Project 1 Report

Mao-Ho Wang

Executive summary:

The aim of this project is to develop a machine learning model to detect fraudulent transactions and estimate the potential savings from deploying the model. A False Discovery Rate of 3% will be used for out-of-time transactions to evaluate the performance of the models and select the best one. The estimated savings from deploying the model will be calculated based on the reduction in fraudulent transactions detected and the associated financial losses prevented.

Description of the data:

The dataset is Credit Card Transaction Data, which contains Transaction and Company Identifying Information of credit cards. The data came from a U.S. government organization. There are 10 fields and 96,753 records.

Numeric Fields Table

Field Name	# Records With Values	% Populated	# Zeros	Min	Max	Mean	Most Common	Stdev
Date	96,753	100.00%	0	1/1/2010	12/31/2010	6/25/2010	2/28/2010	98 days 21:38:58
Amount	96,753	100.00%	0	0.01	3,102,045.53	427.885677	3.62	10,006.14

Categorical Fields Table

Field Name	# Records With Values	% Populated	# Zeros	# Unique Values	Most Common
Recnum	96,753	100.00%	0	96,753	1
Cardnum	96,753	100.00%	0	1,645	5142148452
Merchnum	93,378	96.51%	0	13,091	930090121224
Merch description	96,753	100.00%	0	13,126	GSA-FSS-ADV
Merch state	95,558	98.76%	0	227	TN
Merch zip	92,097	95.19%	0	4,567	38118
Transtype	96,753	100.00%	0	4	P
Fraud	96,753	100.00%	95,694	2	0

Data cleaning:

Cleaning and imputation:

I first drop the rows that contain null value in all columns.

```
data.dropna(how='all', axis=1, inplace=True)
data['Date'] = pd.to_datetime(data['Date'])
data.info()
```

I further filter Accounts that are not in the appropriate condition for the prediction (transtype that are not 'P') and the outlier I found from HW1 (Transaction amount that is higher than 3,000,000).

```
data = data[data['Transtype'] == 'P']
data = data[data['Amount'] <= 3000000]
data.shape
```

I convert the merchant number that equals to 0 into null value as merchant number cannot be 0.

```
data['Merchnum'] = data['Merchnum'].replace({'0':np.nan})
for index, zip5 in data[data['Merch zip'].notnull()]['Merch zip'].items():
    if zip5 not in zip_state:
        zip_state[zip5] = data.loc[index, 'Merch state']

zip_state['00926'] = 'PR'
zip_state['00929'] = 'PR'
zip_state['00934'] = 'PR'
zip_state['00902'] = 'PR'
zip_state['00738'] = 'PR'
zip_state['90805'] = 'CA'
zip_state['76302'] = 'TX'
zip_state['00914'] = 'PR'
zip_state['95461'] = 'CA'
zip_state['00680'] = 'PR'
zip_state['00623'] = 'PR'
zip_state['00726'] = 'PR'
zip_state['00936'] = 'PR'
zip_state['12108'] = 'NY'
zip_state['00791'] = 'PR'
zip_state['00907'] = 'PR'
zip_state['00922'] = 'PR'
zip_state['00920'] = 'PR'
zip_state['00801'] = 'VI'
zip_state['31040'] = 'GA'
zip_state['41160'] = 'KY'
zip_state['00681'] = 'PR'
```

As same merchant description should share the merchant number, I used those rows that have merchant number to fill in those who don't have but share the same merchant description.

```
merchdes_merchnum = {}
for index, merchdes in data[data['Merch description'].notnull()][data['Merchnum'].notnull()]['Merch description'].items():
    if pd.isnull(merchdes) == True:
        continue
    elif merchdes not in merchdes_merchnum:
        merchdes_merchnum[merchdes] = data.loc[index, 'Merchnum']
```

```
# fill in by mapping with Merch description
data['Merchnum'] = data['Merchnum'].fillna(data['Merch description'].map(merchdes_merchnum))
```

Since there are still 508 rows that have null value on Merchnum, I assign them a new merchnum value, which is the max merchant number plus one.

```
# adding new merchnums
# each new unique merchnum will be max(merchnum) + 1
merchnum_create = {}
max_merchnum = pd.to_numeric(data.Merchnum, errors='coerce').max()
for merch_desc in data.loc[data.Merchnum.isna(), 'Merch description'].unique():
    merchnum_create[merch_desc] = str(int(max_merchnum + 1))
    max_merchnum += 1
```

```
# fill in by mapping with Merch description (newly created merchnums)
data['Merchnum'] = data['Merchnum'].fillna(data['Merch description'].map(merchnum_create))
```

After cleaning and impute merchant number, I started cleaning and impute merchant state. I first created three dictionaries (zip_state, merchum_state, merchdes_state) that contain what zip code, merchant number, and merchant description correspond to what state. For zip_state dictionary, Ialso input zip codes that are not in the "Merch zip" with corresponding state.

```
merchdes_state = {}
for index, merchdes in data[data['Merch description'].notnull()]['Merch description'].items():
    if merchdes not in merchdes_state :
        merchdes_state [merchdes] = data.loc[index, 'Merch state']
```

After creating dictionaries, I use them to help me change null value to correct value within “Merch state” by matching their zip code, merchant number, and merchant description.

```
# fill in by mapping with zip, merchnum and merch description
data['Merch state'] = data['Merch state'].fillna(data['Merch zip'].map(zip_state))
data['Merch state'] = data['Merch state'].fillna(data['Merchnum'].map(merchnum_state))
data['Merch state'] = data['Merch state'].fillna(data['Merch description'].map(merchdes_state))
```

I further assign adjustments transactions records’ merchant state with “unknown” value since they are used to correct errors in financial records.

```
# assign unknown for adjustments transactions
data['Merch state'] = data['Merch state'].mask(data['Merch description'] == 'RETAIL CREDIT ADJUSTMENT', 'unknown')
data['Merch state'] = data['Merch state'].mask(data['Merch description'] == 'RETAIL DEBIT ADJUSTMENT', 'unknown')
```

Finally, I put “foreign” tag to records that are not in U.S, and “unknown” to the remaining records.

```
# change non-US states
# might actually be useful cus fraud could be foreign transactions
# maybe put a 'foreign' tag or just leave them as is

states = ["AL", "AK", "AZ", "AR", "CA", "CO", "CT", "DC", "DE", "FL", "GA",
          "HI", "ID", "IL", "IN", "IA", "KS", "KY", "LA", "ME", "MD",
          "MA", "MI", "MN", "MS", "MO", "MT", "NE", "NV", "NH", "NJ",
          "NM", "NY", "NC", "ND", "OH", "OK", "OR", "PA", "RI", "SC",
          "SD", "TN", "TX", "UT", "VT", "VA", "WA", "WV", "WI", "WY",
          'VI', 'PR', np.nan, 'unknown']

for index, state in data['Merch state'].items():
    if state not in states:
        data.loc[index, 'Merch state'] = 'foreign'
```

```
data['Merch state'].fillna('unknown',inplace=True)
```

Next, I started to clean and impute zip column. I also created two dictionaries (merchum_zip, merchdes_zip) that contain what merchant number and merchant description correspond to what zip code.

```
merchnum_zip = {}
for index, merchnum in data[data['Merchnum'].notnull()]['Merchnum'].items():
    if merchnum not in merchnum_zip :
        merchnum_zip [merchnum] = data.loc[index, 'Merch zip']
```

```
merchdes_zip = {}
for index, merchdes in data[data['Merch description'].notnull()]['Merch description'].items():
    if merchdes not in merchdes_zip :
        merchdes_zip [merchdes] = data.loc[index, 'Merch zip']
```

After creating dictionaries, I use them to help me change null value to correct value within “Merch zip” by matching their merchant number and merchant description.

```
# fill in by mapping with merchnum and merch description
data['Merch zip'] = data['Merch zip'].fillna(data['Merchnum'].map(merchnum_zip))
data['Merch zip'] = data['Merch zip'].fillna(data['Merch description'].map(merchdes_zip))
```

Still, I assign adjustments transactions records’ merchant zip code with “unknown” value since they are used to correct errors in financial records.

```
# assign unknown for adjustments transactions
data['Merch zip'] = data['Merch zip'].mask(data['Merch zip'] == 'RETAIL CREDIT ADJUSTMENT', 'unknown')
data['Merch zip'] = data['Merch zip'].mask(data['Merch zip'] == 'RETAIL DEBIT ADJUSTMENT', 'unknown')
```

Finally, put “unknown” tag to the remaining records.

```
data['Merch zip'].fillna('unknown', inplace=True)
data['Merch zip'].isnull().sum()
```

After cleaning data, I started to deal with categorical columns. Here I used target encoding instead of one-hot encoding since I don’t want to generate too many fields. I first create a new column called “Dow” to get the day of the week of each transaction.

```
## find the day of the week
df['Dow'] = df.Date.apply(lambda x: calendar.day_name[x.weekday()])
```

To conduct target encoding, I created a new dataframe called train_test to separate the training set. Later I calculated the fraud probability of each day of the week and apply smoothing formula to get the target encoding value for DOW.

```
## we want to not use the oot for target encoding variables
train_test = df[df.Date < '2010-11-01']
c = 4; nmid = 20; y_avg = train_test['Fraud'].mean()
y_dow = train_test.groupby('Dow')['Fraud'].mean()
num = train_test.groupby('Dow').size()
y_dow_smooth = y_avg + (y_dow - y_avg)/(1 + np.exp(-(num - nmid)/c))
df['Dow_Risk'] = df.Dow.map(y_dow_smooth)
```

I also apply target encoding to the state variable as using one hot encoding could result in many new fields.

```
# statistical smoothing
c = 4
nmid = 20
y_avg = train_test['Fraud'].mean()
y_state = train_test.groupby('Merch state')['Fraud'].mean()
num = train_test.groupby('Merch state').size()
y_state_smooth = y_avg + (y_state - y_avg)/(1 + np.exp(-(num-nmid)/c))
df['state_risk'] = df['Merch state'].map(y_state_smooth)
```

After finishing encoding, I change the following columns from integer to string.

```
df['Cardnum'] = df['Cardnum'].apply(str)
df['Merchnum'] = df['Merchnum'].apply(str)
df['Merch zip'] = df['Merch zip'].apply(str)
```

Since zip codes were previously integer, I add leading 0 to the zip code to get accurate format. I also delete white spaces in the “Merch description” column.

```
### add leading 0 to zips
### note: there are some zips that are state abbrv. as we imputed them ealier, so pandas read the column as str

def leading_0(x):

    if '.0' in x:
        x = x[:-2]
        if len(x) == 5:
            return x
        else:
            return '0'*(5-len(x)) + x
    else:
        return '0'*(5-len(x)) + x

# df['Merch zip'] = df['Merch zip'].apply(leading_0)
```

```
### delete white spaces in merch description
df['Merch description'] = df['Merch description'].str.replace(r'\s', '')
```

Variable creation:

After doing data cleaning and encoding, I started to create several entities.

```
df['card_merch'] = df['Cardnum'] + df['Merchnum']
df['card_zip'] = df['Cardnum'] + df['Merch zip']
df['card_state'] = df['Cardnum'] + df['Merch state']
df['merch_zip'] = df['Merchnum'] + df['Merch zip']
df['merch_state'] = df['Merchnum'] + df['Merch state']
df['state_des'] = df['Merch state'] + df['Merch description']

# these next entity take a long time to calculate the variables for, and I don't know why
# df['state_zip'] = df['Merch state'] + df['Merch zip']

df['zip3'] = df['Merch zip'].str[:3]
df['card_zip3'] = df.Cardnum + df['zip3']
# df['merchnum_zip'] = df.Merchnum + df['Merch zip']
# df['merchnum_zip3'] = df.Merchnum + df['zip3']
df['Card_Merchdesc'] = df['Cardnum'] + df['Merch description']
df['Card_dow'] = df['Cardnum'] + df['Dow']
df['Merchnum_desc'] = df['Merchnum'] + df['Merch description']
df['Merchnum_dow'] = df['Merchnum'] + df['Dow']
# df['Merchdesc_State'] = df['Merch description'] + df['Merch state']
# df['Merchdesc_Zip'] = df['Merch description'] + df['Merch zip']
df['Merchdesc_dow'] = df['Merch description'] + df['Dow']
df['Card_Merchnum_desc'] = df['Cardnum'] + df['Merchnum'] + df['Merch description']
# df['Card_Merchnum_State'] = df['Cardnum'] + df['Merchnum'] + df['Merch state']
df['Card_Merchnum_Zip'] = df['Cardnum'] + df['Merchnum'] + df['Merch zip']
# df['Card_Merchdesc_State'] = df['Cardnum'] + df['Merch description'] + df['Merch state']
df['Card_Merchdesc_Zip'] = df['Cardnum'] + df['Merch description'] + df['Merch zip']
df['Merchnum_desc_State'] = df['Merchnum'] + df['Merch description'] + df['Merch state']
# df['Merchnum_desc_Zip'] = df['Merchnum'] + df['Merch description'] + df['Merch zip']
```

I also create new fields using Benford's law. I first apply it to the transaction amount.

```
# another way to get the first digit
bf = data.copy()
bf['amount_100'] = (bf['Amount'] * 100).astype(str)
bf['first_digit'] = bf['amount_100'].str[0]
bf['first_digit'].value_counts()
```

Since for the transactions whose merchant is FEDEX, the first digit of transaction amount is very likely to be 3, I filtered these records to make sure Benford's law can be applied correctly.

```
dropfedex = bf[bf['Merch description'].str.contains('FEDEX')]
```

```
droplist = dropfedex.index.tolist()
droplist[:10]
```

```
bf1 = bf.drop(droplist)
bf1.shape
```

Later, I created a new column called bin and assigned records whose first digit of transaction amount is 1 or 2 to "low" else "high".

```
bf1['bin']= bf1['first_digit'].apply(lambda x: "low" if x == "1" else ("low" if x == "2" else "high"))
bf1.head(5)
```

I further calculated the number of low bin and high bin for each card number. To make it calculable, I set the null value to 1 if card numbers don't have either low bin or high bin.

```
# calculating n_low and n_high
card_bf =  bf1.groupby(['Cardnum','bin']).agg({'bin': ['count']}).reset_index()
card_bf.columns=['Cardnum', 'bin', 'count']
card bf
```

```
card_bf =  card_bf.pivot_table(index='Cardnum',columns='bin',values='count',aggfunc='sum').reset_index()
card_bf.columns=['Cardnum', 'n_high', 'n_low']
card_bf
```

```
# if either n_low or n_high is zero, set it to 1
card_bf = card_bf.fillna(1)
card_bf
```

Finally, I apply Benford's equation to each card number.

```
# calclating R, 1/R, U, n, t U_smoothed
c=3
n_mid=15
card_bf['R'] = (1.096 * card_bf['n_low']/card_bf['n_high'])
card_bf['1/R'] = (1/card_bf['R'])
card_bf['U'] = list(map(lambda x, y : max(x,y),card_bf['R'],card_bf['1/R']))
card_bf['n'] = card_bf['n_high'] + card_bf['n_low']
card_bf['t'] = ((card_bf['n']-n_mid)/c)
card_bf['U_smoothed']= list(map(lambda x, y : (1 + (x-1)/(1+exp(-y))),card_bf['U'],card_bf['t']))
```

I also apply Benford's equation to each merchant number.

```
# calculating n_low and n_high
merch_bf =  bf1.groupby(['Merchnum','bin']).agg({'bin': ['count']}).reset_index()
merch_bf.columns=['Merchnum', 'bin', 'count']
merch_bf =  merch_bf.pivot_table(index='Merchnum',columns='bin',values='count',aggfunc='sum').reset_index()
merch_bf.columns=['Merchnum', 'n_high', 'n_low']
merch_bf.head()
```

```
# if either n_low or n_high is zero, set it to 1
merch_bf = merch_bf.fillna(1)
merch_bf
```

```
# calclating R, 1/R, U, n, t U_smoothed
merch_bf['R'] = (1.096 * merch_bf['n_low']/merch_bf['n_high'])
merch_bf['1/R'] = (1/merch_bf['R'])
merch_bf['U'] = list(map(lambda x, y : max(x,y),merch_bf['R'],merch_bf['1/R']))
merch_bf['n'] = merch_bf['n_high'] + merch_bf['n_low']
merch_bf['t'] = ((merch_bf['n']-n_mid)/c)
merch_bf['U_smoothed']= list(map(lambda x, y : (1 + (x-1)/(1+exp(-y))),merch_bf['U'],merch_bf['t']))
```

I further combine these two columns I created from Benford's equation to the previous dataframe and fill in the null value. However, since Benford's Law is not useful in this case, I decided to drop these two columns.

```
card_bf['Cardnum'] = card_bf['Cardnum'].apply(str)
merch_bf['Merchnum'] = merch_bf['Merchnum'].apply(str)
card_bf.info()
```

```
card_bf.set_index('Cardnum',inplace=True)
```

```
card_Ustar = pd.DataFrame(card_bf['U_smoothed'])
card_Ustar.sort_values(['U_smoothed'], ascending = False).head(10)
```

```
merch_bf.set_index('Merchnum',inplace=True)
```

```
merch_Ustar = pd.DataFrame(merch_bf['U_smoothed'])
merch_Ustar.sort_values(['U_smoothed'], ascending = False).head(10)
```

```
final = final.merge(card_Ustar, how = 'left',left_on='Cardnum',right_on=card_Ustar.index)
final = final.rename(columns={'U_smoothed':'U*_cardnum'})
final = final.merge(merch_Ustar, how = 'left',left_on='Merchnum',right_on=merch_Ustar.index)
final = final.rename(columns={'U_smoothed':'U*_merchnum'})
```

```
final['U*_cardnum'].fillna(1,inplace=True)
final['U*_merchnum'].fillna(1,inplace=True)
```

```
print(final.shape)
final.drop(columns=['U*_cardnum','U*_merchnum'],inplace=True)
print(final.shape)
```

Later, I tried to make as many entities as could be useful. But to avoid time and memory issues. I remove some entities while creating new entities.

```
# If you want, remove some entities that take a long time
# these take a long time and don't add much
entities.remove('state_risk')
entities.remove('zip3')
# entities.remove('Merchnum_desc_State')
entities.remove('Card_Merchdesc_Zip')
# entities.remove('Card_Merchnum_desc')
entities.remove('Card_Merchnum_Zip')
# entities.remove('Merchdesc_dow')
# entities.remove('Merchnum_dow')
# entities.remove('Merchnum_desc')
# entities.remove('Card_dow')
# entities.remove('Card_Merchdesc')
entities.remove('card_zip3')
```

I further expand the entities amount by using number of days since an application with that entity was seen, frequency, and amount.

```
%%time
start = timeit.default_timer()
for entity in entities:
    try: print(entity,'Run time for the this entity ---------------- {}s'.format(timeit.default_timer() - st))
    except: print('')
    st = timeit.default_timer()

# Day-since variables:
    df_l = df1[['Recnum', 'Date', entity]]
    df_r = df1[['check_record', 'check_date', entity, 'Amount']]
    temp = pd.merge(df_l, df_r, left_on = entity, right_on = entity)
    temp1 = temp[temp.Recnum > temp.check_record][['Recnum','Date','check_date']]\
                                                .groupby('Recnum')[['Date', 'check_date']].last()
    mapper = (temp1.Date - temp1.check_date).dt.days
    final[entity + '_day_since'] = final.Recnum.map(mapper)
    final[entity + '_day_since'].fillna((final.Date - pd.to_datetime('2006-01-01')).dt.days, inplace = True)
    print('\n' + entity + '_day_since ---> Done')

# Frequency & Amount variables:
    for time in [0,1,3,7,14,30,45,60]:
        temp2 = temp[(temp.check_date >= (temp.Date - dt.timedelta(time))) &\
                     (temp.Recnum >= temp.check_record)][['Recnum', entity, 'Amount']]
        col_name = entity + '_count_' + str(time)
        mapper2 = temp2.groupby('Recnum')[entity].count()
        final[col_name] = final.Recnum.map(mapper2)
        print(col_name + ' ---> Done')
        final[entity + '_avg_' + str(time)] = final.Recnum.map(temp2.groupby('Recnum')['Amount'].mean())

        final[entity + '_std_' + str(time)] = final.Recnum.map(temp2.groupby('Recnum')['Amount'].std())

        final[entity + '_max_' + str(time)] = final.Recnum.map(temp2.groupby('Recnum')['Amount'].max())
        final[entity + '_med_' + str(time)] = final.Recnum.map(temp2.groupby('Recnum')['Amount'].median())
        final[entity + '_total_' + str(time)] = final.Recnum.map(temp2.groupby('Recnum')['Amount'].sum())
        final[entity + '_actual/avg_' + str(time)] = final['Amount'] / final[entity + '_avg_' + str(time)]

        final[entity + '_actual/std_' + str(time)] = final['Amount'] / final[entity + '_std_' + str(time)]

        final[entity + '_actual/max_' + str(time)] = final['Amount'] / final[entity + '_max_' + str(time)]
        final[entity + '_actual/med_' + str(time)] = final['Amount'] / final[entity + '_med_' + str(time)]
        final[entity + '_actual/toal_' + str(time)] = final['Amount'] / final[entity + '_total_' + str(time)]
        print(entity + ' amount variables over past ' + str(time) + ' ---> Done')
    del df_l
    del df_r
    del temp
    del temp1
    del temp2
    del mapper2
```

Relative velocity

```
%%time
start = timeit.default_timer()
for ent in entities:
    for d in ['0', '1']:
        for dd in ['7', '14', '30', '60']:
            final[ent + '_count_' + d + '_by_' + dd] =\
            final[ent + '_count_' + d]/(final[ent + '_count_' + dd])/float(dd)
            final[ent + '_total_amount_'+d+'_by_' + dd]=\
            final[ent +'_total_'+d]/(final[ent+'_total_'+dd])/float(dd)

print('run time: {}s'.format(timeit.default_timer() - start))
```

```
start = timeit.default_timer()
for ent in entities:
    for d in ['0', '1']:
        for dd in ['7', '14', '30', '60']:
            final[ent + '_vdratio_' + d +'by' + dd] =\
            final[ent + '_count_' + d + '_by_' + dd]/(final[ent + '_day_since']+1

print('run time: {}s'.format(timeit.default_timer() - start))
```

Difference of transaction amount of the same entity

```
start = timeit.default_timer()
for entity in entities:
    try: print('Run time for the last entity ---------------- {}s'.format(timeit.default_timer() - st))
    except:
        print('')
    st = timeit.default_timer()
    df_l = df1[['Recnum', 'Date', entity,'Amount']]
    df_r = df1[['check_record', 'check_date', entity, 'Amount']]
    temp = pd.merge(df_l, df_r, left_on = entity, right_on = entity)

    for time in [0,1,3,7,14,30]:
        temp2 = temp[(temp.check_date >= (temp.Date - dt.timedelta(time))) &\
                     (temp.Recnum >= temp.check_record)][['Recnum', 'check_record',entity, 'Amount_x','Amount_y']]
        temp2['Amount_diff']=temp2['Amount_y']-temp2['Amount_x']

        col_name = entity + '_variability_avg_' + str(time)
        mapper2 = temp2.groupby('Recnum')['Amount_diff'].mean()
        final[col_name] = final.Recnum.map(mapper2)
        print(col_name + ' ---> Done')

        col_name = entity + '_variability_max_' + str(time)
        mapper2 = temp2.groupby('Recnum')['Amount_diff'].max()
        final[col_name] = final.Recnum.map(mapper2)
        print(col_name + ' ---> Done')

        col_name = entity + '_variability_med_' + str(time)
        mapper2 = temp2.groupby('Recnum')['Amount_diff'].median()
        final[col_name] = final.Recnum.map(mapper2)
        print(col_name + ' ---> Done')

        print(entity + ' amount variables over past ' + str(time) + ' ---> Done')
    del df_l
    del df_r
    del temp
    del temp2
```

Number of unique values within an entity and specific time range when another entity is fixed.

```
%%time
# this cell can take a long time.
start = timeit.default_timer()
for i in entities:
    for v in entities:
        if i==v:
            continue
        else:
            df_c=df1[['Recnum','Date',i]]
            df_d=df1[['check_record','check_date',i,v]]
            temp=pd.merge(df_c,df_d,left_on=i,right_on=i)

        for t in [1,3,7,14,30,60]:
            count_day_df=temp[(temp.check_date>=(temp.Date-dt.timedelta(t)))&(temp.Recnum>=temp.check_record)]
            col_name=f'{i}_unique_count_for_{v}_{t}'
            mapper=count_day_df.groupby(['Recnum'])[v].nunique()
            final[col_name]=final.Recnum.map(mapper)

print('Total run time: {}mins'.format((timeit.default_timer() - start)/60))
```

Relative velocity with square

```
start = timeit.default_timer()
for ent in entities:
    print(ent)
    for d in ['0', '1']:
        for dd in ['7', '14', '30', '60']:
            final[ent + '_count_' + d + '_by_' + dd + "_sq"] =\
            final[ent + '_count_' + d]/(final[ent + '_count_' + dd])/pow(float(dd),2)
print('run time: {}s'.format(timeit.default_timer() - start))
```

I create a new column that divide the Transaction amount into five equal-sized bins

```
# Amount bins
AMOUNT = True
if AMOUNT:
      final['amount_cat'] = pd.qcut(final.Amount, q=5,labels=[1,2,3,4,5])

      final['amount_cat'].value_counts().plot(kind='barh')
      plt.show()

      qcut_series, qcut_intervals = pd.qcut(final.Amount, q=5,labels=[1,2,3,4,5],retbins=True)

      qcut_series.value_counts()
```

I used US zipcode to create a new column called foreign that shows whether the transaction are fromforeign countries or not.

```
mapping = list(map(lambda x: x not in zip_state, final['Merch zip']))
final = pd.concat([final, pd.DataFrame({'foreign': mapping})], axis = 1)
```

If there are null value within the dataframe, I filled in will zero.

```
final.fillna(0,inplace=True)
```

Finally, I dropped some redundant columns.

```
%%time
# if the kernel dies in this cell it's likely due to memory problems.
# In that case, just write out the data file as is and you can read it in another notebook that just does deduping
print(final.shape)
final = final.T.drop_duplicates().T
final.shape
```

```
# careful about this line. Modify it so you only keep the variables (including the record # and dependent variable)
final_vars = final.iloc[:, np.r_[8, 10, 11, len(entities)+10:len(final.columns)]]
```

Feature selection:

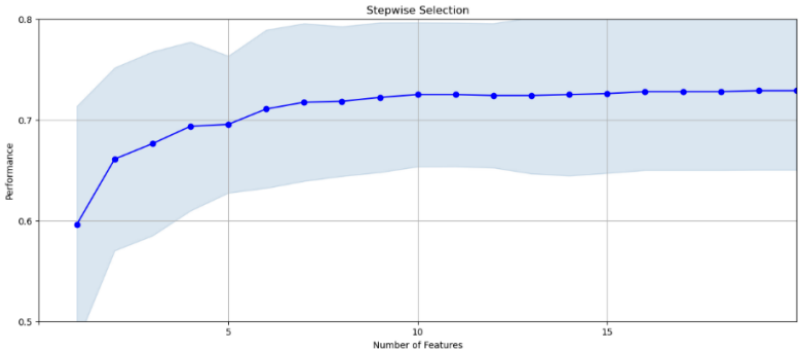
I further conducted feature selection by applying filter and wrapper techniques to choose 20 final variables for the model. For the filter, I set the number to 300 to get the best 300 variables (11%) that have a good score for KolmogorovSmirnov test.

	variable	filter score
0	Fraud	1.000000
1	card_zip3_total_7	0.676549
2	card_zip_total_7	0.666816
3	card_zip3_total_3	0.660260
4	card_zip3_total_14	0.659257
5	card_state_total_7	0.654135
6	card_zip_total_14	0.652244
7	card_zip_total_3	0.652217
8	card_state_total_3	0.644602
9	card_state_total_14	0.641512
10	card_merch_total_7	0.637702
11	card_zip_total_30	0.637171
12	Card_Merchnum_Zip_total_7	0.635141
13	card_zip3_max_7	0.630957
14	card_merch_total_3	0.630782

Picture below shows 10 variables that have the lowest KS score.

	variable	filter score
2334	Merchnum_desc_unique_count_for_merch_state_1	0.000065
2335	Card_Merchnum_desc_unique_count_for_card_state_7	0.000043
2336	Card_Merchdesc_Zip_unique_count_for_card_state_7	0.000032
2337	Card_Merchnum_Zip_unique_count_for_card_state_30	0.000032
2338	Card_Merchnum_desc_unique_count_for_card_state_3	0.000022
2339	merch_zip_unique_count_for_merch_state_7	0.000022
2340	Card_Merchdesc_Zip_unique_count_for_card_state_1	0.000022
2341	card_zip3_unique_count_for_card_state_1	0.000017
2342	card_merch_unique_count_for_card_state_1	0.000011
2343	card_merch_unique_count_for_Cardnum_1	0.000000

After getting top 300 variables, I used wrapper to further narrow it down the variables. I used forward stepwise selection and LGBM (n_estimators=40, num_leaves=4, cv=4) for the wrapper. Below is the performance plot for the wrapper.



I noticed that the number of features has similar performance after 10 variables. I decided to be more conservative and keep twice as many variables as the saturation indicates. Therefore, I selected 20 variables for the final model. Picture below shows the chosen 20 variables sorted by the wrapper along with their univariate KS.

	wrapper order	variable	filter score
0	1	card_merch_total_14	0.630048
1	2	card_zip3_max_14	0.629515
2	3	zip3_actual/avg_60	0.511141
3	4	Card_Merchdesc_med_7	0.489783
4	5	Cardnum_total_14	0.534929
5	6	card_zip_total_1	0.610773
6	7	card_merch_total_0	0.548902
7	8	card_state_total_60	0.559189
8	9	card_merch_med_7	0.496411
9	10	Card_Merchnum_Zip_med_7	0.495467
10	11	Card_Merchnum_desc_avg_1	0.511187
11	12	Card_Merchdesc_Zip_total_60	0.592687
12	13	card_state_med_7	0.492601
13	14	card_merch_avg_3	0.525502
14	15	Card_Merchdesc_avg_1	0.514152
15	16	Card_Merchdesc_med_0	0.490874
16	17	Card_Merchnum_Zip_avg_3	0.524558
17	18	Card_Merchnum_desc_avg_3	0.520716
18	19	Card_Merchnum_desc_med_1	0.498893
19	20	card_merch_avg_1	0.515008

Preliminary model explores:

Model	Parameter							Average FDR at 3%		
Logistic Regressior	Number of variables	penalty	C	Solver	l1_ratio			Train	Test	OOT
1	5	l2	1	lbfgs	None			0.65503	0.66218	0.43017
2	10	l2	1	lbfgs	None			0.66583	0.66935	0.34693
3	15	l2	1	lbfgs	None			0.67254	0.66217	0.34749
4	20	l2	1	lbfgs	None			0.67176	0.66307	0.35643
5	10	l1	1	saga	None			0.6728	0.65473	0.35643
6	15	l1	1	saga	None			0.667	0.67524	0.33464
7	20	l1	1	saga	None			0.67214	0.66199	0.36201
8	10	l2	0.5	lbfgs	None			0.67055	0.66078	0.3514
9	15	l2	0.75	lbfgs	None			0.68009	0.64555	0.3352
10	20	l2	0.1	lbfgs	None			0.67201	0.6654	0.36145
Decision Tree	Number of variables	criterion	Splitter	max_depth	min_samples_spli	min_samples_leaf		Train	Test	OOT
1	15	gini	best	None	2	1		1	0.6062	0.26369
2	15	gini	random	None	15	80		0.6483	0.60167	0.32346
3	15	gini	best	20	30	160		0.76126	0.70481	0.45698
4	20	gini	best	None	2	1		1	0.62128	0.24246
5	20	gini	random	None	20	100		0.69548	0.67674	0.3648
6	20	gini	best	20	40	200		0.73418	0.69677	0.46648
Random Forest	Number of variables	n_estimators	criterion	max_depth	min_samples_spli	min_samples_leaf		Train	Test	OOT
1	15	50	gini	None	2	1		1	0.81488	0.45251
2	15	100	entropy	20	100	30		0.88042	0.79715	0.50391
3	15	150	gini	20	300	50		0.81534	0.77707	0.48045
4	20	50	gini	None	2	1		1	0.80721	0.44693
5	20	100	gini	20	150	30		0.85109	0.7834	0.4838
6	20	150	entropy	20	450	50		0.79491	0.76254	0.50838
LGBM	Number of variables	num_leaves	max_dept	learning ra	n_estimators			Train	Test	OOT
1	15	31	-1	0.1	100			0.99984	0.80486	0.4
2	15	100	3	0.1	200			0.92893	0.81121	0.49218
3	20	31	-1	0.1	100			1	0.80378	0.39721
4	20	100	2	0.01	500			0.77107	0.74391	0.50615
5	20	150	3	0.1	300			0.95493	0.8313	0.48939

Neural Network	Number of variables	hidden_layer_sizes	activatior	alpha	learning rate	learning rate_init	max_iter	Train	Test	OOT
1	15	(150,)	relu	0.0001	adaptive	0.01	50	0.81127	0.76537	0.43911
2	15	(200,)	relu	0.001	constant	0.005	300	0.8246	0.77478	0.45196
3	20	(100,)	relu	0.0001	constant	0.001	200	0.83418	0.76827	0.48939
4	20	(200,)	relu	0.001	adaptive	0.01	100	0.80407	0.76878	0.43855
5	20	(500,)	relu	0.0001	constant	0.005	500	0.87114	0.80269	0.42737

Final model performance:

Considering overfitting problem and OOT performance, I choose Random Forest with 20 variables, 150 num_leaves, entropy criterion, 20 max_depth, 450 min_samples_split, 50 min_samples_leaf as the final model. Tables below show the result on training, testing, and OOT set.

Training set:

	bin	#recs	#g	#b	%g	%b	tot	cg	cb	%cg	FDR	KS	FPR	Fraud Savings	FP Loss	Overall Savings
1	1.0	588.0	276.0	312.0	46.938776	53.061224	588.0	276.0	312.0	0.474480	51.147541	50.673061	0.884615	312000.0	8280.0	303720.0
2	2.0	588.0	473.0	115.0	80.442177	19.557823	1176.0	749.0	427.0	1.287627	70.000000	68.712373	1.754098	427000.0	22470.0	404530.0
3	3.0	587.0	531.0	56.0	90.459966	9.540034	1763.0	1280.0	483.0	2.200485	79.180328	76.979843	2.650104	483000.0	38400.0	444600.0
4	4.0	588.0	548.0	40.0	93.197279	6.802721	2351.0	1828.0	523.0	3.142567	85.737705	82.595138	3.495220	523000.0	54840.0	468160.0
5	5.0	588.0	569.0	19.0	96.768707	3.231293	2939.0	2397.0	542.0	4.120752	88.852459	84.731707	4.422509	542000.0	71910.0	470090.0
6	6.0	588.0	578.0	10.0	98.299320	1.700680	3527.0	2975.0	552.0	5.114408	90.491803	85.377395	5.389493	552000.0	89250.0	462750.0
7	7.0	588.0	580.0	8.0	98.639456	1.360544	4115.0	3555.0	560.0	6.111503	91.803279	85.691776	6.348214	560000.0	106650.0	453350.0
8	8.0	587.0	577.0	10.0	98.296422	1.703578	4702.0	4132.0	570.0	7.103440	93.442623	86.339183	7.249123	570000.0	123960.0	446040.0
9	9.0	588.0	584.0	4.0	99.319728	0.680272	5290.0	4716.0	574.0	8.107411	94.098361	85.990949	8.216028	574000.0	141480.0	432520.0
10	10.0	588.0	582.0	6.0	98.979592	1.020408	5878.0	5298.0	580.0	9.107944	95.081967	85.974023	9.134483	580000.0	158940.0	421060.0
11	11.0	588.0	582.0	6.0	98.979592	1.020408	6466.0	5880.0	586.0	10.108477	96.065574	85.957097	10.034130	586000.0	176400.0	409600.0
12	12.0	587.0	582.0	5.0	99.148211	0.851789	7053.0	6462.0	591.0	11.109010	96.885246	85.776236	10.934010	591000.0	193860.0	397140.0
13	13.0	588.0	580.0	8.0	98.639456	1.360544	7641.0	7042.0	599.0	12.106105	98.196721	86.090617	11.756260	599000.0	211260.0	387740.0
14	14.0	588.0	585.0	3.0	99.489796	0.510204	8229.0	7627.0	602.0	13.111795	98.688525	85.576730	12.669435	602000.0	228810.0	373190.0
15	15.0	588.0	582.0	6.0	98.979592	1.020408	8817.0	8209.0	608.0	14.112328	99.672131	85.559803	13.501645	608000.0	246270.0	361730.0
16	16.0	588.0	587.0	1.0	99.829932	0.170068	9405.0	8796.0	609.0	15.121456	99.836066	84.714609	14.443350	609000.0	263880.0	345120.0
17	17.0	587.0	586.0	1.0	99.829642	0.170358	9992.0	9382.0	610.0	16.128866	100.000000	83.871134	15.380328	610000.0	281460.0	328540.0
18	18.0	588.0	588.0	0.0	100.000000	0.000000	10580.0	9970.0	610.0	17.139714	100.000000	82.860286	16.344262	610000.0	299100.0	310900.0
19	19.0	588.0	588.0	0.0	100.000000	0.000000	11168.0	10558.0	610.0	18.150561	100.000000	81.849439	17.308197	610000.0	316740.0	293260.0
20	20.0	588.0	588.0	0.0	100.000000	0.000000	11756.0	11146.0	610.0	19.161409	100.000000	80.838591	18.272131	610000.0	334380.0	275620.0

Test set:

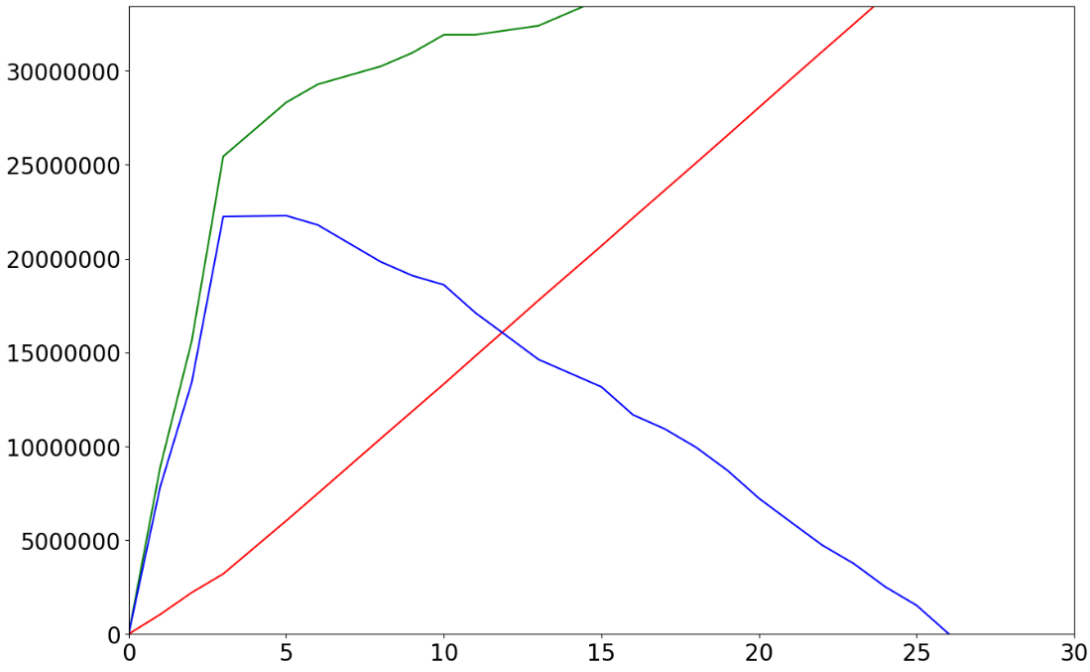
	bin	#recs	#g	#b	%g	%b	tot	cg	cb	%cg	FDR	KS	FPR	Fraud Savings	FP Loss	Overall Savings
1	1.0	252.0	107.0	145.0	42.460317	57.539683	252.0	107.0	145.0	0.429357	53.703704	53.274347	0.737931	145000.0	3210.0	141790.0
2	2.0	252.0	212.0	40.0	84.126984	15.873016	504.0	319.0	185.0	1.280045	68.518519	67.238474	1.724324	185000.0	9570.0	175430.0
3	3.0	252.0	230.0	22.0	91.269841	8.730159	756.0	549.0	207.0	2.202961	76.666667	74.463705	2.652174	207000.0	16470.0	190530.0
4	4.0	252.0	235.0	17.0	93.253968	6.746032	1008.0	784.0	224.0	3.145941	82.962963	79.817022	3.500000	224000.0	23520.0	200480.0
5	5.0	252.0	245.0	7.0	97.222222	2.777778	1260.0	1029.0	231.0	4.129048	85.555556	81.426508	4.454545	231000.0	30870.0	200130.0
6	6.0	251.0	248.0	3.0	98.804781	1.195219	1511.0	1277.0	234.0	5.124192	86.666667	81.542474	5.457265	234000.0	38310.0	195690.0
7	7.0	252.0	250.0	2.0	99.206349	0.793651	1763.0	1527.0	236.0	6.127362	87.407407	81.280045	6.470339	236000.0	45810.0	190190.0
8	8.0	252.0	250.0	2.0	99.206349	0.793651	2015.0	1777.0	238.0	7.130532	88.148148	81.017616	7.466387	238000.0	53310.0	184690.0
9	9.0	252.0	250.0	2.0	99.206349	0.793651	2267.0	2027.0	240.0	8.133702	88.888889	80.755186	8.445833	240000.0	60810.0	179190.0
10	10.0	252.0	249.0	3.0	98.809524	1.190476	2519.0	2276.0	243.0	9.132860	90.000000	80.867140	9.366255	243000.0	68280.0	174720.0
11	11.0	252.0	248.0	4.0	98.412698	1.587302	2771.0	2524.0	247.0	10.128004	91.481481	81.353477	10.218623	247000.0	75720.0	171280.0
12	12.0	252.0	251.0	1.0	99.603175	0.396825	3023.0	2775.0	248.0	11.135187	91.851852	80.716665	11.189516	248000.0	83250.0	164750.0
13	13.0	252.0	252.0	0.0	100.000000	0.000000	3275.0	3027.0	248.0	12.146383	91.851852	79.705469	12.205645	248000.0	90810.0	157190.0
14	14.0	252.0	250.0	2.0	99.206349	0.793651	3527.0	3277.0	250.0	13.149553	92.592593	79.443040	13.108000	250000.0	98310.0	151690.0
15	15.0	252.0	252.0	0.0	100.000000	0.000000	3779.0	3529.0	250.0	14.160748	92.592593	78.431845	14.116000	250000.0	105870.0	144130.0
16	16.0	252.0	252.0	0.0	100.000000	0.000000	4031.0	3781.0	250.0	15.171943	92.592593	77.420649	15.124000	250000.0	113430.0	136570.0
17	17.0	251.0	251.0	0.0	100.000000	0.000000	4282.0	4032.0	250.0	16.179126	92.592593	76.413467	16.128000	250000.0	120960.0	129040.0
18	18.0	252.0	252.0	0.0	100.000000	0.000000	4534.0	4284.0	250.0	17.190321	92.592593	75.402271	17.136000	250000.0	128520.0	121480.0
19	19.0	252.0	252.0	0.0	100.000000	0.000000	4786.0	4536.0	250.0	18.201517	92.592593	74.391076	18.144000	250000.0	136080.0	113920.0
20	20.0	252.0	250.0	2.0	99.206349	0.793651	5038.0	4786.0	252.0	19.204687	93.333333	74.128647	18.992063	252000.0	143580.0	108420.0

OOT:

	bin	#recs	#g	#b	%g	%b	tot	cg	cb	%cg	FDR	KS	FPR
1	1.0	124.0	87.0	37.0	70.161290	29.838710	124.0	87.0	37.0	0.710320	20.670391	19.960071	2.351351
2	2.0	125.0	97.0	28.0	77.600000	22.400000	249.0	184.0	65.0	1.502286	36.312849	34.810563	2.830769
3	3.0	124.0	83.0	41.0	66.935484	33.064516	373.0	267.0	106.0	2.179948	59.217877	57.037929	2.518868
4	4.0	124.0	118.0	6.0	95.161290	4.838710	497.0	385.0	112.0	3.143370	62.569832	59.426462	3.437500
5	5.0	124.0	118.0	6.0	95.161290	4.838710	621.0	503.0	118.0	4.106793	65.921788	61.814995	4.262712
6	6.0	125.0	121.0	4.0	96.800000	3.200000	746.0	624.0	122.0	5.094709	68.156425	63.061715	5.114754
7	7.0	124.0	122.0	2.0	98.387097	1.612903	870.0	746.0	124.0	6.090790	69.273743	63.182953	6.016129
8	8.0	124.0	122.0	2.0	98.387097	1.612903	994.0	868.0	126.0	7.086871	70.391061	63.304190	6.888889
9	9.0	124.0	121.0	3.0	97.580645	2.419355	1118.0	989.0	129.0	8.074788	72.067039	63.992251	7.666667
10	10.0	125.0	121.0	4.0	96.800000	3.200000	1243.0	1110.0	133.0	9.062704	74.301676	65.238972	8.345865
11	11.0	124.0	124.0	0.0	100.000000	0.000000	1367.0	1234.0	133.0	10.075114	74.301676	64.226562	9.278195
12	12.0	124.0	123.0	1.0	99.193548	0.806452	1491.0	1357.0	134.0	11.079360	74.860335	63.780975	10.126866
13	13.0	125.0	124.0	1.0	99.200000	0.800000	1616.0	1481.0	135.0	12.091770	75.418994	63.327224	10.970370
14	14.0	124.0	121.0	3.0	97.580645	2.419355	1740.0	1602.0	138.0	13.079686	77.094972	64.015286	11.608696
15	15.0	124.0	121.0	3.0	97.580645	2.419355	1864.0	1723.0	141.0	14.067603	78.770950	64.703347	12.219858
16	16.0	124.0	124.0	0.0	100.000000	0.000000	1988.0	1847.0	141.0	15.080013	78.770950	63.690937	13.099291
17	17.0	125.0	122.0	3.0	97.600000	2.400000	2113.0	1969.0	144.0	16.076094	80.446927	64.370833	13.673611
18	18.0	124.0	122.0	2.0	98.387097	1.612903	2237.0	2091.0	146.0	17.072175	81.564246	64.492071	14.321918
19	19.0	124.0	123.0	1.0	99.193548	0.806452	2361.0	2214.0	147.0	18.076421	82.122905	64.046484	15.061224
20	20.0	124.0	124.0	0.0	100.000000	0.000000	2485.0	2338.0	147.0	19.088831	82.122905	63.034074	15.904762

Financial curves and recommended cutoff:

The plot below shows the financial curves of the model. I assume $400 gain for every fraud that's caught and $20 loss for every false positive (a good that's flagged as a bad). The green curve shows Fraud $'s Caught, red curve shows Lost Revenue, and blue curve shows Overall Savings. As the result shows, the overall saving reach the highest at 3%. Therefore, I will recommend a cutoff at 3%. It can generate approximately $22.3 million saving from 12,427 transactions.



Summary:

In this project, I used 96,753 rows of Credit Card Transaction Data to build a fraud detection model that helps the company increase saving.

To achieve the goal, I first created a data quality report (attached in appendix) for better understanding of the dataset. Later, I conducted data cleaning by addressing outliers and exclusion and applying various methods for imputation. I further used target encoding to convert categorical fields to numeric. I also create amount variables, frequency variables, days-since variables, and velocity change variables for the model.

After creating thousands of variables, I used Kolmogorov Smirnov test to filter 300 variables. Later, I applied wrapper to further narrow it down to 20 variables. I used forward stepwise selection and LGBM (n_estimators=40, num_leaves=4, cv=4) for the wrapper.

Using 20 chosen variables, I built Logistic Regression, Decision Tree, Random Forest, LGBM, and Neural Network models to predict fraud. After tuning the models and comparing performance based on FDR@3% for out of time data, I choose Random Forest with 20 variables, 150 num_leaves, entropy criterion, 20 max_depth, 450 min_samples_split, 50 min_samples_leaf as the final model.

I plotted the financial curve of the final model by applying to the OOT set. The result shows the overall savings keep increasing until the cutoff is set to 3%. If the cutoff is set to be larger than 5%, the overall saving will start decreasing. Therefore, I recommend a cutoff at 3%. It can generate approximately $22.3 million saving from 12,427 transactions.

Appendix:

Data Quality Report

1. Data Description

The dataset is **Credit Card Transaction Data**, which contains **Transaction and Company Identifying Information** of credit cards. The data came from a U.S. government organization. There are **10 fields** and **96,753 records**.

2. Summary Tables

Numeric Fields Table

Field Name	# Records With Values	% Populated	# Zeros	Min	Max	Mean	Most Common	Stdev
Date	96,753	100.00%	0	1/1/2010	12/31/2010	6/25/2010	2/28/2010	98 days 21:38:58
Amount	96,753	100.00%	0	0.01	3,102,045.53	427.885677	3.62	10,006.14

Categorical Fields Table

Field Name	# Records With Values	% Populated	# Zeros	# Unique Values	Most Common
Recnum	96,753	100.00%	0	96,753	1
Cardnum	96,753	100.00%	0	1,645	5142148452
Merchnum	93,378	96.51%	0	13,091	930090121224
Merch description	96,753	100.00%	0	13,126	GSA-FSS-ADV
Merch state	95,558	98.76%	0	227	TN
Merch zip	92,097	95.19%	0	4,567	38118
Transtype	96,753	100.00%	0	4	P
Fraud	96,753	100.00%	95,694	2	0

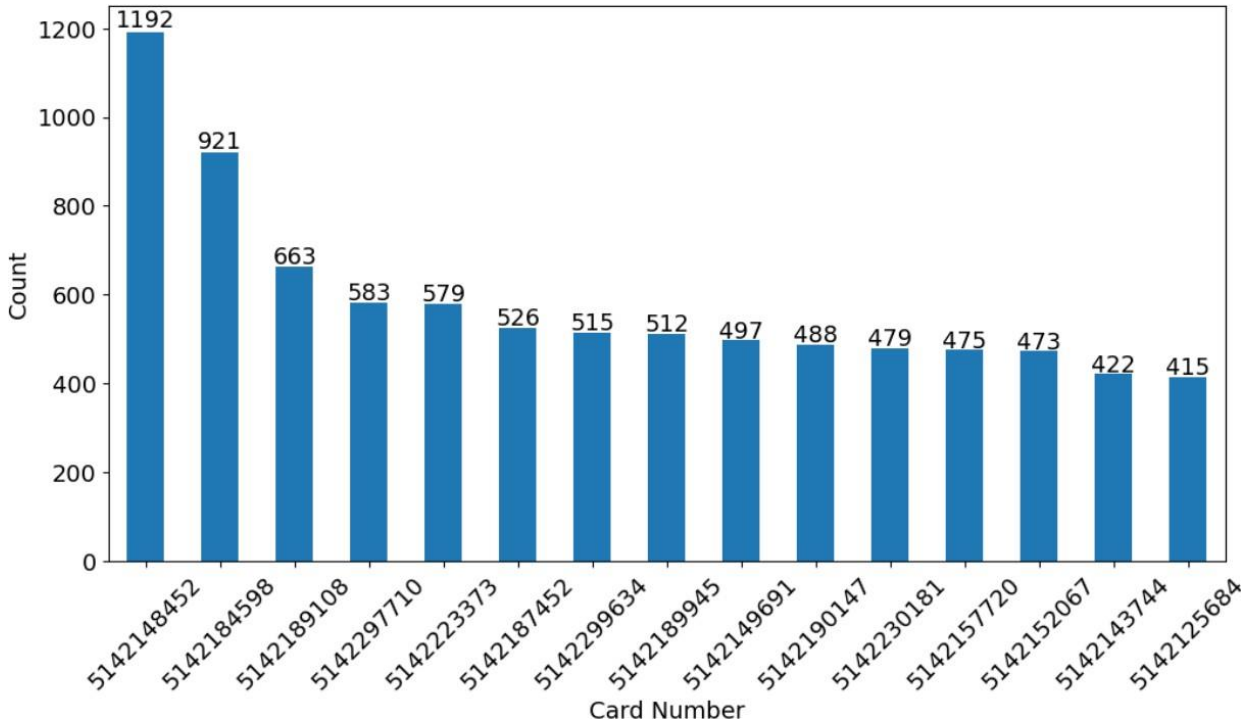
3. Visualization of Each Field

1) Field Name: Recnum

Description: Ordinal unique positive integer for each application record, from 1 to 96,753.

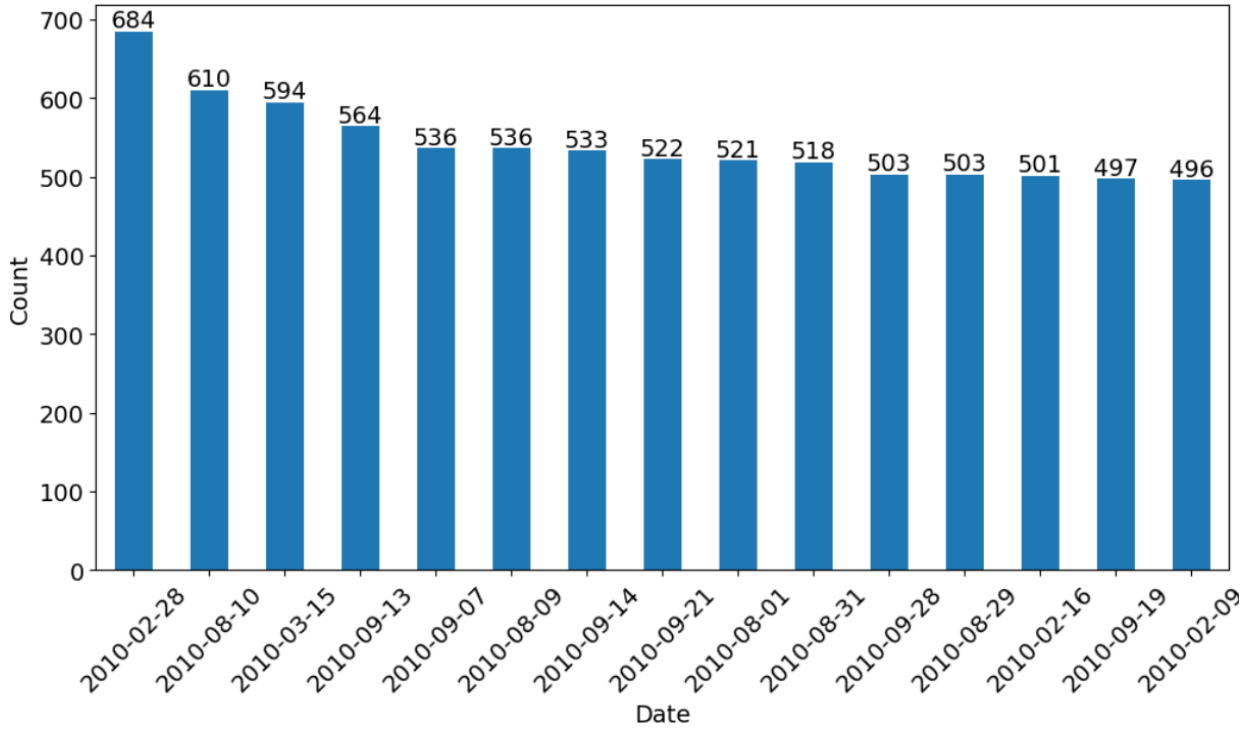
2) Field Name: Cardnum

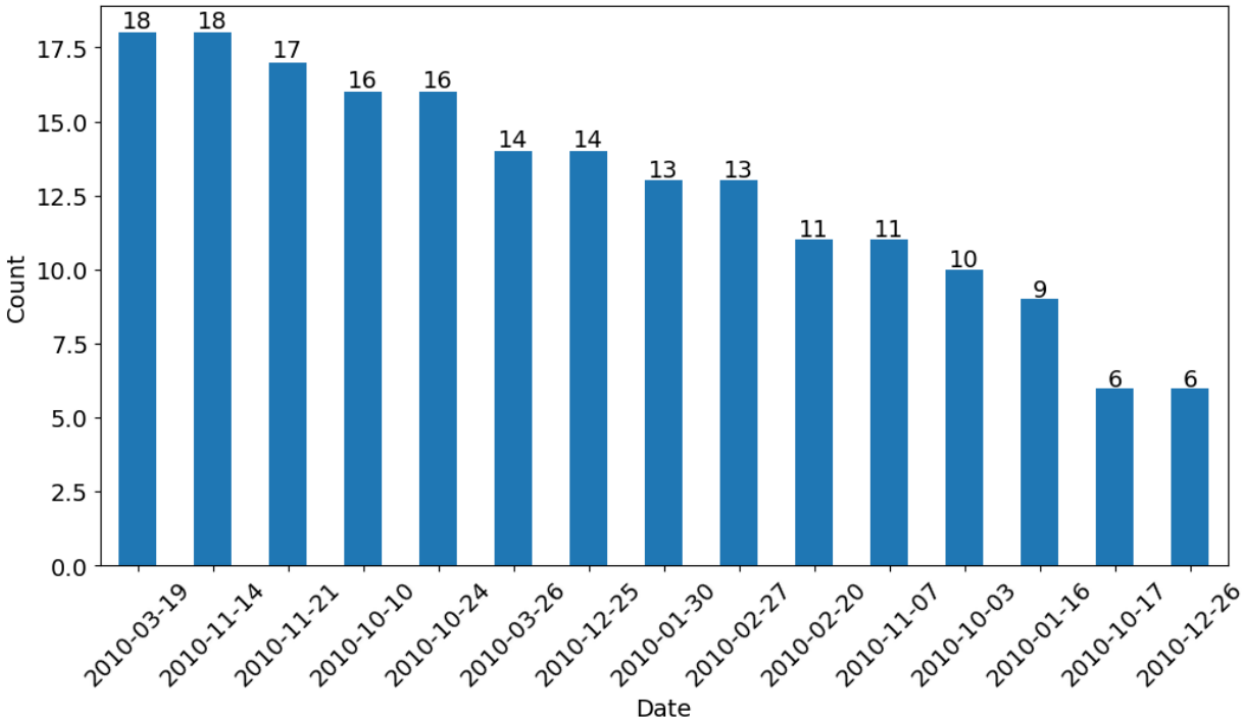
Description: Card number. The distribution shows the top 15 field values of card number.The most common card number is 5142148452, with a total count of 1,192.

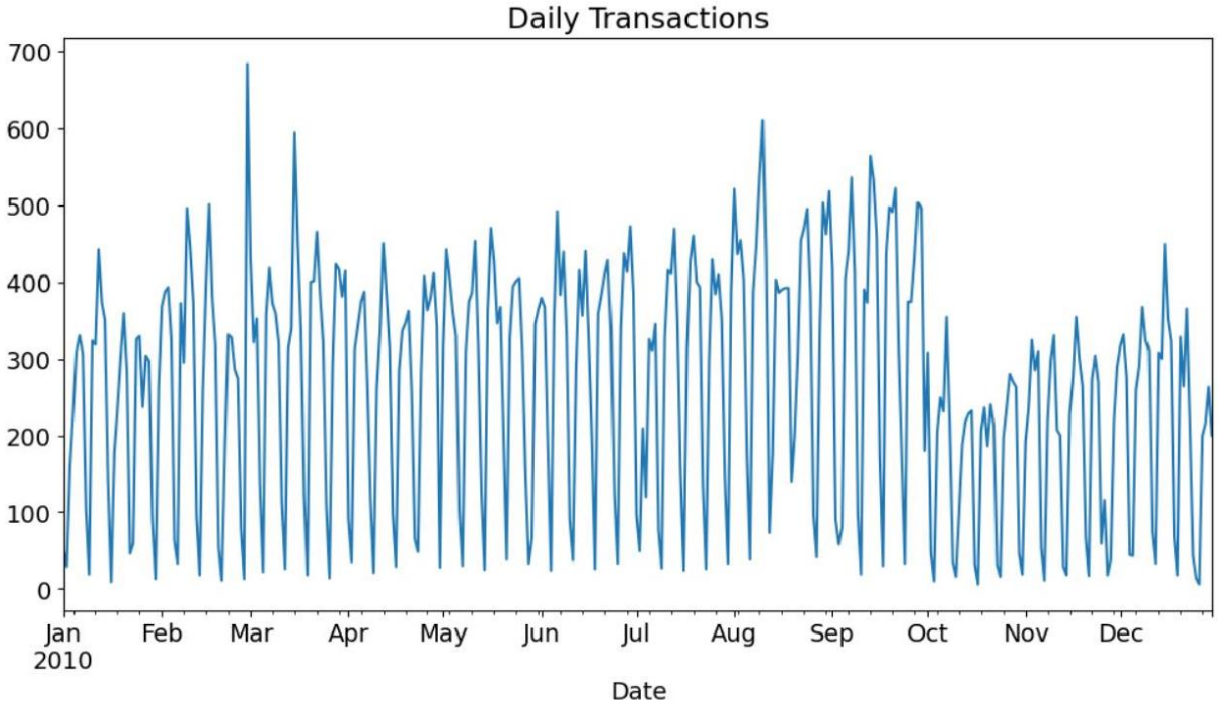


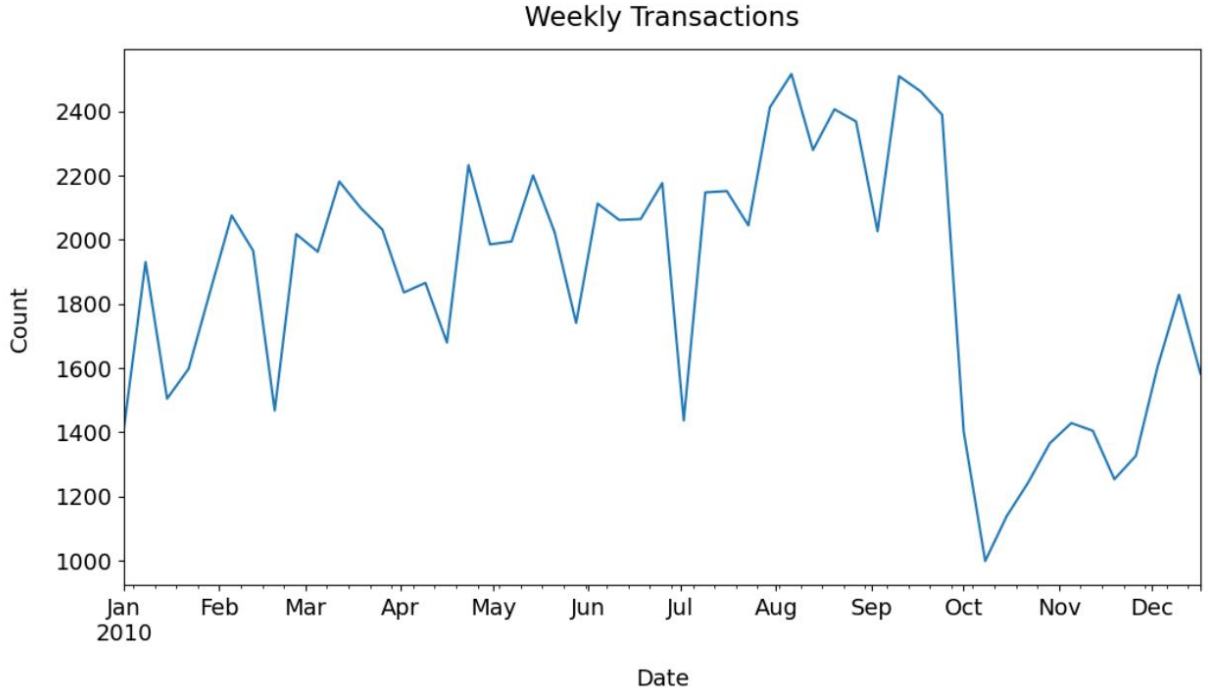
3) **Field Name: date**

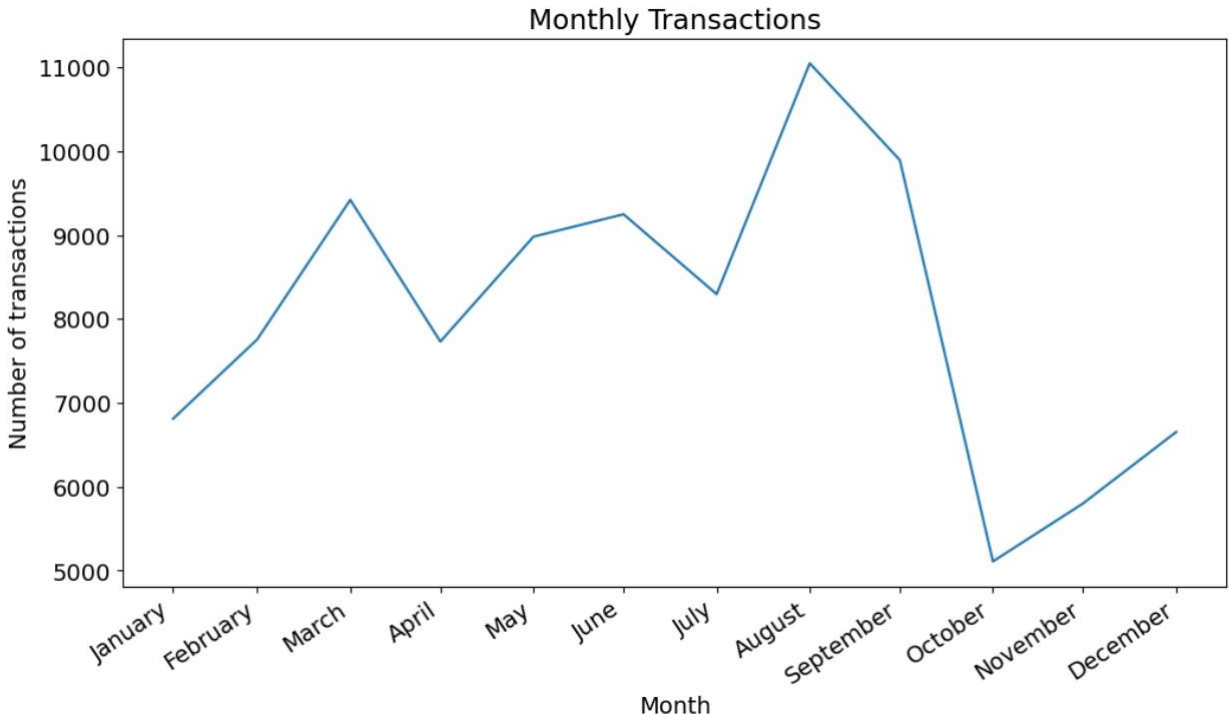
Description: Transaction date. The first distribution shows the top 15 field values transaction date. The most common transaction date is 2010-02-28, with a total count of684. applications across time. The second distribution shows the lowest 15 field values transaction date. The least common transaction dates are 2010-12-26 and 2010-10-17, with a total count of 6. The third chart shows the number of daily transactions across time. The fourth chart shows the number of weekly transactions across time. The fifth chart shows the number of monthly transactions across time.


Count
Date
684
610
594
564
536
536
533
522
521
518
503
503
501
497
496
2010-02-28
2010-08-10
2010-03-15
2010-09-13
2010-09-07
2010-08-09
2010-09-14
2010-09-21
2010-08-01
2010-08-31
2010-09-28
2010-08-29
2010-02-16
2010-09-19
2010-02-09


Count
Date
18
18
17
16
16
14
14
13
13
11
11
10
9
6
6
2010-03-19
2010-11-14
2010-11-21
2010-10-10
2010-10-24
2010-03-26
2010-12-25
2010-01-30
2010-02-27
2010-02-20
2010-11-07
2010-10-03
2010-01-16
2010-10-17
2010-12-26

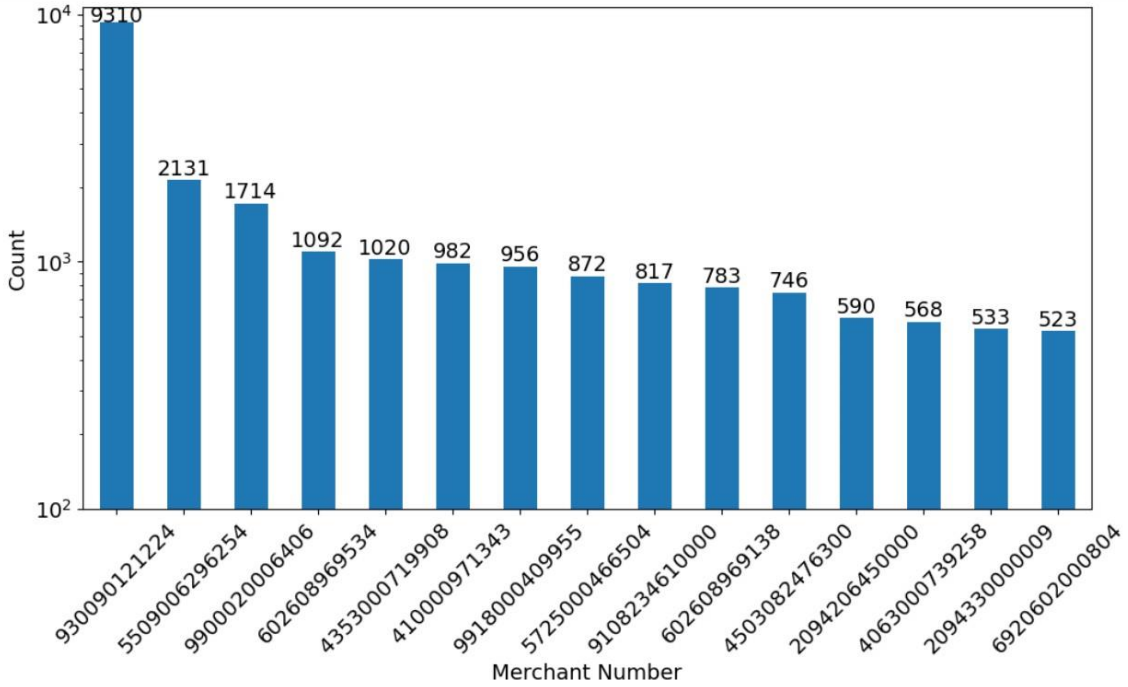

Daily Transactions
700
600
500
400
300
200
100
0
Jan
2010
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec
Date


Weekly Transactions
Count
2400
2200
2000
1800
1600
1400
1200
1000
Jan
2010
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec
Date



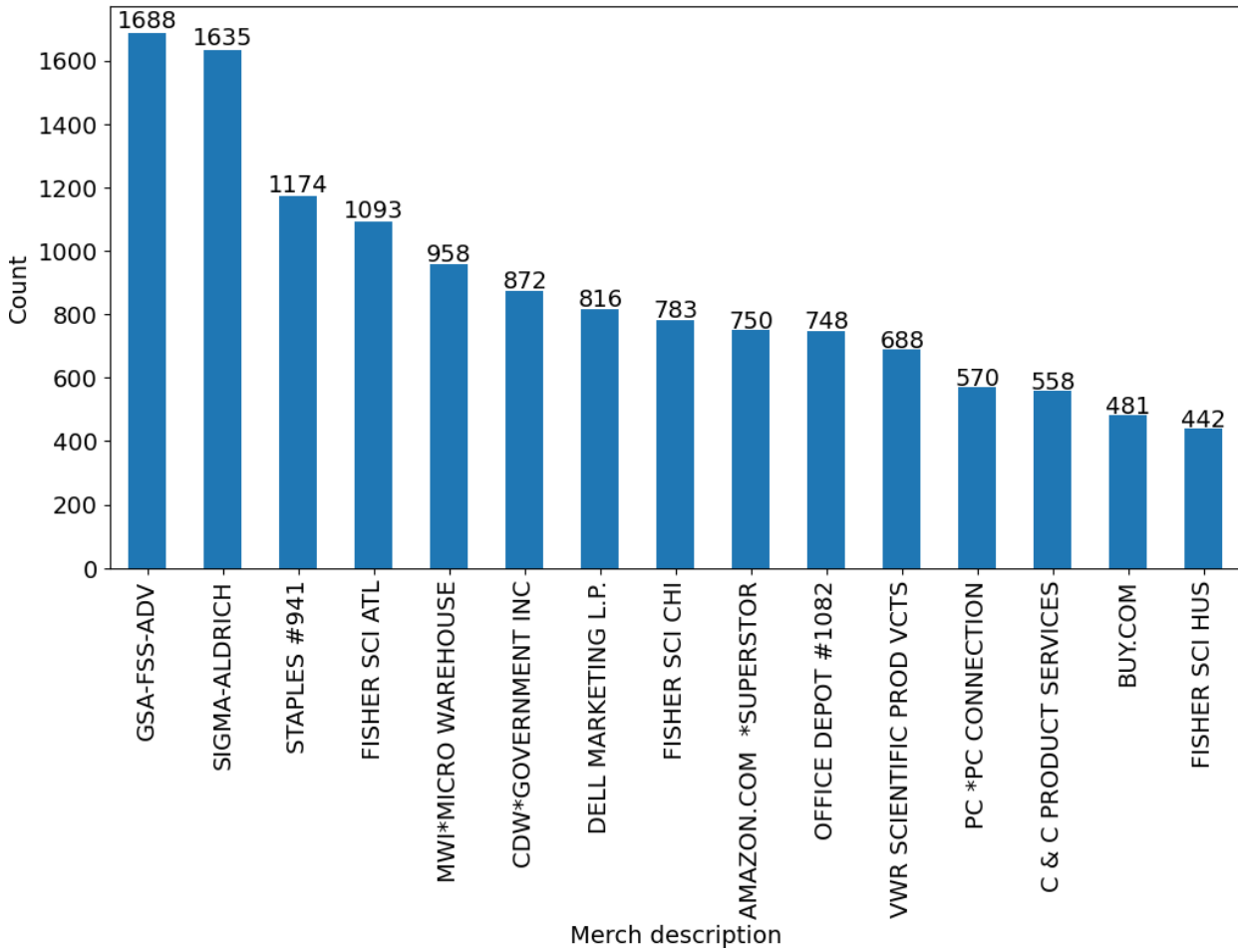
4) Field Name: Merchnum

Description: Merchant number. The distribution shows the top 15 field values of merchant number. The most common merchant number is 930090121224, with a totalcount of 9,310.



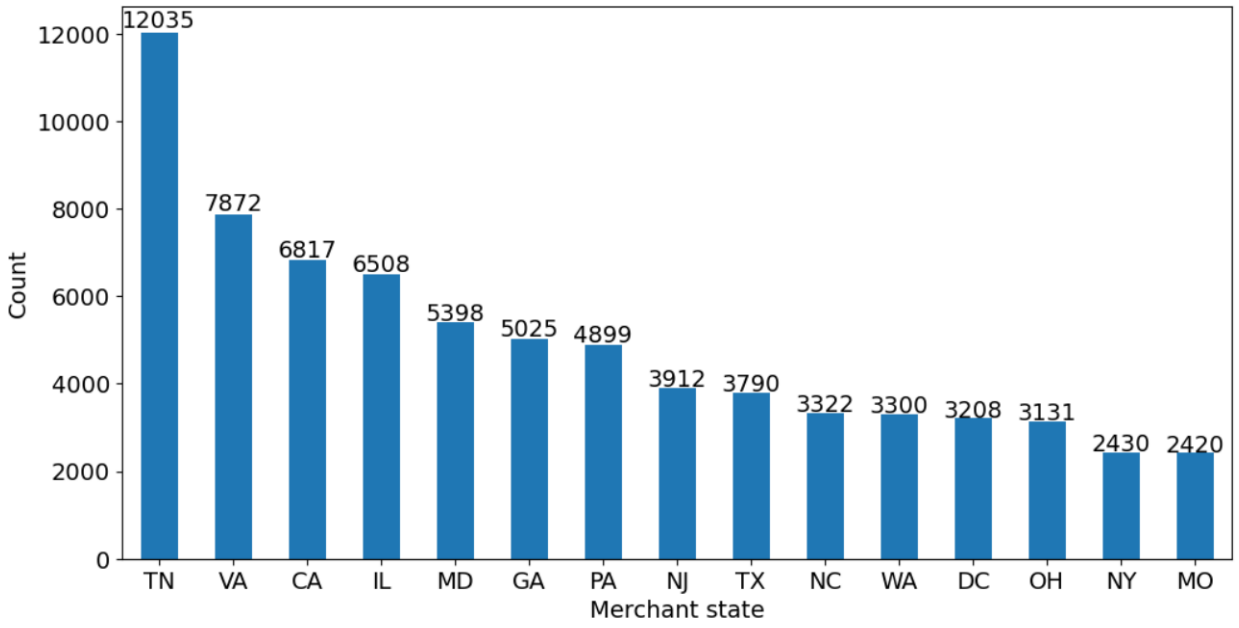
5) **Field Name: Merch description**

Description: Merchant description. The distribution shows the top 15 field values of merchant description. The most common merchant description is GSA-FSS-ADV, with atotal count of 1,688.



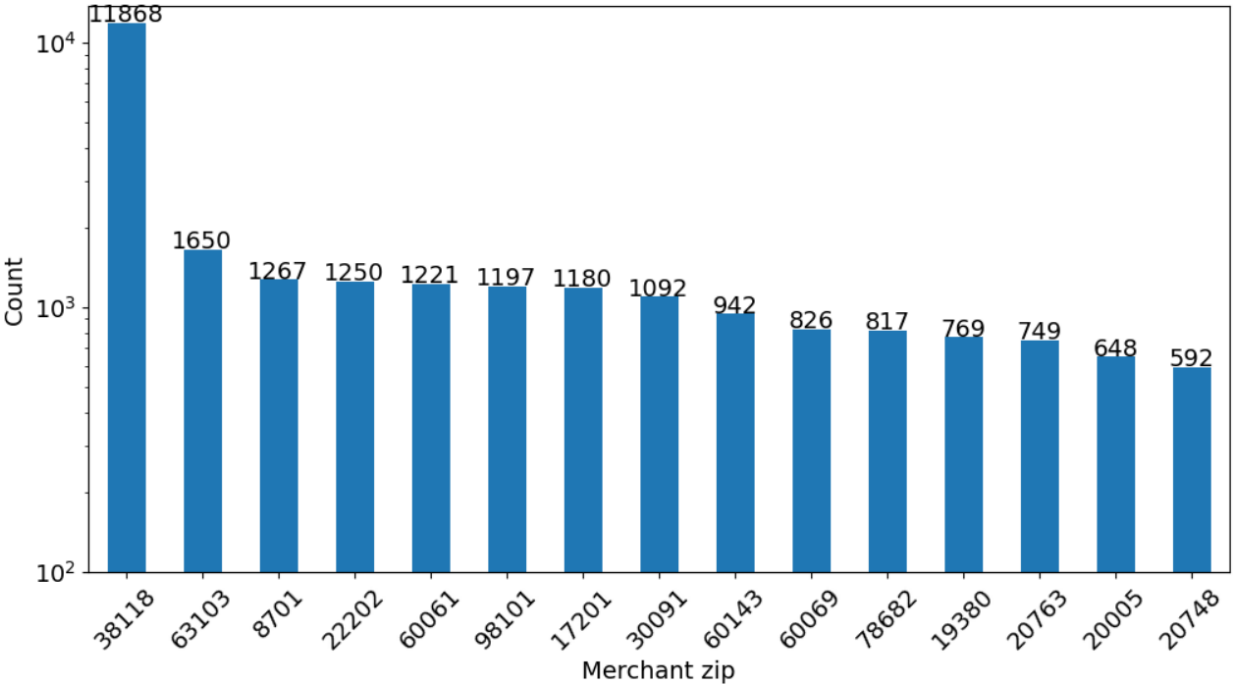
6) **Field Name: Merch state**

Description: Merchant state. The distribution shows the top 15 field values of merchantstate. The most common merchant state is TN, with a total count of 12,035.



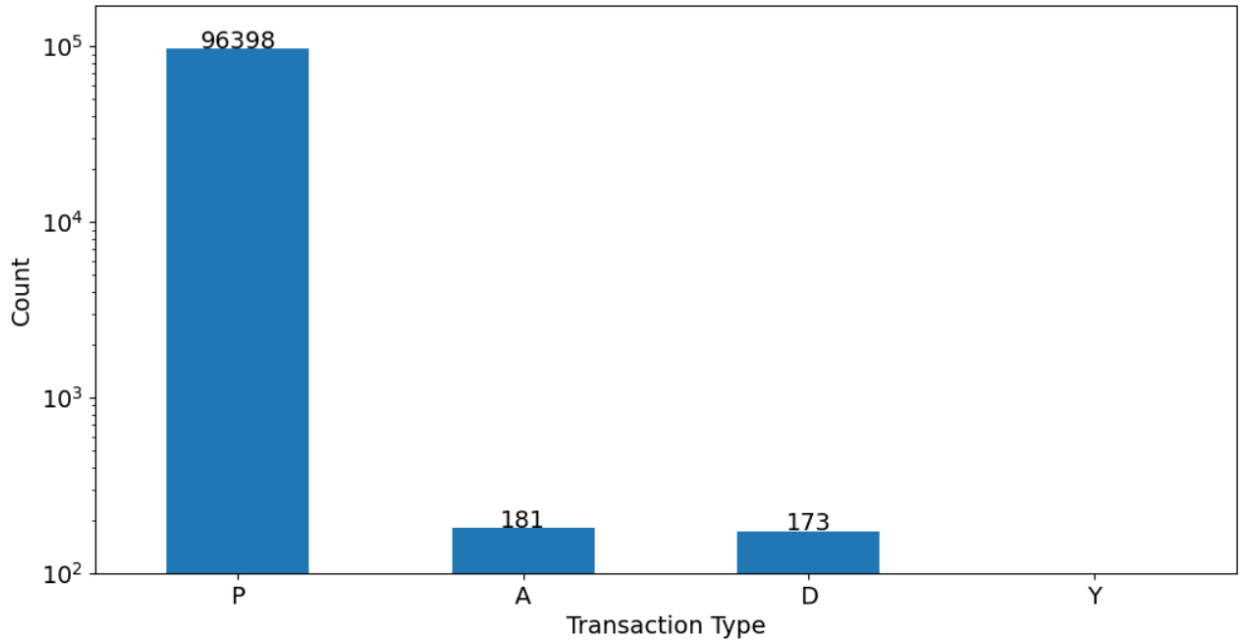
7) **Field Name: Merch zip**

Description: Merchant zip code. The distribution shows the top 15 field values Merchantzip code. The most common Merchant zip code is 38118, with a total count of 11,868.



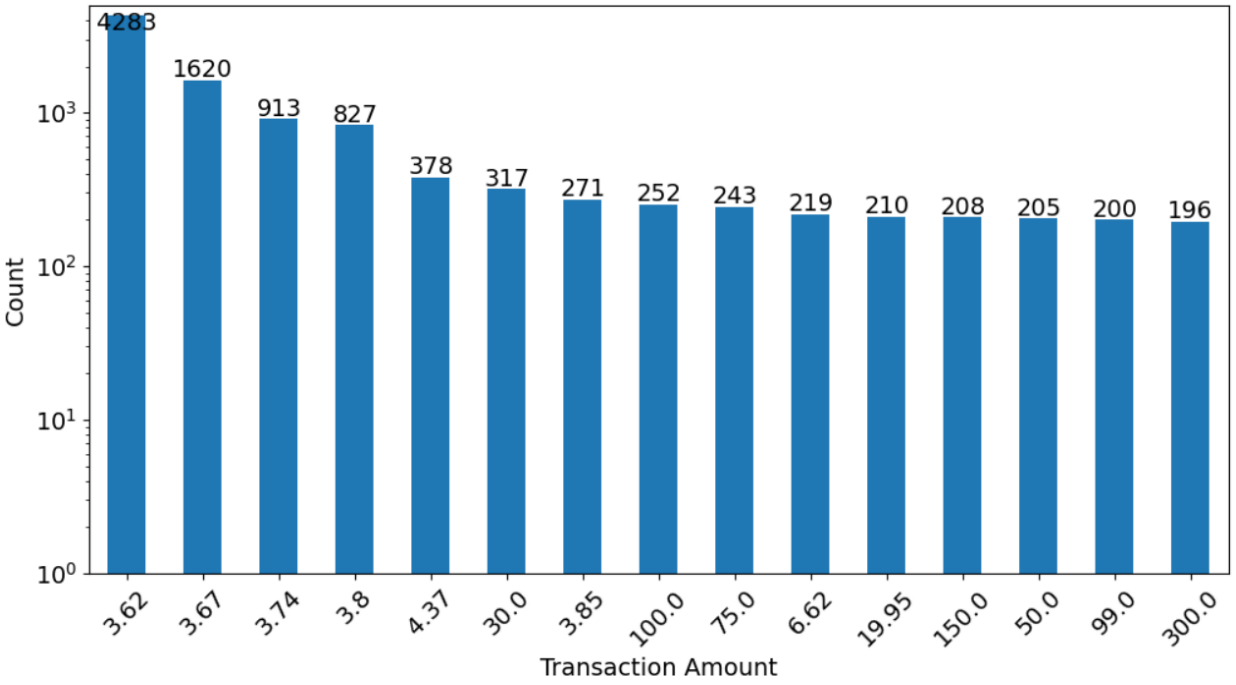
8) **Field Name: Transtype**

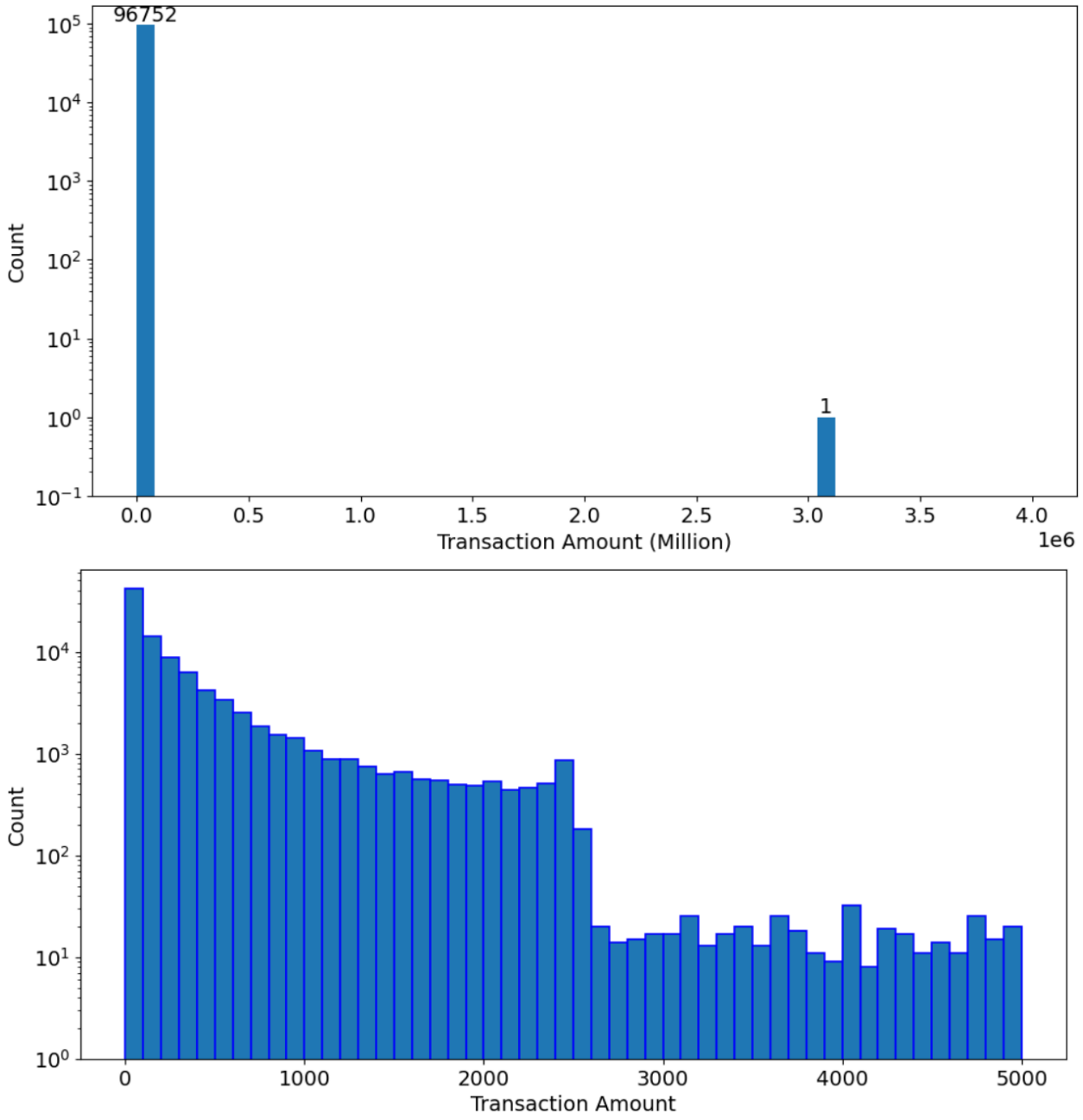
Description: Transaction type. The total count of Transtype = P is 96,398. The total countof Transtype = A is 181. The total count of Transtype = D is 173. The total count of Transtype = Y is 1.



9) Field Name: Amount

Description: Transaction amount. The first distribution shows the top 15 field values of transaction amount. The most common transaction amount is 3.62, with a total count of 4,283. The second distribution shows the total count of transaction amount < 500,000 is 96,752. The total count of transaction amount >3,000,000 < 3,500,000 is 1. The third distribution shows the total count of transaction amount after filtering the highest transaction amount.

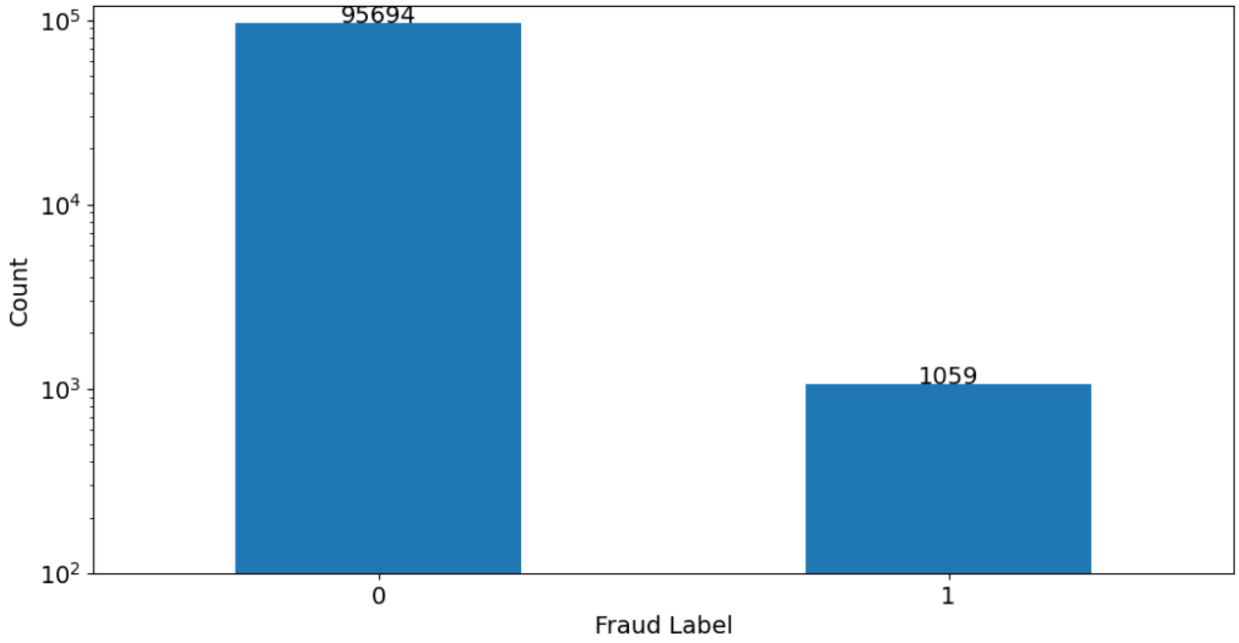




10) Field Name: Fraud

Description: Fraud identification label. Fraud = 0 (Not fraudulent), Fraud =1 (Fraud identified).

The total count of fraud_label = 0 is 95,694. The total count of fraud_label =1 is 1,059.


95694
1059
10^5
10^4
10^3
10^2
Count
0
1
Fraud Label