (+/- 1.91%)
duration:  0:53:41.132467
```

Required Plots

In [ ]:

```
# Import the datasets from Mrinal
df_pred_all = pd.read_csv('trn_tst_oot.csv').sort_values('Recnum', ascending=True)
df_pred_trn_test = pd.read_csv('trn_tst.csv').sort_values('Predict Proba', ascending=False
)
df_pred_trn = pd.read_csv('train.csv').sort_values('Predict Proba', ascending=False)
df_pred_tst = pd.read_csv('test.csv').sort_values('Predict Proba', ascending=False)
df_pred_oot = pd.read_csv('oot.csv').sort_values('Predict Proba', ascending=False)
```

In [ ]:

```
# Get the Recnum, fraud scores (Predict Proba), and predicted value from the imported data sets
df_pred_all_scores = df_pred_all[['Recnum','Predict Proba','Predicted']]
df_pred_all_scores
```

Out[ ]:

	Recnum	Predict Proba	Predicted
58853	3345	0.004879	0
29494	3346	0.001278	0
61662	3347	0.003077	0
29338	3348	0.001278	0
10787	3349	0.001270	0
...	...	...	...
93054	96749	0.001272	0
93055	96750	0.001907	0
93056	96751	0.002581	0
93057	96752	0.007735	0
93058	96753	0.074006	0

93059 rows × 3 columns

In [ ]:

```
# Merge the fraud score info (df_pred_all_scores) with the original dataset (df_original)
df_orig_scored = pd.merge(df_original.copy(), df_pred_all_scores, how='outer', on='Recnum'
)
df_orig_scored
```

Out[ ]:

	Recnum	Cardnum	Date	Merchnum	Merch description	Merch state	Merch zip	Transtype
0	1	5142190439	2010-01-01	5509006296254	FEDEX SHP 12/23/09 AB#	TN	38118.0	P
1	2	5142183973	2010-01-01	61003026333	SERVICE MERCHANDISE #81	MA	1803.0	P
2	3	5142131721	2010-01-01	4503082993600	OFFICE DEPOT #191	MD	20706.0	P
3	4	5142148452	2010-01-01	5509006296254	FEDEX SHP 12/28/09 AB#	TN	38118.0	P
4	5	5142190439	2010-01-01	5509006296254	FEDEX SHP 12/23/09 AB#	TN	38118.0	P
...	...	...	...	...	...	...	...	...
96748	96749	5142276053	2010-12-31	3500000006160	BEST BUY 00001610	KY	41042.0	P
96749	96750	5142225701	2010-12-31	8090710030950	MARKUS OFFICE SUPPLIES	OH	45248.0	P
96750	96751	5142226486	2010-12-31	4503057341100	TECH PAC, INC	OH	45150.0	P
96751	96752	5142244619	2010-12-31	8834000695412	BUY.COM	CA	92656.0	P
96752	96753	5142243247	2010-12-31	9108347680006	STAPLES NATIONAL #471	NJ	7606.0	P

96753 rows × 12 columns

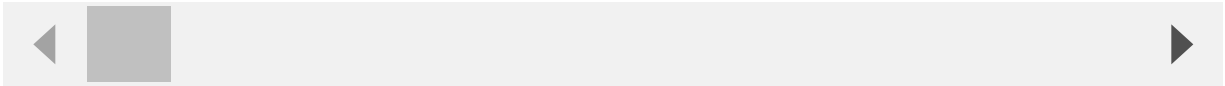
In [ ]:

```
# Merge the fraud score info (df_pred_all_scores) with the variables we created (df_final)
df_final_scored = pd.merge(df_final.copy(), df_pred_all_scores, how='outer', on='Recnum')
df_final_scored
```

Out[ ]:

	Recnum	Cardnum	Date	Merchnum	Merch description	Merch state	Merch zip	Transtype	A
0	1	5142190439	2010-01-01	5509006296254	FEDEX SHP 12/23/09 AB#	TN	38118	P	
1	2	5142183973	2010-01-01	61003026333	SERVICE MERCHANDISE #81	MA	1803	P	
2	3	5142131721	2010-01-01	4503082993600	OFFICE DEPOT #191	MD	20706	P	
3	4	5142148452	2010-01-01	5509006296254	FEDEX SHP 12/28/09 AB#	TN	38118	P	
4	5	5142190439	2010-01-01	5509006296254	FEDEX SHP 12/23/09 AB#	TN	38118	P	
...	...	...	...	...	...	...	...	...	
96392	96749	5142276053	2010-12-31	3500000006160	BEST BUY 00001610	KY	41042	P	
96393	96750	5142225701	2010-12-31	8090710030950	MARKUS OFFICE SUPPLIES	OH	45248	P	
96394	96751	5142226486	2010-12-31	4503057341100	TECH PAC, INC	OH	45150	P	
96395	96752	5142244619	2010-12-31	8834000695412	BUY.COM	CA	92656	P	2
96396	96753	5142243247	2010-12-31	9108347680006	STAPLES NATIONAL #471	NJ	7606	P	

96397 rows × 75 columns



In [ ]:

```
# Find a Recnum in the test results that had values of 1 for Predicted, Predict Proba (fra
ud score, and Fraud (true value)
tst_fraud = df_pred_tst.loc[(df_pred_tst['Predicted']==1) & (df_pred_tst['Fraud']==1) & (d
f_pred_tst['Predict Proba']==1.0)]
tst_fraud_recnums = tst_fraud['Recnum']
tst_fraud
```

Out[ ]:

	Recnum	Predicted	Predict Proba	Fraud
452	59934	1	1.0	1
8271	59720	1	1.0	1
10091	59999	1	1.0	1
10099	59918	1	1.0	1
17431	59911	1	1.0	1
8320	70608	1	1.0	1
12496	60028	1	1.0	1
13576	31294	1	1.0	1
197	62535	1	1.0	1
14499	70756	1	1.0	1
1309	31463	1	1.0	1
358	59750	1	1.0	1
11452	42200	1	1.0	1
2611	62556	1	1.0	1
22978	42187	1	1.0	1
7322	31399	1	1.0	1
15268	70528	1	1.0	1
22633	80512	1	1.0	1
8619	80500	1	1.0	1
13520	35924	1	1.0	1
16333	31667	1	1.0	1

In [ ]:

```
# Determine the Recnum values in the original dataset that match what we wanted for tst_fr
aud_recnums (see previous cell)
df_tst_fraud = df_orig_scored.loc[df_orig_scored['Recnum'].isin(tst_fraud_recnums)]
print(df_tst_fraud['Cardnum'].unique())
print(df_tst_fraud['Merchnum'].unique())
```

```
[5142179617 5142271065 5142111125 5142138135 5142182128 5142220919
 5142183973]
['253052983001' '4620009957157' '5725000466504' '4620009750086'
 '900009045549' '8730008064698' '3831009006589' '17519283486'
 '6848556518938' '5900000064440' '6005030600003']
```

In [ ]:

```
# View the fraud score information and the variables for the different Merchnum values dis
played above
# 5725000466504, 4353000719908
merchnum_to_plot = df_final_scored.loc[df_final_scored['Merchnum']=='4353000719908']
merchnum_to_plot = merchnum_to_plot[['Cardnum','Merchnum','Date','Predict Proba','Merchnum
_freq0_Date','Merchnum_freq1_Date','Merchnum_freq7_Date','Merchnum_freq30_Date']]
merchnum_to_plot['Predict Proba'].fillna(0, inplace=True)
merchnum_to_plot
```

Out[ ]:

	Cardnum	Merchnum	Date	Predict Proba	Merchnum_freq0_Date	Merchnum_freq1_Date
23	5142217905	4353000719908	2010-01-01	0.000000	1	1
157	5142162882	4353000719908	2010-01-03	0.000000	1	1
183	5142217905	4353000719908	2010-01-03	0.000000	2	2
198	5142162882	4353000719908	2010-01-03	0.000000	3	3
248	5142137916	4353000719908	2010-01-04	0.000000	1	4
...	...	...	...	...	...	..
96174	5142146217	4353000719908	2010-12-30	0.001304	1	3
96209	5142190418	4353000719908	2010-12-30	0.001278	2	4
96268	5142178848	4353000719908	2010-12-30	0.001289	3	5
96316	5142217905	4353000719908	2010-12-30	0.001269	4	6
96335	5142236933	4353000719908	2010-12-30	0.001318	5	7

1022 rows × 8 columns

In [ ]:

```
# Create a dataframe for the selected Merchnum to use with the presentation slide
merchnum_slide = df_orig_scored.loc[(df_orig_scored['Merchnum']=='4353000719908') & (df_orig_scored['Fraud']==1)]
merchnum_slide.rename(columns={'Predict Proba':'Fraud Score'}, inplace=True)
merchnum_slide
```

```
C:\Users\chrissy\AppData\Roaming\Python\Python36\site-packages\pandas\core\fr
ame.py:4238: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  return super().rename(**kwargs)
```

Out[ ]:

	Recnum	Cardnum	Date	Merchnum	Merch description	Merch state	Merch zip	Transtyp
19995	19996	5142214614	2010-03-20	4353000719908	AMAZON.COM *SUPERSTOR	WA	98101.0	
39770	39771	5142116864	2010-05-27	4353000719908	AMAZON.COM *SUPERSTOR	WA	98101.0	
39805	39806	5142116864	2010-05-27	4353000719908	AMAZON.COM *SUPERSTOR	WA	98101.0	
39815	39816	5142116864	2010-05-27	4353000719908	AMAZON.COM *SUPERSTOR	WA	98101.0	
39832	39833	5142116864	2010-05-27	4353000719908	AMAZON.COM *SUPERSTOR	WA	98101.0	
39834	39835	5142116864	2010-05-27	4353000719908	AMAZON.COM *SUPERSTOR	WA	98101.0	
39847	39848	5142116864	2010-05-27	4353000719908	AMAZON.COM *SUPERSTOR	WA	98101.0	
39853	39854	5142116864	2010-05-27	4353000719908	AMAZON.COM *SUPERSTOR	WA	98101.0	
39858	39859	5142116864	2010-05-27	4353000719908	AMAZON.COM *SUPERSTOR	WA	98101.0	
39866	39867	5142116864	2010-05-27	4353000719908	AMAZON.COM *SUPERSTOR	WA	98101.0	
39892	39893	5142116864	2010-05-28	4353000719908	AMAZON.COM *SUPERSTOR	WA	98101.0	
39899	39900	5142116864	2010-05-28	4353000719908	AMAZON.COM *SUPERSTOR	WA	98101.0	
39900	39901	5142116864	2010-05-28	4353000719908	AMAZON.COM *SUPERSTOR	WA	98101.0	
39902	39903	5142116864	2010-05-28	4353000719908	AMAZON.COM *SUPERSTOR	WA	98101.0	
39908	39909	5142116864	2010-05-28	4353000719908	AMAZON.COM *SUPERSTOR	WA	98101.0	
39912	39913	5142116864	2010-05-28	4353000719908	AMAZON.COM *SUPERSTOR	WA	98101.0	
39914	39915	5142116864	2010-05-28	4353000719908	AMAZON.COM *SUPERSTOR	WA	98101.0	
39920	39921	5142116864	2010-05-28	4353000719908	AMAZON.COM *SUPERSTOR	WA	98101.0	
39921	39922	5142116864	2010-05-28	4353000719908	AMAZON.COM *SUPERSTOR	WA	98101.0	
39922	39923	5142116864	2010-05-28	4353000719908	AMAZON.COM *SUPERSTOR	WA	98101.0	
39924	39925	5142116864	2010-05-28	4353000719908	AMAZON.COM *SUPERSTOR	WA	98101.0	
39930	39931	5142116864	2010-05-29	4353000719908	AMAZON.COM *SUPERSTOR	WA	98101.0	

	Recnum	Cardnum	Date	Merchnum	Merch description	Merch state	Merch zip	Transtyp
39939	39940	5142116864	2010-05-29	4353000719908	AMAZON.COM *SUPERSTOR	WA	98101.0	
39955	39956	5142116864	2010-05-29	4353000719908	AMAZON.COM *SUPERSTOR	WA	98101.0	
39959	39960	5142116864	2010-05-29	4353000719908	AMAZON.COM *SUPERSTOR	WA	98101.0	
39960	39961	5142116864	2010-05-29	4353000719908	AMAZON.COM *SUPERSTOR	WA	98101.0	
39966	39967	5142116864	2010-05-29	4353000719908	AMAZON.COM *SUPERSTOR	WA	98101.0	
39976	39977	5142116864	2010-05-29	4353000719908	AMAZON.COM *SUPERSTOR	WA	98101.0	
87587	87588	5142206786	2010-11-17	4353000719908	AMAZON.COM *SUPERSTOR	WA	98101.0	
87686	87687	5142206786	2010-11-17	4353000719908	AMAZON.COM *SUPERSTOR	WA	98101.0	
89364	89365	5142235211	2010-11-25	4353000719908	AMAZON.COM *SUPERSTOR	WA	98101.0	
89365	89366	5142235211	2010-11-25	4353000719908	ACI*AMAZON.COM INC	WA	98101.0	
89366	89367	5142235211	2010-11-25	4353000719908	AMAZON.COM *SUPERSTOR	WA	98101.0	
89368	89369	5142235211	2010-11-25	4353000719908	AMAZON.COM *SUPERSTOR	WA	98101.0	
89374	89375	5142235211	2010-11-25	4353000719908	AMAZON.COM *SUPERSTOR	WA	98101.0	
89377	89378	5142235211	2010-11-25	4353000719908	AMAZON.COM *SUPERSTOR	WA	98101.0	
89380	89381	5142235211	2010-11-25	4353000719908	ACI*AMAZON.COM INC	WA	98101.0	
89383	89384	5142235211	2010-11-25	4353000719908	ACI*AMAZON.COM INC	WA	98101.0	
89391	89392	5142235211	2010-11-25	4353000719908	AMAZON.COM *SUPERSTOR	WA	98101.0	
89392	89393	5142235211	2010-11-25	4353000719908	AMAZON.COM *SUPERSTOR	WA	98101.0	
89398	89399	5142235211	2010-11-25	4353000719908	AMAZON.COM *SUPERSTOR	WA	98101.0	
89400	89401	5142235211	2010-11-25	4353000719908	AMAZON.COM *SUPERSTOR	WA	98101.0	
89403	89404	5142235211	2010-11-25	4353000719908	AMAZON.COM *SUPERSTOR	WA	98101.0	
89411	89412	5142235211	2010-11-25	4353000719908	AMAZON.COM *SUPERSTOR	WA	98101.0	
89417	89418	5142235211	2010-11-25	4353000719908	AMAZON.COM *SUPERSTOR	WA	98101.0	

	Recnum	Cardnum	Date	Merchnum	Merch description	Merch state	Merch zip	Transtyp
89418	89419	5142235211	2010-11-25	4353000719908	AMAZON.COM *SUPERSTRE	WA	98101.0	
89420	89421	5142235211	2010-11-25	4353000719908	AMAZON.COM *SUPERSTOR	WA	98101.0	
89425	89426	5142235211	2010-11-26	4353000719908	AMAZON.COM *SUPERSTOR	WA	98101.0	
89426	89427	5142235211	2010-11-26	4353000719908	AMAZON.COM *SUPERSTOR	WA	98101.0	
89434	89435	5142235211	2010-11-26	4353000719908	ACI*AMAZON.COM INC	WA	98101.0	
89456	89457	5142235211	2010-11-26	4353000719908	ACI*AMAZON.COM INC	WA	98101.0	
89458	89459	5142235211	2010-11-26	4353000719908	AMAZON.COM *SUPERSTOR	WA	98101.0	
89461	89462	5142235211	2010-11-26	4353000719908	AMAZON.COM *SUPERSTOR	WA	98101.0	
89464	89465	5142235211	2010-11-26	4353000719908	AMAZON.COM *SUPERSTOR	WA	98101.0	
89465	89466	5142235211	2010-11-26	4353000719908	AMAZON.COM *SUPERSTOR	WA	98101.0	
89472	89473	5142235211	2010-11-26	4353000719908	AMAZON.COM *SUPERSTOR	WA	98101.0	
89473	89474	5142235211	2010-11-26	4353000719908	AMAZON.COM *SUPERSTRE	WA	98101.0	
89474	89475	5142235211	2010-11-26	4353000719908	AMAZON.COM *SUPERSTOR	WA	98101.0	
89478	89479	5142235211	2010-11-26	4353000719908	AMAZON.COM *SUPERSTRE	WA	98101.0	
89519	89520	5142235211	2010-11-26	4353000719908	AMAZON.COM *SUPERSTOR	WA	98101.0	
89528	89529	5142235211	2010-11-26	4353000719908	AMAZON.COM *SUPERSTOR	WA	98101.0	
89531	89532	5142235211	2010-11-26	4353000719908	AMAZON.COM *SUPERSTOR	WA	98101.0	
90785	90786	5142199009	2010-12-03	4353000719908	ACI*AMAZON.COM INC	WA	98101.0	
90844	90845	5142199009	2010-12-03	4353000719908	ACI*AMAZON.COM INC	WA	98101.0	
91010	91011	5142199009	2010-12-03	4353000719908	ACI*AMAZON.COM INC	WA	98101.0	
91108	91109	5142199009	2010-12-05	4353000719908	ACI*AMAZON.COM INC	WA	98101.0	
91208	91209	5142199009	2010-12-06	4353000719908	ACI*AMAZON.COM INC	WA	98101.0	
91217	91218	5142199009	2010-12-06	4353000719908	ACI*AMAZON.COM INC	WA	98101.0	

	Recnum	Cardnum	Date	Merchnum	Merch description	Merch state	Merch zip	Transtyp
91269	91270	5142199009	2010-12-06	4353000719908	ACI*AMAZON.COM INC	WA	98101.0	
91478	91479	5142199009	2010-12-07	4353000719908	ACI*AMAZON.COM INC	WA	98101.0	
91607	91608	5142199009	2010-12-07	4353000719908	ACI*AMAZON.COM INC	WA	98101.0	
91757	91758	5142199009	2010-12-08	4353000719908	ACI*AMAZON.COM INC	WA	98101.0	
92090	92091	5142199009	2010-12-09	4353000719908	ACI*AMAZON.COM INC	WA	98101.0	
92340	92341	5142199009	2010-12-09	4353000719908	ACI*AMAZON.COM INC	WA	98101.0	
92694	92695	5142199009	2010-12-11	4353000719908	ACI*AMAZON.COM INC	WA	98101.0	
92702	92703	5142199009	2010-12-11	4353000719908	ACI*AMAZON.COM INC	WA	98101.0	
92741	92742	5142199009	2010-12-12	4353000719908	ACI*AMAZON.COM INC	WA	98101.0	
92745	92746	5142199009	2010-12-12	4353000719908	ACI*AMAZON.COM INC	WA	98101.0	
92862	92863	5142199009	2010-12-13	4353000719908	ACI*AMAZON.COM INC	WA	98101.0	
92866	92867	5142199009	2010-12-13	4353000719908	ACI*AMAZON.COM INC	WA	98101.0	
92896	92897	5142199009	2010-12-13	4353000719908	ACI*AMAZON.COM INC	WA	98101.0	
92951	92952	5142199009	2010-12-13	4353000719908	ACI*AMAZON.COM INC	WA	98101.0	
93119	93120	5142199009	2010-12-14	4353000719908	ACI*AMAZON.COM INC	WA	98101.0	
93168	93169	5142199009	2010-12-14	4353000719908	ACI*AMAZON.COM INC	WA	98101.0	
93251	93252	5142199009	2010-12-14	4353000719908	ACI*AMAZON.COM INC	WA	98101.0	
93595	93596	5142199009	2010-12-15	4353000719908	ACI*AMAZON.COM INC	WA	98101.0	
93672	93673	5142199009	2010-12-15	4353000719908	ACI*AMAZON.COM INC	WA	98101.0	
93905	93906	5142199009	2010-12-16	4353000719908	ACI*AMAZON.COM INC	WA	98101.0	
93976	93977	5142199009	2010-12-16	4353000719908	ACI*AMAZON.COM INC	WA	98101.0	
94046	94047	5142199009	2010-12-16	4353000719908	ACI*AMAZON.COM INC	WA	98101.0	
94091	94092	5142199009	2010-12-16	4353000719908	ACI*AMAZON.COM INC	WA	98101.0	

	Recnum	Cardnum	Date	Merchnum	Merch description	Merch state	Merch zip	Transtyp
94102	94103	5142199009	2010-12-16	4353000719908	ACI*AMAZON.COM INC	WA	98101.0	
94104	94105	5142199009	2010-12-16	4353000719908	ACI*AMAZON.COM INC	WA	98101.0	
94568	94569	5142199009	2010-12-19	4353000719908	ACI*AMAZON.COM INC	WA	98101.0	
94576	94577	5142199009	2010-12-19	4353000719908	ACI*AMAZON.COM INC	WA	98101.0	
94581	94582	5142199009	2010-12-19	4353000719908	ACI*AMAZON.COM INC	WA	98101.0	
94914	94915	5142199009	2010-12-21	4353000719908	ACI*AMAZON.COM INC	WA	98101.0	
94923	94924	5142199009	2010-12-21	4353000719908	ACI*AMAZON.COM INC	WA	98101.0	
94981	94982	5142199009	2010-12-21	4353000719908	ACI*AMAZON.COM INC	WA	98101.0	
95043	95044	5142199009	2010-12-21	4353000719908	ACI*AMAZON.COM INC	WA	98101.0	
95157	95158	5142199009	2010-12-21	4353000719908	ACI*AMAZON.COM INC	WA	98101.0	
95220	95221	5142199009	2010-12-22	4353000719908	ACI*AMAZON.COM INC	WA	98101.0	
95276	95277	5142199009	2010-12-22	4353000719908	ACI*AMAZON.COM INC	WA	98101.0	
95517	95518	5142199009	2010-12-22	4353000719908	ACI*AMAZON.COM INC	WA	98101.0	
95553	95554	5142199009	2010-12-23	4353000719908	AMAZON.COM *SUPERSTRE	WA	98101.0	
95554	95555	5142199009	2010-12-23	4353000719908	ACI*AMAZON.COM INC	WA	98101.0	
95810	95811	5142199009	2010-12-25	4353000719908	AMAZON.COM *SUPERSTRE	WA	98101.0	

In [ ]:

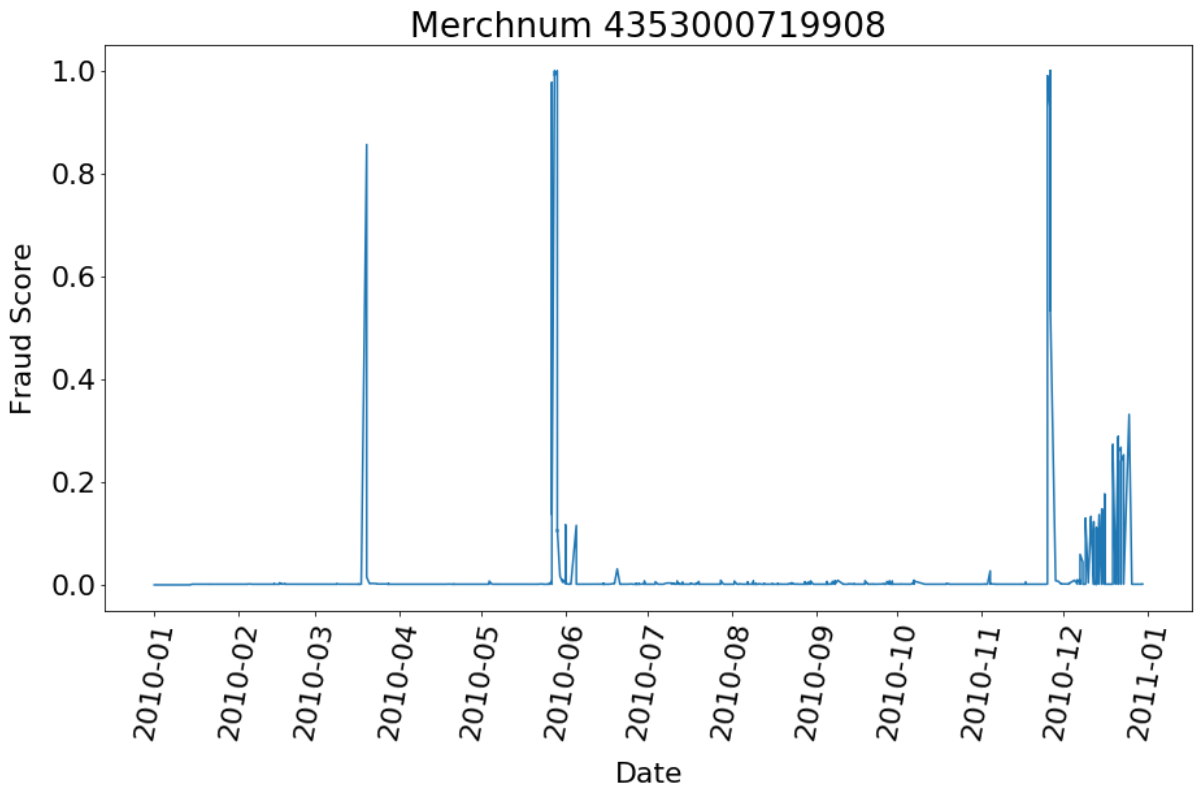
```
# # Save the Merchnum dataframe for the presentation slide
# merchnum_slide.to_csv('merchnum_slide.csv')
```

In [ ]:

```
plt.rcParams["figure.figsize"] = [15,8]
plt.rcParams['font.size'] = 22
fig = plt.figure()
ax = plt.axes()

plt.plot(merchnum_to_plot['Date'], merchnum_to_plot['Predict Proba'])
months = mdates.MonthLocator()  # every month
ax.xaxis.set_major_locator(months)
plt.xticks(rotation='80')
plt.xlabel('Date', labelpad=15)
plt.ylabel('Fraud Score', labelpad=10)
plt.title("Merchnum 4353000719908")
```

Out[ ]:

```
Text(0.5, 1.0, 'Merchnum 4353000719908')
```

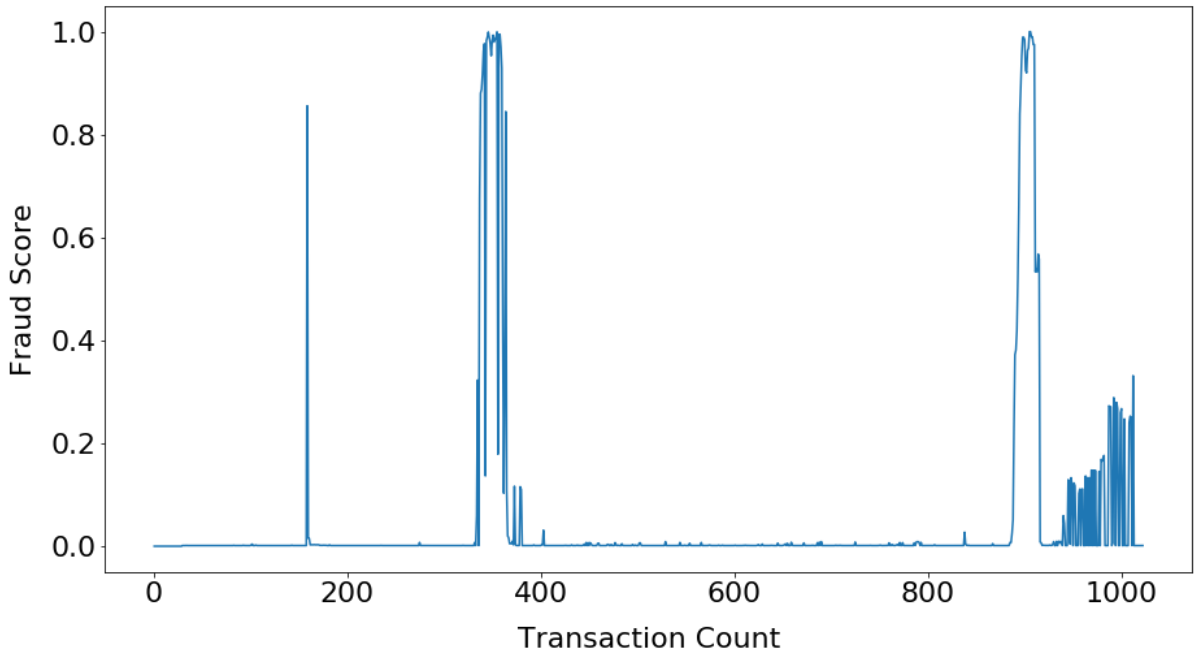


In [ ]:

```
plt.rcParams["figure.figsize"] = [15,8]
plt.rcParams['font.size'] = 22
fig = plt.figure()
ax = plt.axes()

plt.plot(range(1, len(merchnum_to_plot)+1), merchnum_to_plot['Predict Proba'])
# plt.xticks()
plt.xlabel('Transaction Count', labelpad=15)
plt.ylabel('Fraud Score', labelpad=10)
# plt.title("Merchnum 5725000466504 (CDWG, Inc.)")
```

Out[ ]:

```
Text(0, 0.5, 'Fraud Score')
```



In [ ]:

```
# Find the burst of activity
merchnum_slide.groupby(['Date'])['Merchnum'].count()
```

Out[ ]:

```
Date
2010-03-20     1
2010-05-27     9
2010-05-28    11
2010-05-29     7
2010-11-17     2
2010-11-25    17
2010-11-26    15
2010-12-03     3
2010-12-05     1
2010-12-06     3
2010-12-07     2
2010-12-08     1
2010-12-09     2
2010-12-11     2
2010-12-12     2
2010-12-13     4
2010-12-14     3
2010-12-15     2
2010-12-16     6
2010-12-19     3
2010-12-21     5
2010-12-22     3
2010-12-23     2
2010-12-25     1
Name: Merchnum, dtype: int64
```

In [ ]:

```
# Find the burst of activity
merchnum_to_plot.groupby(['Date'])['Merchnum'].count()
```

Out[ ]:

Date	
2010-01-01	1
2010-01-03	3
2010-01-04	2
2010-01-05	3
2010-01-06	2
2010-01-07	1
2010-01-10	1
2010-01-11	4
2010-01-12	6
2010-01-13	1
2010-01-14	5
2010-01-15	1
2010-01-16	1
2010-01-18	3
2010-01-19	2
2010-01-20	5
2010-01-21	5
2010-01-24	3
2010-01-25	1
2010-01-26	3
2010-01-27	4
2010-01-28	5
2010-01-29	1
2010-01-31	3
2010-02-01	2
2010-02-02	5
2010-02-03	3
2010-02-04	6
2010-02-05	1
2010-02-06	2
2010-02-07	1
2010-02-08	1
2010-02-09	1
2010-02-10	3
2010-02-14	3
2010-02-15	3
2010-02-16	5
2010-02-17	3
2010-02-18	3
2010-02-19	2
2010-02-23	3
2010-02-25	3
2010-02-28	3
2010-03-02	3
2010-03-03	3
2010-03-06	3
2010-03-07	6
2010-03-08	3
2010-03-09	6
2010-03-10	1
2010-03-11	1
2010-03-13	1
2010-03-14	2
2010-03-16	5

2010-03-17	4
2010-03-18	1
2010-03-20	3
2010-03-21	6
2010-03-22	2
2010-03-23	2
2010-03-24	3
2010-03-25	1
2010-03-26	1
2010-03-27	3
2010-03-28	4
2010-03-29	8
2010-03-30	1
2010-03-31	2
2010-04-03	1
2010-04-04	4
2010-04-05	1
2010-04-06	3
2010-04-07	4
2010-04-08	1
2010-04-09	1
2010-04-10	1
2010-04-11	7
2010-04-12	4
2010-04-13	4
2010-04-14	3
2010-04-15	2
2010-04-17	1
2010-04-18	3
2010-04-19	1
2010-04-20	4
2010-04-21	3
2010-04-22	1
2010-04-23	1
2010-04-25	3
2010-04-26	4
2010-04-27	4
2010-04-28	4
2010-05-01	4
2010-05-02	6
2010-05-04	6
2010-05-05	1
2010-05-06	4
2010-05-07	2
2010-05-10	5
2010-05-11	5
2010-05-12	4
2010-05-13	1
2010-05-15	3
2010-05-16	7
2010-05-18	3
2010-05-19	3
2010-05-20	1
2010-05-22	5
2010-05-23	1
2010-05-24	4
2010-05-25	4

2010-05-26	3
2010-05-27	12
2010-05-28	11
2010-05-29	11
2010-05-30	2
2010-05-31	4
2010-06-01	3
2010-06-02	2
2010-06-03	2
2010-06-05	3
2010-06-06	1
2010-06-07	1
2010-06-08	2
2010-06-09	1
2010-06-11	1
2010-06-12	1
2010-06-15	9
2010-06-16	2
2010-06-17	1
2010-06-19	2
2010-06-20	1
2010-06-21	8
2010-06-22	4
2010-06-23	7
2010-06-24	3
2010-06-25	1
2010-06-26	2
2010-06-27	6
2010-06-28	5
2010-06-29	4
2010-06-30	12
2010-07-01	2
2010-07-04	3
2010-07-05	2
2010-07-06	5
2010-07-07	1
2010-07-08	1
2010-07-10	3
2010-07-11	3
2010-07-12	3
2010-07-13	3
2010-07-14	7
2010-07-15	2
2010-07-17	5
2010-07-18	1
2010-07-19	5
2010-07-20	6
2010-07-21	5
2010-07-24	4
2010-07-25	3
2010-07-26	5
2010-07-27	1
2010-07-28	4
2010-07-29	2
2010-07-31	6
2010-08-01	4
2010-08-02	3

2010-08-03	1
2010-08-04	4
2010-08-05	1
2010-08-06	2
2010-08-07	8
2010-08-08	3
2010-08-09	8
2010-08-10	6
2010-08-11	7
2010-08-12	2
2010-08-13	3
2010-08-14	6
2010-08-15	4
2010-08-16	3
2010-08-17	5
2010-08-18	8
2010-08-19	4
2010-08-21	3
2010-08-22	2
2010-08-23	5
2010-08-24	5
2010-08-25	6
2010-08-26	1
2010-08-27	1
2010-08-28	5
2010-08-29	7
2010-08-30	5
2010-08-31	3
2010-09-01	3
2010-09-02	3
2010-09-03	1
2010-09-04	1
2010-09-05	6
2010-09-06	5
2010-09-07	5
2010-09-08	3
2010-09-09	1
2010-09-11	4
2010-09-12	6
2010-09-13	5
2010-09-14	3
2010-09-15	6
2010-09-18	6
2010-09-19	5
2010-09-20	10
2010-09-21	5
2010-09-22	2
2010-09-23	4
2010-09-24	1
2010-09-25	3
2010-09-26	3
2010-09-27	7
2010-09-28	12
2010-09-29	4
2010-09-30	1
2010-10-01	1
2010-10-02	2

2010-10-04	2
2010-10-05	2
2010-10-06	1
2010-10-07	8
2010-10-11	1
2010-10-12	1
2010-10-13	3
2010-10-14	5
2010-10-16	1
2010-10-18	1
2010-10-19	2
2010-10-20	2
2010-10-21	2
2010-10-22	1
2010-10-23	3
2010-10-24	1
2010-10-25	1
2010-10-26	2
2010-10-27	3
2010-10-28	4
2010-10-29	2
2010-10-30	1
2010-11-01	2
2010-11-02	5
2010-11-03	1
2010-11-04	5
2010-11-06	2
2010-11-08	3
2010-11-09	2
2010-11-10	3
2010-11-12	2
2010-11-13	1
2010-11-14	1
2010-11-16	8
2010-11-17	4
2010-11-18	4
2010-11-19	4
2010-11-22	3
2010-11-23	3
2010-11-24	1
2010-11-25	17
2010-11-26	15
2010-11-28	1
2010-11-29	1
2010-11-30	2
2010-12-02	2
2010-12-03	8
2010-12-05	1
2010-12-06	7
2010-12-07	3
2010-12-08	3
2010-12-09	3
2010-12-10	1
2010-12-11	2
2010-12-12	4
2010-12-13	9
2010-12-14	6

```
2010-12-15     3
2010-12-16    12
2010-12-17     2
2010-12-19     4
2010-12-20     2
2010-12-21     8
2010-12-22     9
2010-12-23     3
2010-12-25     1
2010-12-26     1
2010-12-27     1
2010-12-28     1
2010-12-29     2
2010-12-30     5
Name: Merchnum, dtype: int64
```

In [ ]:

```
# Locate additional information regarding bursts of activity for the presentation
merchnum_slide2 = df_orig_scored.loc[(df_orig_scored['Date'] >= '2010-11-25') & (df_orig_s
cored['Date'] <= '2010-11-27') & (df_orig_scored['Merchnum']=='4353000719908')]
merchnum_slide2.rename(columns={'Predict Proba':'Fraud Score'}, inplace=True)
merchnum_slide2[['Date','Merch description','Fraud','Predicted','Fraud Score']]
```

Out[ ]:

	Date	Merch description	Fraud	Predicted	Fraud Score
89364	2010-11-25	AMAZON.COM *SUPERSTOR	1	0.0	0.001508
89365	2010-11-25	ACI*AMAZON.COM INC	1	0.0	0.007318
89366	2010-11-25	AMAZON.COM *SUPERSTOR	1	0.0	0.007318
89368	2010-11-25	AMAZON.COM *SUPERSTOR	1	0.0	0.025684
89374	2010-11-25	AMAZON.COM *SUPERSTOR	1	0.0	0.051398
89377	2010-11-25	AMAZON.COM *SUPERSTOR	1	0.0	0.232366
89380	2010-11-25	ACI*AMAZON.COM INC	1	0.0	0.372040
89383	2010-11-25	ACI*AMAZON.COM INC	1	0.0	0.381526
89391	2010-11-25	AMAZON.COM *SUPERSTOR	1	0.0	0.418836
89392	2010-11-25	AMAZON.COM *SUPERSTOR	1	1.0	0.521906
89398	2010-11-25	AMAZON.COM *SUPERSTOR	1	1.0	0.660092
89400	2010-11-25	AMAZON.COM *SUPERSTOR	1	1.0	0.839343
89403	2010-11-25	AMAZON.COM *SUPERSTOR	1	1.0	0.891692
89411	2010-11-25	AMAZON.COM *SUPERSTOR	1	1.0	0.953155
89417	2010-11-25	AMAZON.COM *SUPERSTOR	1	1.0	0.989346
89418	2010-11-25	AMAZON.COM *SUPERSTRE	1	1.0	0.989346
89420	2010-11-25	AMAZON.COM *SUPERSTOR	1	1.0	0.984346
89425	2010-11-26	AMAZON.COM *SUPERSTOR	1	1.0	0.925045
89426	2010-11-26	AMAZON.COM *SUPERSTOR	1	1.0	0.920045
89434	2010-11-26	ACI*AMAZON.COM INC	1	1.0	0.963083
89456	2010-11-26	ACI*AMAZON.COM INC	1	1.0	0.968083
89458	2010-11-26	AMAZON.COM *SUPERSTOR	1	1.0	1.000000
89461	2010-11-26	AMAZON.COM *SUPERSTOR	1	1.0	1.000000
89464	2010-11-26	AMAZON.COM *SUPERSTOR	1	1.0	0.990000
89465	2010-11-26	AMAZON.COM *SUPERSTOR	1	1.0	0.990000
89472	2010-11-26	AMAZON.COM *SUPERSTOR	1	1.0	0.975000
89473	2010-11-26	AMAZON.COM *SUPERSTRE	1	1.0	0.975000
89474	2010-11-26	AMAZON.COM *SUPERSTOR	1	1.0	0.533291
89478	2010-11-26	AMAZON.COM *SUPERSTRE	1	1.0	0.538291
89519	2010-11-26	AMAZON.COM *SUPERSTOR	1	1.0	0.533291
89528	2010-11-26	AMAZON.COM *SUPERSTOR	1	1.0	0.567778
89531	2010-11-26	AMAZON.COM *SUPERSTOR	1	1.0	0.557778