IEEE-FRAUD-DETECTION COMPETITION KAGGLE

CST4050 CW2

Abduarahman Bazena
Ab2663@live.mdx.ac.uk

Table of Contents

Introduction

This report outlines the dataset descripts and a brief overview of the challenge in this machine learning competition. Additionally, it outlines the adapted machine learning pipeline in this assignment which includes exploratory data analysis, feature selection and missing data handling. Class balancing, feature scaling and dimensionality reduction, hyperparameter tuning and model training and validation. Furthermore, it shows the evaluation of the models along with the submission of the results to the Kaggle competition which managed to get a 0.72 score. Applied models managed to achieve between 0.88 and 0.87 during the local training/testing evaluation.

Report

Dataset

The selected data set was obtained from Kaggle specifically published for IEEE Fraud Detection competition. The dataset was provided by Vesta Corporation. Vesta is one of the forerunners in guaranteed payment solutions for e-commerce. Vesta mainly operates for card not present transaction and today having over $18B in transactions annually. Vesta is seeking to advance the fraud prevention industry by providing this dataset to researchers to try and build machine learning models to predict fraud transactions.

A training dataset and a testing dataset were provided. Each of the datasets is made of two tables Transactions and Identity. The transaction table contains information about the transaction such as data and time, transaction amount etc. where the identity table contains information with masked field names, features include device type and network information. Full feature list and description is provided below.

The test dataset does not contain the label column and the actual classification score can be only obtained by submitting the results on Kaggle.

The provided training dataset is of a large scale having 590540 instances and 435 features, 404 of which are numerical values and the other 31 are categorical. Each transaction has a unique transaction ID which can be used to identify each transaction.

Features and their description as provided by the dataset owner.

Transaction table:

The TransactionDT feature indicates that the first value corresponds to the number of seconds in a day (60 * 60 * 24 = 86400) which indicates that the time unit is in seconds and the maximum value indicates that the data spans over a period of 6 months.

Labelling logic:

The isFraud is the label for this data set where 1 means it is a fraud transaction and 0 means it is a legit transaction.

- TransactionDT: Timedelta from a given reference datatime
- TransactionAMT: Transaction payment amount in US dollars
- ProductCD: Product code for each transaction
- card1 - card6: Payment card information such as card type, card category, etc
- addr: Address
- dist: distance
- P_ and (R__) emaildomain: purchaser and recipient email domain

- C1-C14: Counting, such as how many addresses are found to be associated with the payment card. Actual meaning is masked
- D1-D15: timedelta, such as days between previous transaction, etc.
- M1-M9: match, such as names on card and address, etc.
- Vxxx: Vesta engineered rich features, including ranking, counting, and other entity relations.

Categorical Features:

- ProductCD
- card1 - card6
- addr1, addr2
- P*emaildomain* *R*emaildomain
- M1 - M9

Identity table:

Variables in this table are identity information – network connection information (IP, ISP, Proxy, etc) and digital signature (UA/browser/os/version, etc) associated with transactions.

Field names are masked due to privacy reasons.

Categorical Features:

- DeviceType
- DeviceInfo
- id*12* - *id*38

the training dataset is extremely imbalanced where 96.5% of the data belongs to the 0 class (not fraud) and only 3.5% is labelled as fraud which indicated that the data either needs to be balanced before using it to train a model or select a model that can compensate for such a scenario

Appendix 1 shows Exploratory Data Analysis of the training data set which includes feature value counts and transaction amount distribution. Count of features with null values per datatype etc.

The accepted challenge is to explore the data, carry out data pre-processing, preparation of the data and training and evaluating a model to predict fraud transactions.

Literature Review:

this competition was hosted on Kaggle and it was closed 2 months ago. The wining prize was $20K. the winning team managed to score 0.945884 and the second place had a score of 0.94421 and the third place had a score of 0.943769.

the wining team used XGBoost as their model however they relied on feature engineering and created what they called the magical feature which helped their model improve after initially scoring around 0.89

ML Pipeline:

Exploratory Data Analyses - EDA

The adopted pipeline consists of first, running general exploratory data analyses to understand the data and gain insights about the data set. The EDA helped in understanding the dataset and made working with it easier by knowing which features can be used and made the scope of feature selection clearer since the dataset is of large scale and could result in technical challenges and limitations which were found at a later stage due to the hardware limitations.

Please refer to appendix 1 for full EDA Notebook.

Feature Selection and Missing Data Processing:

feature selection was based on two main aspects first aspect was selected mainly to ensure that the employed missing data handling methods.

First step of feature selection was dropping features with more than a specific percentage of missing data. For example, for numerical features any feature with more than 60% missing data was dropped because that feature will negatively affect the model since even statistical approaches will fail to fill the missing data with appropriate values. The categorical values had a lower percentage, where any categorical feature with more than 40% missing data was dropped to ensure that the used features have enough data for the machine learning model that will try to predict the values based on the values of the other features.

After dropping features with too much missing data from both numerical and categorical features. The number of features was reduced from 435 to 211 numerical features and 12 categorical features.

The second aspect of feature selection will be applied to numerical features only since it is based on the correlation between each feature and the label feature. However, before calculating the correlation coefficient of each feature it needed to be with no missing values. Which required processing numerical features for missing data before selecting features based on the correlation. For this regular statistical method was (replacing missing values with the mean) applied in a controlled manner. This was done by first splitting the data by class (fraud / not fraud) and then filling the missing values with mean based on the class. This proved to help the model at a later stage after testing with both methods. In other words, replacing the missing values with the mean of the whole dataset resulted in confusing the model and where replacing the null with the mean based on the class allowed the model to perform better.

After processing the numerical features and ensuring they have no nulls it is now possible to calculate the correlation coefficient for each feature with the class label and dropping any features with close to 0 correlation. This helped in selecting the best 155 features out of the 211 numerical features

extracted from the previous step. All selected features have a correlation higher that +0.01 or lower than -0.01.

The selected numerical features will be first used to train a model to predict the missing values in the categorical features which is one of the best approaches in handling the missing values from the categorical features. And for this each categorical feature was processed individually. First the feature that is being processed is encoded to a numerical value using a Label Encoder which gives each category a unique numerical representation. Then all instances with null values are selected to be the testing set and the instance with present values are used for the training set. The previous step of selecting features with less than 40% missing data was mainly to ensure that the model will have more training data to be able to predict accurately.

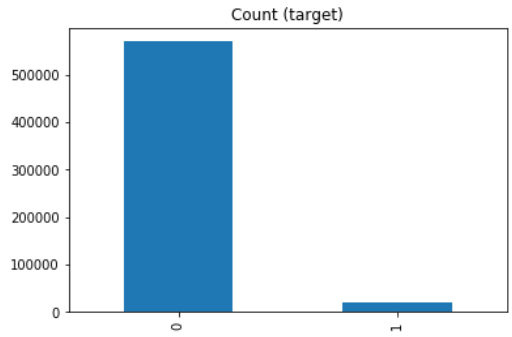
Two models were trained for this task. Firstly, a K-Nearest-Neighbour which scored between 78 and 95 for all features. However, the model was extremely slow. The second model was Extreme Gradient Boosting, which scored between 70 and 93 in predicting the categorical values when cross validated with the training data. After predicting the missing values, the data was inverse transformed its state before encoding by the label encoder. After predicating missing values for all categorical values, all categorical features were converted to numerical using one-hot-encoding to prepare it for the next stage.

Please refer to appendix 2 for Data processing Notebook.

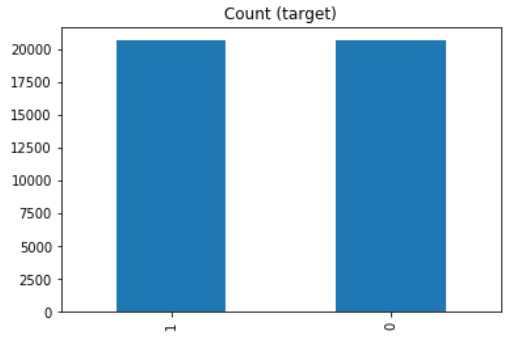
Class Balancing:

As previously mentioned, the training dataset is extremely imbalanced having 96.5% of class not fraud and only 3.5% of class fraud. Which mean if a model always predicts not fraud it will be 96% of the time right.

The two most popular methods of balancing a dataset is oversampling (replicating the minority class instances to match the number of the majority class) and under-sampling selecting number of the majority class and make it match the number of the minority class. In realty both methods have a downside. For instance, oversampling could result in the model cheating where it will sometimes have the exact same instances in both the training and testing sets if not applied in a statistical fashion. On the other side, under-sampling will result in data loss since some instances will be omitted from the majority class resulting in the model not having enough data. In this instance the selected method is under-sampling because the dataset is very large, and it is challenging to have enough hardware to train and test the model on. At the same time since the data is large, we can still risk losing some data as we will still have enough to train and test the model. Please refer to appendix 3 for data balancing notebook.



Class label count before balancing



Class label count after balancing

Data Scaling and Dimensionality Reduction:

The training data set was scaled to range between 0 and 1 using Minimax scaler to ensure that any features with large difference and variance in scale will not affect the model negatively and enhance model performance speed wise. Additionally, the scaler was fitted to the training data only and was used to transform the testing data. Separate fitting and transformation of the data was ensured even during k-fold cross validation.

Since the dataset is of high dimensionality, PCA Principle Component Analysis was performed on the training data to reduce its dimensionality to only 5 dimensions to improve model performance and run time. The number of dimensions was selected after manual parameter tuning where less than 5 dimensions results in extra data loss and the model scores low. Fitting and transforming the training and testing data was performed separately even during the k-fold cross validation.

Validation:

Three model were trained on the data. Two of which are XGBoost and the other one was RandomForest. Two methods of validation were carried out which are, the regular train test split with 80% of the data used for the training and 20% of the data was used for the testing both models had good scores on this validation test. The second validation method was Stratified K-Fold cross validation where k was set to 10. The RandomForest scored 0.874 on the K-Fold and 0.871 on the train test split. Where the XGBoost scored 0.883 on both validation methods. These models were trained and tested on the balanced dataset.

The variant of XGBoost was implemented with scale_pos_weight parameter indicating the ratio of the classes to be (1: 27.580) this parameter should allow the model to cater for the class imbalance. The train test split result for this model was 0.972 and the K-Fold cross validation score was 0.971. however, further evaluation showed that the model started with 96% accuracy on the learning curve test.

Please refer to appendix 4 for data scaling and PCA along with model validation.

Hyperparameter Tuning:

Both models were tuned using hyperparameter tuning methods. However, due to the large size of the dataset XGBoost hyperparameter tuning could not be run since it demanded a lot more memory that what is available. The random forest Randomized Search was used, and it was the reason it managed to score as a good result as the XGBoost. The XGBoost is faster in run time and more accurate than the RandomForest even without hyperparameter tuning the algorithm.

Hyperparameter tuning code can be found at the bottom of notebooks appendix 4.

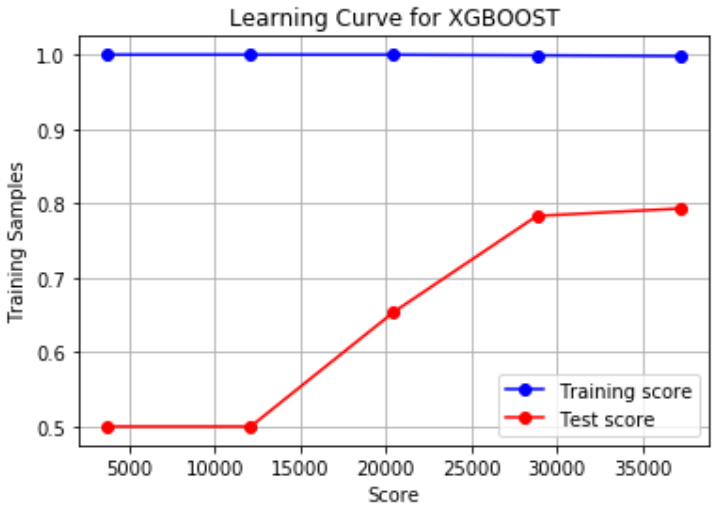
Evaluation:

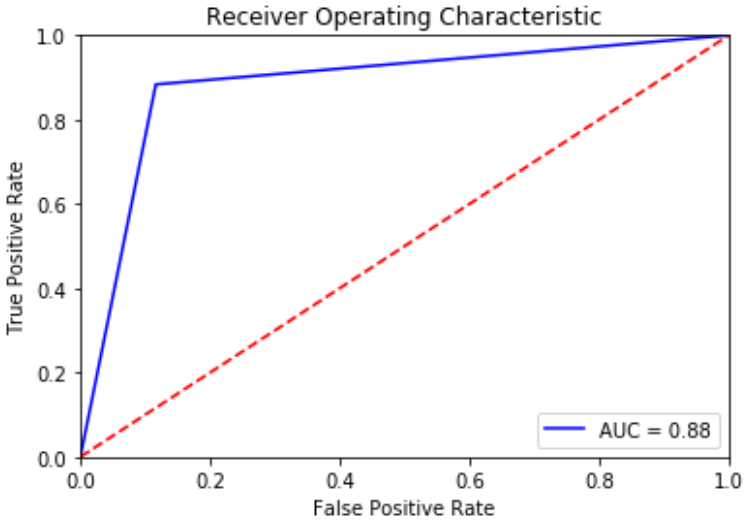
The learning curve for XGBoost shows that the training score is at max and the test score increasing by increasing the number of samples used. The following evaluation was carried out to diagnose weather the model is underfitting, overfitting or well fitted. Where if both train and test scores are both low mean the estimator is underfitting. And if the training score is high and the testing score is low mean the estimator is overfitting. Otherwise the model is well fitted. The following learning curve diagrams highly show the model are well fitted.

Additionally, Receiver Operating Characteristic Curves were plotted to show the performance of the models. The ROC plots show the diagnostic ability of the classifiers as its discrimination threshold is varied. The following ROC evaluations indicate that XGBoost managed to score 0.88, random forest managed to score 0.87 which are considered to be very good scores.

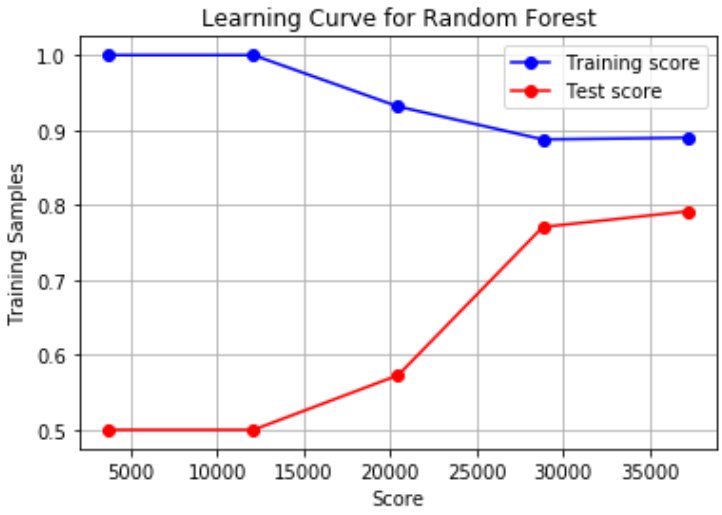
Code is included with the model training.

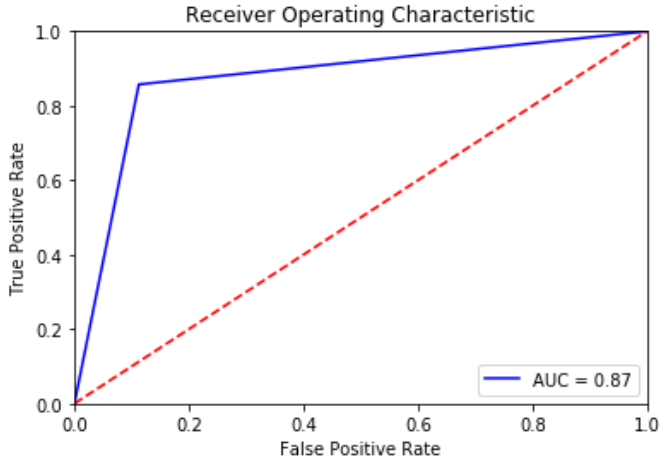
XGBoost Evaluation:



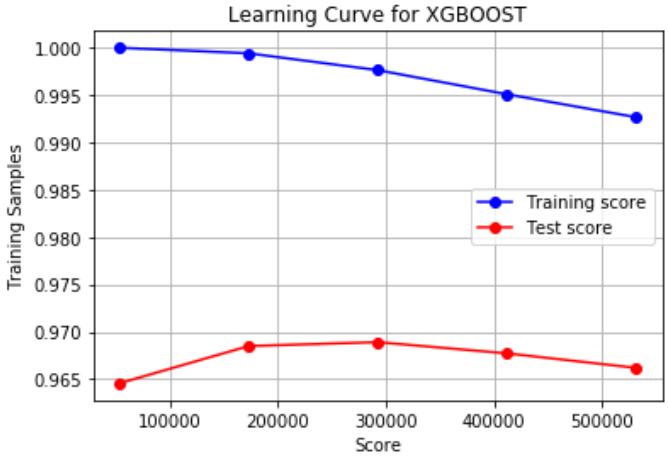


RandomForest Evaluation:



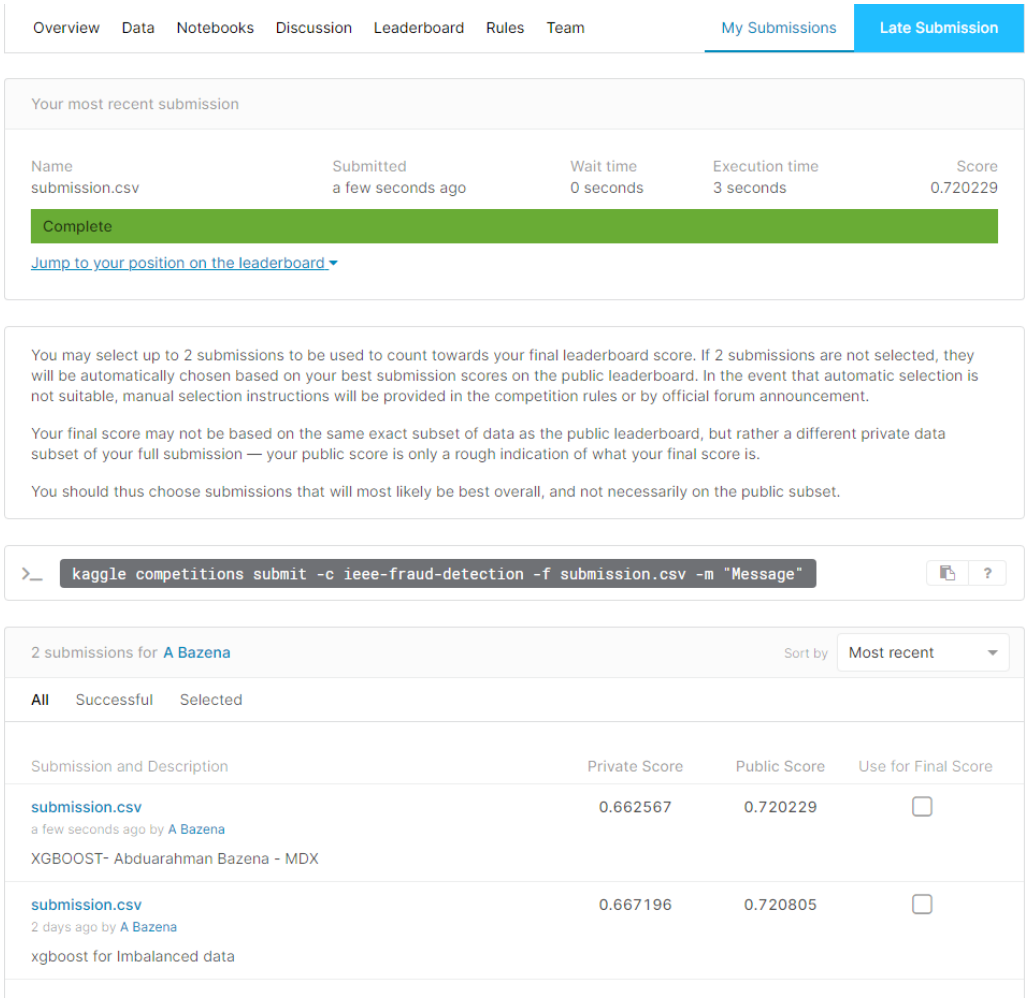


XGBoost Imbalanced Data:



Kaggle Submission:

A variant of the XGBoost model was trained on the full training dataset and tested on the testing dataset after applying the same data cleaning and selecting the features previously selected for the training. The model managed to score 0.72 on the testing dataset after submitting the results to Kaggle.



Conclusion

In conclusion, the trained classifiers managed to score very good results on trained and tested on the training dataset using the appropriate validation methods such as the train test split and K-Fold cross validation where both models scored 0.88 for the XGBoost and 0.87 for the RandomForest

Appendixes

ExploratoryDataAnalysis

April 22, 2020

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

[2]:
```
df_id = pd.read_csv("./input/raw/train_identity.csv")
df_trans = pd.read_csv("./input/raw/train_transaction.csv")
```

[3]:
```
df_trans.shape
```

[3]: (590540, 394)

[4]:
```
df_id.shape
```

[4]: (144233, 41)

[5]:
```
df_trans.head()
```

[5]:

	TransactionID	isFraud	TransactionDT	TransactionAmt	ProductCD	card1	\
0	2987000	0	86400	68.5	W	13926	
1	2987001	0	86401	29.0	W	2755	
2	2987002	0	86469	59.0	W	4663	
3	2987003	0	86499	50.0	W	18132	
4	2987004	0	86506	50.0	H	4497	

	card2	card3	card4	card5	…	V330	V331	V332	V333	V334	V335	\
0	NaN	150.0	discover	142.0	…	NaN	NaN	NaN	NaN	NaN	NaN	
1	404.0	150.0	mastercard	102.0	…	NaN	NaN	NaN	NaN	NaN	NaN	
2	490.0	150.0	visa	166.0	…	NaN	NaN	NaN	NaN	NaN	NaN	
3	567.0	150.0	mastercard	117.0	…	NaN	NaN	NaN	NaN	NaN	NaN	
4	514.0	150.0	mastercard	102.0	…	0.0	0.0	0.0	0.0	0.0	0.0	

	V336	V337	V338	V339
0	NaN	NaN	NaN	NaN
1	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN

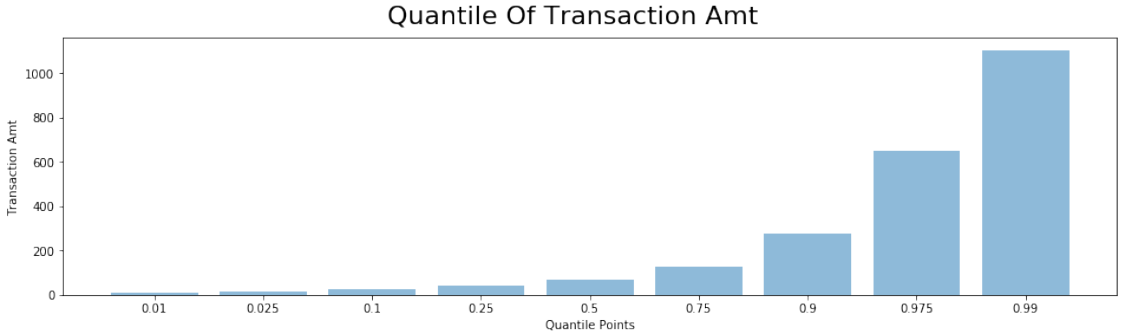
```
4  0.0   0.0   0.0   0.0

[5 rows x 394 columns]
```

[6]:
```
df_trans.tail()
```

[6]:
```
        TransactionID  isFraud  TransactionDT  TransactionAmt ProductCD  \
590535        3577535        0       15811047           49.00         W
590536        3577536        0       15811049           39.50         W
590537        3577537        0       15811079           30.95         W
590538        3577538        0       15811088          117.00         W
590539        3577539        0       15811131          279.95         W

        card1  card2  card3       card4  card5  … V330  V331  V332  V333  \
590535   6550    NaN  150.0        visa  226.0  …  NaN   NaN   NaN   NaN
590536  10444  225.0  150.0  mastercard  224.0  …  NaN   NaN   NaN   NaN
590537  12037  595.0  150.0  mastercard  224.0  …  NaN   NaN   NaN   NaN
590538   7826  481.0  150.0  mastercard  224.0  …  NaN   NaN   NaN   NaN
590539  15066  170.0  150.0  mastercard  102.0  …  NaN   NaN   NaN   NaN

        V334 V335 V336  V337  V338  V339
590535   NaN  NaN  NaN   NaN   NaN   NaN
590536   NaN  NaN  NaN   NaN   NaN   NaN
590537   NaN  NaN  NaN   NaN   NaN   NaN
590538   NaN  NaN  NaN   NaN   NaN   NaN
590539   NaN  NaN  NaN   NaN   NaN   NaN

[5 rows x 394 columns]
```

[7]:
```
df = df_trans['isFraud'].value_counts()

fraud =  (100 * float(df[1])/float(df[0]+df[1]))
noFraud =  (100 * float(df[0])/float(df[0]+df[1]))

# Data to plot
labels = 'Fraud', 'No Fraud'
sizes = [fraud, noFraud]
colors = ['red', 'lightskyblue']
explode = (0.15, 0,)  # explode 1st slice
plt.figure(figsize=(16,4))
# Plot
plt.pie(sizes, explode=explode, labels=labels, colors=colors,
autopct='%1.1f%%', shadow=False, startangle=140)
plt.axis('equal')
plt.suptitle('Fraud Vs No Fraud', fontsize=22)
plt.show()
```

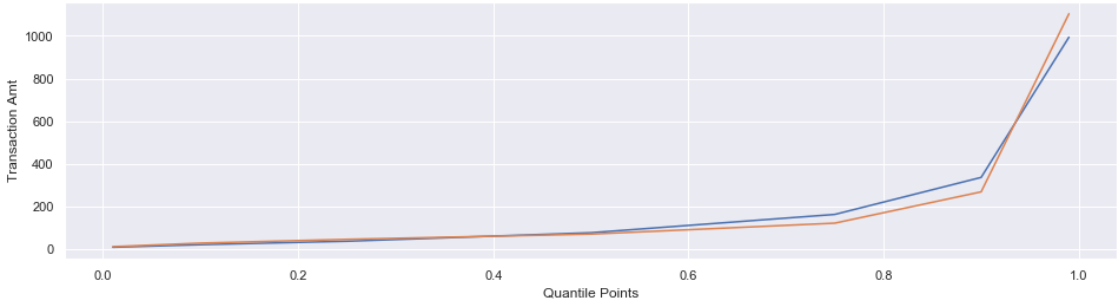
[8]:
```
df_trans['TransactionAmt'] = df_trans['TransactionAmt'].astype(float)

x = [.01, .025, .1, .25, .5, .75, .9, .975, .99]
y = np.array(df_trans['TransactionAmt'].quantile([.01, .025, .1, .25, .5, .75, .
 ↪9, .975, .99]))
plt.figure(figsize=(16,4))
y_pos = np.arange(len(x))
plt.bar(y_pos, y, align='center', alpha=0.5)
plt.xticks(y_pos, x)
plt.ylabel('Transaction Amt')
plt.xlabel('Quantile Points')
plt.suptitle('Quantile Of Transaction Amt', fontsize=22)
plt.show()
```



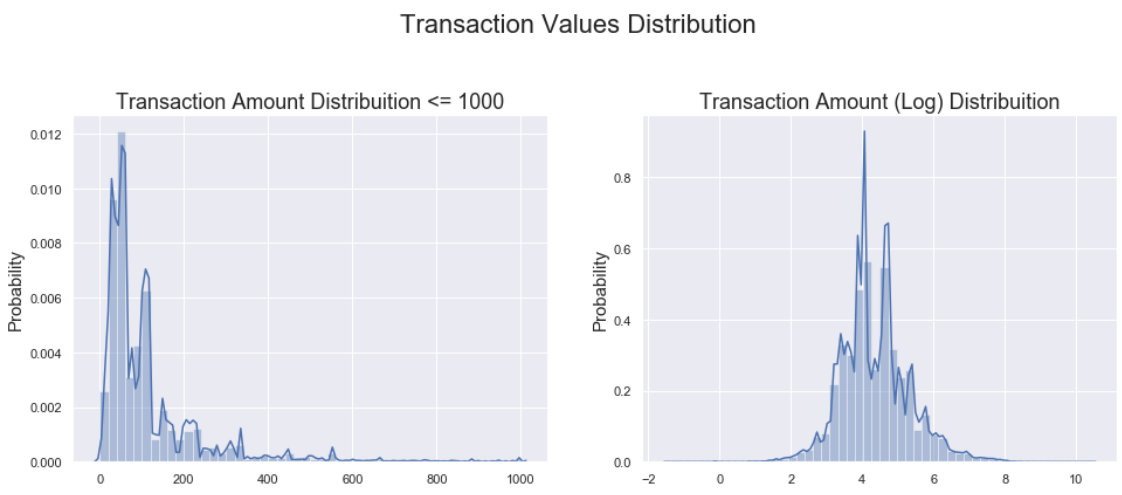
[ ]:
```

```

[9]:
```
df = pd.concat([df_trans[df_trans['isFraud'] == 1]['TransactionAmt']\
                .quantile([.01, .1, .25, .5, .75, .9, .99])\
                .reset_index(),
               df_trans[df_trans['isFraud'] == 0]['TransactionAmt']\
                .quantile([.01, .1, .25, .5, .75, .9, .99])\
                .reset_index()],
```

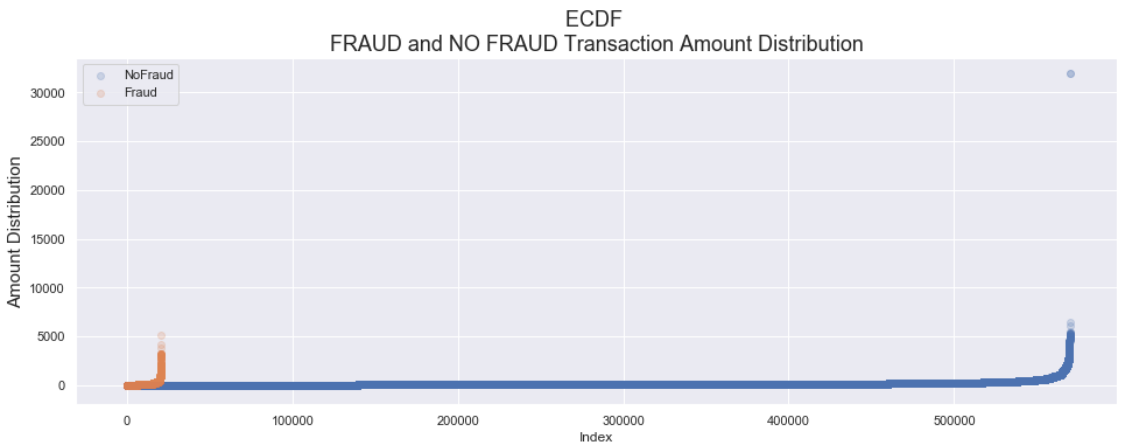
```
                axis=1, keys=['Fraud', "No Fraud"])

df['FraudIndex'] = df['Fraud']['index']
df['FraudTransactionAmt'] = df['Fraud']['TransactionAmt']
df['NoFraudIndex'] = df['No Fraud']['index']
df['NoFraudTransactionAmt'] = df['No Fraud']['TransactionAmt']

sns.set()
plt.figure(figsize=(16,4))
ax = sns.lineplot(x="FraudIndex", y="FraudTransactionAmt", data=df)
ax = sns.lineplot(x="NoFraudIndex", y="NoFraudTransactionAmt", data=df).
 ↪set(xlabel="Quantile Points",

 ↪ylabel = "Transaction Amt")
```



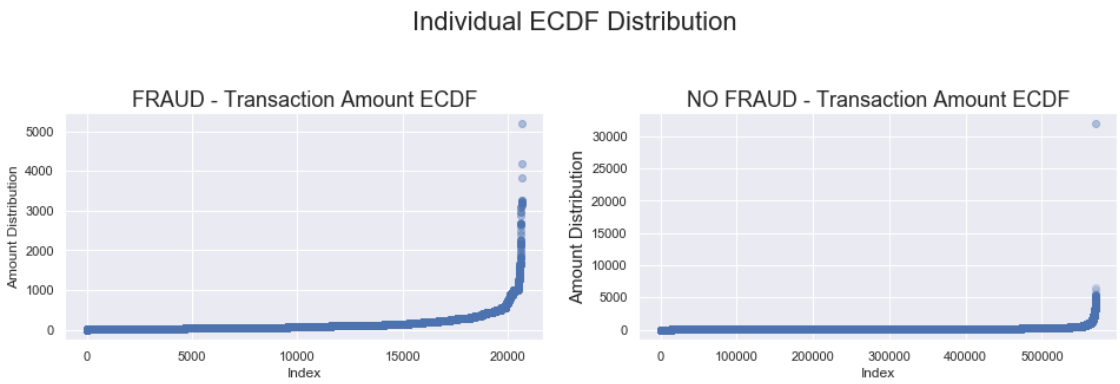
[10]:
```
plt.figure(figsize=(16,12))
plt.suptitle('Transaction Values Distribution', fontsize=22)
plt.subplot(221)
g = sns.distplot(df_trans[df_trans['TransactionAmt'] <= 1000]['TransactionAmt'])
g.set_title("Transaction Amount Distribuition <= 1000", fontsize=18)
g.set_xlabel("")
g.set_ylabel("Probability", fontsize=15)

plt.subplot(222)
g1 = sns.distplot(np.log(df_trans['TransactionAmt']))
g1.set_title("Transaction Amount (Log) Distribuition", fontsize=18)
g1.set_xlabel("")
g1.set_ylabel("Probability", fontsize=15)

plt.show()
```



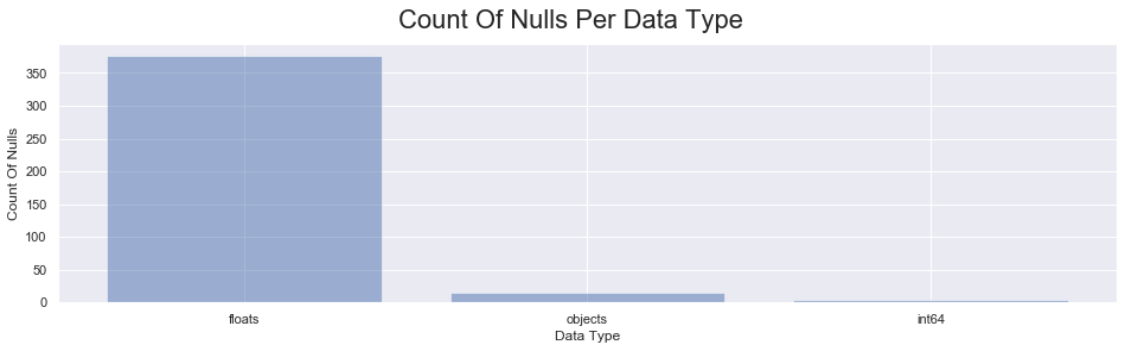
[11]:
```
plt.figure(figsize=(16,12))
plt.subplot(212)
g4 = plt.scatter(range(df_trans[df_trans['isFraud'] == 0].shape[0]),
                 np.sort(df_trans[df_trans['isFraud'] == 0]['TransactionAmt'].
 ↪values),
                 label='NoFraud', alpha=.2)
g4 = plt.scatter(range(df_trans[df_trans['isFraud'] == 1].shape[0]),
                 np.sort(df_trans[df_trans['isFraud'] == 1]['TransactionAmt'].
 ↪values),
                 label='Fraud', alpha=.2)
g4= plt.title("ECDF \nFRAUD and NO FRAUD Transaction Amount Distribution",␣
 ↪fontsize=18)
g4 = plt.xlabel("Index")
g4 = plt.ylabel("Amount Distribution", fontsize=15)
g4 = plt.legend()
plt.show()
```



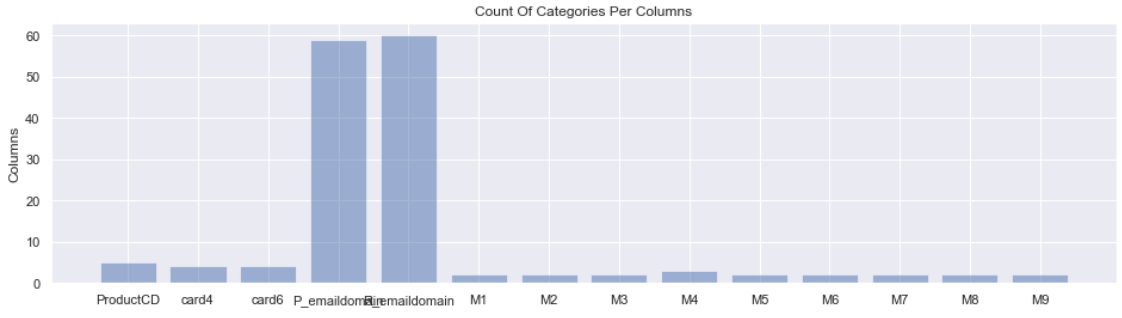
[12]:
```
plt.figure(figsize=(16,12))
plt.subplot(321)
g = plt.scatter(range(df_trans[df_trans['isFraud'] == 1].shape[0]),
                 np.sort(df_trans[df_trans['isFraud'] == 1]['TransactionAmt'].
 ↪values),
                label='isFraud', alpha=.4)
plt.title("FRAUD - Transaction Amount ECDF", fontsize=18)
plt.xlabel("Index")
plt.ylabel("Amount Distribution", fontsize=12)

plt.subplot(322)
g1 = plt.scatter(range(df_trans[df_trans['isFraud'] == 0].shape[0]),
                 np.sort(df_trans[df_trans['isFraud'] == 0]['TransactionAmt'].
 ↪values),
                 label='NoFraud', alpha=.2)
g1= plt.title("NO FRAUD - Transaction Amount ECDF", fontsize=18)
g1 = plt.xlabel("Index")
g1 = plt.ylabel("Amount Distribution", fontsize=15)

plt.suptitle('Individual ECDF Distribution', fontsize=22)
plt.show()
```



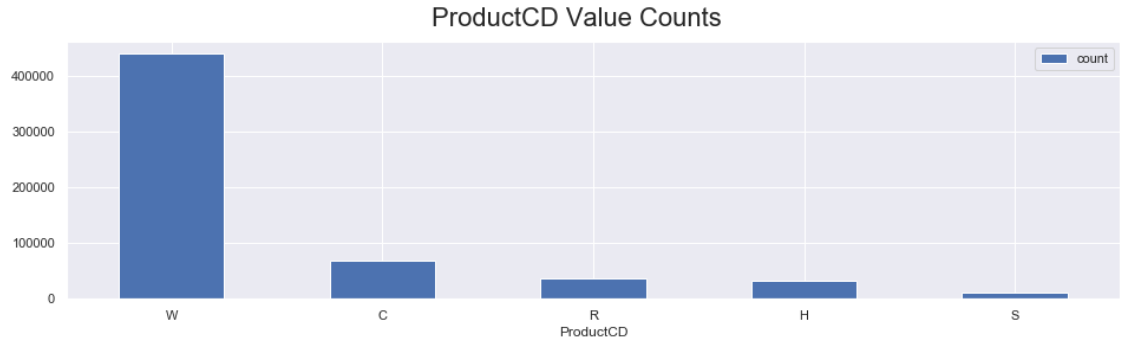
[13]:
```
df = df_trans.dtypes.value_counts()
plt.figure(figsize=(16,4))
cols = ['floats', 'objects', 'int64']
y_pos = np.arange(len(cols))
plt.bar(y_pos, df, align='center', alpha=0.5)
plt.xticks(y_pos, cols)
plt.ylabel('Count Of Nulls')
plt.xlabel('Data Type')
plt.suptitle('Count Of Nulls Per Data Type', fontsize=22)
plt.show()
```

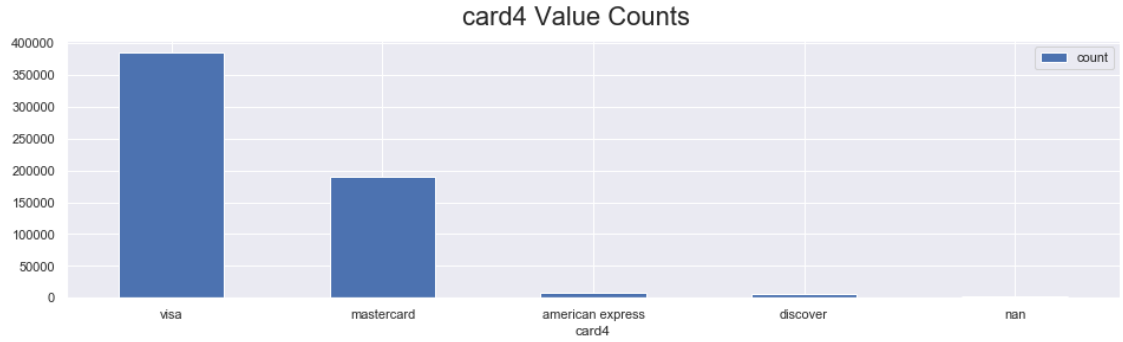


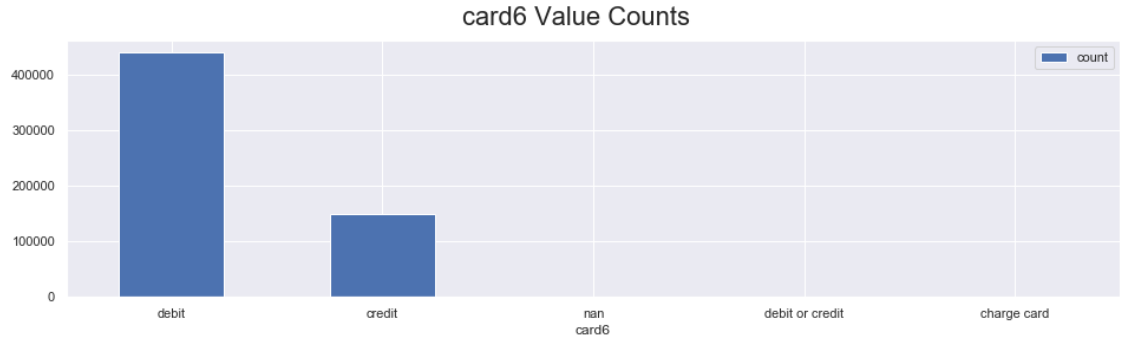
[14]:
```
ndf = df_trans[df_trans._get_numeric_data().columns]
cdf = df_trans.drop(df_trans._get_numeric_data().columns , axis=1)

cols = []
counts = []

for (columnName, columnData) in cdf.iteritems():
    cols.append(columnName)
    counts.append(len(columnData.value_counts()))


plt.figure(figsize=(16,4))

y_pos = np.arange(len(cols))
plt.bar(y_pos, counts, align='center', alpha=0.5)
plt.xticks(y_pos, cols)
plt.ylabel('Categories Count')
plt.ylabel('Columns')

plt.title('Count Of Categories Per Columns')
plt.show()
```

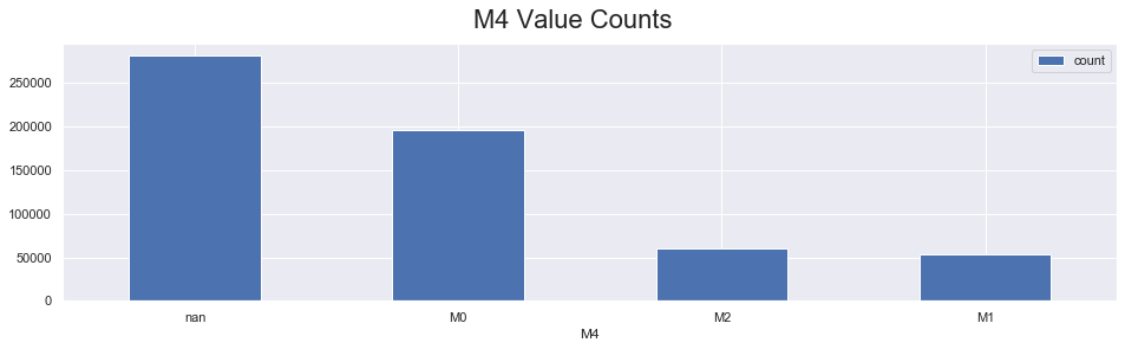


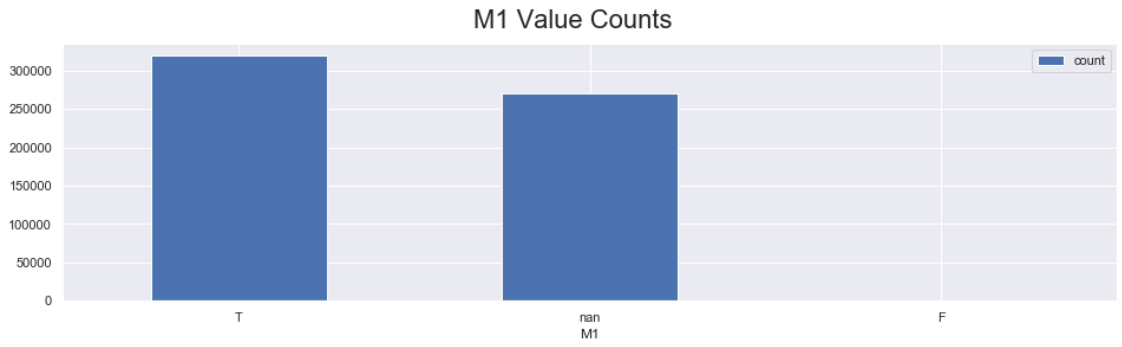
[42]:
```
for i in ['ProductCD', 'card4', 'card6', 'M4', 'M1', 'M2', 'M3', 'M5', 'M6',␣
↪'M7', 'M8', 'M9']:
    feature_count = df_trans[i].value_counts(dropna=False).reset_index().
↪rename(columns={i: 'count', 'index': i})
    ax = feature_count.plot.bar(x=i, y='count', rot=0, figsize=(16,4))
    plt.suptitle((i + ' Value Counts'), fontsize=22)
```

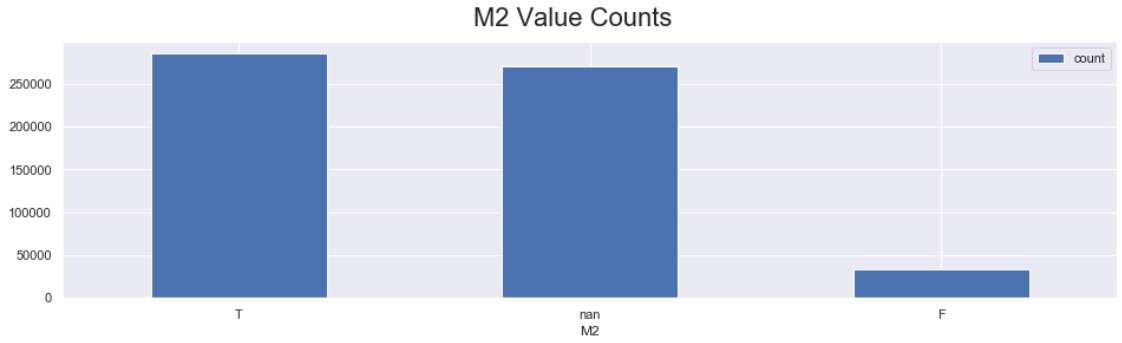


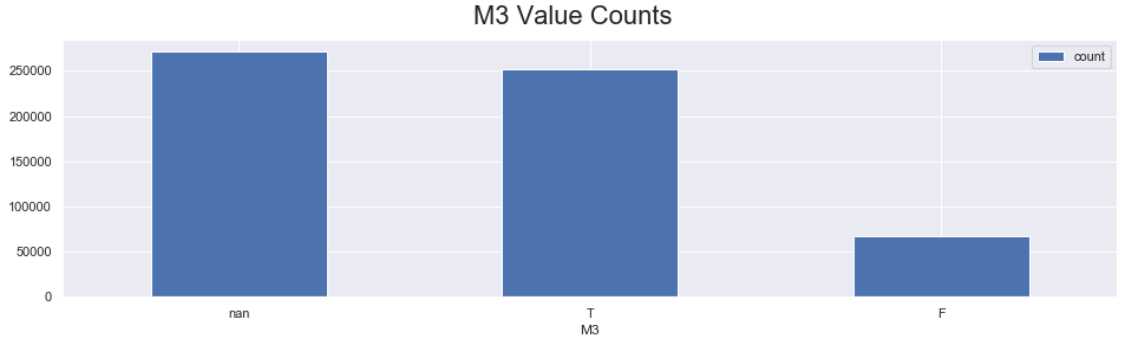


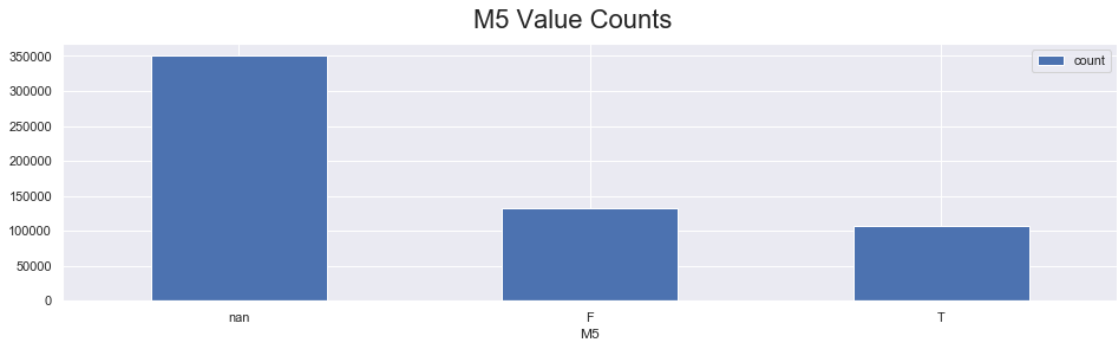


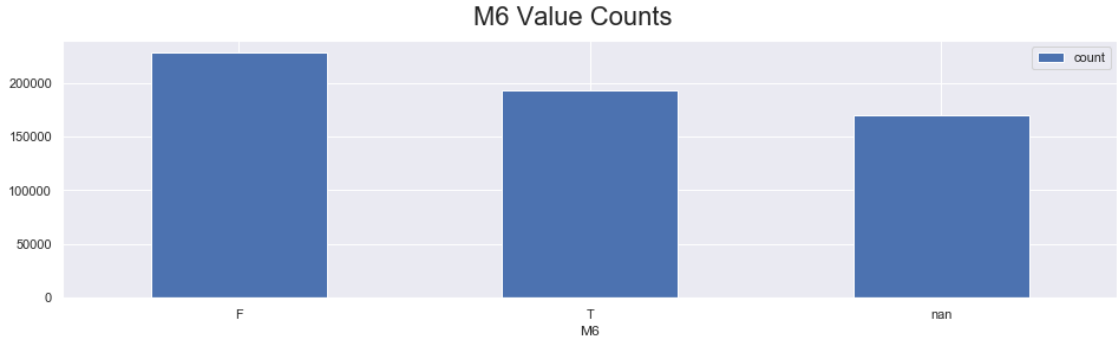

M4 Value Counts
count
250000
200000
150000
100000
50000
0
nan
M0
M2
M1
M4

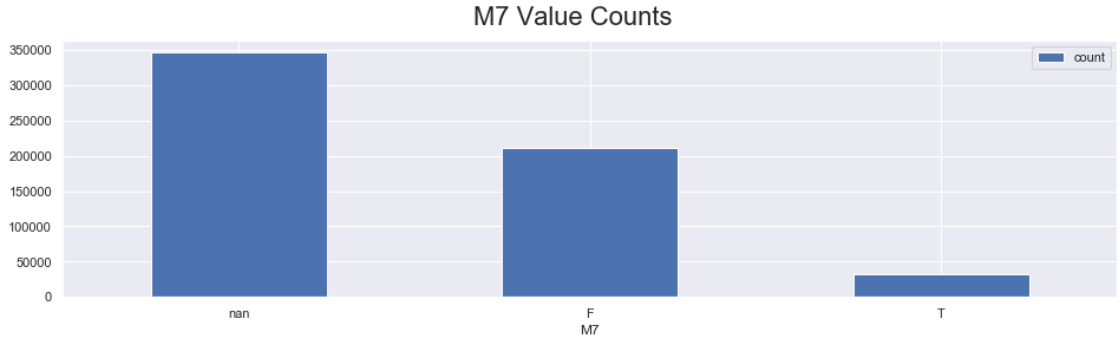

M1 Value Counts
count
300000
250000
200000
150000
100000
50000
0
T
nan
F
M1

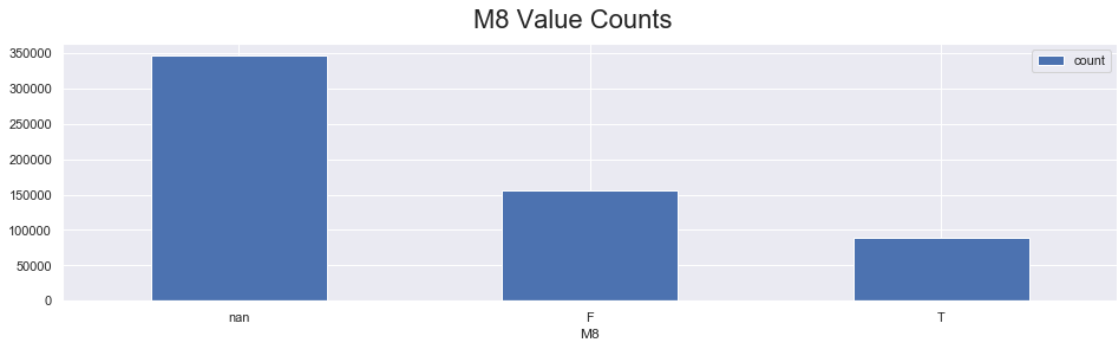

M2 Value Counts
count
250000
200000
150000
100000
50000
0
T
nan
F
M2

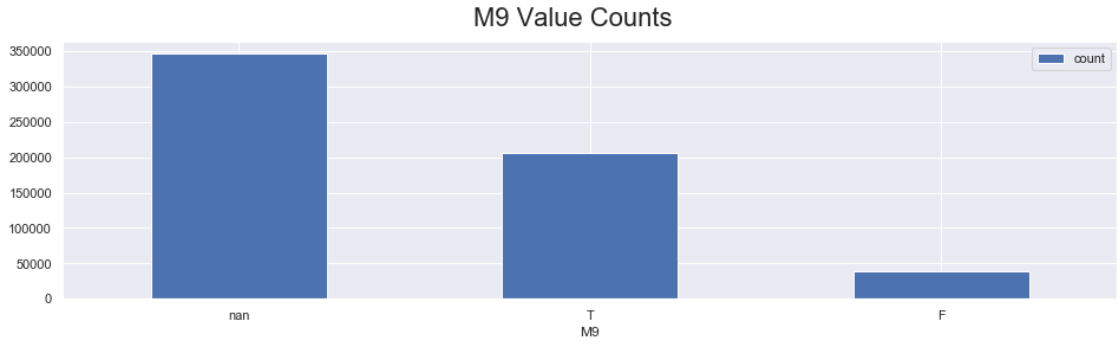

M3 Value Counts
count
250000
200000
150000
100000
50000
0
nan
T
F
M3


M5 Value Counts
count
350000
300000
250000
200000
150000
100000
50000
0
nan
F
T
M5

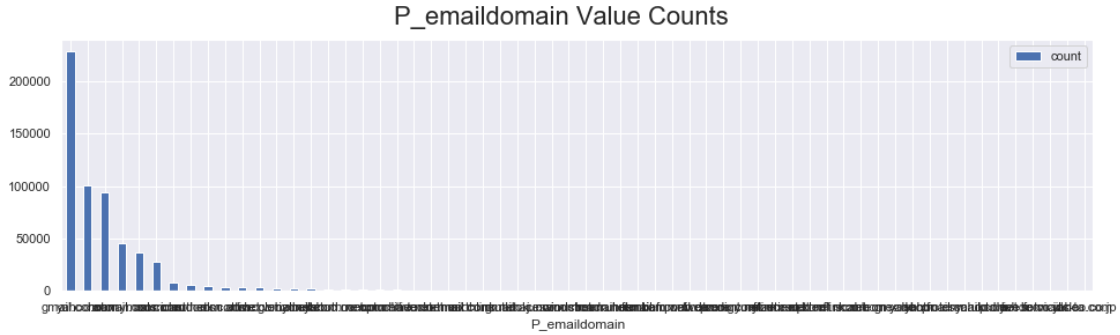

M6 Value Counts
count
200000
150000
100000
50000
0
F
T
nan
M6

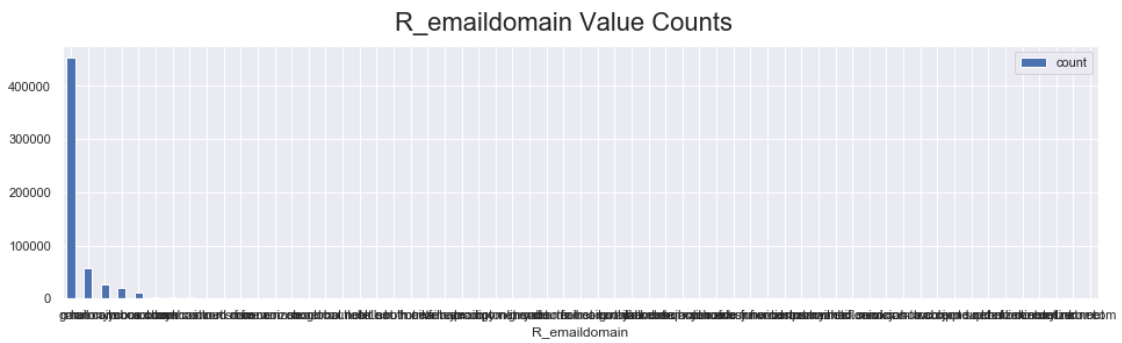

M7 Value Counts
count
350000
300000
250000
200000
150000
100000
50000
0
nan
F
T
M7

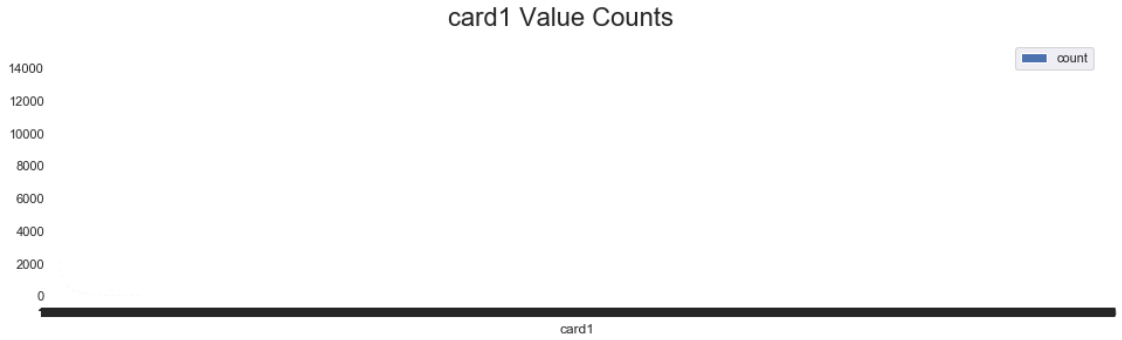


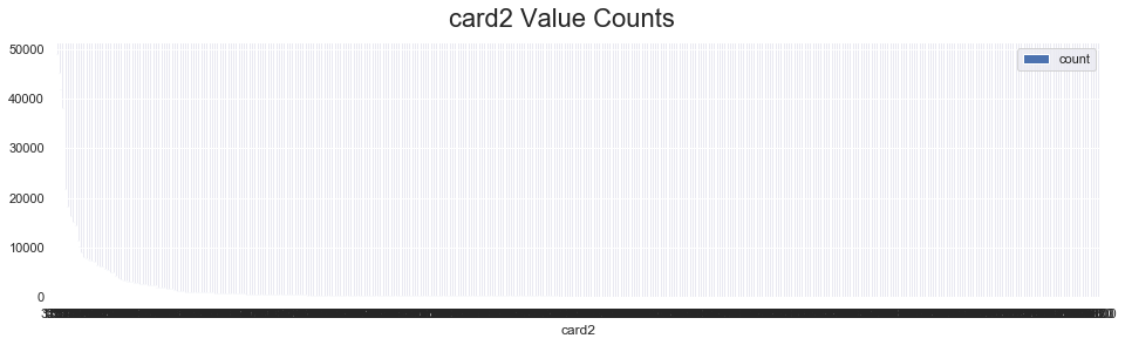


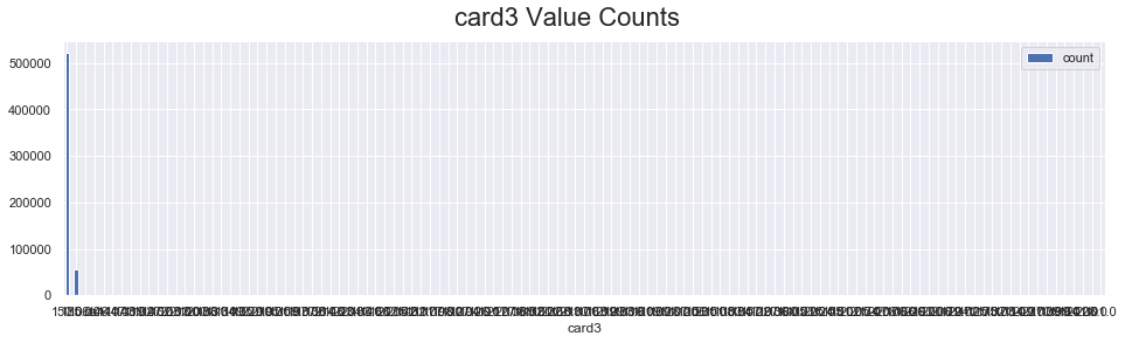
[43]:
```
for i in ['P_emaildomain', 'R_emaildomain', 'card1', 'card2', 'card3', ␣
↪'card5', 'addr1', 'addr2']:
    feature_count = df_trans[i].value_counts(dropna=False).reset_index().
↪rename(columns={i: 'count', 'index': i})
    ax = feature_count.plot.bar(x=i, y='count', rot=0, figsize=(16,4))
    plt.suptitle((i + ' Value Counts'), fontsize=22)
```

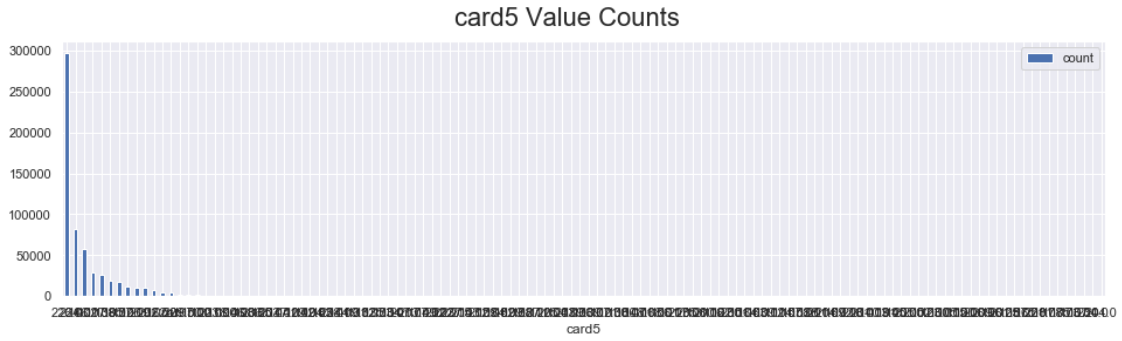


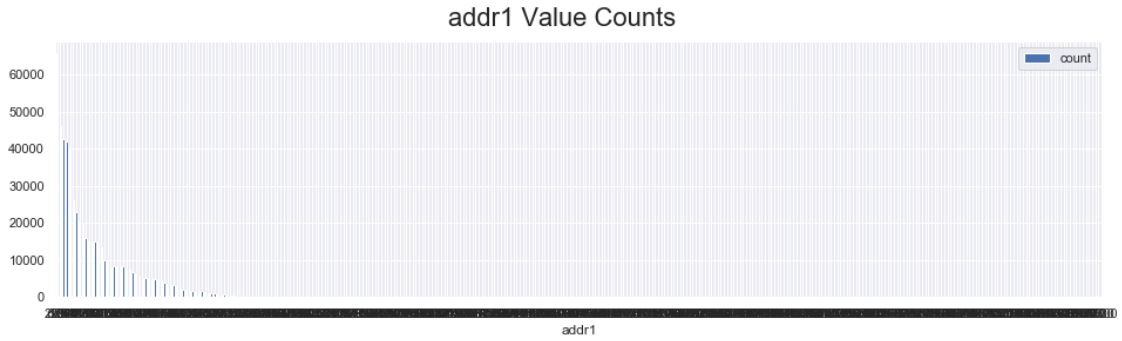

R_emaildomain Value Counts
count
400000
300000
200000
100000
0
R_emaildomain

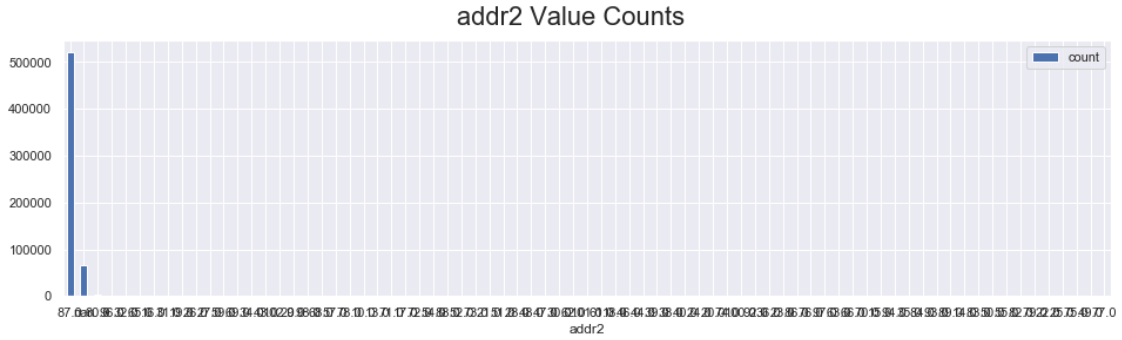

card1 Value Counts
count
14000
12000
10000
8000
6000
4000
2000
0
card1


card2 Value Counts
count
50000
40000
30000
20000
10000
0
card2


card3 Value Counts
count
500000
400000
300000
200000
100000
0
card3





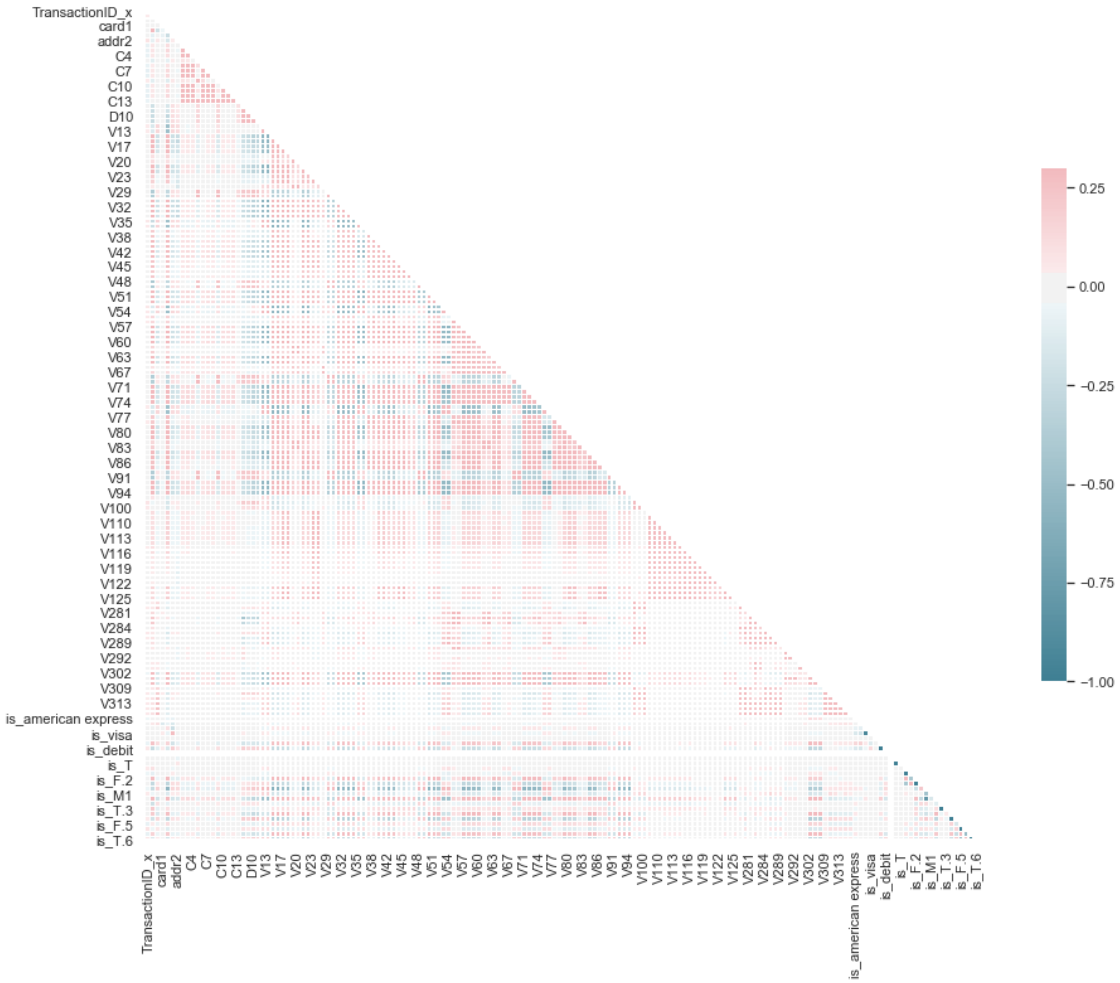


[2]:
```
df = pd.read_csv("./input/processed/under-sampled.csv")
```

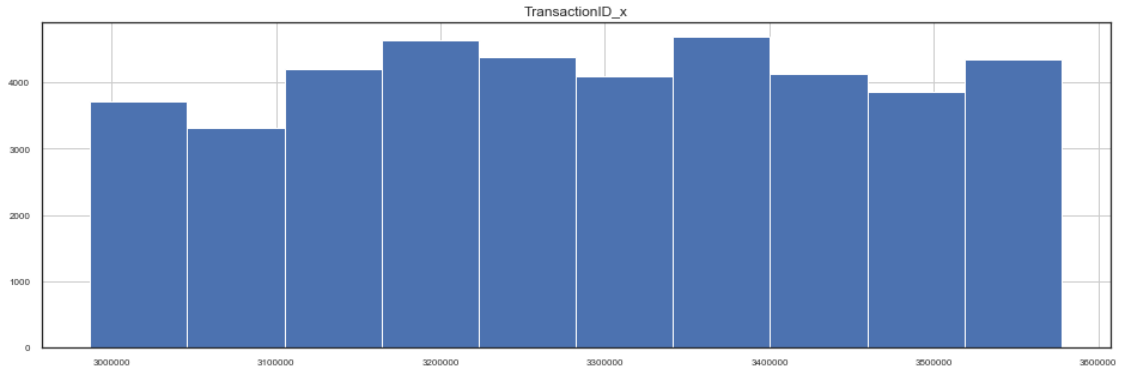
[3]:
```
sns.set(style="white")
corr = df.corr()
# Generate a mask for the upper triangle
mask = np.zeros_like(corr, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True
```

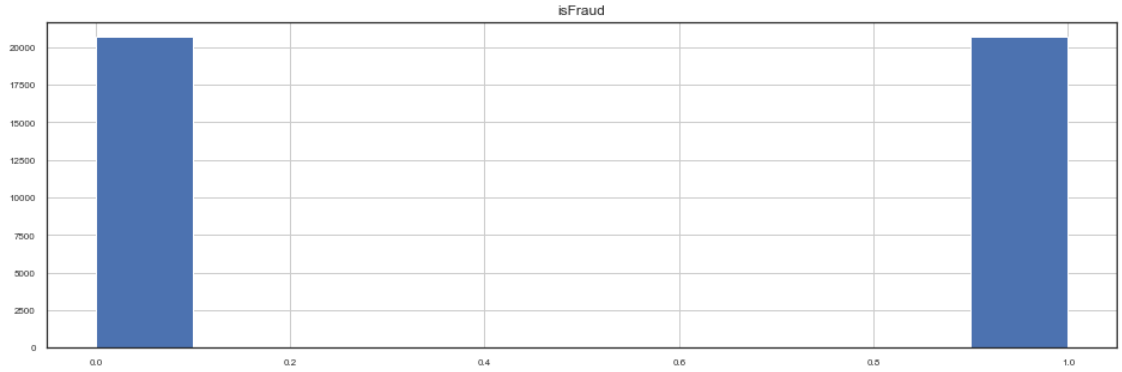
```
# Set up the matplotlib figure
f, ax = plt.subplots(figsize=(15, 15))

# Generate a custom diverging colormap
cmap = sns.diverging_palette(220, 10, as_cmap=True)

# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(corr, mask=mask, cmap=cmap, vmax=.3, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .5})
```

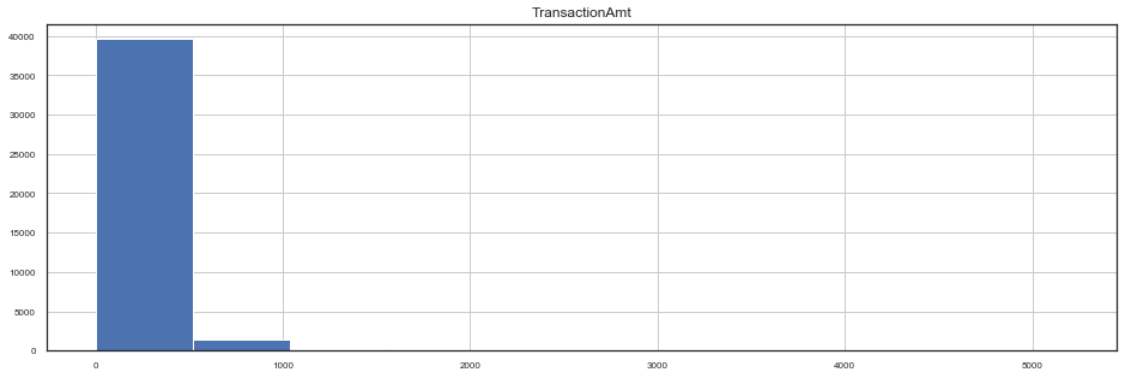
[3]: <matplotlib.axes._subplots.AxesSubplot at 0x1f0d3a32208>

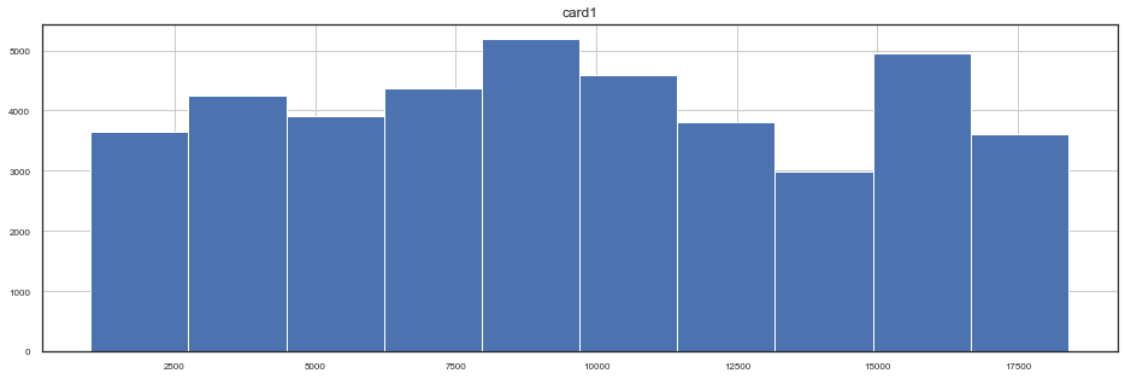


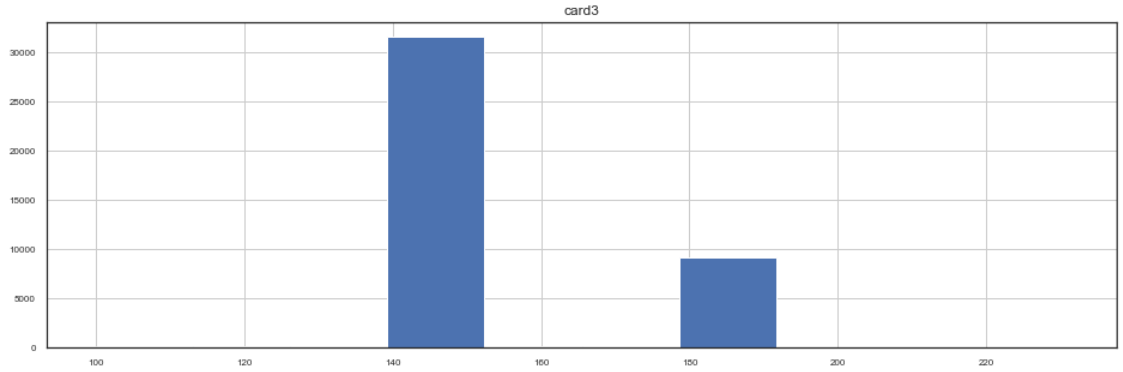
[13]:
```
for i in df.columns:
    df.hist(column=i, figsize=(16, 5), xlabelsize=8, ylabelsize=8)
```

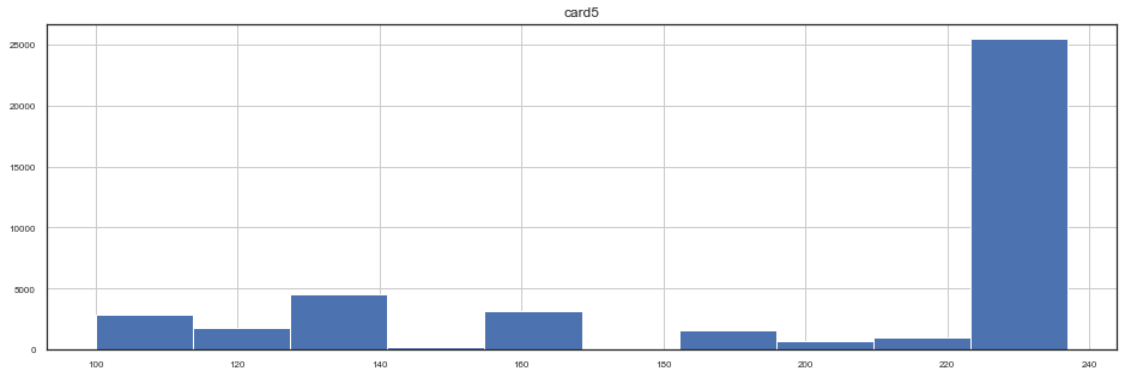

TransactionID_x
4000
3000
2000
1000
0
3000000
3100000
3200000
3300000
3400000
3500000
3600000

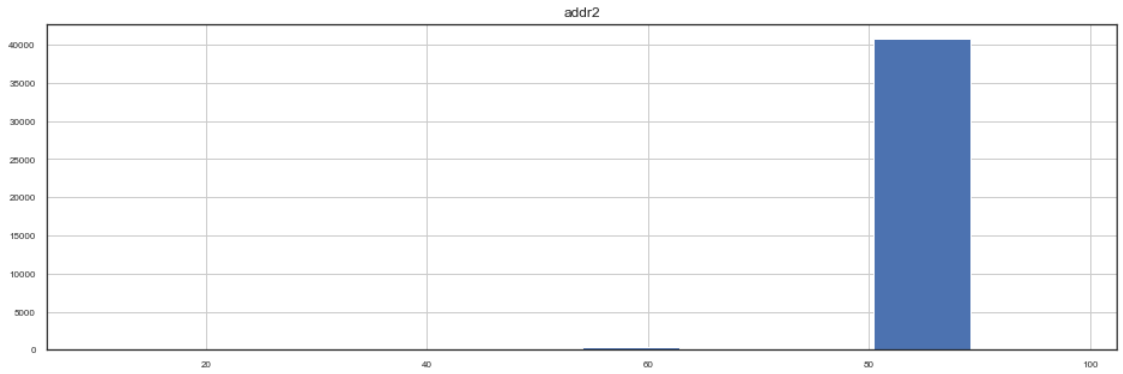

isFraud
20000
17500
15000
12500
10000
7500
5000
2500
0
0.0
0.2
0.4
0.6
0.8
1.0

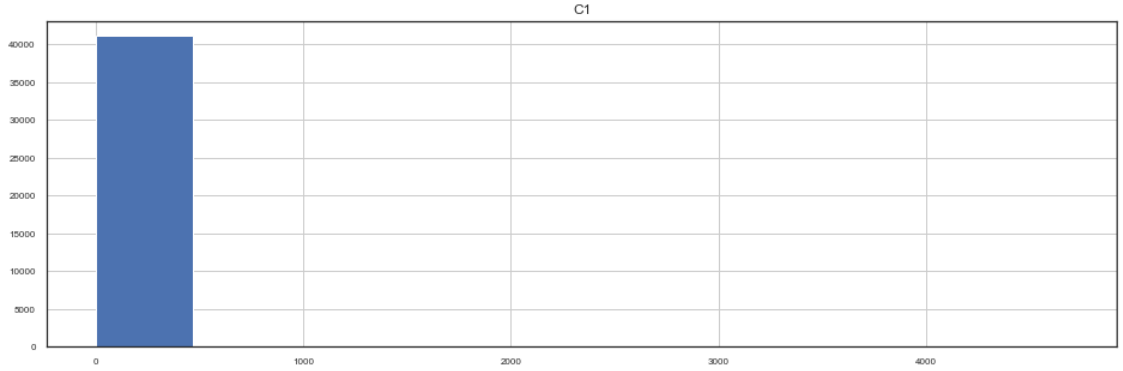

TransactionAmt
40000
35000
30000
25000
20000
15000
10000
5000
0
0
1000
2000
3000
4000
5000

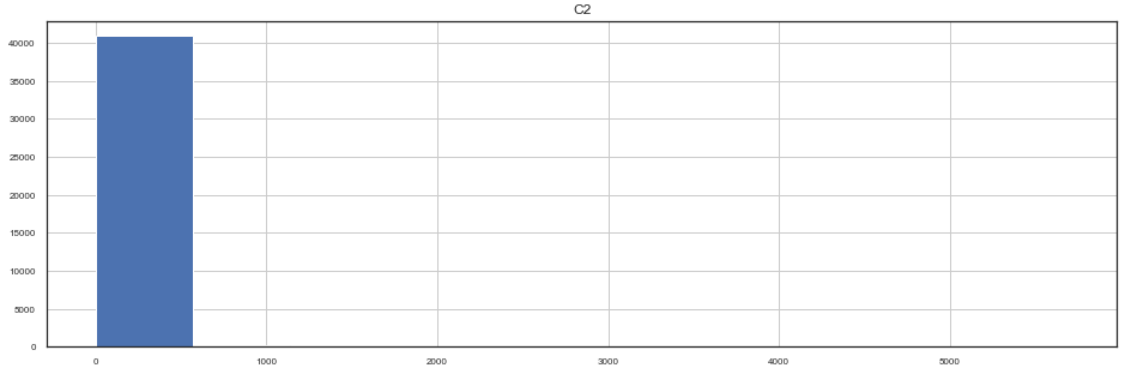

card1
5000
4000
3000
2000
1000
0
2500
5000
7500
10000
12500
15000
17500

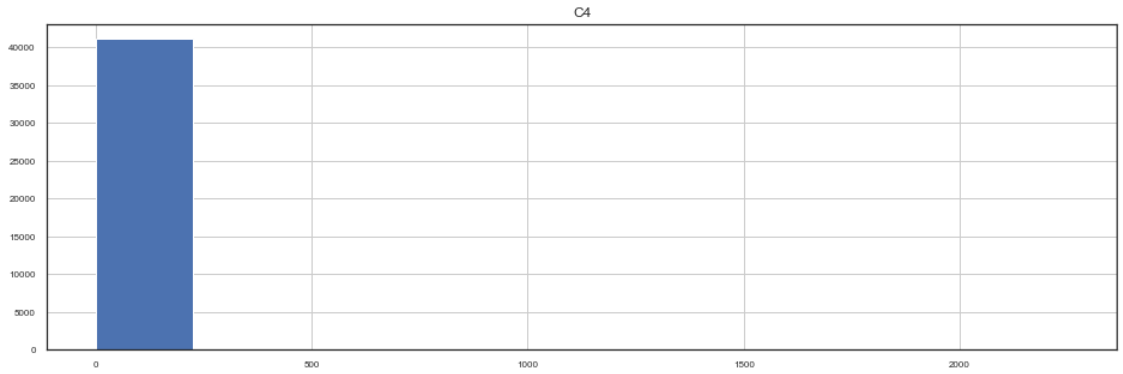

card3
30000
25000
20000
15000
10000
5000
0
100
120
140
160
180
200
220

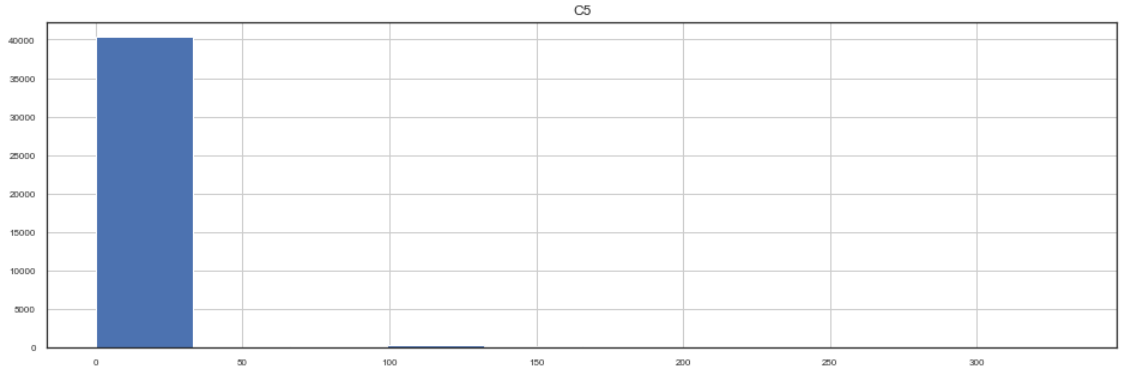

card5
25000
20000
15000
10000
5000
0
100
120
140
160
180
200
220
240

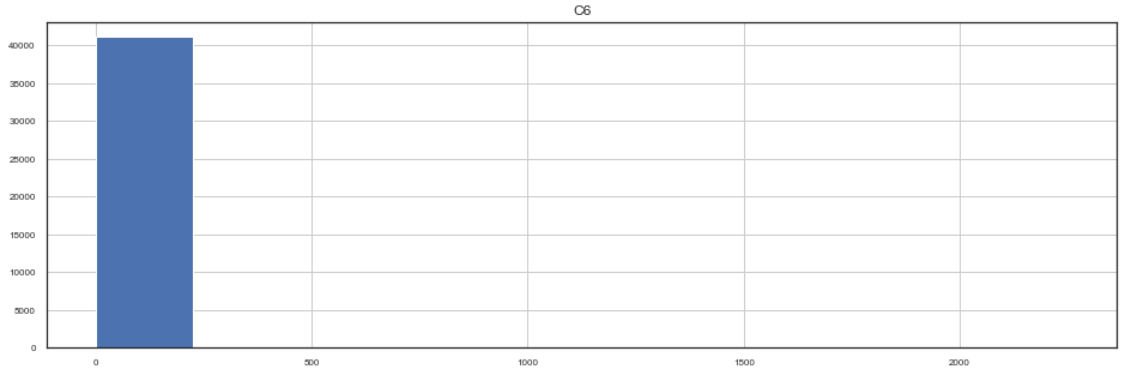

addr2
40000
35000
30000
25000
20000
15000
10000
5000
0
20
40
60
80
100

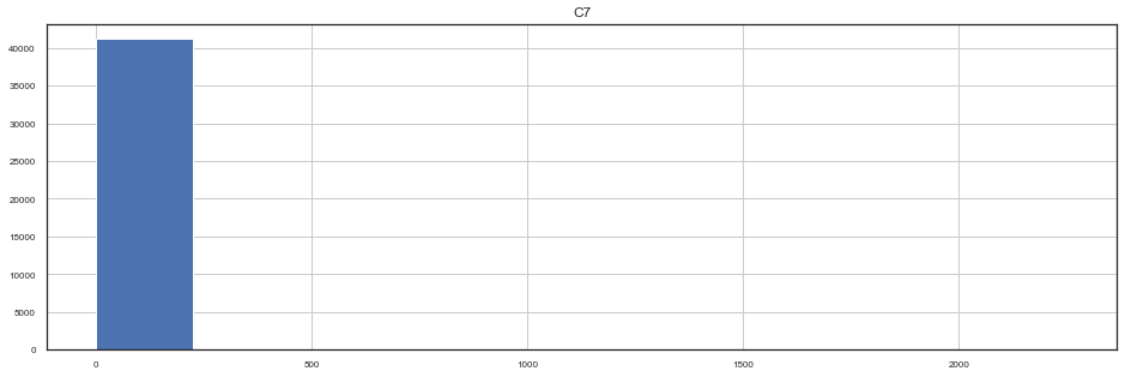

C1
40000
35000
30000
25000
20000
15000
10000
5000
0
0
1000
2000
3000
4000

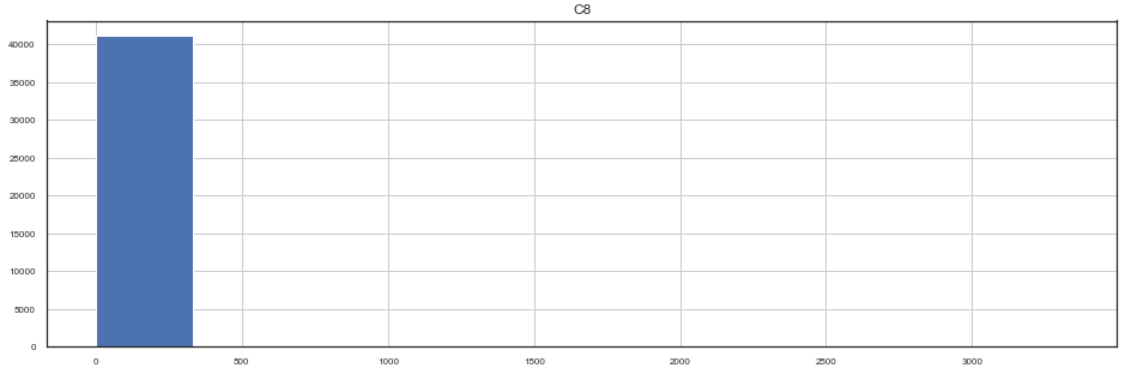

C2
40000
35000
30000
25000
20000
15000
10000
5000
0
0
1000
2000
3000
4000
5000

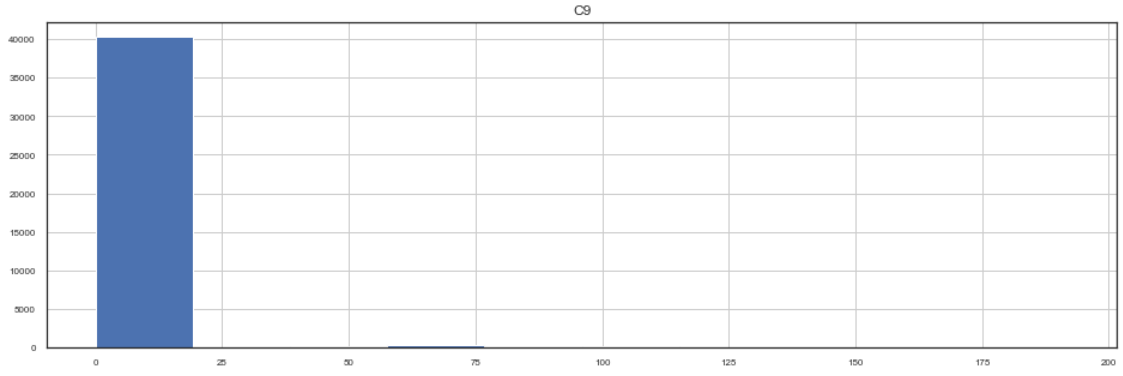

C4
40000
35000
30000
25000
20000
15000
10000
5000
0
0
500
1000
1500
2000

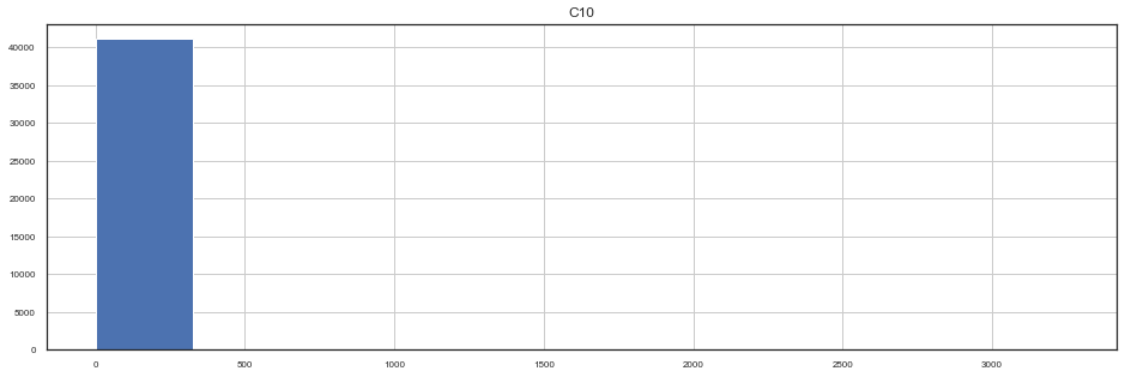

C5
40000
35000
30000
25000
20000
15000
10000
5000
0
0
50
100
150
200
250
300

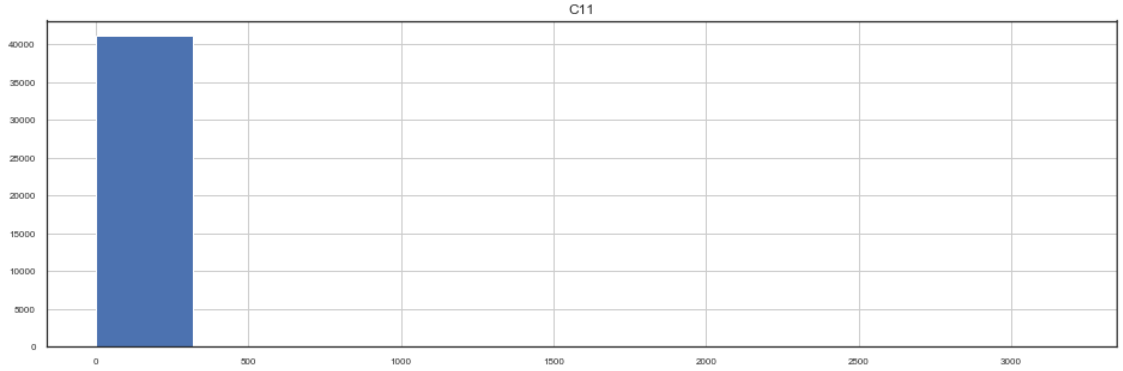

C6
40000
35000
30000
25000
20000
15000
10000
5000
0
0
500
1000
1500
2000

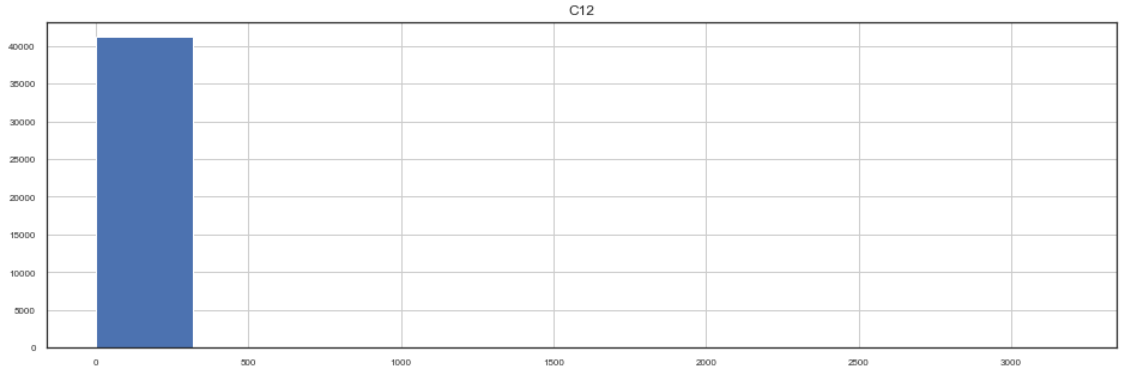

C7
40000
35000
30000
25000
20000
15000
10000
5000
0
0
500
1000
1500
2000

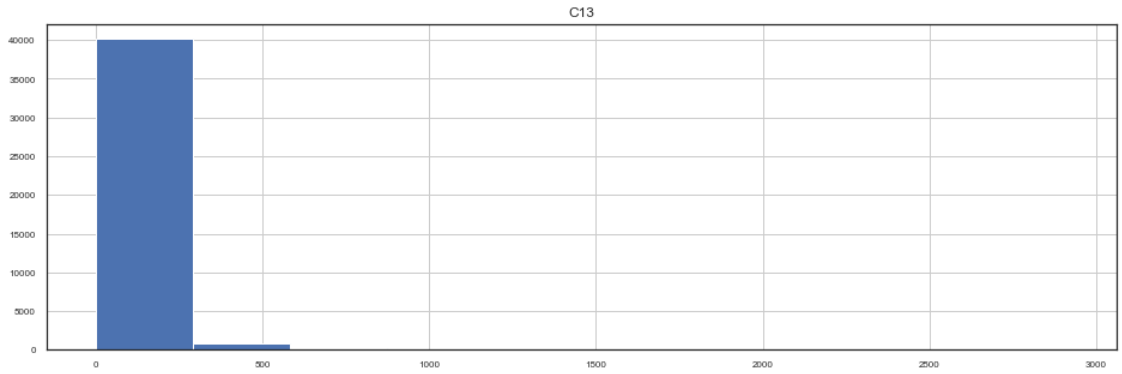

C8
40000
35000
30000
25000
20000
15000
10000
5000
0
0
500
1000
1500
2000
2500
3000

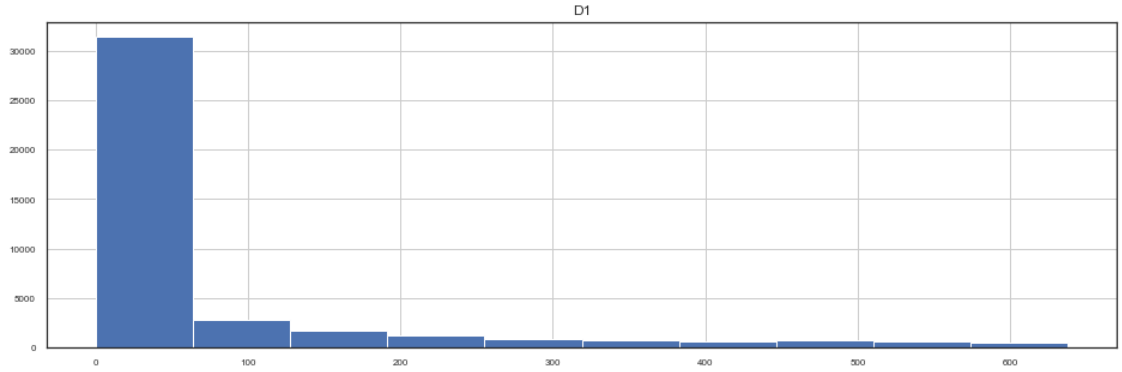

C9
40000
35000
30000
25000
20000
15000
10000
5000
0
0
25
50
75
100
125
150
175
200

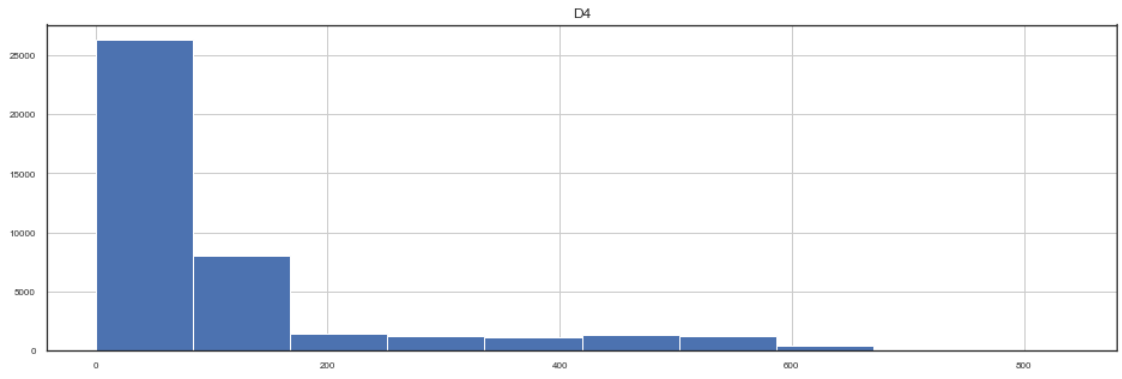

C10
40000
35000
30000
25000
20000
15000
10000
5000
0
0
500
1000
1500
2000
2500
3000

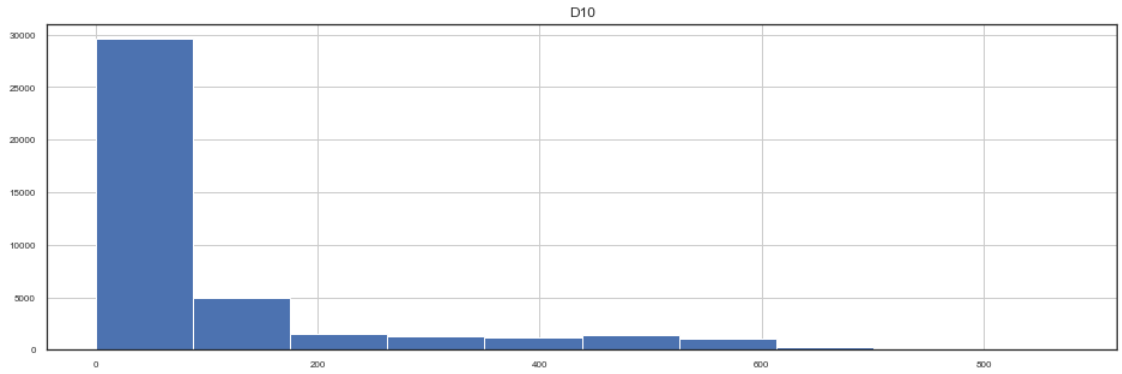

C11
40000
35000
30000
25000
20000
15000
10000
5000
0
0
500
1000
1500
2000
2500
3000

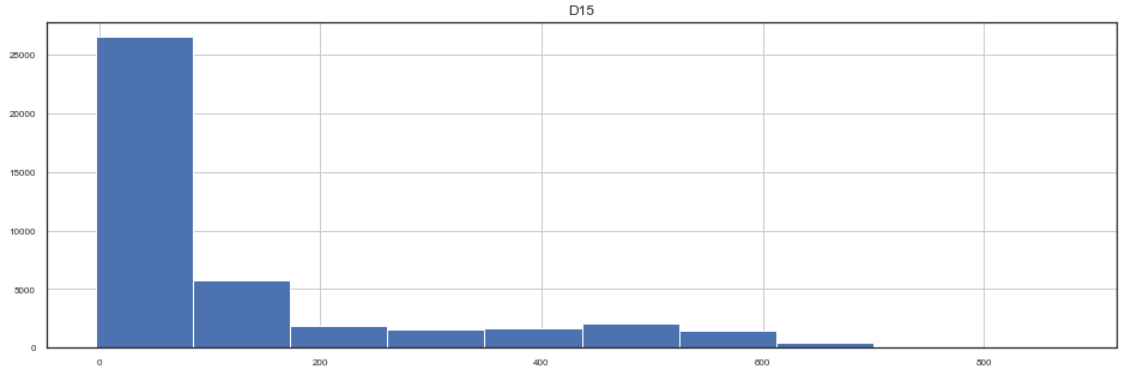

C12
40000
35000
30000
25000
20000
15000
10000
5000
0
0
500
1000
1500
2000
2500
3000

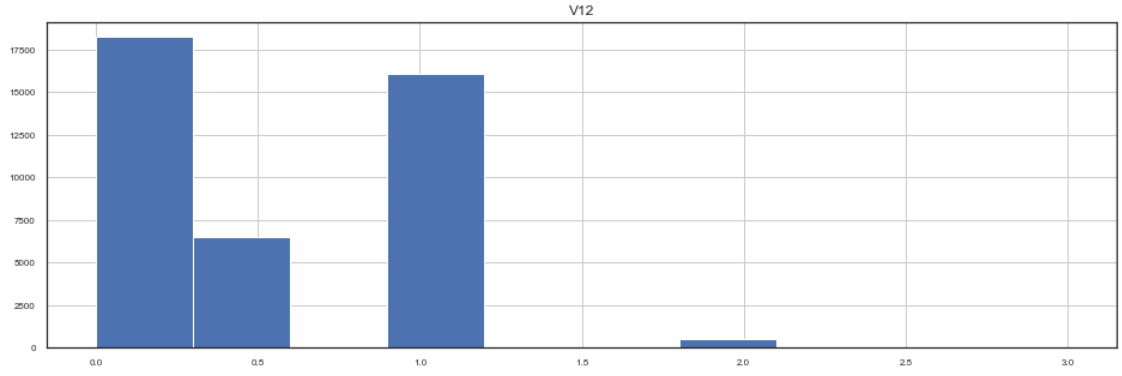

C13
40000
35000
30000
25000
20000
15000
10000
5000
0
0
500
1000
1500
2000
2500
3000

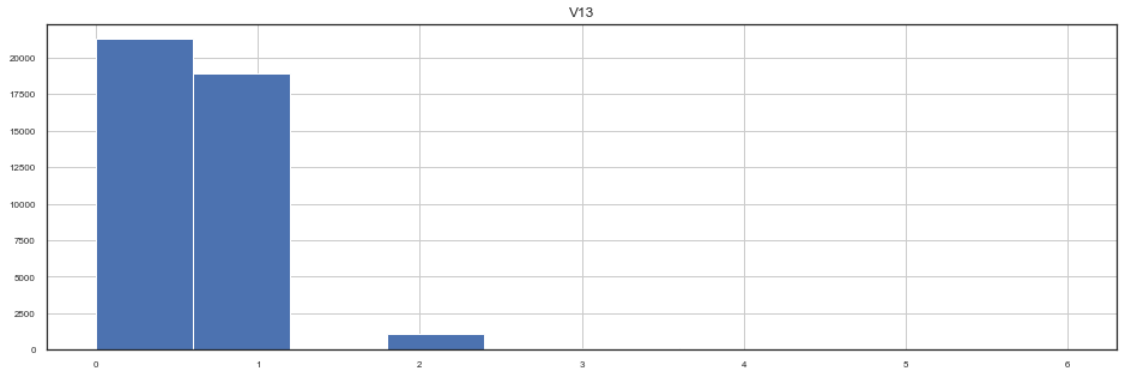

D1
30000
25000
20000
15000
10000
5000
0
0
100
200
300
400
500
600

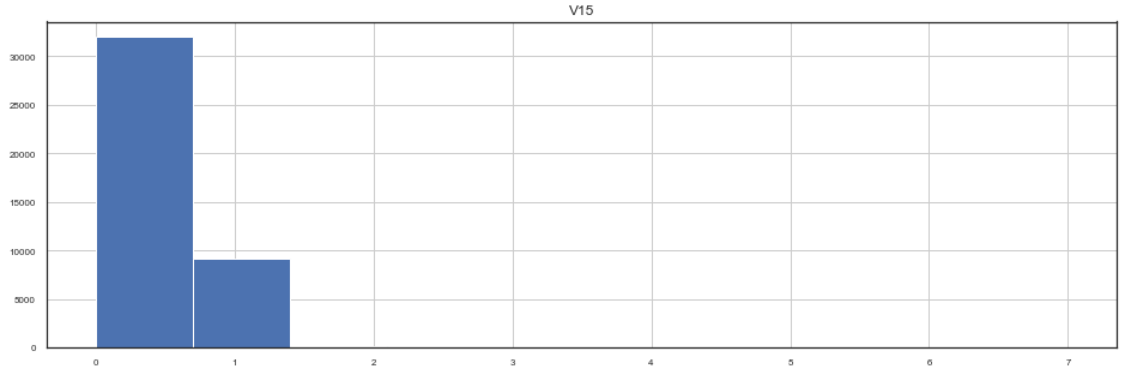

D4
25000
20000
15000
10000
5000
0
0
200
400
600
800

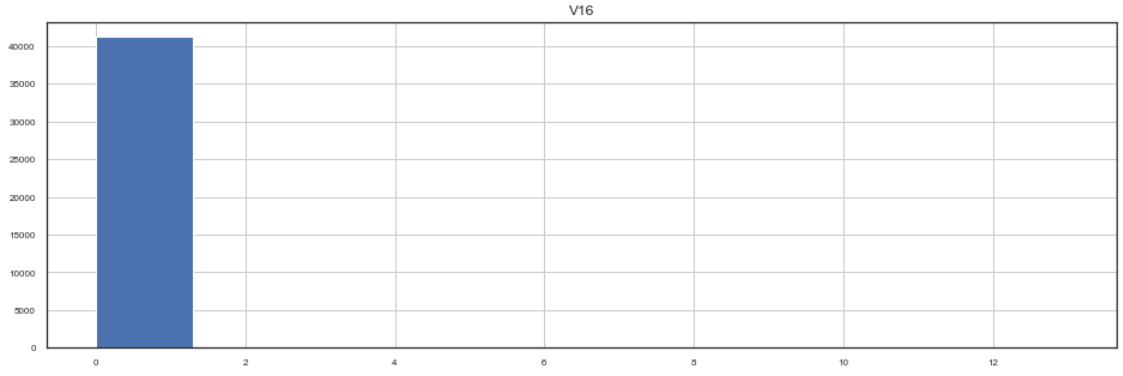

D10
30000
25000
20000
15000
10000
5000
0
0
200
400
600
800

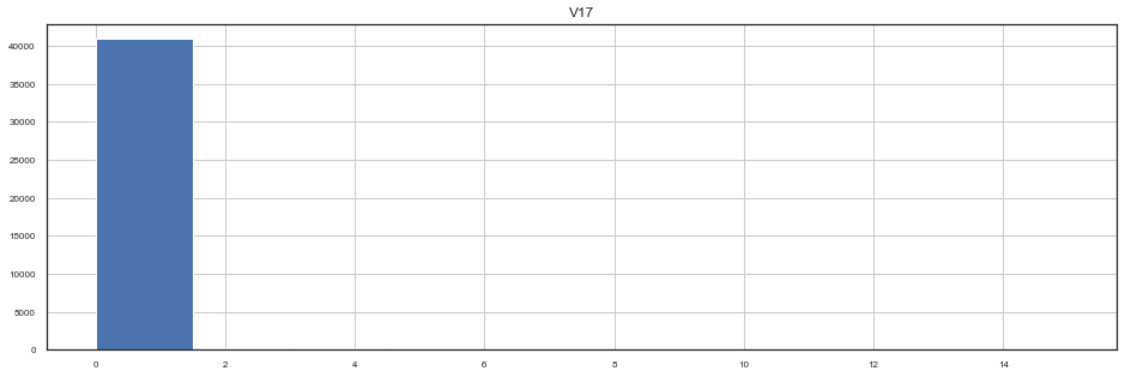

D15
25000
20000
15000
10000
5000
0
0
200
400
600
800

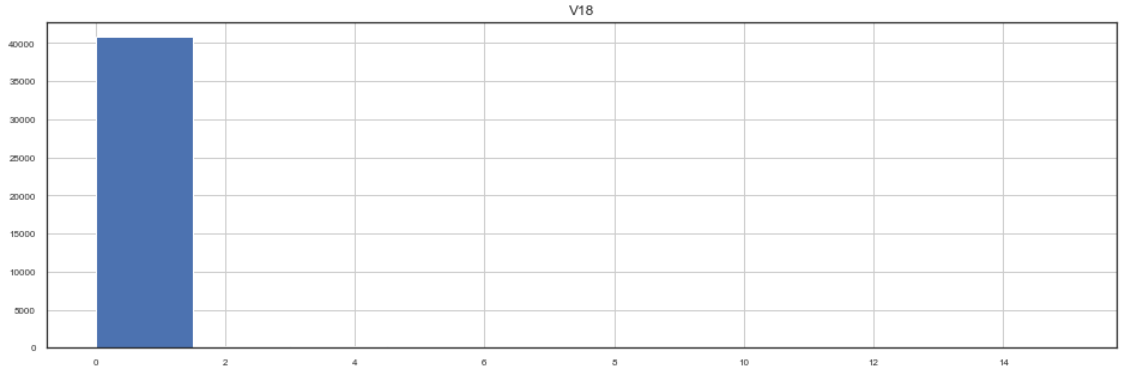

V12
17500
15000
12500
10000
7500
5000
2500
0
0.0
0.5
1.0
1.5
2.0
2.5
3.0

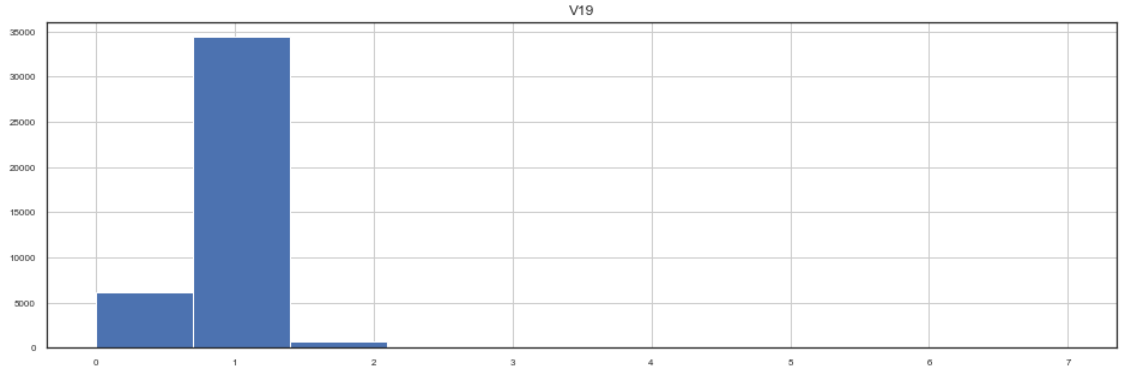

V13
20000
17500
15000
12500
10000
7500
5000
2500
0
0
1
2
3
4
5
6

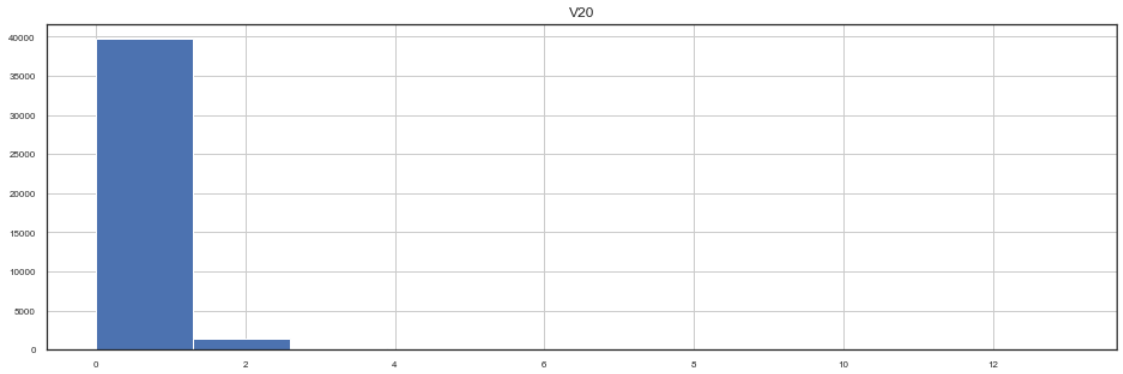

V15
30000
25000
20000
15000
10000
5000
0
0
1
2
3
4
5
6
7

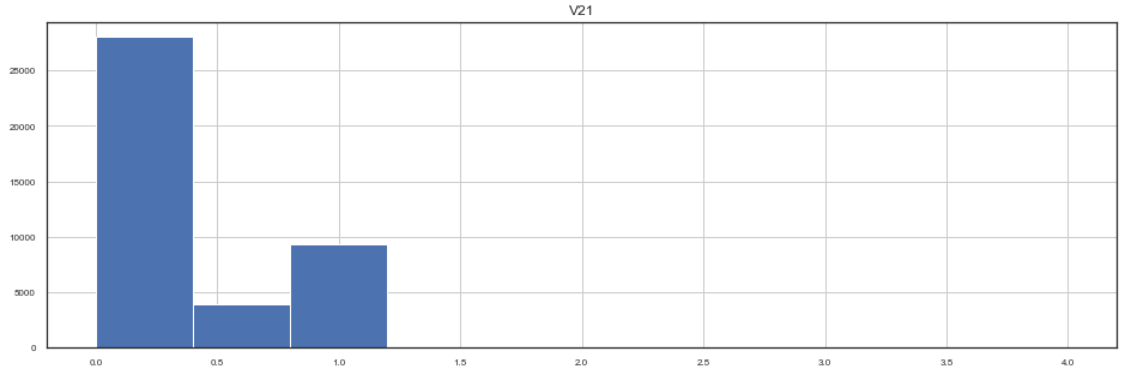

V16
40000
35000
30000
25000
20000
15000
10000
5000
0
0
2
4
6
8
10
12

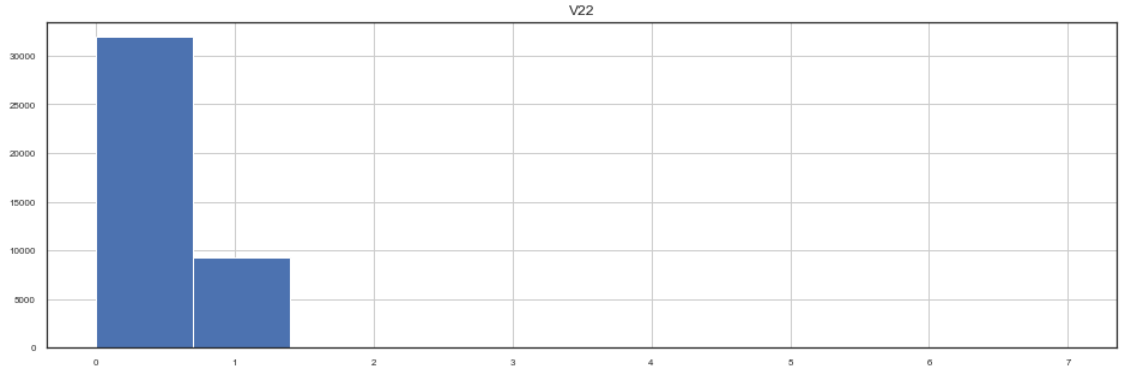

V17
40000
35000
30000
25000
20000
15000
10000
5000
0
0
2
4
6
8
10
12
14

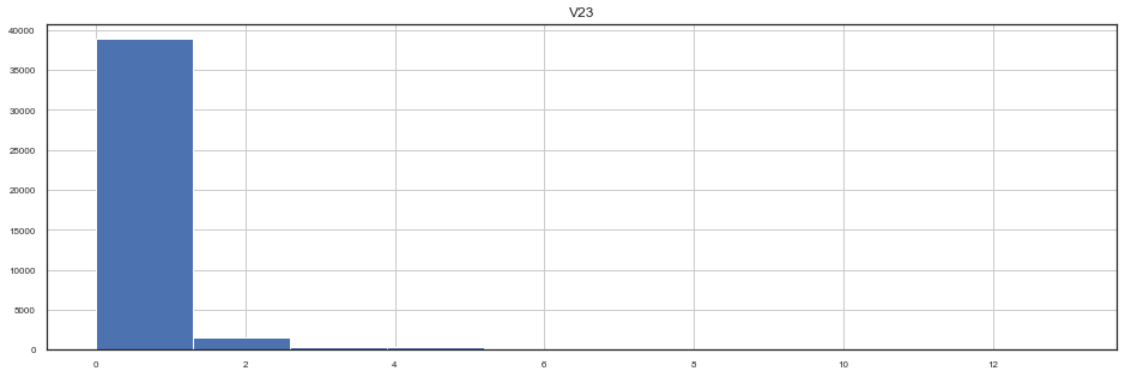

V18
40000
35000
30000
25000
20000
15000
10000
5000
0
0
2
4
6
8
10
12
14

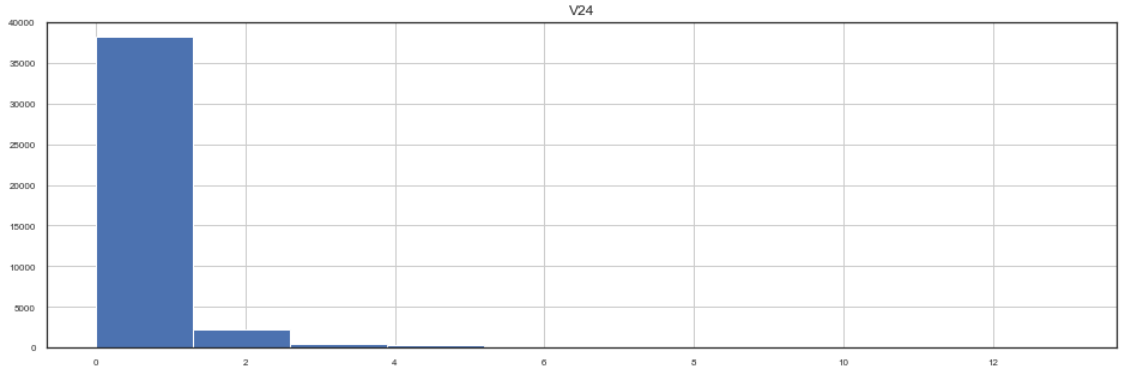

V19
35000
30000
25000
20000
15000
10000
5000
0
0
1
2
3
4
5
6
7

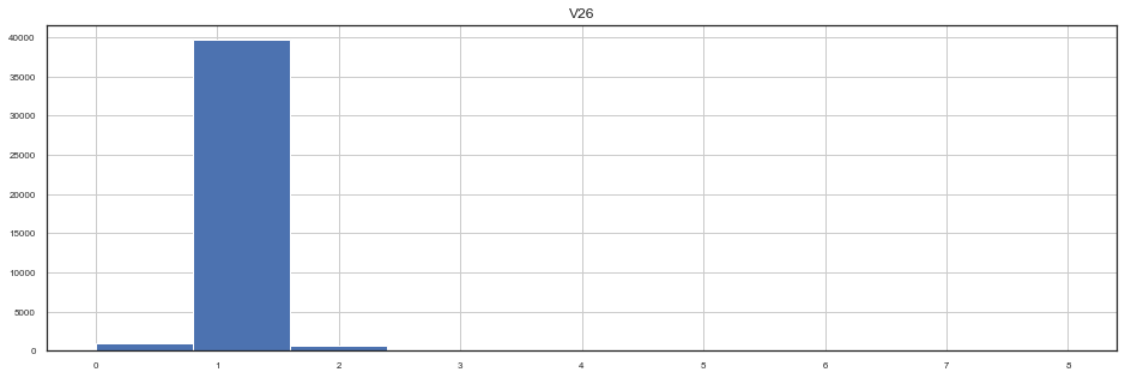

V20
40000
35000
30000
25000
20000
15000
10000
5000
0
0
2
4
6
8
10
12

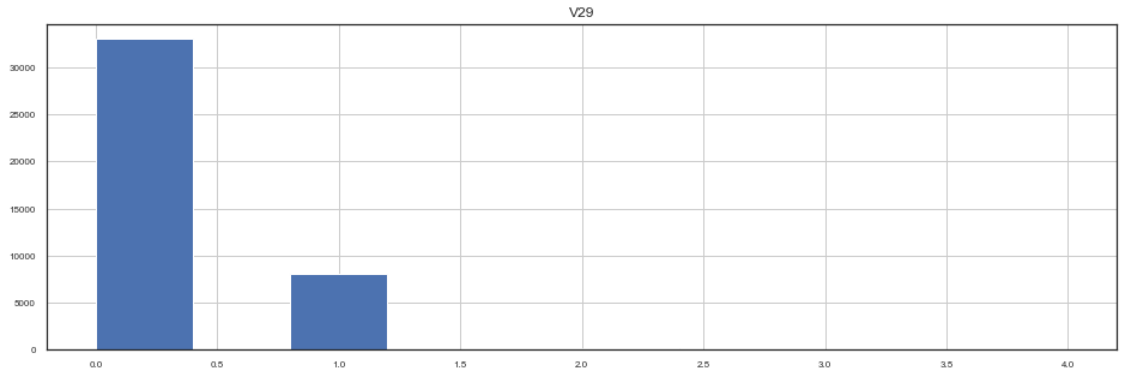

V21
25000
20000
15000
10000
5000
0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0

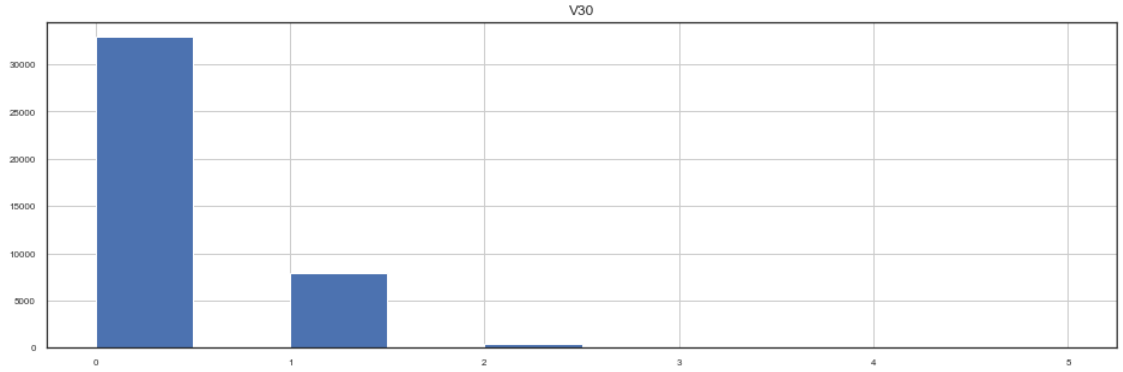

V22
30000
25000
20000
15000
10000
5000
0
0
1
2
3
4
5
6
7

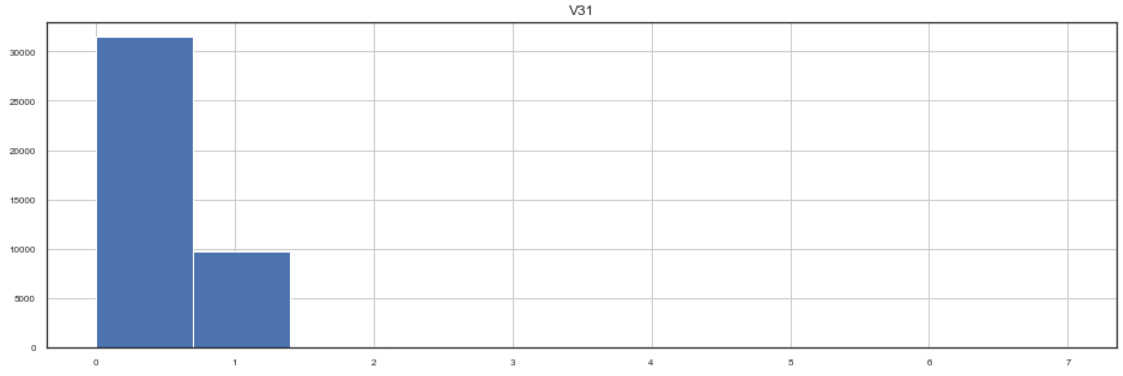

V23
40000
35000
30000
25000
20000
15000
10000
5000
0
0
2
4
6
8
10
12

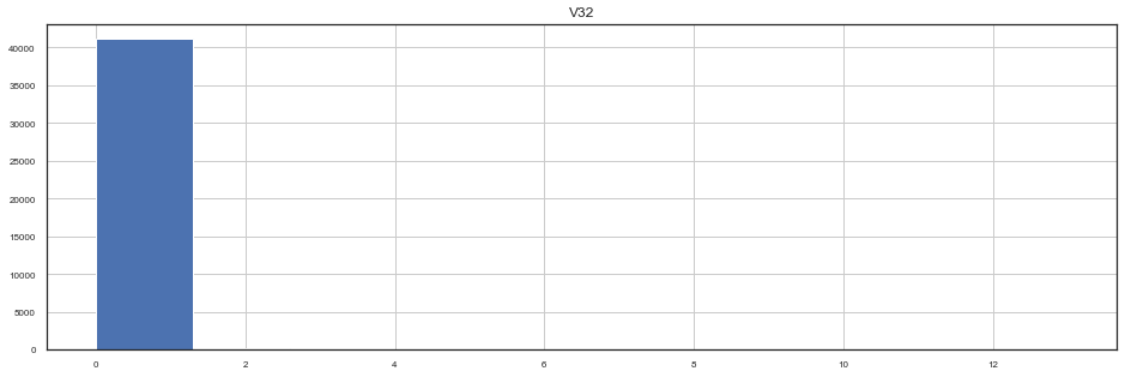

V24
40000
35000
30000
25000
20000
15000
10000
5000
0
0
2
4
6
8
10
12

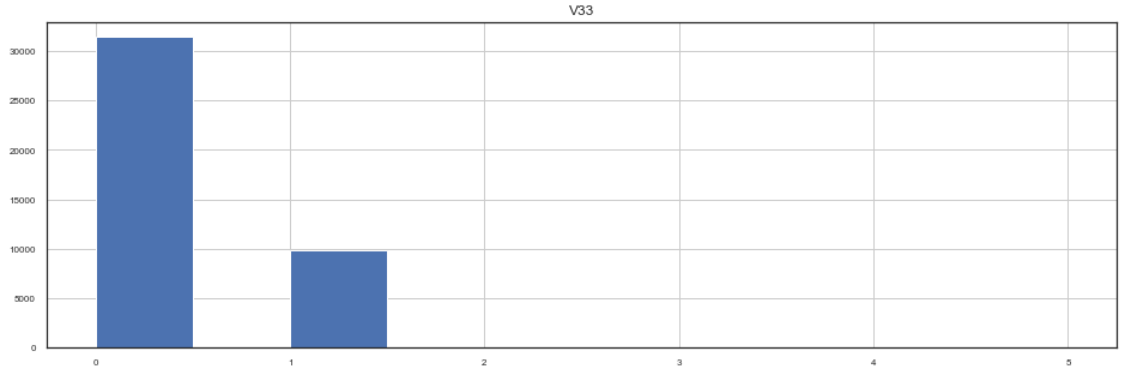

V26
40000
35000
30000
25000
20000
15000
10000
5000
0
0
1
2
3
4
5
6
7
8

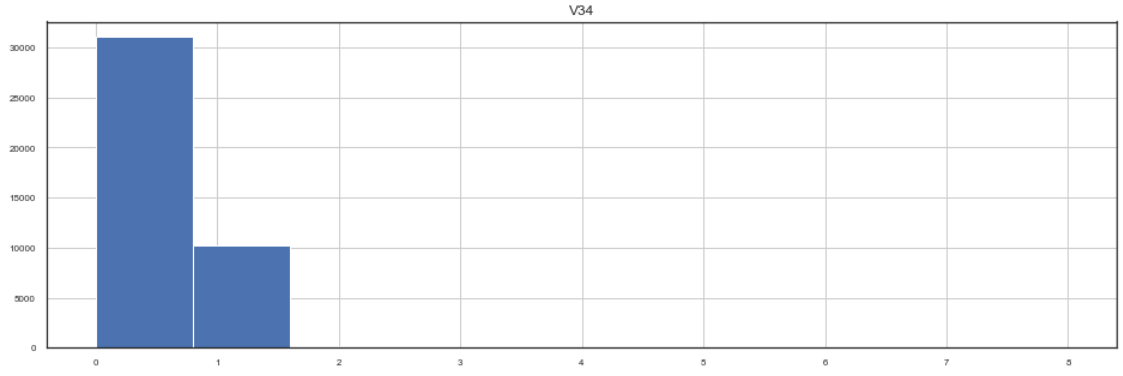

V29
30000
25000
20000
15000
10000
5000
0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0

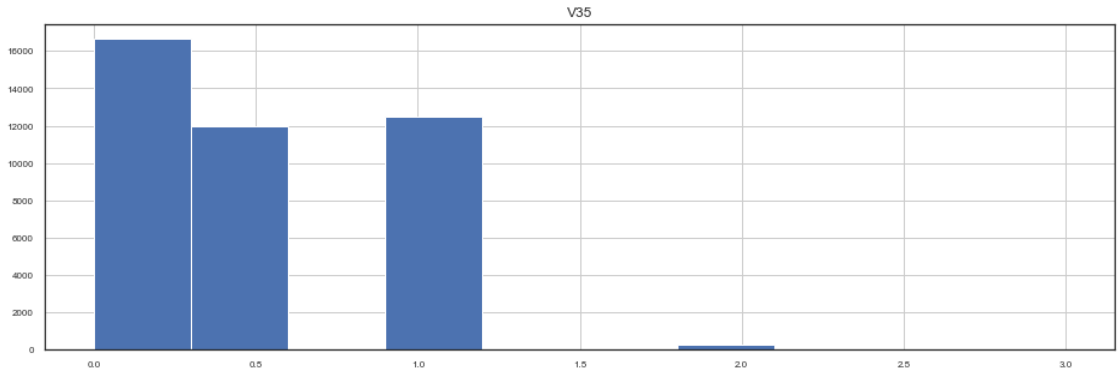

V30
30000
25000
20000
15000
10000
5000
0
0
1
2
3
4
5

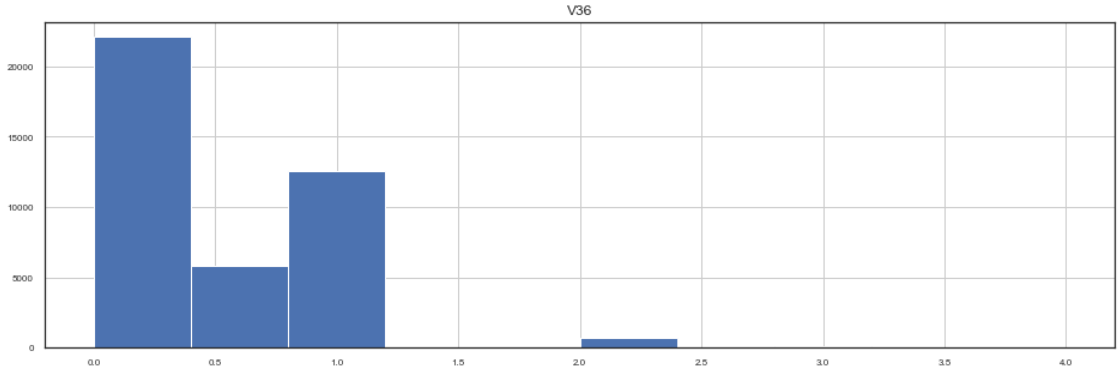

V31
30000
25000
20000
15000
10000
5000
0
0
1
2
3
4
5
6
7

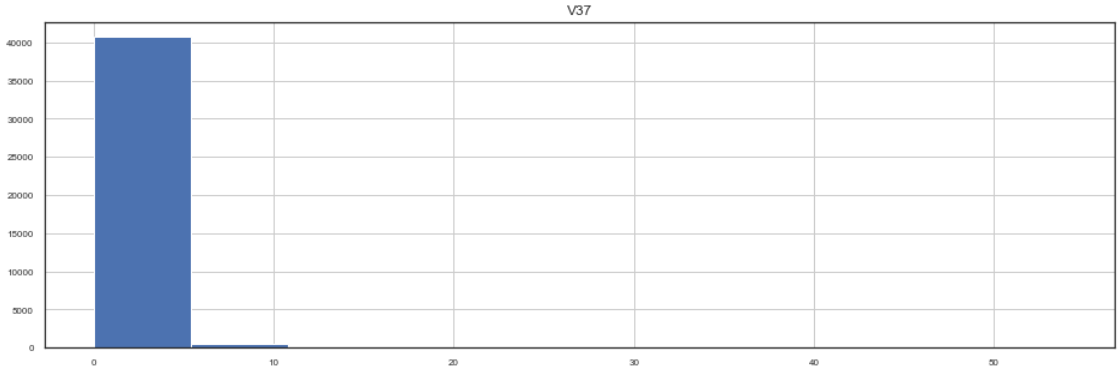

V32
40000
35000
30000
25000
20000
15000
10000
5000
0
0
2
4
6
8
10
12

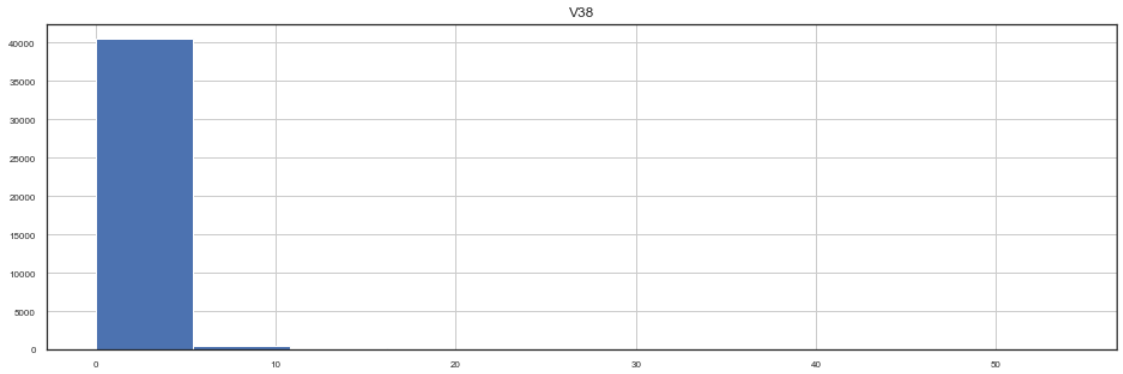

V33
30000
25000
20000
15000
10000
5000
0
0
1
2
3
4
5

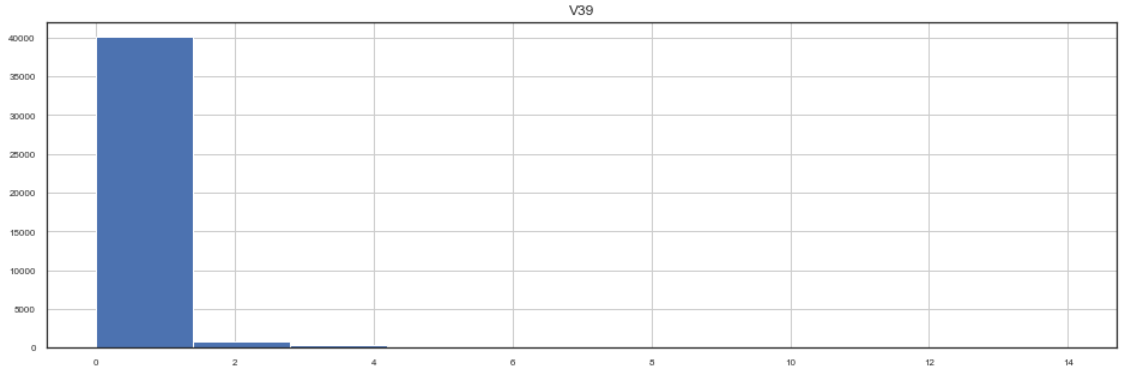

V34
30000
25000
20000
15000
10000
5000
0
0
1
2
3
4
5
6
7
8

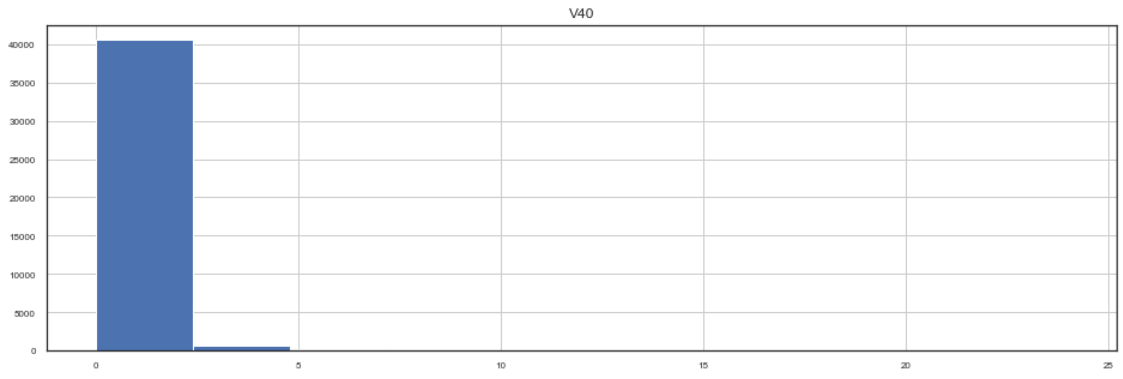

V35
16000
14000
12000
10000
8000
6000
4000
2000
0
0.0
0.5
1.0
1.5
2.0
2.5
3.0

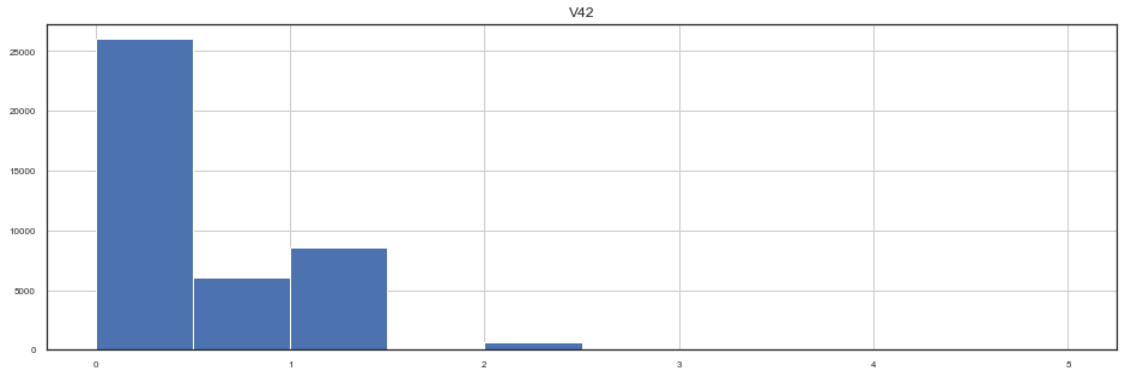

V36
20000
15000
10000
5000
0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0

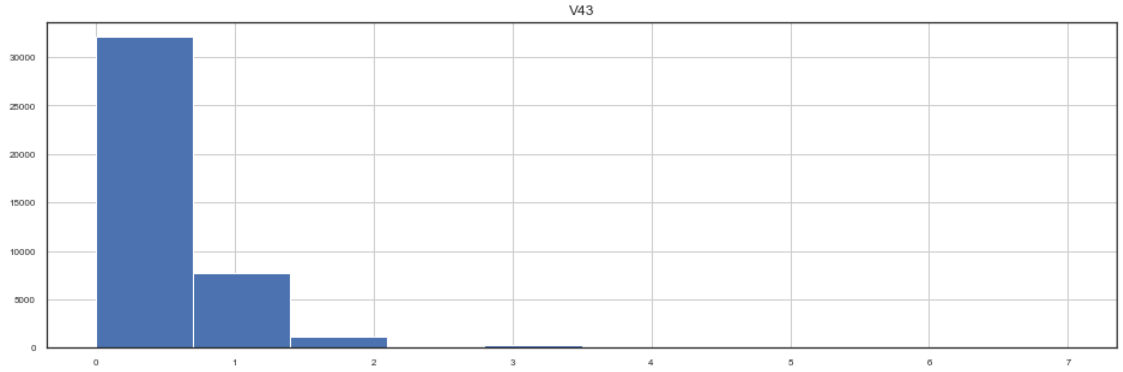

V37
40000
35000
30000
25000
20000
15000
10000
5000
0
0
10
20
30
40
50

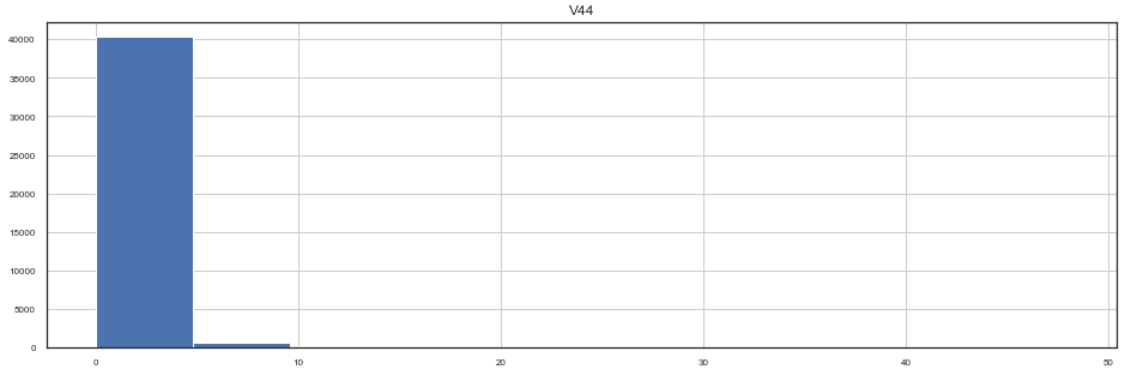

V38
40000
35000
30000
25000
20000
15000
10000
5000
0
0
10
20
30
40
50

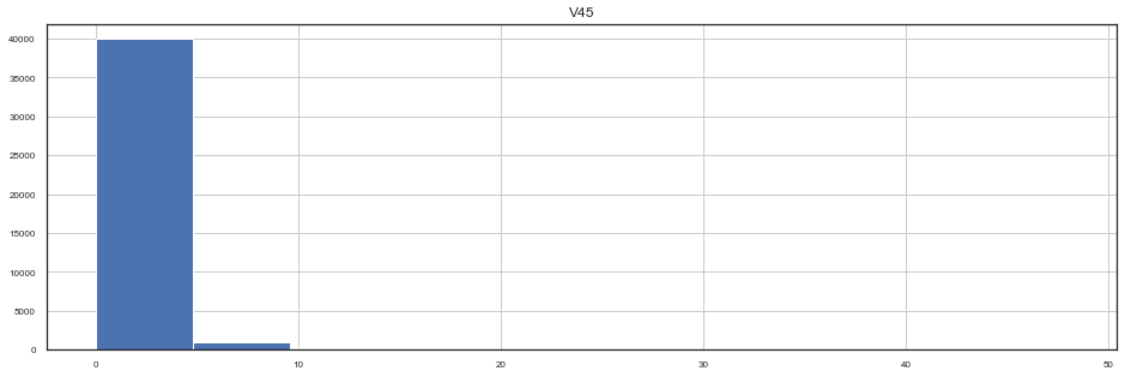

V39
40000
35000
30000
25000
20000
15000
10000
5000
0
0
2
4
6
8
10
12
14

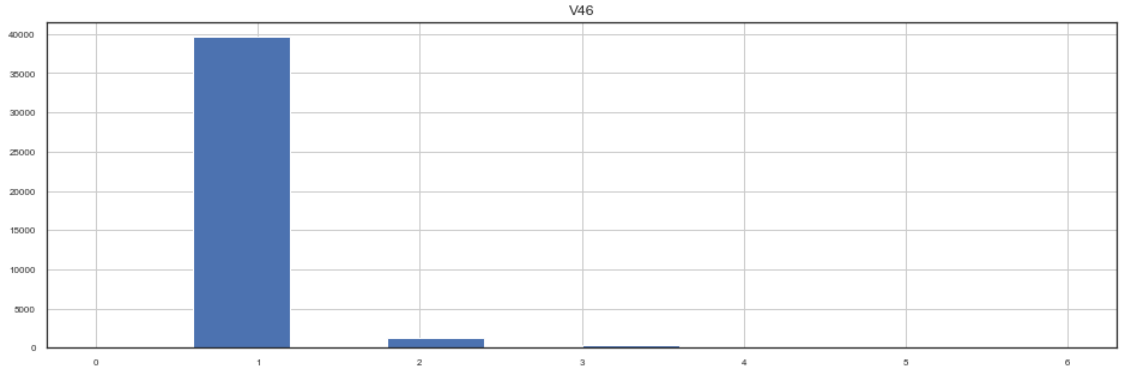

V40
40000
35000
30000
25000
20000
15000
10000
5000
0
0
5
10
15
20
25

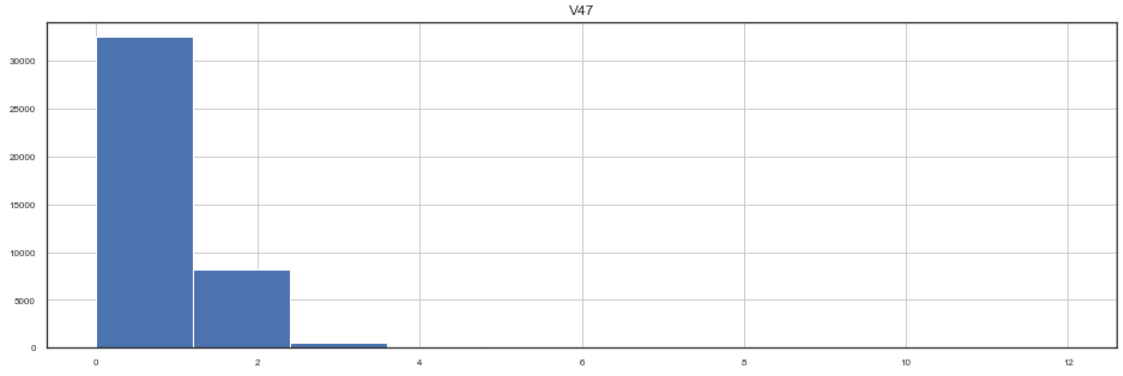

V42
25000
20000
15000
10000
5000
0
0
1
2
3
4
5

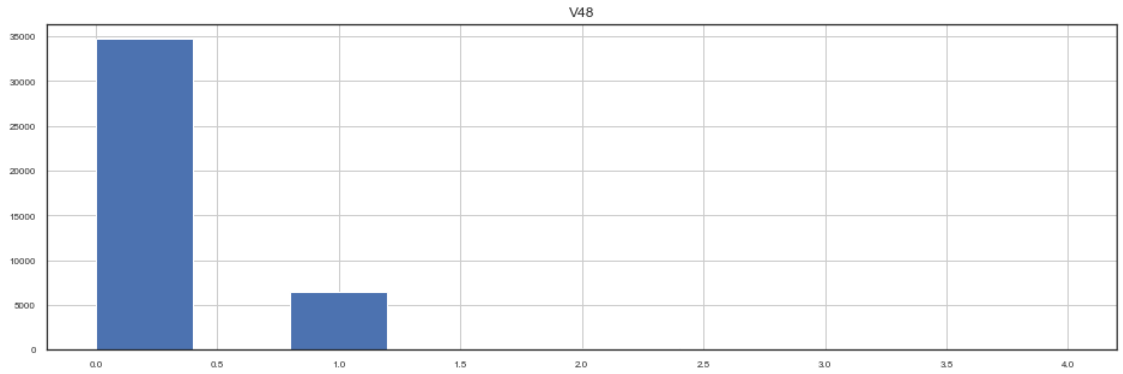

V43
30000
25000
20000
15000
10000
5000
0
0
1
2
3
4
5
6
7

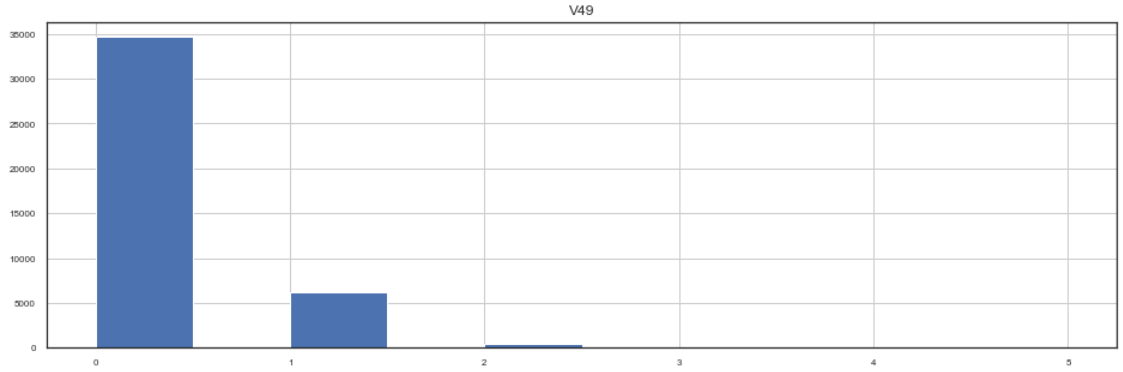

V44
40000
35000
30000
25000
20000
15000
10000
5000
0
0
10
20
30
40
50

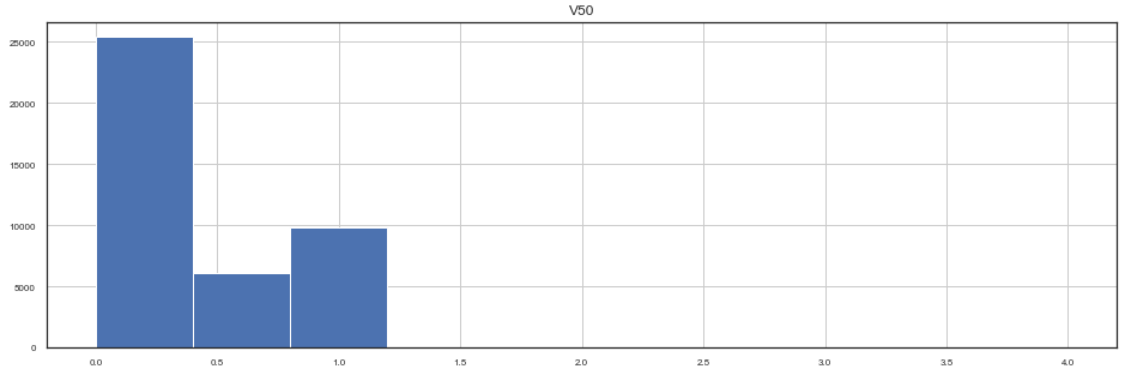

V45
40000
35000
30000
25000
20000
15000
10000
5000
0
0
10
20
30
40
50

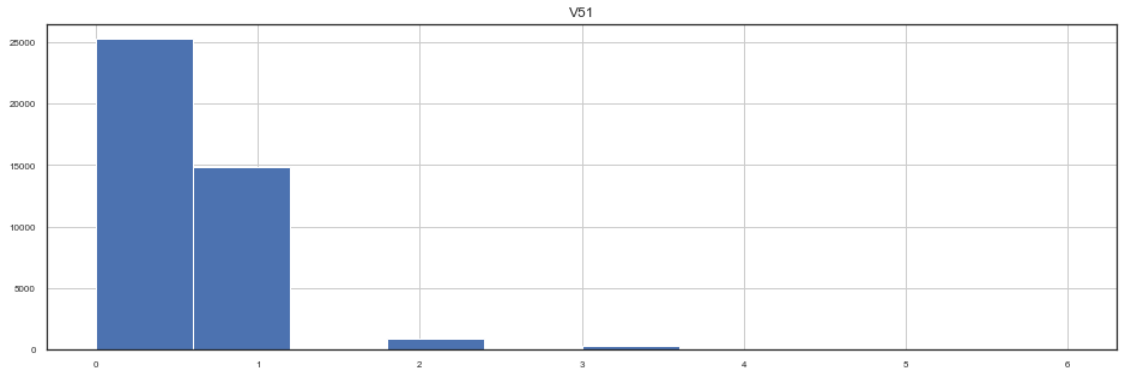

V46
40000
35000
30000
25000
20000
15000
10000
5000
0
0
1
2
3
4
5
6

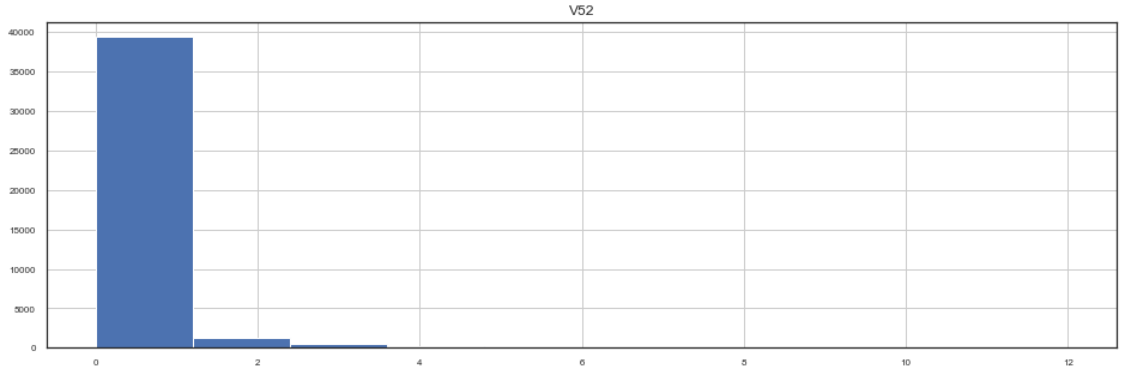

V47
30000
25000
20000
15000
10000
5000
0
0
2
4
6
8
10
12

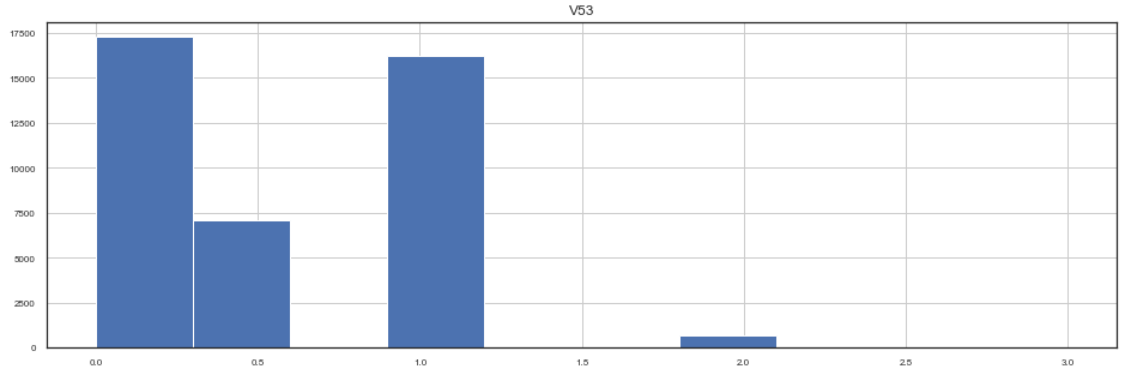

V48
35000
30000
25000
20000
15000
10000
5000
0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0

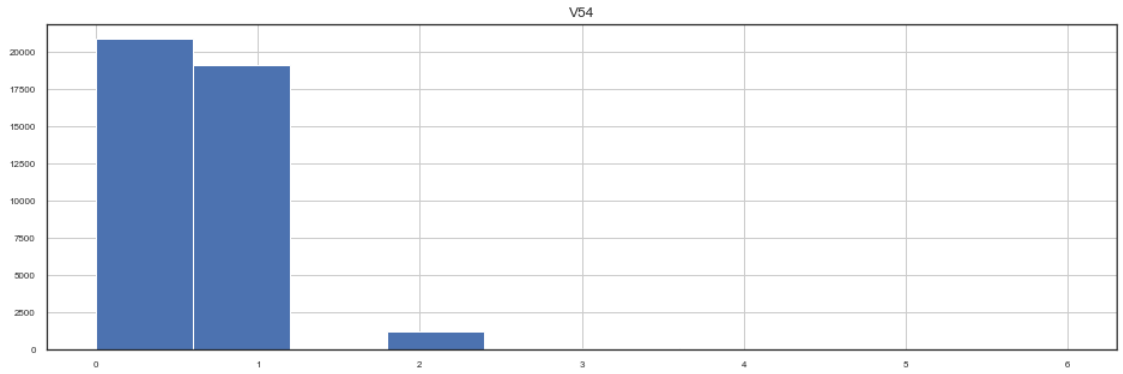

V49
35000
30000
25000
20000
15000
10000
5000
0
0
1
2
3
4
5

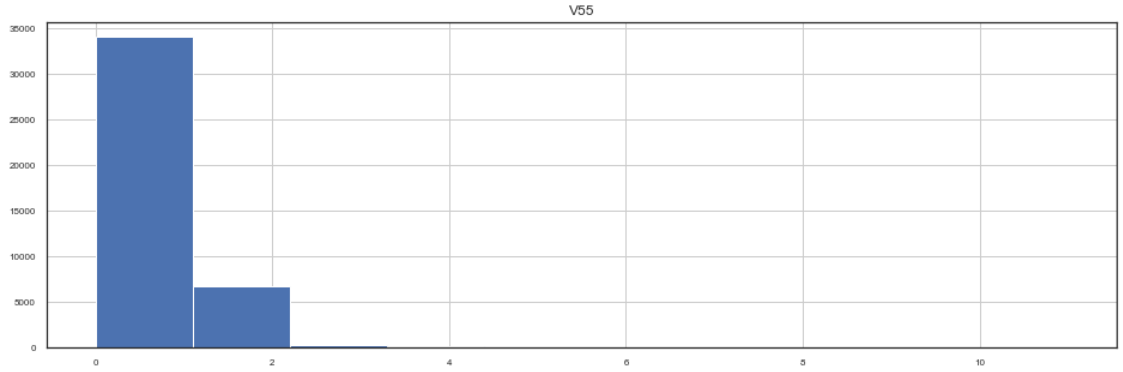

V50
25000
20000
15000
10000
5000
0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0

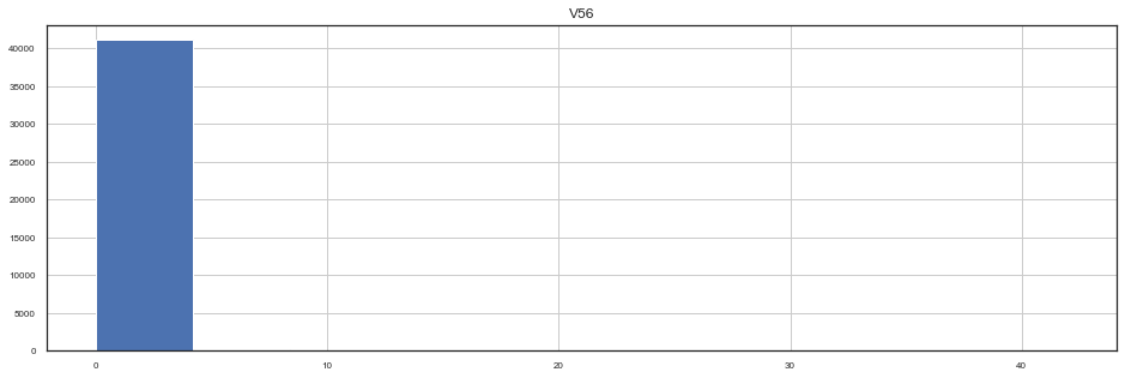

V51
25000
20000
15000
10000
5000
0
0
1
2
3
4
5
6

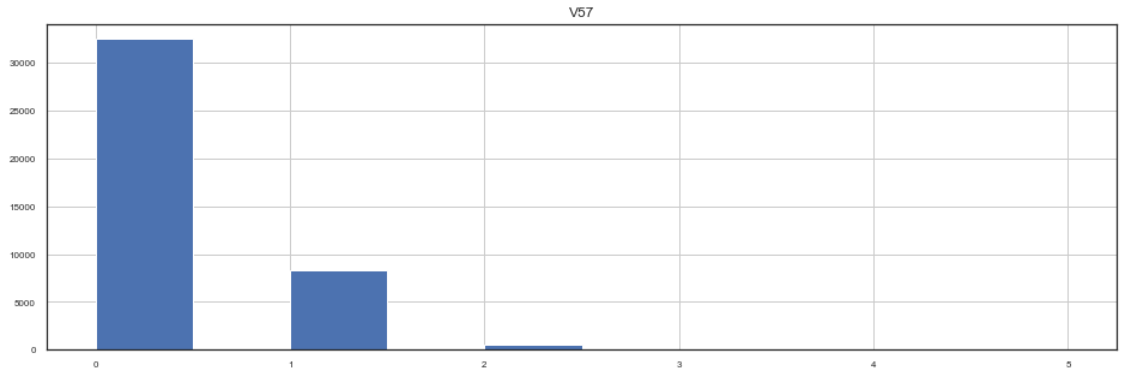

V52
40000
35000
30000
25000
20000
15000
10000
5000
0
0
2
4
6
8
10
12

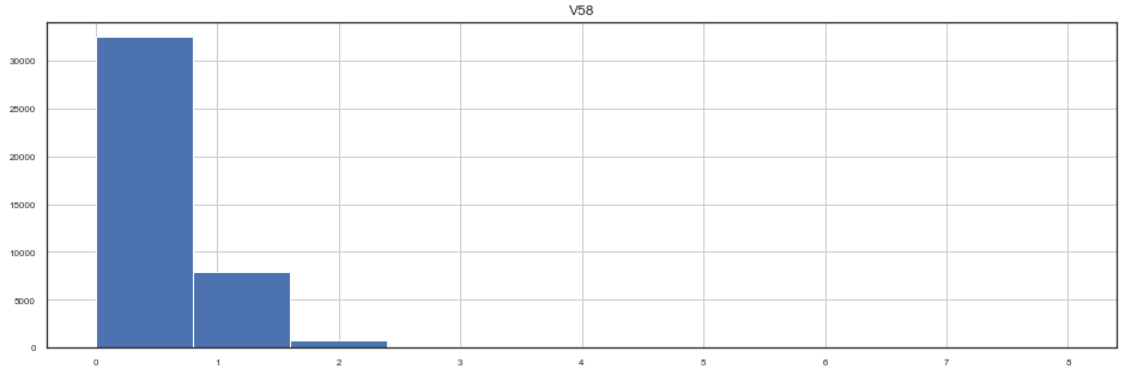

V53
17500
15000
12500
10000
7500
5000
2500
0
0.0
0.5
1.0
1.5
2.0
2.5
3.0

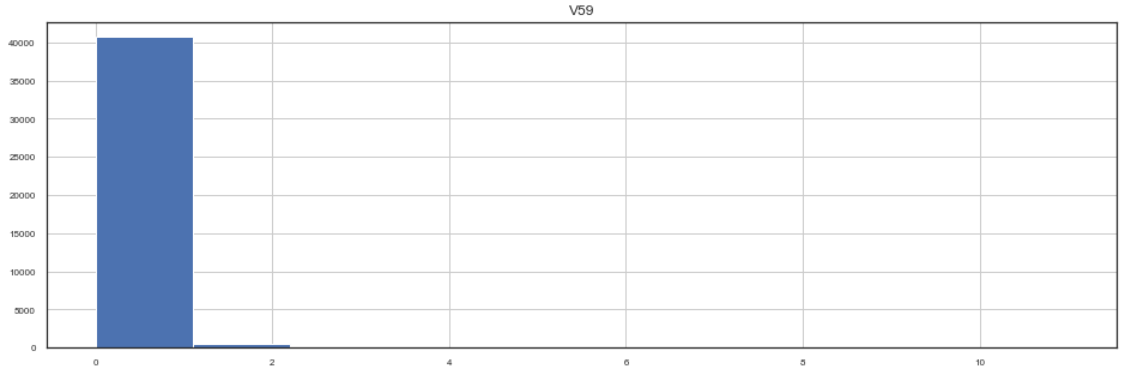

V54
20000
17500
15000
12500
10000
7500
5000
2500
0
0
1
2
3
4
5
6

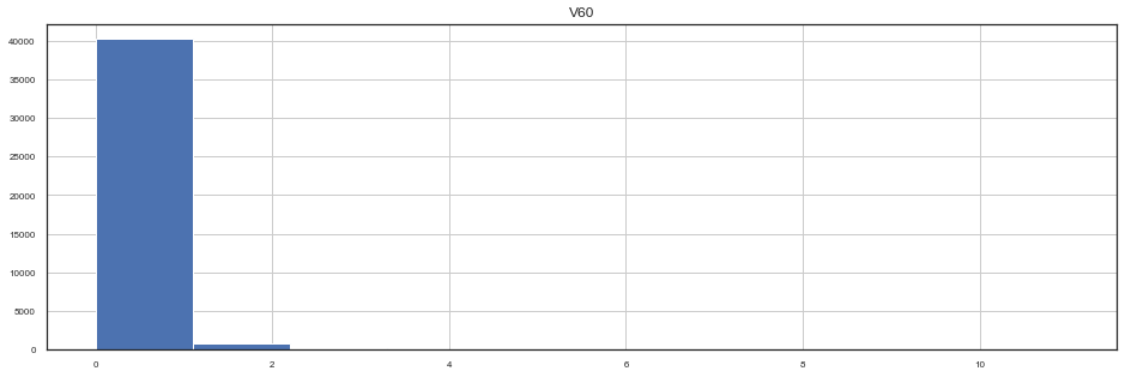

V55
35000
30000
25000
20000
15000
10000
5000
0
0
2
4
6
8
10

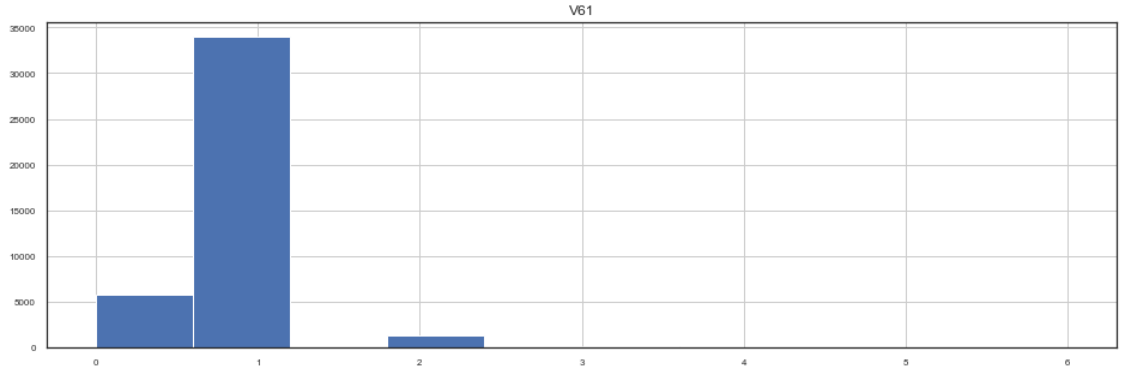

V56
40000
35000
30000
25000
20000
15000
10000
5000
0
0
10
20
30
40

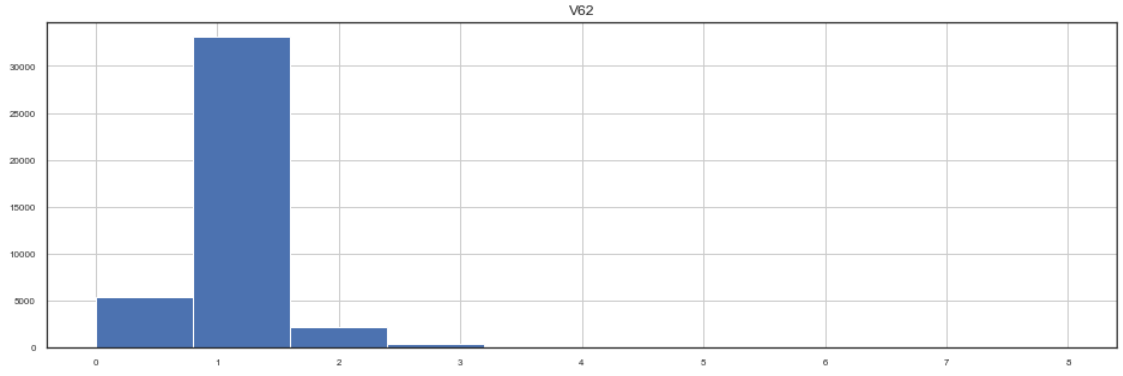

V57
30000
25000
20000
15000
10000
5000
0
0
1
2
3
4
5

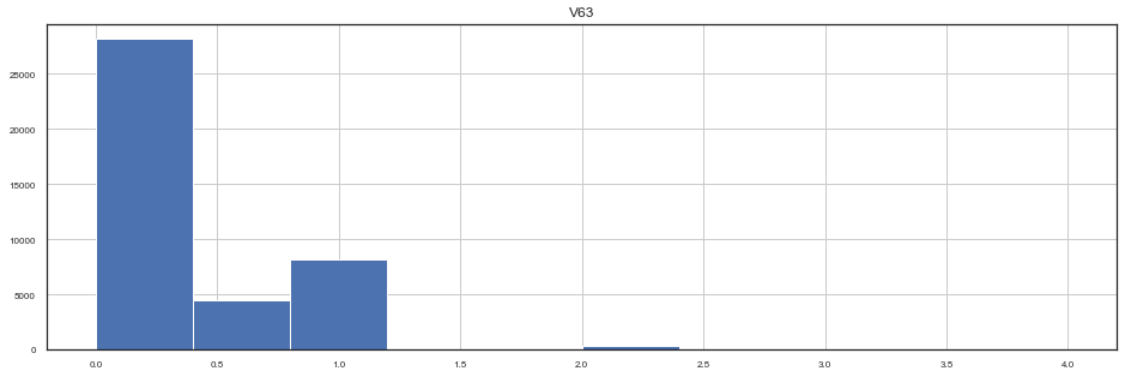

V58
30000
25000
20000
15000
10000
5000
0
0
1
2
3
4
5
6
7
8

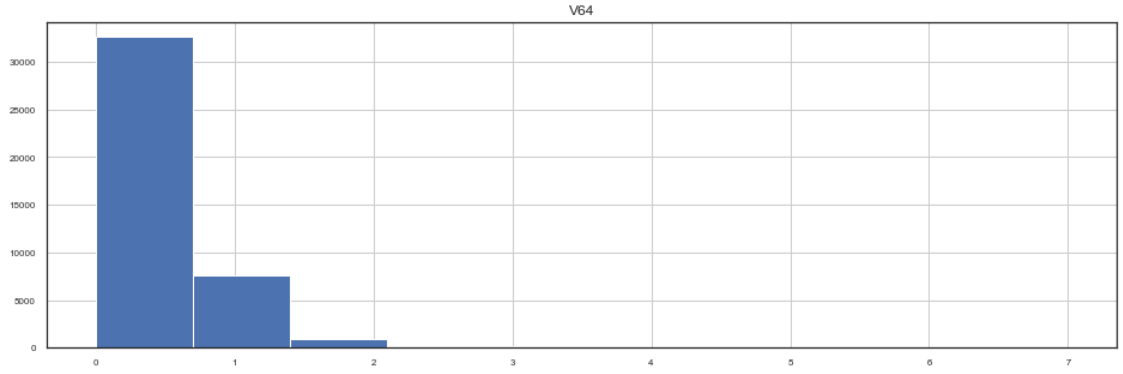

V59
40000
35000
30000
25000
20000
15000
10000
5000
0
0
2
4
6
8
10

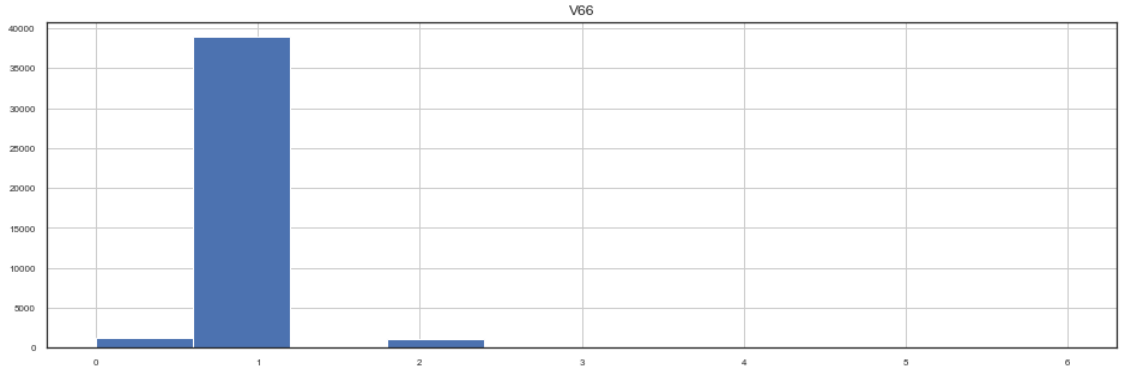

V60
40000
35000
30000
25000
20000
15000
10000
5000
0
0
2
4
6
8
10

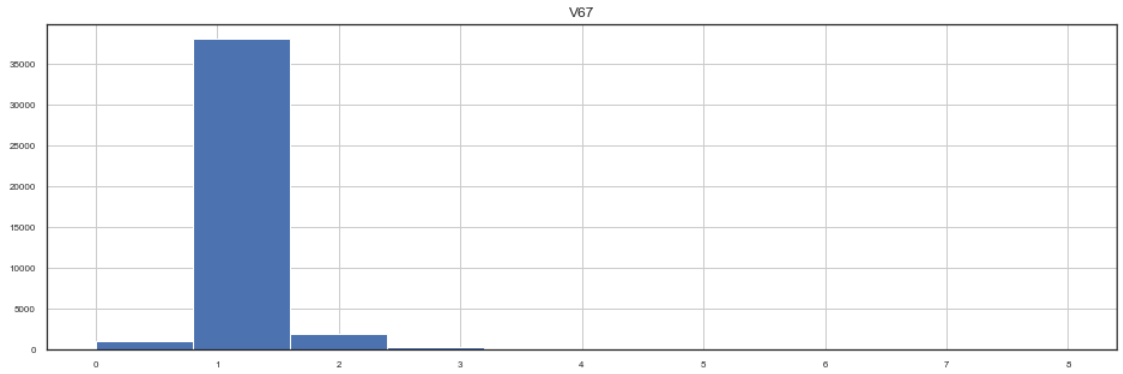

V61
35000
30000
25000
20000
15000
10000
5000
0
0
1
2
3
4
5
6

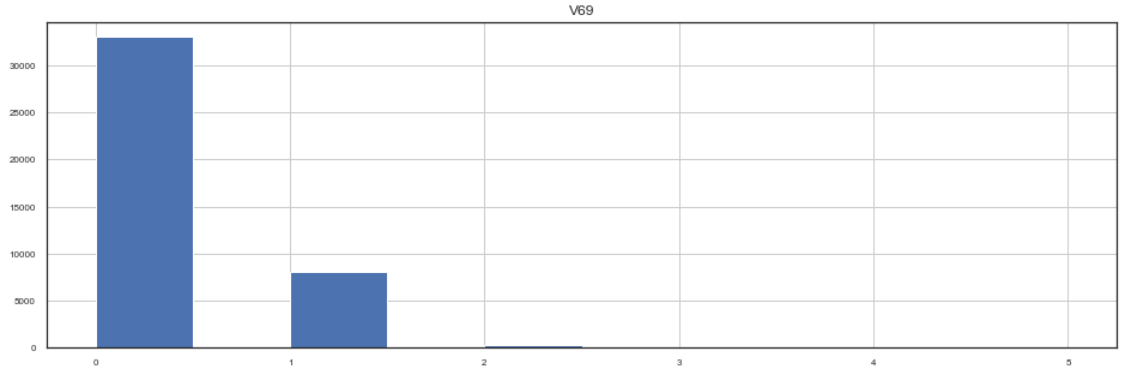

V62
30000
25000
20000
15000
10000
5000
0
0
1
2
3
4
5
6
7
8

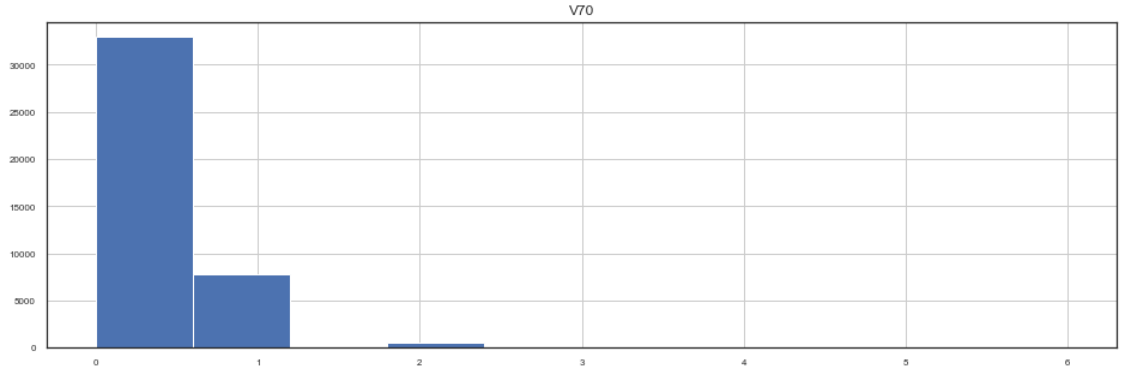

V63
25000
20000
15000
10000
5000
0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0

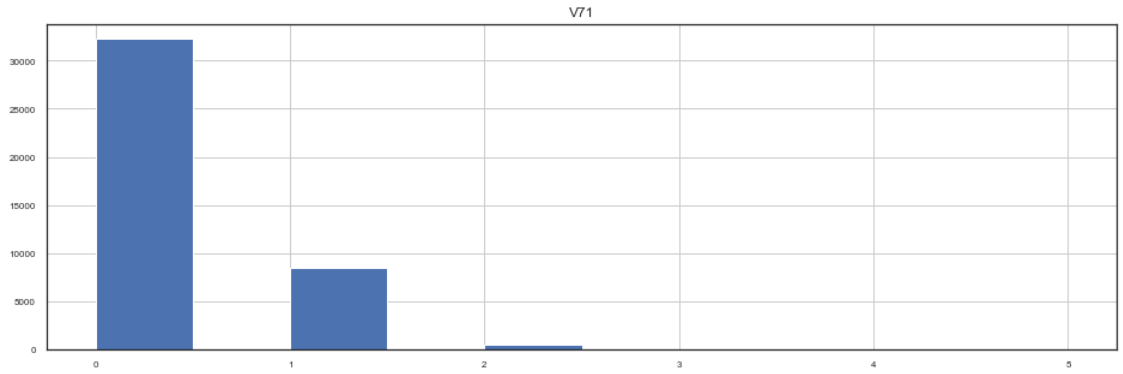

V64
30000
25000
20000
15000
10000
5000
0
0
1
2
3
4
5
6
7

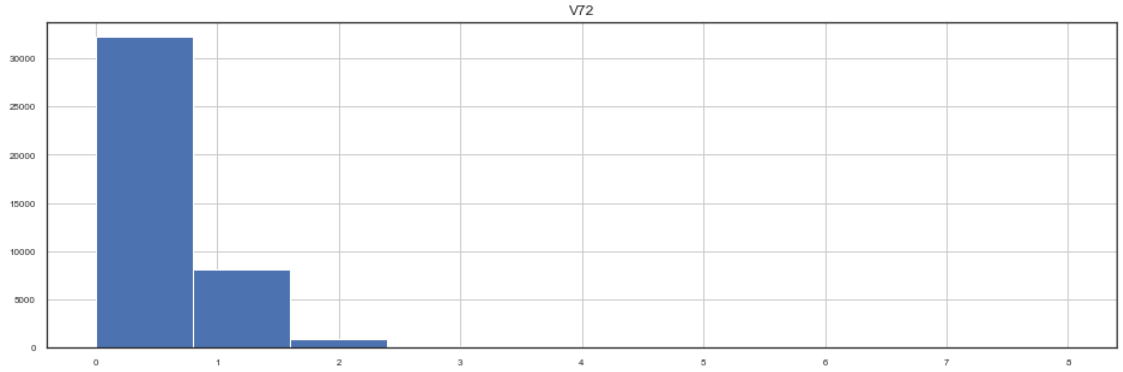

V66
40000
35000
30000
25000
20000
15000
10000
5000
0
0
1
2
3
4
5
6

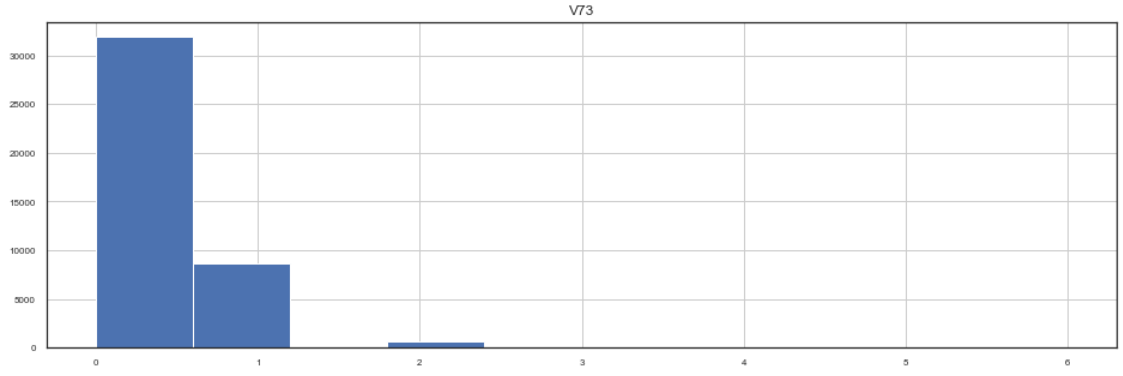

V67
35000
30000
25000
20000
15000
10000
5000
0
0
1
2
3
4
5
6
7
8

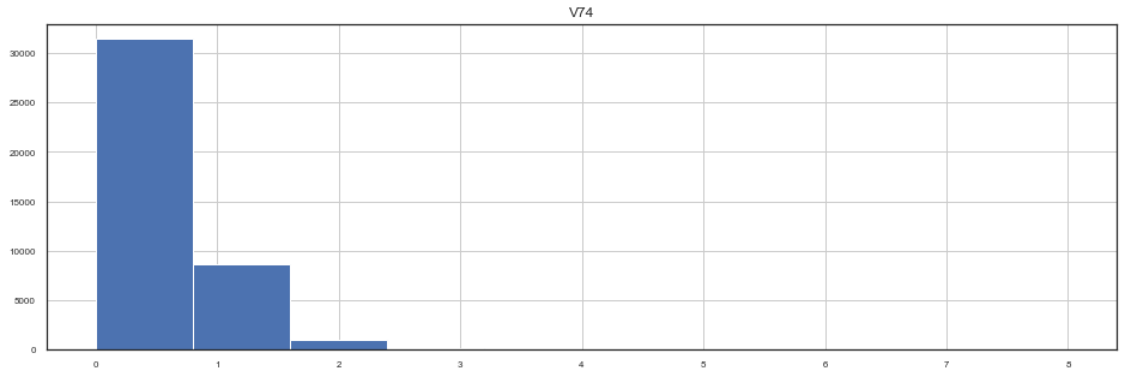

V69
30000
25000
20000
15000
10000
5000
0
0
1
2
3
4
5

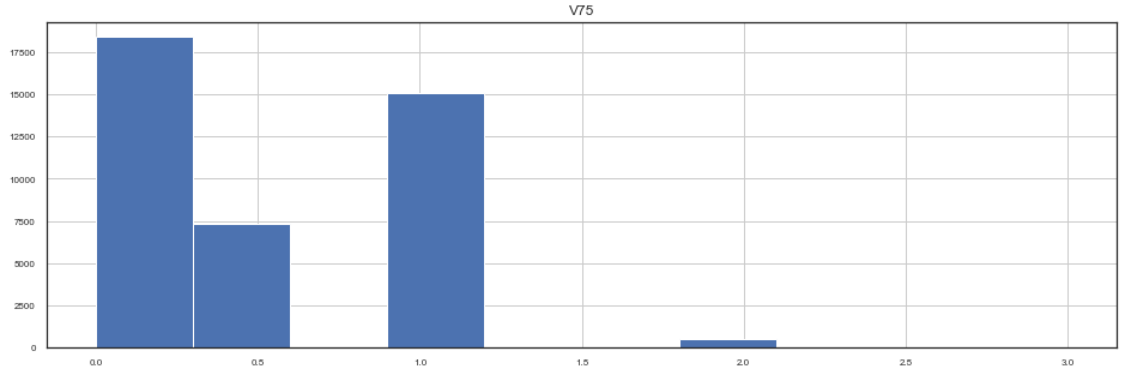

V70
30000
25000
20000
15000
10000
5000
0
0
1
2
3
4
5
6

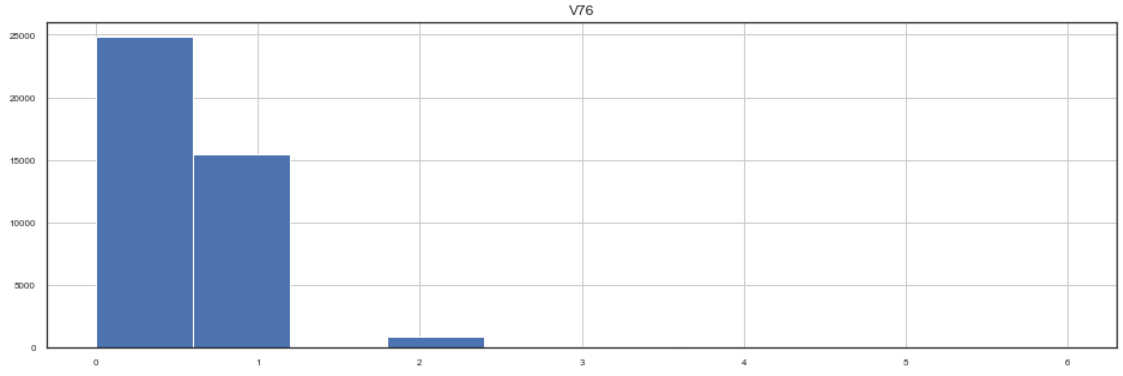

V71
30000
25000
20000
15000
10000
5000
0
0
1
2
3
4
5

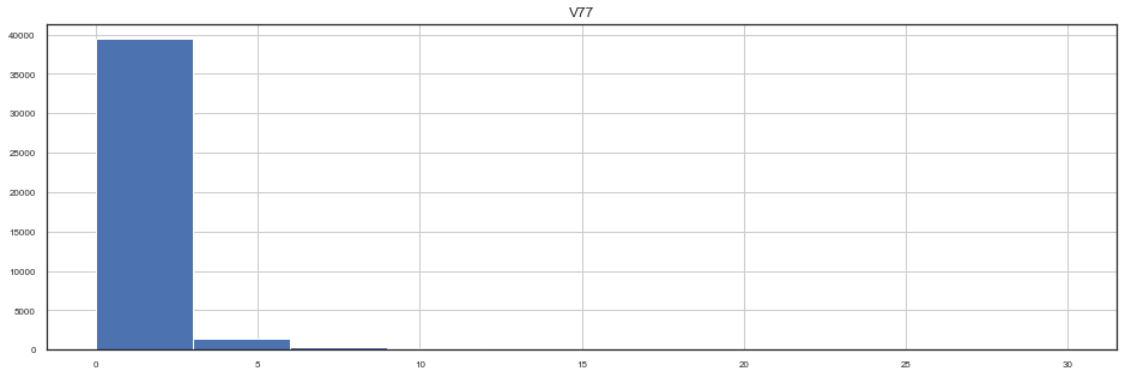

V72
30000
25000
20000
15000
10000
5000
0
0
1
2
3
4
5
6
7
8

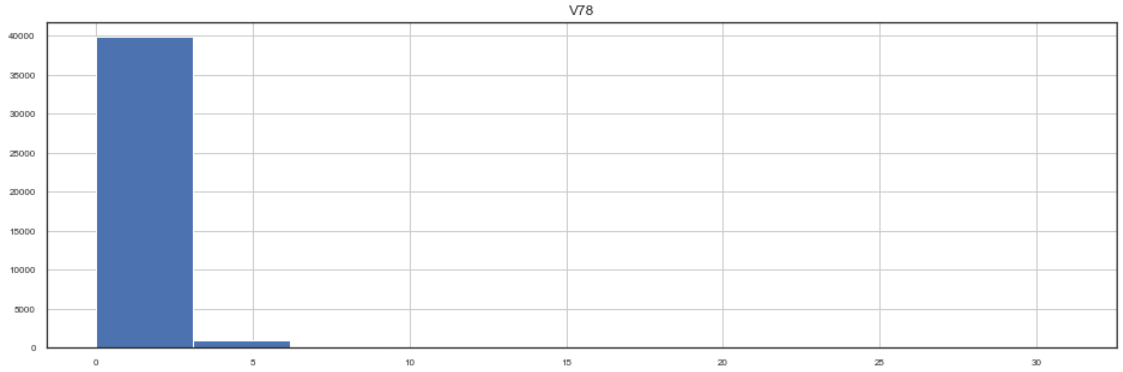

V73
30000
25000
20000
15000
10000
5000
0
0
1
2
3
4
5
6

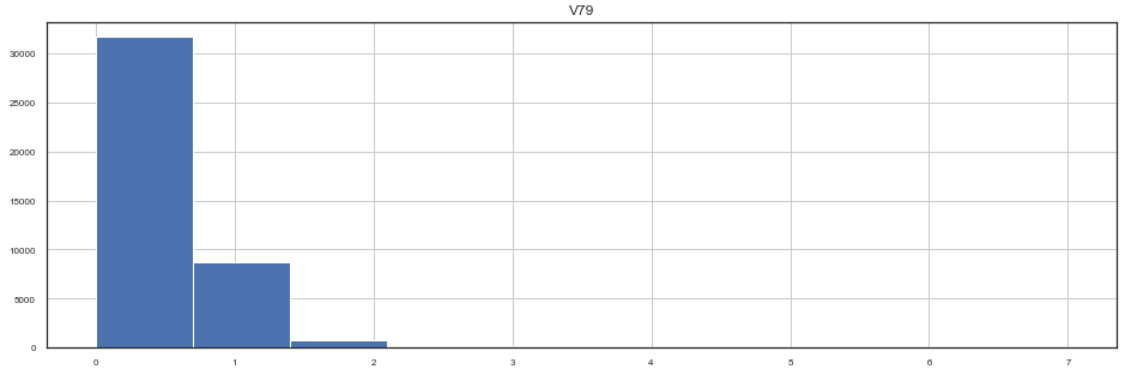

V74
30000
25000
20000
15000
10000
5000
0
0
1
2
3
4
5
6
7
8

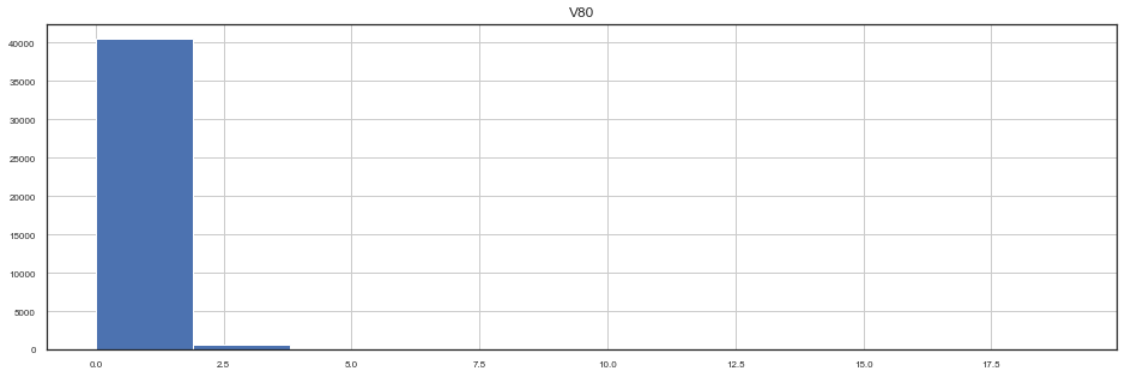

V75
17500
15000
12500
10000
7500
5000
2500
0
0.0
0.5
1.0
1.5
2.0
2.5
3.0

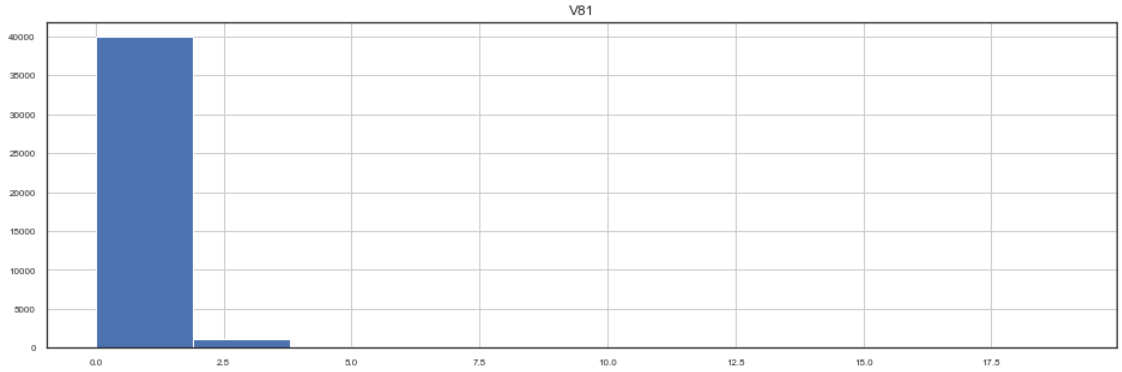

V76
25000
20000
15000
10000
5000
0
0
1
2
3
4
5
6

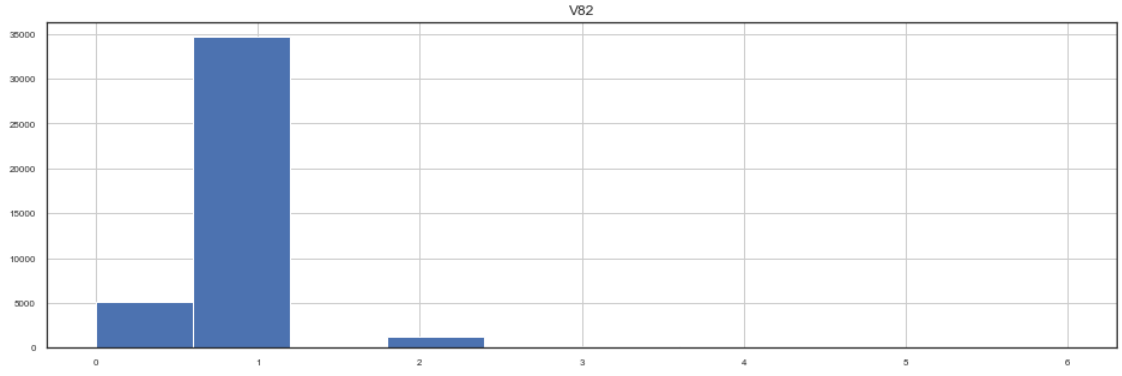

V77
40000
35000
30000
25000
20000
15000
10000
5000
0
0
5
10
15
20
25
30

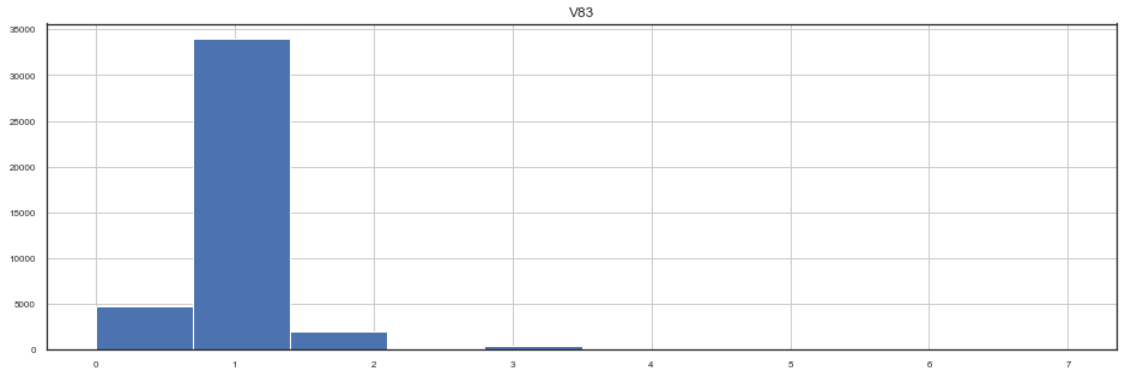

V78
40000
35000
30000
25000
20000
15000
10000
5000
0
0
5
10
15
20
25
30

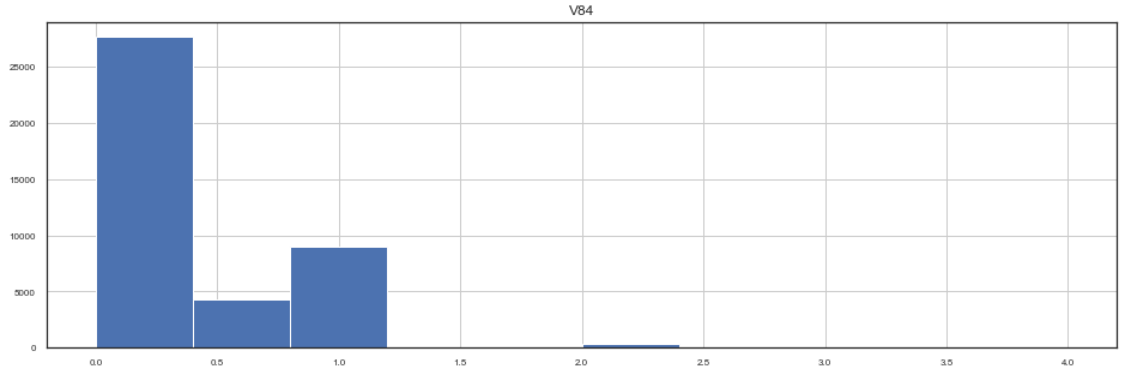

V79
30000
25000
20000
15000
10000
5000
0
0
1
2
3
4
5
6
7

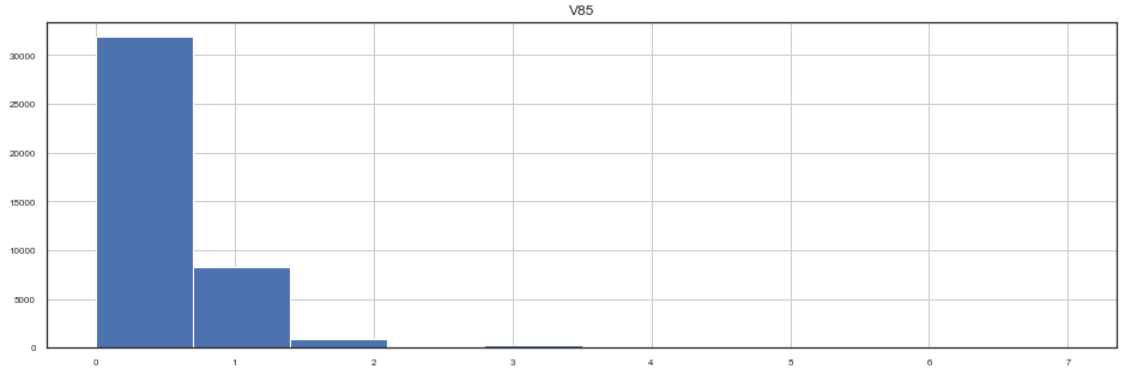

V80
40000
35000
30000
25000
20000
15000
10000
5000
0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

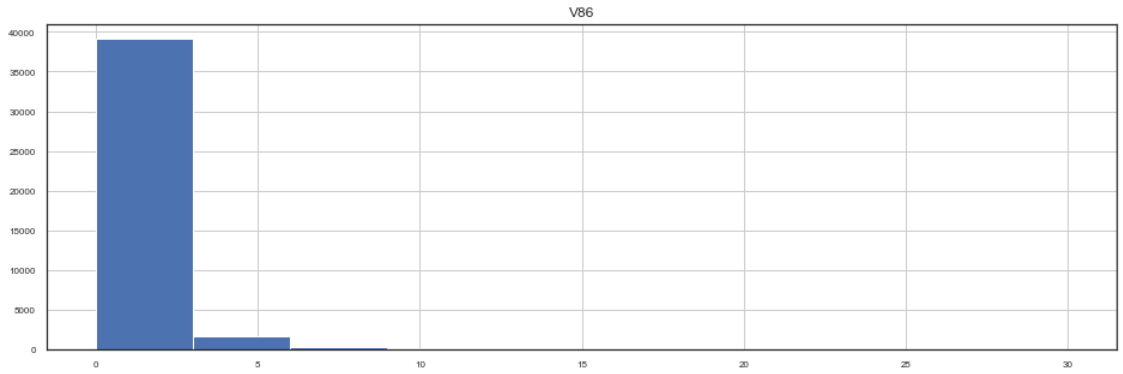

V81
40000
35000
30000
25000
20000
15000
10000
5000
0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

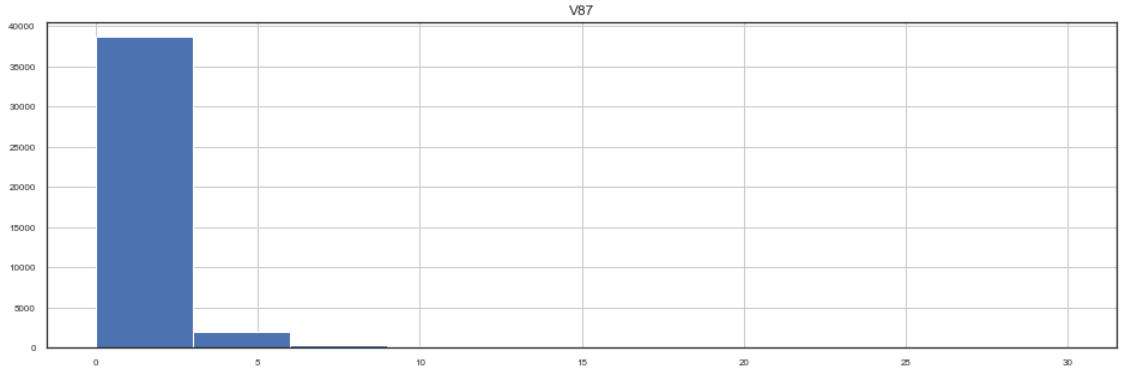

V82
35000
30000
25000
20000
15000
10000
5000
0
0
1
2
3
4
5
6

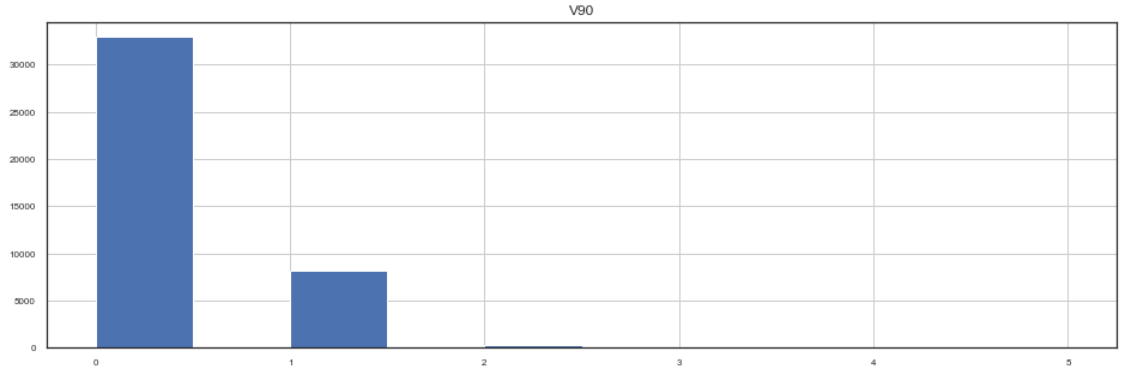

V83
35000
30000
25000
20000
15000
10000
5000
0
0
1
2
3
4
5
6
7

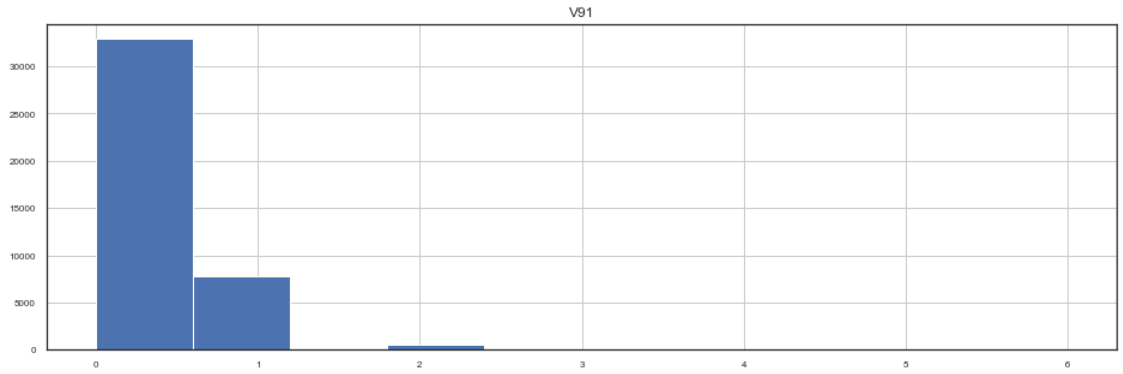

V84
25000
20000
15000
10000
5000
0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0

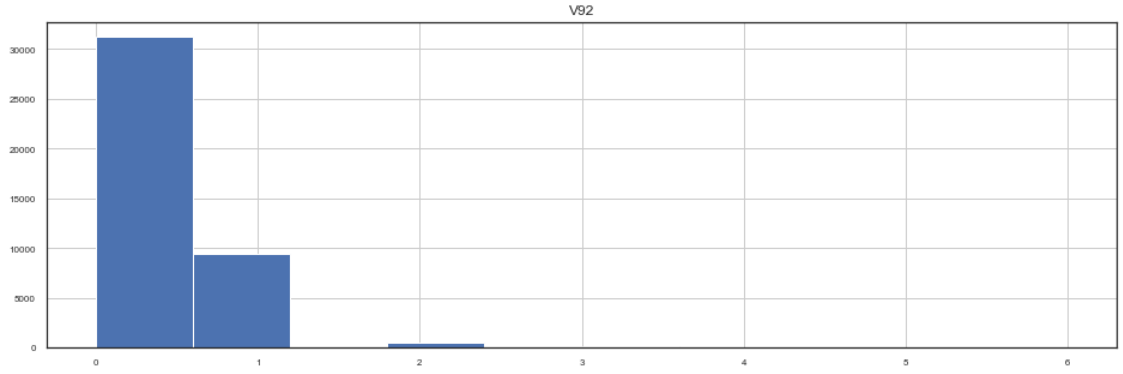

V85
30000
25000
20000
15000
10000
5000
0
0
1
2
3
4
5
6
7

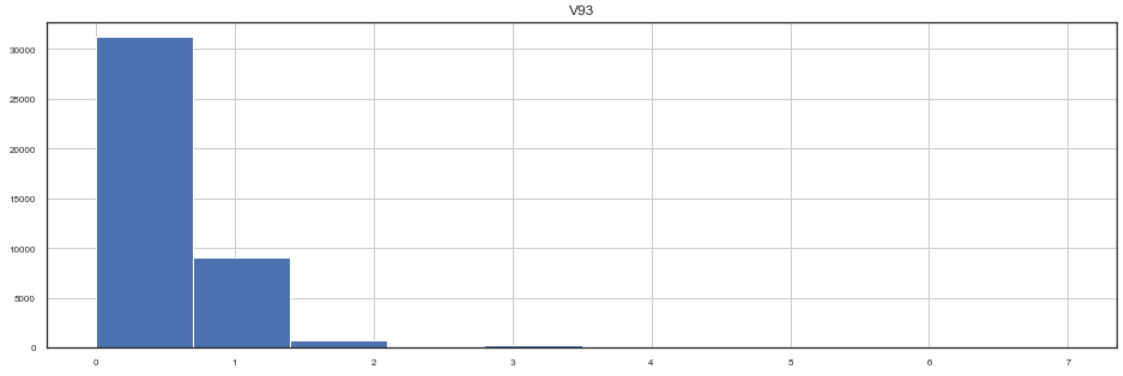

V86
40000
35000
30000
25000
20000
15000
10000
5000
0
0
5
10
15
20
25
30

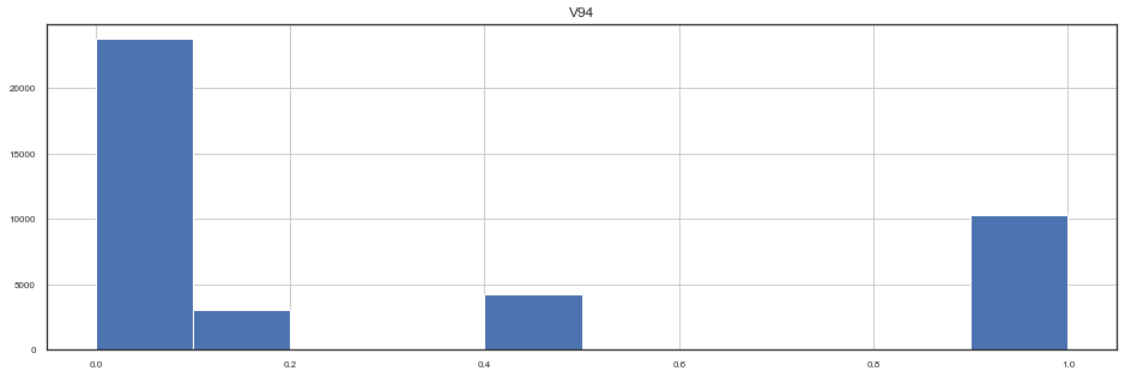

V87
40000
35000
30000
25000
20000
15000
10000
5000
0
0
5
10
15
20
25
30

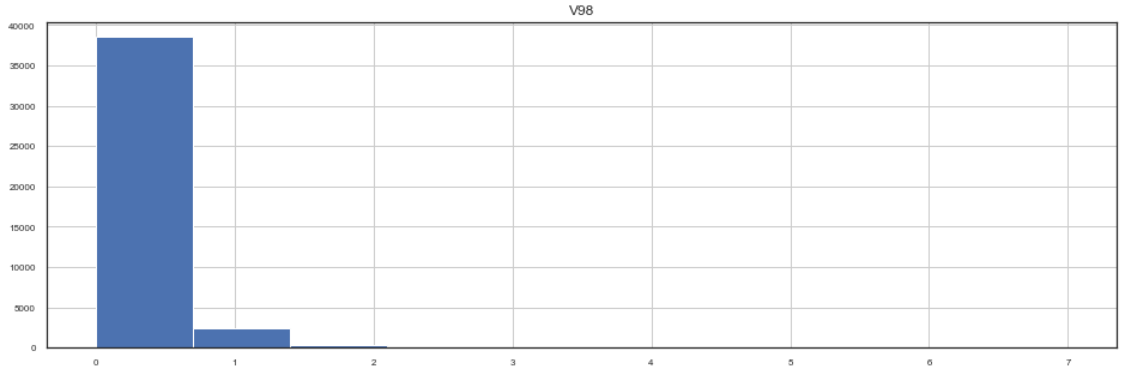

V90
30000
25000
20000
15000
10000
5000
0
0
1
2
3
4
5

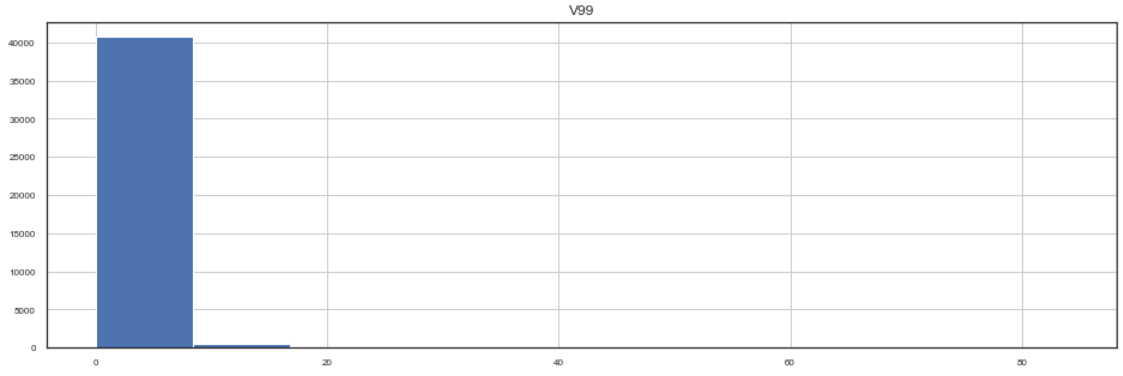

V91
30000
25000
20000
15000
10000
5000
0
0
1
2
3
4
5
6

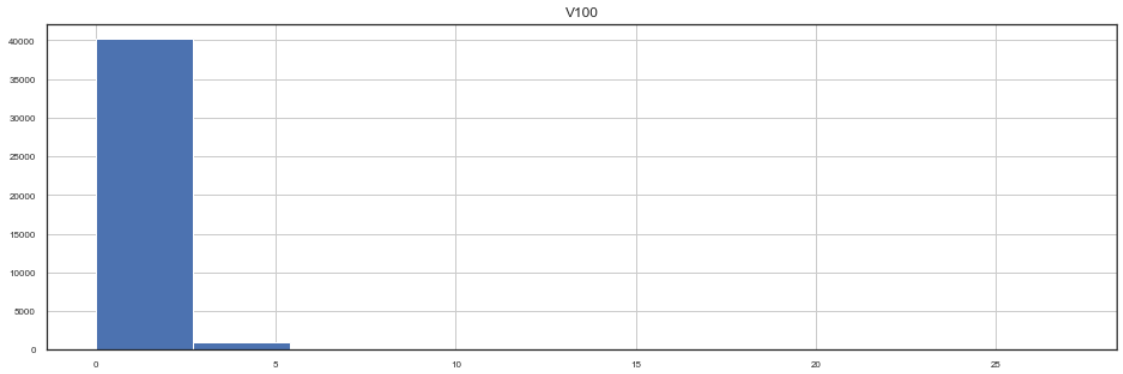

V92
30000
25000
20000
15000
10000
5000
0
0
1
2
3
4
5
6

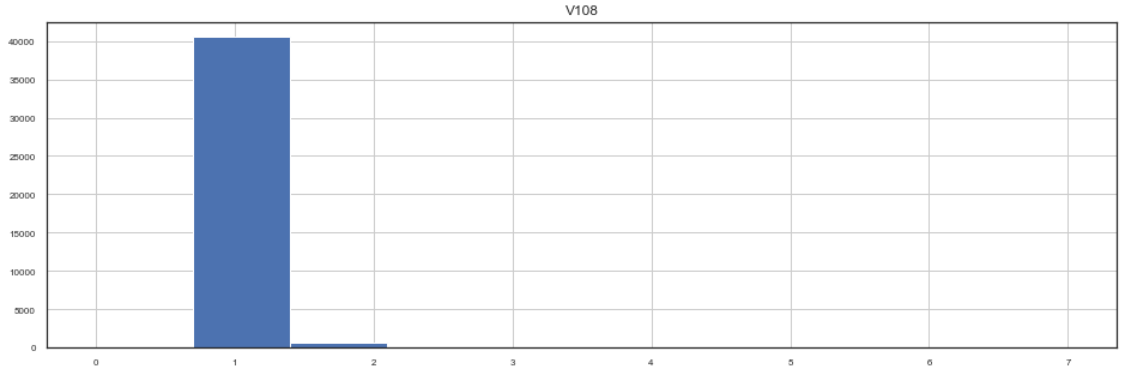

V93
30000
25000
20000
15000
10000
5000
0
0
1
2
3
4
5
6
7

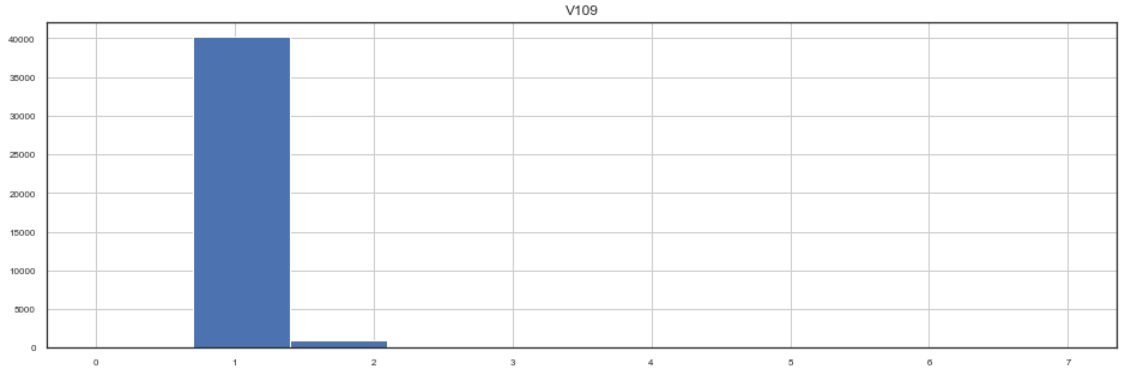

V94
20000
15000
10000
5000
0
0.0
0.2
0.4
0.6
0.8
1.0

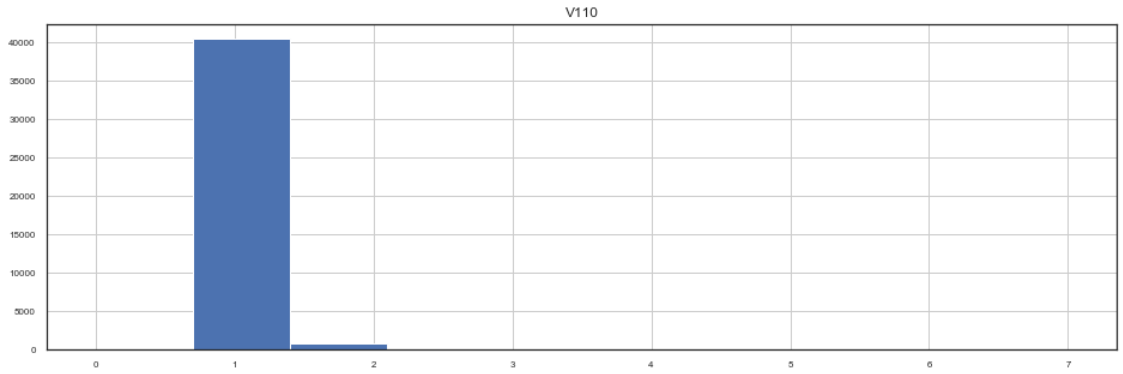

V98
40000
35000
30000
25000
20000
15000
10000
5000
0
0
1
2
3
4
5
6
7

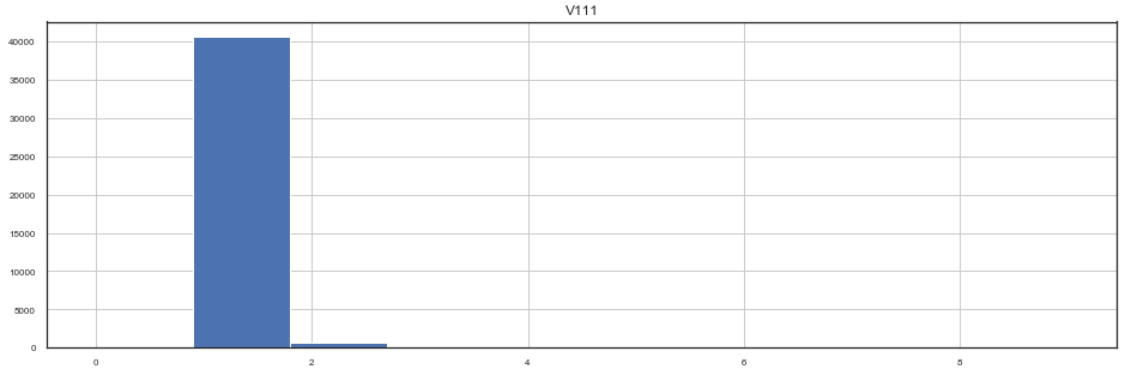

V99
40000
35000
30000
25000
20000
15000
10000
5000
0
0
20
40
60
80

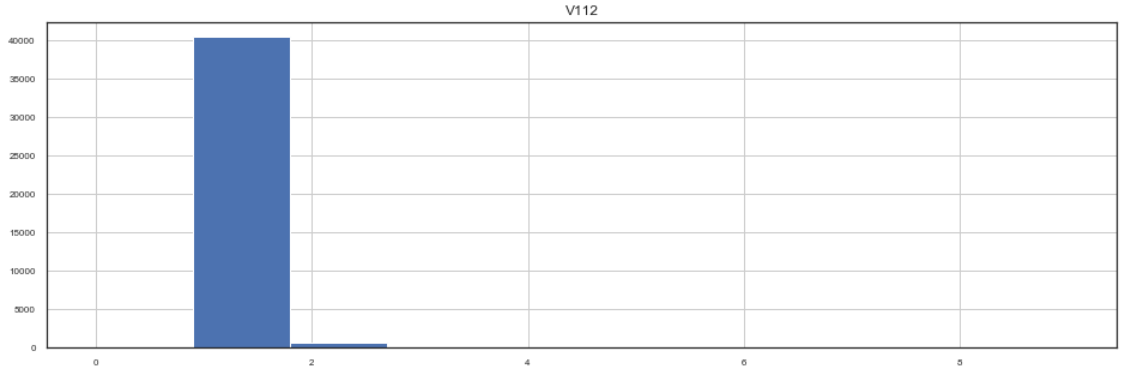

V100
40000
35000
30000
25000
20000
15000
10000
5000
0
0
5
10
15
20
25

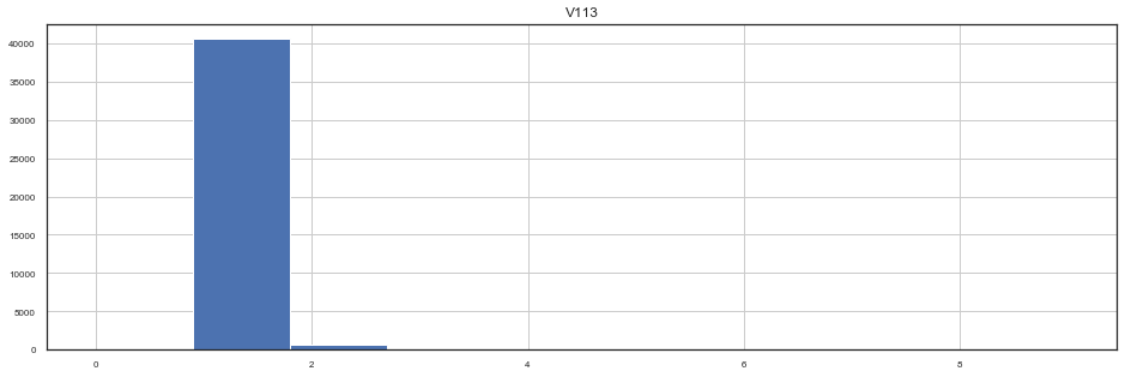

V108
40000
35000
30000
25000
20000
15000
10000
5000
0
0
1
2
3
4
5
6
7

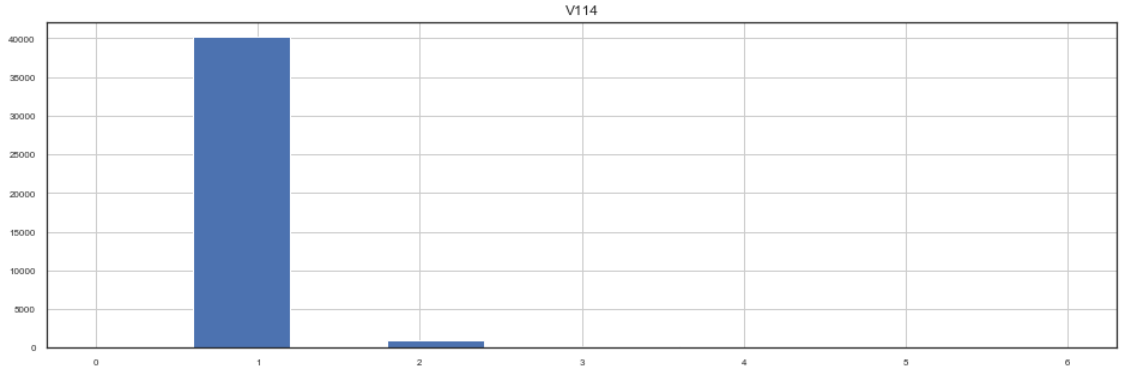

V109
40000
35000
30000
25000
20000
15000
10000
5000
0
0
1
2
3
4
5
6
7

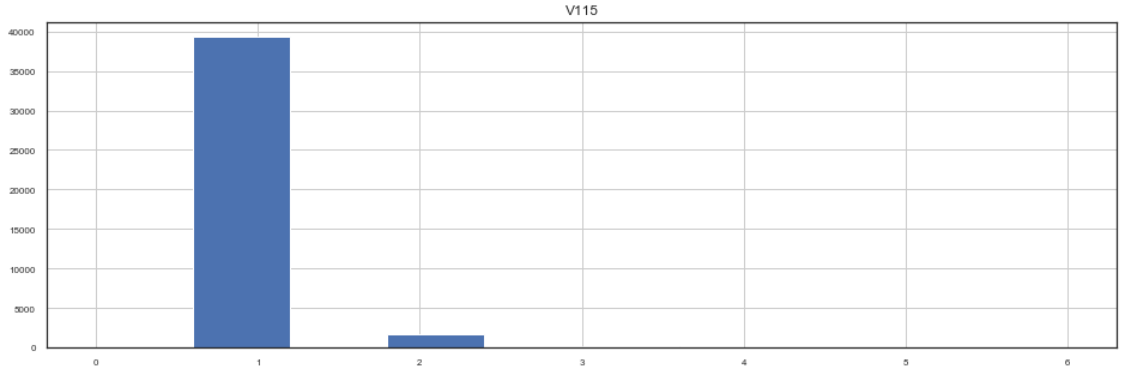

V110
40000
35000
30000
25000
20000
15000
10000
5000
0
0
1
2
3
4
5
6
7

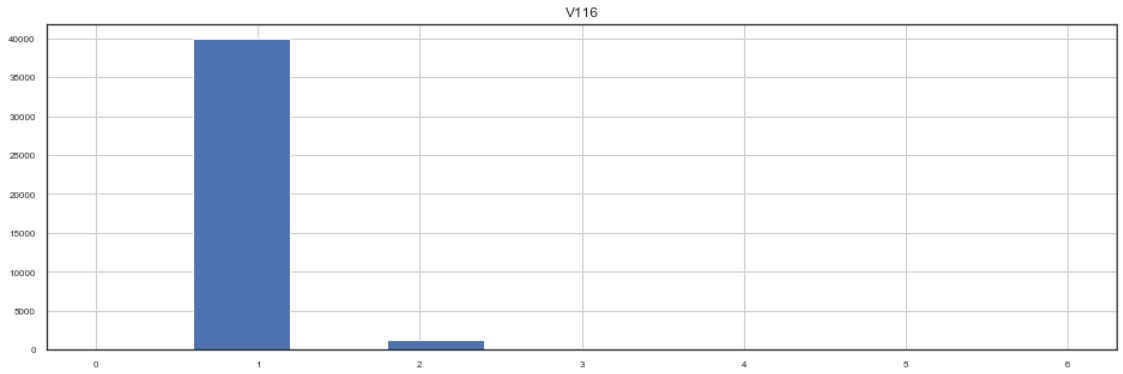

V111
40000
35000
30000
25000
20000
15000
10000
5000
0
0
2
4
6
8

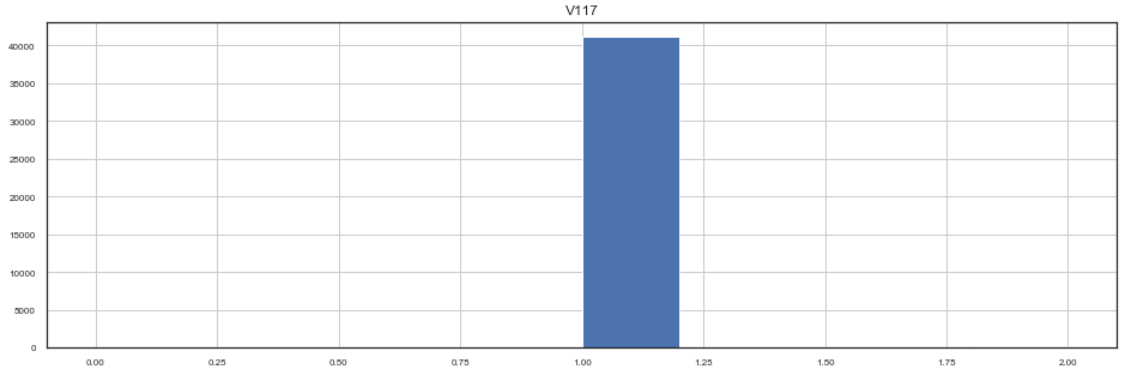

V112
40000
35000
30000
25000
20000
15000
10000
5000
0
0
2
4
6
8

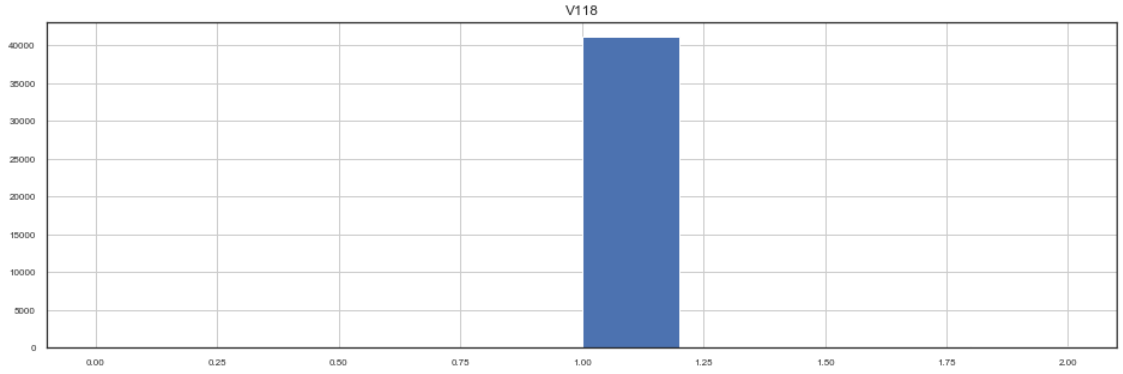

V113
40000
35000
30000
25000
20000
15000
10000
5000
0
0
2
4
6
8

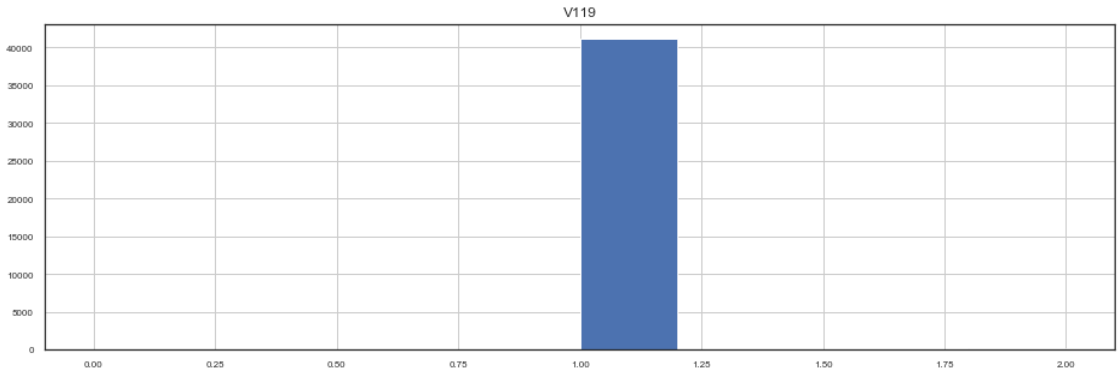

V114
40000
35000
30000
25000
20000
15000
10000
5000
0
0
1
2
3
4
5
6

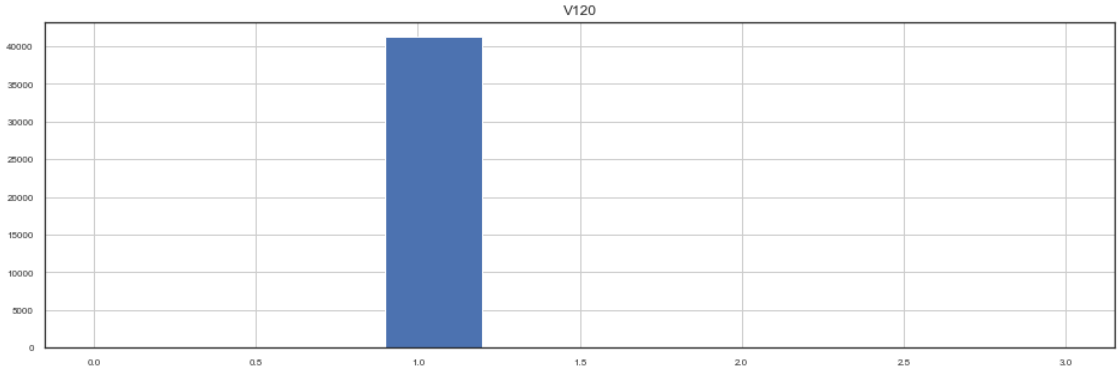

V115
40000
35000
30000
25000
20000
15000
10000
5000
0
0
1
2
3
4
5
6

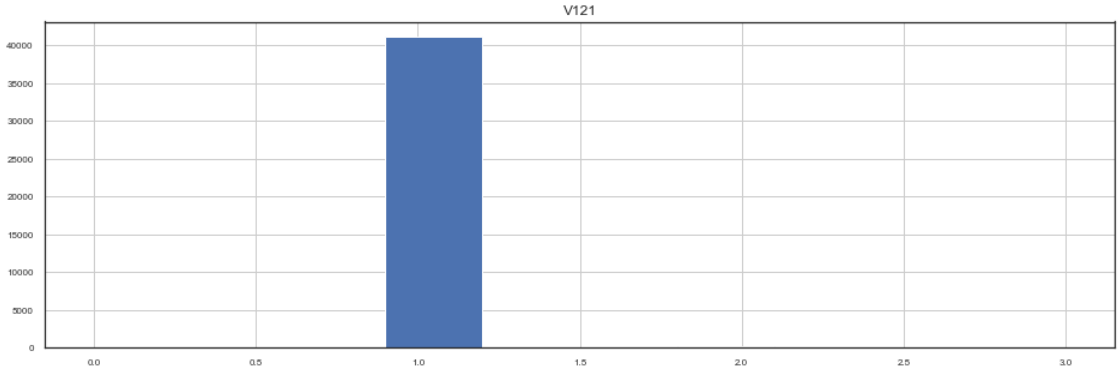

V116
40000
35000
30000
25000
20000
15000
10000
5000
0
0
1
2
3
4
5
6

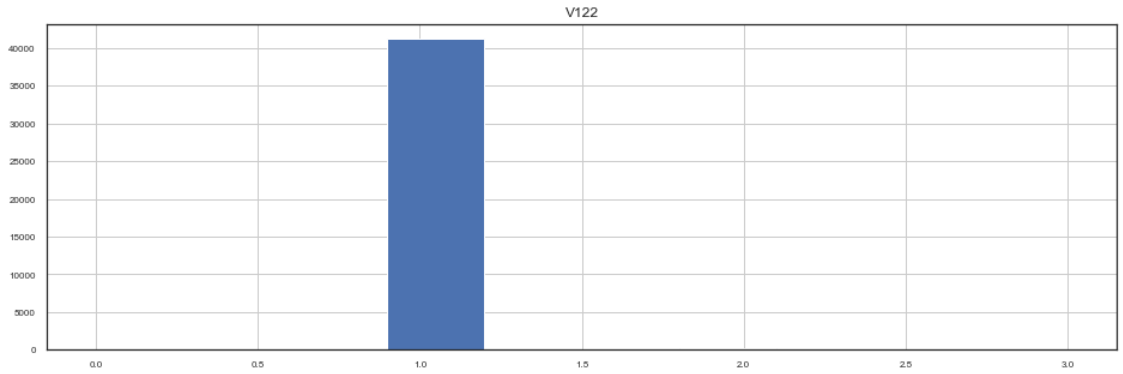

V117
40000
35000
30000
25000
20000
15000
10000
5000
0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

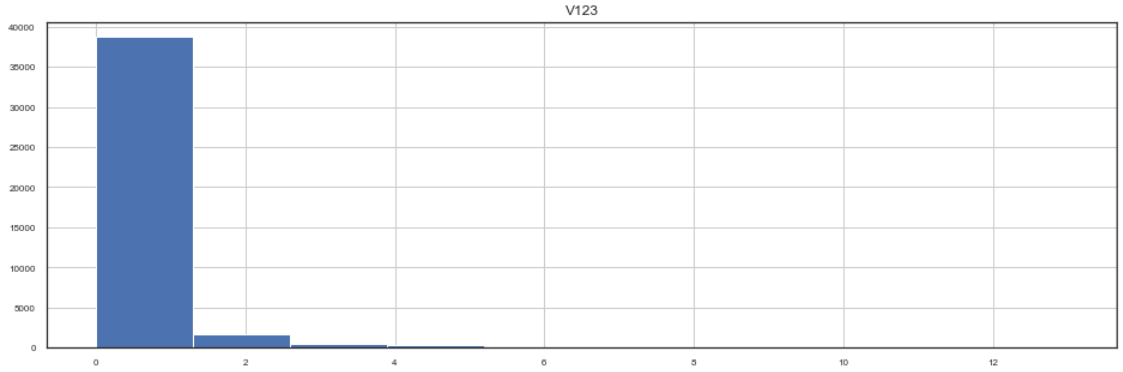

V118
40000
35000
30000
25000
20000
15000
10000
5000
0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

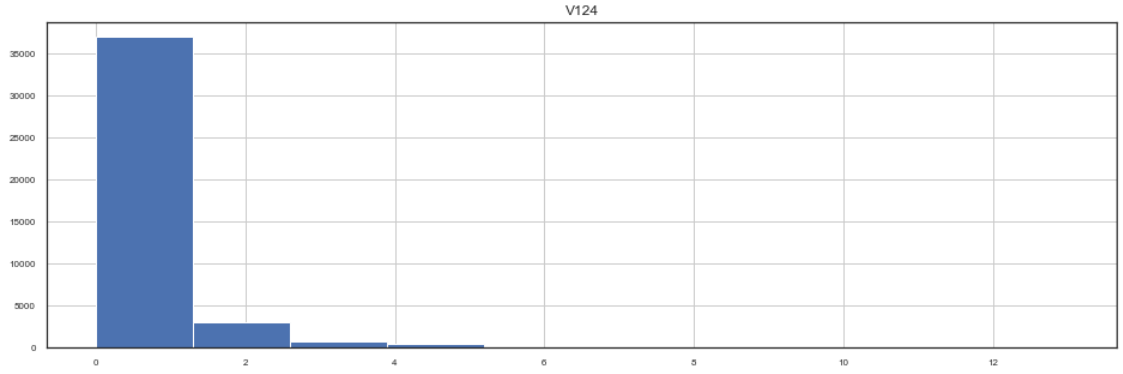

V119
40000
35000
30000
25000
20000
15000
10000
5000
0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

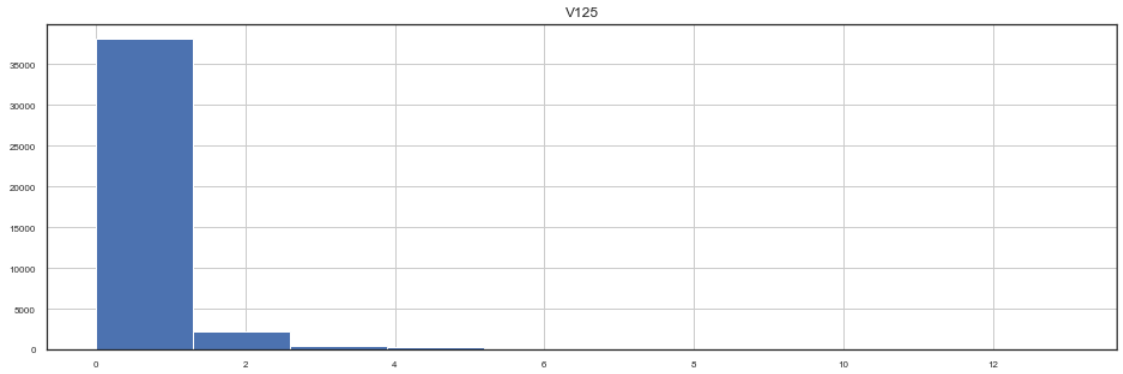

V120
40000
35000
30000
25000
20000
15000
10000
5000
0
0.0
0.5
1.0
1.5
2.0
2.5
3.0


V121
40000
35000
30000
25000
20000
15000
10000
5000
0
0.0
0.5
1.0
1.5
2.0
2.5
3.0

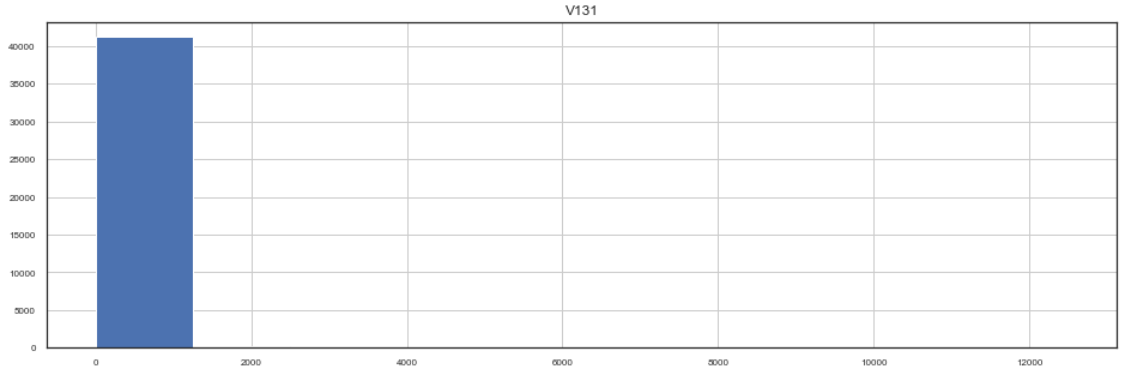

V122
40000
35000
30000
25000
20000
15000
10000
5000
0
0.0
0.5
1.0
1.5
2.0
2.5
3.0

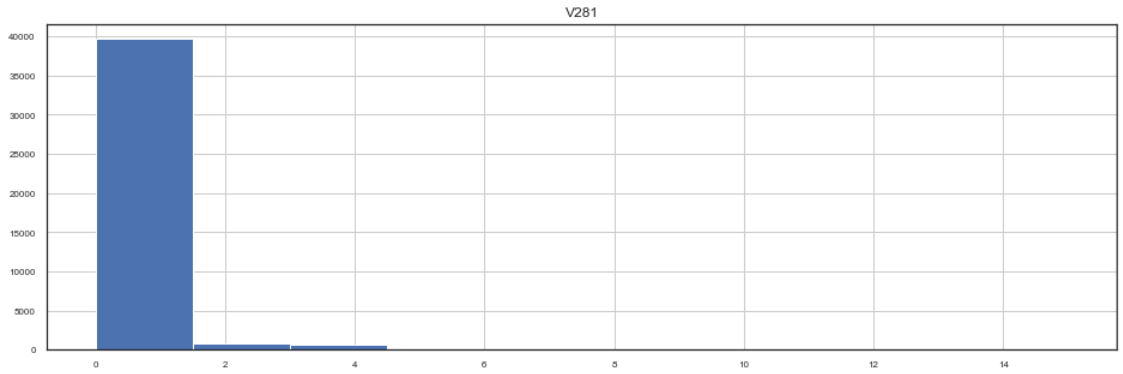

V123
40000
35000
30000
25000
20000
15000
10000
5000
0
0
2
4
6
8
10
12

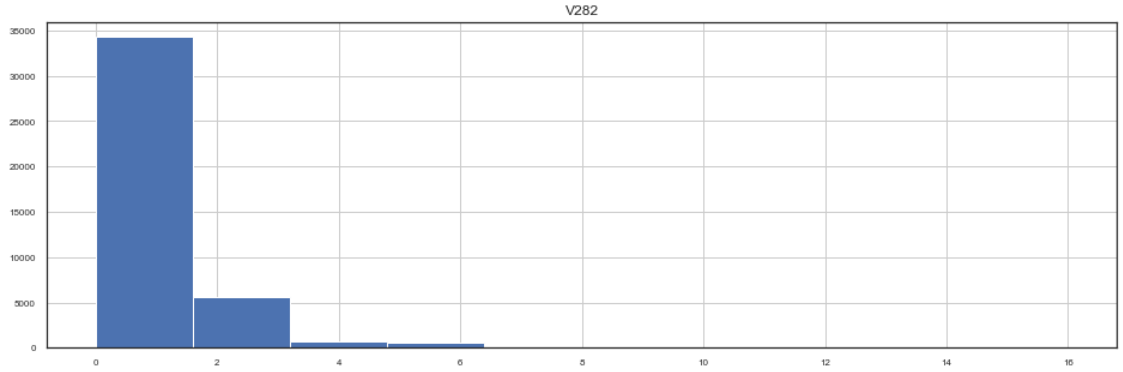

V124
35000
30000
25000
20000
15000
10000
5000
0
0
2
4
6
8
10
12

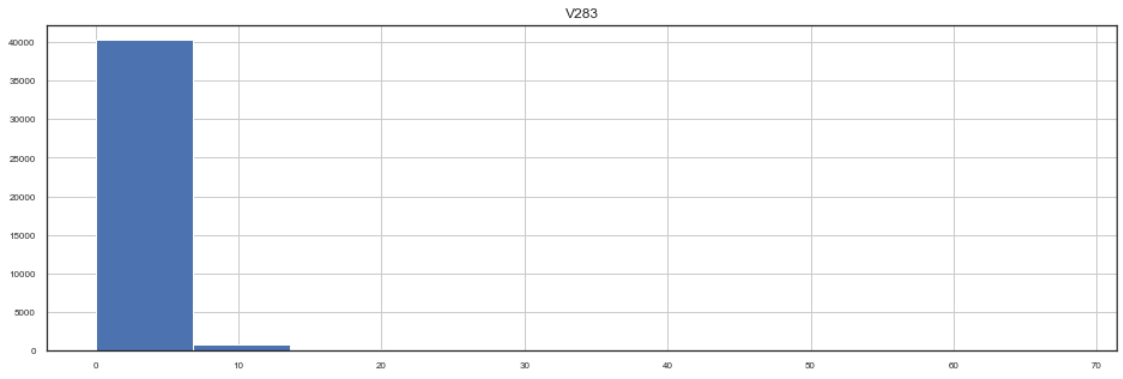

V125
35000
30000
25000
20000
15000
10000
5000
0
0
2
4
6
8
10
12

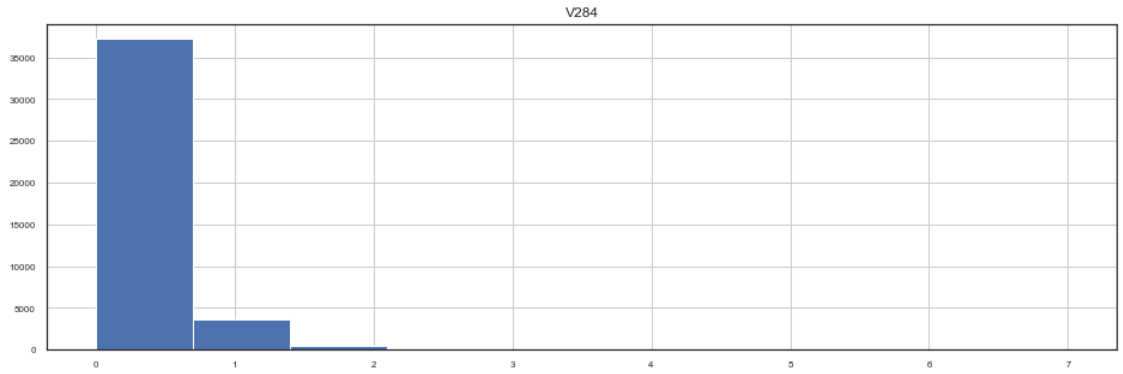

V129
40000
35000
30000
25000
20000
15000
10000
5000
0
0
2000
4000
6000
8000
10000
12000

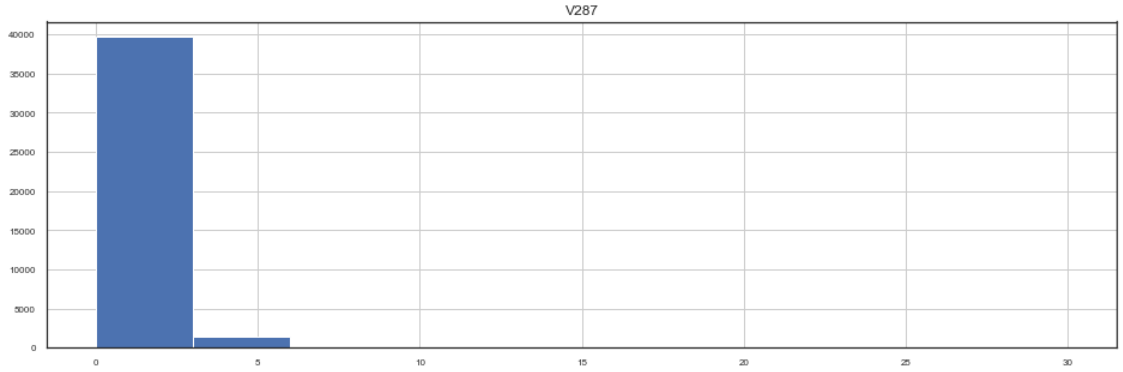

V131
40000
35000
30000
25000
20000
15000
10000
5000
0
0
2000
4000
6000
8000
10000
12000

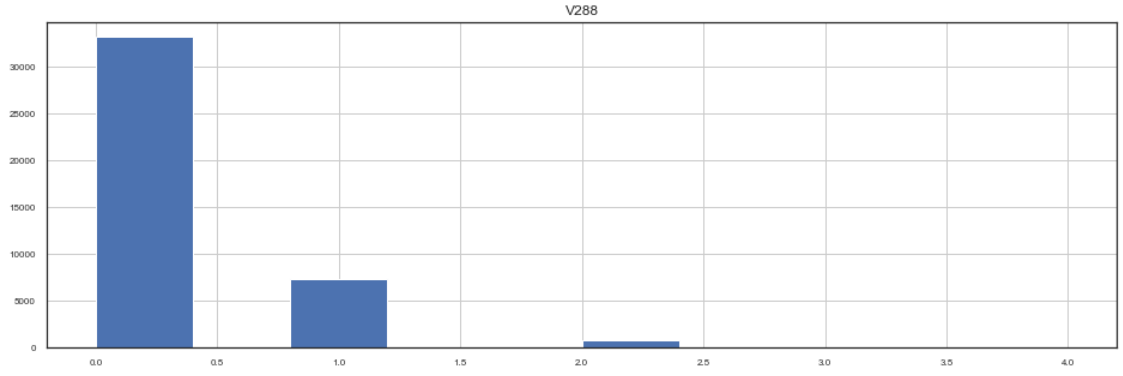

V281
40000
35000
30000
25000
20000
15000
10000
5000
0
0
2
4
6
8
10
12
14

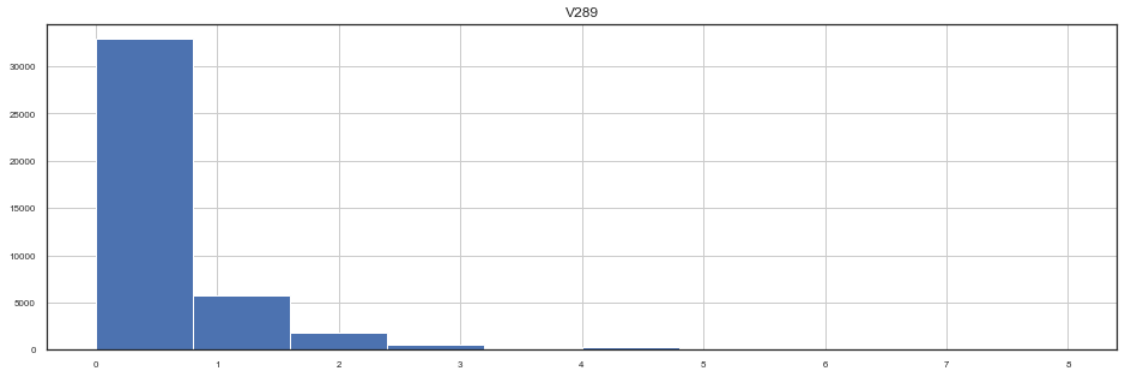

V282
35000
30000
25000
20000
15000
10000
5000
0
0
2
4
6
8
10
12
14
16

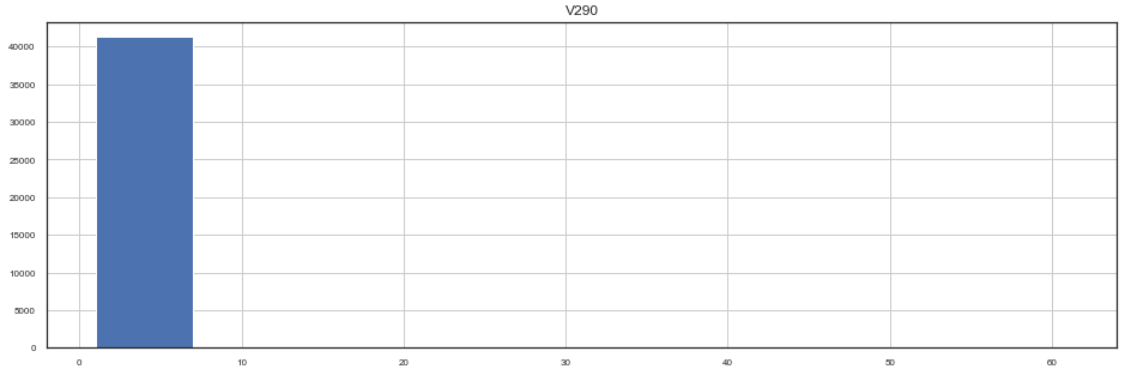

V283
40000
35000
30000
25000
20000
15000
10000
5000
0
0
10
20
30
40
50
60
70

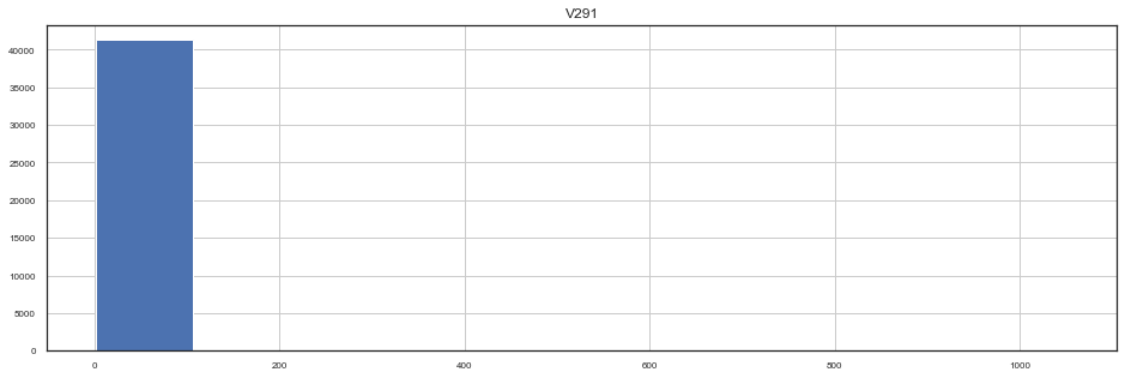

V284
35000
30000
25000
20000
15000
10000
5000
0
0
1
2
3
4
5
6
7

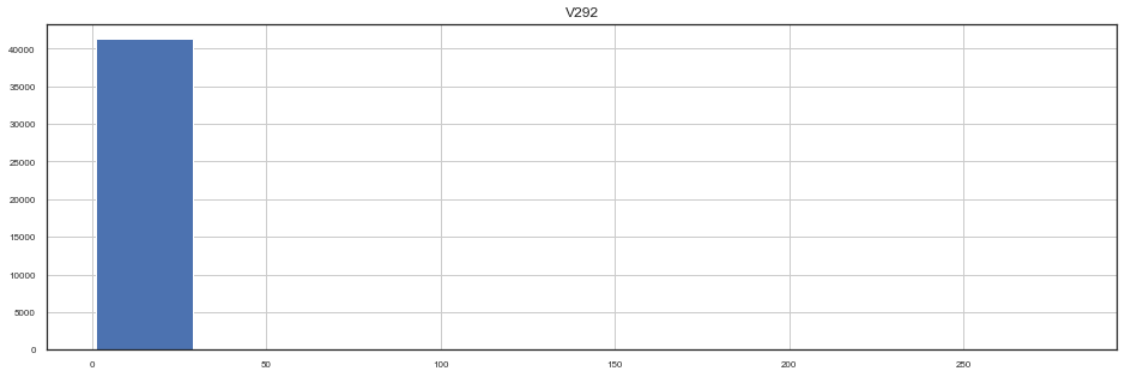

V287
40000
35000
30000
25000
20000
15000
10000
5000
0
0
5
10
15
20
25
30

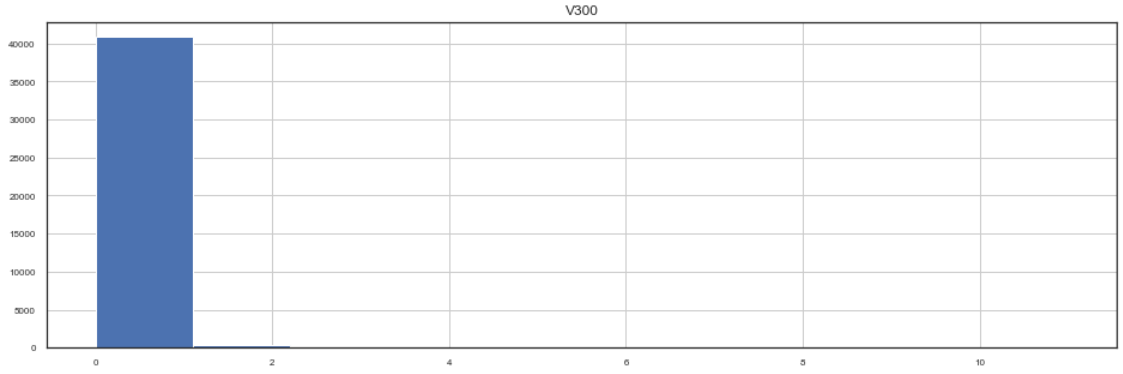

V288
30000
25000
20000
15000
10000
5000
0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0

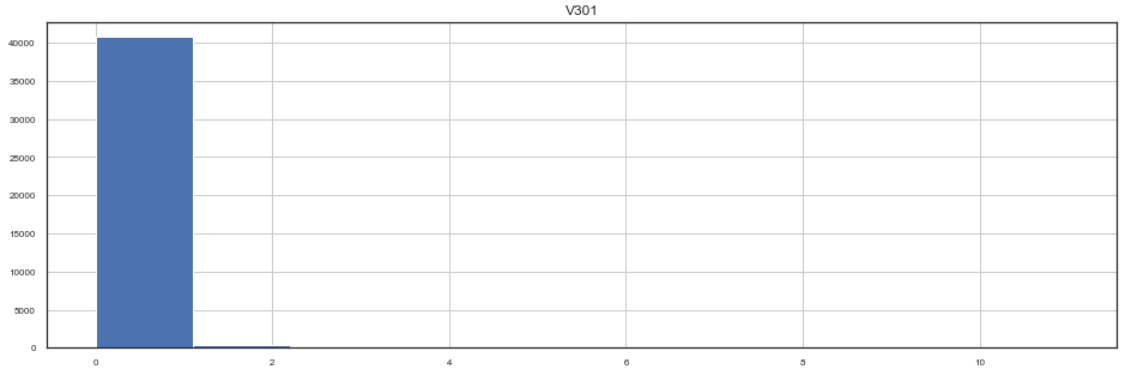

V289
30000
25000
20000
15000
10000
5000
0
0
1
2
3
4
5
6
7
8

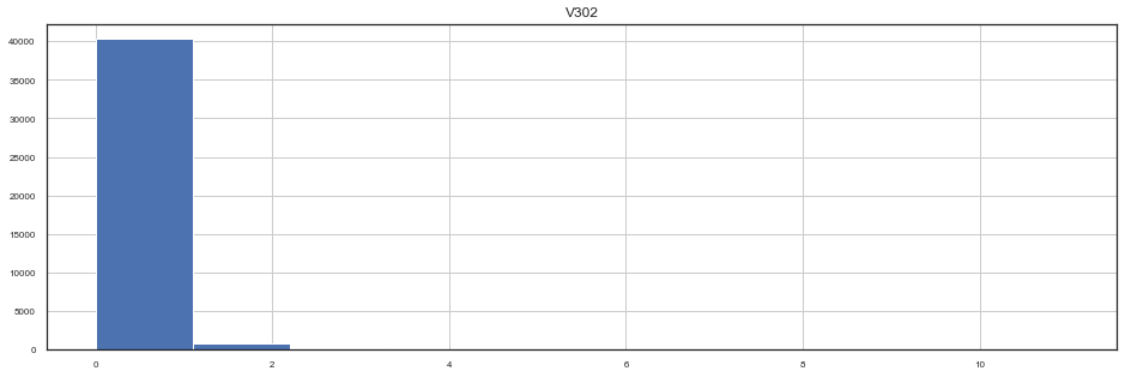

V290
40000
35000
30000
25000
20000
15000
10000
5000
0
0
10
20
30
40
50
60

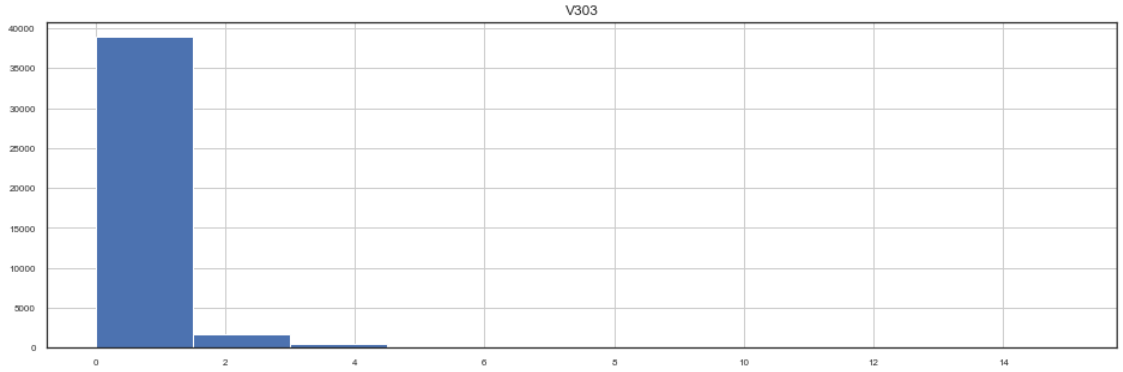

V291
40000
35000
30000
25000
20000
15000
10000
5000
0
0
200
400
600
800
1000

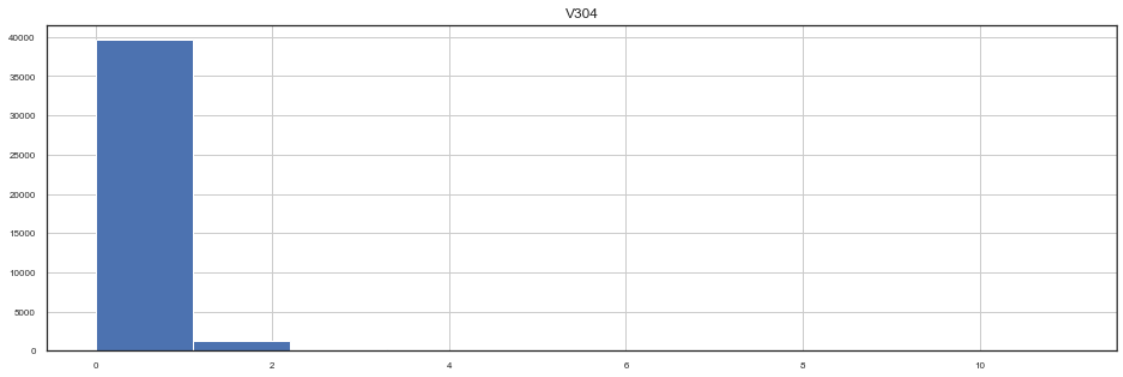

V292
40000
35000
30000
25000
20000
15000
10000
5000
0
0
50
100
150
200
250

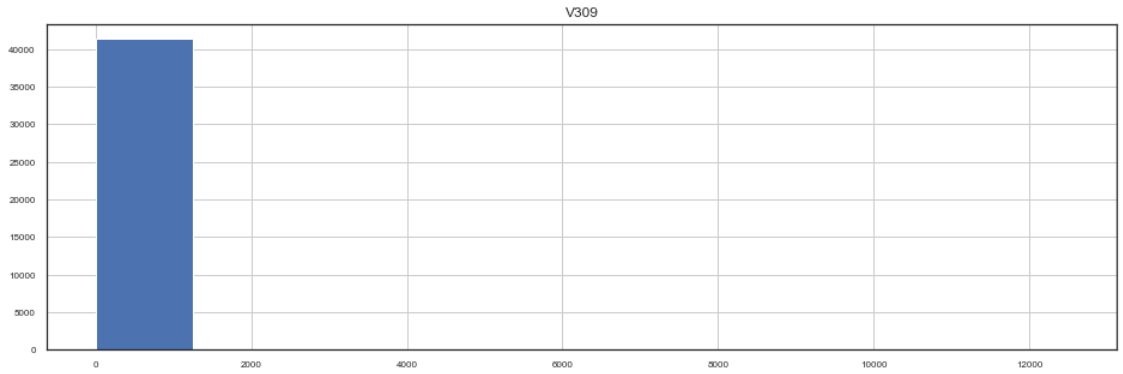

V300
40000
35000
30000
25000
20000
15000
10000
5000
0
0
2
4
6
5
10

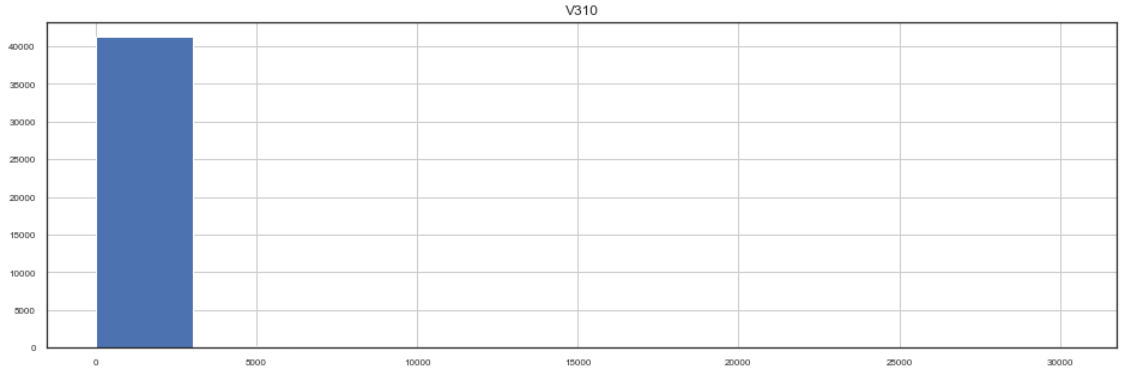

V301
40000
35000
30000
25000
20000
15000
10000
5000
0
0
2
4
6
5
10


V302
40000
35000
30000
25000
20000
15000
10000
5000
0
0
2
4
6
8
10


V303
40000
35000
30000
25000
20000
15000
10000
5000
0
0
2
4
6
8
10
12
14

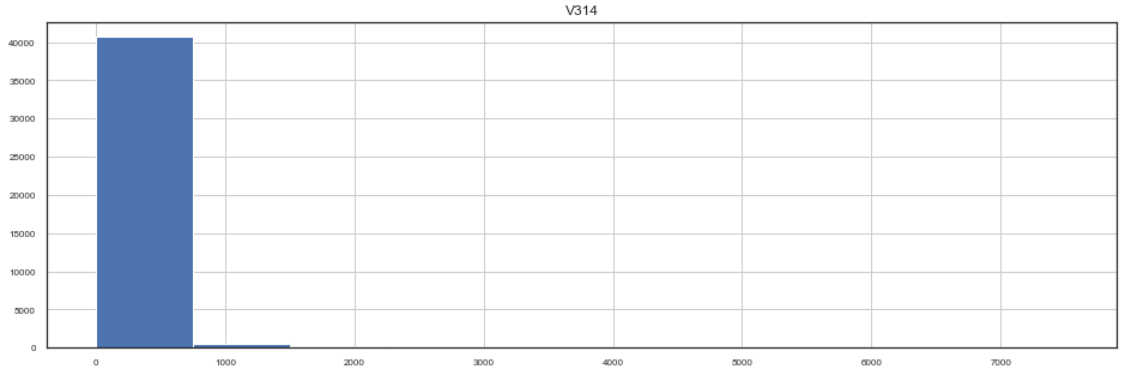

V304
40000
35000
30000
25000
20000
15000
10000
5000
0
0
2
4
6
8
10

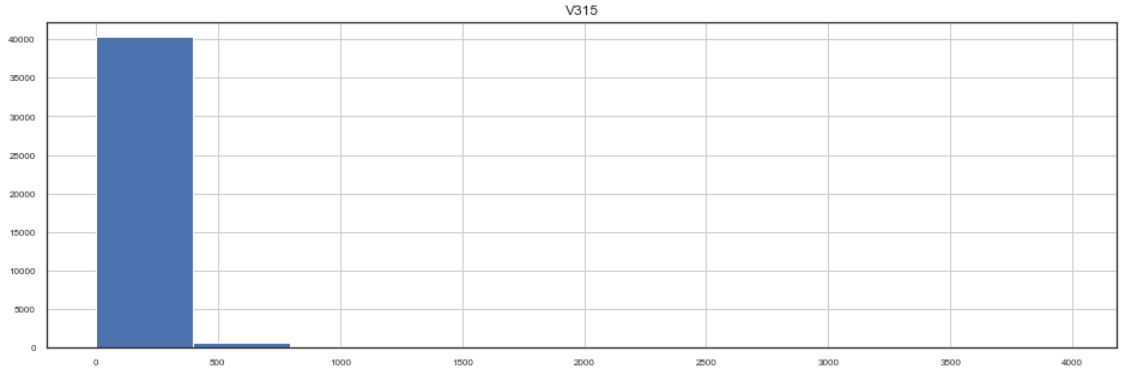

V309
40000
35000
30000
25000
20000
15000
10000
5000
0
0
2000
4000
6000
8000
10000
12000

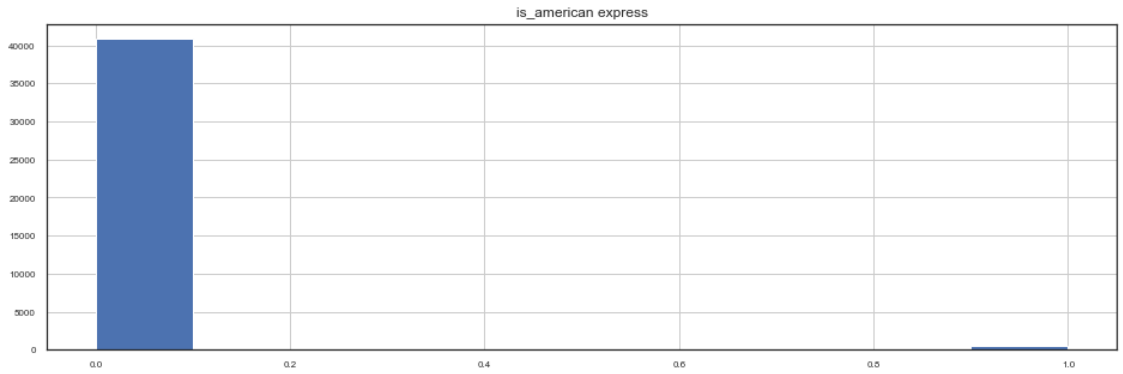

V310
40000
35000
30000
25000
20000
15000
10000
5000
0
0
5000
10000
15000
20000
25000
30000

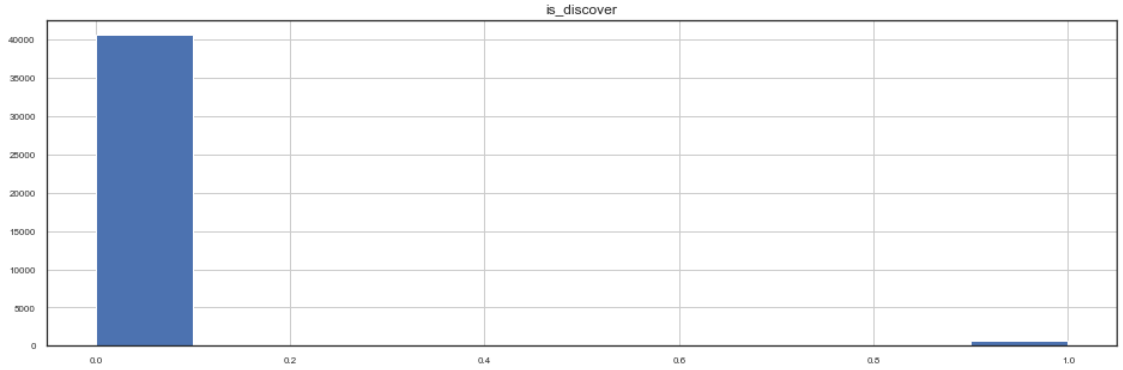

V312
40000
35000
30000
25000
20000
15000
10000
5000
0
0
2000
4000
6000
8000
10000
12000

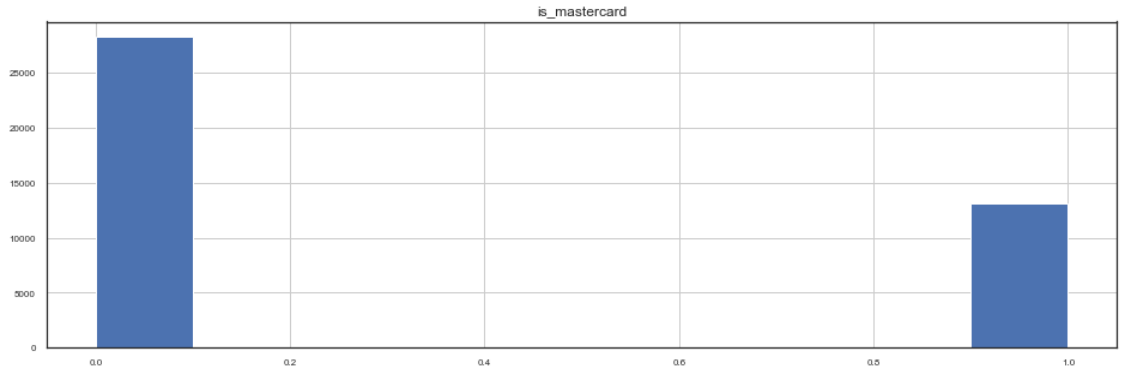

V313
40000
35000
30000
25000
20000
15000
10000
5000
0
0
500
1000
1500
2000
2500
3000

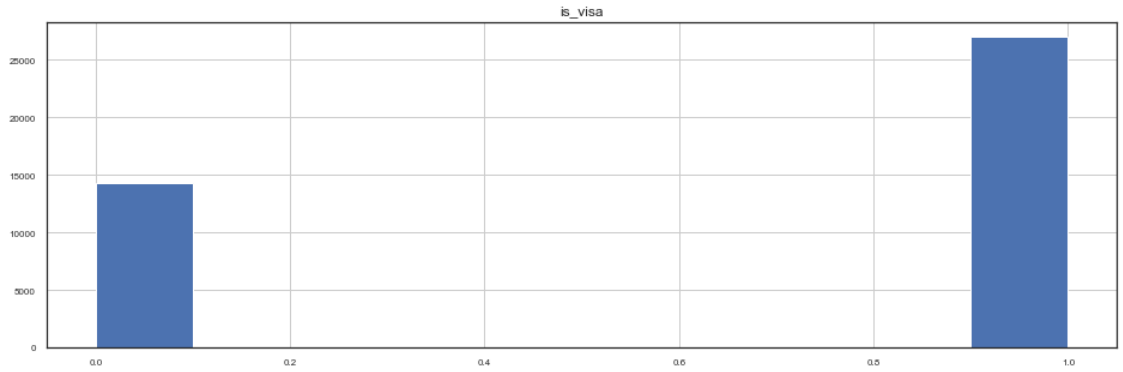

V314
40000
35000
30000
25000
20000
15000
10000
5000
0
0
1000
2000
3000
4000
5000
6000
7000


V315
40000
35000
30000
25000
20000
15000
10000
5000
0
0
500
1000
1500
2000
2500
3000
3500
4000

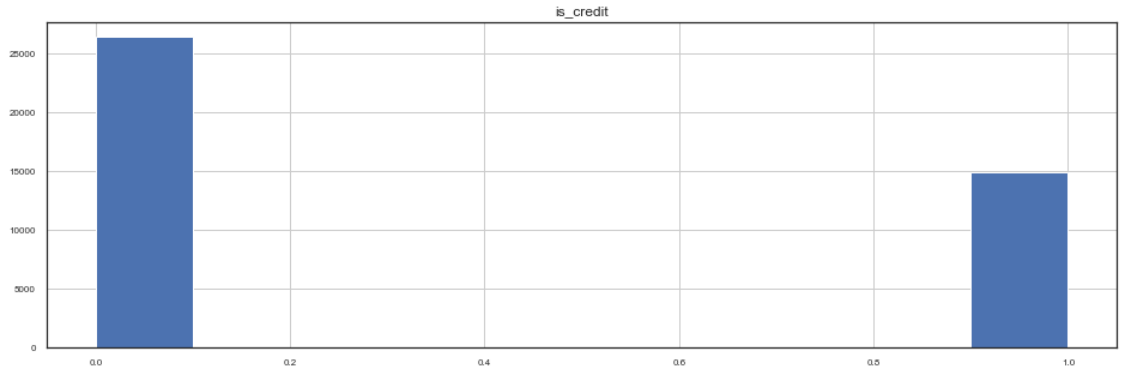

is_american express
40000
35000
30000
25000
20000
15000
10000
5000
0
0.0
0.2
0.4
0.6
0.8
1.0

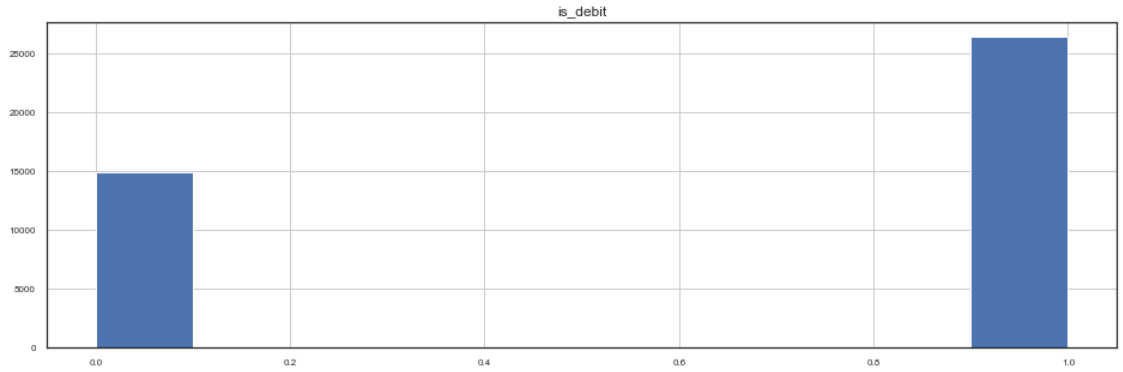

is_discover
40000
35000
30000
25000
20000
15000
10000
5000
0
0.0
0.2
0.4
0.6
0.8
1.0

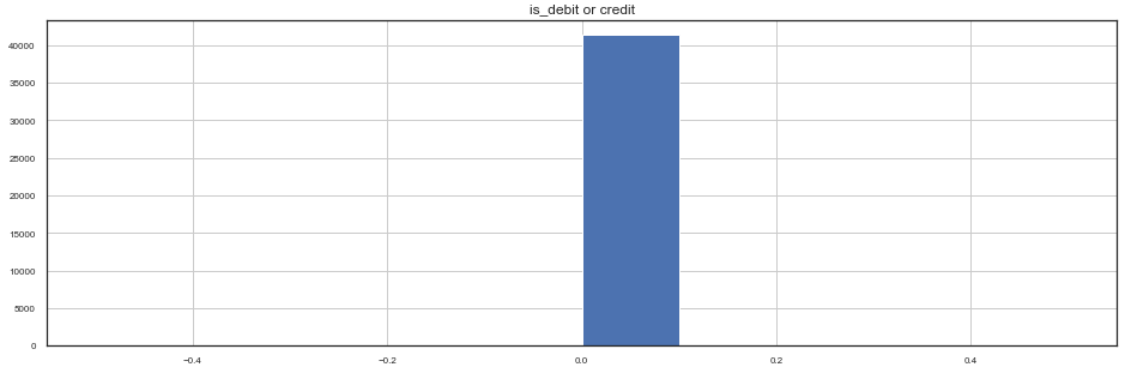

is_mastercard
25000
20000
15000
10000
5000
0
0.0
0.2
0.4
0.6
0.8
1.0

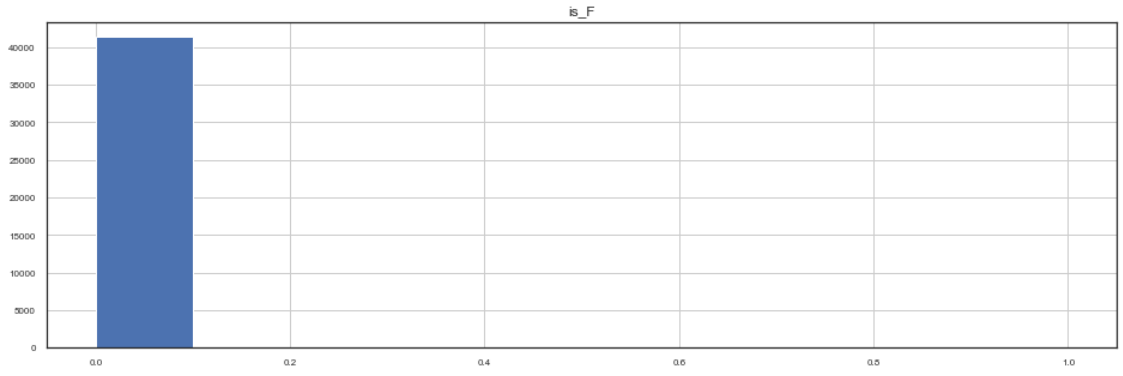

is_visa
25000
20000
15000
10000
5000
0
0.0
0.2
0.4
0.6
0.8
1.0

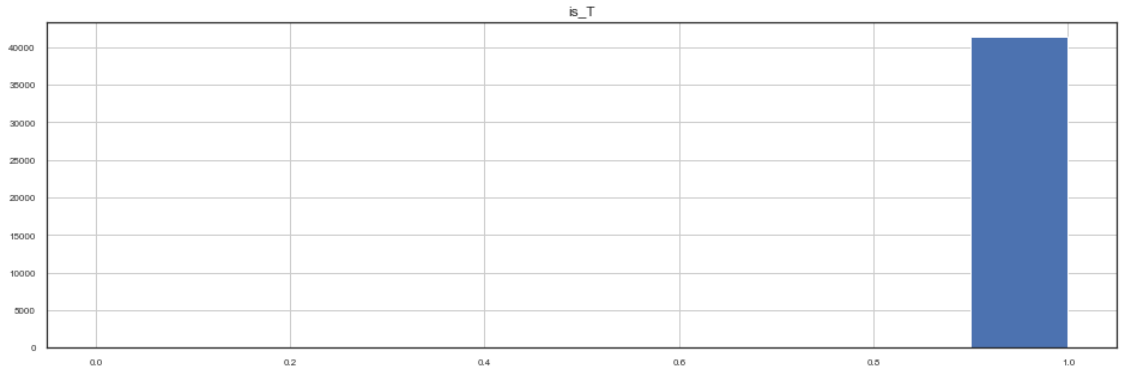

is_charge card
40000
35000
30000
25000
20000
15000
10000
5000
0
0.0
0.2
0.4
0.6
0.8
1.0

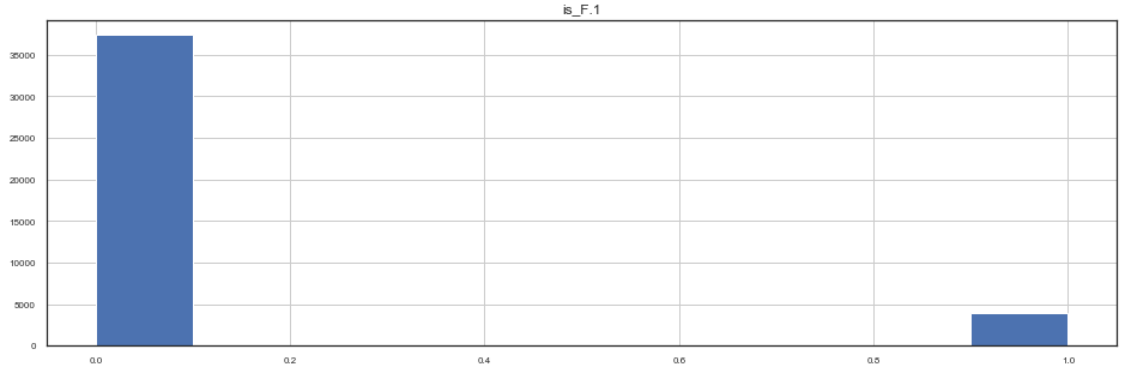

is_credit
25000
20000
15000
10000
5000
0
0.0
0.2
0.4
0.6
0.8
1.0

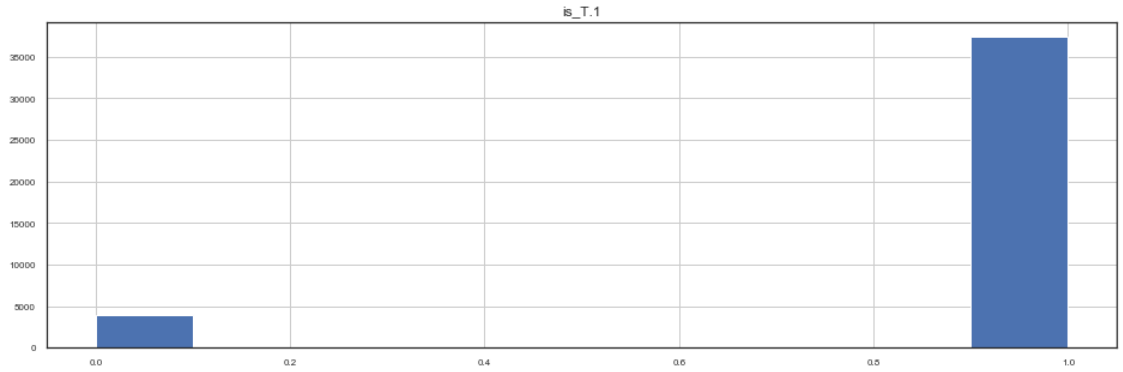

is_debit
25000
20000
15000
10000
5000
0
0.0
0.2
0.4
0.6
0.8
1.0

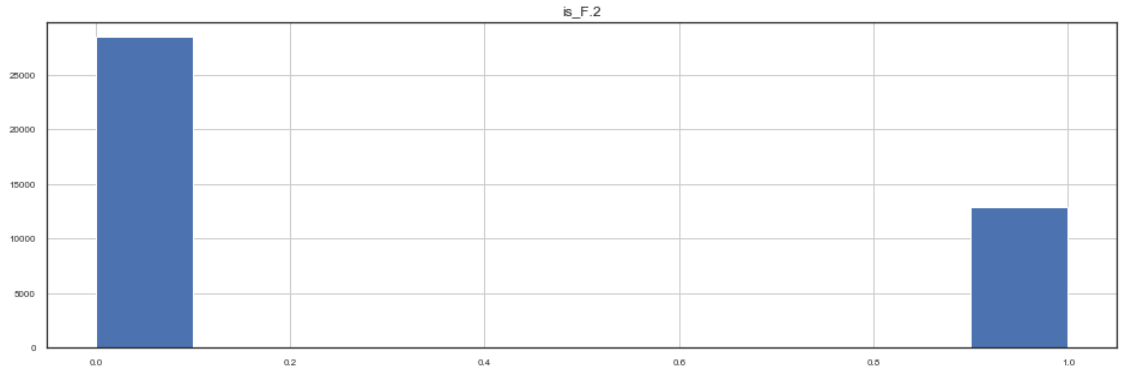

is_debit or credit
40000
35000
30000
25000
20000
15000
10000
5000
0
−0.4
−0.2
0.0
0.2
0.4

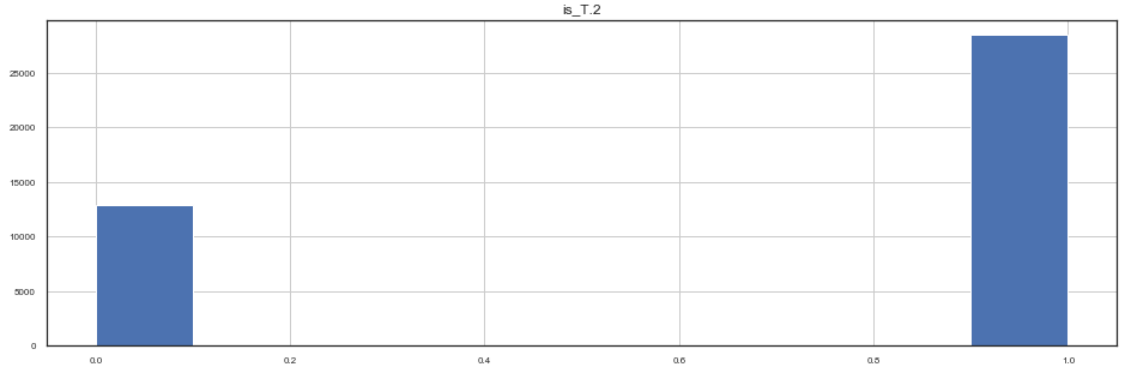

is_F
40000
35000
30000
25000
20000
15000
10000
5000
0
0.0
0.2
0.4
0.6
0.8
1.0

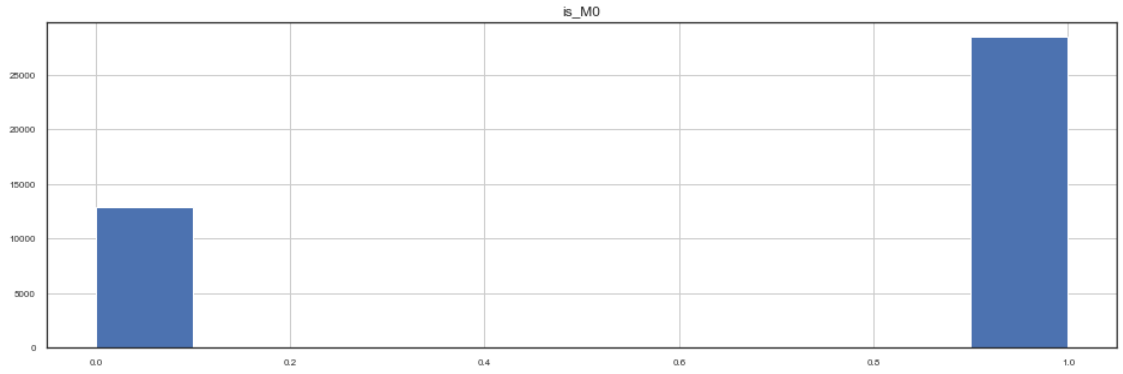

is_T
40000
35000
30000
25000
20000
15000
10000
5000
0
0.0
0.2
0.4
0.6
0.8
1.0

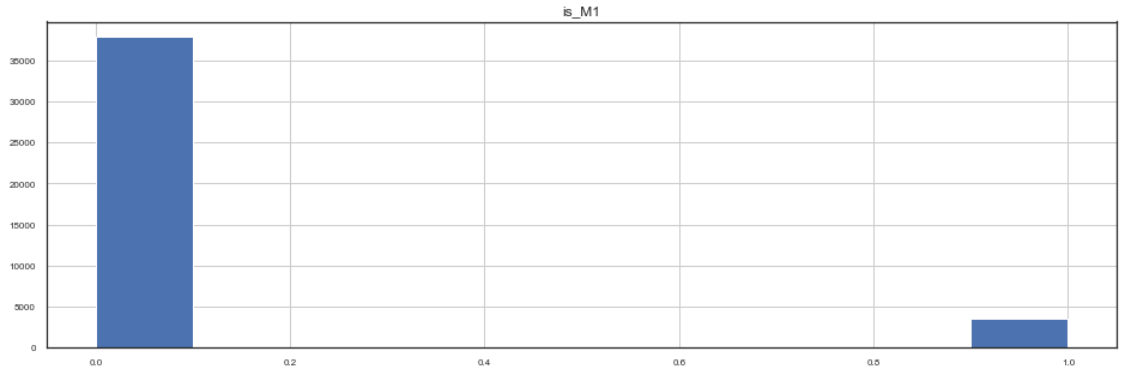

is_F.1
35000
30000
25000
20000
15000
10000
5000
0
0.0
0.2
0.4
0.6
0.8
1.0

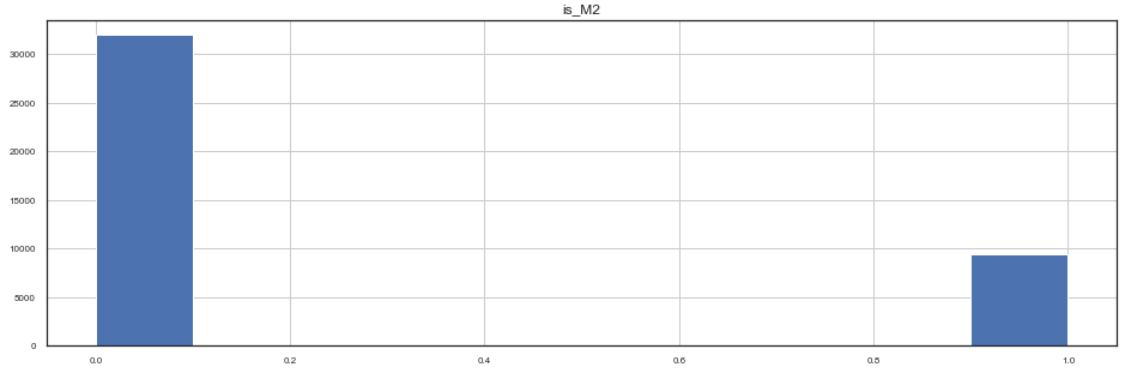

is_T.1
35000
30000
25000
20000
15000
10000
5000
0
0.0
0.2
0.4
0.6
0.8
1.0

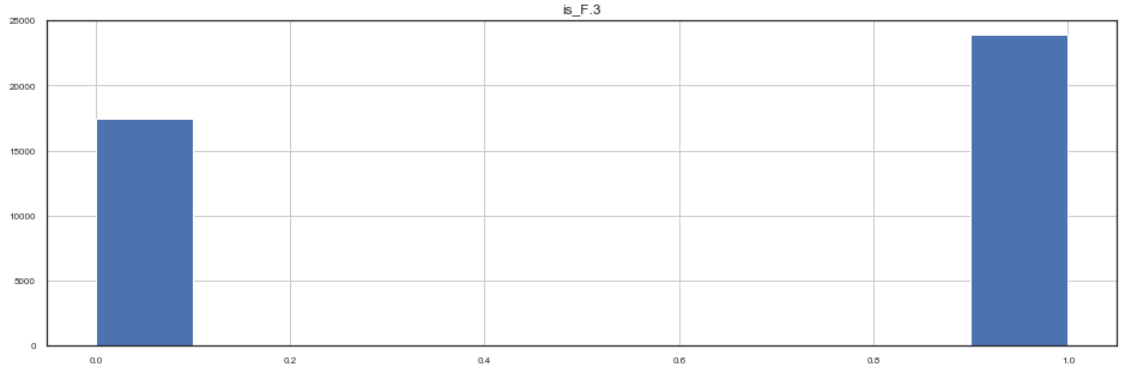

is_F.2
25000
20000
15000
10000
5000
0
0.0
0.2
0.4
0.6
0.8
1.0

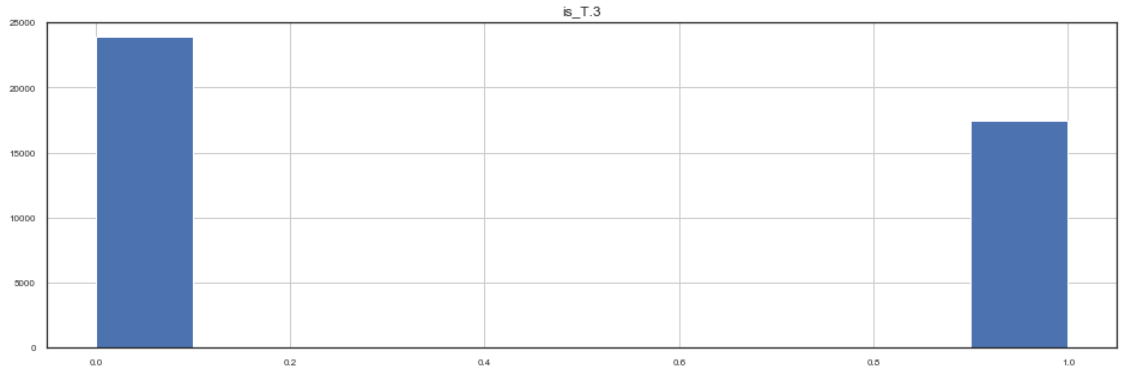

is_T.2
25000
20000
15000
10000
5000
0
0.0
0.2
0.4
0.6
0.8
1.0

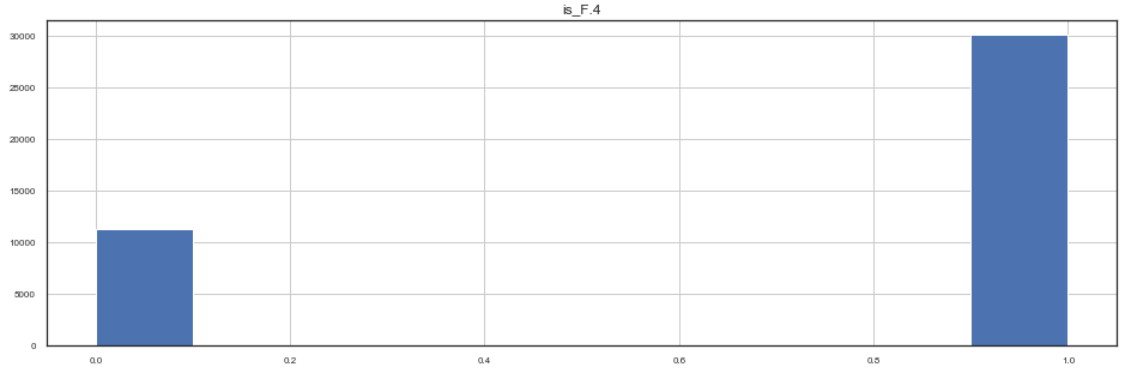

is_M0
25000
20000
15000
10000
5000
0
0.0
0.2
0.4
0.6
0.8
1.0

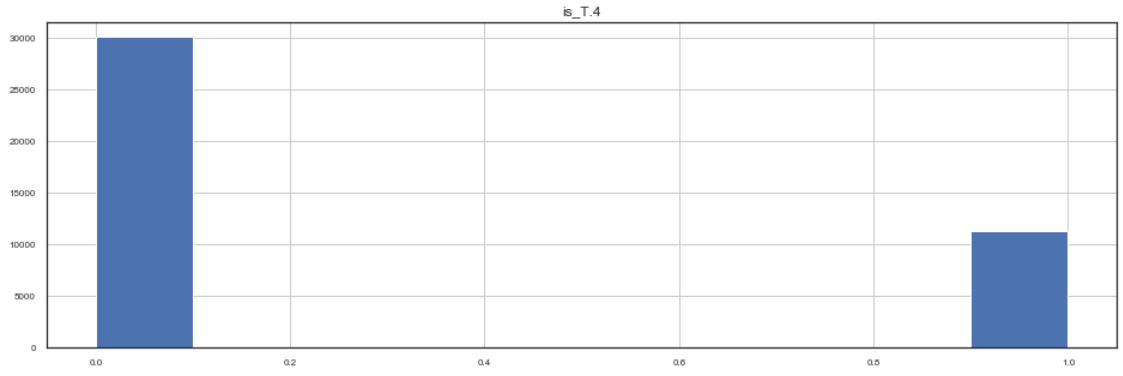

is_M1
35000
30000
25000
20000
15000
10000
5000
0
0.0
0.2
0.4
0.6
0.8
1.0

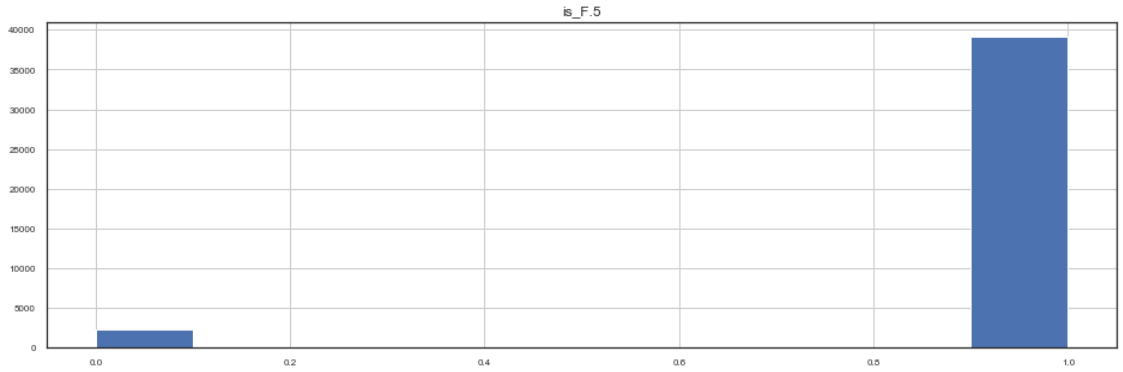

is_M2
30000
25000
20000
15000
10000
5000
0
0.0
0.2
0.4
0.6
0.8
1.0

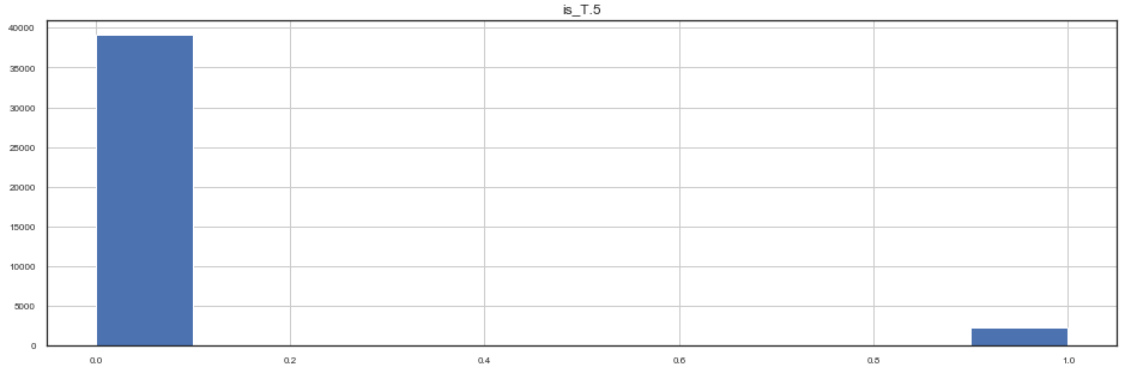

is_F.3
25000
20000
15000
10000
5000
0
0.0
0.2
0.4
0.6
0.8
1.0

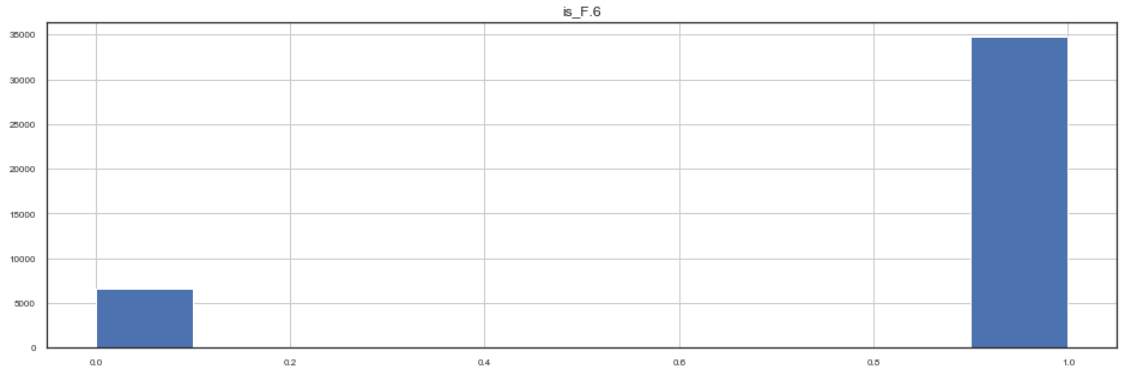

is_T.3
25000
20000
15000
10000
5000
0
0.0
0.2
0.4
0.6
0.8
1.0

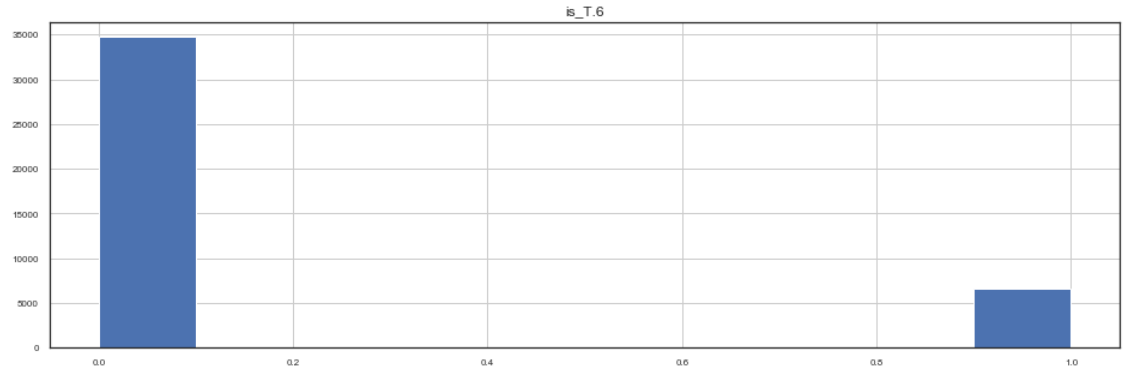

is_F.4
30000
25000
20000
15000
10000
5000
0
0.0
0.2
0.4
0.6
0.8
1.0

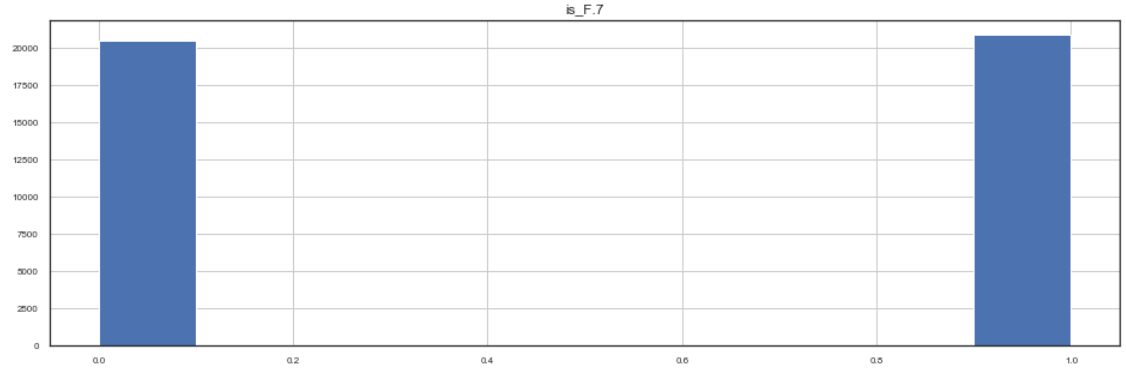

is_T.4
30000
25000
20000
15000
10000
5000
0
0.0
0.2
0.4
0.6
0.8
1.0

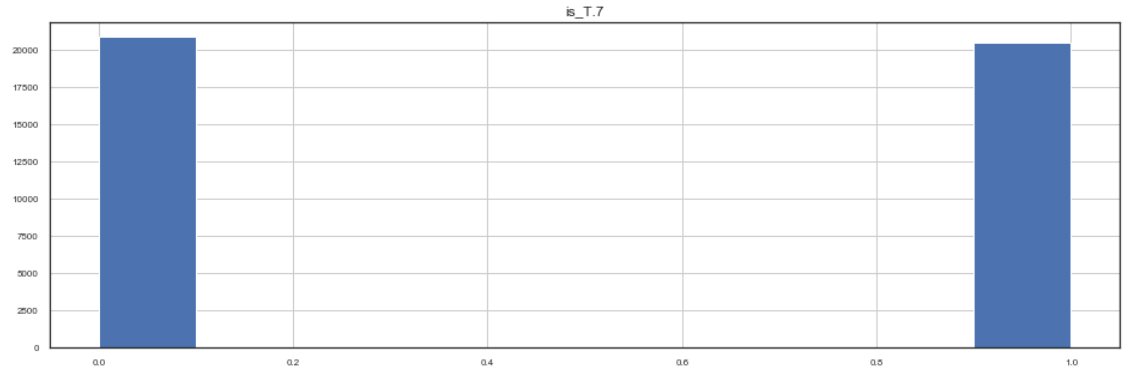

is_F.5
40000
35000
30000
25000
20000
15000
10000
5000
0
0.0
0.2
0.4
0.6
0.8
1.0


is_T.5
40000
35000
30000
25000
20000
15000
10000
5000
0
0.0
0.2
0.4
0.6
0.8
1.0


is_F.6
35000
30000
25000
20000
15000
10000
5000
0
0.0
0.2
0.4
0.6
0.8
1.0


is_T.6
35000
30000
25000
20000
15000
10000
5000
0
0.0
0.2
0.4
0.6
0.8
1.0


is_F.7
20000
17500
15000
12500
10000
7500
5000
2500
0
0.0
0.2
0.4
0.6
0.8
1.0


is_T.7
20000
17500
15000
12500
10000
7500
5000
2500
0
0.0
0.2
0.4
0.6
0.8
1.0

[ ]:

TrainingDataPreprocessing

April 22, 2020

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import gc
```

[2]:
```
#Load Training Data
train = pd.read_csv("./input/raw/train_transaction.csv",)
train_id = pd.read_csv("./input/raw/train_identity.csv",)

train = train.merge(train_id, how='left', left_index=True, right_index=True)
print(train.shape)
del train_id; x=gc.collect()
```

```
(590540, 435)
```

[3]:
```
print(train.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 590540 entries, 0 to 590539
Columns: 435 entries, TransactionID_x to DeviceInfo
dtypes: float64(400), int64(4), object(31)
memory usage: 1.9+ GB
None
```

[3]:
```
_60Percent = (60 * len(train)) / 100.0
_40Percent = (40 * len(train)) / 100.0
df = pd.DataFrame()
dfc = pd.DataFrame()
df["isFraud"] = train['isFraud'].copy()
```

[4]:
```
for (columnName, columnData) in train.iteritems():
        if(columnData.dtype != object):
            if(_60Percent >= columnData.isna().sum()):#columnName !=␣
↪'TransactionID' and if(columnName != 'TransactionDT'):
                if(columnName != 'TransactionDT'):
                    df[columnName] = columnData
        else:
```

```
            if(_40Percent >= columnData.isna().sum()):
                dfc[columnName] = columnData
```

[5]:
```
fraud = df[df['isFraud'] == 1]
fraud = fraud.fillna(fraud.mean())
notfraud = df[df['isFraud'] == 0]
notfraud = notfraud.fillna(notfraud.mean())
fraud.reset_index(drop=True, inplace=True)
notfraud.reset_index(drop=True, inplace=True)
```

[6]:
```
dfc['TransactionID_x'] = df['TransactionID_x']
dfc = dfc.sort_values('TransactionID_x')
dfc = dfc.set_index('TransactionID_x')
dfc = dfc.drop(columns=['P_emaildomain'])

df =pd.concat([fraud, notfraud], axis=0)
df = df.sort_values('TransactionID_x')
df = df.set_index('TransactionID_x')
del fraud, notfraud; x=gc.collect()
```

[7]:
```
df = df.fillna(df.mean())
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 590540 entries, 2987000 to 3577539
Columns: 211 entries, isFraud to V321
dtypes: float64(209), int64(2)
memory usage: 955.2 MB
```

[8]:
```
for (columnName, columnData) in df.iteritems():
    if(df['isFraud'].corr(columnData) > 0.01 or df['isFraud'].corr(columnData)␣
 ↪< -0.01):
        continue
    else:
        df = df.drop(columns=[columnName])
```

[9]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 590540 entries, 2987000 to 3577539
Columns: 155 entries, isFraud to V315
dtypes: float64(153), int64(2)
memory usage: 702.9 MB
```

[10]:
```
def predictMissing(model, encoder, X_train, y_train, X_test):
    y_train = encoder.fit_transform(y_train)
    model.fit(X_train, y_train)
```

```
    y_pred = model.predict(X_test)
    y_pred = encoder.inverse_transform(y_pred)
    y_train = encoder.inverse_transform(y_train)
    return y_pred, y_train
```

[11]:
```
from sklearn import preprocessing
from xgboost import XGBClassifier


for (columnName, columnData) in dfc.iteritems():
    print('Started Processing', columnName)
    dftmp = df
    dftmp[columnName] = columnData

    tmptrain = dftmp.dropna()
    tmptest = dftmp[dftmp[columnName].isna()]

    target = tmptrain[columnName].copy()
    tmptrain = tmptrain.select_dtypes(exclude=['object'])
    tmptest = tmptest.select_dtypes(exclude=['object'])

    clf=KNeighborsClassifier(n_neighbors=7, n_jobs=-1)
    clf_xgb = XGBClassifier(max_depth = 10,
                            objective = 'multi:softmax',
                            num_class = columnData.nunique(),
                            n_estimators = 400,
                            tree_method='gpu_hist',
                            gpu_id=0)

    y_pred, y_train = predictMissing(clf_xgb, preprocessing.
 ↪LabelEncoder(),tmptrain,target, tmptest)

    tmptrain[columnName] = y_train
    tmptest[columnName] = y_pred

    dftmp = pd.concat([tmptrain, tmptest], axis=0, sort=True)
    df.sort_index(inplace=True)
    dftmp.sort_index(inplace=True)

    df[columnName] = dftmp[columnName]
    print('Finished Processing', columnName)
    del dftmp, tmptrain, tmptest; x=gc.collect()
```

```
Started Processing ProductCD
Finished Processing ProductCD
Started Processing card4
Finished Processing card4
```

```
Started Processing card6
Finished Processing card6
Started Processing M6
Finished Processing M6
```

[12]:
```
dfc = df.select_dtypes('object')
df = df.select_dtypes(exclude=['object'])
dfDummies = pd.get_dummies(dfc, prefix='is')
#['ProductCD','card4' ,'card6' ,'M1' ,'M2' ,'M3' ,'M4' ,'M5' ,'M6' ,'M7' ,'M8'
 ↪,'M9']
```

[13]:
```
df = pd.concat([df, dfDummies], axis=1, sort=False)
```

[14]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 590540 entries, 2987000 to 3577539
Columns: 170 entries, isFraud to is_T
dtypes: float64(153), int64(2), uint8(15)
memory usage: 711.3 MB
```

[15]:
```
df.to_csv (r'./input/processed/data.csv', index = True, header=True)
```

DataBalancing

April 22, 2020

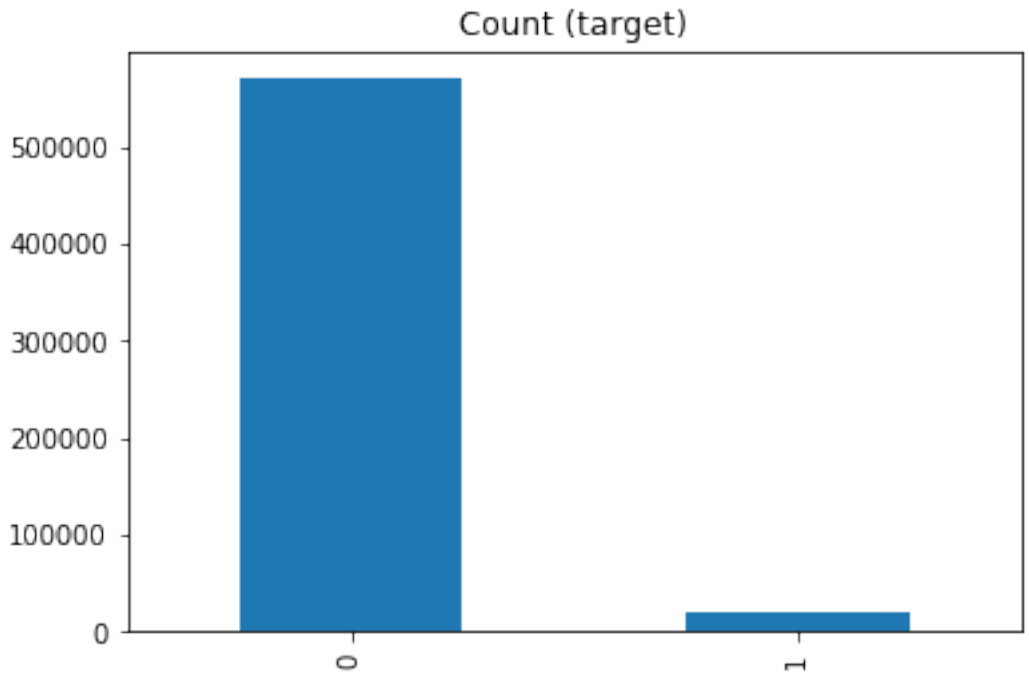
[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import gc
```

[2]:
```
#Load Data
train = pd.read_csv("./input/processed/data.csv",index_col='TransactionID_x',)
```

[3]:
```
train.info()
```

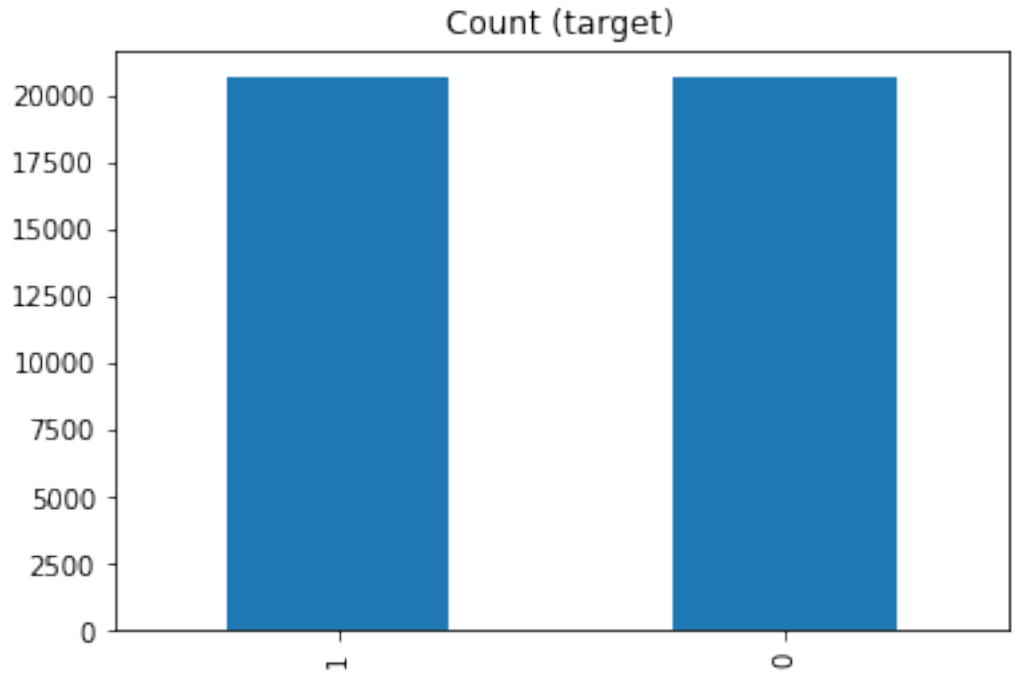
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 590540 entries, 2987000 to 3577539
Columns: 170 entries, isFraud to is_T
dtypes: float64(153), int64(17)
memory usage: 770.4 MB
```

[4]:
```
cnt = train["isFraud"].value_counts().plot(kind='bar', title='Count (target)')
```



[5]:
```
from sklearn.utils import resample

df_majority = train[train["isFraud"] == 0]
df_minority = train[train["isFraud"] == 1]

df_majority_resampled = resample(df_majority,
                                 replace=True,     # to match minority class
                                 n_samples=len(df_minority),
                                 random_state= 27)  # reproducible results
del train; x = gc.collect()
train = pd.concat([df_minority, df_majority_resampled])
del df_minority, df_majority_resampled, df_majority ; x = gc.collect()
```

[6]:
```
cnt = train["isFraud"].value_counts().plot(kind='bar', title='Count (target)')
```



[7]:
```
train.sort_index()
train.head()
```

[7]:

TransactionID_x	isFraud	TransactionAmt	card1	card3	card5	addr2	\
2987203	1	445.000	18268	150.0	226.0	87.000000	
2987240	1	37.098	13413	185.0	137.0	86.286024	

```
2987243                1          37.098  13413  185.0  137.0  86.286024
2987245                1          37.098  13413  185.0  137.0  86.286024
2987288                1         155.521  16578  185.0  226.0  86.286024

                     dist1   C1   C2   C4  …  is_american express  \
TransactionID_x                                …
2987203          174.588854  2.0  2.0  0.0  …                    0
2987240          174.588854  0.0  1.0  1.0  …                    0
2987243          174.588854  1.0  1.0  1.0  …                    0
2987245          174.588854  2.0  1.0  1.0  …                    0
2987288          174.588854  1.0  1.0  1.0  …                    0

                 is_discover  is_mastercard  is_visa  is_charge card  \
TransactionID_x
2987203                    0              0        1               0
2987240                    0              0        1               0
2987243                    0              0        1               0
2987245                    0              0        1               0
2987288                    0              0        1               0

                 is_credit  is_debit  is_debit or credit  is_F  is_T
TransactionID_x
2987203                  1         0                   0     0     1
2987240                  1         0                   0     1     0
2987243                  1         0                   0     1     0
2987245                  1         0                   0     1     0
2987288                  1         0                   0     1     0

[5 rows x 170 columns]
```

[8]:
```
train.to_csv (r'./input/processed/under-sampled.csv', index = True, header=True)
```

RandomForset

April 22, 2020

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import gc
```

[2]:
```
#Load Training Data
train = pd.read_csv("./input/processed/under-sampled.
 ↪csv",index_col='TransactionID_x',)
```

[3]:
```
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
X = train.drop(columns=['isFraud'])
y = train["isFraud"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20,␣
 ↪random_state=101, stratify=y)

scaler = MinMaxScaler(feature_range=(0, 1))

X_train = scaler.fit_transform(X_train)
X_test =  scaler.transform(X_test)

X_train = np.array(X_train)
X_test = np.array(X_test)
del train; x = gc.collect()
```

[4]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score
from sklearn.decomposition import PCA
from sklearn.metrics import (accuracy_score, classification_report)

pca = PCA(n_components=5)
train_pca = pca.fit_transform(X_train)
test_pca = pca.transform(X_test)

rf = RandomForestClassifier(
```

```
    bootstrap = True , min_samples_leaf= 10, n_estimators= 227,
    max_features='sqrt', min_samples_split= 12, max_depth=40, n_jobs=-1)

rf.fit(train_pca, y_train)
pred_y = rf.predict(test_pca)


print(accuracy_score(y_test, pred_y))
print(classification_report(y_test, pred_y))
```

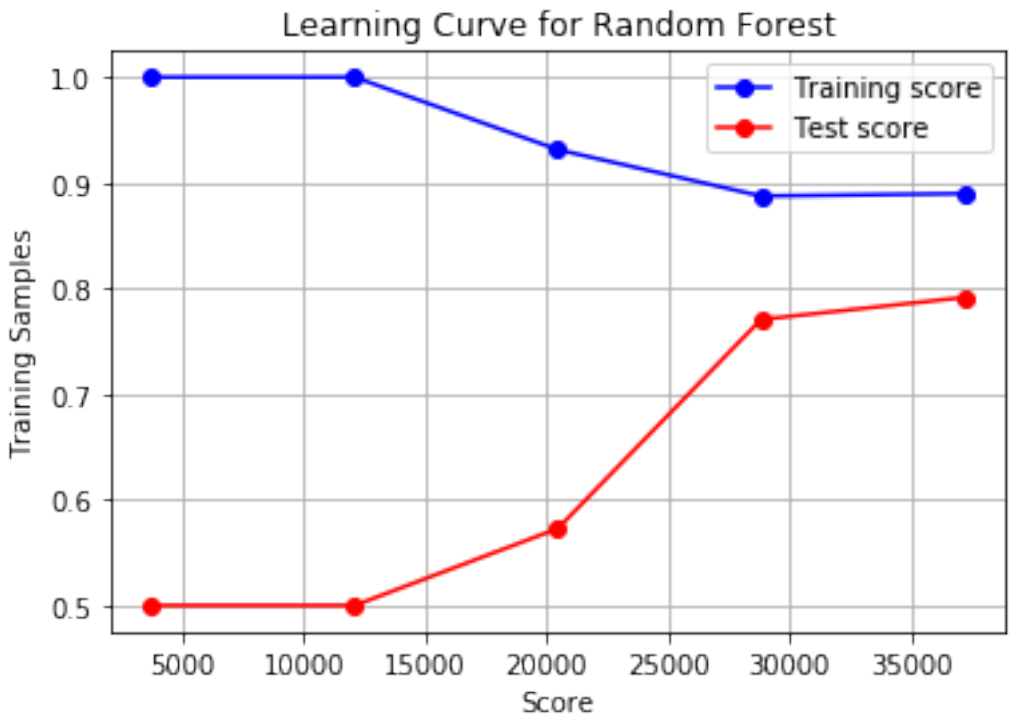
```
0.8718848294217275
              precision    recall  f1-score   support

           0       0.86      0.89      0.87      4133
           1       0.88      0.86      0.87      4133

    accuracy                           0.87      8266
   macro avg       0.87      0.87      0.87      8266
weighted avg       0.87      0.87      0.87      8266
```

[5]:
```
from sklearn.model_selection import (StratifiedKFold)

X = np.array(X)
y = np.array(y)


kfold = StratifiedKFold(n_splits=10, random_state=101, shuffle=True)

pca = PCA(n_components=5)

scaler = MinMaxScaler(feature_range=(0, 1))

rf = RandomForestClassifier(
    bootstrap = True , min_samples_leaf= 10, n_estimators= 227,
    max_features='sqrt', min_samples_split= 12, max_depth=40, n_jobs=-1)

scores = []
for train, test in kfold.split(X, y):

    X_trainCV, X_testCV, y_trainCV, y_testCV = X[train], X[test], y[train],␣
 ↪y[test]

    X_trainCV = scaler.fit_transform(X_trainCV)
    X_testCV =  scaler.transform(X_testCV)

    train_pcaCV = pca.fit_transform(X_trainCV)
```

```
    test_pcaCV = pca.transform(X_testCV)

    rf.fit(train_pcaCV, y_trainCV)
    pred_y = rf.predict(test_pcaCV)

    scores.append(accuracy_score(y_testCV, pred_y))

print(np.mean(scores))
```

```
0.8743892510146065
```

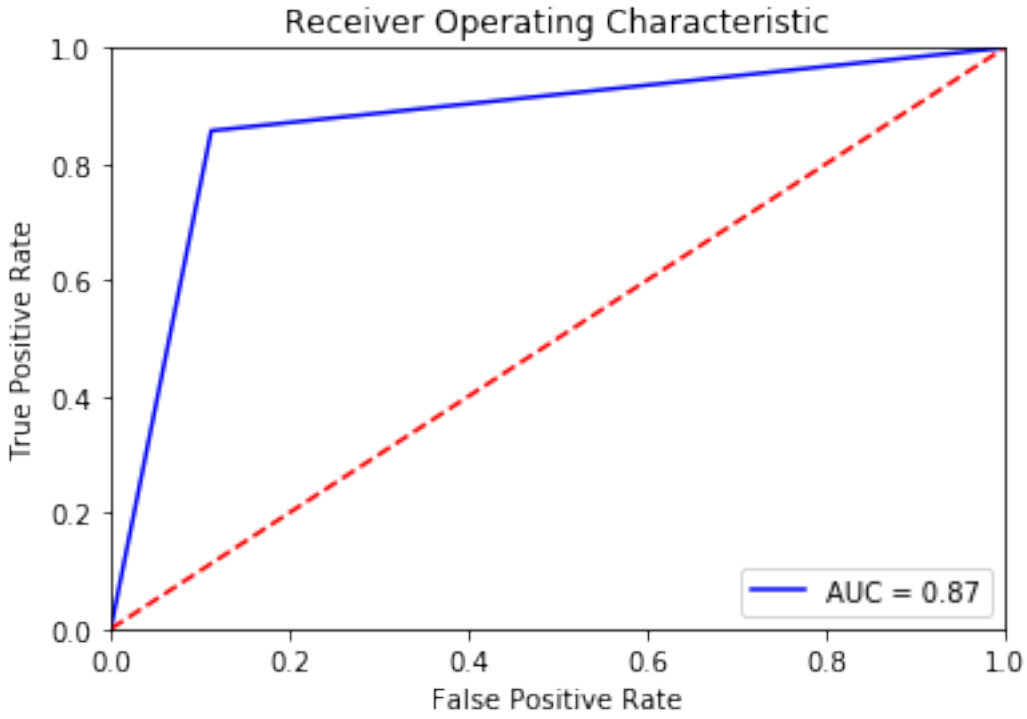
[6]:
```
from sklearn.model_selection import (learning_curve,validation_curve)

pca_product = pca.transform(X)

train_size, train_score, test_score = learning_curve(estimator=rf,␣
 ↪X=pca_product, y=y, cv=10 )

train_score_m = np.mean(train_score, axis=1)
test_score_m = np.mean(test_score, axis=1)
plt.plot(train_size, train_score_m, 'o-', color="b")
plt.plot(train_size, test_score_m, 'o-', color="r")
plt.legend(('Training score', 'Test score'), loc='best')
plt.xlabel("Score")
plt.ylabel("Training Samples")
plt.title("Learning Curve for Random Forest")
plt.grid()
plt.figure(figsize=(16, 5))
plt.show()
```



<Figure size 1152x360 with 0 Axes>

[7]:
```
import sklearn.metrics as metrics
# calculate the fpr and tpr for all thresholds of the classification
rf = RandomForestClassifier(
    bootstrap = True , min_samples_leaf= 10, n_estimators= 227,
    max_features='sqrt', min_samples_split= 12, max_depth=40, n_jobs=-1)

rf.fit(train_pca, y_train)
pred_y = rf.predict(test_pca)

probs = rf.predict_proba(pca_product)
preds = probs[:,1]
fpr, tpr, threshold = metrics.roc_curve(y_test, pred_y)
roc_auc = metrics.auc(fpr, tpr)

plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
```

```
plt.xlabel('False Positive Rate')
plt.show()
```



[8]:
```
# from sklearn.model_selection import RandomizedSearchCV

# # Number of trees in random forest
# n_estimators = [int(x) for x in np.linspace(start = 10, stop = 500, num = 10)]
# # Number of features to consider at every split
# max_features = ['auto', 'sqrt','log2']
# # Maximum number of levels in tree
# max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
# max_depth.append(None)
# # Minimum number of samples required to split a node
# min_samples_split = [12, 50, 100]
# # Minimum number of samples required at each leaf node
# min_samples_leaf = [10, 20, 40]
# # Method of selecting samples for training each tree
# bootstrap = [True, False]
# # Create the random grid
# random_grid = {'n_estimators': n_estimators,
#                'max_features': max_features,
#                'max_depth': max_depth,
#                'min_samples_split': min_samples_split,
```

```
#                'min_samples_leaf': min_samples_leaf,
#                'bootstrap': bootstrap}


# # Use the random grid to search for best hyperparameters
# # Random search of parameters, using 3 fold cross validation,
# # search across 100 different combinations, and use all available cores
# rf_random = RandomizedSearchCV(estimator = rf, param_distributions =␣
 ↪random_grid, n_iter = 400, cv = 3, verbose=2, random_state=42, n_jobs = 2)
# # Fit the random search model
# rf_random.fit(X_train, y_train)

# print(rf_random.best_params_)
# #{'n_estimators': 227, 'min_samples_split': 12, 'min_samples_leaf': 10,␣
 ↪'max_features': 'auto', 'max_depth': 40, 'bootstrap': False}
```

XGBOOST

April 22, 2020

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import gc
```

[2]:
```
#Load Training Data
train = pd.read_csv("./input/processed/under-sampled.
 ↪csv",index_col='TransactionID_x',)
```

[3]:
```
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
X = train.drop(columns=['isFraud'])
y = train["isFraud"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20,␣
 ↪random_state=101, stratify=y)

scaler = MinMaxScaler(feature_range=(0, 1))

X_train = scaler.fit_transform(X_train)
X_test =  scaler.transform(X_test)

X_train = np.array(X_train)
X_test = np.array(X_test)
```

[4]:
```
from sklearn.decomposition import PCA
from xgboost import XGBClassifier
from sklearn.metrics import (accuracy_score, classification_report)

pca = PCA(n_components=5)
X_train_pca = pca.fit_transform(X_train)
X_test_PCA = pca.transform(X_test)

clf_xgb = XGBClassifier(max_depth = 10,
                        objective = 'binary:logistic',
                        n_estimators = 400,
```

```
                            tree_method='gpu_hist',
                            gpu_id=0)

clf_xgb.fit(X_train_pca, y_train)
pred_y = clf_xgb.predict(X_test_PCA)
print(accuracy_score(y_test, pred_y))
print(classification_report(y_test, pred_y))
```

```
0.8836196467457053
              precision    recall  f1-score   support

           0       0.88      0.88      0.88      4133
           1       0.88      0.88      0.88      4133

    accuracy                           0.88      8266
   macro avg       0.88      0.88      0.88      8266
weighted avg       0.88      0.88      0.88      8266
```

[5]:
```
from sklearn.model_selection import (StratifiedKFold)

X = np.array(X)
y = np.array(y)


kfold = StratifiedKFold(n_splits=10, random_state=101, shuffle=True)

pca = PCA(n_components=5)

scaler = MinMaxScaler(feature_range=(0, 1))

clf_xgb = XGBClassifier(max_depth = 10,
                            objective = 'binary:logistic',
                            n_estimators = 400,
                            tree_method='gpu_hist',
                            gpu_id=0)

scores = []
for train, test in kfold.split(X, y):

    X_trainCV, X_testCV, y_trainCV, y_testCV = X[train], X[test], y[train],␣
 ↪y[test]

    X_trainCV = scaler.fit_transform(X_trainCV)
    X_testCV =  scaler.transform(X_testCV)

    train_pcaCV  = pca.fit_transform(X_trainCV)
```

```
    test_pcaCV  = pca.transform(X_testCV)

    clf_xgb.fit(train_pcaCV, y_trainCV)
    pred_y = clf_xgb.predict(test_pcaCV)

    scores.append(accuracy_score(y_testCV, pred_y))

print(np.mean(scores))
```

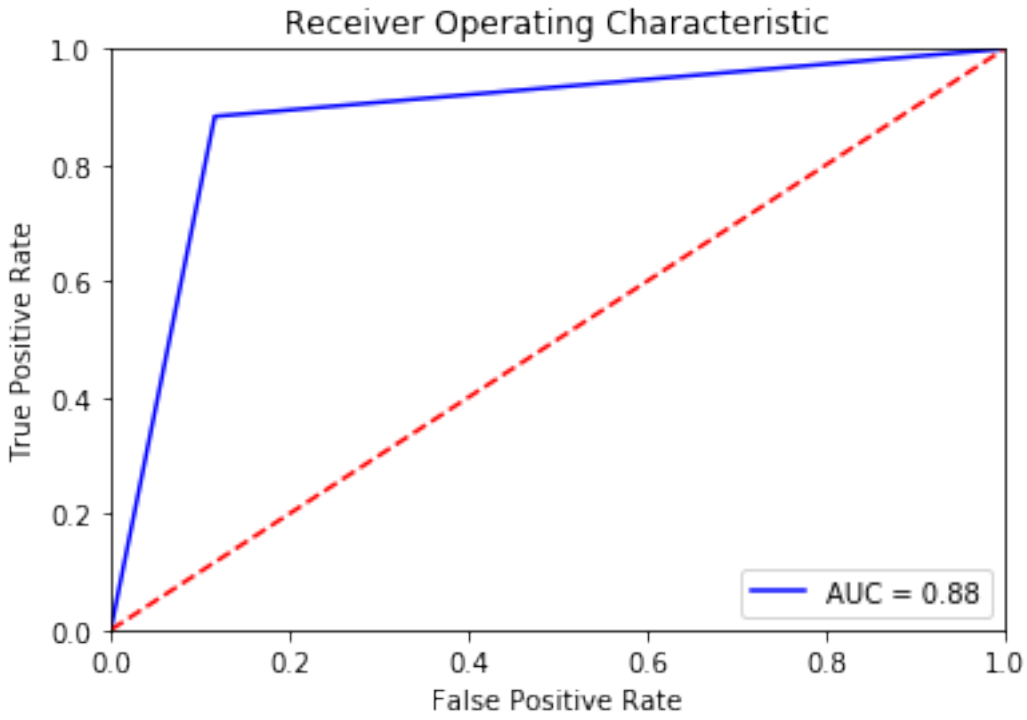
```
0.8838745661264411
```

[6]:
```
from sklearn.model_selection import (learning_curve,validation_curve)

pca_product = pca.transform(X)

train_size, train_score, test_score = learning_curve(estimator=clf_xgb,␣
 ↪X=pca_product, y=y, cv=10 )

train_score_m = np.mean(train_score, axis=1)
test_score_m = np.mean(test_score, axis=1)
plt.plot(train_size, train_score_m, 'o-', color="b")
plt.plot(train_size, test_score_m, 'o-', color="r")
plt.legend(('Training score', 'Test score'), loc='best')
plt.xlabel("Score")
plt.ylabel("Training Samples")
plt.title("Learning Curve for XGBOOST")
plt.grid()
plt.figure(figsize=(16, 5))
plt.show()
```


```
<Figure size 1152x360 with 0 Axes>
```

[7]:
```
import sklearn.metrics as metrics

# calculate the fpr and tpr for all thresholds of the classification
clf_xgb = XGBClassifier(max_depth = 10,
                            objective = 'binary:logistic',
                            n_estimators = 400,
                            tree_method='gpu_hist',
                            gpu_id=0)

clf_xgb.fit(X_train_pca, y_train)
pred_y = clf_xgb.predict(X_test_PCA)



# calculate the fpr and tpr for all thresholds of the classification
probs = clf_xgb.predict_proba(pca_product)
preds = probs[:,1]
fpr, tpr, threshold = metrics.roc_curve(y_test, pred_y)
roc_auc = metrics.auc(fpr, tpr)

plt.title('Receiver Operating Characteristic')
```

```
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



[8]:
```
# from sklearn.model_selection import StratifiedKFold
# from xgboost import XGBClassifier
# from scipy import stats
# from scipy.stats import randint
# from sklearn.model_selection import RandomizedSearchCV
# from sklearn.metrics import␣
 ↪precision_score,recall_score,accuracy_score,f1_score,roc_auc_score
# from sklearn.model_selection import (train_test_split, KFold, StratifiedKFold)


# clf_xgb = XGBClassifier(objective = 'binary:logistic',␣
 ↪tree_method='gpu_hist', gpu_id=0)
# param_dist = {'n_estimators': stats.randint(150, 1000),
#               'learning_rate': stats.uniform(0.01, 0.6),
```

```
#                'subsample': stats.uniform(0.3, 0.9),
#                'max_depth': [3, 4, 5, 6, 7, 8, 9],
#                'colsample_bytree': stats.uniform(0.5, 0.9),
#                'min_child_weight': [1, 2, 3, 4]
#              }

# numFolds = 5
# kfold_5 = KFold(n_splits=2, random_state=101, shuffle=True)
# clf = RandomizedSearchCV(clf_xgb,
#                          param_distributions = param_dist,
#                          cv = kfold_5,
#                          n_iter = 5, # you want 5 here not 25 if I understand␣
 ↪you correctly
#                          scoring = 'roc_auc',
#                          error_score = 0,
#                          verbose = 3,
#                          n_jobs = -1)
```

[ ]:

XGB-Imbalanced

April 22, 2020

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import gc
```

[2]:
```
#Load Training Data
train = pd.read_csv("./input/processed/data.csv",index_col='TransactionID_x',)
```

[3]:
```
from collections import Counter
# count examples in each class
counter = Counter(train["isFraud"])
# estimate scale_pos_weight value
estimate = counter[0] / counter[1]
print('Estimate: %.3f' % estimate)
```

```
Estimate: 27.580
```

[4]:
```
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
X = train.drop(columns=['isFraud'])
y = train["isFraud"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20,␣
↪random_state=101, stratify=y)

scaler = MinMaxScaler(feature_range=(0, 1))

X_train = scaler.fit_transform(X_train)
X_test =  scaler.transform(X_test)

X_train = np.array(X_train)
X_test = np.array(X_test)
```

[5]:
```
from sklearn.decomposition import PCA
from xgboost import XGBClassifier
from sklearn.metrics import (accuracy_score, classification_report)
```

```
pca = PCA(n_components=5)
X_train_pca = pca.fit_transform(X_train)
X_test_PCA = pca.transform(X_test)

clf_xgb = XGBClassifier(max_depth = 10,
                            objective = 'binary:logistic',
                            n_estimators = 400,
                            tree_method='gpu_hist',
                            gpu_id=0,
                            scale_pos_weight=estimate)

clf_xgb.fit(X_train_pca, y_train)
pred_y = clf_xgb.predict(X_test_PCA)
print(accuracy_score(y_test, pred_y))
print(classification_report(y_test, pred_y))
```

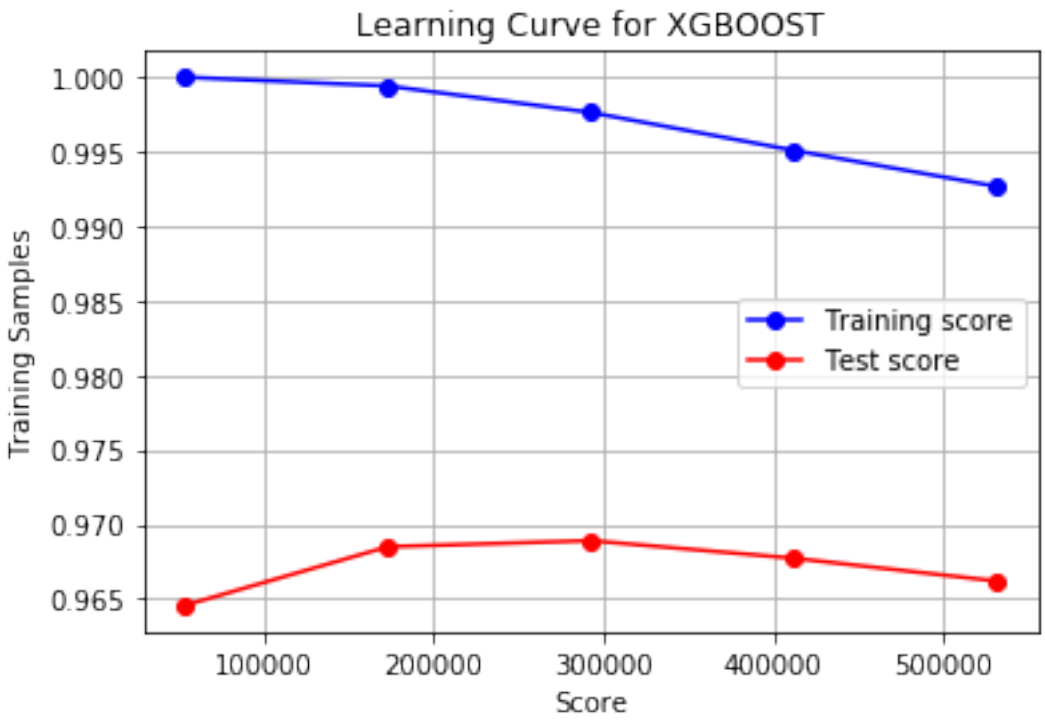
```
0.9729061536898432
              precision    recall  f1-score   support

           0       0.99      0.98      0.99    113975
           1       0.59      0.77      0.66      4133

    accuracy                           0.97    118108
   macro avg       0.79      0.87      0.83    118108
weighted avg       0.98      0.97      0.97    118108
```

[6]:
```
from sklearn.model_selection import (StratifiedKFold)

X = np.array(X)
y = np.array(y)


kfold = StratifiedKFold(n_splits=10, random_state=101, shuffle=True)

pca = PCA(n_components=5)

scaler = MinMaxScaler(feature_range=(0, 1))

clf_xgb = XGBClassifier(max_depth = 10,
                            objective = 'binary:logistic',
                            n_estimators = 400,
                            tree_method='gpu_hist',
                            gpu_id=0,
                            scale_pos_weight=estimate)
```

```
scores = []
for train, test in kfold.split(X, y):

    X_trainCV, X_testCV, y_trainCV, y_testCV = X[train], X[test], y[train],␣
 ↪y[test]

    X_trainCV = scaler.fit_transform(X_trainCV)
    X_testCV =  scaler.transform(X_testCV)

    train_pcaCV  = pca.fit_transform(X_trainCV)
    test_pcaCV  = pca.transform(X_testCV)

    clf_xgb.fit(train_pcaCV, y_trainCV)
    pred_y = clf_xgb.predict(test_pcaCV)

    scores.append(accuracy_score(y_testCV, pred_y))

print(np.mean(scores))
```

```
0.9716666102211537
```

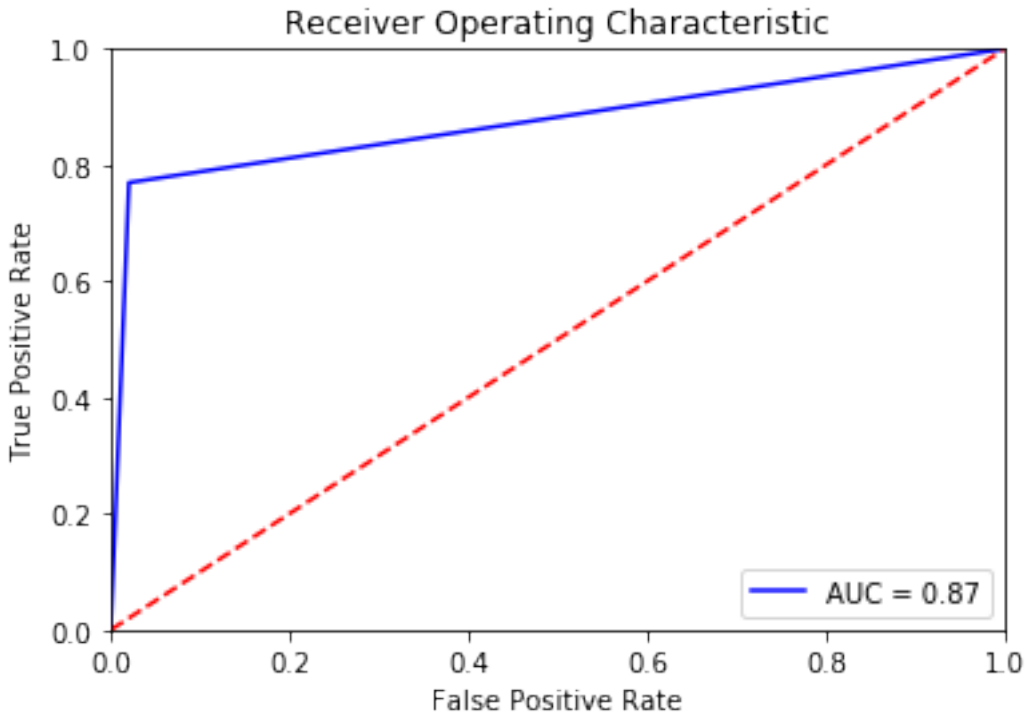
[7]:
```
from sklearn.model_selection import (learning_curve,validation_curve)

pca_product = pca.transform(X)

train_size, train_score, test_score = learning_curve(estimator=clf_xgb,␣
 ↪X=pca_product, y=y, cv=10 )

train_score_m = np.mean(train_score, axis=1)
test_score_m = np.mean(test_score, axis=1)
plt.plot(train_size, train_score_m, 'o-', color="b")
plt.plot(train_size, test_score_m, 'o-', color="r")
plt.legend(('Training score', 'Test score'), loc='best')
plt.xlabel("Score")
plt.ylabel("Training Samples")
plt.title("Learning Curve for XGBOOST")
plt.grid()
plt.figure(figsize=(16, 5))
plt.show()
```



<Figure size 1152x360 with 0 Axes>

[8]:
```
import sklearn.metrics as metrics

# calculate the fpr and tpr for all thresholds of the classification
clf_xgb = XGBClassifier(max_depth = 10,
                            objective = 'binary:logistic',
                            n_estimators = 400,
                            tree_method='gpu_hist',
                            gpu_id=0,
                            scale_pos_weight=estimate)

clf_xgb.fit(X_train_pca, y_train)
pred_y = clf_xgb.predict(X_test_PCA)



# calculate the fpr and tpr for all thresholds of the classification
probs = clf_xgb.predict_proba(pca_product)
preds = probs[:,1]
fpr, tpr, threshold = metrics.roc_curve(y_test, pred_y)
roc_auc = metrics.auc(fpr, tpr)
```

```
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



[ ]:

TestingDataPreprocessing

April 22, 2020

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import gc
```

[2]:
```
#Load Training Data
test = pd.read_csv("./input/raw/test_transaction.csv",)
test_id = pd.read_csv("./input/raw/test_identity.csv",)

train = pd.read_csv("./input/processed/data.csv",)
columns = train.columns
test = test.merge(test_id, how='left', left_index=True, right_index=True)
del test_id, train; x=gc.collect()
```

[3]:
```
columns = [ x for x in columns if "is_" not in x ]
columns.extend(['ProductCD','card4' ,'card6' ,'M1' ,'M2' ,'M3' ,'M4' ,'M5'␣
↪,'M6' ,'M7' ,'M8' ,'M9'])
columns.remove('isFraud')
test = test[columns]
test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506691 entries, 0 to 506690
Columns: 167 entries, TransactionID_x to M9
dtypes: float64(153), int64(2), object(12)
memory usage: 645.6+ MB
```

[4]:
```
dfc = test.select_dtypes('object')
df = test.select_dtypes(exclude=['object'])
del test; x=gc.collect()
```

[5]:
```
df = df.fillna(df.mean())
dfc['TransactionID_x'] = df['TransactionID_x']
df = df.set_index('TransactionID_x')
dfc = dfc.set_index('TransactionID_x')
```

[6]:
```
def predictMissing(model, encoder, X_train, y_train, X_test):
    y_train = encoder.fit_transform(y_train)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    y_pred = encoder.inverse_transform(y_pred)
    y_train = encoder.inverse_transform(y_train)
    return y_pred, y_train
```

[7]:
```
from sklearn import preprocessing
from sklearn.neighbors import KNeighborsClassifier
from xgboost import XGBClassifier


for (columnName, columnData) in dfc.iteritems():
    print('Started Processing', columnName)
    dftmp = df
    dftmp[columnName] = columnData

    tmptrain = dftmp.dropna()
    tmptest = dftmp[dftmp[columnName].isna()]

    target = tmptrain[columnName].copy()
    tmptrain = tmptrain.select_dtypes(exclude=['object'])
    tmptest = tmptest.select_dtypes(exclude=['object'])

    clf=KNeighborsClassifier(n_neighbors=7, n_jobs=-1)
    clf_xgb = XGBClassifier(max_depth = 10,
                            objective = 'multi:softmax',
                            num_class = columnData.nunique(),
                            n_estimators = 400,
                            tree_method='gpu_hist',
                            gpu_id=0)

    y_pred, y_train = predictMissing(clf_xgb, preprocessing.
 ↪LabelEncoder(),tmptrain,target, tmptest)

    tmptrain[columnName] = y_train
    tmptest[columnName] = y_pred

    dftmp = pd.concat([tmptrain, tmptest], axis=0, sort=True)
    df.sort_index(inplace=True)
    dftmp.sort_index(inplace=True)

    df[columnName] = dftmp[columnName]
    print('Finished Processing', columnName)
    del dftmp, tmptrain, tmptest; x=gc.collect()
```

```
Started Processing ProductCD
Finished Processing ProductCD
Started Processing card4
Finished Processing card4
Started Processing card6
Finished Processing card6
Started Processing M1
Finished Processing M1
Started Processing M2
Finished Processing M2
Started Processing M3
Finished Processing M3
Started Processing M4
Finished Processing M4
Started Processing M5
Finished Processing M5
Started Processing M6
Finished Processing M6
Started Processing M7
Finished Processing M7
Started Processing M8
Finished Processing M8
Started Processing M9
Finished Processing M9
```

[8]:
```
dfc = df.select_dtypes('object')
df = df.select_dtypes(exclude=['object'])
dfDummies = pd.get_dummies(dfc, prefix='is')
```

[9]:
```
df = pd.concat([df, dfDummies], axis=1, sort=False)
```

[10]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 506691 entries, 3663549 to 4170239
Columns: 185 entries, TransactionAmt to is_T
dtypes: float64(153), int64(1), uint8(31)
memory usage: 614.2 MB
```

[11]:
```
df.to_csv (r'./input/final_test_data.csv', index = True, header=True)
```

Kaggle

April 22, 2020

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import gc
```

[2]:
```
#Load Training Data
train = pd.read_csv("./input/processed/data.csv",index_col='TransactionID_x',)
test = pd.read_csv("./input/final_test_data.csv",index_col='TransactionID_x',)

print(train.info())
print(test.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 590540 entries, 2987000 to 3577539
Columns: 170 entries, isFraud to is_T
dtypes: float64(153), int64(17)
memory usage: 770.4 MB
None
<class 'pandas.core.frame.DataFrame'>
Int64Index: 506691 entries, 3663549 to 4170239
Columns: 185 entries, TransactionAmt to is_T.7
dtypes: float64(153), int64(32)
memory usage: 719.0 MB
None
```

[3]:
```
items = set(test.columns)
cols = [i for i in train.columns if i in items]
```

[4]:
```
print(cols)
```

```
['TransactionAmt', 'card1', 'card3', 'card5', 'addr2', 'dist1', 'C1', 'C2',
'C4', 'C5', 'C6', 'C7', 'C8', 'C9', 'C10', 'C11', 'C12', 'C13', 'D1', 'D2',
'D3', 'D4', 'D5', 'D10', 'D11', 'D15', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8',
'V9', 'V10', 'V11', 'V12', 'V13', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20',
'V21', 'V22', 'V23', 'V24', 'V26', 'V29', 'V30', 'V31', 'V32', 'V33', 'V34',
'V35', 'V36', 'V37', 'V38', 'V39', 'V40', 'V42', 'V43', 'V44', 'V45', 'V46',
'V47', 'V48', 'V49', 'V50', 'V51', 'V52', 'V53', 'V54', 'V55', 'V56', 'V57',
```

```
'V58', 'V59', 'V60', 'V61', 'V62', 'V63', 'V64', 'V66', 'V67', 'V69', 'V70',
'V71', 'V72', 'V73', 'V74', 'V75', 'V76', 'V77', 'V78', 'V79', 'V80', 'V81',
'V82', 'V83', 'V84', 'V85', 'V86', 'V87', 'V90', 'V91', 'V92', 'V93', 'V94',
'V98', 'V99', 'V100', 'V108', 'V109', 'V110', 'V111', 'V112', 'V113', 'V114',
'V115', 'V116', 'V117', 'V118', 'V119', 'V120', 'V121', 'V122', 'V123', 'V124',
'V125', 'V129', 'V131', 'V281', 'V282', 'V283', 'V284', 'V287', 'V288', 'V289',
'V290', 'V291', 'V292', 'V300', 'V301', 'V302', 'V303', 'V304', 'V309', 'V310',
'V312', 'V313', 'V314', 'V315', 'is_C', 'is_H', 'is_R', 'is_S', 'is_W',
'is_american express', 'is_discover', 'is_mastercard', 'is_visa', 'is_charge
card', 'is_credit', 'is_debit', 'is_F', 'is_T']
```

[5]:
```
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split

X_train = train.drop(columns=['isFraud'])
y_train = train["isFraud"]

X_test = test[cols]
X_train = X_train[cols]
```

[6]:
```
print(X_train.info())
print(X_test.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 590540 entries, 2987000 to 3577539
Columns: 168 entries, TransactionAmt to is_T
dtypes: float64(153), int64(15)
memory usage: 761.4 MB
None
<class 'pandas.core.frame.DataFrame'>
Int64Index: 506691 entries, 3663549 to 4170239
Columns: 168 entries, TransactionAmt to is_T
dtypes: float64(153), int64(15)
memory usage: 653.3 MB
None
```

[7]:
```
scaler = MinMaxScaler(feature_range=(0, 1))

X_train = scaler.fit_transform(X_train)
X_test =  scaler.transform(X_test)

X_train = np.array(X_train)
X_test = np.array(X_test)
y_train = np.array(y_train)
```

[8]:
```
from collections import Counter
# count examples in each class
```

```
counter = Counter(y_train)
# estimate scale_pos_weight value
estimate = counter[0] / counter[1]
print('Estimate: %.3f' % estimate)
```

```
Estimate: 27.580
```

[9]:
```
from sklearn.metrics import (accuracy_score, classification_report)
from xgboost import XGBClassifier

clf_xgb = XGBClassifier(max_depth = 10,
                            objective = 'binary:logistic',
                            n_estimators = 400,
                            tree_method='gpu_hist',
                            gpu_id=0,
                            scale_pos_weight=estimate)
clf_xgb.fit(X_train, y_train)
pred_y = clf_xgb.predict(X_test)
```

[10]:
```
results =  pd.DataFrame()

results['TransactionID'] = list(test.index.values)
results['isFraud'] = pred_y
```

[11]:
```
results['isFraud'].value_counts()
```

[11]:
```
0    505977
1       714
Name: isFraud, dtype: int64
```

[12]:
```
results.to_csv (r'./input/submission.csv', index = False, header=True)
```

[ ]:
```

```