Application (Identity) Fraud Analysis

In [1]:

```
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 sys

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

In [2]:

```
import psutil

from IPython.display import display

from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import StratifiedKFold
from sklearn.feature_selection import RFECV

%matplotlib inline
```

Function Definition Area

Optimize Data Fields

This function is used to reduce the size of the dataframes to enable more efficient running of the code. When using this function, it may take a while for a dataframe to be processed.

In [3]:

```
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)))
```

Load the dataset

In [4]:

```
file_path = '/home/thanos/Documents/8th_semester/DSO_562_Fraud_analytics/Project_2_DSO_56
2/HW5/'
df = pd.read_csv(file_path+'applications data.csv', parse_dates = ["date"],nrows=2000) # r
ead the dataset
```

Data Imputation

After inspection of the dataset, we replace the frivolous values with the record number.

In [5]:

```
# Replace all frivolous values with the record number.
start = pd.datetime.now()
df['ssn'] = np.where(df['ssn'] == 999999999, df['record'], df['ssn'])
df['address'] = np.where(df['address'] == "123 MAIN ST", df['record'], df['address'])
df['dob'] = np.where(df['dob'] == 19070626, df['record'], df['dob'])
df['homephone'] = np.where(df['homephone'] == 9999999999, df['record'], df['homephone'])
print(pd.datetime.now() - start)
```

```
0:00:00.003341
```

In [6]:

```
df = optimize(df)
```

Variable Creation

In this part we perform feature engineering and we create new features based on the ones available in the dataset.

Categorical Candidate Variables

In [7]:

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

# 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_label'])] # Don't convert the date or f
raud label
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 0:00:00.012435
```

Out[7]:

```
record                  object
date            datetime64[ns]
ssn                     object
firstname               object
lastname                object
address                 object
zip5                    object
dob                     object
homephone               object
fraud_label               int8
dtype: object
```

In [8]:

```
# Make combinations with the name
df_var['fullname'] = df_var['firstname'] + df_var['lastname']
df_var['fullname-dob'] = df_var['fullname'] + df_var['dob']
df_var['fullname-ssn'] = df_var['fullname'] + df_var['ssn']
df_var['fullname-homephone'] = df_var['fullname'] + df_var['homephone']
df_var['fullname-address'] = df_var['fullname'] + df_var['address']
df_var['fullname-address-zip'] = df_var['fullname'] + df_var['address'] + df_var['zip5']
df_var['fullname-dob-homephone'] = df_var['fullname'] + df_var['dob'] + df_var['homephone'
]
df_var['fullname-dob-zip'] = df_var['fullname'] + df_var['dob'] + df_var['zip5']
df_var['fullname-zip'] = df_var['fullname'] + df_var['zip5']
df_var['firstname-dob'] = df_var['firstname'] + df_var['dob']
df_var['lastname-dob'] = df_var['lastname'] + df_var['dob']
df_var['firstname-homephone'] = df_var['firstname'] + df_var['homephone']
df_var['lastname-homephone'] = df_var['lastname'] + df_var['homephone']
```

In [9]:

```
# Make combinations with the ssn
df_var['ssn-firstname'] = df_var['ssn'] + df_var['firstname']
df_var['ssn-lastname'] = df_var['ssn'] + df_var['lastname']
df_var['ssn-zip'] = df_var['ssn'] + df_var['zip5']
df_var['ssn-dob'] = df_var['ssn'] + df_var['dob']
df_var['ssn-homephone'] = df_var['ssn'] + df_var['homephone']
df_var['ssn-address'] = df_var['ssn'] + df_var['address']
df_var['ssn-address-zip'] = df_var['ssn'] + df_var['address'] + df_var['zip5']
df_var['ssn-fullname-dob'] = df_var['ssn'] + df_var['fullname'] + df_var['dob']
```

In [10]:

```
# Make combinations of other data fields
df_var['address-zip'] = df_var['address'] + df_var['zip5']
df_var['address-zip-fullname-dob'] = df_var['address'] + df_var['zip5'] + df_var['fullnam
e'] + df_var['dob']
df_var['address-zip-homephone'] = df_var['address'] + df_var['zip5'] + df_var['homephone']
df_var['zip-homephone'] = df_var['zip5'] + df_var['homephone']
df_var['zip-dob'] = df_var['zip5'] + df_var['dob']
df_var['homephone-dob'] = df_var['homephone'] + df_var['dob']
```

Convert Data Types

Convert strings back to integers where possible.

In [11]:

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

cols_int = ['record','ssn','zip5','dob','homephone','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('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: ssn-homephone
convert time 0:00:00.007776
```

Out[11]:

```
record                                int64
date                         datetime64[ns]
ssn                                   int64
firstname                            object
lastname                             object
address                              object
zip5                                  int64
dob                                   int64
homephone                             int64
fraud_label                            int8
fullname                             object
fullname-dob                         object
fullname-ssn                         object
fullname-homephone                   object
fullname-address                     object
fullname-address-zip                 object
fullname-dob-homephone               object
fullname-dob-zip                     object
fullname-zip                         object
firstname-dob                        object
lastname-dob                         object
firstname-homephone                  object
lastname-homephone                   object
ssn-firstname                        object
ssn-lastname                         object
ssn-zip                               int64
ssn-dob                               int64
ssn-homephone                        object
ssn-address                          object
ssn-address-zip                      object
ssn-fullname-dob                     object
address-zip                          object
address-zip-fullname-dob             object
address-zip-homephone                object
zip-homephone                         int64
zip-dob                               int64
homephone-dob                         int64
dtype: object
```

In [12]:

```
df_var['ssn-homephone'] = df_var['ssn-homephone'].astype('float32')
```

Numerical Candidate Variables

Create Columns for the Necessary Time Periods

This makes new columns for the various time periods.

In [13]:

```
# Make a list of variable combinations to iterate through and create time-related variable
s
cols_drop = ['record','date','firstname','lastname','zip5','fraud_label']
var_combos = df_var.drop(cols_drop,axis=1).columns

# Create column names
time_list = [0,1,3,7,14,30,90,180]
time_joined =['join_ts1']
for num in time_list:
    time_joined.append('join_ts2_'+str(num))

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()
df_var2['join_ts1'] = df_var2['date']
for time in time_list:
    temp_endTime = 'join_ts2_' + str(time)
    df_var2[temp_endTime] = df_var2['date'] + dt.timedelta(time)
print('first loop', pd.datetime.now()-start_loop)
```

```
copy time 0:00:00.001031
first loop 0:00:00.010868
```

Velocity Candidate Variables

This makes the velocity variables (features). It's counting the number of applications (records) it sees based on the time period. For instance, it counts all the applications for an SSN it sees over the last 3 days based on the date and record number. It only counts the current record and those records in the past for those last 3 days.

In [14]:

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

for item in var_combos:
    df_var3 = df_var1[['record','date',item]]
    temp_list = time_joined + [item]
    df_var4 = df_var2[temp_list + ['record']].copy()
    df_var4.rename(columns={'record':'record2'},inplace=True)

    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[temp_endTime]) & (df_temp['record2'
] <= df_temp['record'])]

        temp_groupby = df2_temp[['record','date']].groupby('record')

        temp_name = item + '_' + 'velocity' + 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 0:00:02.472726
284
```

Out[14]:

	date	ssn	firstname	lastname	address	zip5	dob	homephone	fra
record									
1	2016-01-01	379070012	XRRAMMTR	SMJETJMJ	6861 EUTST PL	2765	1	1797504115	
2	2016-01-01	387482503	MAMSTUJR	RTTEMRRR	7280 URASA PL	57169	19340615	4164239415	
3	2016-01-01	200332444	SZMMUJEZS	EUSEZRAE	5581 RSREX LN	56721	3	216537580	
4	2016-01-01	747451317	SJJZSXRSZ	ETJXTXXS	1387 UJZXJ RD	35286	19440430	132144161	
5	2016-01-01	24065868	SSSXUEJMS	SSUUJXUZ	279 EAASA WY	3173	19980315	6101082272	

In [15]:

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

```
Memory percentage used 25.6
```

Relativey Velocity Candidate Variables

This cell is making the relative velocity variables (features). Velocity variables denote the number of applications with that group seen in the recent past divided by the number of applications with that same group seen in the past 1, 3, 7, 14, 30, etc. days / num days

In [16]:

```
# Save the results from the previous loop before sending the df_final dataframe through th
e 3rd loop
df_final_loop2 = deepcopy(df_final)
```

In [17]:

```
start_loop3=pd.datetime.now()
groupbyvar_denom = deepcopy(var_combos)
days_numer = ['0','1']
days_denom = ['3','7','14','30','90','180']

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

```
third loop 0:00:00.362101
```

In [18]:

```
df_final
```

Out[18]:

	date	ssn	firstname	lastname	address	zip5	dob	homephone	fra
record									
1	2016-01-01	379070012	XRRAMMTR	SMJETJMJ	6861 EUTST PL	2765	1	1797504115	
2	2016-01-01	387482503	MAMSTUJR	RTTEMRRR	7280 URASA PL	57169	19340615	4164239415	
3	2016-01-01	200332444	SZMMUJEZS	EUSEZRAE	5581 RSREX LN	56721	3	216537580	
4	2016-01-01	747451317	SJJZSXRSZ	ETJXTXXS	1387 UJZXJ RD	35286	19440430	132144161	
5	2016-01-01	24065868	SSSXUEJMS	SSUUJXUZ	279 EAASA WY	3173	19980315	6101082272	
...	...	...	...	...	...	...	...	...	
1996	2016-01-01	678419447	RAEZAZMM	UURSTRRE	240 EMTX AVE	19335	19460925	7917597273	
1997	2016-01-01	374898285	UTTXTJTEZ	UXEXUUEX	6224 UMAJJ ST	96509	19880628	1164067356	
1998	2016-01-01	339884520	EMAJUUJMX	UTJZMJES	426 RXEEJ DR	31469	19340904	1998	
1999	2016-01-01	872433283	UAMURZJEM	ZMSJAMT	1034 UTJM AVE	86555	19100823	6491219288	
2000	2016-01-01	343941790	XESSAEZMS	SXESSMMR	7098 UAURM ST	93840	19460214	6959543525	

2000 rows × 656 columns

In [19]:

```
df_final = optimize(df_final)
# df_final = df_final.sort_index(inplace=True)
# sort index
df_final =  df_final.reset_index()
```

In [20]:

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

```
Memory percentage used 25.7
```

In [21]:

```
# uncomment to see the list of variables in memory
# dir()
```

In [22]:

```
del df2_temp, df_final_loop2, df_temp, df_var1, df_var2, df_var3, df_var4
```

In [23]:

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

```
Memory percentage used 25.7
```

"Days Since" Candidate Variables

Find the number of days since we last saw each variable (feature).

In [24]:

```
# Prof's notebook code for the Days-Since variables
# function to find days since variables
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 [25]:

```
time_ds_all=pd.datetime.now()
# Iterate through all the variables and send them through the "ds" function to find the "d
ays since" values
ds_dict={}
for item in var_combos:
    curr_time=pd.datetime.now()

    # Determine the days-since variable to calculate and its name
    groupby_cols2 = item.split('-')
    groupby_cols1 = groupby_cols2 + ['date']
    curr_name = item + '_daysSince'

    # Calculate the days-since variable (ds) and assign it to a global variable (curr_nam
e)
    try:
        vars()[curr_name] = ds(df_var, groupby_cols1, groupby_cols2, curr_name)
        ds_dict[curr_name] = vars()[curr_name] # Save results to a dictionary
    except KeyError:
        zip_index1 = groupby_cols1.index('zip')
        groupby_cols1[zip_index1] = 'zip5'
        zip_index2 = groupby_cols2.index('zip')
        groupby_cols2[zip_index2] = 'zip5'

        vars()[curr_name] = ds(df_var, groupby_cols1, groupby_cols2, curr_name)
        ds_dict[curr_name] = vars()[curr_name] # Save results to a dictionary
    except:
        print("ERROR INFO ->", sys.exc_info()[0],sys.exc_info()[1])

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

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

```
Done with: ssn ; Time: 0:00:01.547825
Done with: address ; Time: 0:00:01.421248
Done with: dob ; Time: 0:00:01.499184
Done with: homephone ; Time: 0:00:01.449560
Done with: fullname ; Time: 0:00:01.415906
Done with: fullname-dob ; Time: 0:00:01.751118
Done with: fullname-ssn ; Time: 0:00:01.754956
Done with: fullname-homephone ; Time: 0:00:01.735272
Done with: fullname-address ; Time: 0:00:01.618336
Done with: fullname-address-zip ; Time: 0:00:02.073212
Done with: fullname-dob-homephone ; Time: 0:00:02.069464
Done with: fullname-dob-zip ; Time: 0:00:02.145195
Done with: fullname-zip ; Time: 0:00:01.810480
Done with: firstname-dob ; Time: 0:00:01.750920
Done with: lastname-dob ; Time: 0:00:01.750946
Done with: firstname-homephone ; Time: 0:00:01.741356
Done with: lastname-homephone ; Time: 0:00:01.749730
Done with: ssn-firstname ; Time: 0:00:01.730493
Done with: ssn-lastname ; Time: 0:00:01.737218
Done with: ssn-zip ; Time: 0:00:01.961605
Done with: ssn-dob ; Time: 0:00:01.868683
Done with: ssn-homephone ; Time: 0:00:01.888819
Done with: ssn-address ; Time: 0:00:01.739644
Done with: ssn-address-zip ; Time: 0:00:02.172966
Done with: ssn-fullname-dob ; Time: 0:00:02.147046
Done with: address-zip ; Time: 0:00:01.868856
Done with: address-zip-fullname-dob ; Time: 0:00:02.411252
Done with: address-zip-homephone ; Time: 0:00:02.160744
Done with: zip-homephone ; Time: 0:00:01.938475
Done with: zip-dob ; Time: 0:00:01.947039
Done with: homephone-dob ; Time: 0:00:01.863946
DONE WITH ALL! 0:00:56.727412
```

In [26]:

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

# Merge the days-since variables with the main dataset
for item in ds_dict.keys():
    col_variable = item.split('_')[0]
    print('Merging on',col_variable)
    try:
        df_final = pd.merge(df_final, ds_dict[item], how='left', left_on=[col_variable,'da
te'], right_on=[col_variable,'date'])

    except KeyError:
        right_col_variable = col_variable.split('-')

        if 'zip' in right_col_variable:
            zip_index = right_col_variable.index('zip')
            right_col_variable[zip_index] = 'zip5'

        temp = ds_dict[item].copy()
        temp[col_variable]=temp[right_col_variable].astype(str).sum(axis=1)
        temp.drop(columns=right_col_variable, inplace=True)
        temp = optimize(temp)
        df_final = pd.merge(df_final, temp, how='left', left_on=[col_variable,'date'], rig
ht_on=[col_variable,'date'])

    except:
        print("ERROR INFO ->", sys.exc_info()[0],sys.exc_info()[1])

print("DONE WITH ALL!", pd.datetime.now()-time_ds_merge)
```

```
Merging on ssn
Merging on address
Merging on dob
Merging on homephone
Merging on fullname
Merging on fullname-dob
Merging on fullname-ssn
Merging on fullname-homephone
Merging on fullname-address
Merging on fullname-address-zip
Merging on fullname-dob-homephone
Merging on fullname-dob-zip
Merging on fullname-zip
Merging on firstname-dob
Merging on lastname-dob
Merging on firstname-homephone
Merging on lastname-homephone
Merging on ssn-firstname
Merging on ssn-lastname
Merging on ssn-zip
Merging on ssn-dob
Merging on ssn-homephone
Merging on ssn-address
Merging on ssn-address-zip
Merging on ssn-fullname-dob
Merging on address-zip
Merging on address-zip-fullname-dob
Merging on address-zip-homephone
Merging on zip-homephone
Merging on zip-dob
Merging on homephone-dob
DONE WITH ALL! 0:00:01.713871
```

In [27]:

```
df_final
```

Out[27]:

	record	date	ssn	firstname	lastname	address	zip5	dob	homepho
0	1	2016-01-01	379070012	XRRAMMTR	SMJETJMJ	6861 EUTST PL	2765	1	17975041
1	2	2016-01-01	387482503	MAMSTUJR	RTTEMRRR	7280 URASA PL	57169	19340615	41642394
2	3	2016-01-01	200332444	SZMMUJEZS	EUSEZRAE	5581 RSREX LN	56721	3	2165375
3	4	2016-01-01	747451317	SJJZSXRSZ	ETJXTXXS	1387 UJZXJ RD	35286	19440430	1321441
4	5	2016-01-01	24065868	SSSXUEJMS	SSUUJXUZ	279 EAASA WY	3173	19980315	61010822
...	...	...	...	...	...	...	...	...	
1995	1996	2016-01-01	678419447	RAEZAZMM	UURSTRRE	240 EMTX AVE	19335	19460925	79175972
1996	1997	2016-01-01	374898285	UTTXTJTEZ	UXEXUUEX	6224 UMAJJ ST	96509	19880628	11640673
1997	1998	2016-01-01	339884520	EMAJUUJMX	UTJZMJES	426 RXEEJ DR	31469	19340904	19
1998	1999	2016-01-01	872433283	UAMURZJEM	ZMSJAMT	1034 UTJM AVE	86555	19100823	64912192
1999	2000	2016-01-01	343941790	XESSAEZMS	SXESSMMR	7098 UAURM ST	93840	19460214	69595435

2000 rows × 688 columns

In [30]:

```
df_final[['record','date','zip5','homephone','zip-homephone','zip-homephone_daysSince']]
```

Out[30]:

	record	date	zip5	homephone	zip-homephone	zip-homephone_daysSince
0	1	2016-01-01	2765	1797504115	27651797504115	NaN
1	2	2016-01-01	57169	4164239415	571694164239415	NaN
2	3	2016-01-01	56721	216537580	56721216537580	NaN
3	4	2016-01-01	35286	132144161	35286132144161	NaN
4	5	2016-01-01	3173	6101082272	31736101082272	NaN
...	...	...	...	...	...	...
1995	1996	2016-01-01	19335	7917597273	193357917597273	NaN
1996	1997	2016-01-01	96509	1164067356	965091164067356	NaN
1997	1998	2016-01-01	31469	1998	314691998	NaN
1998	1999	2016-01-01	86555	6491219288	865556491219288	NaN
1999	2000	2016-01-01	93840	6959543525	938406959543525	NaN

2000 rows × 6 columns

In [31]:

```
df_final['ssn-homephone_daysSince']
```

Out[31]:

```
0       0.0
1       0.0
2       0.0
3       0.0
4       0.0
       ...
1995    0.0
1996    0.0
1997    0.0
1998    0.0
1999    0.0
Name: ssn-homephone_daysSince, Length: 2000, dtype: float32
```

In [48]:

```
# find features with NaN values
# df_final.isnull().any().values
df_final.iloc[:,df_final.isnull().any().values]
```

Out[48]:

	ssn-zip_daysSince	ssn-dob_daysSince	zip-homephone_daysSince	zip-dob_daysSince	homephone-dob_daysSince
0	NaN	NaN	NaN	0.0	NaN
1	NaN	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	0.0	NaN
3	NaN	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...
1995	NaN	NaN	NaN	NaN	NaN
1996	NaN	NaN	NaN	NaN	NaN
1997	NaN	NaN	NaN	NaN	NaN
1998	NaN	NaN	NaN	NaN	NaN
1999	NaN	NaN	NaN	NaN	NaN

2000 rows × 5 columns

In [56]:

```
df_final[['ssn','zip5','dob','homephone','ssn-zip','ssn-dob','zip-homephone','zip-dob','ho
mephone-dob']].dtypes
```

Out[56]:

```
ssn              int32
zip5             int32
dob              int32
homephone        int64
ssn-zip          int64
ssn-dob          int64
zip-homephone    int64
zip-dob          int64
homephone-dob    int64
dtype: object
```

Variable Selection

Scale all variables (standard scale with 0 mean and 1 std)

In [28]:

```
# Get only the numerical variables
numerical_index = df_final.columns.get_loc("homephone-dob")+1 #this is the index of the fi
rst numerical feature/column in the dataframe
df_numerical = pd.concat([df_final['fraud_label'],df_final.iloc[:,numerical_index:]], axis
=1)
```

In [29]:

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

Out[29]:

	fraud_label	rand_num	ssn_velocity0_date	ssn_velocity1_date	ssn_velocity3_date	ssn_veloci
0	0	3601444	1	1	1	
1	1	2574455	1	1	1	
2	0	596049	1	1	1	
3	0	125965	1	1	1	
4	0	1267693	1	1	1	

5 rows × 653 columns

In [30]:

```
# Flip the "Days Since" variables (max of col - current value)
days_since_cols = df_numerical.columns[-14:]

for col in days_since_cols:
    max_val = df_numerical[col].max()
    df_numerical[col] = max_val - df_numerical[col]
```

In [31]:

```
df_numerical.head()
```

Out[31]:

	fraud_label	rand_num	ssn_velocity0_date	ssn_velocity1_date	ssn_velocity3_date	ssn_veloci
0	0	3601444	1	1	1	
1	1	2574455	1	1	1	
2	0	596049	1	1	1	
3	0	125965	1	1	1	
4	0	1267693	1	1	1	

5 rows × 653 columns

In [ ]:

```
# z-scale the numerical variables
df_numerical_zscale = df_numerical.apply(zscore)
df_numerical_zscale['fraud_label'] = df_numerical['fraud_label']
df_numerical_zscale.head(100)
```

Out[ ]:

	fraud_label	rand_num	ssn_velocity0_date	ssn_velocity1_date	ssn_velocity3_date	ssn_veloc
0	0	0.203390	-0.032706	-0.039149	-0.047591	
1	1	-1.425404	-0.032706	-0.039149	-0.047591	
2	0	0.192602	-0.032706	-0.039149	-0.047591	
3	0	-1.576339	-0.032706	-0.039149	-0.047591	
4	0	-1.657700	-0.032706	-0.039149	-0.047591	
5	0	-0.042599	-0.032706	-0.039149	-0.047591	
6	0	1.410810	-0.032706	-0.039149	-0.047591	
7	0	0.519229	-0.032706	-0.039149	-0.047591	
8	0	-1.579988	-0.032706	-0.039149	-0.047591	
9	0	-0.819175	-0.032706	-0.039149	-0.047591	
10	0	-0.773911	-0.032706	-0.039149	-0.047591	
11	0	-1.356085	-0.032706	-0.039149	-0.047591	
12	0	0.089777	-0.032706	-0.039149	-0.047591	
13	0	-0.039046	-0.032706	-0.039149	-0.047591	
14	0	-1.713307	-0.032706	-0.039149	-0.047591	
15	0	-1.065900	-0.032706	-0.039149	-0.047591	
16	0	1.665199	-0.032706	-0.039149	-0.047591	
17	0	-1.480517	-0.032706	-0.039149	-0.047591	
18	0	0.975559	-0.032706	-0.039149	-0.047591	
19	0	1.065310	-0.032706	-0.039149	-0.047591	
20	0	-1.454363	-0.032706	-0.039149	-0.047591	
21	0	-0.393378	-0.032706	-0.039149	-0.047591	
22	0	1.631469	-0.032706	-0.039149	-0.047591	
23	0	-0.401829	-0.032706	-0.039149	-0.047591	
24	0	0.330491	-0.032706	-0.039149	-0.047591	
25	0	1.312440	-0.032706	-0.039149	-0.047591	
26	0	0.622796	-0.032706	-0.039149	-0.047591	
27	0	-0.187754	-0.032706	-0.039149	-0.047591	
28	0	-0.634997	-0.032706	-0.039149	-0.047591	
29	0	0.200514	-0.032706	-0.039149	-0.047591	
30	0	-0.005869	-0.032706	-0.039149	-0.047591	
31	0	0.702458	-0.032706	-0.039149	-0.047591	
32	0	0.274308	-0.032706	-0.039149	-0.047591	

	fraud_label	rand_num	ssn_velocity0_date	ssn_velocity1_date	ssn_velocity3_date	ssn_veloc
33	0	0.495567	-0.032706	-0.039149	-0.047591	
34	0	-0.619073	-0.032706	-0.039149	-0.047591	
35	0	0.213990	-0.032706	-0.039149	-0.047591	
36	0	1.659729	-0.032706	-0.039149	-0.047591	
37	0	-0.304101	-0.032706	-0.039149	-0.047591	
38	0	0.994008	-0.032706	-0.039149	-0.047591	
39	0	1.263624	-0.032706	-0.039149	-0.047591	
40	0	0.177718	-0.032706	-0.039149	-0.047591	
41	0	1.144215	-0.032706	-0.039149	-0.047591	
42	0	0.682046	-0.032706	-0.039149	-0.047591	
43	0	0.305642	-0.032706	-0.039149	-0.047591	
44	0	0.781019	-0.032706	-0.039149	-0.047591	
45	0	0.480613	-0.032706	-0.039149	-0.047591	
46	0	-0.675676	-0.032706	-0.039149	-0.047591	
47	0	1.518630	-0.032706	-0.039149	-0.047591	
48	0	-1.070624	-0.032706	-0.039149	-0.047591	
49	0	-0.519082	-0.032706	-0.039149	-0.047591	
50	0	-1.359477	-0.032706	-0.039149	-0.047591	
51	0	-1.632762	-0.032706	-0.039149	-0.047591	
52	0	1.111708	-0.032706	-0.039149	-0.047591	
53	0	0.481180	-0.032706	-0.039149	-0.047591	
54	0	0.596187	-0.032706	-0.039149	-0.047591	
55	0	0.924722	-0.032706	-0.039149	-0.047591	
56	0	0.600147	-0.032706	-0.039149	-0.047591	
57	0	-1.141631	-0.032706	-0.039149	-0.047591	
58	0	-0.189685	-0.032706	-0.039149	-0.047591	
59	0	1.436045	-0.032706	-0.039149	-0.047591	
60	0	0.864517	-0.032706	-0.039149	-0.047591	
61	0	1.019769	-0.032706	-0.039149	-0.047591	
62	0	1.079635	-0.032706	-0.039149	-0.047591	
63	0	-1.560302	-0.032706	-0.039149	-0.047591	
64	0	0.219744	-0.032706	-0.039149	-0.047591	
65	0	-0.254103	-0.032706	-0.039149	-0.047591	
66	0	-0.183079	-0.032706	-0.039149	-0.047591	
67	0	0.813074	-0.032706	-0.039149	-0.047591	

	fraud_label	rand_num	ssn_velocity0_date	ssn_velocity1_date	ssn_velocity3_date	ssn_veloc
68	0	1.537234	-0.032706	-0.039149	-0.047591	
69	0	-1.254434	-0.032706	-0.039149	-0.047591	
70	0	-0.780449	-0.032706	-0.039149	-0.047591	
71	0	0.061366	-0.032706	-0.039149	-0.047591	
72	0	1.382017	-0.032706	-0.039149	-0.047591	
73	0	-0.918889	-0.032706	-0.039149	-0.047591	
74	0	1.530757	-0.032706	-0.039149	-0.047591	
75	0	1.296879	-0.032706	-0.039149	-0.047591	
76	0	-1.582238	-0.032706	-0.039149	-0.047591	
77	0	0.752823	-0.032706	-0.039149	-0.047591	
78	0	-0.219217	-0.032706	-0.039149	-0.047591	
79	0	-0.987377	-0.032706	-0.039149	-0.047591	
80	0	1.330271	-0.032706	-0.039149	-0.047591	
81	0	-1.594929	-0.032706	-0.039149	-0.047591	
82	0	0.001598	-0.032706	-0.039149	-0.047591	
83	0	-0.476958	-0.032706	-0.039149	-0.047591	
84	0	0.889126	-0.032706	-0.039149	-0.047591	
85	0	0.523688	-0.032706	-0.039149	-0.047591	
86	0	0.009716	-0.032706	-0.039149	-0.047591	
87	0	-0.546300	-0.032706	-0.039149	-0.047591	
88	0	0.477244	-0.032706	-0.039149	-0.047591	
89	0	-0.215320	-0.032706	-0.039149	-0.047591	
90	0	-0.187401	-0.032706	-0.039149	-0.047591	
91	0	0.198610	-0.032706	-0.039149	-0.047591	
92	0	-0.642244	-0.032706	-0.039149	-0.047591	
93	0	-0.854496	-0.032706	-0.039149	-0.047591	
94	0	0.030140	-0.032706	-0.039149	-0.047591	
95	0	1.127531	-0.032706	-0.039149	-0.047591	
96	0	-0.430925	-0.032706	-0.039149	-0.047591	
97	0	1.380128	-0.032706	-0.039149	-0.047591	
98	0	1.523020	-0.032706	-0.039149	-0.047591	
99	0	-1.544420	-0.032706	-0.039149	-0.047591	

100 rows × 636 columns

In [ ]:

```
# verify there are no NaN values
df_numerical_zscale[df_numerical_zscale.isna().any(axis=1)]
```

Out[ ]:

fraud_label	rand_num	ssn_velocity0_date	ssn_velocity1_date	ssn_velocity3_date	ssn_velocity

0 rows × 636 columns

In [ ]:

```
df_numerical_zscale = optimize(df_numerical_zscale)
df_numerical_zscale.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000000 entries, 0 to 999999
Columns: 636 entries, fraud_label to lastnamessn_diff
dtypes: float32(635), int8(1)
memory usage: 2.4 GB
```

In [ ]:

```
# df_numerical_zscale.to_csv('df_numerical_zscale.csv')
```

Separate data into modeling/Out of Time (OOT)

OOT data is the validation set and includes all aplications that happened after Nov. 1st 2016. Training/test set contains all applications performed until Oct. 31st 2016.

In [ ]:

```
df_oot = df_numerical_zscale.loc[(df_final['date'] >= '2016-11-01')]
df_modeling_scaled = df_numerical_zscale.loc[((df_final['date'] < '2016-11-01') & (df_fina
l['date'] > '2016-01-14'))]
```

In [ ]:

```
df_modeling_scaled.head()
```

Out[ ]:

	record	date	ssn	firstname	lastname	address	zip5	dob	homepho
38511	38512	2016-01-15	476774243	RASTAZMM	EEJTAXEZ	1420 SJXAM WY	58008	19570328	73550865
38512	38513	2016-01-15	432844033	ESXAURS	SMJZXZMZ	2314 SZSRJ AVE	60458	19850530	8377884
38513	38514	2016-01-15	185477074	XMAEAEXSX	RTZRTZAS	9310 RMZTT AVE	65654	19180130	91300929
38514	38515	2016-01-15	933119335	UUTXTTUAE	EMUTUJS	6950 XJERT AVE	56324	38515	385
38515	38516	2016-01-15	845202954	RTMMTSZRZ	SZJMSMUJ	4007 RTERR CT	50477	19271109	385

5 rows × 671 columns

Filter & Wrapper Methods

In [ ]:

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

```
Memory percentage used 76.9
```

Feature selection (filter method)

In [ ]:

```
data_Jan14 = deepcopy(df_modeling_scaled)
```

In [ ]:

```
# find good and bad indices
good_ind = np.where(data_Jan14['fraud_label'] == 0)
bad_ind = np.where(data_Jan14['fraud_label'] == 1)
```

In [ ]:

```
inliers_class = data_Jan14.iloc[good_ind]
outlier_class = data_Jan14.iloc[bad_ind]
```

In [ ]:

```
no_of_total_frauds = data_Jan14['fraud_label'].sum()
no_of_total_frauds
```

Out[ ]:

```
11486
```

In [ ]:

```
data_Jan14.iloc[good_ind].tail()
```

Out[ ]:

	record	ssn	zip5	dob	homephone	fraud_label	ssn-zip	ssn-dob	r
833502	833503	0.419383	-1.575062	0.466513	-0.273241	0	-1.237361	0.575598	
833503	833504	-1.604572	0.639394	0.375156	0.836262	0	-1.369516	-1.292136	
833504	833505	0.385874	0.095196	0.362550	-1.489848	0	0.517010	0.544676	
833505	833506	-1.088515	0.277483	0.334230	1.678888	0	-0.880402	-0.815911	
833506	833507	1.684368	-1.406054	0.328013	0.260979	0	-1.117467	1.742945	

5 rows × 647 columns

In [ ]:

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

```
Memory percentage used 84.8
```

Find FDR and KS statistic for each feature

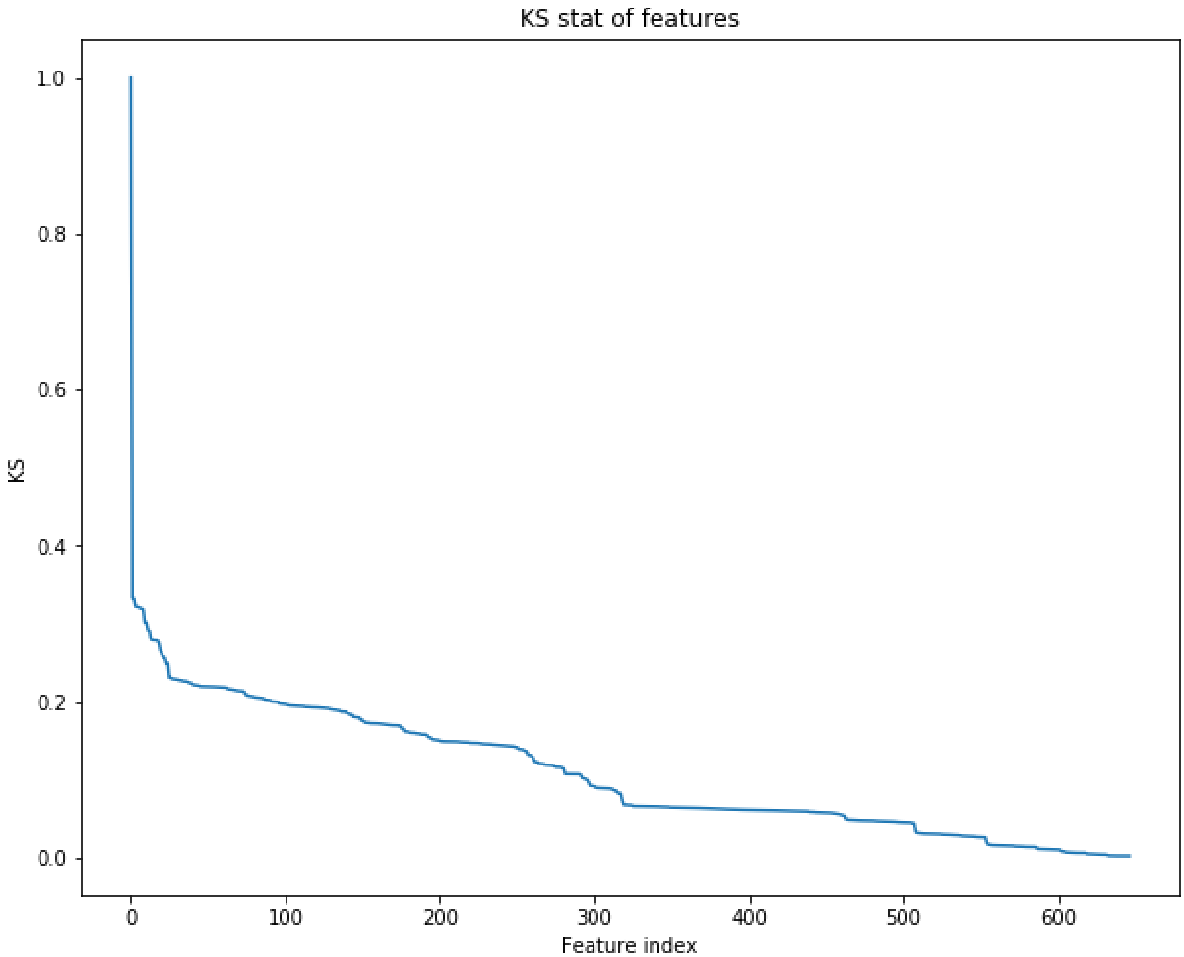
In [ ]:

```
KSFDR = np.zeros([2, data_Jan14.shape[1]])
numbads = data_Jan14['fraud_label'].sum()
topRows = int(round(len(data_Jan14)*0.03))
print('Top 3% rows:',topRows)
print('No of frauds',numbads)

start_time = pd.datetime.now()

j = 0
for column in data_Jan14:
#     KS statistic
    KSFDR[0][j] = stats.ks_2samp(inliers_class[column],outlier_class[column])[0]
#     FDR
    temp = data_Jan14.sort_values(column,ascending=False)
    temp1 = temp.head(topRows)
    temp2 = temp.tail(topRows)
    needed1 = temp1.loc[:,'fraud_label']
    needed2 = temp2.loc[:,'fraud_label']
    FDR1 = sum(needed1)/numbads
    FDR2 = sum(needed2)/numbads
    FDRate = np.maximum(FDR1,FDR2)
    KSFDR[1][j] = FDRate
    j = j +1
print('duration: ', pd.datetime.now() - start_time)
```

```
Top 3% rows: 23850
No of frauds 11486
duration:  0:11:37.001310
```

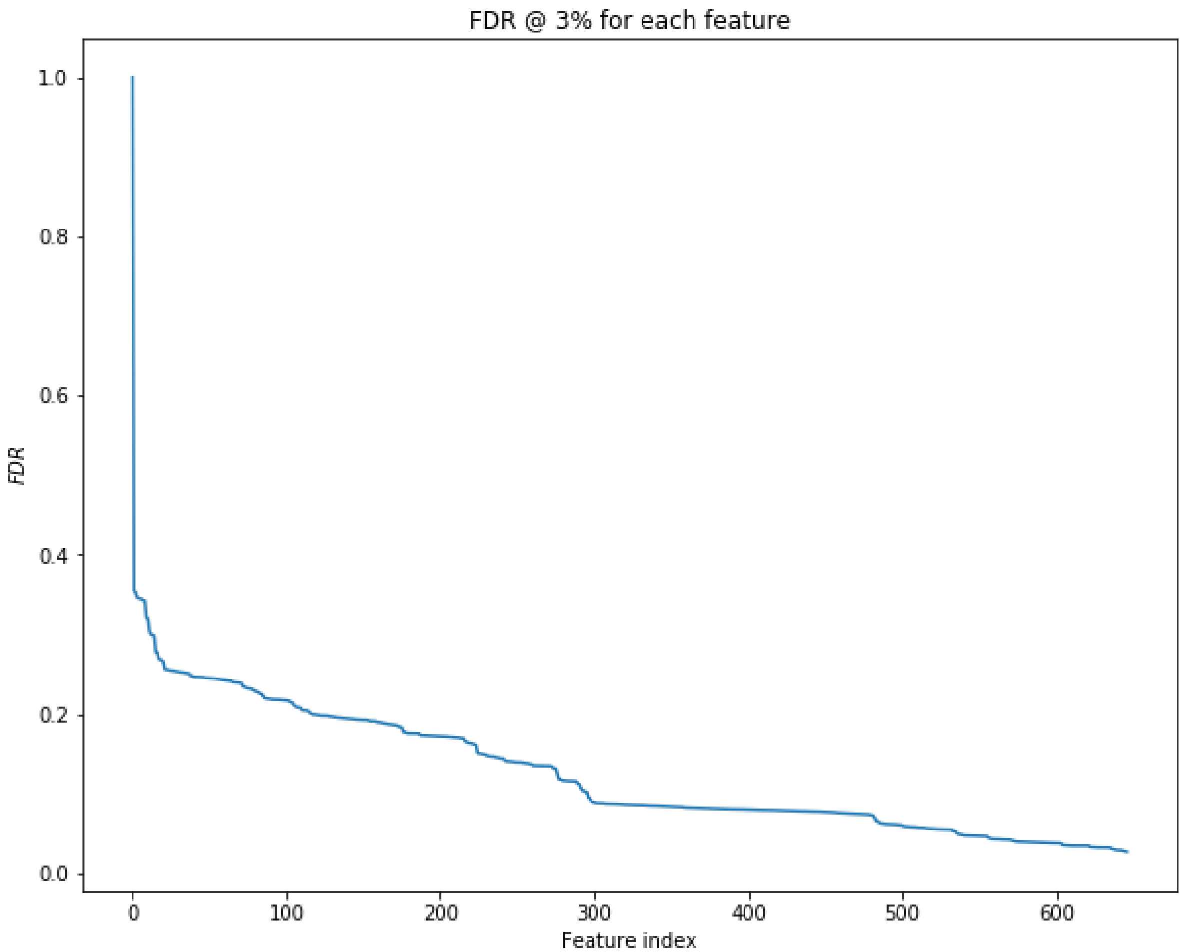
In [ ]:

```
plt.figure(figsize=(10,8))
plt.plot(-np.sort(-KSFDR[0][:]))
plt.title("KS stat of features")
plt.ylabel('KS')
plt.xlabel('Feature index')
# plt.savefig("KS_HW5.png", dpi=200)
plt.show()
```



In [ ]:

```
plt.figure(figsize=(10,8))
plt.plot(-np.sort(-KSFDR[1][:]))
plt.title("FDR @ 3% for each feature")
plt.ylabel('$FDR$')
plt.xlabel('Feature index')
plt.savefig("FDR_at_3.png", dpi=200)
plt.show()
```



Aggregated results with ranking for filter method

In [ ]:

```
col_name_red = data_Jan14.columns
res_df_prof = pd.DataFrame.from_dict({'KS':KSFDR[0][:],'FDR':KSFDR[1][:]},orient='index',
                      columns=col_name_red)
```

In [ ]:

```
res_df_prof = res_df_prof.transpose()
res_df_prof['Avg'] = res_df_prof.mean(axis = 1)
res_df_prof['KS Rank'] = res_df_prof['KS'].rank(ascending = False)
res_df_prof['FDR Rank'] = res_df_prof['FDR'].rank(ascending=False)
res_df_prof['Average Rank'] = res_df_prof[['KS Rank','FDR Rank']].mean(axis = 1)
res_df_prof_filter = res_df_prof.sort_values(by='Average Rank',ascending=True)
# res_df_filter = res_df_filter.drop(columns='rank_avg')
res_df_prof_filter
```

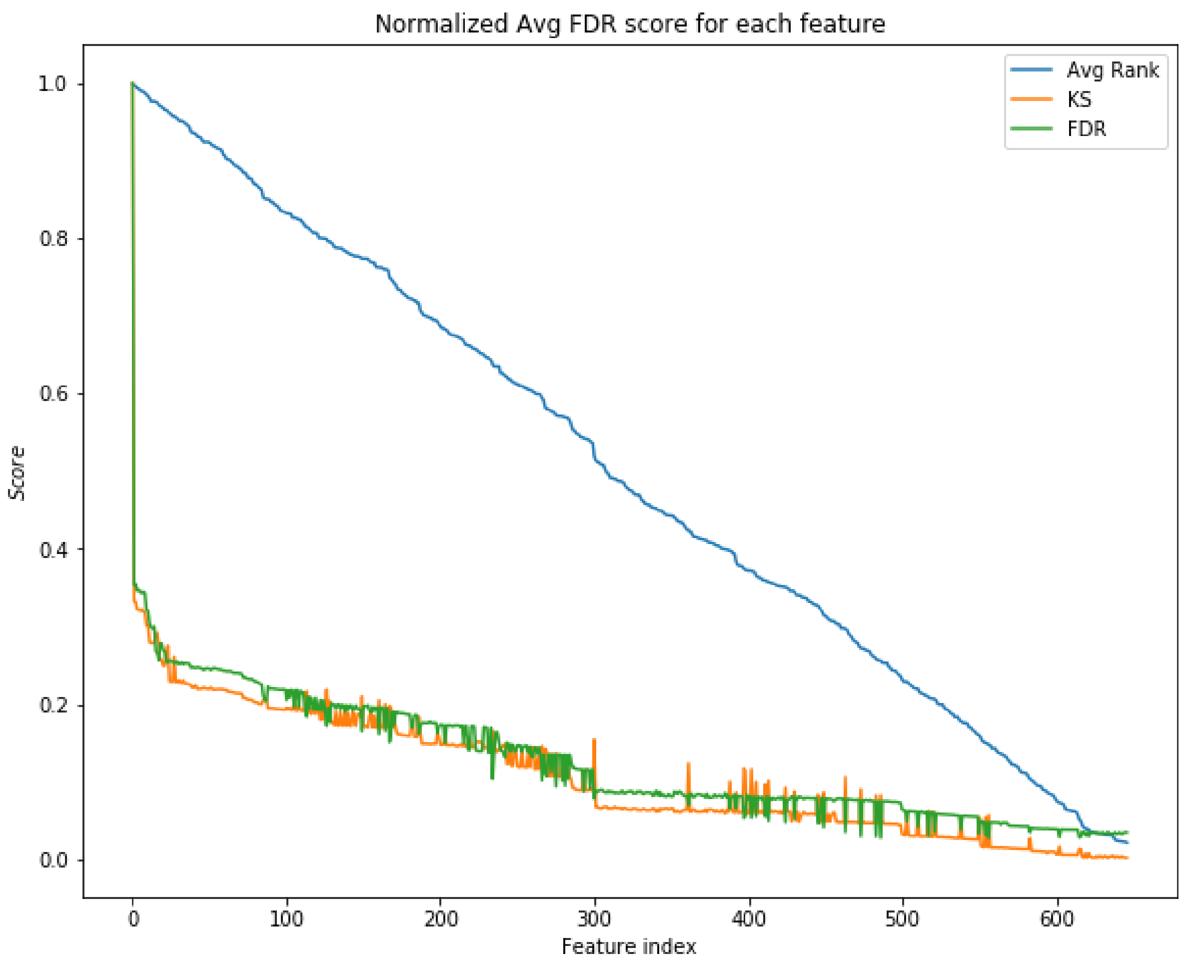
Out[ ]:

	KS	FDR	Avg	KS Rank	FDR Rank	Average Rank
fraud_label	1.000000	1.000000	1.000000	1.0	1.0	1.00
address_velocity30_date	0.332725	0.353300	0.343012	2.0	3.0	2.50
address-zip_velocity30_date	0.332032	0.354954	0.343493	3.0	2.0	2.50
address_velocity14_date	0.322252	0.345812	0.334032	4.0	5.0	4.50
address_velocity90_date	0.321087	0.346857	0.333972	6.0	4.0	5.00
...	...	...	...	...	...	...
phonessn_diff	0.003577	0.032561	0.018069	629.0	635.0	632.00
ssn_diff	0.003442	0.032648	0.018045	632.0	632.5	632.25
ssn-address-zip_velocity0_date	0.001774	0.034825	0.018299	647.0	618.0	632.50
address-zip-fullname-dob_velocity0_date	0.001813	0.034738	0.018276	644.0	621.5	632.75
ssn-homephone_velocity0_date	0.001793	0.034738	0.018265	645.0	621.5	633.25

647 rows × 6 columns

In [ ]:

```
plt.figure(figsize=(10,8))
# plt.plot(KSFDR[1][:])
plt.plot(1-res_df_prof_filter['Average Rank'].values/res_df_prof_filter.shape[0],label =
'Avg Rank')
plt.plot(res_df_prof_filter['KS'].values,label = 'KS')
plt.plot(res_df_prof_filter['FDR'].values,label = 'FDR')
plt.title("Normalized Avg FDR score for each feature")
plt.ylabel('$Score$')
plt.xlabel('Feature index')
plt.legend()
# plt.savefig("Avg_Score.png", dpi=200)
plt.show()
```



Filter method Results

We take the top 100 features from the filter method

In [ ]:

```
Y_labels = data_Jan14['fraud_label']
features_chosen_filter = res_df_prof_filter.index.values[1:101] #1st column is the fraud l
abel
features_chosen_filter
```

In [ ]:

```
X_data = data_Jan14[features_chosen_filter]
print(X_data.shape)
X_data.head()
```

```
(794996, 100)
```

Out[ ]:

	address_velocity30_date	address-zip_velocity30_date	address_velocity14_date	address_velocity90_
38511	-0.117706	-0.101958	-0.088541	-0.18:
38512	-0.117706	-0.101958	-0.088541	-0.18:
38513	-0.117706	-0.101958	-0.088541	-0.18:
38514	-0.117706	-0.101958	-0.088541	-0.18:
38515	-0.117706	-0.101958	-0.088541	-0.18:

Wrapper Method

We chose the top 100 features from the wraper method

In [ ]:

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

```
Memory percentage used 81.8
```

Recursive Feature Elimination

Recursive feature elimination using logistic regression with FDR @ 3% metric

In [ ]:

```
# creates FDR 3% metric to integrate with sklearn
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))
    return (top3_res1['label'].sum()/sum(y_true))

my_fdr_metric = make_scorer(custom_FDR, greater_is_better=True,needs_proba = True)
```

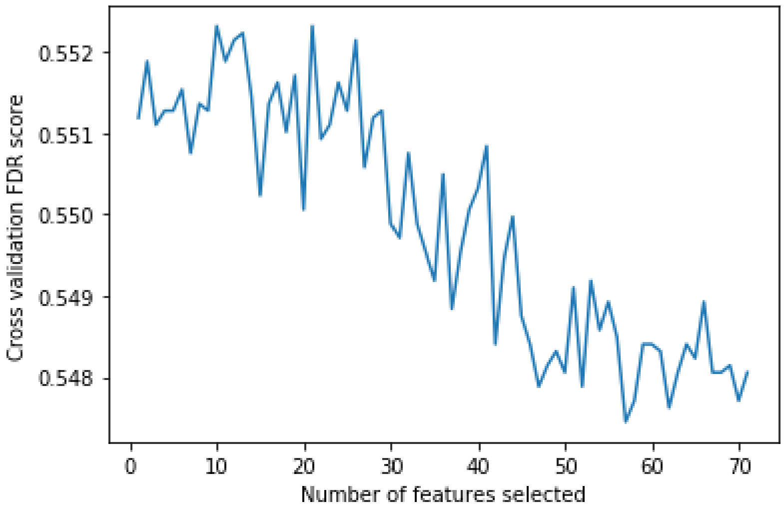
In [ ]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import StratifiedKFold
from sklearn.feature_selection import RFECV

from warnings import filterwarnings # this is to ignore convergence warnings
filterwarnings('ignore')

start_time = pd.datetime.now()
print("started at ",start_time)

log_reg = LogisticRegression()
# The "accuracy" scoring is proportional to the number of correct
# classifications
rfecv_logreg = RFECV(estimator=log_reg, step=1, cv=StratifiedKFold(2),min_features_to_sele
ct=30,
              scoring=my_fdr_metric,n_jobs=-1)
rfecv_logreg.fit(X_data, Y_labels)
print('duration: ', pd.datetime.now() - start_time)

print("Number of features chosen: %d" % rfecv_logreg.n_features_)
```

```
started at  2020-05-13 21:54:21.111047
duration:  0:19:45.027942
Optimal number of features : 39
```

In [ ]:

```
# Plot number of features VS. cross-validation scores
plt.figure()
plt.xlabel("Number of features selected")
plt.ylabel("Cross validation FDR score")
plt.plot(range(1, len(rfecv_logreg.grid_scores_) + 1), rfecv_logreg.grid_scores_)
plt.show()

#---------------------save to dataframes ----------------------------
var_selected = pd.DataFrame(sorted(zip(map(lambda x: round(x), rfecv_logreg.ranking_), X_d
ata)),
                            columns = ['ranking', 'variable'])
# var_selected.to_csv('fs_logreg_ranks.csv')

scores_csv = pd.DataFrame(rfecv_logreg.grid_scores_,columns = ['Score'])
# scores_csv.to_csv('fs_logreg_scores.csv')

features_chosen_wrapper = X_data.columns[rfecv_logreg.support_].values
print(features_chosen_wrapper)
```




```
['address-zip_velocity30_date'
 'address-zip_0_dayvel_div_7_dayvel_relvelocity'
 'address-zip_velocity1_date'
 'address_0_dayvel_div_180_dayvel_relvelocity'
 'zip-homephone_velocity30_date' 'address-zip-homephone_velocity30_date'
 'ssn-dob_velocity30_date'
 'address-zip_0_dayvel_div_180_dayvel_relvelocity'
 'fullname-dob_velocity30_date' 'ssn-fullname-dob_velocity30_date'
 'ssn-firstname_velocity30_date' 'ssn-lastname_velocity30_date'
 'fullname-ssn_velocity30_date' 'ssn-dob_velocity90_date'
 'ssn-dob_velocity180_date' 'address-zip-homephone_velocity180_date'
 'fullname-dob_velocity90_date' 'firstname-dob_velocity90_date'
 'ssn-fullname-dob_velocity90_date' 'ssn-lastname_velocity180_date'
 'ssn-firstname_velocity180_date' 'ssn-lastname_velocity90_date'
 'fullname-ssn_velocity90_date' 'ssn-fullname-dob_velocity180_date'
 'fullname-ssn_velocity180_date' 'lastname-dob_velocity14_date'
 'firstname-dob_velocity14_date' 'ssn-dob_velocity14_date'
 'ssn-lastname_velocity14_date'
 'fullname-dob_0_dayvel_div_30_dayvel_relvelocity'
 'ssn-fullname-dob_0_dayvel_div_30_dayvel_relvelocity'
 'address-zip-homephone_0_dayvel_div_30_dayvel_relvelocity'
 'zip-homephone_velocity7_date' 'address-zip-homephone_velocity7_date'
 'ssn_0_dayvel_div_30_dayvel_relvelocity'
 'lastname-dob_0_dayvel_div_14_dayvel_relvelocity'
 'firstname-dob_0_dayvel_div_14_dayvel_relvelocity'
 'ssn-dob_0_dayvel_div_14_dayvel_relvelocity' 'homephone_velocity3_date']
```

In [ ]:

```
# dataframe with RFECV ranking results
var_selected.head()
```

Out[ ]:

	ranking	variable
0	1	address-zip-homephone_0_dayvel_div_30_dayvel_r...
1	1	address-zip-homephone_velocity180_date
2	1	address-zip-homephone_velocity30_date
3	1	address-zip-homephone_velocity7_date
4	1	address-zip_0_dayvel_div_180_dayvel_relvelocity

Run recursive feature elimination (RFE) without CV for better ranking

In [ ]:

```
from sklearn.feature_selection import RFE

start_time = pd.datetime.now()
print("started at ",start_time)

log_reg = LogisticRegression()
# The "accuracy" scoring is proportional to the number of correct
# classifications
rfe_logreg = RFE(estimator=log_reg, step=1, n_features_to_select=1)
rfe_logreg.fit(X_data[features_chosen_wrapper], Y_labels)
end_time = pd.datetime.now()
print('duration: ', end_time - start_time)
```

```
started at  2020-05-14 02:15:33.806556
duration:  0:03:39.903815
```

In [ ]:

```
#--------------------split cell-----------------------------
var_selected_rfe = pd.DataFrame(sorted(zip(map(lambda x: round(x), rfe_logreg.ranking_), X
_data[features_chosen_wrapper])),
                            columns = ['ranking', 'variable'])
# var_selected_rfe.to_csv('fs_logreg_ranks91.csv')

#--------------------split cell-----------------------------
selected_columns_rfe = X_data[features_chosen_wrapper].columns[rfe_logreg.support_].values
print(selected_columns_rfe)
```

```
['ssn-lastname_velocity180_date']
```

In [ ]:

```
var_selected_rfe
```

Out[ ]:

	ranking	variable
0	1	ssn-lastname_velocity180_date
1	2	fullname-ssn_velocity180_date
2	3	ssn-firstname_velocity180_date
3	4	address-zip_velocity30_date
4	5	fullname-dob_velocity30_date
5	6	fullname-dob_0_dayvel_div_30_dayvel_relvelocity
6	7	firstname-dob_velocity14_date
7	8	firstname-dob_0_dayvel_div_14_dayvel_relvelocity
8	9	ssn-dob_velocity180_date
9	10	ssn-fullname-dob_velocity180_date
10	11	ssn-fullname-dob_velocity30_date
11	12	fullname-ssn_velocity90_date
12	13	ssn-lastname_velocity90_date
13	14	ssn-fullname-dob_0_dayvel_div_30_dayvel_relvel...
14	15	ssn-dob_velocity14_date
15	16	ssn-dob_0_dayvel_div_14_dayvel_relvelocity
16	17	ssn-firstname_velocity30_date
17	18	homephone_velocity3_date
18	19	ssn_0_dayvel_div_30_dayvel_relvelocity
19	20	ssn-lastname_velocity30_date
20	21	ssn-lastname_velocity14_date
21	22	ssn-dob_velocity30_date
22	23	address-zip_velocity1_date
23	24	fullname-ssn_velocity30_date
24	25	fullname-dob_velocity90_date
25	26	address-zip-homephone_velocity180_date
26	27	address-zip-homephone_velocity30_date
27	28	ssn-dob_velocity90_date
28	29	ssn-fullname-dob_velocity90_date
29	30	address-zip-homephone_velocity7_date
30	31	zip-homephone_velocity30_date
31	32	zip-homephone_velocity7_date
32	33	address-zip-homephone_0_dayvel_div_30_dayvel_r...
33	34	address-zip_0_dayvel_div_7_dayvel_relvelocity

	ranking	variable
34	35	address-zip_0_dayvel_div_180_dayvel_relvelocity
35	36	address_0_dayvel_div_180_dayvel_relvelocity
36	37	lastname-dob_velocity14_date
37	38	lastname-dob_0_dayvel_div_14_dayvel_relvelocity
38	39	firstname-dob_velocity90_date

In [ ]:

```
# choose the top 30 features from the RFE method
final_wrapper_columns = var_selected_rfe.variable.values[:30]
final_wrapper_columns
```

Out[ ]:

```
array(['ssn-lastname_velocity180_date', 'fullname-ssn_velocity180_date',
       'ssn-firstname_velocity180_date', 'address-zip_velocity30_date',
       'fullname-dob_velocity30_date',
       'fullname-dob_0_dayvel_div_30_dayvel_relvelocity',
       'firstname-dob_velocity14_date',
       'firstname-dob_0_dayvel_div_14_dayvel_relvelocity',
       'ssn-dob_velocity180_date', 'ssn-fullname-dob_velocity180_date',
       'ssn-fullname-dob_velocity30_date', 'fullname-ssn_velocity90_date',
       'ssn-lastname_velocity90_date',
       'ssn-fullname-dob_0_dayvel_div_30_dayvel_relvelocity',
       'ssn-dob_velocity14_date',
       'ssn-dob_0_dayvel_div_14_dayvel_relvelocity',
       'ssn-firstname_velocity30_date', 'homephone_velocity3_date',
       'ssn_0_dayvel_div_30_dayvel_relvelocity',
       'ssn-lastname_velocity30_date', 'ssn-lastname_velocity14_date',
       'ssn-dob_velocity30_date', 'address-zip_velocity1_date',
       'fullname-ssn_velocity30_date', 'fullname-dob_velocity90_date',
       'address-zip-homephone_velocity180_date',
       'address-zip-homephone_velocity30_date', 'ssn-dob_velocity90_date',
       'ssn-fullname-dob_velocity90_date',
       'address-zip-homephone_velocity7_date'], dtype=object)
```

Model Inputs

In [ ]:

```
# Inputs for models
X_models = X_data[final_wrapper_columns] # training set
Y_labels #labels
X_oot = df_oot[final_wrapper_columns] #OOT set
Y_oot = df_oot['fraud_label']
```

In [ ]:

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

Logistic Regression

Logistic regression with 10-fold stratified cross validation

Outputs 3% FDR for Train & Test sets for each fold repetition

In [ ]:

```
# creates FDR 3% metric to integrate with sklearn
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))
    return (top3_res1['label'].sum()/sum(y_true))

my_fdr_metric = make_scorer(custom_FDR, greater_is_better=True,needs_proba = True)
```

In [ ]:

```
from sklearn.model_selection import cross_validate
from sklearn.model_selection import StratifiedKFold
from sklearn.linear_model import LogisticRegression

lgr = LogisticRegression(C=0.1,penalty='l2',solver="liblinear",tol=1e-4)#,class_weight={0:
1,1:15})

current_model = lgr
cv_scores = cross_validate(current_model, X_models,Y_labels, cv=StratifiedKFold(10), scori
ng = my_fdr_metric,
                           return_train_score=True,return_estimator=True, n_jobs=-1)


print('Cross validation FDR scores for',type(current_model).__name__)
print('Train:',cv_scores['train_score'])
print('Test:',cv_scores['test_score'])

print('Average train score:',np.mean(cv_scores['train_score']))
print('Average test score:',np.mean(cv_scores['test_score']))
```

OOT FDR @ 3% for each model fitted by cross validation

In [ ]:

```
for i_model in range(0,cv_scores['fit_time'].shape[0]):
    y_est_oot = cv_scores['estimator'][i_model].predict_proba(X_oot)
    print('OOT FDR score for',type(current_model).__name__ ,str(i_model+1),':',get_FDR(y_e
st_oot[:,1],Y_oot))
```

In this part we manually tested different parameters for the Logistic Regression model

In [ ]:

```
from sklearn.model_selection import train_test_split

X_train,X_test,y_train,y_test=train_test_split(X_models,Y_labels,train_size=0.7,test_size=
0.3)
```

In [ ]:

```
#Logistic regression model build with Cross Validation
from sklearn.linear_model import LogisticRegressionCV

clf = LogisticRegressionCV(cv=10,Cs=[0.01]).fit(X_train, y_train)
#get model accuracy on TRN
clf.score(X_train, y_train)
```

Compute FDR:

Train

In [ ]:

```
#get probability on y for TRN
prob1 = clf.predict_proba(X_train)
p1 = prob1[:,1]
```

In [ ]:

```
#build a dataframe for TRN with prob(descending) and original Fraud label
df_train = X_train
df_train['prob'] = p1
df_train['org'] = y_train
df_train.sort_values(by=['prob'],ascending = False, inplace = True)
df_train.head()
```

In [ ]:

```
#FDR 3% for TRN
bads1 = df_train[df_train['org'] == 1]
numbads1 = len(bads1)

topRows1 = int(round(len(df_train)*0.03))
temp1 = df_train.head(topRows1)
needed1 = temp1.loc[:,'org']
FDR1 = sum(needed1)/numbads1

print('Train FDR is ',FDR1)
```

Test

In [ ]:

```
#get model accuracy on TEST
clf.score(X_test, y_test)
```

In [ ]:

```
#get probability on y for TEST
prob2 = clf.predict_proba(X_test)
p2 = prob2[:,1]
```

In [ ]:

```
#build a dataframe for TEST with prob(descending) and original Fraud label
df_test = X_test
df_test['prob'] = p2
df_test['org'] = y_test
df_test.sort_values(by=['prob'],ascending = False, inplace = True)
df_test.head()
```

In [ ]:

```
#FDR 3% for TEST
bads2 = df_test[df_test['org'] == 1]
numbads2 = len(bads2)
topRows2 = int(round(len(df_test)*0.03))
temp2 = df_test.head(topRows2)
needed2 = temp2.loc[:,'org']

FDR2 = sum(needed2)/numbads2
print('Test FDR is ',FDR2)
```

OOT

In [ ]:

```
#get model accuracy on OOT
clf.score(X_oot, Y_oot)
```

In [ ]:

```
#get probability on y for OOT
# p3=[]
prob3 = clf.predict_proba(X_oot)
# for probabilty in prob3:
    # p3.append(probabilty[1])
p3 =  prob3[:,1]
```

In [ ]:

```
#build a dataframe for OOT with prob(descending) and original Fraud Label
df_OOT = copy.deepcopy(X_oot)
df_OOT['prob'] = p3
df_OOT['org'] = Y_oot
df_OOT.sort_values(by=['prob'],ascending = False, inplace = True)
df_OOT.head()
```

In [ ]:

```
#FDR 3% for OOT
bads3 = df_OOT[df_OOT['org'] == 1]
numbads3 = len(bads3)
topRows3 = int(round(len(df_OOT)*0.03))
temp3 = df_OOT.head(topRows3)
needed3 = temp3.loc[:,'org']
FDR3 = sum(needed3)/numbads3

print('OOT FDR is ',FDR3)
```

Neural Network

In [ ]:

```
#  function to compute FDR
def get_FDR(y_scores, y_true):
    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))
```

In [ ]:

```
import keras.backend as K
import tensorflow as tf

def recall_m(y_true, y_pred):
    y_true = K.ones_like(y_true)
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    all_positives = K.sum(K.round(K.clip(y_true, 0, 1)))

    recall = true_positives / (all_positives + K.epsilon())
    return recall

def precision_m(y_true, y_pred):
    y_true = K.ones_like(y_true)
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))

    predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
    precision = true_positives / (predicted_positives + K.epsilon())
    return precision

def f1_score(y_true, y_pred):
    precision = precision_m(y_true, y_pred)
    recall = recall_m(y_true, y_pred)
    return 2*((precision*recall)/(precision+recall+K.epsilon()))
```

In [ ]:

```
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import Adam

num_hidden = 30 # number of nodes in hidden layer
adam_par = Adam(learning_rate= 0.01, beta_1=0.9, beta_2=0.999, amsgrad=False) # adam algor
ithm parameters


model = Sequential()
model.add(Dense(num_hidden, input_dim=X_models.shape[1], activation='relu'))
# uncomment to add extra hidden layers
# model.add(Dense(20, input_dim=12, activation='relu'))
# model.add(Dense(15, input_dim=30, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
# Compile model
# model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy', f1_score,
precision_m, recall_m, tf.keras.metrics.AUC()])
```

In [ ]:

```
no_of_epochs = 3
batch = 1000 # assign batch size in training
class_weights = {0: 0.5, 1: 8.5} # assign class weight
start_time = pd.datetime.now()
history = model.fit(x=X_models, y=Y_labels, batch_size = batch,class_weight = class_weight
s, epochs = no_of_epochs, validation_split = 0.3)
print('duration: ', pd.datetime.now() - start_time)
```

In [ ]:

```
clas_pred_train = model.predict(X_models) # get class scores
clas_pred_OOT =  model.predict(X_oot) # get class scores
print('Train FDR @ 3% is',get_FDR(np.squeeze(clas_pred),Y_labels))
print('OOT FDR is:',get_FDR(np.squeeze(clas_pred_OOT),Y_oot))   # add oot labels
```

Plots of Accuracy, AUC & Loss per epoch

In [ ]:

```
# Plot training & validation accuracy values
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```

In [ ]:

```
# plot ROC per epoch
plt.plot(history.history['auc'])
plt.plot(history.history['val_auc'])
plt.title('AOC per epoch')
plt.xlabel('Epochs')
plt.show()
```

In [ ]:

```
# Plot training & validation loss values per epoch
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```

Plot the model architecture

In [ ]:

```
from keras.utils import plot_model
plot_model(model,show_shapes=True)
```

Boosted Trees

In [ ]:

```
import xgboost as xgb
from numpy import loadtxt
from xgboost import XGBClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import train_test_split, cross_validate, GridSearchCV, KFold,
StratifiedKFold, cross_val_score, cross_val_predict
```

In [ ]:

```
# Create Train and Test sets
X_train, X_test, y_train, y_test = train_test_split(X_models, Y_labels, test_size=0.3, tra
in_size=0.7)
```

In [ ]:

```
# Determine the ratio between the negative and positive classes to use with the scale_pos_
weight parameter in XGBoost
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)
```

```
10147
689853
67.98590716467922
```

In [ ]:

```
# Instantiate the XGBoost model (XGBClassifier) so that we use it with our data
# https://medium.com/@jmcneilkeller/a-complete-classification-project-part-9-feature-selec
tion-52f746370f0c
# https://xgboost.readthedocs.io/en/latest/parameter.html

# 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')

gbc = GradientBoostingClassifier()
```

In [ ]:

```
# Define the parameters to use with GridSearchCV
parameters_xgb = {
    'max_depth': range (4, 8, 1),
    'n_estimators': range(800, 1500, 200),
    'eta': [0.1, 0.01, 0.001, 0.2, 0.3],
    'scale_pos_weight':[ratio, 1],
}
```

In [ ]:

```
# Run GridSearchCV for a boosted tree algorithm
xgb_gsCV = GridSearchCV(
    estimator=xgbClass,
    param_grid=parameters_xgb,
    scoring = 'accuracy',
    n_jobs = -1,
    cv = 2,
    verbose=True
)
```

In [ ]:

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

```
Fitting 2 folds for each of 64 candidates, totalling 128 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed: 59.7min
[Parallel(n_jobs=-1)]: Done 128 out of 128 | elapsed: 224.2min finished

DONE! 4:01:31.648245
```

In [ ]:

```
# print 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=0.1, gamma=0,
              learning_rate=0.1, max_delta_step=0, max_depth=6,
              min_child_weight=1, missing=None, n_estimators=1000, n_jobs=1,
              nthread=None, objective='binary:logistic', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=0.0147795737355760
1,
              seed=None, silent=None, subsample=1, verbosity=1)
```

In [ ]:

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

Out[ ]:

```
{'eta': 0.1, 'max_depth': 6, 'n_estimators': 1000}
```

In [ ]:

```
# output the grid search CV results to a csv file
df_xgb_gsCV = pd.DataFrame.from_dict(xgb_gsCV.cv_results_)
# df_xgb_gsCV.to_csv("df_xgb_gsCV.csv")
```

In [ ]:

```
# View gridSearch CV results
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
31	1386.755927	10.816632	27.546608	1.277331	0.01	6
47	1372.338441	0.490635	26.602365	1.206347	0.001	6
15	1402.840367	20.087639	23.702411	2.738587	0.1	6
63	988.165556	2.056438	12.175195	0.608607	0.03	6
27	1266.022478	8.725523	20.707657	0.696592	0.01	5
43	1264.481920	0.674582	19.719268	0.063076	0.001	5
11	1251.509983	15.645363	25.074892	0.953702	0.1	5
59	1251.190455	1.302172	19.476198	1.666064	0.03	5
46	1123.414586	14.853112	19.562619	1.740797	0.001	6
30	1117.876305	8.840691	18.417526	1.442322	0.01	6
14	1093.222583	13.894578	19.628337	0.722670	0.1	6
62	968.438518	15.649784	11.725000	0.080900	0.03	6
42	982.884779	2.678326	17.989223	0.416525	0.001	5
10	1007.065719	1.724184	16.362137	1.612473	0.1	5

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_eta	param_max_depth
26	994.758751	4.109844	15.766289	0.079691	0.01	5
58	988.324076	4.271145	18.226666	0.049770	0.03	5
55	1073.138961	25.016662	18.899977	1.609164	0.03	4
39	1086.384488	17.996328	20.540524	0.219883	0.001	4
7	1092.886232	25.734871	18.011058	1.095239	0.1	4
23	1114.981813	9.776365	17.035167	0.966169	0.01	4
45	823.887291	3.910665	12.290898	1.046584	0.001	6
29	801.160596	0.751207	12.300894	0.674129	0.01	6
13	826.117075	0.043452	13.546753	0.744760	0.1	6
61	829.006554	9.876885	11.198978	0.104891	0.03	6
57	753.976204	0.208111	11.197759	1.252346	0.03	5
25	736.093353	0.372643	12.539146	0.453957	0.01	5
9	758.472627	13.540424	11.866348	0.538101	0.1	5
41	764.614457	2.904616	12.673706	0.328299	0.001	5

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_eta	param_max_depth
22	873.783678	12.871075	13.188777	0.593733	0.01	4
6	859.423344	6.284130	12.922853	0.504922	0.1	4
38	864.314238	24.169728	13.554603	1.406892	0.001	4
54	875.440241	3.411567	13.542648	1.399335	0.03	4
3	908.641123	21.122371	14.089783	0.597302	0.1	3
35	925.636228	9.553643	13.368253	0.913104	0.001	3
51	928.636710	11.720538	14.031452	0.562820	0.03	3
19	903.687419	3.241815	13.340261	0.934445	0.01	3
53	652.496422	7.186276	9.198481	0.130779	0.03	4
37	661.379244	21.499829	10.026868	0.590506	0.001	4
21	638.988438	7.093909	9.820722	0.710404	0.01	4
5	649.061299	7.853430	10.868630	0.496827	0.1	4
28	568.198394	7.889100	7.727494	0.360597	0.01	6
12	534.387034	8.529765	7.270114	0.020348	0.1	6

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_eta	param_max_depth
60	544.563784	7.352594	7.873753	0.302408	0.03	6
44	529.735531	11.151198	7.930525	0.258003	0.001	6
56	511.281031	12.521060	6.418445	0.760823	0.03	5
8	499.666889	3.500345	6.549583	0.683860	0.1	5
40	505.021012	3.349094	7.303161	0.194215	0.001	5
24	476.259776	6.685044	6.592497	0.466496	0.01	5
18	741.188389	21.157295	11.288720	0.195300	0.01	3
50	736.964539	16.973906	10.326994	1.124572	0.03	3
34	735.115584	21.987119	10.319352	1.284332	0.001	3
2	733.854870	11.632546	10.982034	0.174450	0.1	3
17	549.346061	12.245809	7.820648	0.073500	0.01	3
1	535.889121	8.884735	6.237387	0.120504	0.1	3
49	547.509497	10.461042	7.291661	0.532030	0.03	3
33	557.456668	12.581310	6.793473	0.014799	0.001	3

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_eta	param_max_depth
4	440.529781	0.348576	6.398714	0.203650	0.1	4
36	425.622041	1.095743	5.343126	0.193460	0.001	4
20	433.156206	3.697633	6.256798	0.208771	0.01	4
52	429.821677	9.415452	5.459770	0.481633	0.03	4
48	375.396802	13.118127	4.725834	0.005556	0.03	3
32	361.823882	6.420458	4.415355	0.162433	0.001	3
16	385.066238	21.883579	4.339853	0.023735	0.01	3
0	373.683886	4.110579	4.689392	0.035417	0.1	3

In [ ]:

```
# Obtain the tng score/accuracy when trained with the features GridSearchCV selected
tng_score_xgb_gsCV = xgb_gsCV.score(X_train, y_train)
tng_score_xgb_gsCV
```

Out[ ]:

```
0.9898040780094053
```

In [ ]:

```
# # Obtain the test score/accuracy when trained with the features GridSearchCV selected
test_score_xgb_gsCV = xgb_gsCV.score(X_test, y_test)
# # test_score_list.append(test_score)
test_score_xgb_gsCV
```

Out[ ]:

```
0.9899328718359406
```

In [ ]:

```
# Obtain the OOT score/accuracy when trained with the features GridSearchCV selected
oot_score_xgb_gsCV = xgb_gsCV.score(X_oot, Y_oot)
oot_score_xgb_gsCV
```

Out[ ]:

```
0.9896512165676635
```

Run XGBoost

Run XGBoost with different parameters and analyze the output

In [ ]:

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

In [ ]:

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

In [ ]:

```

```

In [ ]:

```
start_xgbfit=pd.datetime.now()
xgb_1 =  xgb.XGBClassifier(objective='binary:logistic', learning_rate = 0.01,
                           n_estimators=600, max_depth=5, scale_pos_weight=ratio)
xgb_1.fit(X_train, y_train)
print("DONE!", pd.datetime.now()-start_xgbfit)
```

```
DONE! 0:07:59.189273
```

In [ ]:

```
start_xgbfit=pd.datetime.now()
xgb_2 =  xgb.XGBClassifier(objective='binary:logistic', learning_rate = 0.01,
                           n_estimators=800, max_depth=5, scale_pos_weight=ratio)
xgb_2.fit(X_train, y_train)
print("DONE!", pd.datetime.now()-start_xgbfit)
```

```
DONE! 0:10:31.176258
```

In [ ]:

```
start_xgbfit=pd.datetime.now()
xgb_3 =  xgb.XGBClassifier(objective='binary:logistic', learning_rate = 0.01,
                           n_estimators=1000, max_depth=5, scale_pos_weight=ratio)
xgb_3.fit(X_train, y_train)
print("DONE!", pd.datetime.now()-start_xgbfit)
```

```
DONE! 0:12:56.761847
```

In [ ]:

```
start_xgbfit=pd.datetime.now()
xgb_4 =  xgb.XGBClassifier(objective='binary:logistic', learning_rate = 0.01,
                           n_estimators=600, max_depth=4, scale_pos_weight=ratio)
xgb_4.fit(X_train, y_train)
print("DONE!", pd.datetime.now()-start_xgbfit)
```

```
DONE! 0:06:40.565662
```

In [ ]:

```
start_xgbfit=pd.datetime.now()
xgb_5 =  xgb.XGBClassifier(objective='binary:logistic', learning_rate = 0.01,
                           n_estimators=800, max_depth=4, scale_pos_weight=ratio)
xgb_5.fit(X_train, y_train)
print("DONE!", pd.datetime.now()-start_xgbfit)
```

```
DONE! 0:08:51.674261
```

In [ ]:

```
start_xgbfit=pd.datetime.now()
xgb_6 =  xgb.XGBClassifier(objective='binary:logistic', learning_rate = 0.01,
                           n_estimators=1000, max_depth=4, scale_pos_weight=ratio)
xgb_6.fit(X_train, y_train)
print("DONE!", pd.datetime.now()-start_xgbfit)
```

```
DONE! 0:11:05.207002
```

In [ ]:

```
start_xgbfit=pd.datetime.now()
xgb_7 =  xgb.XGBClassifier(objective='binary:logistic', learning_rate = 0.001,
                           n_estimators=600, max_depth=5, scale_pos_weight=ratio)
xgb_7.fit(X_train, y_train)
print("DONE!", pd.datetime.now()-start_xgbfit)
```

DONE! 0:07:45.501011

In [ ]:

```
start_xgbfit=pd.datetime.now()
xgb_8 =  xgb.XGBClassifier(objective='binary:logistic', learning_rate = 0.001,
                           n_estimators=800, max_depth=5, scale_pos_weight=ratio)
xgb_8.fit(X_train, y_train)
print("DONE!", pd.datetime.now()-start_xgbfit)
```

DONE! 0:10:19.677419

In [ ]:

```
start_xgbfit=pd.datetime.now()
xgb_9 =  xgb.XGBClassifier(objective='binary:logistic', learning_rate = 0.001,
                           n_estimators=1000, max_depth=5, scale_pos_weight=ratio)
xgb_9.fit(X_train, y_train)
print("DONE!", pd.datetime.now()-start_xgbfit)
```

DONE! 0:12:56.165312

In [ ]:

```
start_xgbfit=pd.datetime.now()
xgb_10 =  xgb.XGBClassifier(objective='binary:logistic', learning_rate = 0.001,
                           n_estimators=600, max_depth=4, scale_pos_weight=ratio)
xgb_10.fit(X_train, y_train)
print("DONE!", pd.datetime.now()-start_xgbfit)
```

DONE! 0:06:40.209160

In [ ]:

```
start_xgbfit=pd.datetime.now()
xgb_11 =  xgb.XGBClassifier(objective='binary:logistic', learning_rate = 0.001,
                           n_estimators=800, max_depth=4, scale_pos_weight=ratio)
xgb_11.fit(X_train, y_train)
print("DONE!", pd.datetime.now()-start_xgbfit)
```

DONE! 0:08:56.770541

In [ ]:

```
start_xgbfit=pd.datetime.now()
xgb_12 =  xgb.XGBClassifier(objective='binary:logistic', learning_rate = 0.001,
                           n_estimators=1000, max_depth=4, scale_pos_weight=ratio)
xgb_12.fit(X_train, y_train)
print("DONE!", pd.datetime.now()-start_xgbfit)
```

DONE! 0:11:08.132761

In [ ]:

```
start_xgbfit=pd.datetime.now()
xgb_13 =  xgb.XGBClassifier(objective='binary:logistic', learning_rate = 0.01,
                           n_estimators=600, max_depth=6, scale_pos_weight=ratio)
xgb_13.fit(X_train, y_train)
print("DONE!", pd.datetime.now()-start_xgbfit)
```

DONE! 0:11:51.525173

In [ ]:

```
start_xgbfit=pd.datetime.now()
xgb_14 =  xgb.XGBClassifier(objective='binary:logistic', learning_rate = 0.01,
                           n_estimators=800, max_depth=6, scale_pos_weight=ratio)
xgb_14.fit(X_train, y_train)
print("DONE!", pd.datetime.now()-start_xgbfit)
```

DONE! 0:16:51.773824

In [ ]:

```
start_xgbfit=pd.datetime.now()
xgb_15 =  xgb.XGBClassifier(objective='binary:logistic', learning_rate = 0.01,
                           n_estimators=1000, max_depth=6, scale_pos_weight=ratio)
xgb_15.fit(X_train, y_train)
print("DONE!", pd.datetime.now()-start_xgbfit)
```

DONE! 0:16:35.006191

In [ ]:

```
start_xgbfit=pd.datetime.now()
xgb_16 =  xgb.XGBClassifier(objective='binary:logistic', learning_rate = 0.01,
                           n_estimators=1200, max_depth=6, scale_pos_weight=ratio)
xgb_16.fit(X_train, y_train)
print("DONE!", pd.datetime.now()-start_xgbfit)
```

DONE! 0:20:10.236810

Analyze the Results

Analyze the results from the various boosted tree models. They are stored in results_dict.

In [ ]:

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

results_dict_XGB={}
for num in range(1,17):
    curr_time=pd.datetime.now()
    curr_model_name = "xgb_" + str(num)

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

    # 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_XGB[curr_model_name]['scores']['train_score'] = train_score
    results_dict_XGB[curr_model_name]['scores']['test_score'] = test_score
    results_dict_XGB[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_XGB[curr_model_name]['data']['train_data'] = train_data
    results_dict_XGB[curr_model_name]['data']['test_data'] = test_data
    results_dict_XGB[curr_model_name]['data']['oot_data'] = oot_data

    # Calculate the FDRs
    train_FDR = fdr_XGB(train_data, 0.03)
    test_FDR = fdr_XGB(test_data, 0.03)
    oot_FDR = fdr_XGB(oot_data, 0.03)

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

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

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

In [ ]:

```
for model in results_dict_XGB.keys():
    print(model)
    print(results_dict_XGB[model]['FDR'])
    print("")
```

Random Forest

In [ ]:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import StratifiedKFold
```

In [ ]:

```
#splitting training into 70:30 ratio for training and testing
# Not using Random State here in order to see the variation in the results
X_train,X_test,Y_train,Y_test = train_test_split(X_models,Y_labels,train_size=0.7,test_siz
e = 0.30)
```

In [ ]:

```
start = pd.datetime.now()
# Create the parameter grid
param_grid = {
    'bootstrap': [True],
    'max_depth': [80, 90, 100, 110],
    'max_features': [6, 7, 8, 9],
    'n_estimators': [300,400,500,600,700,800]
}

rf = RandomForestClassifier()

# Instantiate the grid search model
grid_search = GridSearchCV(estimator = rf, param_grid = param_grid,
                          cv = 2, n_jobs = -1, verbose = 2,scoring='neg_mean_squared_erro
r')
print(pd.datetime.now()-start)
```

In [ ]:

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

# Fit grid model with training data
grid_search.fit(X_train, Y_train)

# Print the best parameters
grid_search.best_params_

print(pd.datetime.now()-start)
```

In [ ]:

```
# The following models are being fitted for fine tuning the parameters by manually
# fitting with parameters by slightly varying them around the best parameters for better t
est fdr
```

In [ ]:

```
random_forest_1 = RandomForestClassifier(max_features = 7,
                                            n_estimators = 400, max_depth=60)
random_forest_1.fit(X_train,Y_train)
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:3: DataConversio
nWarning: A column-vector y was passed when a 1d array was expected. Please c
hange the shape of y to (n_samples,), for example using ravel().
  This is separate from the ipykernel package so we can avoid doing imports u
ntil
```

Out[ ]:

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=60, max_features=7,
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=400,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

In [ ]:

```
random_forest_2 = RandomForestClassifier(max_features = 7,
                                            n_estimators = 400,max_depth=70)
random_forest_2.fit(X_train,Y_train)
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:3: DataConversio
nWarning: A column-vector y was passed when a 1d array was expected. Please c
hange the shape of y to (n_samples,), for example using ravel().
  This is separate from the ipykernel package so we can avoid doing imports u
ntil
```

Out[ ]:

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=70, max_features=7,
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=400,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

In [ ]:

```
random_forest_3 = RandomForestClassifier(max_features = 7,
                                            n_estimators = 400,max_depth=80)
random_forest_3.fit(X_train,Y_train)
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:3: DataConversio
nWarning: A column-vector y was passed when a 1d array was expected. Please c
hange the shape of y to (n_samples,), for example using ravel().
  This is separate from the ipykernel package so we can avoid doing imports u
ntil
```

Out[ ]:

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=80, max_features=7,
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=400,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

In [ ]:

```
random_forest_4 = RandomForestClassifier(max_features = 7,
                                            n_estimators = 500,max_depth=60)
random_forest_4.fit(X_train,Y_train)
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:3: DataConversio
nWarning: A column-vector y was passed when a 1d array was expected. Please c
hange the shape of y to (n_samples,), for example using ravel().
  This is separate from the ipykernel package so we can avoid doing imports u
ntil
```

Out[ ]:

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=60, max_features=7,
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=500,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

In [ ]:

```
random_forest_5 = RandomForestClassifier(max_features = 7,
                                         n_estimators = 500,max_depth=70)
random_forest_5.fit(X_train,Y_train)
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:3: DataConversio
nWarning: A column-vector y was passed when a 1d array was expected. Please c
hange the shape of y to (n_samples,), for example using ravel().
  This is separate from the ipykernel package so we can avoid doing imports u
ntil
```

Out[ ]:

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=70, max_features=7,
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=500,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

In [ ]:

```
random_forest_6 = RandomForestClassifier(max_features = 7,
                                         n_estimators = 500,max_depth=80)
random_forest_6.fit(X_train,Y_train)
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:3: DataConversio
nWarning: A column-vector y was passed when a 1d array was expected. Please c
hange the shape of y to (n_samples,), for example using ravel().
  This is separate from the ipykernel package so we can avoid doing imports u
ntil
```

Out[ ]:

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=80, max_features=7,
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=500,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

In [ ]:

```
random_forest_7 = RandomForestClassifier(max_features = 7,
                                            n_estimators = 500,max_depth=90)
random_forest_7.fit(X_train,Y_train)
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:3: DataConversio
nWarning: A column-vector y was passed when a 1d array was expected. Please c
hange the shape of y to (n_samples,), for example using ravel().
  This is separate from the ipykernel package so we can avoid doing imports u
ntil
```

Out[ ]:

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=90, max_features=7,
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=500,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

In [ ]:

```
random_forest_8 = RandomForestClassifier(max_features = 7,
                                            n_estimators = 600,max_depth=60)
random_forest_8.fit(X_train,Y_train)
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:3: DataConversio
nWarning: A column-vector y was passed when a 1d array was expected. Please c
hange the shape of y to (n_samples,), for example using ravel().
  This is separate from the ipykernel package so we can avoid doing imports u
ntil
```

Out[ ]:

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=60, max_features=7,
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=600,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

In [ ]:

```
random_forest_9 = RandomForestClassifier(max_features = 7,
                                            n_estimators = 600,max_depth=70)
random_forest_9.fit(X_train,Y_train)
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:3: DataConversio
nWarning: A column-vector y was passed when a 1d array was expected. Please c
hange the shape of y to (n_samples,), for example using ravel().
  This is separate from the ipykernel package so we can avoid doing imports u
ntil
```

Out[ ]:

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=70, max_features=7,
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=600,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

In [ ]:

```
random_forest_10 = RandomForestClassifier(max_features = 7,n_estimators = 600,max_depth=80
)
random_forest_10.fit(X_train,Y_train)
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:2: DataConversio
nWarning: A column-vector y was passed when a 1d array was expected. Please c
hange the shape of y to (n_samples,), for example using ravel().
```

Out[ ]:

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=80, max_features=7,
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=600,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

In [ ]:

```
random_forest_11 = RandomForestClassifier(max_features = 7,
                                          n_estimators = 600,max_depth=90)
random_forest_11.fit(X_train,Y_train)
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:3: DataConversio
nWarning: A column-vector y was passed when a 1d array was expected. Please c
hange the shape of y to (n_samples,), for example using ravel().
  This is separate from the ipykernel package so we can avoid doing imports u
ntil
```

Out[ ]:

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=90, max_features=7,
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=600,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

In [ ]:

```
random_forest_12 = RandomForestClassifier(max_features = 7,
                                          n_estimators = 650,max_depth=80)
random_forest_12.fit(X_train,Y_train)
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:3: DataConversio
nWarning: A column-vector y was passed when a 1d array was expected. Please c
hange the shape of y to (n_samples,), for example using ravel().
  This is separate from the ipykernel package so we can avoid doing imports u
ntil
```

Out[ ]:

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=80, max_features=7,
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=650,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

Random Forest Results

In [ ]:

```
# Function to calculate Fraud detection rate at 3% depth for different models
def fdr_RF(model,X_data,Y_data):
    Y_data = pd.DataFrame(Y_data)
    Y_data['Fraud Proba'] = model.predict_proba(X_data)[:,1].tolist()
    Y_data = Y_data.sort_values(by='Fraud Proba',ascending=False)
    total_bads = Y_data['fraud_label'][Y_data['fraud_label']==1].count()
    top_rows = int(len(X_data)*.03)
    sum_bads = Y_data['fraud_label'].head(top_rows)[Y_data['fraud_label']==1].count()
    fdr = sum_bads/total_bads
    return fdr*100
```

In [ ]:

```
time_results = pd.datetime.now()
results_dict_RF = {}
for i in range(1,13):
    curr_time = pd.datetime.now()
    model_name = "random_forest_"+str(i)
    results_dict_RF[model_name] = {'fdr':{}}

    # calculate fdr for training, testing, and validation sets
    results_dict_RF[model_name]['fdr']['train_fdr_30']=fdr_RF(vars()[model_name],X_train,Y
_train)
    results_dict_RF[model_name]['fdr']['test_fdr_30']=fdr_RF(vars()[model_name],X_test,Y_t
est)
    results_dict_RF[model_name]['fdr']['oot_fdr_30']=fdr_RF(vars()[model_name],X_oot,Y_oot
)

    print("Done with:",model_name, ";  time:",pd.datetime.now()-curr_time)

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

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:1: FutureWarnin
g: The pandas.datetime class is deprecated and will be removed from pandas in
a future version. Import from datetime instead.
  """Entry point for launching an IPython kernel.
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:4: FutureWarnin
g: The pandas.datetime class is deprecated and will be removed from pandas in
a future version. Import from datetime instead.
  after removing the cwd from sys.path.
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:13: FutureWarnin
g: The pandas.datetime class is deprecated and will be removed from pandas in
a future version. Import from datetime instead.
  del sys.path[0]
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:4: FutureWarnin
g: The pandas.datetime class is deprecated and will be removed from pandas in
a future version. Import from datetime instead.
  after removing the cwd from sys.path.

Done with: random_forest_1 ;  time: 0:00:58.535585

/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:13: FutureWarnin
g: The pandas.datetime class is deprecated and will be removed from pandas in
a future version. Import from datetime instead.
  del sys.path[0]
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:4: FutureWarnin
g: The pandas.datetime class is deprecated and will be removed from pandas in
a future version. Import from datetime instead.
  after removing the cwd from sys.path.


Done with: random_forest_2 ;  time: 0:00:57.979901

/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:13: FutureWarnin
g: The pandas.datetime class is deprecated and will be removed from pandas in
a future version. Import from datetime instead.
  del sys.path[0]
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:4: FutureWarnin
g: The pandas.datetime class is deprecated and will be removed from pandas in
a future version. Import from datetime instead.
  after removing the cwd from sys.path.

Done with: random_forest_3 ;  time: 0:00:58.158208

/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:13: FutureWarnin
g: The pandas.datetime class is deprecated and will be removed from pandas in
a future version. Import from datetime instead.
  del sys.path[0]
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:4: FutureWarnin
g: The pandas.datetime class is deprecated and will be removed from pandas in
a future version. Import from datetime instead.
  after removing the cwd from sys.path.

Done with: random_forest_4 ;  time: 0:01:12.995941
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:13: FutureWarnin
g: The pandas.datetime class is deprecated and will be removed from pandas in
a future version. Import from datetime instead.
  del sys.path[0]
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:4: FutureWarnin
g: The pandas.datetime class is deprecated and will be removed from pandas in
a future version. Import from datetime instead.
  after removing the cwd from sys.path.

Done with: random_forest_5 ;  time: 0:01:12.583308

/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:13: FutureWarnin
g: The pandas.datetime class is deprecated and will be removed from pandas in
a future version. Import from datetime instead.
  del sys.path[0]
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:4: FutureWarnin
g: The pandas.datetime class is deprecated and will be removed from pandas in
a future version. Import from datetime instead.
  after removing the cwd from sys.path.

Done with: random_forest_6 ;  time: 0:01:17.933517

/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:13: FutureWarnin
g: The pandas.datetime class is deprecated and will be removed from pandas in
a future version. Import from datetime instead.
  del sys.path[0]
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:4: FutureWarnin
g: The pandas.datetime class is deprecated and will be removed from pandas in
a future version. Import from datetime instead.
  after removing the cwd from sys.path.


Done with: random_forest_7 ;  time: 0:01:12.659784

/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:13: FutureWarnin
g: The pandas.datetime class is deprecated and will be removed from pandas in
a future version. Import from datetime instead.
  del sys.path[0]
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:4: FutureWarnin
g: The pandas.datetime class is deprecated and will be removed from pandas in
a future version. Import from datetime instead.
  after removing the cwd from sys.path.

Done with: random_forest_8 ;  time: 0:01:24.139886

/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:13: FutureWarnin
g: The pandas.datetime class is deprecated and will be removed from pandas in
a future version. Import from datetime instead.
  del sys.path[0]
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:4: FutureWarnin
g: The pandas.datetime class is deprecated and will be removed from pandas in
a future version. Import from datetime instead.
  after removing the cwd from sys.path.

Done with: random_forest_9 ;  time: 0:01:27.896470
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:13: FutureWarnin
g: The pandas.datetime class is deprecated and will be removed from pandas in
a future version. Import from datetime instead.
  del sys.path[0]
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:4: FutureWarnin
g: The pandas.datetime class is deprecated and will be removed from pandas in
a future version. Import from datetime instead.
  after removing the cwd from sys.path.

Done with: random_forest_10 ;  time: 0:01:28.380382

/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:13: FutureWarnin
g: The pandas.datetime class is deprecated and will be removed from pandas in
a future version. Import from datetime instead.
  del sys.path[0]
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:4: FutureWarnin
g: The pandas.datetime class is deprecated and will be removed from pandas in
a future version. Import from datetime instead.
  after removing the cwd from sys.path.

Done with: random_forest_11 ;  time: 0:01:28.400079
Done with: random_forest_12 ;  time: 0:01:35.220182
DONE! 0:15:14.889182

/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:13: FutureWarnin
g: The pandas.datetime class is deprecated and will be removed from pandas in
a future version. Import from datetime instead.
  del sys.path[0]
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:15: FutureWarnin
g: The pandas.datetime class is deprecated and will be removed from pandas in
a future version. Import from datetime instead.
  from ipykernel import kernelapp as app
```

In [ ]:

```
# Print the train, test, and validation set fdr results
results_dict_RF
```

Out[ ]:

```
{'random_forest_1': {'fdr': {'oot_fdr_30': 51.55071248952221,
   'test_fdr_30': 52.93955285674855,
   'train_fdr_30': 53.66173664122137}},
 'random_forest_10': {'fdr': {'oot_fdr_30': 51.46689019279128,
   'test_fdr_30': 52.85674855092465,
   'train_fdr_30': 53.69751908396947}},
 'random_forest_11': {'fdr': {'oot_fdr_30': 51.592623637887684,
   'test_fdr_30': 52.85674855092465,
   'train_fdr_30': 53.73330152671756}},
 'random_forest_12': {'fdr': {'oot_fdr_30': 51.55071248952221,
   'test_fdr_30': 52.691139939276844,
   'train_fdr_30': 53.73330152671756}},
 'random_forest_2': {'fdr': {'oot_fdr_30': 51.46689019279128,
   'test_fdr_30': 52.663538504002204,
   'train_fdr_30': 53.69751908396947}},
 'random_forest_3': {'fdr': {'oot_fdr_30': 51.46689019279128,
   'test_fdr_30': 52.691139939276844,
   'train_fdr_30': 53.76908396946565}},
 'random_forest_4': {'fdr': {'oot_fdr_30': 51.676445934618606,
   'test_fdr_30': 52.663538504002204,
   'train_fdr_30': 53.68559160305344}},
 'random_forest_5': {'fdr': {'oot_fdr_30': 51.424979044425825,
   'test_fdr_30': 52.663538504002204,
   'train_fdr_30': 53.75715648854962}},
 'random_forest_6': {'fdr': {'oot_fdr_30': 51.508801341156754,
   'test_fdr_30': 52.580734198178305,
   'train_fdr_30': 53.76908396946565}},
 'random_forest_7': {'fdr': {'oot_fdr_30': 51.55071248952221,
   'test_fdr_30': 52.85674855092465,
   'train_fdr_30': 53.64980916030534}},
 'random_forest_8': {'fdr': {'oot_fdr_30': 51.592623637887684,
   'test_fdr_30': 52.88434998619928,
   'train_fdr_30': 53.590171755725194}},
 'random_forest_9': {'fdr': {'oot_fdr_30': 51.592623637887684,
   'test_fdr_30': 52.691139939276844,
   'train_fdr_30': 53.74522900763359}}}
```

In [ ]:

```
# Get the best model number with highest test fdr
for k,v in results_dict_RF.items():
  maximum_key = max(results_dict_RF, key=lambda v: results_dict_RF[v]['fdr']['test_fdr_30'
])
vars()[maximum_key]
```

Final model results tables

The following function generates the table of the final results of the model we chose

In [ ]:

```
def output_table(y_res_valid,Y_valid):
# returs the output tables
# y_res_valis is y_pred
# Y_valid is y_true
# returns:
# cumulative dataset, Bin Statistics
    no_of_bads =Y_valid.sum()
    no_of_records = len(Y_valid)
    no_of_goods = no_of_records - no_of_bads
    print('no_of_records',no_of_records)
    print('No of bads',no_of_bads)
    print('no_of_goods',no_of_goods)

    fin_tabl_df = pd.DataFrame({'score':y_res_valid,'label': Y_valid}).sort_values(by='sco
re',ascending=False)
    df_pres_cum = pd.DataFrame(columns=['Total # Records','# Goods','# Bads'])
    for i_tbl_df in range(1,21): #21 is the final
    #     print(i_tbl_df)
        top3_res2 = fin_tabl_df.head(round(Y_valid.shape[0]*(i_tbl_df/100)))
        top3_res2#['label'].sum()/sum(Y_valid)
    #     print('No of records:',top3_res2.shape[0])
        df_pres_cum.loc[i_tbl_df-1,'Total # Records'] = top3_res2.shape[0]
    #     print('No of bads:',top3_res2['label'].sum())
        df_pres_cum.loc[i_tbl_df-1,'# Bads'] = top3_res2['label'].sum()
    #     print('No of goods:',top3_res2.shape[0]- top3_res2['label'].sum())
        df_pres_cum.loc[i_tbl_df-1,'# Goods'] = top3_res2.shape[0]- top3_res2['label'].sum
()

    df_pres = df_pres_cum.diff()
    df_pres.loc[0] = df_pres_cum.loc[0]
    df_pres = df_pres.rename(columns={"Total # Records": "# Records"})
    df_pres['% Goods'] = 100*(df_pres['# Goods']/df_pres['# Records'])
    df_pres['% Bads'] = 100*(df_pres['# Bads']/df_pres['# Records'])
    print('Bin statistics')
    df_pres.to_csv(path+'Test_Bin_stats.csv')
    display(df_pres)

    df_pres_cum['% Goods'] = 100*(df_pres_cum['# Goods']/no_of_goods)
    df_pres_cum['% Bads'] = 100*(df_pres_cum['# Bads']/no_of_bads)
    df_pres_cum['KS'] = df_pres_cum['% Bads'] - df_pres_cum['% Goods']
    df_pres_cum['FPR'] = df_pres_cum['# Goods']/df_pres_cum['# Bads']
    df_pres_cum = df_pres_cum.rename(columns={"# Goods": "Cumulative Goods",'# Bads':'Cumu
lative Bads','% Bads':'% Bads (FDR)'})
    print('Cumulative results')
    # df_pres_cum.to_csv(path+'Test_Cum_stats.csv') To save into a csv file
    display(df_pres_cum)
    return df_pres_cum, df_pres
```

In [ ]:

```
# Concatenating Fraud probability fitted with the best model with actual fraud label for c
alculating bin statistics
Y_train['Fraud Proba'] =  vars()[maximum_key].predict_proba(X_train)[:,1].tolist()
Y_test['Fraud Proba'] =  vars()[maximum_key].predict_proba(X_test)[:,1].tolist()
Y_oot['Fraud Proba'] =  vars()[maximum_key].predict_proba(X_oot)[:,1].tolist()
```

Final results for the training set

In [ ]:

```
# Training Set
output_table(Y_train['Fraud Proba'],Y_train['fraud_label'])
```

```
no_of_records 583454
No of bads 8339
no_of_goods 575115
Bin statistics
```

	# Records	# Goods	# Bads	% Goods	% Bads
0	5835	1592	4243	27.2836	72.7164
1	5834	5689	145	97.5146	2.48543
2	5835	5778	57	99.0231	0.976864
3	5834	5785	49	99.1601	0.839904
4	5835	5791	44	99.2459	0.75407
5	5834	5786	48	99.1772	0.822763
6	5835	5800	35	99.4002	0.599829
7	5834	5787	47	99.1944	0.805622
8	5835	5795	40	99.3145	0.685518
9	5834	5792	42	99.2801	0.719918
10	5835	5792	43	99.2631	0.736932
11	5834	5792	42	99.2801	0.719918
12	5835	5799	36	99.383	0.616967
13	5835	5785	50	99.1431	0.856898
14	5834	5784	50	99.143	0.857045
15	5835	5796	39	99.3316	0.66838
16	5834	5784	50	99.143	0.857045
17	5835	5786	49	99.1602	0.83976
18	5834	5802	32	99.4515	0.548509
19	5835	5798	37	99.3659	0.634105

```
Cumulative results
```

	Total # Records	Cumulative Goods	Cumulative Bads	% Goods	% Bads (FDR)	KS	FPR
0	5835	1592	4243	0.276814	50.8814	50.6046	0.375206
1	11669	7281	4388	1.26601	52.6202	51.3542	1.6593
2	17504	13059	4445	2.27068	53.3038	51.0331	2.93791
3	23338	18844	4494	3.27656	53.8914	50.6148	4.19315
4	29173	24635	4538	4.28349	54.419	50.1355	5.4286
5	35007	30421	4586	5.28955	54.9946	49.7051	6.63345
6	40842	36221	4621	6.29804	55.4143	49.1163	7.83835
7	46676	42008	4668	7.30428	55.9779	48.6737	8.99914
8	52511	47803	4708	8.3119	56.4576	48.1457	10.1536
9	58345	53595	4750	9.31901	56.9613	47.6423	11.2832
10	64180	59387	4793	10.3261	57.4769	47.1508	12.3904
11	70014	65179	4835	11.3332	57.9806	46.6474	13.4807
12	75849	70978	4871	12.3415	58.4123	46.0707	14.5715
13	81684	76763	4921	13.3474	59.0119	45.6645	15.5991
14	87518	82547	4971	14.3531	59.6115	45.2583	16.6057
15	93353	88343	5010	15.3609	60.0791	44.7182	17.6333
16	99187	94127	5060	16.3666	60.6787	44.3121	18.6022
17	105022	99913	5109	17.3727	61.2663	43.8936	19.5563
18	110856	105715	5141	18.3815	61.6501	43.2685	20.5631
19	116691	111513	5178	19.3897	62.0938	42.7041	21.5359

Out[ ]:

```
(   Total # Records Cumulative Goods Cumulative Bads   % Goods % Bads (FDR)
KS        FPR
 0           5835              1592            4243  0.276814      50.8814
50.6046  0.375206
 1          11669              7281            4388   1.26601      52.6202
51.3542    1.6593
 2          17504             13059            4445   2.27068      53.3038
51.0331   2.93791
 3          23338             18844            4494   3.27656      53.8914
50.6148   4.19315
 4          29173             24635            4538   4.28349       54.419
50.1355    5.4286
 5          35007             30421            4586   5.28955      54.9946
49.7051   6.63345
 6          40842             36221            4621   6.29804      55.4143
49.1163   7.83835
 7          46676             42008            4668   7.30428      55.9779
48.6737   8.99914
 8          52511             47803            4708    8.3119      56.4576
48.1457   10.1536
 9          58345             53595            4750   9.31901      56.9613
47.6423   11.2832
 10         64180             59387            4793   10.3261      57.4769
47.1508   12.3904
 11         70014             65179            4835   11.3332      57.9806
46.6474   13.4807
 12         75849             70978            4871   12.3415      58.4123
46.0707   14.5715
 13         81684             76763            4921   13.3474      59.0119
45.6645   15.5991
 14         87518             82547            4971   14.3531      59.6115
45.2583   16.6057
 15         93353             88343            5010   15.3609      60.0791
44.7182   17.6333
 16         99187             94127            5060   16.3666      60.6787
44.3121   18.6022
 17        105022             99913            5109   17.3727      61.2663
43.8936   19.5563
 18        110856            105715            5141   18.3815      61.6501
43.2685   20.5631
 19        116691            111513            5178   19.3897      62.0938
42.7041   21.5359,
    # Records # Goods # Bads  % Goods    % Bads
 0       5835    1592   4243  27.2836   72.7164
 1       5834    5689    145  97.5146   2.48543
 2       5835    5778     57  99.0231  0.976864
 3       5834    5785     49  99.1601  0.839904
 4       5835    5791     44  99.2459   0.75407
 5       5834    5786     48  99.1772  0.822763
 6       5835    5800     35  99.4002  0.599829
 7       5834    5787     47  99.1944  0.805622
 8       5835    5795     40  99.3145  0.685518
 9       5834    5792     42  99.2801  0.719918
 10      5835    5792     43  99.2631  0.736932
 11      5834    5792     42  99.2801  0.719918
```

12	5835	5799	36	99.383	0.616967
13	5835	5785	50	99.1431	0.856898
14	5834	5784	50	99.143	0.857045
15	5835	5796	39	99.3316	0.66838
16	5834	5784	50	99.143	0.857045
17	5835	5786	49	99.1602	0.83976
18	5834	5802	32	99.4515	0.548509
19	5835	5798	37	99.3659	0.634105)

In [ ]:

```
# Testing Set
output_table(Y_test['Fraud Proba'],Y_test['fraud_label'])
```

```
no_of_records 250053
No of bads 3668
no_of_goods 246385
Bin statistics
```

	# Records	# Goods	# Bads	% Goods	% Bads
0	2501	604	1897	24.1503	75.8497
1	2500	2439	61	97.56	2.44
2	2501	2480	21	99.1603	0.839664
3	2500	2482	18	99.28	0.72
4	2501	2483	18	99.2803	0.719712
5	2500	2480	20	99.2	0.8
6	2501	2489	12	99.5202	0.479808
7	2500	2480	20	99.2	0.8
8	2501	2487	14	99.4402	0.559776
9	2500	2481	19	99.24	0.76
10	2501	2482	19	99.2403	0.759696
11	2500	2484	16	99.36	0.64
12	2501	2480	21	99.1603	0.839664
13	2500	2482	18	99.28	0.72
14	2501	2489	12	99.5202	0.479808
15	2500	2479	21	99.16	0.84
16	2501	2489	12	99.5202	0.479808
17	2501	2483	18	99.2803	0.719712
18	2500	2483	17	99.32	0.68
19	2501	2482	19	99.2403	0.759696

```
Cumulative results
```

	Total # Records	Cumulative Goods	Cumulative Bads	% Goods	% Bads (FDR)	KS	FPR
0	2501	604	1897	0.245145	51.7176	51.4724	0.318397
1	5001	3043	1958	1.23506	53.3806	52.1455	1.55414
2	7502	5523	1979	2.24161	53.9531	51.7115	2.7908
3	10002	8005	1997	3.24898	54.4438	51.1949	4.00851
4	12503	10488	2015	4.25675	54.9346	50.6778	5.20496
5	15003	12968	2035	5.26331	55.4798	50.2165	6.37248
6	17504	15457	2047	6.27352	55.807	49.5335	7.55105
7	20004	17937	2067	7.28007	56.3522	49.0722	8.67779
8	22505	20424	2081	8.28947	56.7339	48.4444	9.81451
9	25005	22905	2100	9.29643	57.2519	47.9555	10.9071
10	27506	25387	2119	10.3038	57.7699	47.4661	11.9807
11	30006	27871	2135	11.312	58.2061	46.8941	13.0543
12	32507	30351	2156	12.3185	58.7786	46.4601	14.0775
13	35007	32833	2174	13.3259	59.2694	45.9435	15.1026
14	37508	35322	2186	14.3361	59.5965	45.2604	16.1583
15	40008	37801	2207	15.3422	60.169	44.8268	17.1278
16	42509	40290	2219	16.3525	60.4962	44.1437	18.1568
17	45010	42773	2237	17.3602	60.9869	43.6267	19.1207
18	47510	45256	2254	18.368	61.4504	43.0824	20.0781
19	50011	47738	2273	19.3754	61.9684	42.593	21.0022

Out[ ]:

```
(   Total # Records Cumulative Goods Cumulative Bads   % Goods % Bads (FDR)
KS       FPR
 0           2501               604            1897  0.245145       51.7176
51.4724  0.318397
 1           5001              3043            1958   1.23506       53.3806
52.1455   1.55414
 2           7502              5523            1979   2.24161       53.9531
51.7115    2.7908
 3          10002              8005            1997   3.24898       54.4438
51.1949   4.00851
 4          12503             10488            2015   4.25675       54.9346
50.6778   5.20496
 5          15003             12968            2035   5.26331       55.4798
50.2165   6.37248
 6          17504             15457            2047   6.27352        55.807
49.5335   7.55105
 7          20004             17937            2067   7.28007       56.3522
49.0722   8.67779
 8          22505             20424            2081   8.28947       56.7339
48.4444   9.81451
 9          25005             22905            2100   9.29643       57.2519
47.9555   10.9071
 10         27506             25387            2119   10.3038       57.7699
47.4661   11.9807
 11         30006             27871            2135    11.312       58.2061
46.8941   13.0543
 12         32507             30351            2156   12.3185       58.7786
46.4601   14.0775
 13         35007             32833            2174   13.3259       59.2694
45.9435   15.1026
 14         37508             35322            2186   14.3361       59.5965
45.2604   16.1583
 15         40008             37801            2207   15.3422        60.169
44.8268   17.1278
 16         42509             40290            2219   16.3525       60.4962
44.1437   18.1568
 17         45010             42773            2237   17.3602       60.9869
43.6267   19.1207
 18         47510             45256            2254    18.368       61.4504
43.0824   20.0781
 19         50011             47738            2273   19.3754       61.9684
42.593   21.0022,
    # Records # Goods # Bads  % Goods    % Bads
 0       2501     604   1897  24.1503   75.8497
 1       2500    2439     61    97.56      2.44
 2       2501    2480     21  99.1603  0.839664
 3       2500    2482     18    99.28      0.72
 4       2501    2483     18  99.2803  0.719712
 5       2500    2480     20     99.2       0.8
 6       2501    2489     12  99.5202  0.479808
 7       2500    2480     20     99.2       0.8
 8       2501    2487     14  99.4402  0.559776
 9       2500    2481     19    99.24      0.76
 10      2501    2482     19  99.2403  0.759696
 11      2500    2484     16    99.36      0.64
```

12	2501	2480	21	99.1603	0.839664
13	2500	2482	18	99.28	0.72
14	2501	2489	12	99.5202	0.479808
15	2500	2479	21	99.16	0.84
16	2501	2489	12	99.5202	0.479808
17	2501	2483	18	99.2803	0.719712
18	2500	2483	17	99.32	0.68
19	2501	2482	19	99.2403	0.759696)

In [ ]:

```
# Validation Set
output_table(Y_oot['Fraud Proba'],Y_oot['fraud_label'])
```

```
no_of_records 166493
No of bads 2386
no_of_goods 164107
Bin statistics
```

	# Records	# Goods	# Bads	% Goods	% Bads
0	1665	492	1173	29.5495	70.4505
1	1665	1619	46	97.2372	2.76276
2	1665	1657	8	99.5195	0.48048
3	1665	1652	13	99.2192	0.780781
4	1665	1657	8	99.5195	0.48048
5	1665	1653	12	99.2793	0.720721
6	1665	1654	11	99.3393	0.660661
7	1664	1646	18	98.9183	1.08173
8	1665	1651	14	99.1592	0.840841
9	1665	1651	14	99.1592	0.840841
10	1665	1657	8	99.5195	0.48048
11	1665	1645	20	98.7988	1.2012
12	1665	1655	10	99.3994	0.600601
13	1665	1654	11	99.3393	0.660661
14	1665	1655	10	99.3994	0.600601
15	1665	1655	10	99.3994	0.600601
16	1665	1652	13	99.2192	0.780781
17	1665	1655	10	99.3994	0.600601
18	1665	1649	16	99.039	0.960961
19	1665	1658	7	99.5796	0.42042

```
Cumulative results
```

	Total # Records	Cumulative Goods	Cumulative Bads	% Goods	% Bads (FDR)	KS	FPR
0	1665	492	1173	0.299804	49.1618	48.862	0.419437
1	3330	2111	1219	1.28636	51.0897	49.8033	1.73175
2	4995	3768	1227	2.29606	51.425	49.1289	3.0709
3	6660	5420	1240	3.30272	51.9698	48.6671	4.37097
4	8325	7077	1248	4.31243	52.3051	47.9927	5.67067
5	9990	8730	1260	5.3197	52.808	47.4883	6.92857
6	11655	10384	1271	6.32758	53.2691	46.9415	8.16994
7	13319	12030	1289	7.33058	54.0235	46.6929	9.33282
8	14984	13681	1303	8.33663	54.6102	46.2736	10.4996
9	16649	15332	1317	9.34268	55.197	45.8543	11.6416
10	18314	16989	1325	10.3524	55.5323	45.1799	12.8219
11	19979	18634	1345	11.3548	56.3705	45.0157	13.8543
12	21644	20289	1355	12.3633	56.7896	44.4263	14.9734
13	23309	21943	1366	13.3712	57.2506	43.8795	16.0637
14	24974	23598	1376	14.3796	57.6697	43.2901	17.1497
15	26639	25253	1386	15.3881	58.0889	42.7007	18.2201
16	28304	26905	1399	16.3948	58.6337	42.2389	19.2316
17	29969	28560	1409	17.4033	59.0528	41.6495	20.2697
18	31634	30209	1425	18.4081	59.7234	41.3153	21.1993
19	33299	31867	1432	19.4184	60.0168	40.5983	22.2535

Out[ ]:

```
(   Total # Records Cumulative Goods Cumulative Bads   % Goods % Bads (FDR)
KS        FPR
 0            1665               492            1173  0.299804      49.1618
48.862  0.419437
 1            3330              2111            1219   1.28636      51.0897
49.8033   1.73175
 2            4995              3768            1227   2.29606       51.425
49.1289    3.0709
 3            6660              5420            1240   3.30272      51.9698
48.6671   4.37097
 4            8325              7077            1248   4.31243      52.3051
47.9927   5.67067
 5            9990              8730            1260    5.3197       52.808
47.4883   6.92857
 6           11655             10384            1271   6.32758      53.2691
46.9415   8.16994
 7           13319             12030            1289   7.33058      54.0235
46.6929   9.33282
 8           14984             13681            1303   8.33663      54.6102
46.2736   10.4996
 9           16649             15332            1317   9.34268       55.197
45.8543   11.6416
 10          18314             16989            1325   10.3524      55.5323
45.1799   12.8219
 11          19979             18634            1345   11.3548      56.3705
45.0157   13.8543
 12          21644             20289            1355   12.3633      56.7896
44.4263   14.9734
 13          23309             21943            1366   13.3712      57.2506
43.8795   16.0637
 14          24974             23598            1376   14.3796      57.6697
43.2901   17.1497
 15          26639             25253            1386   15.3881      58.0889
42.7007   18.2201
 16          28304             26905            1399   16.3948      58.6337
42.2389   19.2316
 17          29969             28560            1409   17.4033      59.0528
41.6495   20.2697
 18          31634             30209            1425   18.4081      59.7234
41.3153   21.1993
 19          33299             31867            1432   19.4184      60.0168
40.5983   22.2535,
    # Records # Goods # Bads  % Goods    % Bads
 0       1665     492   1173  29.5495   70.4505
 1       1665    1619     46  97.2372   2.76276
 2       1665    1657      8  99.5195   0.48048
 3       1665    1652     13  99.2192  0.780781
 4       1665    1657      8  99.5195   0.48048
 5       1665    1653     12  99.2793  0.720721
 6       1665    1654     11  99.3393  0.660661
 7       1664    1646     18  98.9183   1.08173
 8       1665    1651     14  99.1592  0.840841
 9       1665    1651     14  99.1592  0.840841
 10      1665    1657      8  99.5195   0.48048
 11      1665    1645     20  98.7988    1.2012
```

12	1665	1655	10	99.3994	0.600601
13	1665	1654	11	99.3393	0.660661
14	1665	1655	10	99.3994	0.600601
15	1665	1655	10	99.3994	0.600601
16	1665	1652	13	99.2192	0.780781
17	1665	1655	10	99.3994	0.600601
18	1665	1649	16	99.039	0.960961
19	1665	1658	7	99.5796	0.42042)