P02_Modeling

Recap for Categorical features

- x3 is nominal categorical feature (needs OneHotEncoder transformer) and custom function edit is needed
- x7 column: the % sign should be cleaned from strings and I need to change the dtype and introduce it as numeric feature.
- x19 column: the % sign should be cleaned from strings and I need to change the dtype and introduce it as numeric feature.
- x24 column: is nominal category and needs to be transformed using One Hot Encoding
- x31 column: is nominal category and needs to be transformed using One Hot Encoding
- x33 column: is nominal category and needs to be transformed using One Hot Encoding
- x39 should be droped the column
- x60 is nominal and OneHotEncoder should be used to transfer this column
- x65 is nominal and OneHotEncoder should be used to transfer this column
- x77 is nominal and OneHotEncoder should be used to transfer this column
- x93 is nominal and OneHotEncoder should be used to transfer this column
- X99 should be droped - it has just one category.
- The x24, x33, and x77 columns NaN values needs to be replaced by their mode.

Recap for Numerical features

- Column 'x30', 'x44', 'x57' have more than 50% nan and should be droped
- Three features inside the numeric df need to be transformed using One Hot Encoding. These are features with 0-1 values and should be treated like category features. These are 0-1 features and should be treated like category features. columns are x59 and x79 and x98
- I may use Isolation Forest or LOF for outlier detection. I will check the AUC to see the differences.
- cols with high skewness are `['x13', 'x21', 'x32', 'x35', 'x44', 'x59', 'x67', 'x73', 'x75', 'x79', 'x84', 'x89']` . I will apply power transformer.
- High score numerical features are as follow: `['x16', 'x18', 'x28', 'x32', 'x35', 'x40', 'x47', 'x52', 'x57', 'x62', 'x68', 'x70', 'x75', 'x78', 'x81', 'x88', 'x89', 'x95', 'x96']`
- With n_components=12, I get 0.62 AUC and as it is seen 12 principal components have 62% of variance. I may apply PCA if I found not using it computationally time expensive.

import libraries

In [1]:
```
import pandas as pd
import numpy as np
import regex as re
import warnings
warnings.filterwarnings('ignore')
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
from nltk.stem import WordNetLemmatizer
import pickle

import matplotlib.pyplot as plt
import re
```

In [2]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import VarianceThreshold
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import Normalizer

from sklearn.preprocessing import MaxAbsScaler
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import PowerTransformer
```

In [3]:
```
from sklearn.neighbors import LocalOutlierFactor
from sklearn.preprocessing import MinMaxScaler
```

In [4]:
```
from sklearn.decomposition import PCA
from sklearn.model_selection import RepeatedStratifiedKFold
```

In [5]:
```
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_classif
from matplotlib import pyplot
```

In [6]:
```
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import make_pipeline
from sklearn.compose import ColumnTransformer
```

In [7]:
```
from sklearn.covariance import EllipticEnvelope
from sklearn.ensemble import IsolationForest
from sklearn.preprocessing import FunctionTransformer
```

In [8]:
```
from numpy import mean
from numpy import std
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.pipeline import Pipeline
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression
from matplotlib import pyplot
```

In [9]:
```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_classif
from sklearn.pipeline import Pipeline
from sklearn.model_selection import RepeatedStratifiedKFold
from matplotlib import pyplot
```

In [10]:
```
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OrdinalEncoder
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import mutual_info_classif
from matplotlib import pyplot
```

In [11]:
```
import seaborn as sns
from sklearn.model_selection import train_test_split
```

In [12]:
```
%config InlineBackend.figure_format = 'retina'
```

In [13]:
```
import warnings
warnings.filterwarnings('ignore')
```

In [14]:
```
import pandas as pd
from sklearn.compose import make_column_transformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import cross_val_score
```

Read data set

In [15]:
```
file_path = "../DataSet/"
file_name = "exercise_40_train.csv"
df = pd.read_csv(file_path+file_name)
```

In [16]:
```
df.shape
```

Out[16]:
```
(40000, 101)
```

In [17]:
```
df.head()
```

Out[17]:

	y	x1	x2	x3	x4	x5	x6	x7	x8	x9	...	x91	x92	x93	x94	x95	x96	x97	x98	x99	x100
0	0	0.165254	18.060003	Wed	1.077380	-1.339233	-1.584341	0.0062%	0.220784	1.816481	...	-0.397427	0.909479	no	5.492487	NaN	10.255579	7.627730	0	yes	104.251338
1	1	2.441471	18.416307	Friday	1.482586	0.920817	-0.759931	0.0064%	1.192441	3.513950	...	0.656651	9.093466	no	3.346429	4.321172	NaN	10.505284	1	yes	101.230645
2	1	4.427278	19.188092	Thursday	0.145652	0.366093	0.709962	-8e-04%	0.952323	0.782974	...	2.059615	0.305170	no	4.456565	NaN	8.754572	7.810979	0	yes	109.345215
3	0	3.925235	19.901257	Tuesday	1.763602	-0.251926	-0.827461	-0.0057%	-0.520756	1.825586	...	0.899392	5.971782	no	4.100022	1.151085	NaN	9.178325	1	yes	103.021970
4	0	2.868802	22.202473	Sunday	3.405119	0.083162	1.381504	0.0109%	-0.732739	2.151990	...	3.003595	1.046096	yes	3.234033	2.074927	9.987006	11.702664	0	yes	92.925935

5 rows × 101 columns

In [18]:
```
df_numeric_features = df.drop(['y'],axis=1)
```

In [19]:
```
df.dtypes.value_counts()
```

Out[19]:
```
float64    86
object     12
int64       3
dtype: int64
```

In [20]:
```
df_numeric_features = df.select_dtypes(include=['float64', 'int64'])
df_numeric_features = df_numeric_features.drop(['y'],axis=1)
```

In [21]:
```
df_cat_features = df.select_dtypes(include=['object'])
df_cat_features.head()
```

Out[21]:

	x3	x7	x19	x24	x31	x33	x39	x60	x65	x77	x93	x99
0	Wed	0.0062%	$-908.650758424405	female	no	Colorado	5-10 miles	August	farmers	mercedes	no	yes
1	Friday	0.0064%	$-1864.9622875143	male	no	Tennessee	5-10 miles	April	allstate	mercedes	no	yes
2	Thursday	-8e-04%	$-543.187402955527	male	no	Texas	5-10 miles	September	geico	subaru	no	yes
3	Tuesday	-0.0057%	$-182.626380634258	male	no	Minnesota	5-10 miles	September	geico	nissan	no	yes
4	Sunday	0.0109%	$967.007090837503	male	yes	New York	5-10 miles	January	geico	toyota	yes	yes

Make custom transformation for categorical features

X3:

```
x3 is nominal categorical feature (needs OneHotEncoder transformer) and custom function edit is needed
```

Making a custom eddit transformation for x3 column as follow.

In [22]:
```
def custom_edit_x3(df_categ):

    # https://stackoverflow.com/questions/60237488/python-replace-only-exact-word-in-string
    for i, string in enumerate(df_categ['x3'].values):
        df_categ['x3'].values[i] = re.sub(r'\bTue\b', 'Tuesday', string)

    for i, string in enumerate(df_categ['x3'].values):
        df_categ['x3'].values[i] = re.sub(r'\bSun\b', 'Sunday', string)

    for i, string in enumerate(df_categ['x3'].values):
        df_categ['x3'].values[i] = re.sub(r'\bSat\b', 'Saturday', string)

    for i, string in enumerate(df_categ['x3'].values):
        df_categ['x3'].values[i] = re.sub(r'\bFri\b', 'Friday', string)

    for i, string in enumerate(df_categ['x3'].values):
        df_categ['x3'].values[i] = re.sub(r'\bWed\b', 'Wednesday', string)

    for i, string in enumerate(df_categ['x3'].values):
        df_categ['x3'].values[i] = re.sub(r'\bMon\b', 'Monday', string)

    for i, string in enumerate(df_categ['x3'].values):
        df_categ['x3'].values[i] = re.sub(r'\bThur\b', 'Thursday', string)
    return df_categ

    return df_categ
```

make a column transformer fucntion and make sure it works

In [23]:
```
custom_edit_x3_tf = FunctionTransformer(custom_edit_x3)
ct = make_column_transformer((custom_edit_x3_tf, ['x3']))
```

In [24]:
```
ct.fit_transform(df)
```

Out[24]:
```
array([['Wednesday'],
       ['Friday'],
       ['Thursday'],
       ...,
       ['Monday'],
       ['Tuesday'],
       ['Thursday']], dtype=object)
```

as it is seen the new column transformer works just fine. I need to perform two tasks on this column therefore I make a pipeline and then I make a new column transformer out of it. Note that introducing two separate functions into the column transformer generates two new features out of one input column. This is due to the fact that the function transformer work in parallel on one input.

I want my custom transformer function to first edit the column and does OneHotEncoding.

In [25]:
```
x3_pipe = make_pipeline(custom_edit_x3_tf, SimpleImputer(strategy="most_frequent"), OneHotEncoder())
ct = make_column_transformer((x3_pipe, ['x3']))
ct.fit_transform(df).toarray()[0].size
```

Out[25]:
```
7
```

Nice! now I have a customized transformer for a specific x3 feature! let's practice the same procedure based on recaps we had for categorical features - the P01 part of this study.

X7:

`x7 column: the % sign should be cleaned from strings and I need to change the dtype and introduce it as numeric feature.`

I just filter it out the % sign using custom tf.

make a column transformer fucntion and make sure it works/

In [26]:
```
# from tqdm import tqdm

# def custom_edit_x7(df_categ):
#     for i, string in (enumerate(tqdm(df_categ['x7'].values))):
# #         print(i)
#         df_categ['x7'].values[i] = df['x7'].str.replace('%','')
# #     df['x7'] = df['x7'].str.replace('%','')

# custom_edit_x7_tf = FunctionTransformer(custom_edit_x7)
# ct = make_column_transformer((custom_edit_x7_tf, ['x7']))
# ct.fit_transform(df)
```

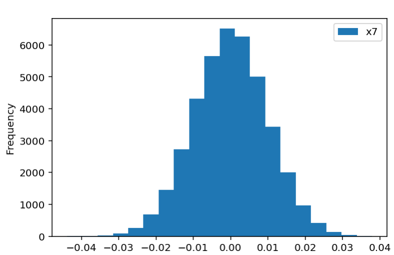
In [27]:
```
def custom_edit_x7(df):
    df = df.apply(lambda x: (x.str.replace('%','')))

    return (np.float_(df))

custom_edit_x7_tf = FunctionTransformer(custom_edit_x7)
ct = make_column_transformer((custom_edit_x7_tf, ['x7']))
x7 = ct.fit_transform(df)
```

In [28]:
```
x7_pd = pd.DataFrame(x7, columns=['x7'])
x7_pd.plot(kind='hist', bins=20)
```

Out[28]:
```
<AxesSubplot:ylabel='Frequency'>
```



I like the x7 distribution. so no need for any other transformation.

In [29]:
```
x7_pd.isnull().sum()
```

Out[29]:
```
x7    0
dtype: int64
```

no null value.

X7 is numerical!I will take care of it in numerical section.

In [30]:
```
# x7_pipe = make_pipeline(custom_edit_x7_tf, SimpleImputer(strategy="most_frequent"), OneHotEncoder())
# ct = make_column_transformer((x7_pipe, ['x7']))
# ct.fit_transform(df).toarray()[0]
```

x19

```
x19 column: the % sign should be cleaned from strings and I need to change the dtype and introduce it as numeric feature.
```

In [31]:
```
df['x19'].head(2)
```

Out[31]:
```
0    $-908.650758424405
1     $-1864.9622875143
Name: x19, dtype: object
```

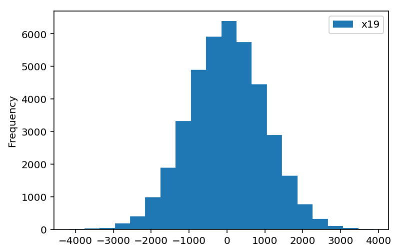
In [32]:
```
def custom_edit_x19(df):
    df = df.apply(lambda x: (x.str.replace('$','')))

    return (np.float_(df))

custom_edit_x19_tf = FunctionTransformer(custom_edit_x19)
ct = make_column_transformer((custom_edit_x19_tf, ['x19']))
x19 = ct.fit_transform(df)
```

In [33]:
```
x19_pd = pd.DataFrame(x19, columns=['x19'])
x19_pd.plot(kind='hist', bins=20)
```

Out[33]: <AxesSubplot:ylabel='Frequency'>



good.

In [34]:
```
x19_pd.isnull().sum()
```

Out[34]:
```
x19    0
dtype: int64
```

X19 is numerical!I will take care of it in numerical section.

In [35]:
```
# x19_pipe = make_pipeline(custom_edit_x19_tf, SimpleImputer(strategy="most_frequent"), OneHotEncoder())
# ct = make_column_transformer((x19_pipe, ['x19']))
# ct.fit_transform(df).toarray()[0]
```

Rest of categorical features

```
x24 column: is nominal category and needs to be transformed using One Hot Encoding
x31 column: is nominal category and needs to be transformed using One Hot Encoding
x33 column: is nominal category and needs to be transformed using One Hot Encoding
x60 is nominal and OneHotEncoder should be used to transfer this column
x65 is nominal and OneHotEncoder should be used to transfer this column
x77 is nominal and OneHotEncoder should be used to transfer this column
x93 is nominal and OneHotEncoder should be used to transfer this column
```

In [36]:
```
for col in ['x24','x31','x33','x60','x65','x77','x93']:
    print(df[col].dtypes)
    print(df[col].unique(),'\n')
```

```
object
['female' 'male' nan]

object
['no' 'yes']

object
['Colorado' 'Tennessee' 'Texas' 'Minnesota' 'New York' 'Florida'
 'Nebraska' 'California' nan 'North Dakota' 'Arizona' 'Alabama' 'Ohio'
 'Pennsylvania' 'Iowa' 'Indiana' 'Vermont' 'Arkansas' 'Massachusetts'
 'Illinois' 'Georgia' 'West Virginia' 'Connecticut' 'Virginia'
 'North Carolina' 'Montana' 'New Mexico' 'New Hampshire' 'Michigan' 'DC'
 'Washington' 'Louisiana' 'Kentucky' 'Utah' 'Missouri' 'Oregon' 'Oklahoma'
 'Nevada' 'Wisconsin' 'New Jersey' 'Maryland' 'Maine' 'Alaska' 'Idaho'
 'Wyoming' 'Rhode Island' 'South Dakota' 'Mississippi' 'Kansas' 'Delaware'
 'Hawaii' 'South Carolina']

object
['August' 'April' 'September' 'January' 'December' 'March' 'July'
 'November' 'June' 'February' 'October' 'May']

object
['farmers' 'allstate' 'geico' 'progressive' 'esurance']

object
['mercedes' 'subaru' 'nissan' 'toyota' nan 'chevrolet' 'buick' 'ford']

object
['no' 'yes']
```

let's make sure about NaNs in above categorical columns

In [37]:
```
two_step_transformer_pipe = make_pipeline(SimpleImputer(strategy="most_frequent"),
                                          OneHotEncoder())
```

In [38]:
```
two_step_transformer_pipe.fit_transform(df[['x24','x31','x33','x60','x65','x77','x93']]).toarray()[0].size
```

Out[38]: 81

In [39]:
```
ct = make_column_transformer((x3_pipe, ['x3']),
                             (two_step_transformer_pipe, ['x24','x31','x33','x60','x65','x77','x93']),
                            )
```

In [40]:
```
ct.fit_transform(df).toarray()[0].size
```

Out[40]: 88

nice!

remeber that I have two columns x7,x19 which have to be consider in numerical section.

Numerical features pipeline

'x30', 'x44', 'x57'

```
Column 'x30', 'x44', 'x57' have more than 50% nan and should be droped
```

```
Three features inside the numeric df need to be transformed using One Hot Encoding. These are features with 0-1 values and should be treated like category
features. These are 0-1 features and should be treated like category features. columns are x59 and x79 and x98
```

x59 and x79 and x98

In [41]:
```
for col in ['x59', 'x79', 'x98']:
    print(df[col].dtypes)
    print(df[col].unique())
    print(len(df[col].unique()),'\n')
```

```
int64
[0 1]
2

float64
[ 1. nan  0.]
3

int64
[0 1]
2
```

In [42]:
```
numer_OHE_x59_x79_x98 = make_pipeline(SimpleImputer(strategy="most_frequent"),
                                      OneHotEncoder())

numer_OHE_x59_x79_x98.fit_transform(np.array(df['x59']).reshape(-1, 1)).toarray()[0].size
```

Out[42]: 2

In [43]:
```
len(numer_OHE_x59_x79_x98.fit_transform(np.array(df['x59']).reshape(-1, 1)).toarray())
```

Out[43]: 40000

In [44]:
```
ct = make_column_transformer((numer_OHE_x59_x79_x98, ['x59', 'x79', 'x98']))
```

In [45]:
```
ct.fit_transform(df)
```

Out[45]:
```
array([[1., 0., 0., 1., 1., 0.],
       [1., 0., 0., 1., 0., 1.],
       [1., 0., 0., 1., 1., 0.],
       ...,
       [1., 0., 0., 1., 0., 1.],
       [1., 0., 0., 1., 1., 0.],
       [0., 1., 0., 1., 0., 1.]])
```

'x13', 'x21', 'x32', 'x35', 'x44', 'x59', 'x67', 'x73', 'x75', 'x79', 'x84', 'x89'

- cols with high skewness are `['x13', 'x21', 'x32', 'x35', 'x44', 'x59', 'x67', 'x73', 'x75', 'x79', 'x84', 'x89']`. I will apply power transformer.

In [46]:
```
numer_skewness = make_pipeline(SimpleImputer(strategy="mean"),
                               StandardScaler(),
                                PowerTransformer(),
#                                 VarianceThreshold(),
                              )

numer_skewness.fit_transform(np.array(df['x13']).reshape(-1, 1))
```

Out[46]:
```
array([[ 0.52252661],
       [-0.98789216],
       [-1.11580409],
       ...,
       [-0.26185934],
       [-2.10399184],
       [-0.25390223]])
```

In [47]:
```
ct = make_column_transformer((numer_skewness, ['x13', 'x21', 'x32', 'x35',
                                               'x44', 'x59', 'x67', 'x73',
                                               'x75', 'x79', 'x84', 'x89']))
```

In [48]:
```
ct.fit_transform(df)
```

Out[48]:
```
array([[ 0.52252661,  1.6190496 , -0.68519579, ...,  0.42510903,
        -2.88844405,  0.15809134],
       [-0.98789216, -0.40425833,  0.60652813, ..., -1.8907331 ,
         0.0591937 ,  0.08691384],
       [-1.11580409,  0.16008288,  1.24974419, ...,  0.42510903,
         0.0591937 ,  0.21667148],
       ...,
       [-0.26185934, -0.73586194,  0.13156308, ...,  0.42510903,
         2.05710353, -1.38447206],
       [-2.10399184,  0.05575166,  1.33614618, ...,  0.42510903,
         0.0591937 ,  0.21667148],
       [-0.25390223, -0.87867438,  0.34614004, ...,  0.42510903,
         0.0591937 ,  0.21667148]])
```

Rest of numeric features

filtering out numeric intact numeric features

In [49]:
```
rest_numeric_features = []
for col in df_numeric_features.columns:
    if col != ['x13', 'x21', 'x32', 'x35', 'x44', 'x59', 'x67', 'x73', 'x75', 'x79', 'x84', 'x89']:
        rest_numeric_features.append(col)
print(rest_numeric_features)
```

```
['x1', 'x2', 'x4', 'x5', 'x6', 'x8', 'x9', 'x10', 'x11', 'x12', 'x13', 'x14', 'x15', 'x16', 'x17', 'x18', 'x20', 'x21', 'x22', 'x23', 'x25', 'x26', 'x27', 'x28', 'x29', 'x30', 'x32', 'x34', 'x35', 'x36', 'x37', 'x38', 'x40', 'x41', 'x42', 'x43', 'x44', 'x45', 'x46', 'x47', 'x48', 'x49', 'x50', 'x51', 'x52', 'x53', 'x54', 'x55', 'x56', 'x57', 'x58', 'x59', 'x61', 'x62', 'x63', 'x64', 'x66', 'x67', 'x68', 'x69', 'x70', 'x71', 'x72', 'x73', 'x74', 'x75', 'x76', 'x78', 'x79', 'x80', 'x81', 'x82', 'x83', 'x84', 'x85', 'x86', 'x87', 'x88', 'x89', 'x90', 'x91', 'x92', 'x94', 'x95', 'x96', 'x97', 'x98', 'x100']
```

In [50]:
```
Rest_numeric_features = make_pipeline(SimpleImputer(strategy="mean"),
                               StandardScaler(),
                                PowerTransformer(),
#                                 VarianceThreshold(),
#                              PCA(n_components=20),
                                     )

# Rest_numeric_features.fit_transform(np.array(df['x1']).reshape(-1, 1))
```

In [51]:
```
ct = make_column_transformer((Rest_numeric_features, rest_numeric_features))
```

In [52]:
```
ct.fit_transform(df)
```

Out[52]:
```
array([[-1.52210183, -1.21321633,  0.73538846, ..., -1.19734632,
        -0.99531099,  0.80499414],
       [-0.21778515, -0.99034322,  1.01178638, ...,  0.25640949,
         1.0047111 ,  0.22810827],
       [ 0.75277741, -0.50800169,  0.09907159, ..., -1.10434356,
        -0.99531099,  1.77989024],
       ...,
       [-0.62132649, -1.36064831, -0.39842492, ..., -1.17905537,
         1.0047111 , -0.0022491 ],
       [ 0.53960265,  0.17657422,  0.29347708, ..., -0.68188951,
        -0.99531099,  1.80192371],
       [-0.14750603, -2.22596216, -0.79910404, ...,  1.26723974,
         1.0047111 ,  1.96954651]])
```

Combining all transformers into one

In [53]:
```
ct_all_features = make_column_transformer((x3_pipe, ['x3']),
                            (two_step_transformer_pipe, ['x24','x31','x33','x60','x65','x77','x93']),
                            (numer_OHE_x59_x79_x98, ['x59', 'x79', 'x98']),

                             # features with high skewness
                            (numer_skewness, ['x13', 'x21', 'x32', 'x35',
                                              'x44', 'x59', 'x67', 'x73',
                                              'x75', 'x79', 'x84', 'x89']),

                             # features from categories-> edited to numeric-no null found
                             (custom_edit_x7_tf, ['x7']),
                             (custom_edit_x19_tf, ['x19']),

                             # rest numeric features
                             (Rest_numeric_features, rest_numeric_features),
                            )

ct_all_features.fit_transform(df)[0].size
```

Out[53]: 196

so far so good.

In [54]:
```
weights = {0:0.85, 1:0.14}
logreg = LogisticRegression(solver='sag', class_weight=weights)
```

In [159]:
```
X_all_features = df.drop(columns=['y'], axis=1)
y = df['y']
```

In [56]:
```
Final_pipe = make_pipeline(ct_all_features, PCA(n_components=15), logreg)
cross_val_score(Final_pipe, X_all_features, y, cv=5, scoring='roc_auc', n_jobs=-1)
```

Out[56]: array([0.61616909, 0.60814428, 0.64553577, 0.60456741, 0.61493456])

Using just numeric features

In [57]:
```
ct_numeric_features = make_column_transformer(
#                              (x3_pipe, ['x3']),
#                              (two_step_transformer_pipe, ['x24','x31','x33','x60','x65','x77','x93']),
#                              (numer_OHE_x59_x79_x98, ['x59', 'x79', 'x98']),

                             # features with high skewness
                            (numer_skewness, ['x13', 'x21', 'x32', 'x35',
                                              'x44', 'x59', 'x67', 'x73',
                                              'x75', 'x79', 'x84', 'x89']),

                             # features from categories-> edited to numeric-no null found
                             (custom_edit_x7_tf, ['x7']),
                             (custom_edit_x19_tf, ['x19']),

                             # rest numeric features
                             (Rest_numeric_features, rest_numeric_features),
                            )

ct_numeric_features.fit_transform(df)[0].size
```

Out[57]: 102

In [58]:
```
ct_numeric_features.fit_transform(df)[0]
```

Out[58]:
```
array([ 5.22526612e-01,  1.61904960e+00, -6.85195795e-01,  1.12600949e+00,
       -6.91197613e-01, -3.32684806e-01, -1.44362004e+00,  4.46384718e-01,
        2.56006418e-01,  4.25109025e-01, -2.88844405e+00,  1.58091341e-01,
        6.20000000e-03, -9.08650758e+02, -1.52210183e+00, -1.21321633e+00,
        7.35388464e-01, -1.06857966e+00, -1.17217998e+00,  1.52149506e-01,
       -3.81399222e-01,  6.62628694e-01,  7.70770661e-01,  4.47015177e-01,
        5.22526612e-01,  1.15252078e+00, -4.06315664e-01,  1.29240785e+00,
       -1.64745508e+00, -3.36256364e-01,  6.68488814e-01,  1.61904960e+00,
        1.81992298e+00,  1.04048861e+00, -4.27167241e-01,  1.55459881e+00,
       -8.99619529e-01, -6.89990170e-02, -1.20700476e-01, -3.70836348e-03,
       -6.85195795e-01, -1.08639369e+00,  1.12600949e+00, -1.29287150e-01,
        4.22211953e-01,  8.03718211e-02,  9.75140583e-01,  1.20746002e-03,
        3.42940061e-01,  6.98794445e-01, -6.91197613e-01, -1.44299199e+00,
        1.21988343e-01,  7.60525530e-01, -3.19161441e-01,  2.77286798e+00,
       -5.23125852e-01,  8.36160265e-01,  4.73079334e-01,  4.39733390e-01,
        1.42687191e-01,  1.02575840e-01, -1.39962548e+00, -1.88182094e-03,
        2.16755070e-03, -3.32684806e-01, -5.84134453e-02,  1.21674877e+00,
        8.57178394e-01, -6.49825658e-01,  8.20792043e-01, -1.44362004e+00,
        2.01134855e-01,  3.85186079e-01, -4.49141141e-01, -2.14518586e+00,
        5.49346092e-01,  4.46384718e-01, -1.66378019e-03,  2.56006418e-01,
       -3.73414390e-02,  3.83339156e-01,  4.25109025e-01,  4.60051071e-01,
       -3.14925348e-01,  1.21007498e+00, -1.90768034e+00, -2.88844405e+00,
        2.96281855e+00, -4.54190924e-01, -1.31720586e-01,  6.02442226e-01,
        1.58091341e-01, -1.96768655e-01, -2.93977095e-01, -1.06472668e+00,
        1.06006105e+00,  6.99552121e-03, -9.50128160e-02, -1.19734632e+00,
       -9.95310993e-01,  8.04994143e-01])
```

In [59]:
```
Final_pipe = make_pipeline(ct_numeric_features, logreg)
cross_val_score(Final_pipe, df, y, cv=5, scoring='roc_auc', n_jobs=-1)
```

Out[59]: array([0.62904441, 0.61972714, 0.6583932 , 0.61504388, 0.6249599])

Using just categorical features

In [60]:
```
ct_categ_features = make_column_transformer(
                            (x3_pipe, ['x3']),
                            (two_step_transformer_pipe, ['x24','x31','x33','x60','x65','x77','x93']),
                            (numer_OHE_x59_x79_x98, ['x59', 'x79', 'x98']),

#                               # features with high skewness
#                              (numer_skewness, ['x13', 'x21', 'x32', 'x35',
#                                                'x44', 'x59', 'x67', 'x73',
#                                                'x75', 'x79', 'x84', 'x89']),

#                               # features from categories-> edited to numeric-no null found
#                               (custom_edit_x7_tf, ['x7']),
#                               (custom_edit_x19_tf, ['x19']),

#                               # rest numeric features
#                               (Rest_numeric_features, rest_numeric_features),
                            )

ct_categ_features.fit_transform(df)[0].size
```

Out[60]: 11

In [61]:
```
Final_pipe = make_pipeline(ct_categ_features, logreg)
cross_val_score(Final_pipe, df, y, cv=5, scoring='roc_auc', n_jobs=-1)
```

Out[61]: array([0.62876021, 0.60991241, 0.60549845, 0.6194927 , 0.60416943])

Using MLP

In [173]:
```
from sklearn.neural_network import MLPClassifier
```

In [176]:
```
clf = MLPClassifier(random_state=1, max_iter=300)
```

In [177]:
```
Final_pipe = make_pipeline(ct_all_features, clf)
cross_val_score(Final_pipe, df, y, cv=5, scoring='roc_auc', n_jobs=-1)
```

Out[177]: array([0.71408815, 0.74299607, 0.7497773 , 0.750639 , 0.71121597])

GradientBoostingClassifier

In [81]:
```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import cross_val_predict
from sklearn.model_selection import cross_validate
```

In [72]:
```
clf = GradientBoostingClassifier(n_estimators=100, learning_rate=1.0,
        max_depth=1, random_state=0)
```

In [130]:
```
Final_pipe = make_pipeline(ct_all_features, clf)
cross_val_score(Final_pipe, df, y, cv=5, scoring='roc_auc', n_jobs=-1)
```

Out[130]: array([0.79471264, 0.78739237, 0.80152993, 0.79287561, 0.78379988])

In [131]:
```
cv_results = cross_validate(Final_pipe, df, y, cv=5,
                            scoring='roc_auc', n_jobs=-1,
                            return_estimator=True,
                            return_train_score=True)
```

In [132]:
```
clf_fold_0 = cv_results['estimator'][0]
```

In [133]:
```
# clf_fold_0.predict_proba(df)
```

In [134]:
```
resuls = pd.DataFrame(clf_fold_0.predict_proba(df),
                      columns=['0','1'])
resuls.head()
```

Out[134]:

	0	1
0	0.946563	0.053437
1	0.697074	0.302926
2	0.821153	0.178847
3	0.768391	0.231609
4	0.999145	0.000855

ROC curve

In [161]:
```
Final_pipe.fit(df,y)
```

Out[161]:
```
Pipeline(steps=[('columntransformer',
                 ColumnTransformer(transformers=[('pipeline-1',
                                                  Pipeline(steps=[('functiontransformer',
                                                                   FunctionTransformer(func=<function custom_edit_x3 at 0x7fed3548b048>)),
                                                                  ('simpleimputer',
                                                                   SimpleImputer(strategy='most_frequent')),
                                                                  ('onehotencoder',
                                                                   OneHotEncoder())]),
                                                  ['x3']),
                                                 ('pipeline-2',
                                                  Pipeline(steps=[('simpleimputer',
                                                                   SimpleImput...
                                                                  ('standardscaler',
                                                                   StandardScaler()),
                                                                  ('powertransformer',
                                                                   PowerTransformer())]),
                                                  ['x1', 'x2', 'x4', 'x5', 'x6',
                                                   'x8', 'x9', 'x10', 'x11',
                                                   'x12', 'x13', 'x14', 'x15',
                                                   'x16', 'x17', 'x18', 'x20',
                                                   'x21', 'x22', 'x23', 'x25',
                                                   'x26', 'x27', 'x28', 'x29',
                                                   'x30', 'x32', 'x34', 'x35',
                                                   'x36', ...])])),
                ('gradientboostingclassifier',
                 GradientBoostingClassifier(learning_rate=1.0, max_depth=1,
                                            random_state=0))])
```

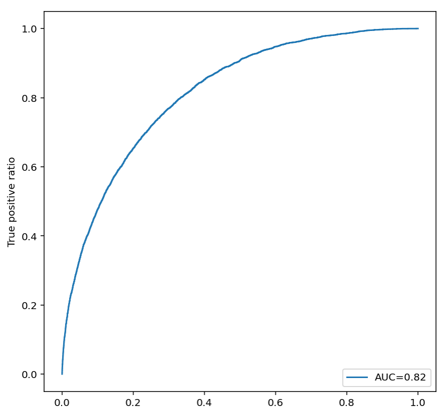
In [163]:
```
Final_pipe.score(df,y)
```

Out[163]:
```
0.86755
```

In [164]:
```
# Final_pipe.fit(df,y)
y_prob = Final_pipe.predict_proba(df)
y_pred_proba = y_prob[:,1]
```

In [189]:
```
from sklearn.metrics import roc_curve
import sklearn.metrics as metrics

plt.figure(figsize=(7,7))
# https://stackoverflow.com/questions/25009284/how-to-plot-roc-curve-in-python
fpr, tpr, _ = metrics.roc_curve(y,  y_pred_proba)
auc = metrics.roc_auc_score(y, y_pred_proba)
plt.plot(fpr,tpr, label="AUC="+str(round(auc,2)))
plt.legend(loc=4)
plt.xlabel('False positive ratio')
plt.ylabel('True positive ratio')
plt.show()
```



In [ ]: