Interpretable Machine Learning Model On HELOC Data

Table of Contents

1. Introduction
2. Preparation
3. Exploratory Data Analysis
4. Feature Engineering
5. Set 1: Without Monotonicity Constraint
6. Set 2: With Monotonicity Constraint
7. Analysis and Conclusion

1. Introduction

Credit scoring is a statistical analysis for banks and other lenders to perform before deciding on extending or denying a person's credit. Due to the increasing number of applications for loans received on a daily basis, it becomes imperative to come up with a model to decide if a person is risky or not.

Different factors can help determine if a model is good or not, namely the accuracy of the model to predict if a person will in fact pay off their credit, and the interpratability, so that we may understand and explain the reasons why someone is considered risky.

This study will explore different models to achieve the most accurate and interpratable model for FICO HELOC (home equity line of credit) data, and explore same models again under monotonic constraints, to come to a conclusion on the best overall model for this problem.

Our HELOC data introduces a dataset containing 23 features influencing the response variable, 'RiskFlag'. The dictionary and explanation for each flag can be found below.

2. Preparation

a. Import Packages
b. Load and preview the dataset
c. Encoding 'RiskFlag'
d. Split the dataset into train and test data set

a. Import Packages

```
#Importing Packages
import pandas as pd
import numpy as np
import sklearn
import warnings
warnings.filterwarnings("ignore")


from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.preprocessing import LabelEncoder
from sklearn import preprocessing


import matplotlib.pyplot as plt
%matplotlib inline

import statsmodels.api as sm
import seaborn as sns
import scorecardpy as sc
```

b. Load and Preview the dataset

```
# Heloc Data
from google.colab import files
uploaded = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving HelocData.csv to HelocData (1).csv
```

```
import io
df = pd.read_csv(io.BytesIO(uploaded['HelocData.csv']))
# Dataset is now stored in a Pandas Dataframe
df
```

	RiskFlag	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13	x14	x
0	Bad	75	169	2	59	21	0	0	100	-7	7	8	22	4	36	
1	Bad	66	502	4	145	34	0	0	97	36	6	6	37	4	27	
2	Good	69	338	2	62	22	0	0	96	12	6	6	23	3	35	

```
# Replacing the missing values -7(Record or No Investigation), -8(Usable/Valid trad
# -9(Condition Not Met (e.g., No Delinquencies, No enquiries)) With NaN

for i in df.columns:
  df[i] = df[i].replace([-7,-8,-9], np.nan)

df.head()
```

	RiskFlag	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13
0	Bad	75.0	169.0	2.0	59.0	21.0	0.0	0.0	100.0	NaN	7.0	8.0	22.0	4.0
1	Bad	66.0	502.0	4.0	145.0	34.0	0.0	0.0	97.0	36.0	6.0	6.0	37.0	4.0
2	Good	69.0	338.0	2.0	62.0	22.0	0.0	0.0	96.0	12.0	6.0	6.0	23.0	3.0
3	Good	75.0	422.0	1.0	91.0	55.0	0.0	0.0	100.0	NaN	7.0	8.0	57.0	4.0
4	Bad	63.0	242.0	2.0	68.0	25.0	0.0	0.0	100.0	NaN	7.0	8.0	26.0	1.0

```
# Heloc Data Dic
uploaded_2 = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

Saving HelocDataDict.xlsx to HelocDataDict.xlsx

```
#Creating dataframe with description of variables
data_dict = pd.read_excel(io.BytesIO(uploaded_2['HelocDataDict.xlsx']))
data_dict
```

	Variable Names	Description	Monotonicity Constraint w.r.t. Prob(Bad = 1)
0	RiskFlag	Paid as negotiated flag (12-36 Months). String...	NaN
1	x1	Consolidated version of risk markers	Monotonically Decreasing
2	x2	Months Since Oldest Trade Open	Monotonically Decreasing
3	x3	Months Since Most Recent Trade Open	Monotonically Decreasing
4	x4	Average Months in File	Monotonically Decreasing
5	x5	Number Satisfactory Trades	Monotonically Decreasing
6	x6	Number Trades 60+ Ever	Monotonically Increasing
7	x7	Number Trades 90+ Ever	Monotonically Increasing
8	x8	Percent Trades Never Delinquent	Monotonically Decreasing
9	x9	Months Since Most Recent Delinquency	Monotonically Decreasing
10	x10	Max Delq/Public Records Last 12	Values 0-7 are monotonically decreasing

```
# Storing feature description in "var_names"
var_names = data_dict['Description'][1:].str.split('(\.| \()', expand=True).iloc[:,
var_names.name = 'Description'
var_names
```

```
1                    Consolidated version of risk markers
2                          Months Since Oldest Trade Open
3                     Months Since Most Recent Trade Open
4                                  Average Months in File
5                              Number Satisfactory Trades
6                                  Number Trades 60+ Ever
7                                  Number Trades 90+ Ever
8                         Percent Trades Never Delinquent
9                    Months Since Most Recent Delinquency
10                 Max Delq/Public Records Last 12 Months
11                                  Max Delinquency Ever
12                                 Number of Total Trades
13                  Number of Trades Open in Last 12 Months
14                             Percent Installment Trades
15                  Months Since Most Recent Inq excl 7days
16                              Number of Inq Last 6 Months
17                 Number of Inq Last 6 Months excl 7days
18                           Net Fraction Revolving Burden
19                         Net Fraction Installment Burden
20                   Number Revolving Trades with Balance
21                 Number Installment Trades with Balance
22    Number Bank/Natl Trades w high utilization ratio
23                            Percent Trades with Balance
Name: Description, dtype: object
```

c. Encoding 'RiskFlag'

```
# In order to enable numerous calculations
# Bad -> 0 and Good -> 1
encode = LabelEncoder()
df['RiskFlag'] = encode.fit_transform(df['RiskFlag'])
print(df)
```

```
       RiskFlag    x1     x2    x3     x4  ...    x19   x20  x21   x22   x23
0             0  75.0  169.0   2.0   59.0  ...  112.0   4.0  6.0   0.0  83.0
1             0  66.0  502.0   4.0  145.0  ...   53.0  17.0  3.0  12.0  83.0
2             1  69.0  338.0   2.0   62.0  ...  100.0   3.0  2.0   1.0  45.0
3             1  75.0  422.0   1.0   91.0  ...   11.0  12.0  2.0   1.0  57.0
4             0  63.0  242.0   2.0   68.0  ...    NaN  12.0  1.0   5.0  87.0
...         ...   ...    ...   ...    ...  ...    ...   ...  ...   ...   ...
10454         1  89.0  425.0  19.0  186.0  ...    NaN   2.0  NaN   0.0  50.0
10455         0  68.0   93.0  16.0   59.0  ...   33.0   1.0  2.0   1.0  50.0
10456         1  87.0  325.0   6.0  102.0  ...   72.0   7.0  5.0   0.0  71.0
10457         1  75.0  413.0   9.0  112.0  ...   49.0   5.0  2.0   1.0  80.0
10458         1  81.0  220.0   3.0   86.0  ...    NaN   1.0  1.0   0.0  14.0

[10459 rows x 24 columns]
```

d. Spliting the data into training and test set

```
np.random.seed(3612202004)
df_train, df_test = train_test_split(df, test_size=0.2, stratify = df['RiskFlag'])
df_train
```

	RiskFlag	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12
2795	1	82.0	178.0	4.0	73.0	18.0	0.0	0.0	95.0	NaN	6.0	6.0	20.0
9142	1	87.0	133.0	11.0	88.0	14.0	0.0	0.0	100.0	NaN	7.0	8.0	22.0
808	0	77.0	229.0	3.0	109.0	23.0	0.0	0.0	100.0	NaN	7.0	8.0	23.0
10432	0	63.0	135.0	2.0	78.0	6.0	4.0	2.0	54.0	23.0	6.0	2.0	13.0
5611	1	81.0	271.0	3.0	85.0	19.0	0.0	0.0	95.0	16.0	6.0	6.0	21.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
9604	1	85.0	243.0	5.0	75.0	22.0	0.0	0.0	100.0	NaN	7.0	8.0	22.0
7360	0	71.0	381.0	3.0	86.0	38.0	0.0	0.0	100.0	NaN	7.0	8.0	39.0
10322	0	84.0	158.0	6.0	83.0	5.0	0.0	0.0	100.0	NaN	7.0	8.0	5.0
1443	1	79.0	256.0	11.0	79.0	42.0	0.0	0.0	100.0	NaN	7.0	8.0	44.0
3522	0	66.0	286.0	1.0	114.0	45.0	0.0	0.0	93.0	68.0	6.0	6.0	46.0

8367 rows × 24 columns

3. Exploratory Data Analysis and Data Cleansing

a. General dataset information
b. Missing values and Imputation (Perhaps better imputation method?)
c. Outlier Analysis
d. Correlation analysis
e. Analysis on individual features

a. General dataset information

```
# Checking dtype of features
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10459 entries, 0 to 10458
Data columns (total 24 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   RiskFlag  10459 non-null  int64  
 1   x1        9861 non-null   float64
 2   x2        9632 non-null   float64
 3   x3        9871 non-null   float64
 4   x4        9871 non-null   float64
 5   x5        9871 non-null   float64
 6   x6        9871 non-null   float64
 7   x7        9871 non-null   float64
 8   x8        9871 non-null   float64
 9   x9        5031 non-null   float64
 10  x10       9871 non-null   float64
 11  x11       9871 non-null   float64
 12  x12       9871 non-null   float64
 13  x13       9871 non-null   float64
 14  x14       9871 non-null   float64
 15  x15       7540 non-null   float64
 16  x16       9871 non-null   float64
 17  x17       9871 non-null   float64
 18  x18       9685 non-null   float64
 19  x19       6452 non-null   float64
 20  x20       9715 non-null   float64
 21  x21       9010 non-null   float64
 22  x22       9288 non-null   float64
 23  x23       9853 non-null   float64
dtypes: float64(23), int64(1)
memory usage: 1.9 MB
```

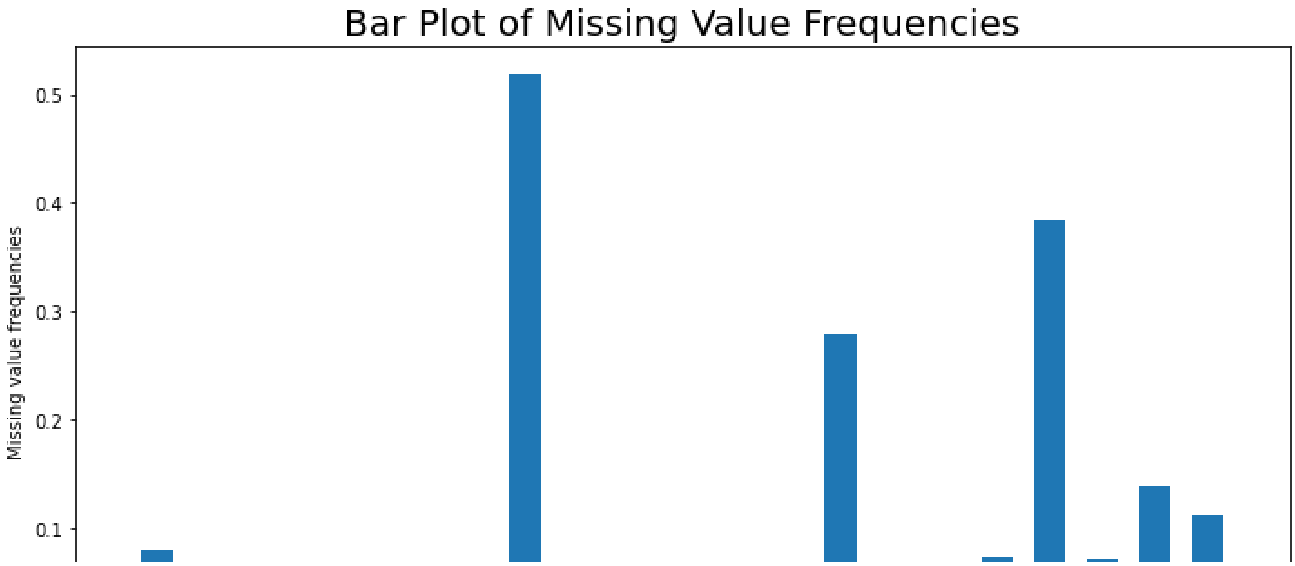
```
#General statistic of the data
data_info = df.describe().T
data_info
```

	count	mean	std	min	25%	50%	75%	max
RiskFlag	10459.0	0.478057	0.499542	0.0	0.0	0.0	1.0	1.0
x1	9861.0	72.060440	9.871795	33.0	64.0	72.0	80.0	94.0
x2	9632.0	200.769103	97.946081	2.0	135.0	186.0	257.0	803.0
x3	9871.0	9.588492	12.963398	0.0	3.0	6.0	12.0	383.0
x4	9871.0	78.778138	34.066063	4.0	57.0	76.0	97.0	383.0
x5	9871.0	21.121467	11.321396	0.0	13.0	20.0	28.0	79.0
x6	9871.0	0.581400	1.238783	0.0	0.0	0.0	1.0	19.0
x7	9871.0	0.384763	0.993223	0.0	0.0	0.0	0.0	19.0
x8	9871.0	92.359943	11.772876	0.0	89.0	97.0	100.0	100.0
x9	5031.0	21.879547	20.808514	0.0	5.0	15.0	34.0	83.0
x10	9871.0	5.757978	1.644518	0.0	5.0	6.0	7.0	9.0
x11	9871.0	6.374531	1.849186	2.0	6.0	6.0	8.0	8.0
x12	9871.0	22.635498	12.999924	0.0	13.0	21.0	30.0	104.0
x13	9871.0	1.863844	1.828099	0.0	0.0	1.0	3.0	19.0
x14	9871.0	34.618681	17.953432	0.0	21.0	33.0	45.0	100.0
x15	7540.0	2.477719	4.760413	0.0	0.0	0.0	3.0	24.0
x16	9871.0	1.455982	2.136161	0.0	0.0	1.0	2.0	66.0
x17	9871.0	1.397123	2.096102	0.0	0.0	1.0	2.0	66.0
x18	9685.0	34.857718	28.896627	0.0	9.0	29.0	56.0	232.0
x19	6452.0	68.537973	24.903776	0.0	53.0	74.0	87.0	471.0

b. Missing values and Imputation

x22	9288.0	1.092270	1.536250	0.0	0.0	1.0	2.0	18.0

```
#plotting the bar plot of missing value frequencies
df_missing_num = df.iloc[:, 1:].isnull().sum()
df_total_num = df.shape[0]
df_missing_freq = df_missing_num / df_total_num
plt.figure(figsize=(12, 6))
df_missing_freq.plot.bar(width=0.60, rot=0, ax=plt.gca())
plt.ylabel('Missing value frequencies')
plt.title('Bar Plot of Missing Value Frequencies', fontsize=20)
plt.show()
```

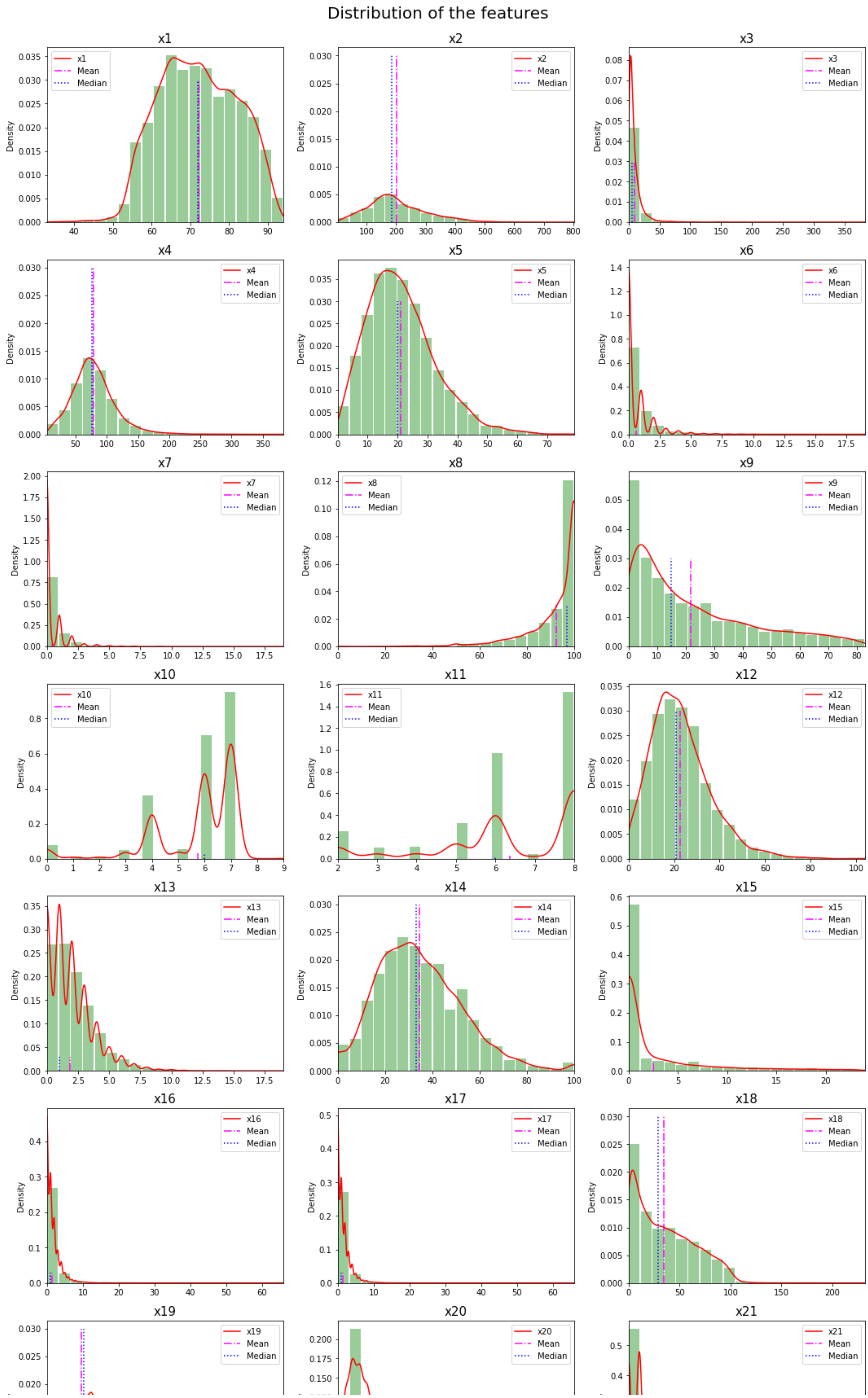


As the value of missing value frequency is large enough, the representativeness of the samples may be reduced. Also, the bias would occur when estimating the parameters in the model. As shown above, X9 (Months Since Most Recent Delinquency) has high frequency of missing values, which is more than a half of samples, **X9 (Months Since Most Recent Delinquency)** is excluded in our model. Also, the values of missing value frequecy of **X15 (Months Since Most Recent Inq excl 7days)** and **x19(Net Fraction Installment Burden)** are considerable. Therefore, in our model, **X9 (Months Since Most Recent Delinquency)**,**X15 (Months Since Most Recent Inq excl 7days)** and **x19(Net Fraction Installment Burden)** are dropped.

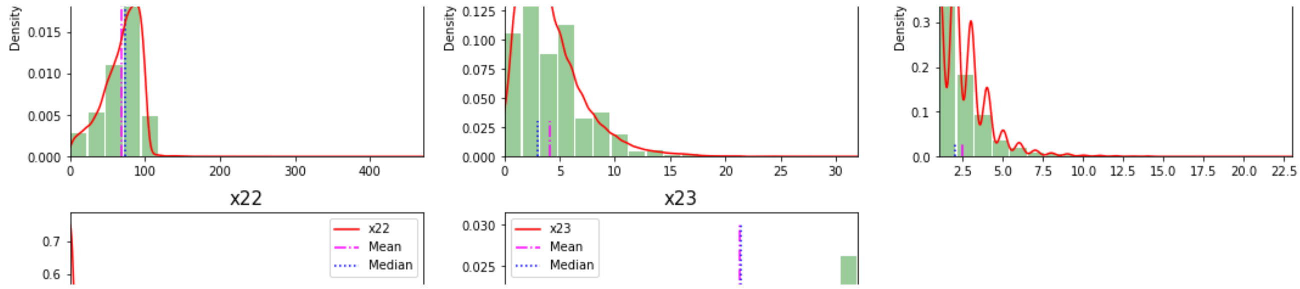
Now, we consider the distribution of each features to impute more reasonable values for missing values in each features.

```
# plotting histogram and the distribution of each features
cols = 3
rows = (df.shape[1]-1)//cols+1

fig = plt.figure(figsize=(30, 5*rows))
for i, var_name in enumerate(df.columns[1:]):
    ax = fig.add_subplot(rows,cols,i+1)
    #histogram
    df[var_name].hist(bins=20, rwidth=0.9, density=True,alpha=0.4,color='green', fi
    #distribution
    df[var_name].plot(kind='density', color='red')

    # drawing mean and median line
    mean_line = df[var_name].mean(skipna=True)
    median_line = df[var_name].median(skipna=True)
    plt.vlines(x = mean_line,ymin=0,ymax=0.03,color='magenta',linestyles='-.',label
    plt.vlines(x = median_line,ymin=0,ymax=0.03,color='blue',linestyles=':',label =

    ax.set_title(var_name, fontsize = 15)
    plt.suptitle('Distribution of the features', fontsize=20)
    plt.tight_layout(rect=[0, 0, 1, 0.97])

    plt.legend()
```

```
    # setting the domain of the plot
    plt.xlim(df.describe().T.loc[var_name]['min'],df.describe().T.loc[var_name]['ma

plt.show()
```

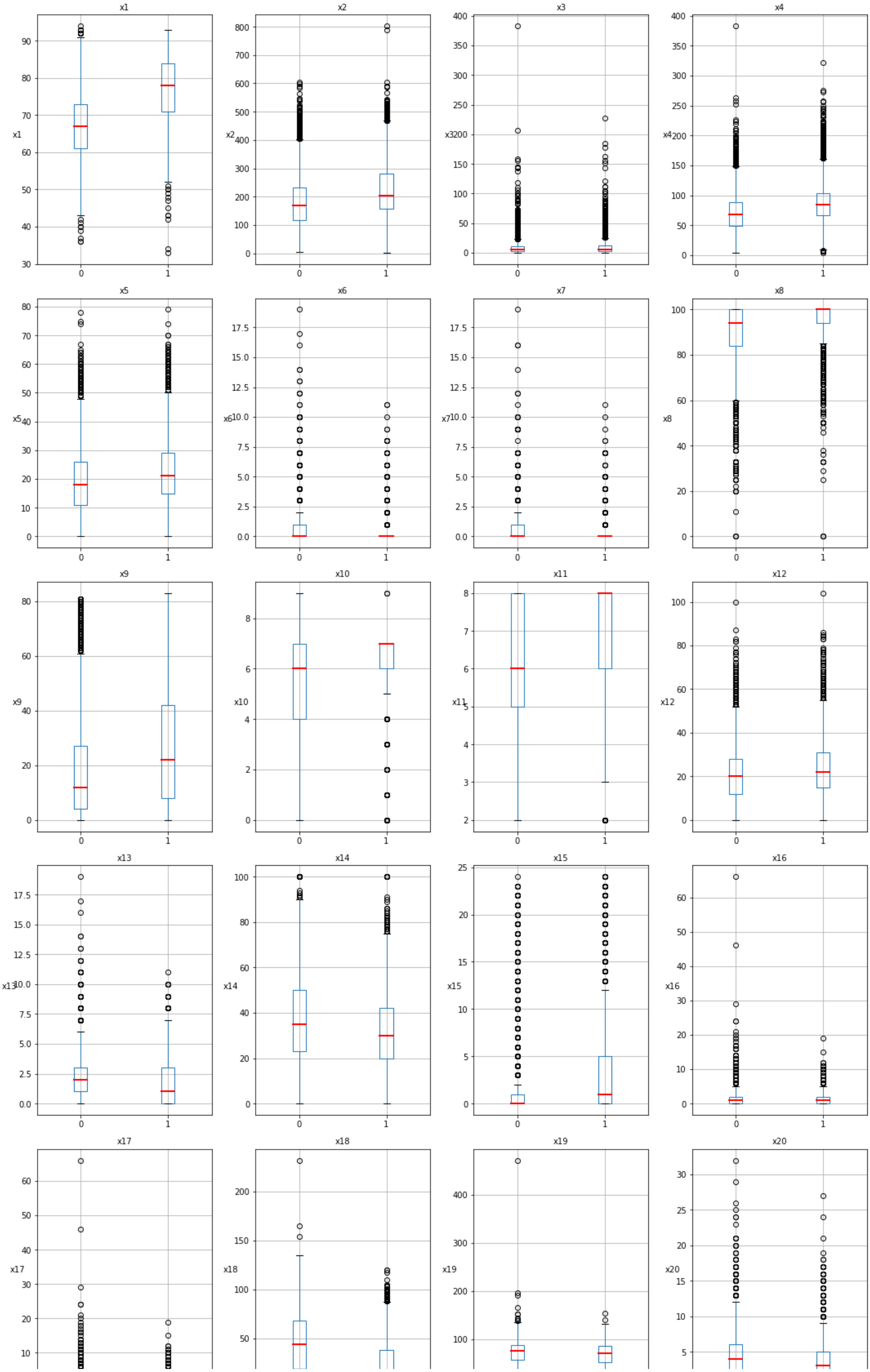


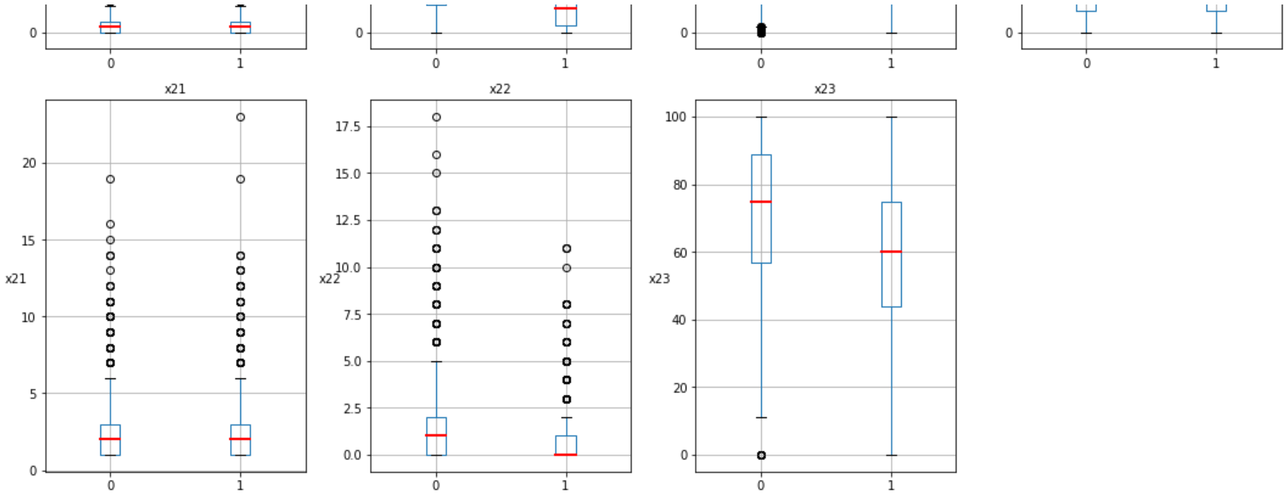
Distribution of the features
x1
x2
x3
x4
x5
x6
x7
x8
x9
x10
x11
x12
x13
x14
x15
x16
x17
x18
x19
x20
x21
Mean
Median
Density




```
#plotting box plot grouped by riskflag
cols = 4
rows = (df.shape[1]-1)//cols + 1

fig = plt.figure(figsize=(15, 5*rows))
for i, var_name in enumerate(df.columns[1:]):
    ax = fig.add_subplot(rows,cols,i+1)
    bp = df.boxplot(column=var_name, by='RiskFlag',
                                ax=ax, return_type='dict')
    [value.set_color('r') for value in bp[0]['medians']]
    [value.set_linewidth(2) for value in bp[0]['medians']]

    ax.set_xlabel('')
    ax.set_ylabel(var_name, rotation=0) # annotate feature names
    ax.set_title(var_name, fontsize=10)

plt.suptitle('Box Plot Grouped by RiskFlag', fontsize=15)
plt.tight_layout(rect=[0, 0, 1, 0.97]) # remove extra space
plt.show()
```

Box Plot Grouped by RiskFlag





As shown above, some distributions are skewed than others (significant difference between the value of median and mean) and some distributions have trends based on their highest density value.

Skewed distribution:

x2 Months Since Oldest Trade Open
x5 Number Satisfactory Trades
x8 Percent Trades Never Delinquent
x13 Number of Trades Open in Last 12 Months
x18 Net Fraction Revolving Burden
x20 Number Revolving Trades with Balance

Therefore, for these 6 features, the missing values are imputed by thier median values to avoid biasedness.

Distribution based on its mode:

x10 Max Delq/Public Records Last 12 Months
x11 Max Delinquency Ever

For these 2 features, the missing values are imputed by their mode values.

```
# impute the missing values for both training sets and test sets
# impute skewed features with median
skewed_dist = ['x2','x5','x8','x13','x18', 'x20']
sk_imp_train = SimpleImputer(missing_values=np.nan,strategy="median")
sk_imp_test = SimpleImputer(missing_values=np.nan, strategy="median")
df_train[skewed_dist]=sk_imp_train.fit_transform(df_train[skewed_dist])
df_test[skewed_dist]=sk_imp_test.fit_transform(df_test[skewed_dist])

# impute the distribution based on its mode with mode
mode_dist = ['x10','x11']
mo_imp_train = SimpleImputer(missing_values=np.nan,strategy="most_frequent")
mo_imp_test = SimpleImputer(missing_values=np.nan, strategy="most_frequent")
df_train[mode_dist]=mo_imp_train.fit_transform(df_train[mode_dist])
df_test[mode_dist]=mo_imp_test.fit_transform(df_test[mode_dist])

# as above features are already imputed, impute the whole remainig missing values b
simple_imp_train = SimpleImputer(missing_values=np.nan,strategy="mean")
simple_imp_test = SimpleImputer(missing_values=np.nan, strategy="mean")
df_train.iloc[:, 1:] = simple_imp_train.fit_transform(df_train.iloc[:, 1:])
df_test.iloc[:, 1:] = simple_imp_test.fit_transform(df_test.iloc[:, 1:])

nans = df_train.iloc[:, 1:].isnull().sum().sum()
print('Number of NaNs remain:', nans)
```

```
Number of NaNs remain: 0
```

c. Outlier analysis

Including the outliers into the model may reduce the predictability of models. Hence, it is important to take few imputation methods for outliers into considerations. There are two methods suggested in convention:

1. Using standard deviation score
2. Using IQR Analysis

However, as shown from above in section c. Imputation, there are too many cases outside the boxes, hence this section will adopt the first method.

The first method considers the data point outside of 3 sd from the mean as outliers.

```
def std_based_outlier(df):
    for i in range(1, len(df.iloc[1])):
        df_outliers = df[np.abs(df.iloc[:,i] - df.iloc[:,i].mean()) > (3*df.iloc[:,
        return df_outliers

df_outlier = std_based_outlier(df_train)
df_outlier
```

	RiskFlag	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12
7708	0	43.0	71.0	11.0	48.0	7.0	1.0	0.0	88.0	1.0	3.0	5.0	8.0
9501	1	33.0	243.0	12.0	88.0	13.0	6.0	1.0	50.0	1.0	3.0	3.0	20.0
5416	0	43.0	165.0	15.0	80.0	20.0	6.0	4.0	61.0	1.0	3.0	3.0	28.0
2695	1	43.0	137.0	2.0	46.0	5.0	5.0	3.0	55.0	3.0	3.0	2.0	11.0
1436	0	36.0	110.0	6.0	35.0	9.0	1.0	1.0	50.0	0.0	4.0	3.0	12.0
8866	1	34.0	157.0	31.0	91.0	1.0	8.0	7.0	33.0	1.0	0.0	2.0	11.0
1394	1	43.0	144.0	6.0	76.0	22.0	1.0	1.0	75.0	1.0	4.0	3.0	28.0
6389	1	42.0	174.0	1.0	66.0	44.0	11.0	8.0	70.0	3.0	0.0	2.0	57.0
7985	0	43.0	255.0	3.0	119.0	29.0	1.0	1.0	63.0	1.0	4.0	6.0	3.0
5559	0	42.0	179.0	1.0	72.0	50.0	9.0	6.0	69.0	0.0	0.0	2.0	62.0
6705	0	41.0	169.0	4.0	85.0	33.0	4.0	0.0	74.0	0.0	3.0	5.0	39.0
6357	0	37.0	263.0	4.0	80.0	26.0	1.0	1.0	78.0	0.0	2.0	4.0	27.0
6459	0	36.0	377.0	1.0	113.0	10.0	7.0	2.0	63.0	0.0	0.0	5.0	17.0
9078	0	40.0	177.0	1.0	64.0	4.0	14.0	10.0	29.0	2.0	0.0	2.0	17.0
9768	0	39.0	80.0	11.0	11.0	0.0	2.0	2.0	0.0	1.0	0.0	2.0	11.0
4020	0	40.0	166.0	3.0	100.0	12.0	11.0	7.0	48.0	1.0	0.0	2.0	23.0

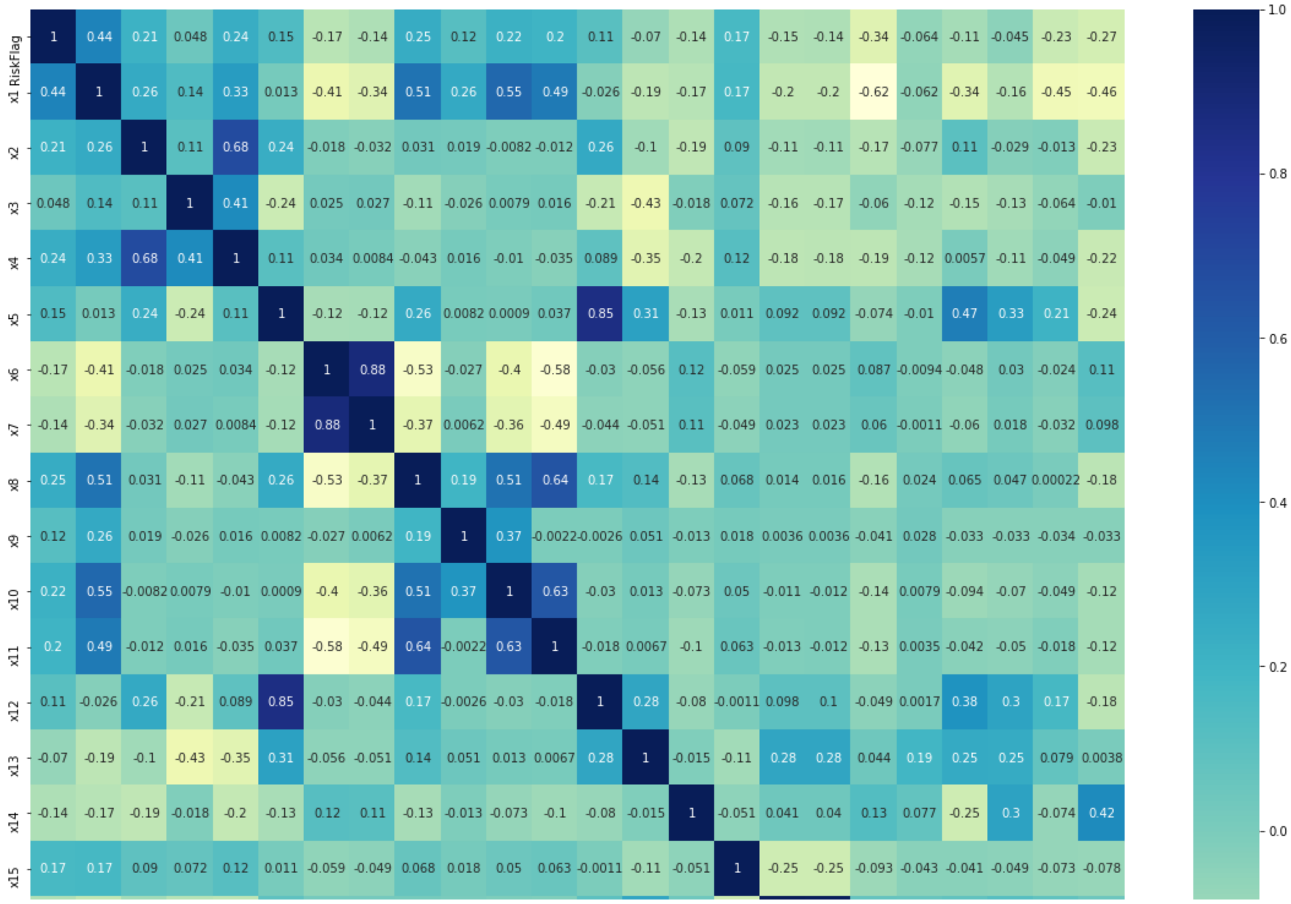
Since the outliers are 16 datapoints from df_train, we may just remove those data points from the df_train.

```
def std_based_outlier_clean(df):
    for i in range(1, len(df.iloc[1])):
        df_outliers = df[~(np.abs(df.iloc[:,i] - df.iloc[:,i].mean()) > (3*df.iloc[
        return df_outliers

df_train = std_based_outlier_clean(df_train)
df_train
```

	RiskFlag	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x
2795	1	82.0	178.0	4.0	73.0	18.0	0.0	0.0	95.0	21.746829	6.0	6.0	2
9142	1	87.0	133.0	11.0	88.0	14.0	0.0	0.0	100.0	21.746829	7.0	8.0	2
808	0	77.0	229.0	3.0	109.0	23.0	0.0	0.0	100.0	21.746829	7.0	8.0	2
10432	0	63.0	135.0	2.0	78.0	6.0	4.0	2.0	54.0	23.000000	6.0	2.0	1

d. Correlation analysis

```
# drawing a heat map
corr_mat = df_train.corr()
plt.figure(figsize=(20,20))
heat_map = sns.heatmap(df_train[corr_mat.index].corr(),annot=True,cmap="YlGnBu")
```



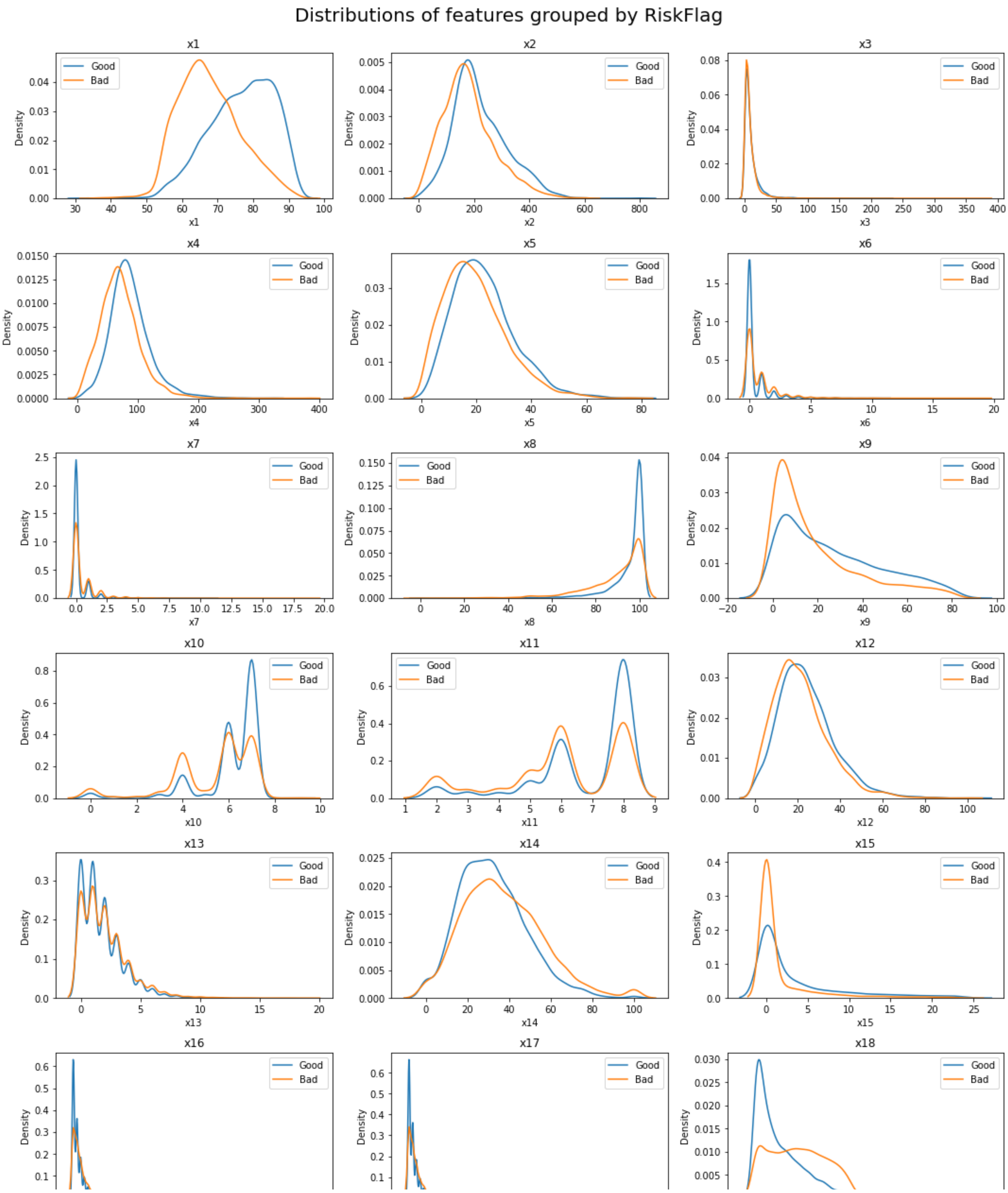
From the heat map, it is shown that

1. **'x6 Number Trades 60+ Ever'** and **'x7 Number Trades 90+ Ever'** are highly correlated
2. **'x16 Number of Inq Last 6 Months'** and **'x17 Number of Inq Last 6 Months excl 7days'** are highly correlated
3. **'x1 Consolidated version of risk markers'** is highly correlated with other features among all features while other features are not significantly correlated.
4. As **'x2 Months Since Oldest Trade Open'**,**'x4 Average Months in File'**,**'x8 Percent Trades Never Delinquent'**,**'x10 Max Delq/Public Records Last 12 Months'**,**'x11 Max Delinquency Ever'** and **'x15 Months Since Most Recent Inq excl 7days'** are relatively correlated to **'x1 Consolidated version of risk markers'**, we can say that **'x1 Consolidated version of risk markers'** can be explained in terms of these features.

e. Analysis on individual features

```
plt.figure(figsize=(15,30))
for i, var_name in enumerate(df.columns[1:]):
    plt.subplot(10, 3, i + 1)
    sns.kdeplot(df.loc[df['RiskFlag'] == 1, var_name], label = 'Good')
    sns.kdeplot(df.loc[df['RiskFlag'] == 0, var_name], label = 'Bad')

    plt.legend()
    plt.ylabel('Density')
    plt.title(var_name)
```

```
plt.suptitle('Distributions of features grouped by RiskFlag', fontsize=20)
plt.tight_layout(rect=[0, 0, 1, 0.97])
```



Now, we can intuitively explain each features grouped by RiskFlag from above distribution table.

1. For **'x6 Number Trades 60+ Ever'** and **'x7 Number Trades 90+ Ever'**, people with bad risk had lower number of trades than people with good risk.
2. For **'x8 Percent Trades Never Delinquent'**, people with good risk have significantly higher percent of trades that is never delinquent while for **'x9 Months Since Most Recent Delinquency'**, the people with bad risk were dominant.
3. For **'x10 Max Delq/Public Records Last 12 Months'** and **'x11 Max Delinquency Ever'**, people with good risk had less delinquent trades, which were mostly 'unknown','current and never delinqeunt', etc.

4. For **'x15 Months Since Most Recent Inq excl 7days'**,**'x16 Number of Inq Last 6 Months'** and **'x17 Number of Inq Last 6 Months excl 7days'** , the people with good risk had significantly lower number of Inq.
5. For **'x18 Net Fraction Revolving Burden'**, **'x22 Number Bank/Natl Trades w high utilization ratio'** and **'x23 Percent Trades with Balance'**, the people with good risk had lower fraction of revolving burden, lower number of trades with high utilization ratio and lower percent trades with balace.

4. Feature Engineering

a. Feature Selection
b. One-hot Encoding for x10 and x11
c. Feature Scaling

Feature engineering is important in order to fit well into the model and to prevent overfitting or underfitting. Furthermore, in order to increase the interpretability, selecting reasonable number of features is crucial.

Since different models may require different feature engineering techniques, this section will determine the features that are absolutely not going to be used through out the analysis.

Since the units of features are majorily different, the scaling process is compulsory in order to increase the model fitting.
In the scaling process the standard scaling method is adopted.

a. Feature Selection

Through feature selection process, we can reduce the running time and reduce the varaiances of models in order to robustly learn data. Plus, better interpretability of the models.

The feature selection metrics are followings:

1. *Missing Values*

   -> Based on the result from section 3b, **X9 (Months Since Most Recent Delinquency)**, **X15 (Months Since Most Recent Inq excl 7days)**, and **x19(Net Fraction Installment Burden)** are excluded in our model.

2. *Multicolinearity*

   -> Since **x6 (Number Trades 60+ Ever)** and **x7 (Number Trades 90+ Ever)** are highly correlated (0.89), and **x16 (Number of Inq Last 6 Months)** and **x17 (Number of Inq Last 6 Months excl 7days)** are extremely highly correlated (0.99), we can drop one feature from each pair.

```
#x_train,x_test,y_train and y_test before the feature selection
X_train=df_train.iloc[:,1:]
X_test=df_test.iloc[:,1:]
y_train=df_train.iloc[:,0]
y_test=df_test.iloc[:,0]
X_train
```

	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13	x
2795	82.0	178.0	4.0	73.0	18.0	0.0	0.0	95.0	21.746829	6.0	6.0	20.0	3.0	4
9142	87.0	133.0	11.0	88.0	14.0	0.0	0.0	100.0	21.746829	7.0	8.0	22.0	1.0	1
808	77.0	229.0	3.0	109.0	23.0	0.0	0.0	100.0	21.746829	7.0	8.0	23.0	2.0	3
10432	63.0	135.0	2.0	78.0	6.0	4.0	2.0	54.0	23.000000	6.0	2.0	13.0	2.0	3
5611	81.0	271.0	3.0	85.0	19.0	0.0	0.0	95.0	16.000000	6.0	6.0	21.0	3.0	3
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
9604	85.0	243.0	5.0	75.0	22.0	0.0	0.0	100.0	21.746829	7.0	8.0	22.0	2.0	2
7360	71.0	381.0	3.0	86.0	38.0	0.0	0.0	100.0	21.746829	7.0	8.0	39.0	3.0	1
10322	84.0	158.0	6.0	83.0	5.0	0.0	0.0	100.0	21.746829	7.0	8.0	5.0	2.0	2
1443	79.0	256.0	11.0	79.0	42.0	0.0	0.0	100.0	21.746829	7.0	8.0	44.0	1.0	2
3522	66.0	286.0	1.0	114.0	45.0	0.0	0.0	93.0	68.000000	6.0	6.0	46.0	1.0	2

8351 rows × 23 columns

```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
best = 10
X_norm = preprocessing.MinMaxScaler().fit_transform(X_train)
chi_selector = SelectKBest(chi2, k=best)
chi_selector.fit(X_norm, y_train)
chi_support = chi_selector.get_support()
chi_feature = X_train.loc[:,chi_support].columns.tolist()
chi_feature
```

```
['x1', 'x2', 'x6', 'x7', 'x10', 'x11', 'x15', 'x18', 'x22', 'x23']
```

The ranking of features under chi2 test are:
'x1', 'x18', '15', 'x22', 'x23', 'x11', 'x6', 'x4', 'x2', 'x10', 'x7', 'x9', 'x14' ...

```
#X_train and X_test with selected features
selected_features=['x1','x2','x3','x4','x5','x6','x8','x10','x11','x12','x13','x14'
X_train_final=X_train[selected_features]
X_test_final=X_test[selected_features]

X_train_final
```

	x1	x2	x3	x4	x5	x6	x8	x10	x11	x12	x13	x14	x16	x18
2795	82.0	178.0	4.0	73.0	18.0	0.0	95.0	6.0	6.0	20.0	3.0	40.0	4.0	4.0
9142	87.0	133.0	11.0	88.0	14.0	0.0	100.0	7.0	8.0	22.0	1.0	14.0	0.0	3.0
808	77.0	229.0	3.0	109.0	23.0	0.0	100.0	7.0	8.0	23.0	2.0	35.0	0.0	38.0
10432	63.0	135.0	2.0	78.0	6.0	4.0	54.0	6.0	2.0	13.0	2.0	31.0	2.0	31.0
5611	81.0	271.0	3.0	85.0	19.0	0.0	95.0	6.0	6.0	21.0	3.0	33.0	1.0	6.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
9604	85.0	243.0	5.0	75.0	22.0	0.0	100.0	7.0	8.0	22.0	2.0	23.0	4.0	4.0
7360	71.0	381.0	3.0	86.0	38.0	0.0	100.0	7.0	8.0	39.0	3.0	18.0	3.0	35.0
10322	84.0	158.0	6.0	83.0	5.0	0.0	100.0	7.0	8.0	5.0	2.0	20.0	0.0	41.0
1443	79.0	256.0	11.0	79.0	42.0	0.0	100.0	7.0	8.0	44.0	1.0	23.0	0.0	17.0
3522	66.0	286.0	1.0	114.0	45.0	0.0	93.0	6.0	6.0	46.0	1.0	20.0	0.0	67.0

8351 rows × 18 columns

b. One hot encoding for x10 and x11

```
X_train = pd.get_dummies(X_train_final, columns=['x10','x11'], drop_first=True)
X_test = pd.get_dummies(X_test_final, columns=['x10','x11'], drop_first=True)
X_train
```

	x1	x2	x3	x4	x5	x6	x8	x12	x13	x14	x16	x18	x20	x21
2795	82.0	178.0	4.0	73.0	18.0	0.0	95.0	20.0	3.0	40.0	4.0	4.0	3.0	2.0
9142	87.0	133.0	11.0	88.0	14.0	0.0	100.0	22.0	1.0	14.0	0.0	3.0	1.0	1.0
808	77.0	229.0	3.0	109.0	23.0	0.0	100.0	23.0	2.0	35.0	0.0	38.0	4.0	3.0
10432	63.0	135.0	2.0	78.0	6.0	4.0	54.0	13.0	2.0	31.0	2.0	31.0	1.0	1.0
5611	81.0	271.0	3.0	85.0	19.0	0.0	95.0	21.0	3.0	33.0	1.0	6.0	3.0	1.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
9604	85.0	243.0	5.0	75.0	22.0	0.0	100.0	22.0	2.0	23.0	4.0	4.0	3.0	1.0
7360	71.0	381.0	3.0	86.0	38.0	0.0	100.0	39.0	3.0	18.0	3.0	35.0	8.0	3.0
10322	84.0	158.0	6.0	83.0	5.0	0.0	100.0	5.0	2.0	20.0	0.0	41.0	1.0	1.0
1443	79.0	256.0	11.0	79.0	42.0	0.0	100.0	44.0	1.0	23.0	0.0	17.0	8.0	3.0
3522	66.0	286.0	1.0	114.0	45.0	0.0	93.0	46.0	1.0	20.0	0.0	67.0	12.0	4.0

8351 rows × 30 columns

c. Feature Scaling

```
std_scaler = preprocessing.StandardScaler().fit(X_train)
X_train_scd = std_scaler.transform(X_train)
X_train_scd     # X_train with standardly scaled
X_test_scd = std_scaler.transform(X_test)
X_test_scd     # X_test with standardly scaled
```

```
array([[-0.54236964, -1.52568484, -0.66585104, ...,  1.6254088 ,
        -0.11338552, -0.98160844],
       [ 0.93416322,  0.75975273,  0.03156782, ..., -0.61522984,
        -0.11338552,  1.01873614],
       [ 0.40683005,  0.11897585, -0.66585104, ..., -0.61522984,
        -0.11338552,  1.01873614],
       ...,
       [ 0.30136342, -0.61791758, -0.27839612, ..., -0.61522984,
        -0.11338552,  1.01873614],
       [-0.96423617, -0.54316027, -0.20090513, ..., -0.61522984,
        -0.11338552, -0.98160844],
       [-1.0697028 ,  1.14421887,  0.72898667, ...,  1.6254088 ,
        -0.11338552, -0.98160844]])
```

5. Set 1: Without Monotonicity Constraints

a. Logistic Regression
b. General Additive Model
c. Decision Tree
d. Support Vector Machine
e. Gradient Boosting: XGBoost
f. Neural Network

From regression model: We have chosen logistic regression (assumed high interpretability) and GAM (assumed high performance).

From tree-based methods: We have chosen decision tree (assumed high interpretability) and XGBoost (assumed high performance).

a. Logistic Regression

We first check the fit to the 10 features

```
#Fitting the logistic model
from sklearn.linear_model import LogisticRegression

logreg = LogisticRegression(C=1e8, solver='newton-cg')
logreg.fit(X_train_scd, y_train)

#Coefficients of the model
print("Coefficients :", np.round(logreg.intercept_,4), np.round(logreg.coef_,4))
```

```
Coefficients : [-0.1295] [[ 0.5115  0.0845 -0.0628  0.2448  0.3766 -0.0809  0.
  -0.1533 -0.2616 -0.243  -0.1289  0.0061 -0.1779  0.0696  0.0291 -0.008
   0.0325 -0.0422 -0.0705  0.1042  0.1209 -0.0293  0.0282  0.0115 -0.0153
  -0.0537  0.0104  0.011 ]]
```

```
from sklearn.metrics import accuracy_score, confusion_matrix

# accuracy score
performance={}

y_pred_train = logreg.predict(X_train_scd)
y_pred_test = logreg.predict(X_test_scd)

accuracy_train = accuracy_score(y_train, y_pred_train)
accuracy_test = accuracy_score(y_test, y_pred_test)
performance['Logistic Regression Raw : ']=np.round(accuracy_test,4)
print('Accuracy on the training set =', np.round(accuracy_train,4))
print('Accuracy on the test set =', np.round(accuracy_test,4))
```

```
Accuracy on the training set = 0.717
Accuracy on the test set = 0.7051
```

```
#Summary of the statistic
X1 = sm.add_constant(X_train)
logreg = sm.Logit(y_train,X1).fit()
print(logreg.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.555938
         Iterations 6
                           Logit Regression Results
==============================================================================
Dep. Variable:               RiskFlag   No. Observations:                 8351
Model:                          Logit   Df Residuals:                     832C
Method:                           MLE   Df Model:                           3C
Date:                Mon, 30 Nov 2020   Pseudo R-squ.:                  0.1969
Time:                        15:35:42   Log-Likelihood:                -4642.6
converged:                       True   LL-Null:                       -5780.7
Covariance Type:            nonrobust   LLR p-value:                     0.00C
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const         -5.6531      0.532    -10.629      0.000      -6.696      -4.611
x1             0.0539      0.006      8.675      0.000       0.042       0.066
x2             0.0009      0.000      2.221      0.026       0.000       0.002
x3            -0.0049      0.002     -1.966      0.049      -0.010   -1.47e-05
x4             0.0074      0.001      5.451      0.000       0.005       0.01C
x5             0.0344      0.005      6.431      0.000       0.024       0.045
x6            -0.0682      0.035     -1.924      0.054      -0.138       0.001
x8             0.0106      0.004      2.536      0.011       0.002       0.019
x12            0.0007      0.004      0.178      0.859      -0.007       0.009
x13           -0.0118      0.018     -0.654      0.513      -0.047       0.024
x14           -0.0088      0.002     -4.483      0.000      -0.013      -0.005
x16           -0.1293      0.016     -7.985      0.000      -0.161      -0.098
x18           -0.0088      0.002     -5.506      0.000      -0.012      -0.006
x20           -0.0447      0.015     -2.996      0.003      -0.074      -0.015
```

```
x21           0.0040      0.020      0.202      0.840     -0.035      0.043
x22          -0.1240      0.029     -4.325      0.000     -0.180     -0.068
x23           0.0033      0.002      1.698      0.089     -0.001      0.007
x10_1.0       0.4062      0.398      1.021      0.307     -0.373      1.186
x10_2.0      -0.1107      0.425     -0.260      0.794     -0.944      0.722
x10_3.0       0.2317      0.255      0.910      0.363     -0.267      0.731
x10_4.0      -0.1170      0.184     -0.636      0.525     -0.477      0.243
x10_5.0      -0.4728      0.258     -1.833      0.067     -0.978      0.033
x10_6.0       0.2274      0.176      1.292      0.197     -0.118      0.572
x10_7.0       0.2425      0.226      1.074      0.283     -0.200      0.685
x10_9.0      -1.1972      1.329     -0.901      0.368     -3.802      1.407
x11_3.0       0.1691      0.191      0.884      0.377     -0.206      0.544
x11_4.0       0.0662      0.190      0.347      0.728     -0.307      0.439
x11_5.0      -0.0521      0.141     -0.369      0.712     -0.329      0.225
x11_6.0      -0.1204      0.135     -0.890      0.373     -0.386      0.145
x11_7.0       0.0928      0.281      0.331      0.741     -0.457      0.643
x11_8.0       0.0221      0.191      0.116      0.908     -0.352      0.396
==============================================================================
```

P|z| of x11_4.0 is very large, which indicates high probability of not being statistically relevant, we can therefore drop it.

```
#Refining the model
def df_preprocesser2(df, x18_breaks=None, x22_breaks=None, set='train'):
    df_tmp = df.copy()
    df_tmp.drop(df_tmp.columns.difference(['RiskFlag','x1','x2','x3','x4','x5','x6'

    # use IV binning
    if set=='train':
      x22_bins = sc.woebin(df_tmp, y='RiskFlag',x='x22', method='tree')
      x22_breaks = np.insert(x22_bins['x22']['breaks'].values.astype(np.float), 0,
    df_tmp['x22'] = pd.cut(df_tmp['x22'], bins=x22_breaks, right=True)
    df_tmp

    if set=='train':
      x18_bins = sc.woebin(df_tmp, y='RiskFlag',x='x18', method='tree')
      x18_breaks = np.insert(x18_bins['x18']['breaks'].values.astype(np.float), 0,
    df_tmp['x18'] = pd.cut(df_tmp['x18'], bins=x18_breaks, right=True)
    df_tmp

   # one-hot encoding
    df_tmp = pd.get_dummies(df_tmp, columns=['x10','x11','x22','x18'], drop_first=T

    return df_tmp, x18_breaks, x22_breaks


df_train_log = df_train.copy()
df_test_log = df_test.copy()

df_train_log, x18_breaks, x22_breaks = df_preprocesser2(df_train, set='train')
df_test_log= df_preprocesser2(df_test_log, x18_breaks, x22_breaks, set='test')[0]

#This bin is dropped due to high chance of not being
df_train_log = df_train_log.drop(columns=['x11_4.0','x11_6.0'])
df_test_log = df_test_log.drop(columns=['x11_4.0','x11_6.0'])
```

```
#x_train,x_test,y_train and y_test before the feature selection
X_train_log=df_train_log.iloc[:,1:]
X_test_log=df_test_log.iloc[:,1:]
y_train_log=df_train_log.iloc[:,0]
y_test_log=df_test_log.iloc[:,0]

X_train_log.head()
```

```
[INFO] creating woe binning ...
[INFO] creating woe binning ...
```

	x1	x2	x3	x4	x5	x6	x8	x12	x13	x14	x16	x20	x21	x23
2795	82.0	178.0	4.0	73.0	18.0	0.0	95.0	20.0	3.0	40.0	4.0	3.0	2.0	63.0
9142	87.0	133.0	11.0	88.0	14.0	0.0	100.0	22.0	1.0	14.0	0.0	1.0	1.0	67.0
808	77.0	229.0	3.0	109.0	23.0	0.0	100.0	23.0	2.0	35.0	0.0	4.0	3.0	58.0
10432	63.0	135.0	2.0	78.0	6.0	4.0	54.0	13.0	2.0	31.0	2.0	1.0	1.0	67.0
5611	81.0	271.0	3.0	85.0	19.0	0.0	95.0	21.0	3.0	33.0	1.0	3.0	1.0	36.0

```
logreg = LogisticRegression(C=1e8, solver='newton-cg')
logreg.fit(X_train_log, y_train_log)

print("Coefficients :", np.round(logreg.intercept_,4), np.round(logreg.coef_,4))
```

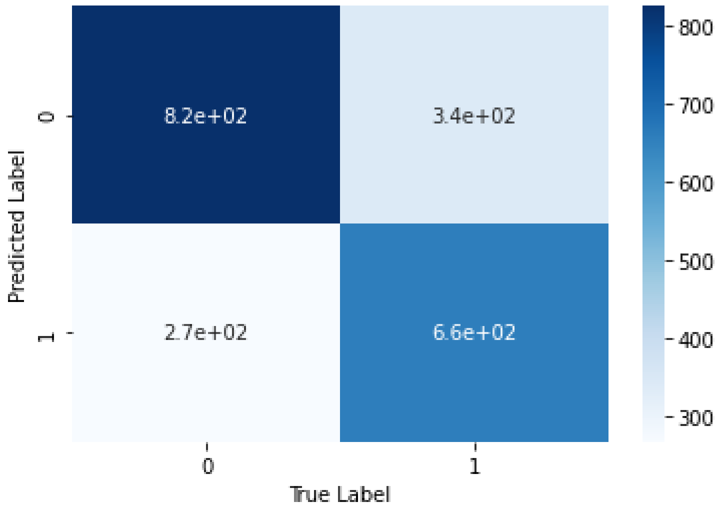
```
Coefficients : [-5.6049] [[ 5.2900e-02  9.0000e-04 -5.0000e-03  7.6000e-03  3.
   9.7000e-03  1.4000e-03 -1.4400e-02 -8.4000e-03 -1.2920e-01 -5.7700e-02
   5.0000e-03  3.3000e-03  3.4970e-01  3.9000e-03  2.1650e-01 -1.3650e-01
  -4.6570e-01  2.2420e-01  3.0510e-01 -1.0237e+00  1.9510e-01  1.1500e-02
   1.9530e-01  1.0670e-01 -2.3890e-01 -4.6490e-01 -2.4760e-01 -3.5950e-01
  -7.4680e-01]]
```

```
# accuracy score
y_pred_train_log = logreg.predict(X_train_log)
y_pred_test_log = logreg.predict(X_test_log)

accuracy_train = accuracy_score(y_train_log, y_pred_train_log)
accuracy_test = accuracy_score(y_test_log, y_pred_test_log)

performance['Logistic Regression with IV binning : ']=np.round(accuracy_test,4)
print('Accuracy on the training set =', np.round(accuracy_train,4))
print('Accuracy on the test set =', np.round(accuracy_test,4))
```

```
Accuracy on the training set = 0.7224
Accuracy on the test set = 0.7089
```

We see that the accuracy for the test data is slightly higher than the accuracy for the training data, which shows that no regularisation is needed due to low chance of overfitting.

```
#Confusion matrix
import seaborn as sn
cf_mat = confusion_matrix(y_test_log, y_pred_test_log).T
sn.heatmap(cf_mat,annot=True,cmap='Blues')
plt.xlabel('True Label')
plt.ylabel('Predicted Label')
plt.show()
```




```
X1 = sm.add_constant(X_train_log)
logreg = sm.Logit(y_train,X1).fit()
print(logreg.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.556266
         Iterations 6
                           Logit Regression Results
==============================================================================
Dep. Variable:               RiskFlag   No. Observations:                 8351
Model:                          Logit   Df Residuals:                     831
Method:                           MLE   Df Model:                           31
Date:                Mon, 30 Nov 2020   Pseudo R-squ.:                  0.1964
Time:                        15:36:31   Log-Likelihood:                -4645.4
converged:                       True   LL-Null:                       -5780.7
Covariance Type:            nonrobust   LLR p-value:                     0.000
==============================================================================
                 coef    std err          z      P>|z|      [0.025
------------------------------------------------------------------------------
const         -5.6049      0.540    -10.382      0.000      -6.663
x1             0.0529      0.006      8.193      0.000       0.040
x2             0.0009      0.000      2.150      0.032    7.71e-05
x3            -0.0050      0.002     -2.026      0.043      -0.010
x4             0.0076      0.001      5.561      0.000       0.005
x5             0.0336      0.005      6.296      0.000       0.023
x6            -0.0437      0.031     -1.404      0.160      -0.105
x8             0.0097      0.004      2.349      0.019       0.002
x12            0.0014      0.004      0.349      0.727      -0.006
x13           -0.0144      0.018     -0.789      0.430      -0.050
x14           -0.0084      0.002     -4.272      0.000      -0.012
x16           -0.1292      0.016     -7.989      0.000      -0.161
x20           -0.0577      0.014     -3.984      0.000      -0.086
x21            0.0050      0.020      0.251      0.801      -0.034
x23            0.0033      0.002      1.743      0.081      -0.000
x10_1.0        0.3497      0.396      0.883      0.377      -0.426
```

```
x10_2.0               0.0038      0.396      0.010      0.992      -0.772
x10_3.0               0.2165      0.250      0.865      0.387      -0.274
x10_4.0              -0.1365      0.176     -0.777      0.437      -0.481
x10_5.0              -0.4657      0.256     -1.821      0.069      -0.967
x10_6.0               0.2242      0.171      1.314      0.189      -0.110
x10_7.0               0.3051      0.226      1.348      0.178      -0.139
x10_9.0              -1.0237      1.341     -0.763      0.445      -3.652
x11_3.0               0.1951      0.169      1.153      0.249      -0.136
x11_5.0               0.0115      0.102      0.113      0.910      -0.188
x11_7.0               0.1953      0.257      0.760      0.447      -0.308
x11_8.0               0.1067      0.163      0.653      0.514      -0.214
x22_(1.0, 3.0]       -0.2389      0.065     -3.654      0.000      -0.367
x22_(3.0, inf]       -0.4649      0.143     -3.254      0.001      -0.745
x18_(15.0, 25.0]     -0.2476      0.093     -2.670      0.008      -0.429
x18_(25.0, 60.0]     -0.3595      0.079     -4.531      0.000      -0.515
x18_(60.0, inf]      -0.7468      0.114     -6.571      0.000      -0.970
==============================================================================
```

x1, x18, x4, x2, x22_(1.0, 3.0] are the most important features, in order for the logistic model

```
data_dict.loc[[1,18,4,2,22]]
```

	Variable Names	Description	Monotonicity Constraint w.r.t. Prob(Bad = 1)
1	x1	Consolidated version of risk markers	Monotonically Decreasing
18	x18	Net Fraction Revolving Burden. This is revolvi...	Monotonically Increasing
4	x4	Average Months in File	Monotonically Decreasing
2	x2	Months Since Oldest Trade Open	Monotonically Decreasing

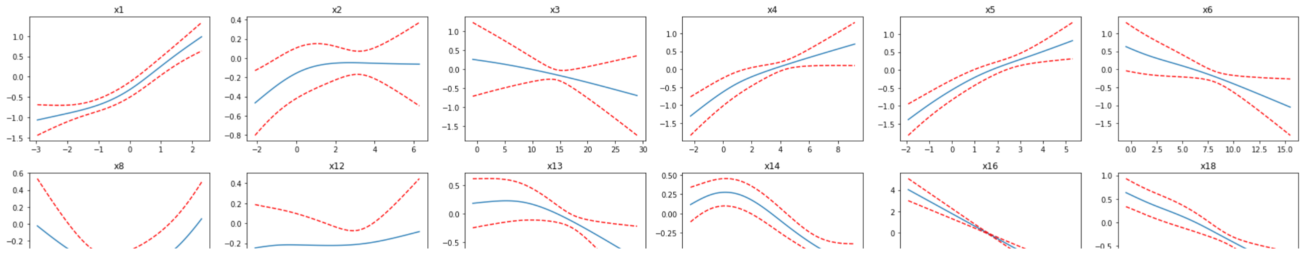
We see that the last bin of x18 has a much higher importance compared to the first two, we can infer that a greater balance to credit limit ratio has more importance to the model. Intuitively this means that the more outstanding balance has compared credit an applicant has, the applicant becomes less risky only when the balance is overwhelmingly high

b. General Additive Model

```
#B Spline

from pygam import LogisticGAM, s

X_train_GAM_bspline = X_train_scd
n_splines = 10

k=s(0,n_splines=n_splines)
for i in range(1,30):
  k += s(i,n_splines=n_splines)
print(k)
```

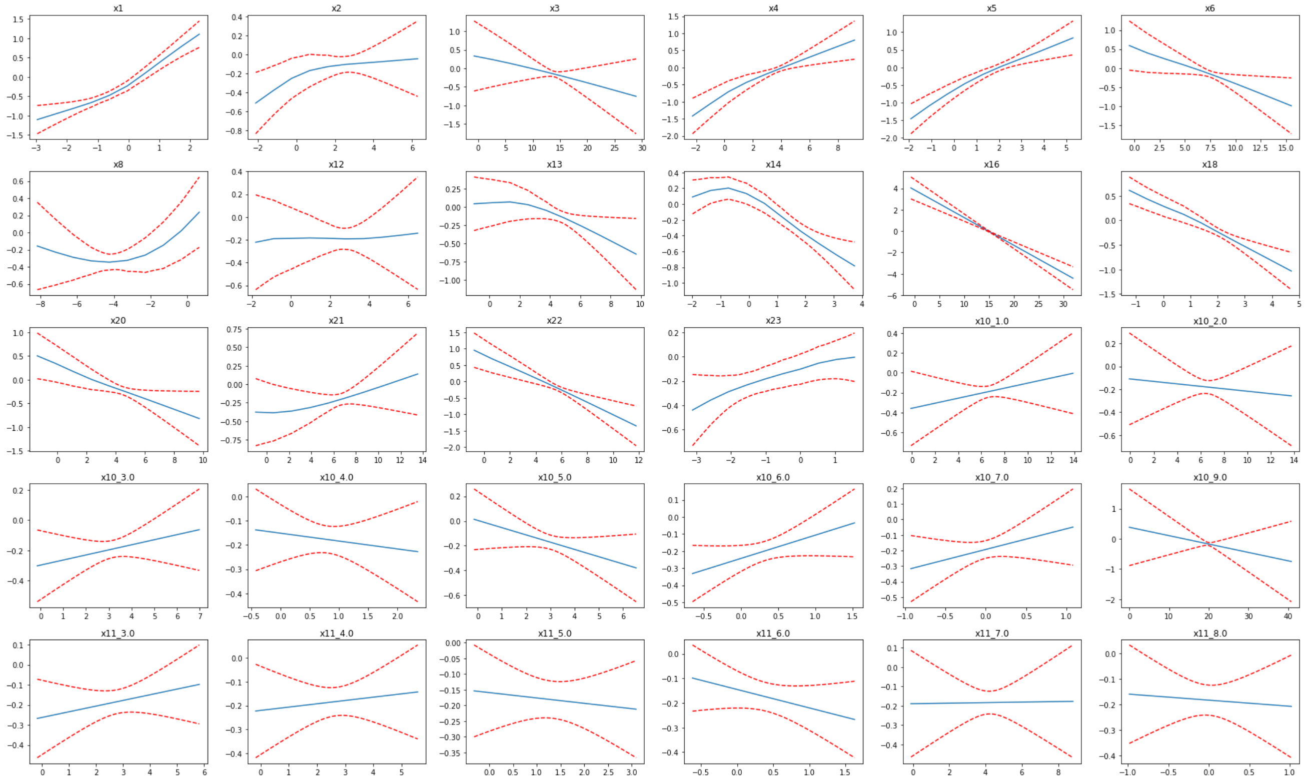
```
p_spl = LogisticGAM(k)
p_spl.gridsearch(X_train_GAM_bspline,y_train)

y_pred_train = p_spl.predict(X_train_GAM_bspline)
y_pred_test = p_spl.predict(X_test_scd)

# performance['General Additive Model with B splines : ']=accuracy_score(y_test,y_p

print('The Accuracy on training set:',accuracy_score(y_train,y_pred_train))
print('The Accuracy on testing set:',accuracy_score(y_test,y_pred_test))

# partial dependence plot
fig, axs = plt.subplots(5,6,figsize=(25,15))
for i, ax in enumerate(axs.flatten()):
    XX = p_spl.generate_X_grid(term=i)
    plt.subplot(ax)
    plt.plot(XX[:,i], p_spl.partial_dependence(term=i, X=XX))
    plt.plot(XX[:,i], p_spl.partial_dependence(term=i, X=XX, width=.95)[1], c='r',
    plt.title(X_train.columns[i])
plt.tight_layout()
```

```
N/A% (0 of 11) |                          | Elapsed Time: 0:00:00 ETA:  --:--:-
  9% (1 of 11) |##                        | Elapsed Time: 0:00:23 ETA:   0:03:5
 18% (2 of 11) |####                      | Elapsed Time: 0:00:37 ETA:   0:02:0
100% (11 of 11) |########################| Elapsed Time: 0:01:00 Time:   0:01:0
The Accuracy on training set: 0.7178780984313256
The Accuracy on testing set: 0.7093690248565966
```




```
#Piecewise ReLu
from pygam import LogisticGAM, s

X_train_GAM_ReLu = X_train_scd
n_splines = 10

k=s(0,n_splines=n_splines, spline_order=1)
for i in range(1,30):
  k += s(i,n_splines=n_splines, spline_order=1)
print(k)

p_spl = LogisticGAM(k)
p_spl.gridsearch(X_train_GAM_ReLu,y_train)

y_pred_train = p_spl.predict(X_train_GAM_ReLu)
y_pred_test = p_spl.predict(X_test_scd)

# performance['General Additive Model with Piecewise ReLU : ']=np.round(accuracy_sc

print('The Accuracy on training set:',accuracy_score(y_train,y_pred_train))
print('The Accuracy on testing set:',accuracy_score(y_test,y_pred_test))

# partial dependence plot
fig, axs = plt.subplots(5,6,figsize=(25,15))
for i, ax in enumerate(axs.flatten()):
    XX = p_spl.generate_X_grid(term=i)
    plt.subplot(ax)
    plt.plot(XX[:,i], p_spl.partial_dependence(term=i, X=XX))
    plt.plot(XX[:,i], p_spl.partial_dependence(term=i, X=XX, width=.95)[1], c='r',
    plt.title(X_train.columns[i])
plt.tight_layout()
```

```
N/A% (0 of 11) |                                        | Elapsed Time: 0:00:00 ETA:  --:--:-
100% (11 of 11) |########################| Elapsed Time: 0:00:22 Time:  0:00:2
The Accuracy on training set: 0.7183570829840737
The Accuracy on testing set: 0.7074569789674953
```



c. Support Vector Machine

```
from sklearn.svm import SVC
from sklearn.linear_model import SGDClassifier
from sklearn.model_selection import GridSearchCV
```

Linear SVC

```
#Standardizing testing data
X_test_scd = std_scaler.transform(X_test)

linSVC = SVC(kernel='linear', C=1, random_state=1)
linSVC.fit(X_train_scd, y_train)
y_train_pred_linSVC = linSVC.predict(X_train_scd)
y_test_pred_linSVC = linSVC.predict(X_test_scd)

performance['Linear SVC : ']=accuracy_score(y_test, y_test_pred_linSVC).round(4)
```

```
print('The accuracy on the train set is: {}'.format(accuracy_score(y_train, y_train
print('The accuracy on the test set is: {}'.format(accuracy_score(y_test, y_test_pr
```

```
The accuracy on the train set is: 0.71608
The accuracy on the test set is: 0.71033
```

```
#Stochastic gradient descent version of svm
sgd = SGDClassifier(loss='hinge')

sgd.fit(X_train_scd, y_train)
y_train_pred_sgd = sgd.predict(X_train_scd)
y_test_pred_sgd = sgd.predict(X_test_scd)

performance['SVM with Stochastic Gradient Descent version : ']=accuracy_score(y_tes
print('The accuracy on the train set is: {}'.format(accuracy_score(y_train, y_train
print('The accuracy on the test set is: {}'.format(accuracy_score(y_test, y_test_pr
```

```
The accuracy on the train set is: 0.70926
The accuracy on the test set is: 0.7108
```

RBF SVC

```
#DEFAULT: Gamma = 'scale' , C = 1.0
rbfSVC = SVC(kernel='rbf')
rbfSVC.fit(X_train_scd, y_train)
y_train_pred_rbfSVC = rbfSVC.predict(X_train_scd)
y_test_pred_rbfSVC = rbfSVC.predict(X_test_scd)

performance['RBF SVC : ']=accuracy_score(y_test, y_test_pred_rbfSVC).round(4)

print('The accuracy on the train set is: {}'.format(accuracy_score(y_train, y_train
print('The accuracy on the test set is: {}'.format(accuracy_score(y_test, y_test_pr
```

```
The accuracy on the train set is: 0.74638
The accuracy on the test set is: 0.71367
```

RBF SVC with hyperparameter tuning

```
tuned_parameters = {'kernel': ['rbf'], 'gamma':[0.0001, 0.001, 0.01, 0.1, 1.0],
                    'C': [0.01,0.1,1.0]}
svc_gs = GridSearchCV(SVC(), tuned_parameters, cv=5, scoring='accuracy')
svc_gs.fit(X_train_scd, y_train)
print('Best Parameters:',svc_gs.best_params_)
```

```
Best Parameters: {'C': 1.0, 'gamma': 0.001, 'kernel': 'rbf'}
```

```
y_train_pred_gs= svc_gs.predict(X_train_scd)
y_test_pred_gs= svc_gs.predict(X_test_scd)

performance['RBF SVC with hyperparameter tuning : ']=accuracy_score(y_test, y_test_
print('The accuracy on the train set is: {}'.format(accuracy_score(y_train, y_train
```

```
print('The accuracy on the test set is: {}'.format(accuracy_score(y_test, y_test_pr
```

```
The accuracy on the train set is: 0.715
The accuracy on the test set is: 0.7108
```

Interpretation

```
!pip install interpret
```

```
Collecting interpret
  Downloading https://files.pythonhosted.org/packages/4c/4a/df3e0d4c47ca7e5734
Collecting interpret-core[dash,debug,decisiontree,ebm,lime,linear,notebook,plc
  Downloading https://files.pythonhosted.org/packages/e0/f4/b3cd256fae2559c83b
     |████████████████████████████████| 5.2MB 5.1MB/s
Collecting dash-table>=4.1.0; extra == "dash"
  Downloading https://files.pythonhosted.org/packages/bb/46/cc839f897cabea3f58
     |████████████████████████████████| 1.8MB 40.7MB/s
Requirement already satisfied: requests>=2.19.0; extra == "dash" in /usr/local
Collecting dash>=1.0.0; extra == "dash"
  Downloading https://files.pythonhosted.org/packages/69/91/ae029886dda55b93b6
     |████████████████████████████████| 81kB 6.4MB/s
Collecting gevent>=1.3.6; extra == "dash"
  Downloading https://files.pythonhosted.org/packages/3f/92/b80b922f08f222faca
     |████████████████████████████████| 5.3MB 34.7MB/s
Collecting dash-cytoscape>=0.1.1; extra == "dash"
  Downloading https://files.pythonhosted.org/packages/a1/98/93b356b47aca71d4fb
     |████████████████████████████████| 3.6MB 38.3MB/s
Collecting psutil>=5.6.2; extra == "debug"
  Downloading https://files.pythonhosted.org/packages/33/e0/82d459af36bda999f8
     |████████████████████████████████| 471kB 31.0MB/s
Requirement already satisfied: joblib>=0.11; extra == "decisiontree" in /usr/l
Collecting lime>=0.1.1.33; extra == "lime"
  Downloading https://files.pythonhosted.org/packages/f5/86/91a13127d83d793ecb
     |████████████████████████████████| 276kB 31.9MB/s
Collecting ipykernel>=5.1.0; extra == "notebook"
  Downloading https://files.pythonhosted.org/packages/52/19/c2812690d8b340987e
     |████████████████████████████████| 122kB 41.2MB/s
Collecting ipython>=7.4.0; extra == "notebook"
  Downloading https://files.pythonhosted.org/packages/23/6a/210816c943c9aeeb29
     |████████████████████████████████| 788kB 45.8MB/s
Requirement already satisfied: plotly>=3.8.1; extra == "plotly" in /usr/local/
Requirement already satisfied: scipy>=0.18.1; extra == "required" in /usr/loca
Requirement already satisfied: pandas>=0.19.2; extra == "required" in /usr/loc
Requirement already satisfied: numpy>=1.11.1; extra == "required" in /usr/loca
Requirement already satisfied: scikit-learn>=0.18.1; extra == "required" in /u
Collecting SALib>=1.3.3; extra == "sensitivity"
  Downloading https://files.pythonhosted.org/packages/ba/36/84735444f4faded327
     |████████████████████████████████| 860kB 40.7MB/s
Collecting shap>=0.28.5; extra == "shap"
  Downloading https://files.pythonhosted.org/packages/85/a3/c0eab9dd6a894165e2
     |████████████████████████████████| 327kB 28.7MB/s
Requirement already satisfied: dill>=0.2.5; extra == "shap" in /usr/local/lib/
Collecting treeinterpreter>=0.2.2; extra == "treeinterpreter"
  Downloading https://files.pythonhosted.org/packages/56/cb/78ec761719d2546d4b
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.6/
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.6/dist-p
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.6/d
Requirement already satisfied: Flask>=1.0.2 in /usr/local/lib/python3.6/dist-p
Collecting flask-compress
  Downloading https://files.pythonhosted.org/packages/b2/7a/9c4641f975fb9daaf9
Collecting dash_renderer==1.8.3
  Downloading https://files.pythonhosted.org/packages/72/fe/59a322edb128ad1520
     |████████████████████████████████| 1.0MB 41.4MB/s
Collecting dash-core-components==1.13.0
  Downloading https://files.pythonhosted.org/packages/52/48/3dd8c7bf93cff3a9dc
     |████████████████████████████████| 3.5MB 37.5MB/s
Collecting dash-html-components==1.1.1
  Downloading https://files.pythonhosted.org/packages/02/ba/bb9427c62feb25bfba
     |████████████████████████████████| 194kB 40.8MB/s
```

```
Requirement already satisfied: future in /usr/local/lib/python3.6/dist-package
Collecting zope.interface
  Downloading https://files.pythonhosted.org/packages/82/b0/da8afd9b3bd50c7665
     |████████████████████████████████| 245kB 40.0MB/s
Collecting greenlet>=0.4.17; platform_python_implementation == "CPython"
  Downloading https://files.pythonhosted.org/packages/80/d0/532e160c777b42f6f3
     |████████████████████████████████| 51kB 5.8MB/s
Requirement already satisfied: setuptools in /usr/local/lib/python3.6/dist-pac
Collecting zope.event
  Downloading https://files.pythonhosted.org/packages/9e/85/b45408c64f3b888976
Requirement already satisfied: matplotlib in /usr/local/lib/python3.6/dist-pac
Requirement already satisfied: tqdm in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: scikit-image>=0.12 in /usr/local/lib/python3.6/
Requirement already satisfied: jupyter-client in /usr/local/lib/python3.6/dist
Requirement already satisfied: traitlets>=4.1.0 in /usr/local/lib/python3.6/di
Requirement already satisfied: tornado>=4.2 in /usr/local/lib/python3.6/dist-p
Requirement already satisfied: decorator in /usr/local/lib/python3.6/dist-pack
Collecting prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0
  Downloading https://files.pythonhosted.org/packages/8a/aa/198e6a857e83ea8b71
     |████████████████████████████████| 358kB 40.2MB/s
Requirement already satisfied: pickleshare in /usr/local/lib/python3.6/dist-pa
Requirement already satisfied: pygments in /usr/local/lib/python3.6/dist-packa
Requirement already satisfied: pexpect; sys_platform != "win32" in /usr/local/
Requirement already satisfied: jedi>=0.10 in /usr/local/lib/python3.6/dist-pac
Requirement already satisfied: backcall in /usr/local/lib/python3.6/dist-packa
Requirement already satisfied: six in /usr/local/lib/python3.6/dist-packages (
Requirement already satisfied: retrying>=1.3.3 in /usr/local/lib/python3.6/dis
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/pythor
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.6/dist-p
Collecting slicer==0.0.3
  Downloading https://files.pythonhosted.org/packages/02/a6/c708c5a0f338e99cfk
Requirement already satisfied: numba in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: Werkzeug>=0.15 in /usr/local/lib/python3.6/dist
Requirement already satisfied: click>=5.1 in /usr/local/lib/python3.6/dist-pac
Requirement already satisfied: itsdangerous>=0.24 in /usr/local/lib/python3.6/
Requirement already satisfied: Jinja2>=2.10.1 in /usr/local/lib/python3.6/dist
Collecting brotli
  Downloading https://files.pythonhosted.org/packages/b4/d3/7c98f05b7b9103e2f3
     |████████████████████████████████| 358kB 44.7MB/s
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.6/dist-p
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /us
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.6/c
Requirement already satisfied: pillow>=4.3.0 in /usr/local/lib/python3.6/dist-
Requirement already satisfied: networkx>=2.0 in /usr/local/lib/python3.6/dist-
Requirement already satisfied: imageio>=2.3.0 in /usr/local/lib/python3.6/dist
Requirement already satisfied: PyWavelets>=0.4.0 in /usr/local/lib/python3.6/c
Requirement already satisfied: jupyter-core>=4.6.0 in /usr/local/lib/python3.6
Requirement already satisfied: pyzmq>=13 in /usr/local/lib/python3.6/dist-pack
Requirement already satisfied: ipython-genutils in /usr/local/lib/python3.6/di
Requirement already satisfied: wcwidth in /usr/local/lib/python3.6/dist-packag
Requirement already satisfied: ptyprocess>=0.5 in /usr/local/lib/python3.6/dis
Requirement already satisfied: parso<0.8.0,>=0.7.0 in /usr/local/lib/python3.6
Requirement already satisfied: llvmlite<0.32.0,>=0.31.0dev0 in /usr/local/lib/
Requirement already satisfied: MarkupSafe>=0.23 in /usr/local/lib/python3.6/di
Building wheels for collected packages: dash-table, dash, dash-cytoscape, psut
  Building wheel for dash-table (setup.py) ... done
  Created wheel for dash-table: filename=dash_table-4.11.0-cp36-none-any.whl s
  Stored in directory: /root/.cache/pip/wheels/ca/37/90/bd45dcc5d6acbe6ac53f75
```

```
!pip install azureml-interpret
```

```
Requirement already satisfied: azureml-interpret in /usr/local/lib/python3.6/c
```

```
Requirement already satisfied: interpret-community==0.15.* in /usr/local/lib/p
Requirement already satisfied: shap<=0.34.0,>=0.20.0 in /usr/local/lib/python3
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.6/dist-p
Collecting interpret-core[required]<=0.2.1,>=0.1.20
  Using cached https://files.pythonhosted.org/packages/49/be/a678bac6f4e65b144
Requirement already satisfied: numpy in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: pandas in /usr/local/lib/python3.6/dist-package
Requirement already satisfied: packaging in /usr/local/lib/python3.6/dist-pack
Requirement already satisfied: scipy in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: tqdm>4.25.0 in /usr/local/lib/python3.6/dist-pa
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.6/dist-p
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.6/dist-p
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python
Requirement already satisfied: six in /usr/local/lib/python3.6/dist-packages (
Requirement already satisfied: pyparsing>=2.0.2 in /usr/local/lib/python3.6/di
ERROR: interpret 0.2.2 has requirement interpret-core[dash,debug,decisiontree,
Installing collected packages: interpret-core
  Found existing installation: interpret-core 0.2.2
    Uninstalling interpret-core-0.2.2:
      Successfully uninstalled interpret-core-0.2.2
Successfully installed interpret-core-0.2.1
  Created wheel for dash-html-components: filename=dash_html_components-1.1.1-
```

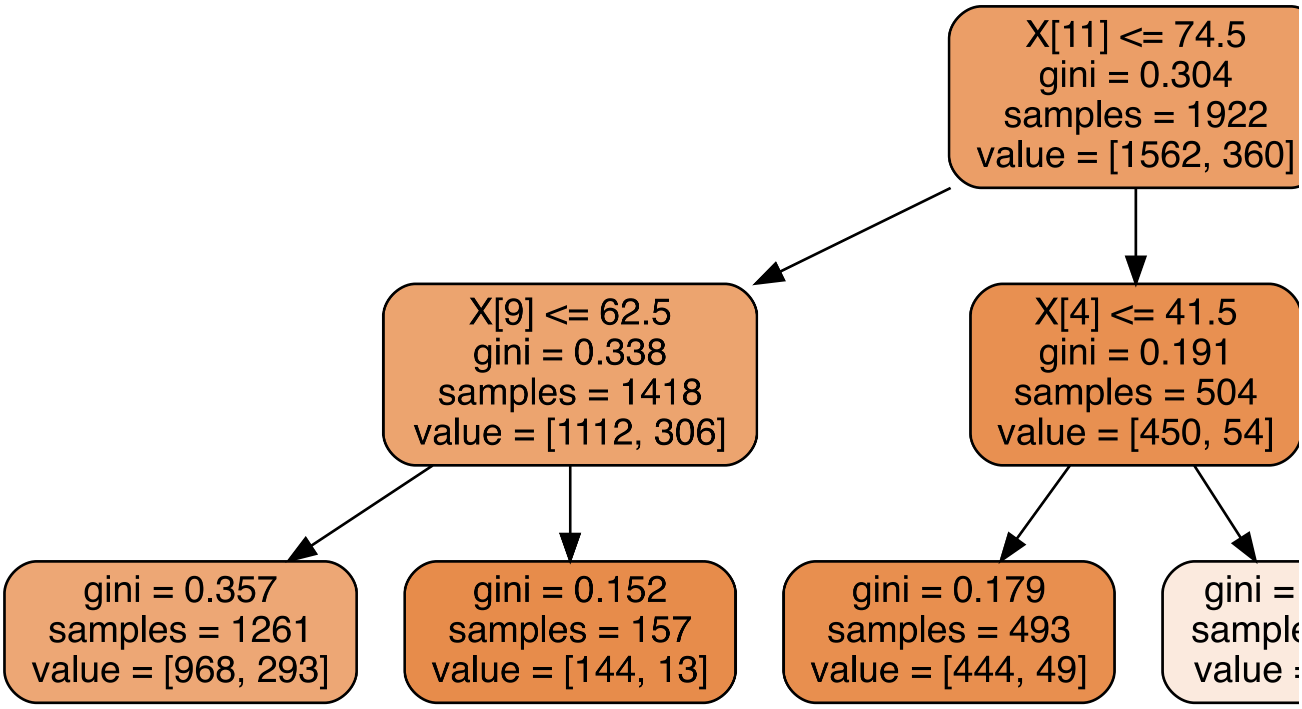
Features: 'x1', 'x2', 'x4', 'x6', 'x14', 'x18', 'x22', 'x23', 'x10'encoded, 'x11'encoded

```
from interpret.blackbox import PartialDependence

pdp = PartialDependence(predict_fn=svc_gs.predict, data=X_train_scd)
pdp_global = pdp.explain_global(name='Partial Dependence Plot')

show(pdp_global)
```

```
      Successfully uninstalled prompt-toolkit-1.0.18
```

d. Decision Tree

```
from sklearn.tree import DecisionTreeClassifier,plot_tree
from sklearn.metrics import accuracy_score
import graphviz
from sklearn.tree import export_graphviz

#Fitting Decision Tree Classifier
dt = DecisionTreeClassifier(max_depth=5)
dt.fit(X_train,y_train)

#Visualizing the tree
dot_graph = export_graphviz(
        dt,
        out_file=None,
        rounded=True,
        filled=True
    )
tree_graph = graphviz.Source(dot_graph)
tree_graph
```




```
#Accuracy on train and test data
performance['Decision Tree : ']=accuracy_score(y_test,dt.predict(X_test)).round(4)
print('Accuracy on train set:',accuracy_score(y_train,dt.predict(X_train)))
print('Accuracy on test set:',accuracy_score(y_test,dt.predict(X_test)))
```

```
Accuracy on train set: 0.7205125134714405
Accuracy on test set: 0.7088910133843213
```

e. XGBoost

y_train

```
2795     1
9142     1
808      0
10432    0
5611     1
        ..
9604     1
7360     0
10322    0
1443     1
3522     0
Name: RiskFlag, Length: 8351, dtype: int64
```

sorted(sklearn.metrics.SCORERS.keys())

```
['accuracy',
 'adjusted_mutual_info_score',
 'adjusted_rand_score',
 'average_precision',
 'balanced_accuracy',
 'completeness_score',
 'explained_variance',
 'f1',
 'f1_macro',
 'f1_micro',
 'f1_samples',
 'f1_weighted',
 'fowlkes_mallows_score',
 'homogeneity_score',
 'jaccard',
 'jaccard_macro',
 'jaccard_micro',
 'jaccard_samples',
 'jaccard_weighted',
 'max_error',
 'mutual_info_score',
 'neg_brier_score',
 'neg_log_loss',
 'neg_mean_absolute_error',
 'neg_mean_gamma_deviance',
 'neg_mean_poisson_deviance',
 'neg_mean_squared_error',
 'neg_mean_squared_log_error',
 'neg_median_absolute_error',
 'neg_root_mean_squared_error',
 'normalized_mutual_info_score',
 'precision',
 'precision_macro',
 'precision_micro',
 'precision_samples',
 'precision_weighted',
 'r2',
 'recall',
 'recall_macro',
 'recall_micro',
 'recall_samples',
 'recall_weighted',
 'roc_auc',
```

```
      'roc_auc_ovo',
      'roc_auc_ovo_weighted',
      'roc_auc_ovr',
      'roc_auc_ovr_weighted',
      'v_measure_score']
```

```
#Tuned model 2 (Low learning rate with its optimal estimator=130)
xgb_tuned2 = XGBClassifier(learning_rate =0.01, n_estimators=130, max_depth=5, min_
                    colsample_bylevel= 0.5, colsample_bynode=0.6, objective= 'binar
xgb_tuned2.fit(X_train, y_train)
y_test_pred_xgb2 = xgb_tuned2.predict(X_test)
y_train_pred_xgb2 = xgb_tuned2.predict(X_train)
performance['XGB with lower learning rate : ']=accuracy_score(y_test, y_test_pred_x
print(modelfit(xgb_tuned2))
print('The accuracy on the train set is: {}'.format(accuracy_score(y_train, y_train
print('The accuracy on the test set is: {}'.format(accuracy_score(y_test, y_test_pr
```

```
     Best AUC: 0.79595 with 64 rounds
     None
     The accuracy on the train set is: 0.73512
     The accuracy on the test set is: 0.71941
```

```
import xgboost as xgb
from xgboost import XGBClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import StratifiedKFold

xgb_model = xgb.XGBClassifier()

parameters = {'nthread':[4], #when use hyperthread, xgboost may become slower
              'objective':['binary:logistic'],
              'learning_rate': [0.01], #so called `eta` value
              'max_depth': [5],
              'scale_pos_weight': [1],
              'gamma': [0.2],
              'min_child_weight': [0.2],
              'subsample': [0.8],
              'colsample_bytree': [0.6],
              'colsample_bylevel': [0.5],
              'colsample_bynode': [0.6],
              'n_estimators': [130], #number of trees, change it to 1000 for better
              'seed': [1],
              'reg_alpha':[0.02],
              'reg_lambda':[0.05]}

clf = GridSearchCV(xgb_model, parameters, n_jobs=5,
                   cv=StratifiedKFold(n_splits=5, shuffle=True),
                   scoring='roc_auc',
                   verbose=2, refit=True)

clf.fit(X_train, y_train)

print(clf.best_params_)
```

```
Fitting 5 folds for each of 4 candidates, totalling 20 fits
[Parallel(n_jobs=5)]: Using backend LokyBackend with 5 concurrent workers.
[Parallel(n_jobs=5)]: Done  20 out of  20 | elapsed:   13.1s finished
{'colsample_bylevel': 0.5, 'colsample_bynode': 0.6, 'colsample_bytree': 0.6, '
```

```
import xgboost as xgb
from xgboost import XGBClassifier

#returns the optimum number of trees required at specific learning rate
def modelfit(alg):
    xgb_param = alg.get_xgb_params()
    xgb_param['eval_metric']='auc'
    dtrain = xgb.DMatrix(X_train, label=y_train)
    dtest = xgb.DMatrix(X_test, label=y_test)
    model = xgb.train(xgb_param, dtrain, num_boost_round=alg.get_params()['n_estima
                      evals=[(dtest,"Test")], early_stopping_rounds=20, verbose_eva

    print("Best AUC: {:.5f} with {} rounds".format(model.best_score, model.best_ite


xgb1 = XGBClassifier(learning_rate =0.1, n_estimators=1000, max_depth=7, min_child_
 colsample_bytree=0.8, objective= 'binary:logistic', nthread=4, scale_pos_weight=1,

#Fit model to find the optimal number of trees at learning rate 0.1
modelfit(xgb1)
```

```
Best AUC: 0.78660 with 42 rounds
```

```
#Parameter test for max_depth and min_child_weight using GridSearchCV
from sklearn.model_selection import GridSearchCV
param_test = {'max_depth': [3,5,10], 'min_child_weight': [1,4,8]}
##optimal number of trees =16
gsearch1 = GridSearchCV(estimator = XGBClassifier(learning_rate =0.1, n_estimators=
                        param_grid = param_test, scoring='roc_auc',n_jobs=4,iid=Fal
gsearch1.fit(X_train,y_train)
print(gsearch1.best_params_)
```

```
{'max_depth': 5, 'min_child_weight': 8}
```

```
# re-calibrate the number of boosting rounds for the updated parameters.(max_depth=
xgb2 = XGBClassifier(learning_rate =0.1, n_estimators=1000, max_depth=5, min_child_
 colsample_bytree=0.6, objective= 'binary:logistic', nthread=1, scale_pos_weight=1,

#Fit model to find the optimal number of trees at learning rate 0.1
modelfit(xgb2)
```

```
Best AUC: 0.79770 with 54 rounds
```

```
#Parameter test for gamma using GridSearchCV
from sklearn.model_selection import GridSearchCV
param_test = {'gamma':[0, 0.1, 0.2, 0.3, 0.4, 0.5]}
#Optimal number of trees=32
gsearch1 = GridSearchCV(estimator = XGBClassifier(learning_rate =0.1, max_depth=5,
                                                  , n_estimators=32, objective= 'bi
```

```
                        param_grid = param_test, scoring='roc_auc',n_jobs=4,iid=Fal
gsearch1.fit(X_train,y_train)
print(gsearch1.best_params_)
```

    {'gamma': 0.1}

```
#Parameter test for subsample and colsample_bytree using GridSearchCV
from sklearn.model_selection import GridSearchCV
param_test = {
 'subsample':[i/10.0 for i in range(6,10)],
 'colsample_bytree':[i/10.0 for i in range(6,10)]
}
#optial number of trees = 32
gsearch1 = GridSearchCV(estimator = XGBClassifier(learning_rate =0.1, max_depth=5,
                                                        , n_estimators=32, objective= 'bi
                        param_grid = param_test, scoring='roc_auc',n_jobs=4,iid=Fal
gsearch1.fit(X_train,y_train)
print(gsearch1.best_params_)
```

    {'colsample_bytree': 0.6, 'subsample': 0.7}

```
# re-calibrate the number of boosting rounds for the updated parameters. (subsample
xgb3 = XGBClassifier(learning_rate =0.1, n_estimators=1000, max_depth=5, min_child_
 colsample_bytree=0.6, objective= 'binary:logistic', nthread=4, scale_pos_weight=1,

#Fit model to find the optimal number of trees at learning rate 0.01
modelfit(xgb3)
```

    Best AUC: 0.79240 with 27 rounds

```
#Tuned model 1 (High learning rate with its optimal estimator=34)
xgb_tuned1 = XGBClassifier(learning_rate =0.1, n_estimators=34, max_depth=5, min_ch
                    objective= 'binary:logistic', nthread=4, scale_pos_weight=1, se
xgb_tuned1.fit(X_train, y_train)
y_test_pred_xgb = xgb_tuned1.predict(X_test)
y_train_pred_xgb = xgb_tuned1.predict(X_train)
performance['XGB with higher learning rate : ']=accuracy_score(y_test, y_test_pred_
print('The accuracy on the train set is: {}'.format(accuracy_score(y_train, y_train
print('The accuracy on the test set is: {}'.format(accuracy_score(y_test, y_test_pr
```

    The accuracy on the train set is: 0.74734
    The accuracy on the test set is: 0.71702

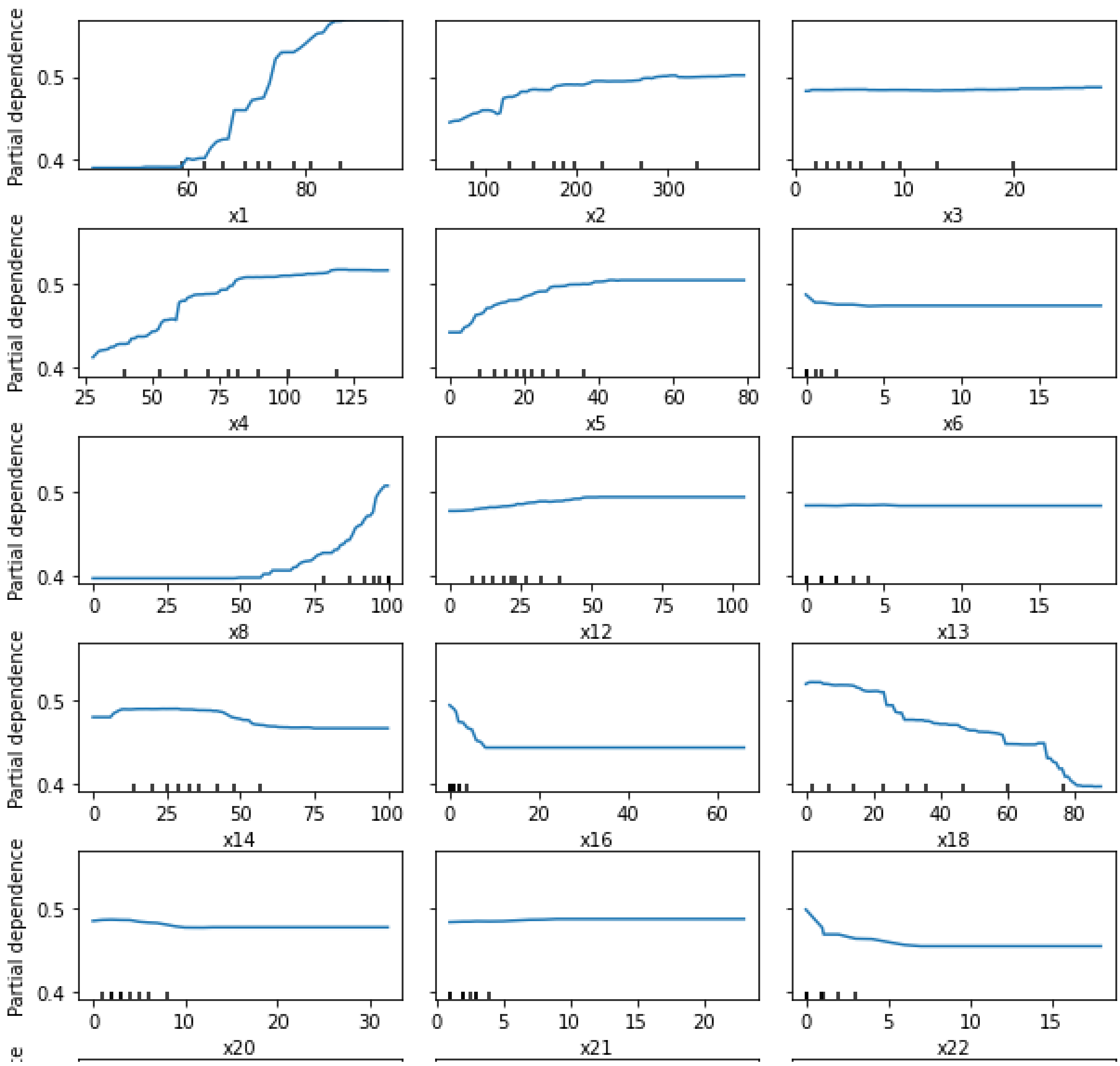
```
#CV with lower learning rate and higher estimators
xgb3 = XGBClassifier(learning_rate =0.01, n_estimators=5000, max_depth=5, min_child
 colsample_bytree=0.6, objective= 'binary:logistic', nthread=4, scale_pos_weight=1,

#Fit model to find the optimal number of trees at learning rate 0.01
modelfit(xgb3)
```

    Best AUC: 0.79291 with 38 rounds

```
#Tuned model 2 (Low learning rate with its optimal estimator=130)
xgb_tuned2 = XGBClassifier(learning_rate =0.01, n_estimators=130, max_depth=5, min_
                    objective= 'binary:logistic', nthread=4, scale_pos_weight=1, se
xgb_tuned2.fit(X_train, y_train)
y_test_pred_xgb2 = xgb_tuned2.predict(X_test)
y_train_pred_xgb2 = xgb_tuned2.predict(X_train)
performance['XGB with lower learning rate : ']=accuracy_score(y_test, y_test_pred_x
print(modelfit(xgb_tuned2))
print('The accuracy on the train set is: {}'.format(accuracy_score(y_train, y_train
print('The accuracy on the test set is: {}'.format(accuracy_score(y_test, y_test_pr
```

```
Best AUC: 0.79289 with 11 rounds
None
The accuracy on the train set is: 0.73823
The accuracy on the test set is: 0.71845
```

```
#partial dependence plot
from sklearn.inspection import plot_partial_dependence
plot_partial_dependence(xgb_tuned2, X_train, X_train.columns.tolist())
fig = plt.gcf()
fig.set_size_inches(10, 20)
fig.subplots_adjust(wspace=0.1, hspace=0.4)
```



f. Neural Network

```
def X_RELU(X, tau=None,set='train'):
    X_tmp = X.copy()
    name=X.name
    if set=='train':
        K = 4
        tau = np.linspace(X_tmp.min(),X_tmp.max(),K+2)[1:-1]
        xphi = X_tmp
        for k in range(len(tau)):
            tmp = [max(x1-tau[k], 0) for x1 in X_tmp]
            xphi = np.column_stack((xphi, tmp))

    xphi = X_tmp
    for k in range(len(tau)):
        tmp = [max(x1-tau[k], 0) for x1 in X_tmp]
        xphi = np.column_stack((xphi, tmp))

    X_tmp = pd.DataFrame(xphi)
    X_tmp.drop(0, axis = 1, inplace = True)
    for i in range(0,X_tmp.shape[1]+1):
        X_tmp = X_tmp.rename(columns={i: name+'_'+str(i)})
```

```
    return X_tmp, tau
```

```
X_train_RELU_final_val=pd.DataFrame()
X_test_RELU_final_val=pd.DataFrame()

for column in X_train.columns[0:]:
    X_train_RELU, tau= X_RELU(X_train[column], set='train')
    X_test_RELU= X_RELU(X_test[column],tau, set='test')[0]
    X_train_RELU_final_val = pd.concat([X_train_RELU_final_val,X_train_RELU],axis=1
    X_test_RELU_final_val = pd.concat([X_test_RELU_final_val,X_test_RELU],axis=1)

X_train_RELU = X_train_RELU_final_val.values
X_test_RELU = X_test_RELU_final_val.values

feature_names_RELU=X_train_RELU_final_val.columns[:].values
feature_names_RELU
```

```
array(['x1_1', 'x1_2', 'x1_3', 'x1_4', 'x2_1', 'x2_2', 'x2_3', 'x2_4',
       'x3_1', 'x3_2', 'x3_3', 'x3_4', 'x4_1', 'x4_2', 'x4_3', 'x4_4',
       'x5_1', 'x5_2', 'x5_3', 'x5_4', 'x6_1', 'x6_2', 'x6_3', 'x6_4',
       'x8_1', 'x8_2', 'x8_3', 'x8_4', 'x12_1', 'x12_2', 'x12_3', 'x12_4',
       'x13_1', 'x13_2', 'x13_3', 'x13_4', 'x14_1', 'x14_2', 'x14_3',
       'x14_4', 'x16_1', 'x16_2', 'x16_3', 'x16_4', 'x18_1', 'x18_2',
       'x18_3', 'x18_4', 'x20_1', 'x20_2', 'x20_3', 'x20_4', 'x21_1',
       'x21_2', 'x21_3', 'x21_4', 'x22_1', 'x22_2', 'x22_3', 'x22_4',
       'x23_1', 'x23_2', 'x23_3', 'x23_4', 'x10_1.0_1', 'x10_1.0_2',
       'x10_1.0_3', 'x10_1.0_4', 'x10_2.0_1', 'x10_2.0_2', 'x10_2.0_3',
       'x10_2.0_4', 'x10_3.0_1', 'x10_3.0_2', 'x10_3.0_3', 'x10_3.0_4',
       'x10_4.0_1', 'x10_4.0_2', 'x10_4.0_3', 'x10_4.0_4', 'x10_5.0_1',
       'x10_5.0_2', 'x10_5.0_3', 'x10_5.0_4', 'x10_6.0_1', 'x10_6.0_2',
       'x10_6.0_3', 'x10_6.0_4', 'x10_7.0_1', 'x10_7.0_2', 'x10_7.0_3',
       'x10_7.0_4', 'x10_9.0_1', 'x10_9.0_2', 'x10_9.0_3', 'x10_9.0_4',
       'x11_3.0_1', 'x11_3.0_2', 'x11_3.0_3', 'x11_3.0_4', 'x11_4.0_1',
       'x11_4.0_2', 'x11_4.0_3', 'x11_4.0_4', 'x11_5.0_1', 'x11_5.0_2',
       'x11_5.0_3', 'x11_5.0_4', 'x11_6.0_1', 'x11_6.0_2', 'x11_6.0_3',
       'x11_6.0_4', 'x11_7.0_1', 'x11_7.0_2', 'x11_7.0_3', 'x11_7.0_4',
       'x11_8.0_1', 'x11_8.0_2', 'x11_8.0_3', 'x11_8.0_4'], dtype=object)
```

```
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import RandomizedSearchCV

tuned_parameters = {'solver': ['adam'],'alpha': 10.0 ** -np.arange(-4, 4,4), 'hidde

MLP_RELU = RandomizedSearchCV(MLPClassifier(max_iter=1000), tuned_parameters, cv=5,
                       scoring='accuracy')
MLP_RELU.fit(X_train_RELU, y_train)

performance['MLP with Piecewise ReLU : ']=accuracy_score(y_test,MLP_RELU.predict(X_
print('Training accuracy for the MLPClassifier with Piecewise ReLU:', accuracy_scor
print('Testing accuracy for the MLPClassifier with Piecewise ReLU:', accuracy_score
MLP_RELU
```

```
Training accuracy for the MLPClassifier with Piecewise ReLU: 0.712
Testing accuracy for the MLPClassifier with Piecewise ReLU: 0.7189
RandomizedSearchCV(cv=5, error_score=nan,
```

```
                             estimator=MLPClassifier(activation='relu', alpha=0.0001,
                                                     batch_size='auto', beta_1=0.9,
                                                     beta_2=0.999, early_stopping=False,
                                                     epsilon=1e-08,
                                                     hidden_layer_sizes=(100,),
                                                     learning_rate='constant',
                                                     learning_rate_init=0.001,
                                                     max_fun=15000, max_iter=1000,
                                                     momentum=0.9, n_iter_no_change=10,
                                                     nesterovs_momentum=True, power_t=0.
                                                     rando...
                                                     solver='adam', tol=0.0001,
                                                     validation_fraction=0.1,
                                                     verbose=False, warm_start=False),
                             iid='deprecated', n_iter=10, n_jobs=None,
                             param_distributions={'alpha': array([1.e+04, 1.e+00]),
                                                  'hidden_layer_sizes': [10, 10, 10],
                                                  'random_state': [0, 1, 2, 3],
                                                  'solver': ['adam']},
                             pre_dispatch='2*n_jobs', random_state=None, refit=True,
                             return_train_score=False, scoring='accuracy', verbose=0)
```

```
import shap
feature_names=X_train.columns.tolist()

def IntergrateSHAP(feature_names,feature_names_raw,shap_range,shap_values):
  feature_list = list(feature_names).copy()
  shap_integrated_all=[]
  for i in range(shap_range):
    shap_integrated=[]
    for column in feature_names_raw:
        shap_integrating = 0
        for feature_name in feature_names:
            if feature_name.startswith(column+'_'):
                feature_index = feature_list.index(feature_name)
                shap_integrating = shap_integrating + shap_values[i][feature_index]
        shap_integrated.append(shap_integrating)
    print(shap_integrated)
    shap_integrated_all.append(shap_integrated)
  return shap_integrated_all

def PlotSHAP(MLP_BinAll,x_train_BinAll,feature_names_BinAll,x_train_raw,feature_nam
  # define the explainer
  explainer = shap.KernelExplainer(MLP_BinAll.predict,shap.sample(x_train_BinAll, 5
  # calculate the shape value on training data
  # set approximate=True for fast processing
  shap_values = explainer.shap_values(x_train_BinAll[:shap_range], approximate=True
  print(shap_values)


  ## shap_values_int = shap values after integration, it is a np.array with shape:
  ## feature_names = np.array(['x1','x2',...,'x23'])
  shap_values_int=np.array(IntergrateSHAP(feature_names_BinAll,feature_names_raw,sh
  feature_names=X_train.columns.tolist()

  shap.summary_plot(np.array(shap_values_int), feature_names=feature_names, plot_ty
  shap.summary_plot(np.array(shap_values_int), feature_names=feature_names)
```

```
  # plot pdp
  for column in feature_names_raw:
    shap.dependence_plot(column, shap_values_int, x_train_raw[:shap_range], feature

  return (shap_values_int,feature_names)

(shap_values_int,feature_names) = PlotSHAP(MLP_RELU,X_train_RELU,feature_names_RELU
```

```
100%                                              50/50 [00:38<00:00, 1.31it/s]

[[ 0.18211336  0.10761349  0.07835413 ...  0.          0.
   0.        ]
 [ 0.19414117  0.11706507  0.09066505 ...  0.          0.
   0.        ]
 [ 0.10429245  0.0418404   0.02103324 ...  0.          0.
   0.00066863]
 ...
 [-0.09213151 -0.02649931 -0.0270788  ...  0.          0.
   0.        ]
 [ 0.          0.         -0.02607195 ...  0.          0.
   0.        ]
 [ 0.14373704  0.07615778  0.0679298  ...  0.          0.
   0.        ]]
[0.3680809726169261, 0.01589022684297184, 0.0, -0.01535764343756809, -0.020567
[0.4018712903005297, -0.022890432516824044, 0.0, 0.022689919050247587, -0.0487
[0.16716609424719064, 0.044585098694840286, 0.0, 0.08327393164408281, 0.026644
[-0.19678108533396424, -0.024611300614300776, 0.0, -0.010473522779842603, -0.0
[0.2940487380156453, 0.0676751120236976, 0.0, 0.00418839725837112, -0.03540720
[-0.034132104986512965, 0.16470335632232488, 0.0, 0.23830777499620617, -0.1343
[-0.09180746598327244, -0.046258144881548086, 0.0, -0.03479512108669447 6, -0.0
[0.16339659136796886, 0.04968449787539503, 0.0, 0.08186749496181446, 0.0961306
[-0.17621160254741813, -0.0333094579537072, 0.0, -0.009664961910561709, -0.033
[-0.2527969493646858, 0.0880560709570437, 0.0, -0.023492035977337004, -0.04391
[-0.23081605891209192, 0.2954160888420392, 0.0, 0.0, 0.31448393826927745, 0.0,
[-0.12161204613265605, -0.052313410441265654, 0.0, -0.016729241087517034, -0.0
[-0.16302050048827654, -0.009155171805086751, 0.0, -0.014470028177721639, -0.0
[-0.400191437254598, 0.10637191686335322, 0.0, 0.19690522307684485, 0.03189127
[-0.2677743658339764, -0.03026754570096575, 0.0, -0.01175904291648229, -0.0517
[-0.19254582388047703, 0.0, 0.0, 0.028681250905369087, -0.0236069835702885 1, 0
[-0.33321039659556967, -0.012161359401431891, 0.0, -0.04024770996451134, 0.112
[-0.07025828370452819, -0.02975071717001404, 0.0, 0.23215643365543925, -0.0912
[0.018817859730566242, 0.02072556082611904, 0.0, -0.03217142543874353, -0.0317
[0.3988706058023255, 0.039504509368260804, 0.0, 0.0, -0.04320534654508265, 0.0
[-0.22783861313394957, 0.062003048484228766, 0.0, 0.10696961227608043, -0.0352
[0.32499692764819277, 0.021093165832737182, 0.0, 0.0, 0.030109291130048066, 0.
[-0.05381431666377881, 0.07797666573357787, 0.0, 0.05072302126049877, 0.217002
[-0.21506704357984635, -0.03846850200454188, 0.0, -0.02619964474512093, 0.0293
[-0.1976460689567586, -0.060896639068040145, 0.0, -0.06030702137635696, -0.088
[0.2701953888641074, 0.03034300533016212, 0.0, 0.0445054538090673 8, 0.05929791
[-0.12288641907938137, 0.05326163883954875, 0.0, 0.23538857389316506, -0.08519
[0.07988130097673277, 0.22508791189743182, 0.0, 0.02042794709053477, 0.0882321
[-0.3297825401430592, -0.05934315978277128, 0.0, -0.0695352853686437, 0.098744
[-0.31087855664437397, -0.0014317398307850393, 0.0, 0.08727071384725948, 0.149
[0.34955125825817984, -0.0516174286527637, 0.0, -0.018491865610923336, -0.0729
[0.01866734529578199, 0.025443378996417975, 0.0, -0.0435364085148768, 0.018923
[-0.174465683511762, 0.0, 0.0, 0.03663753947006548, 0.037369215332410055, 0.0,
[0.08844286765813127, 0.05865692942652341, 0.0, 0.0, 0.1806055629251289, 0.0,
[-0.1602436370134046, -0.2475148733607917, 0.0, -0.037357553096544045, 0.0, 0.
[0.1921882862306837, 0.164862960075 2361, 0.0, 0.0, 0.041240831447339005, 0.0,
[-0.230063674623225, 0.01692888756053866, 0.0, 0.07555583767483026, 0.08488869
[-0.0273216198800 2913, 0.1582823044594433, 0.0, 0.05095408601858661, 0.1802523
[0.3075032303190275, 0.0220915888946477, 0.0, 0.0, 0.00352266852459044, 0.0008
[0.32386829663523387, -0.013663630401344384, 0.0, 0.0, -0.04900844657131481, 0
[-0.21717286612167394, 0.1109461027973652, 0.0, 0.033089223982484324, 0.287689
[-0.05536973758262114, -0.07986196555283359, 0.0, -0.04677022050949717, -0.038
[0.2409470588083871, 0.040173530317886744, 0.0, 0.03202713363838618, 0.0383514
[-0.18153886442317196, -0.032295547865453945, 0.0, -0.012187770728106156, -0.0
[-0.1053012040762545 7, -0.05773801365365993, 0.0, -0.06121860909749002, -0.064
```

```
[-0.31655505585602617, -0.04800905621364215, 0.0, -0.041658263567l2883, 0.0356
[-0.015507246824077031, 0.022682826505784227, 0.0, -0.06586036398593476, 0.0,
[-0.14570961709811436, -0.029516131827756836, 0.0, 0.12417795257896236, -0.064
[-0.026071954800467263, -0.10787811717784179, 0.0, -0.06301229881656195, -0.06
[0.2878246248782219, 0.017960414392968 59, 0.0, 0.05995827872683354, 0.03923347
```