CustomerSatisfactionLevelPrediction

March 20, 2020

1 Predicting Customer Satisfaction Level from Santander

This project is on the kaggle platform at following link: https://www.kaggle.com/c/santander-customer-satisfaction

Description: From frontline support teams to C-suites, customer satisfaction is a key measure of success. Unhappy customers don't stick around. What's more, unhappy customers rarely voice their dissatisfaction before leaving.

The dataset is anonymized and consists of a large number of numeric variables. I built a final model using a K-nearest neighbor algorithm with 99% accuracy when applied to the test data provided by kaggle. The scripts were made in Python, using Pandas, Numpy, scipy and Scikit-Learn frameworks.

[11]:
```python
# Importing libraries and frameworks
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

%matplotlib inline

import scipy.stats
from sklearn.feature_selection import VarianceThreshold
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.feature_selection import f_classif,chi2
from sklearn.model_selection import train_test_split

import random

from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import confusion_matrix, classification_report,␣
 ↪accuracy_score

from sklearn.externals import joblib
```

```python
import warnings
warnings.filterwarnings("ignore")
```

1.1 Importing dataset

[2]:
```python
# Importing train data
df_train = pd.read_csv("data/train.csv")
df_train.head()
```

[2]:
```
   ID  var3  var15  imp_ent_var16_ult1  imp_op_var39_comer_ult1  \
0   1     2     23                 0.0                      0.0
1   3     2     34                 0.0                      0.0
2   4     2     23                 0.0                      0.0
3   8     2     37                 0.0                    195.0
4  10     2     39                 0.0                      0.0

   imp_op_var39_comer_ult3  imp_op_var40_comer_ult1  imp_op_var40_comer_ult3  \
0                      0.0                      0.0                      0.0
1                      0.0                      0.0                      0.0
2                      0.0                      0.0                      0.0
3                    195.0                      0.0                      0.0
4                      0.0                      0.0                      0.0

   imp_op_var40_efect_ult1  imp_op_var40_efect_ult3  ...  \
0                      0.0                      0.0  ...
1                      0.0                      0.0  ...
2                      0.0                      0.0  ...
3                      0.0                      0.0  ...
4                      0.0                      0.0  ...

   saldo_medio_var33_hace2  saldo_medio_var33_hace3  saldo_medio_var33_ult1  \
0                      0.0                      0.0                     0.0
1                      0.0                      0.0                     0.0
2                      0.0                      0.0                     0.0
3                      0.0                      0.0                     0.0
4                      0.0                      0.0                     0.0

   saldo_medio_var33_ult3  saldo_medio_var44_hace2  saldo_medio_var44_hace3  \
0                     0.0                      0.0                      0.0
1                     0.0                      0.0                      0.0
2                     0.0                      0.0                      0.0
3                     0.0                      0.0                      0.0
4                     0.0                      0.0                      0.0

   saldo_medio_var44_ult1  saldo_medio_var44_ult3          var38  TARGET
0                     0.0                     0.0   39205.170000       0
```

```
1                0.0                0.0   49278.030000       0
2                0.0                0.0   67333.770000       0
3                0.0                0.0   64007.970000       0
4                0.0                0.0  117310.979016       0

[5 rows x 371 columns]
```

[3]:

```python
# Importing test data
df_data_test = pd.read_csv("data/test.csv")
df_result_test = pd.read_csv("data/sample_submission.csv")
df_test = df_data_test.merge(df_result_test, on = 'ID')
df_test.head()
```

[3]:

```
   ID  var3  var15  imp_ent_var16_ult1  imp_op_var39_comer_ult1  \
0   2     2     32                 0.0                      0.0
1   5     2     35                 0.0                      0.0
2   6     2     23                 0.0                      0.0
3   7     2     24                 0.0                      0.0
4   9     2     23                 0.0                      0.0

   imp_op_var39_comer_ult3  imp_op_var40_comer_ult1  imp_op_var40_comer_ult3  \
0                      0.0                      0.0                      0.0
1                      0.0                      0.0                      0.0
2                      0.0                      0.0                      0.0
3                      0.0                      0.0                      0.0
4                      0.0                      0.0                      0.0

   imp_op_var40_efect_ult1  imp_op_var40_efect_ult3  ...  \
0                      0.0                      0.0  ...
1                      0.0                      0.0  ...
2                      0.0                      0.0  ...
3                      0.0                      0.0  ...
4                      0.0                      0.0  ...

   saldo_medio_var33_hace2  saldo_medio_var33_hace3  saldo_medio_var33_ult1  \
0                      0.0                      0.0                     0.0
1                      0.0                      0.0                     0.0
2                      0.0                      0.0                     0.0
3                      0.0                      0.0                     0.0
4                      0.0                      0.0                     0.0

   saldo_medio_var33_ult3  saldo_medio_var44_hace2  saldo_medio_var44_hace3  \
0                     0.0                      0.0                     0.0
1                     0.0                      0.0                     0.0
2                     0.0                      0.0                     0.0
3                     0.0                      0.0                     0.0
4                     0.0                      0.0                     0.0
```

```
   saldo_medio_var44_ult1  saldo_medio_var44_ult3      var38  TARGET
0                     0.0                     0.0   40532.10       0
1                     0.0                     0.0   45486.72       0
2                     0.0                     0.0   46993.95       0
3                     0.0                     0.0  187898.61       0
4                     0.0                     0.0   73649.73       0

[5 rows x 371 columns]
```

"var3" seens to be customers nationality. -999999 value indicate an unknown customer nationality. So I checked for missing values and replaced it.

1.2 Exploratory Analysis and Feature Engineering

[7]:

```
# Getting top-10 most common values
print(df_train.var3.value_counts()[:10])

# There are 116 missing values.
print("\nmissing values: ", len(df_train.loc[df_train.var3==-999999]))

# Replacing missing values (-999999) by most common value 2
df_train.var3 = df_train.var3.replace(-999999,2)
df_test.var3 = df_test.var3.replace(-999999,2)
```

```
2     74281
8       138
9       110
3       108
1       105
13       98
7        97
4        86
12       85
6        82
Name: var3, dtype: int64

missing values:  0
```

[8]:

```
# Saving dataset
df_train.to_csv("data/df_train.csv", index=False)
df_test.to_csv("data/df_test.csv", index=False)
```

[9]:

```
# Checking for missing values
print(pd.isna(df_train).any().any())
print(pd.isna(df_test).any().any())
print(pd.isnull(df_train).any().any())
print(pd.isnull(df_test).any().any())
```

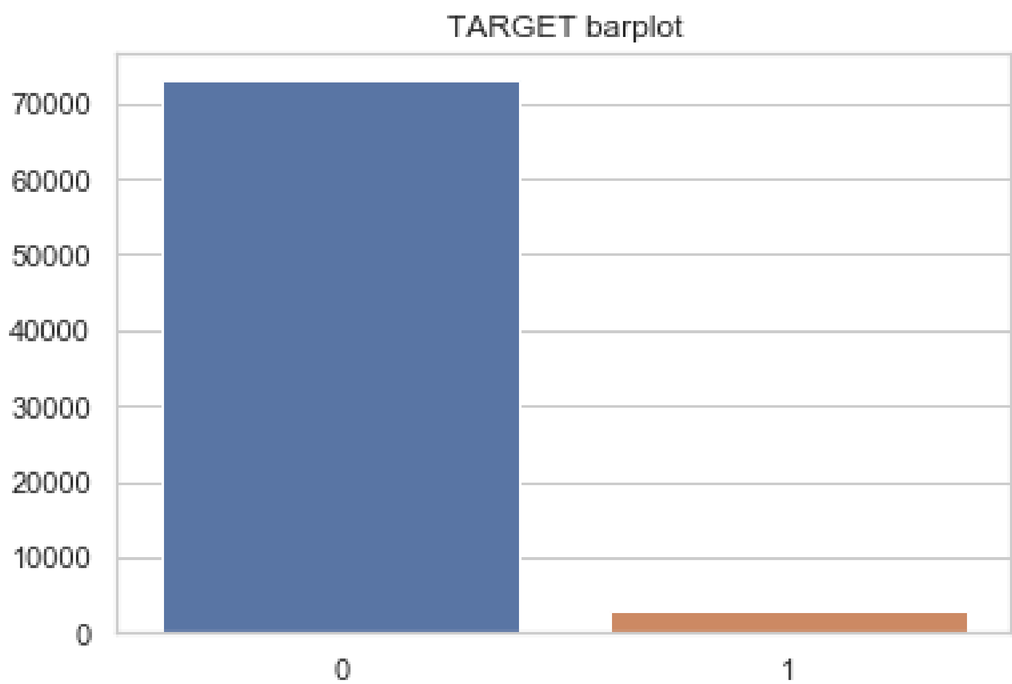
```
False
False
False
False
```

Checking customer satisfation variable distrbution and proportions.

[10]:
```
# TARGET values and proportion
df = pd.DataFrame(pd.Categorical(df_train.TARGET).describe())
display(df)

# TARGET variable barplot
sns.set(style="whitegrid")
sns.barplot(x=[0,1], y=df_train.TARGET.value_counts().values).set_title('TARGET␣
↪barplot')
```

	counts	freqs
categories		
0	73012	0.960431
1	3008	0.039569

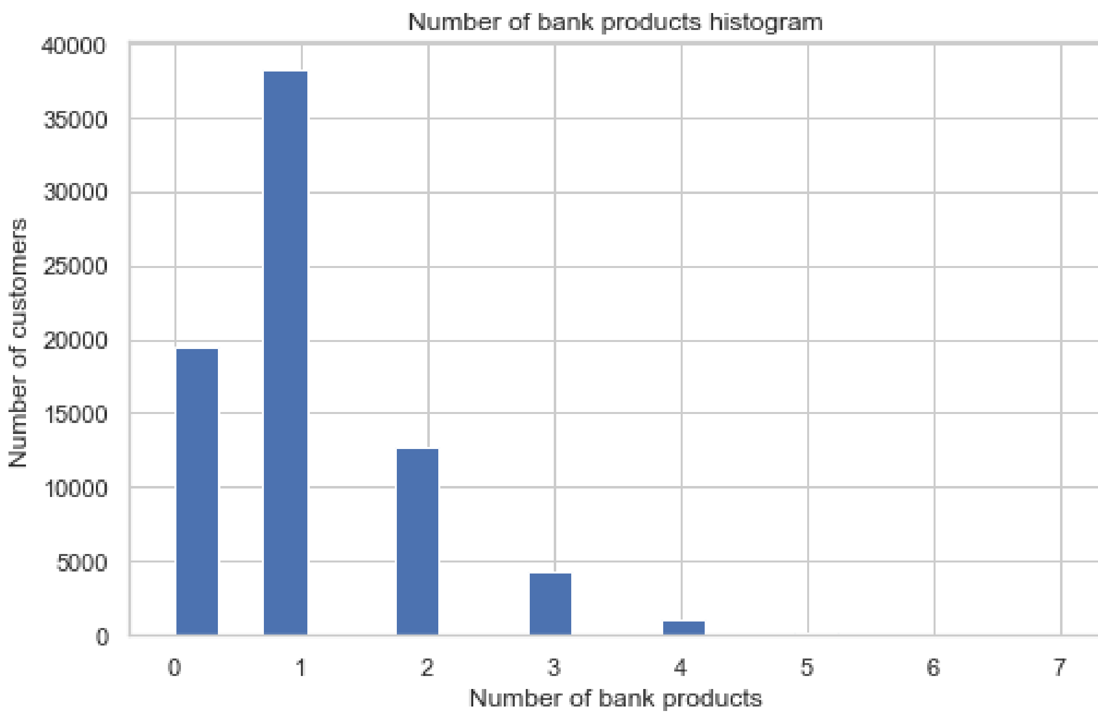
[10]:
```
Text(0.5, 1.0, 'TARGET barplot')
```



As noticed on table and graph above, customer satisfaction feature is unbalanced. Less than 4% of customers are unhappy with banking services and about 96% of customers are satisfied.

Number of bank products ("var4") "var4" is supposed to be the number of bank customers for each customer. So I analysed var4 distribution and relation to customers satisfaction.

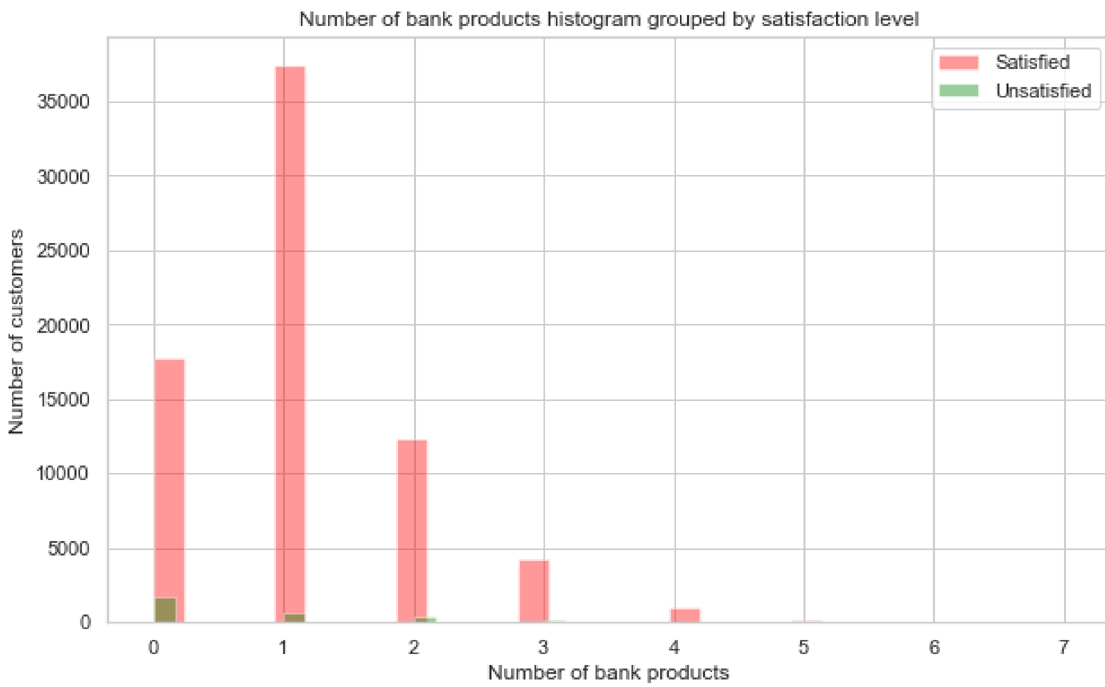
[11]:
```
# num_var4 distribution
plt.figure(figsize=(8,5))
df_train.num_var4.hist(bins=20)
plt.xlabel('Number of bank products')
plt.ylabel('Number of customers')
plt.title('Number of bank products histogram')
plt.show()
```

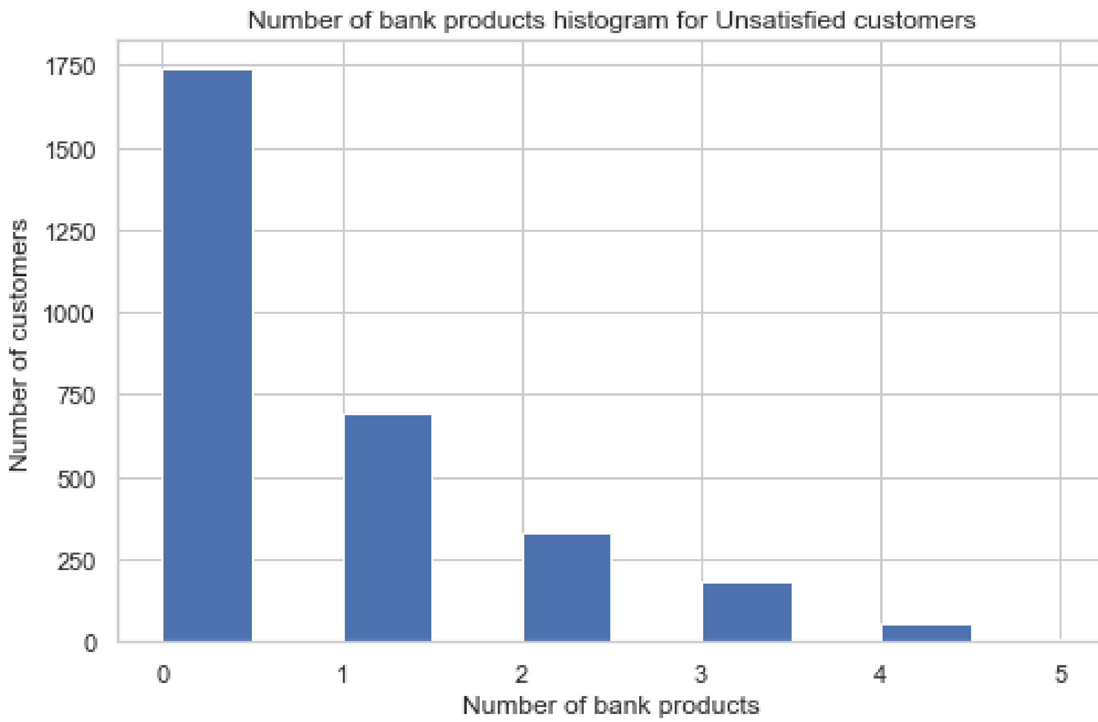


As noticed above, most customers have one banking product.

[12]:
```
# Checking number of bank products distribution grouped by satisfaction level
# Plotting number of bank products distribution grouped by satisfaction level
plt.figure(figsize=(10,6))
sns.distplot(df_train[df_train.TARGET == 0]['num_var4'], color='red',␣
↪kde=False, label='Satisfied', bins = 30)
sns.distplot(df_train[df_train.TARGET == 1]['num_var4'], color='green',␣
↪kde=False, label='Unsatisfied', bins = 30)
plt.legend()
plt.title('Number of bank products histogram grouped by satisfaction level')
plt.xlabel('Number of bank products')
plt.ylabel('Number of customers')
plt.show()
```

```
# Checking number of bank products distribution for unsatisfield customers
plt.figure(figsize=(8,5))
df_train[df_train.TARGET==1].num_var4.hist(bins=10)
plt.xlabel('Number of bank products')
plt.ylabel('Number of customers')
plt.title('Number of bank products histogram for Unsatisfied customers')
plt.show()
```





The graphs above showed that unsatisfied customers have less products.

1.2.1 Removing duplicated features

I eliminated the duplicate columns applying drop_duplicates function to pandas in the transposed dataframe.

[13]:
```
df_no_duplicate_train = df_train.T.drop_duplicates(keep='first').T
df_no_duplicate_test = df_test[list(df_no_duplicate_train.columns)]

# Saving dataset
df_no_duplicate_train.to_csv("data/df_no_duplicate_train.csv", index=False)
df_no_duplicate_test.to_csv("data/df_no_duplicate_test.csv", index=False)
```

1.2.2 Applying more filters to features

1.2.3 Removing constant and Quasi-Constant Features Using Variance Threshold

I removed constant and quasi-constant variables from my dataset using variance threshhold.

[23]:
```
# Removing quasi-constant columns
constant_filter = VarianceThreshold(threshold=0.01)
constant_filter.fit(df_no_duplicate_train.drop("TARGET", axis=1))

filtered_columns = [column for column in df_no_duplicate_train.drop("TARGET",␣
↪axis=1) \
```

```
                   .columns if column not in df_no_duplicate_train.
↪drop("TARGET", axis=1) \
                   .columns[constant_filter.get_support()]]


df_filtered_train = df_no_duplicate_train.drop(filtered_columns, axis=1)
df_filtered_test = df_no_duplicate_test.drop(filtered_columns, axis=1)

# Saving dataset
df_filtered_train.to_csv("data/df_filtered_train.csv", index=False)
df_filtered_test.to_csv("data/df_filtered_test.csv", index=False)
```

1.2.4 Selecting most important features

I applied SelectPercentile for selecting most importance variables. The input used functions were "chi-squared" and "ANOVA F-value".

[24]:
```
# features importance percentage
p = 6

X = df_filtered_train.drop("TARGET", axis=1)
y = df_filtered_train.TARGET

# Applying MinMaxScaler to scaled data to eliminate negative values
X_rescaled = MinMaxScaler().fit_transform(X)

selectChi2 = SelectPercentile(chi2, percentile=p).fit(X_rescaled, y)
selectF_classif = SelectPercentile(f_classif, percentile=p).fit(X, y)

selected = selectChi2.get_support() & selectF_classif.get_support()

features = [col for col, i in zip(X.columns, selected) if i]

print("{} most important features: ".format(len(features)), features)
```

```
11 most important features:  ['ind_var5', 'ind_var8_0', 'ind_var12_0',
'ind_var13_0', 'ind_var13', 'ind_var30', 'num_var5', 'num_var8_0', 'num_var42',
'var36', 'num_meses_var5_ult3']
```

[25]:
```
# Creating new train and test datasets contained selected features
df_train_selected = df_train[features+['TARGET']]
df_test_selected = df_test[features+['TARGET']]
df_train_selected.to_csv("data/df_train_selected.csv", index=False)
df_test_selected.to_csv("data/df_test_selected.csv", index=False)
```

1.2.5 Selected features analysis

Univariate Analysis Getting statistical measures

[26]:

```python
# describe for independents numerical features
df = df_train_selected.drop("TARGET", axis=1)

df_describe = pd.concat([df.describe().T,
              df.mad().rename('mad'),
              df.skew().rename('skew'),
              df.kurt().rename('kurt'),
              df.median().rename('median')
              ], axis=1).T

display(df_describe)
```

```
           ind_var5    ind_var8_0   ind_var12_0   ind_var13_0     ind_var13  \
count  76020.000000  76020.000000  76020.000000  76020.000000  76020.000000
mean       0.663760      0.032833      0.067522      0.052249      0.050855
std        0.472425      0.178202      0.250925      0.222531      0.219703
min        0.000000      0.000000      0.000000      0.000000      0.000000
25%        0.000000      0.000000      0.000000      0.000000      0.000000
50%        1.000000      0.000000      0.000000      0.000000      0.000000
75%        1.000000      0.000000      0.000000      0.000000      0.000000
max        1.000000      1.000000      1.000000      1.000000      1.000000
mad        0.446366      0.063511      0.125925      0.099039      0.096538
skew      -0.693290      5.243259      3.447163      4.024269      4.088762
kurt      -1.519389     25.492434      9.883193     14.195115     14.718362
median     1.000000      0.000000      0.000000      0.000000      0.000000

          ind_var30      num_var5   num_var8_0     num_var42         var36  \
count  76020.000000  76020.000000  76020.000000  76020.000000  76020.000000
mean       0.732833      1.999171      0.098540      2.217995     40.449079
std        0.442483      1.431902      0.534930      1.497703     47.362719
min        0.000000      0.000000      0.000000      0.000000      0.000000
25%        0.000000      0.000000      0.000000      0.000000      2.000000
50%        1.000000      3.000000      0.000000      3.000000      3.000000
75%        1.000000      3.000000      0.000000      3.000000     99.000000
max        1.000000     15.000000      6.000000     18.000000     99.000000
mad        0.391577      1.344405      0.190609      1.278396     46.310836
skew      -1.052423     -0.620356      5.249057     -0.339927      0.426879
kurt      -0.892430     -1.300280     25.605241     -0.011351     -1.816896
median     1.000000      3.000000      0.000000      3.000000      3.000000

        num_meses_var5_ult3
count          76020.000000
mean               1.979979
std                1.298924
min                0.000000
25%                0.000000
50%                3.000000
```

```
75%                3.000000
max                3.000000
mad                1.154518
skew              -0.702865
kurt              -1.312572
median             3.000000
```

[73]:
```
# Checking selected variable types
display(df_train_selected.dtypes)

# Counting features values
display(df_train_selected.apply(lambda x: x.value_counts(), axis=0))
```

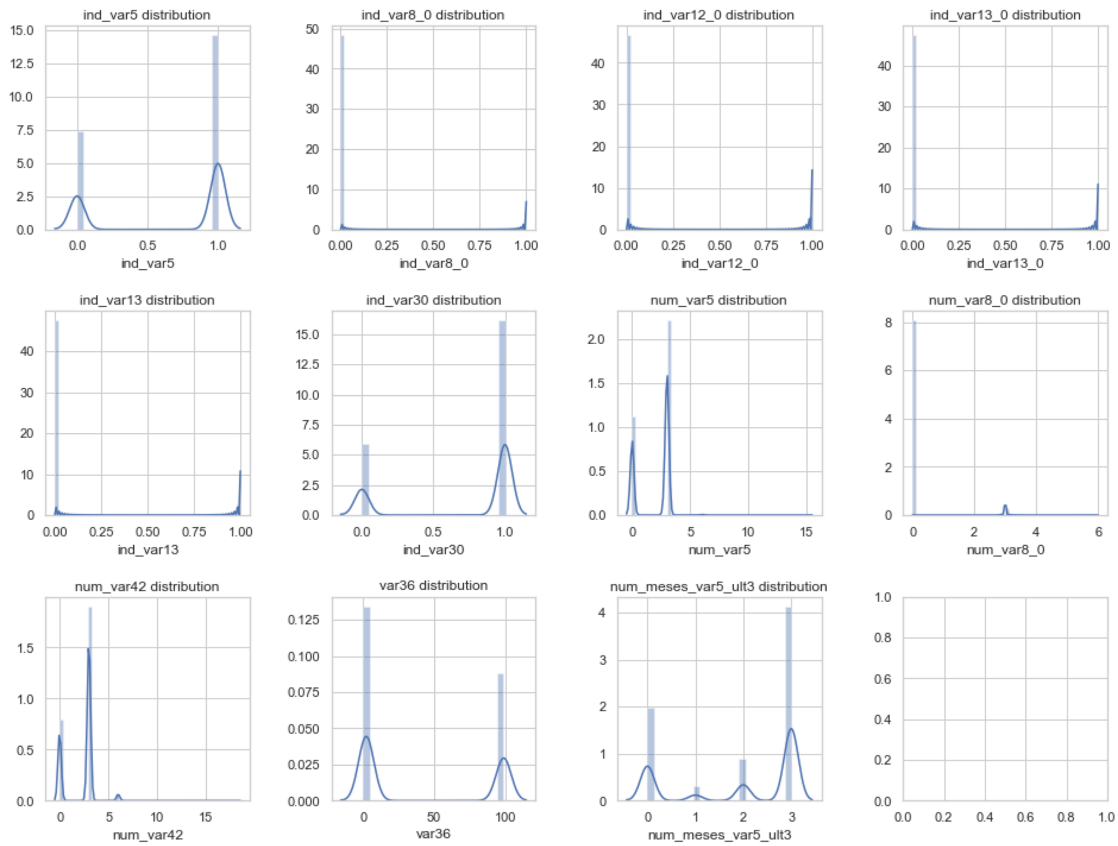
```
ind_var5                int64
ind_var8_0              int64
ind_var12_0             int64
ind_var13_0             int64
ind_var13               int64
ind_var30               int64
num_var5                int64
num_var8_0              int64
num_var42               int64
var36                   int64
num_meses_var5_ult3     int64
TARGET                  int64
dtype: object
```

```
    ind_var5  ind_var8_0  ind_var12_0  ind_var13_0  ind_var13  ind_var30  \
0    25561.0     73524.0      70887.0      72048.0    72154.0    20310.0
1    50459.0      2496.0       5133.0       3972.0     3866.0    55710.0
2        NaN         NaN          NaN          NaN        NaN        NaN
3        NaN         NaN          NaN          NaN        NaN        NaN
6        NaN         NaN          NaN          NaN        NaN        NaN
9        NaN         NaN          NaN          NaN        NaN        NaN
12       NaN         NaN          NaN          NaN        NaN        NaN
15       NaN         NaN          NaN          NaN        NaN        NaN
18       NaN         NaN          NaN          NaN        NaN        NaN
99       NaN         NaN          NaN          NaN        NaN        NaN

    num_var5  num_var8_0  num_var42    var36  num_meses_var5_ult3   TARGET
0    25561.0     73524.0    21908.0    411.0              20546.0  73012.0
1        NaN         NaN        NaN  14664.0               3268.0   3008.0
2        NaN         NaN        NaN   8704.0               9368.0      NaN
3    50265.0      2495.0    52064.0  22177.0              42838.0      NaN
6      190.0         1.0     2012.0      NaN                  NaN      NaN
9        3.0         NaN       31.0      NaN                  NaN      NaN
```

12	NaN	NaN	3.0	NaN	NaN	NaN
15	1.0	NaN	1.0	NaN	NaN	NaN
18	NaN	NaN	1.0	NaN	NaN	NaN
99	NaN	NaN	NaN	30064.0	NaN	NaN

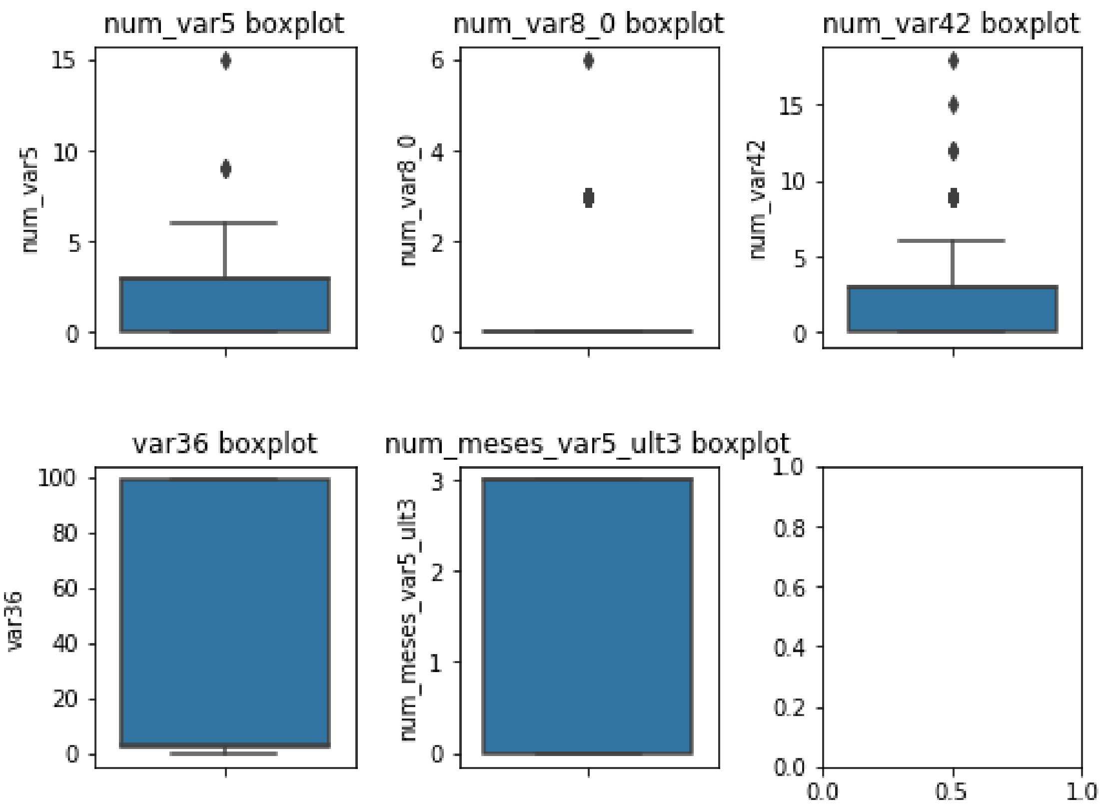
Table above show that there are 6 binary independent features. ###### Checking univariate features distributions

[27]:
```
# Features histograms
df = df_train_selected.drop("TARGET", axis=1)
fig, axs = plt.subplots(ncols=4, nrows=3)
plt.subplots_adjust(hspace = 0.4, wspace = 0.4)
fig.set_size_inches(16, 12, forward=True)

count = 0

for i in range(3):
    for j in range(4):
        plt.sca(ax=axs[i][j])
        if count < df.shape[1]:
            col = df.columns[count]
            sns.distplot(df[col], hue = ).set_title(col +' distribution')
        else:
            break

        count +=1
```



[92]:

```
# Features boxplots
df = df_train_selected[["num_var5", "num_var8_0", "num_var42", "var36", "num_meses_var5_ult3"]]
fig, axs = plt.subplots(ncols=3, nrows=2)
plt.subplots_adjust(hspace = 0.4, wspace = 0.4)
fig.set_size_inches(8, 6, forward=True)

count = 0

for i in range(2):
    for j in range(3):
        plt.sca(ax=axs[i][j])
        if count < df.shape[1]:
            col = df.columns[count]
            sns.boxplot(y=df[col]).set_title(col +' boxplot')
        else:
            break

        count +=1
```

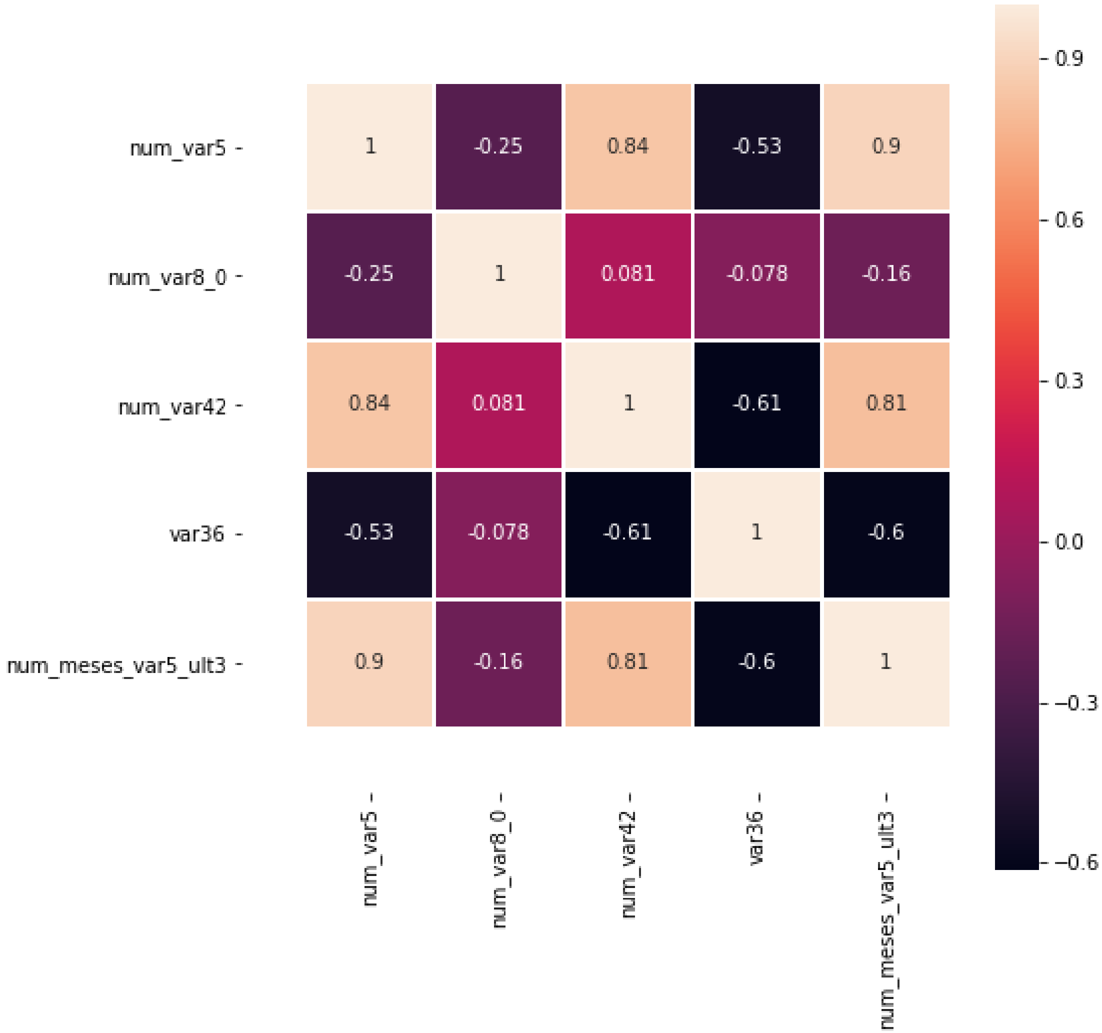


[2]:
```
df_train_selected = pd.read_csv("data/df_train_selected.csv")
df_test_selected = pd.read_csv("data/df_test_selected.csv")
```

Bivariate Analysis Checking correlation between numerical variables

[85]:
```
# heat map of correlation values
corr = df_train_selected[["num_var5", "num_var8_0", "num_var42", "var36",␣
↪"num_meses_var5_ult3"]].corr()
fig, ax = plt.subplots(figsize=(8,8))

g = sns.heatmap(corr, annot=True, ax=ax, square=True, linewidth=0.5)

plt.yticks(rotation=0)
g.set_xticklabels(g.get_xticklabels(), rotation=90)
ax.set_ylim([len(corr) + 0.5, 0])
ax.set_xlim([-0.5, len(corr)])
```

[85]:
```
(-0.5, 5)
```



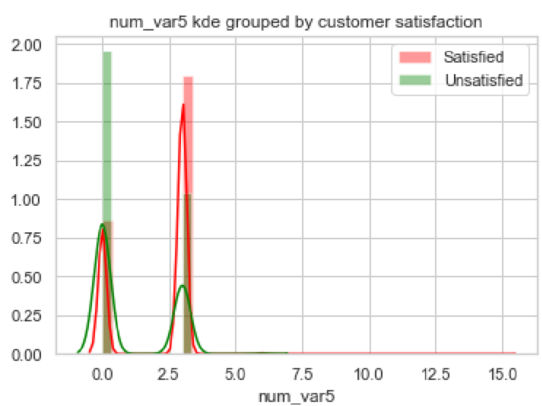
Print high correlated variables

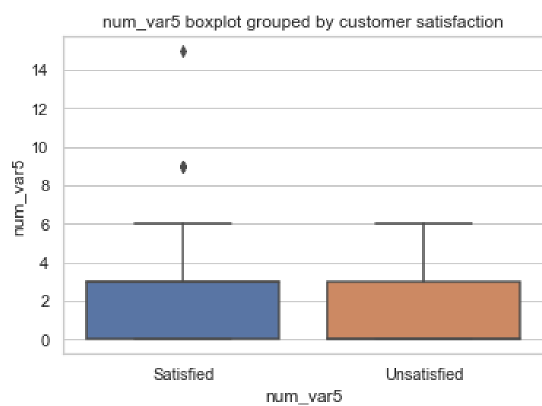
[93]:
```
# Print high correlated variables
corr = df_train_selected[["num_var5", "num_var8_0", "num_var42", "var36", \
                                    "num_meses_var5_ult3", "TARGET"]].corr()

n = len(corr)

[(corr.index[i], corr.columns[j]) for i in range(n-1) for j in range(i+1,n) if␣
 ↪abs(corr.iloc[i][j]) >= 0.8]
```

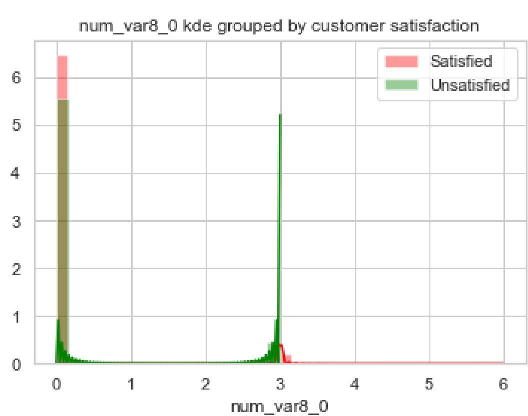
[93]:
```
[('num_var5', 'num_var42'),
 ('num_var5', 'num_meses_var5_ult3'),
 ('num_var42', 'num_meses_var5_ult3')]
```

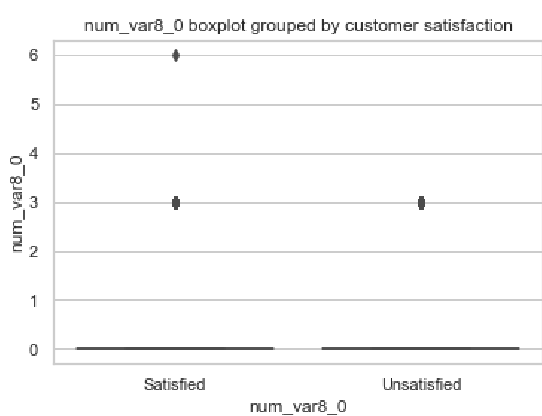
Getting numerical features distribuition grouped by target variable

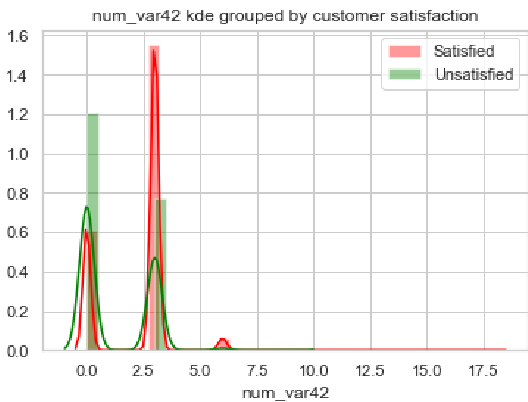
[24]:
```
# Getting numerical features distribuition grouped by target variable
df = df_train_selected[["num_var5", "num_var8_0", "num_var42", "var36", \
                        "num_meses_var5_ult3", "TARGET"]]

for col in df.drop("TARGET", axis=1).columns:
    fig, axs = plt.subplots(ncols=2)
    fig.set_size_inches(13, 4, forward=True)
    sns.distplot(df[df.TARGET == 0][col], color='red', label='Satisfied',␣
 ↪ax=axs[0], bins = 40)
    sns.distplot(df[df.TARGET == 1][col], color='green', label='Unsatisfied',␣
 ↪ax=axs[0], bins = 18)
    axs[0].legend()
    axs[0].set_xlabel(col)
    axs[0].set_title(col + ' kde grouped by customer satisfaction')
    sns.boxplot(y=col, x="TARGET", data = df, ax=axs[1])
    axs[1].set_xlabel(col)
    axs[1].set_xticklabels(['Satisfied', 'Unsatisfied'])
    axs[1].set_title(col + ' boxplot grouped by customer satisfaction')
    plt.show()
```

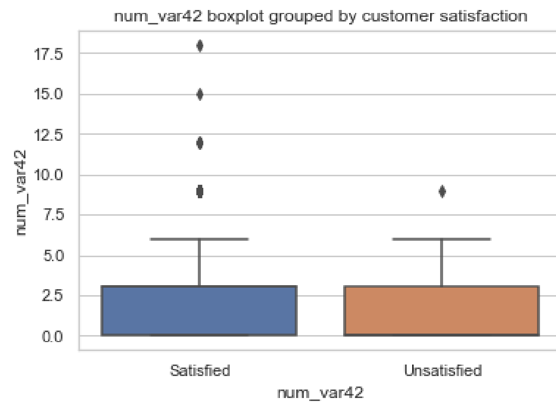


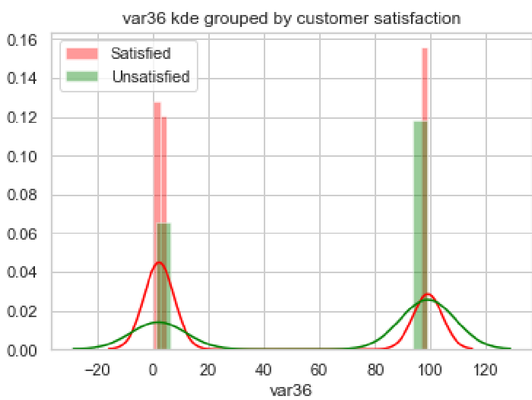


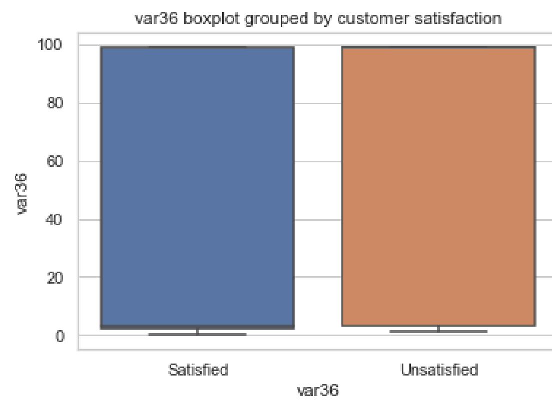


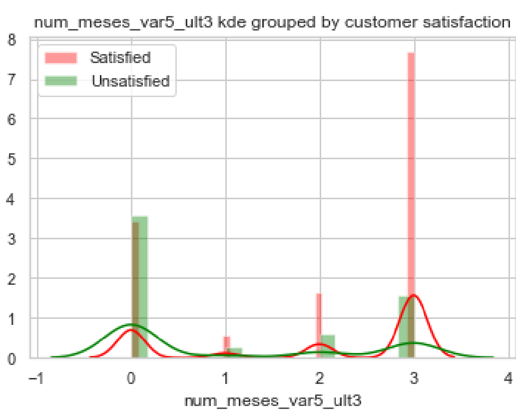



num_var42 kde grouped by customer satisfaction
Satisfied
Unsatisfied
num_var42


num_var42 boxplot grouped by customer satisfaction
num_var42
Satisfied
Unsatisfied
num_var42


var36 kde grouped by customer satisfaction
Satisfied
Unsatisfied
var36


var36 boxplot grouped by customer satisfaction
var36
Satisfied
Unsatisfied
var36

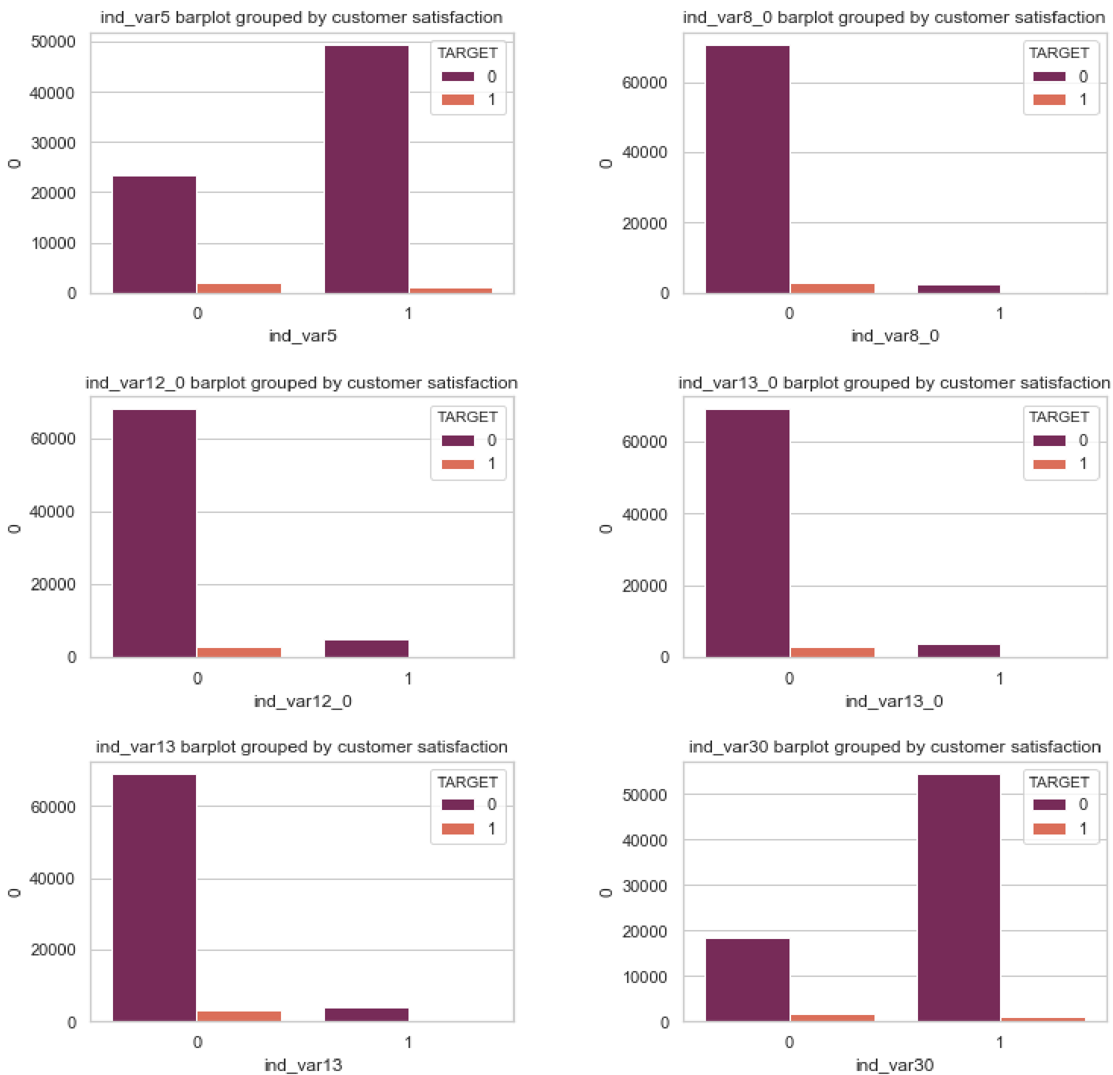

num_meses_var5_ult3 kde grouped by customer satisfaction
Satisfied
Unsatisfied
num_meses_var5_ult3


num_meses_var5_ult3 boxplot grouped by customer satisfaction
num_meses_var5_ult3
Satisfied
Unsatisfied
num_meses_var5_ult3

Looking at the graph, some patterns can be noticed: - **num_var5**: value 3 presents more satisfied customers than unsatisfied customers. - **num_var42**: the value zero concentrates more unsatisfied customers and the value 3 presents a greater number of satisfied customers. - **var36**: the zero value has a higher proportion of satisfied customers. - **num_meses_var5_ult3**: the higher the value, the greater the proportion of satisfied customers. The value 3 has the highest proportion.

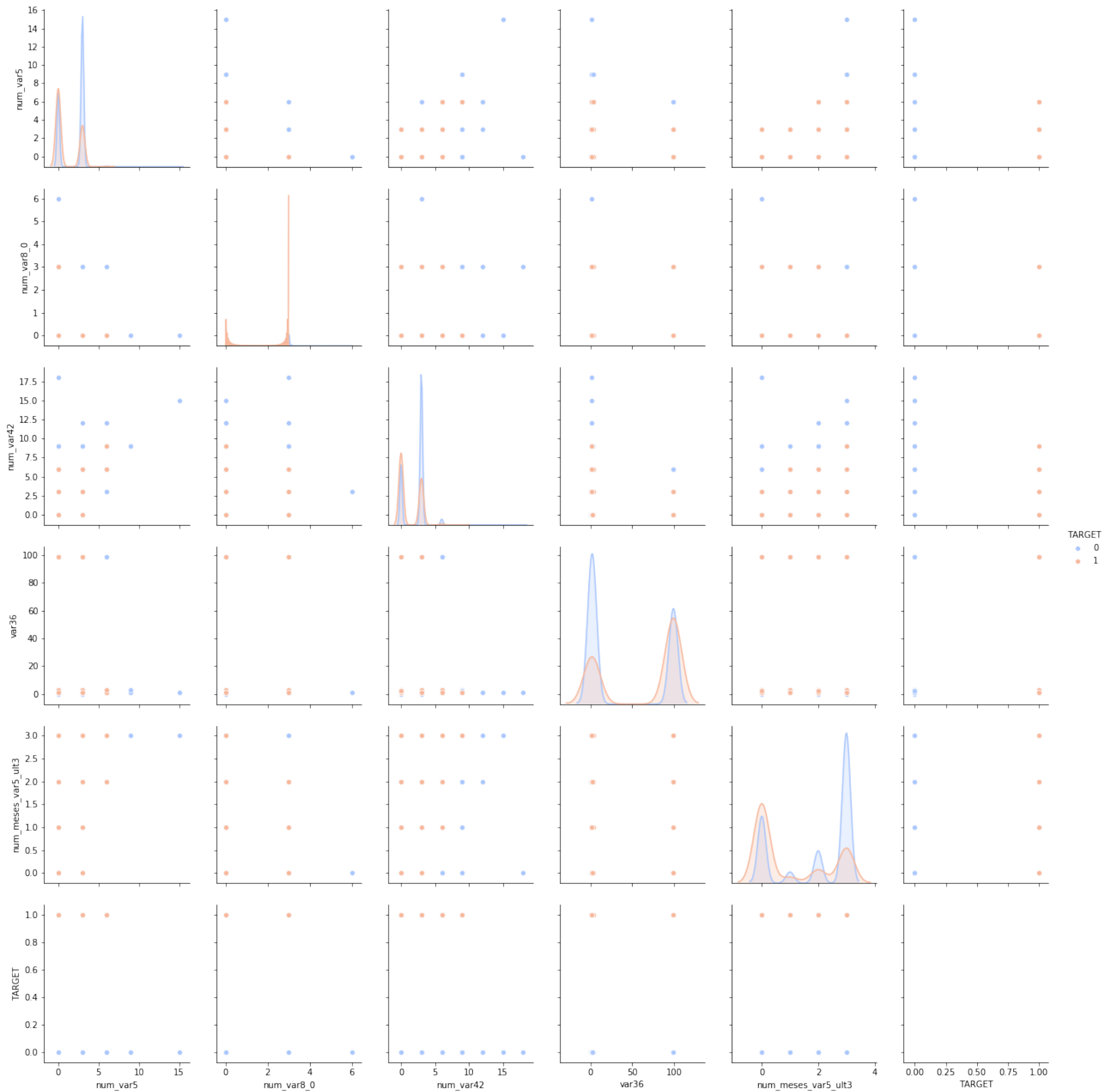
Getting categorical features barplot grouped by target variable

[72]:
```
# Getting categorical features barplot grouped by target variable
df = df_train_selected[["ind_var5", "ind_var8_0", "ind_var12_0", "ind_var13_0",␣
↪\
                        "ind_var13", "ind_var30", "TARGET"]]

fig, axs = plt.subplots(ncols=2, nrows=3)
plt.subplots_adjust(hspace = 0.4, wspace = 0.4)
fig.set_size_inches(12, 12, forward=True)

count = 0

for i in range(3):
    for j in range(2):
        plt.sca(ax=axs[i][j])
        if count < df.shape[1]:
            col = df.columns[count]
            df_g = pd.DataFrame(df.groupby([col, 'TARGET']).size()).
↪reset_index()
            sns.barplot(x=col, y=0, hue="TARGET", data=df_g, palette="rocket")
            axs[i][j].set_xlabel(col)
            axs[i][j].set_title(col + ' barplot grouped by customer␣
↪satisfaction')
        else:
            break

        count +=1
```



In all selected categorical variables, the number of satisfied customers is significantly higher than that of dissatisfied customers for all variable values. #### Multivariate Analysis

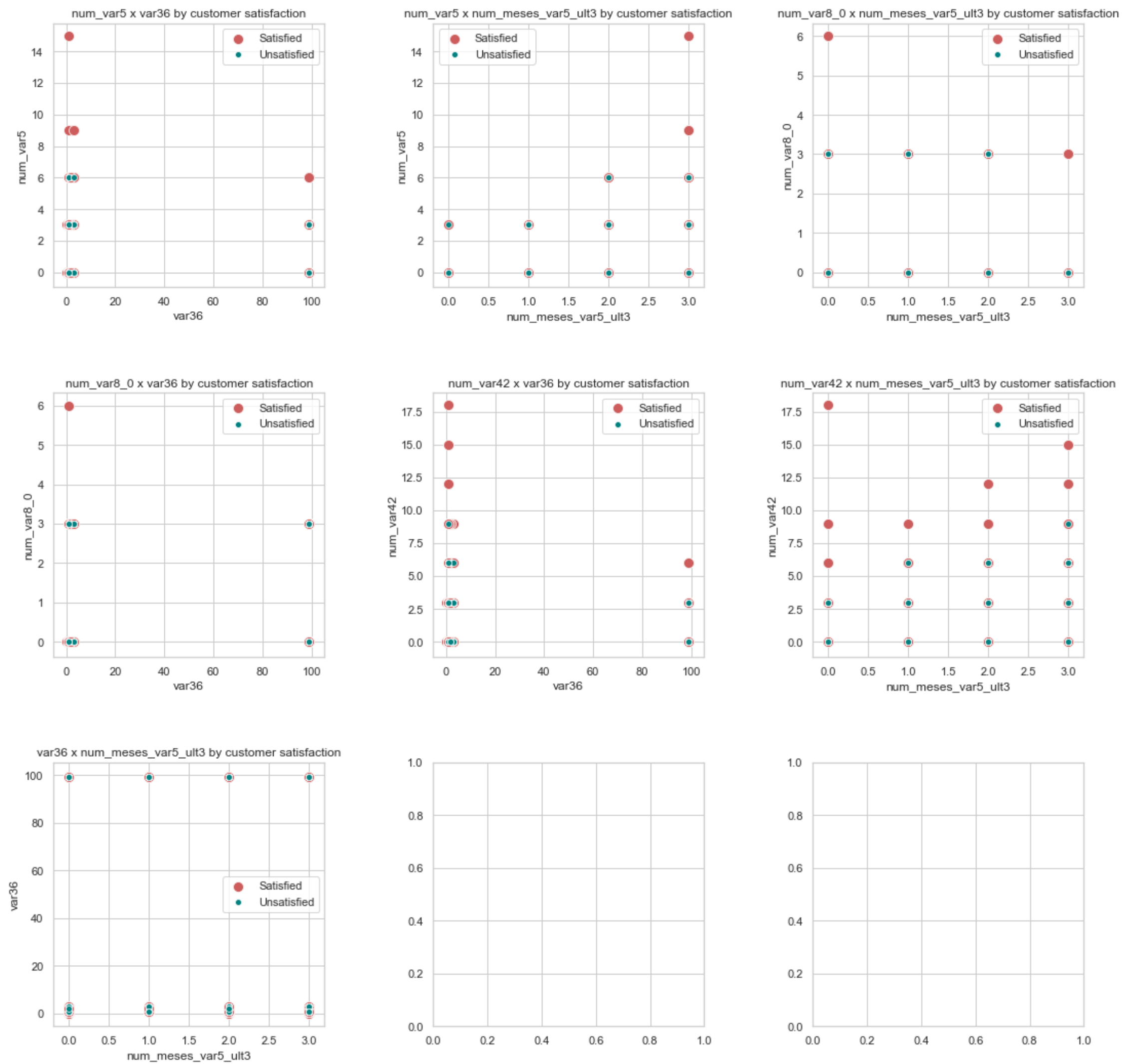
[88]:
```
# pairplot matrix groupedby TARGET variable
g = sns.pairplot(df_train_selected[["num_var5", "num_var8_0", "num_var42",␣
↪"var36", \
                                    "num_meses_var5_ult3", "TARGET"]],␣
↪hue='TARGET', palette='coolwarm')
g.fig.set_size_inches(18,18)
```



I plotted scatterplots grouped by customer satisfaction for some features to check for pattenrs.

[100]:
```
# scatter plots for pairs of features grouped by target variable

df = df_train_selected[["num_var5", "num_var8_0", "num_var42", "var36", \
                        "num_meses_var5_ult3", "TARGET"]]

pair_list = [("num_var5", "var36"), ("num_var5", "num_meses_var5_ult3"),␣
 ↪("num_var8_0", "num_meses_var5_ult3"), \
             ("num_var8_0", "var36"), ("num_var42", "var36"), ("num_var42",␣
 ↪"num_meses_var5_ult3"), \
             ("var36", "num_meses_var5_ult3")]

fig, axs = plt.subplots(ncols=3, nrows=3)
```

```
plt.subplots_adjust(hspace = 0.4, wspace = 0.4)
fig.set_size_inches(18, 18, forward=True)

count = 0

for i in range(3):
    for j in range(3):
        plt.sca(ax=axs[i][j])
        if count < len(pair_list):
            x = pair_list[count][1]
            y = pair_list[count][0]
            sns.scatterplot(x=x, y=y,  data=df[df.TARGET==0],␣
↪color='indianred', label = "Satisfied", s=100)
            sns.scatterplot(x=x, y=y,  data=df[df.TARGET==1], color='teal',␣
↪label = "Unsatisfied")
            axs[i][j].set_xlabel(x)
            axs[i][j].set_title(y + ' x ' + x + ' by customer satisfaction')
        else:
            break

        count +=1
```



As noted in the graphs, high values for num_var5, num_var8_0 and num_var42 are characteristic of satisfied customers. ### Removing Highly Correlated Features

I removed the highly correlated independent features. I created a correlation matrix for filter low-correlated features.

[108]:
```
# Removing highly correlated features
correlation_matrix = df_train_selected[["num_var5", "num_var8_0", "num_var42",␣
↪"var36", \
                                        "num_meses_var5_ult3"]].corr()
correlated_features = set()

[correlated_features.add(rowname) for rowname in correlation_matrix.columns for␣
↪colname in correlation_matrix.columns if \
correlation_matrix.loc[rowname][colname] > 0.8 and rowname != colname]

correlated_features
```

```python
df_less_corr_train = df_train_selected.drop(labels=correlated_features, axis=1)
df_less_corr_test = df_test_selected.drop(labels=correlated_features, axis=1)

# Saving data
df_less_corr_train.to_csv("data/df_less_corr_train.csv", index=False)
df_less_corr_test.to_csv("data/df_less_corr_test.csv", index=False)
```

1.2.6 Changing numerical data scale

[127]:
```python
# Applying MinMaxScaler to scaled data to eliminate negative values
# Train data
df = df_less_corr_train[["num_var8_0", "var36"]]

MinMaxData = pd.DataFrame(MinMaxScaler().fit_transform(df), \
                          columns = df.columns)

minmax_train_data = df_less_corr_train
minmax_train_data[["num_var8_0", "var36"]] = MinMaxData

# Test data
df = df_less_corr_test[["num_var8_0", "var36"]]

MinMaxData = pd.DataFrame(MinMaxScaler().fit_transform(df), \
                          columns = df.columns)

minmax_test_data = df_less_corr_test
minmax_test_data[["num_var8_0", "var36"]] = MinMaxData

# Saving data
minmax_train_data.to_csv("data/minmax_train_data.csv", index=False)
minmax_test_data.to_csv("data/minmax_test_data.csv", index=False)
```

[44]:
```python
# Applying StandardData to scaled data to eliminate negative values
# Train data
df = df_less_corr_train.drop("TARGET", axis=1)

StandardData = pd.DataFrame(StandardScaler().fit_transform(df), \
                          columns = df.columns)

stand_train_data = StandardData
stand_train_data['TARGET'] = df_less_corr_train.TARGET

# Test data
df = df_less_corr_test.drop("TARGET", axis=1)

StandardData = pd.DataFrame(StandardScaler().fit_transform(df), \
```

```
                                columns = df.columns)

stand_test_data = StandardData
stand_test_data['TARGET'] = df_less_corr_test.TARGET

# Saving data
stand_train_data.to_csv("data/stand_train_data.csv", index=False)
stand_test_data.to_csv("data/stand_test_data.csv", index=False)
```

[45]:
```
df_less_corr_train = pd.read_csv("data/df_less_corr_train.csv")
df_less_corr_test = pd.read_csv("data/df_less_corr_test.csv")
minmax_train_data = pd.read_csv("data/minmax_train_data.csv")
minmax_test_data = pd.read_csv("data/minmax_test_data.csv")
stand_train_data = pd.read_csv("data/stand_train_data.csv")
stand_test_data = pd.read_csv("data/stand_test_data.csv")
```

1.2.7 Dealing with unbalanced data and Spliting training data into training and testing

I chose to do a simple resampling without replacement, as my machine does not process the UnderSampling.fit_resample function for larger data and I chose to train with a larger amount of data. I did the resampling, followed by the division of the train data in training and test.

[33]:
```
# Undersamplig data (in original scale)
random.seed(10000)

df1 = df_less_corr_train[df_less_corr_train.TARGET == 0].sample(n = 4512)
df2 = df_less_corr_train[df_less_corr_train.TARGET == 1]

df = pd.concat([df1, df2]).reset_index(drop=True)

# Defining data
features = df.drop('TARGET', axis=1)
targets = df.TARGET

Xo, Xo_test, yo, yo_test = features, df_less_corr_test.drop("TARGET", axis=1),␣
 ↪targets, df_less_corr_test.TARGET
```

[26]:
```
# Undersamplig data (in MinMaxScale scale)
random.seed(10000)

df1 = minmax_train_data[minmax_train_data.TARGET == 0].sample(n = 4512)
df2 = minmax_train_data[minmax_train_data.TARGET == 1]

df = pd.concat([df1, df2]).reset_index(drop=True)

# Defining data
features = df.drop('TARGET', axis=1)
targets = df.TARGET
```

```
Xmn, Xmn_test, ymn, ymn_test = features, minmax_test_data.drop("TARGET",␣
↪axis=1), targets, minmax_test_data.TARGET
```

[46]:
```
# Undersamplig data (in StandardScale scale)
random.seed(10000)

df1 = stand_train_data[stand_train_data.TARGET == 0].sample(n = 4512)
df2 = stand_train_data[stand_train_data.TARGET == 1]

df = pd.concat([df1, df2]).reset_index(drop=True)

# Defining data
features = df.drop('TARGET', axis=1)
targets = df.TARGET

Xsd, Xsd_test, ysd, ysd_test = features, stand_test_data.drop("TARGET",␣
↪axis=1), targets, stand_test_data.TARGET
```

1.3 Training models

To predict customer satisfaction, I chose to test four algorithms and choose the one with the best performance: Logistic Regression, K-Nearest Neighbor, Support Vector Machines and Random Forest. ### K-Nearest Neighbours Classifier

[37]:
```
# KNN algorithm
knn = KNeighborsClassifier(n_neighbors=34)

# training model
knn.fit(Xmn, ymn)

# prediction
pred = knn.predict(Xmn_test)

# Evaluating prediction
print (confusion_matrix(ymn_test,pred))
print (classification_report(ymn_test,pred))
print(accuracy_score(ymn_test,pred))

# Save the model as a pickle in a file
joblib.dump(knn, 'knn.pkl')
```

```
[[74381  1437]
 [    0     0]]
              precision    recall  f1-score   support

           0       1.00      0.98      0.99     75818
           1       0.00      0.00      0.00         0
```

```
    accuracy                           0.98     75818
   macro avg       0.50      0.49      0.50     75818
weighted avg       1.00      0.98      0.99     75818

0.9810467171384104
```

1.3.1 Support Vector Machine

[47]:
```
# svm algorithm
svm = SVC(gamma='auto')

# training model
svm.fit(Xsd, ysd)

# prediction
pred = svm.predict(Xsd_test)

# Evaluating prediction
print (confusion_matrix(ysd_test,pred))
print (classification_report(ysd_test,pred))

# Save the model as a pickle in a file
joblib.dump(svm, 'svm.pkl')
```

```
[[54914 20904]
 [    0     0]]
              precision    recall  f1-score   support

           0       1.00      0.72      0.84     75818
           1       0.00      0.00      0.00         0

    accuracy                           0.72     75818
   macro avg       0.50      0.36      0.42     75818
weighted avg       1.00      0.72      0.84     75818
```

[47]:
```
['svm.pkl']
```

1.3.2 Logistic Regression

[48]:
```
# LogisticRegression algorithm
lr = LogisticRegression(C=1e5)

# training model
lr.fit(Xsd, ysd)
```

```
# prediction
pred = lr.predict(Xsd_test)

# Evaluating prediction
print (confusion_matrix(ysd_test,pred))
print (classification_report(ysd_test,pred))

# Save the model as a pickle in a file
joblib.dump(lr, 'lr.pkl')
```

```
[[53809 22009]
 [    0     0]]
              precision    recall  f1-score   support

           0       1.00      0.71      0.83     75818
           1       0.00      0.00      0.00         0

    accuracy                           0.71     75818
   macro avg       0.50      0.35      0.42     75818
weighted avg       1.00      0.71      0.83     75818
```

[48]: ['lr.pkl']

1.3.3 Random Forest

[39]:
```
# Number of base decision tree estimators
n_est = 100

# Maximum depth of any decision tree estimator
max_depth = 5

# Random state variable
rstate = 42

# RandomForest algorithm
rf = RandomForestClassifier(n_estimators=n_est,
                             max_depth=max_depth,
                             random_state=rstate)

# training model
rf.fit(Xo, yo)

# prediction
pred = rf.predict(Xo_test)

# Evaluating prediction
```

```
print (confusion_matrix(yo_test,pred))
print (classification_report(yo_test,pred))

# Save the model as a pickle in a file
joblib.dump(rf, 'rf.pkl')
```

```
[[54901 20917]
 [    0     0]]
              precision    recall  f1-score   support

           0       1.00      0.72      0.84     75818
           1       0.00      0.00      0.00         0

    accuracy                           0.72     75818
   macro avg       0.50      0.36      0.42     75818
weighted avg       1.00      0.72      0.84     75818
```

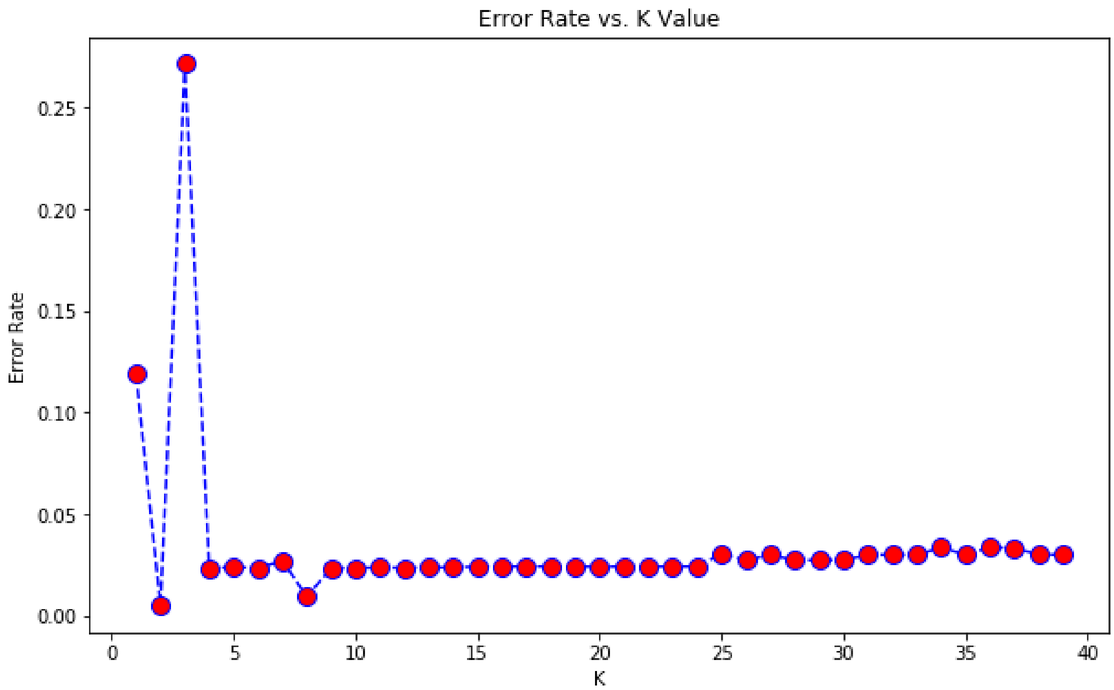
[39]:
```
['rf.pkl']
```

1.4 Trying to optimizate model

As Knn presented the best result, I chose to try to optimize it. ### K-value Optimization

[49]:
```
# Load the model from the file
# knn_from_joblib = joblib.load('filename.pkl')
# Testing values from original scaled data

error_rate = []

for i in range(1,40):

    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(Xo, yo)
    pred_i = knn.predict(Xo_test)
    error_rate.append(np.mean(pred_i != yo_test))

plt.figure(figsize=(10,6))
plt.plot(range(1,40),error_rate,color='blue', linestyle='dashed', marker='o',
         markerfacecolor='red', markersize=10)
plt.title('Error Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')
```

[49]:
```
Text(0, 0.5, 'Error Rate')
```



[51]:
```
for i in range(len(error_rate)):
    if error_rate[i] == min(error_rate):
        print(i)
```

```
1
```

[81]:
```
# KNN algorithm
knn = KNeighborsClassifier(n_neighbors=2)

# training model
knn.fit(Xo, yo)

# prediction
pred = knn.predict(Xo_test)

# Evaluating prediction
print (confusion_matrix(yo_test,pred))
print (classification_report(yo_test,pred))
print(accuracy_score(yo_test,pred))
```

```
[[75422   396]
 [    0     0]]
              precision    recall  f1-score   support
```

```
           0       1.00      0.99      1.00     75818
           1       0.00      0.00      0.00         0

    accuracy                           0.99     75818
   macro avg       0.50      0.50      0.50     75818
weighted avg       1.00      0.99      1.00     75818

0.9947769658920045
```

1.5 Final model version

The final version has 99% accuracy

[76]:
```python
# Testing values from standard scaled data
# KNN algorithm
knn = KNeighborsClassifier(n_neighbors=2)

# training model
knn.fit(Xsd, ysd)

# prediction
pred = knn.predict(Xsd_test)

# Evaluating prediction
print (confusion_matrix(ysd_test,pred))
print (classification_report(ysd_test,pred))
print(accuracy_score(ysd_test,pred))
```

```
[[75532   286]
 [    0     0]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     75818
           1       0.00      0.00      0.00         0

    accuracy                           1.00     75818
   macro avg       0.50      0.50      0.50     75818
weighted avg       1.00      1.00      1.00     75818

0.996227808699781
```

[82]:
```python
# Save the model as a pickle in a file
joblib.dump(knn, 'knn.pkl')

# Load the model from the file
# knn = joblib.load('knn.pkl')
```

1.6 Submission table

[91]:
```
# Importing test data
df_result_test = pd.read_csv("data/sample_submission.csv")
pred = knn.predict(Xsd_test)
df_result_test['PREDICTION'] = pred
display(df_result_test)
```

```
           ID  TARGET  PREDICTION
0           2       0           0
1           5       0           0
2           6       0           0
3           7       0           0
4           9       0           0
...       ...     ...         ...
75813  151831       0           0
75814  151832       0           0
75815  151833       0           0
75816  151834       0           0
75817  151837       0           0

[75818 rows x 3 columns]
```