ITADATAhack2024 Notebook

September 9, 2024

1 ITADATA Hackathon 2024

1.0.1 Team Padowurel

1.1 Predicting Customer Creditworthiness

Shall we give credit to a bank costumer or not? Given a dataset based on real data the main goal is to categorize the debtors in two groups: one contains trustworthy costumers who are likely to pay their debt off, the other group has instead less reliable costumer. Obviously, with the most performant precision-recall trade-off.

1.1.1 A first set-up

`Pandas`, `Numpy` and `Matplotlib` are the libraries which are going to get involved in the following exploration, analysis and prediction of the data. Data are contained in the `training.csv` and `test.csv` files.

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

from imblearn.under_sampling import RandomUnderSampler
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import f1_score, classification_report, confusion_matrix
from sklearn.linear_model import LogisticRegressionCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import LinearSVC
```

[2]:
```
train = pd.read_csv("training.csv")
test = pd.read_csv("test.csv")
```

[3]:
```
cols = train.columns
train[cols[:18]].describe()
```

[3]:
```
          client_id      product8     product10     product13     product12  \
count  2.056160e+06  2.056160e+06  2.056160e+06  2.056160e+06  2.056160e+06
mean   5.173345e+04  2.386609e-01  2.843013e-01  2.253502e-01  2.000948e-01
std    2.983887e+04  4.262651e-01  4.510811e-01  4.178128e-01  4.000712e-01
min    0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
25%    2.592775e+04  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
50%    5.173850e+04  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
75%    7.756425e+04  0.000000e+00  1.000000e+00  0.000000e+00  0.000000e+00
max    1.034850e+05  1.000000e+00  1.000000e+00  1.000000e+00  1.000000e+00

          product11      product4     product17      product2      product3  \
count  2.056160e+06  2.056160e+06  2.056160e+06  2.056160e+06  2.056160e+06
mean   2.254596e-01  2.029195e-01  2.848319e-01  4.942354e-01  5.005564e-01
std    4.178847e-01  4.021732e-01  4.513345e-01  4.999669e-01  4.999998e-01
min    0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
25%    0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
50%    0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00  1.000000e+00
75%    0.000000e+00  0.000000e+00  1.000000e+00  1.000000e+00  1.000000e+00
max    1.000000e+00  1.000000e+00  1.000000e+00  1.000000e+00  1.000000e+00

           product1      product7      product6      product5     product14  \
count  2.056160e+06  2.056160e+06  2.056160e+06  2.056160e+06  2.056160e+06
mean   3.056338e-01  2.057568e-01  2.306109e-01  2.025708e-01  2.026224e-01
std    4.606755e-01  4.042537e-01  4.212240e-01  4.019154e-01  4.019535e-01
min    0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
25%    0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
50%    0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
75%    1.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
max    1.000000e+00  1.000000e+00  1.000000e+00  1.000000e+00  1.000000e+00

          product15     product16      product9
count  2.056160e+06  2.056160e+06  2.056160e+06
mean   4.072699e-01  1.999825e-01  3.214604e-01
std    4.913260e-01  3.999870e-01  4.670372e-01
min    0.000000e+00  0.000000e+00  0.000000e+00
25%    0.000000e+00  0.000000e+00  0.000000e+00
50%    0.000000e+00  0.000000e+00  0.000000e+00
75%    1.000000e+00  0.000000e+00  1.000000e+00
max    1.000000e+00  1.000000e+00  1.000000e+00
```

[4]:
```
train[cols[18:36]].describe()
```

[4]:
```
       has_products       balance     left_bank   joined_bank  \
count  2.056160e+06  2.056160e+06  2.056160e+06  2.056160e+06
mean   8.104972e-01  9.871905e+03  1.423284e-02  9.423391e-03
std    3.919076e-01  7.263675e+04  1.184495e-01  9.661571e-02
min    0.000000e+00 -4.770451e+06  0.000000e+00  0.000000e+00
```

25%	1.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
50%	1.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
75%	1.000000e+00	2.733142e+03	0.000000e+00	0.000000e+00
max	1.000000e+00	1.206681e+07	1.000000e+00	1.000000e+00

	wire_transfers2_amt_inbound	wire_transfers1_amt_inbound \
count	2.056160e+06	2.056160e+06
mean	9.100295e+01	6.221871e+03
std	3.589490e+03	1.244680e+05
min	0.000000e+00	0.000000e+00
25%	0.000000e+00	0.000000e+00
50%	0.000000e+00	0.000000e+00
75%	0.000000e+00	1.509032e+02
max	1.544773e+06	3.176076e+07

	wire_transfers2_amt_outbound	wire_transfers1_amt_outbound \
count	2.056160e+06	2.056160e+06
mean	-7.886732e+01	-4.898460e+03
std	1.658415e+03	1.216337e+05
min	-6.540298e+05	-3.951999e+07
25%	0.000000e+00	0.000000e+00
50%	0.000000e+00	0.000000e+00
75%	0.000000e+00	0.000000e+00
max	0.000000e+00	0.000000e+00

	counter_amt_inbound	counter_amt_outbound	securities_bought_amt \
count	2.056160e+06	2.056160e+06	2.056160e+06
mean	1.028859e+04	-5.549659e+03	3.835457e+04
std	1.622195e+05	1.467409e+05	1.680672e+07
min	0.000000e+00	-1.050320e+08	0.000000e+00
25%	0.000000e+00	-6.330285e+01	0.000000e+00
50%	0.000000e+00	0.000000e+00	0.000000e+00
75%	0.000000e+00	0.000000e+00	0.000000e+00
max	6.184164e+07	0.000000e+00	1.012808e+10

	securities_sold_amt	wire_transfers2_num_inbound \
count	2.056160e+06	2.056160e+06
mean	3.831823e+04	5.407994e-02
std	1.696479e+07	6.545238e-01
min	0.000000e+00	0.000000e+00
25%	0.000000e+00	0.000000e+00
50%	0.000000e+00	0.000000e+00
75%	0.000000e+00	0.000000e+00
max	1.053708e+10	1.570000e+02

	wire_transfers1_num_inbound	wire_transfers2_num_outbound \
count	2.056160e+06	2.056160e+06

mean	2.594861e+00	5.908879e-02
std	8.026009e+01	5.344656e-01
min	0.000000e+00	0.000000e+00
25%	0.000000e+00	0.000000e+00
50%	0.000000e+00	0.000000e+00
75%	1.000000e+00	0.000000e+00
max	3.024900e+04	5.600000e+01

	wire_transfers1_num_outbound	counter_num_inbound	counter_num_outbound
count	2.056160e+06	2.056160e+06	2.056160e+06
mean	1.591083e+00	1.922663e+00	1.620659e+00
std	1.297299e+01	1.435850e+01	6.585753e+00
min	0.000000e+00	0.000000e+00	0.000000e+00
25%	0.000000e+00	0.000000e+00	0.000000e+00
50%	0.000000e+00	0.000000e+00	0.000000e+00
75%	0.000000e+00	0.000000e+00	1.000000e+00
max	2.892000e+03	1.639000e+03	7.540000e+02

[5]:
```
train[cols[36:]].describe()
```

[5]:

	securities_operations	securities_bought	securities_sold	\
count	2.056160e+06	2.056160e+06	2.056160e+06	
mean	6.060715e-02	3.455519e-02	2.594983e-02	
std	3.049697e+00	1.674895e+00	1.462421e+00	
min	0.000000e+00	0.000000e+00	0.000000e+00	
25%	0.000000e+00	0.000000e+00	0.000000e+00	
50%	0.000000e+00	0.000000e+00	0.000000e+00	
75%	0.000000e+00	0.000000e+00	0.000000e+00	
max	1.238000e+03	6.980000e+02	5.910000e+02	

	counter_amt_tot	counter_num_tot	period	category	\
count	2.056160e+06	2.056160e+06	2.056160e+06	2.056160e+06	
mean	4.738936e+03	3.543322e+00	1.050000e+01	2.707649e+00	
std	1.473733e+05	1.799122e+01	5.766283e+00	6.328854e-01	
min	-1.050285e+08	0.000000e+00	1.000000e+00	1.000000e+00	
25%	0.000000e+00	0.000000e+00	5.750000e+00	3.000000e+00	
50%	0.000000e+00	0.000000e+00	1.050000e+01	3.000000e+00	
75%	0.000000e+00	2.000000e+00	1.525000e+01	3.000000e+00	
max	6.173115e+07	1.713000e+03	2.000000e+01	3.000000e+00	

	repays_debt
count	2.056160e+06
mean	3.251692e-03
std	5.693084e-02
min	0.000000e+00
25%	0.000000e+00
50%	0.000000e+00

```
75%    0.000000e+00
max    1.000000e+00
```

[6]:
```python
print(test[cols[:18]].describe())
print(test[cols[18:36]].describe())
print(test[cols[36:]].describe())
```

```
         client_id     product8    product10    product13    product12  \
count  7890.000000  7890.000000  7890.000000  7890.000000  7890.000000
mean   3939.695564     0.353105     0.395057     0.263625     0.201014
std    2278.018908     0.477965     0.488894     0.440626     0.400784
min       1.000000     0.000000     0.000000     0.000000     0.000000
25%    1973.000000     0.000000     0.000000     0.000000     0.000000
50%    3943.000000     0.000000     0.000000     0.000000     0.000000
75%    5912.000000     1.000000     1.000000     1.000000     0.000000
max    7883.000000     1.000000     1.000000     1.000000     1.000000

         product11     product4    product17     product2     product3  \
count  7890.000000  7890.000000  7890.000000  7890.000000  7890.000000
mean      0.257161     0.202662     0.268314     0.377567     0.519392
std       0.437097     0.402008     0.443110     0.484809     0.499655
min       0.000000     0.000000     0.000000     0.000000     0.000000
25%       0.000000     0.000000     0.000000     0.000000     0.000000
50%       0.000000     0.000000     0.000000     0.000000     1.000000
75%       1.000000     0.000000     1.000000     1.000000     1.000000
max       1.000000     1.000000     1.000000     1.000000     1.000000

          product1     product7     product6     product5    product14  \
count  7890.000000  7890.000000  7890.000000  7890.000000  7890.000000
mean      0.333714     0.228010     0.213054     0.212041     0.212421
std       0.471569     0.419576     0.409492     0.408779     0.409047
min       0.000000     0.000000     0.000000     0.000000     0.000000
25%       0.000000     0.000000     0.000000     0.000000     0.000000
50%       0.000000     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

         product15    product16     product9
count  7890.000000  7890.000000  7890.000000
mean      0.368314     0.197972     0.352852
std       0.482378     0.398496     0.477887
min       0.000000     0.000000     0.000000
25%       0.000000     0.000000     0.000000
50%       0.000000     0.000000     0.000000
75%       1.000000     0.000000     1.000000
max       1.000000     1.000000     1.000000
       has_products        balance    left_bank  joined_bank  \
```

count	7890.000000	7890.000000	7890.000000	7890.000000
mean	0.835868	-5052.954934	0.041065	0.005450
std	0.370419	44678.872967	0.198452	0.073627
min	0.000000	-977491.920000	0.000000	0.000000
25%	1.000000	-376.037500	0.000000	0.000000
50%	1.000000	0.000000	0.000000	0.000000
75%	1.000000	0.000000	0.000000	0.000000
max	1.000000	694089.020000	1.000000	1.000000

	wire_transfers2_amt_inbound	wire_transfers1_amt_inbound
count	7890.000000	7890.000000
mean	17.691408	2578.090556
std	420.754049	18549.321282
min	0.000000	0.000000
25%	0.000000	0.000000
50%	0.000000	0.000000
75%	0.000000	0.000000
max	25973.382694	436236.110693

	wire_transfers2_amt_outbound	wire_transfers1_amt_outbound
count	7890.000000	7890.000000
mean	-15.150839	-2863.474999
std	296.795819	21171.393510
min	-14801.993767	-496648.136888
25%	0.000000	0.000000
50%	0.000000	0.000000
75%	0.000000	0.000000
max	0.000000	0.000000

	counter_amt_inbound	counter_amt_outbound	securities_bought_amt
count	7.890000e+03	7.890000e+03	7890.000000
mean	9.947120e+03	-6.234640e+03	2.481547
std	6.989340e+04	4.386962e+04	143.105568
min	0.000000e+00	-1.238064e+06	0.000000
25%	0.000000e+00	0.000000e+00	0.000000
50%	0.000000e+00	0.000000e+00	0.000000
75%	3.489142e+01	0.000000e+00	0.000000
max	3.836421e+06	0.000000e+00	9993.667939

	securities_sold_amt	wire_transfers2_num_inbound
count	7890.000000	7890.000000
mean	12.048478	0.012801
std	674.188069	0.166121
min	0.000000	0.000000
25%	0.000000	0.000000
50%	0.000000	0.000000
75%	0.000000	0.000000
max	53085.832180	5.000000

	wire_transfers1_num_inbound	wire_transfers2_num_outbound
count	7890.000000	7890.000000
mean	0.894550	0.016730
std	3.811809	0.368934
min	0.000000	0.000000
25%	0.000000	0.000000
50%	0.000000	0.000000
75%	0.000000	0.000000
max	86.000000	28.000000

	wire_transfers1_num_outbound	counter_num_inbound	counter_num_outbound
count	7890.000000	7890.000000	7890.000000
mean	1.423828	1.906591	2.097972
std	7.685713	7.509475	7.849914
min	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000
75%	0.000000	1.000000	0.000000
max	137.000000	130.000000	187.000000

	securities_operations	securities_bought	securities_sold
count	7890.000000	7890.000000	7890.000000
mean	0.002408	0.001394	0.001014
std	0.089331	0.081953	0.038988
min	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000
75%	0.000000	0.000000	0.000000
max	5.000000	6.000000	2.000000

	counter_amt_tot	counter_num_tot	period	category
count	7.890000e+03	7890.000000	7890.000000	7890.000000
mean	3.712480e+03	4.004563	6.804436	2.191001
std	5.587432e+04	14.300152	4.427201	0.822362
min	-8.725672e+05	0.000000	1.000000	1.000000
25%	0.000000e+00	0.000000	3.000000	1.000000
50%	0.000000e+00	0.000000	6.000000	2.000000
75%	0.000000e+00	1.000000	10.000000	3.000000
max	3.341436e+06	194.000000	20.000000	3.000000

Categorical variables must be treated as such.

[7]:
```
X_train = train
train_category = train['category']
X_train['category_2'] = (train['category'] == 2).astype(int)
X_train['category_3'] = (train['category'] == 3).astype(int)
X_train = X_train.drop(columns='category')
```

```
[8]: X_test = test
test_category = test['category']
X_test['category_2'] = (test['category'] == 2).astype(int)
X_test['category_3'] = (test['category'] == 3).astype(int)
X_test = X_test.drop(columns='category')
```

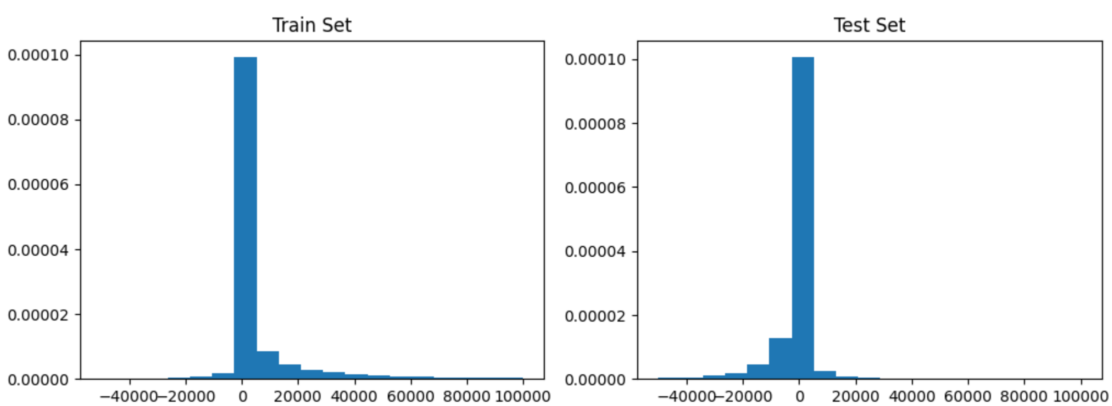
```
[9]: X_train = pd.get_dummies(X_train, columns=['period'], drop_first=True,␣
↪dtype=int)
X_test = pd.get_dummies(X_test, columns=['period'], drop_first=True, dtype=int)
```

1.1.2 Visual Exploration

First of all let's analyze the differences between the compositions of the train and the test set.

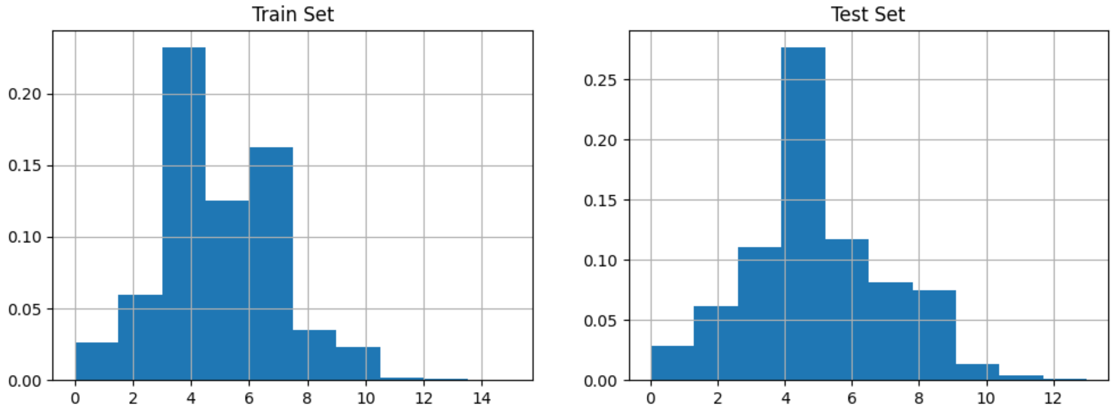
```
[10]: fig, axes = plt.subplots(1, 2, figsize=(12, 4))
axes[0].hist(train['balance'],
         bins=list(np.linspace(-50000, 100000, 20)), density=True)
axes[0].set_title('Train Set')
axes[1].hist(test['balance'],
         bins=list(np.linspace(-50000, 100000, 20)), density=True)
axes[1].set_title('Test Set')
```

```
[10]: Text(0.5, 1.0, 'Test Set')
```



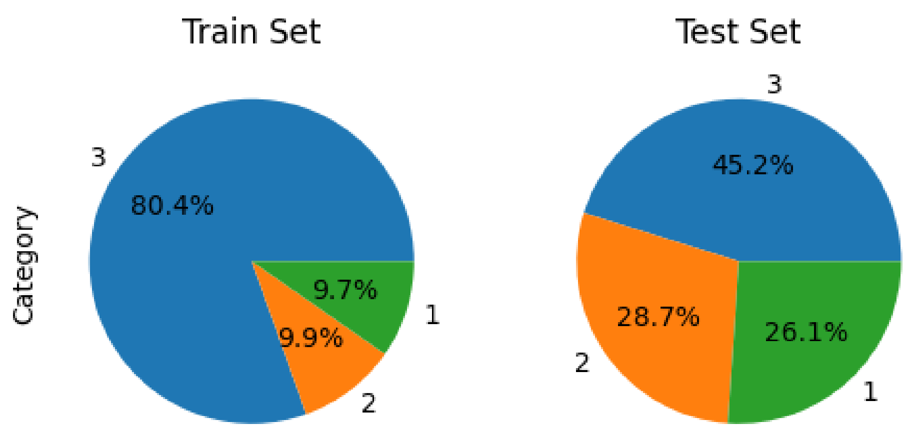

```
[11]: X_train['n_products'] = train[cols[1:18]].apply(sum, axis=1)
X_test['n_products'] = test[cols[1:18]].apply(sum, axis=1)

fig, axes = plt.subplots(1, 2, figsize=(12, 4))
X_train['n_products'].hist(ax = axes[0], density=True)
axes[0].set_title('Train Set')
X_test['n_products'].hist(ax = axes[1], density=True)
axes[1].set_title('Test Set')
```

[11]: Text(0.5, 1.0, 'Test Set')



[12]:
```
fig, axes = plt.subplots(1, 2, figsize=(6, 4))
train['category'].value_counts().plot.pie(ax = axes[0], autopct='%1.1f%%')
axes[0].set_title('Train Set')
axes[0].set_ylabel('Category')
test['category'].value_counts().plot.pie(ax = axes[1], autopct='%1.1f%%')
axes[1].set_title('Test Set')
axes[1].set_ylabel('')
```

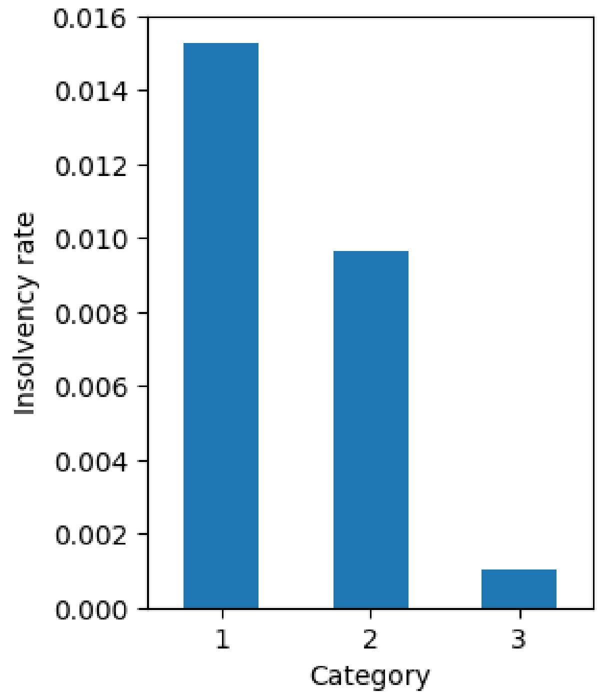
[12]: Text(0, 0.5, '')



[13]:
```
plt.figure(figsize=(3, 4))
train.groupby('category')['repays_debt'].mean().plot(kind='bar')
plt.xticks(rotation=0)
plt.xlabel('Category')
```

```
plt.ylabel('Insolvency rate')
```

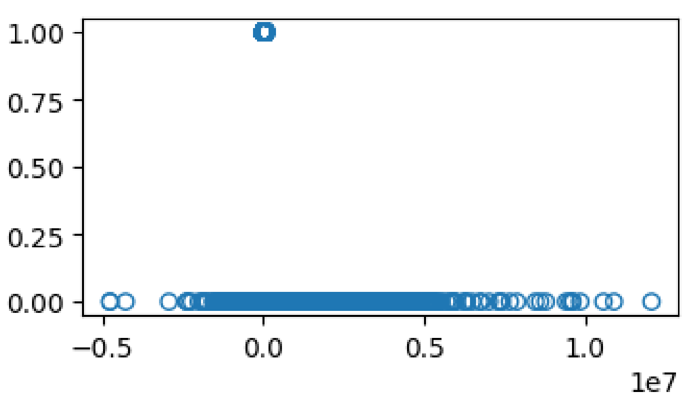
[13]: Text(0, 0.5, 'Insolvency rate')



Costumer's category (firm account, solo proprietorship or private account) is likely to play an important role in the solvency of the debt. Let's go ahead by casting a glance over paired data.

[14]:
```
plt.figure(figsize=(4,2))
plt.scatter(train['balance'], train['repays_debt'],
            marker='o', facecolors='none', edgecolors='tab:blue')
```

[14]: <matplotlib.collections.PathCollection at 0x1d913671490>

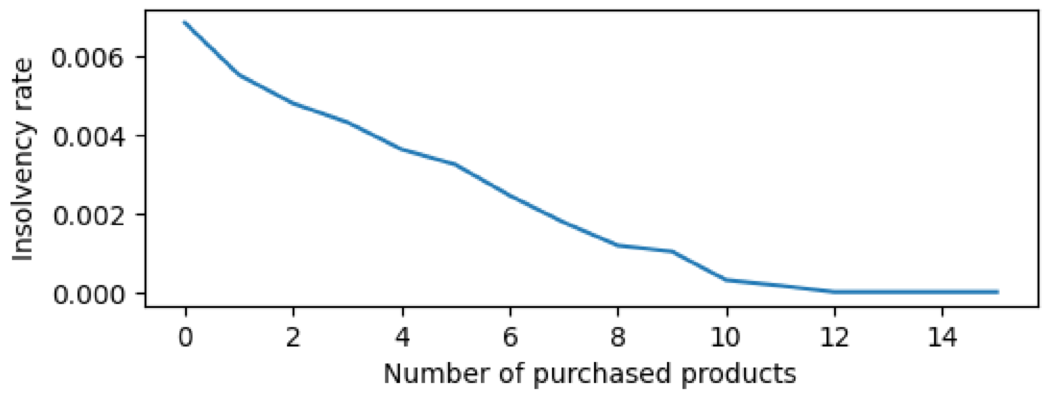


[15]:

```
plt.figure(figsize=(6, 2))
X_train.groupby('n_products')['repays_debt'].mean().plot()
plt.xticks(rotation=0)
plt.xlabel('Number of purchased products')
plt.ylabel('Insolvency rate')
```

[15]:

```
Text(0, 0.5, 'Insolvency rate')
```

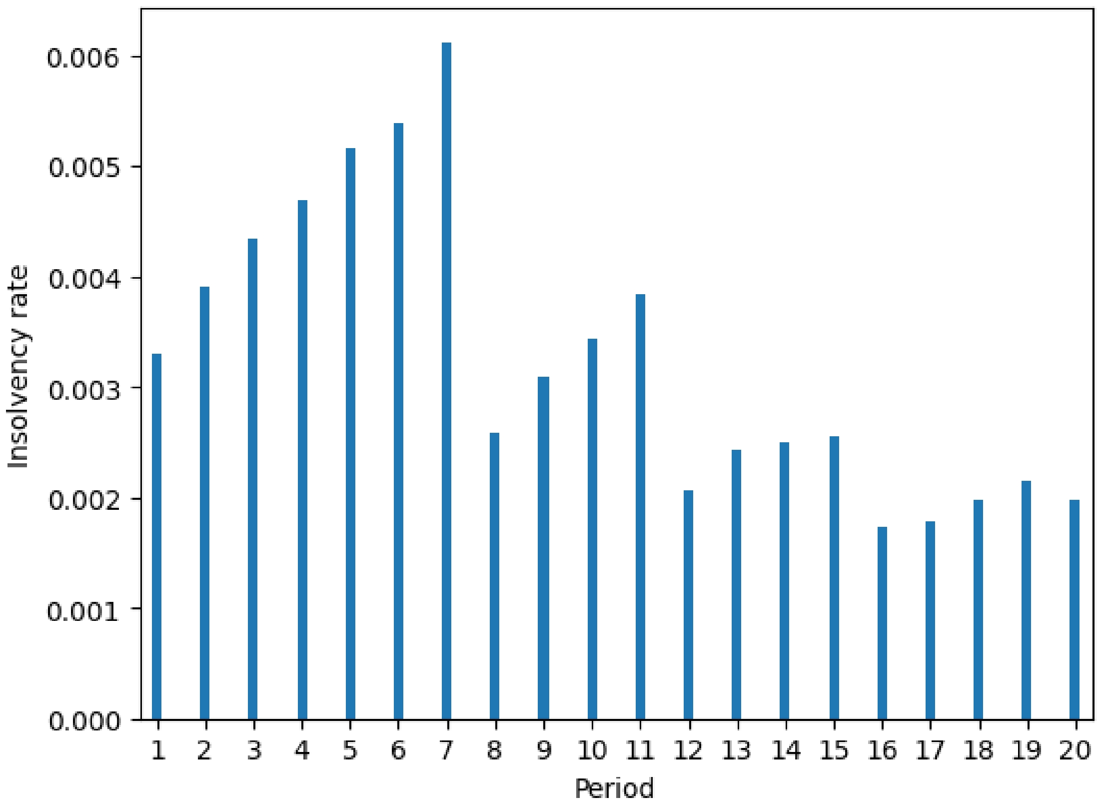


[16]:

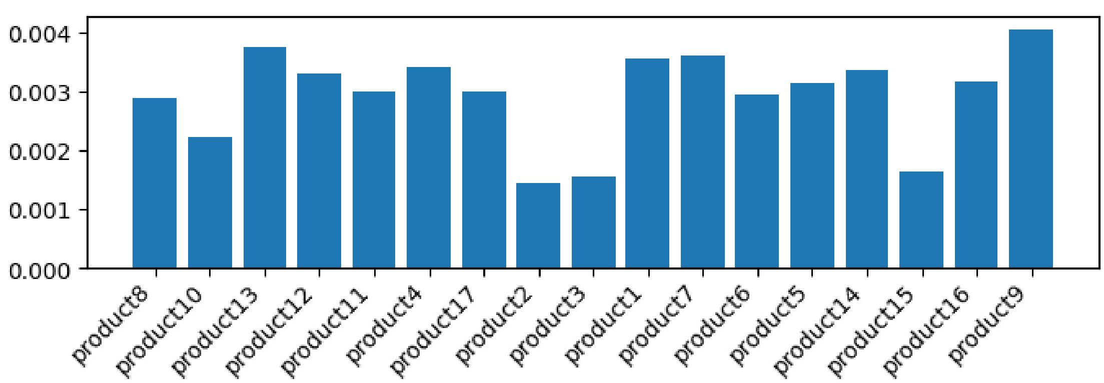
```
train.groupby('period')['repays_debt'].mean().plot.bar(width=0.2)
plt.xticks(rotation=0)
plt.xlabel('Period')
plt.ylabel('Insolvency rate')
```

[16]:

```
Text(0, 0.5, 'Insolvency rate')
```



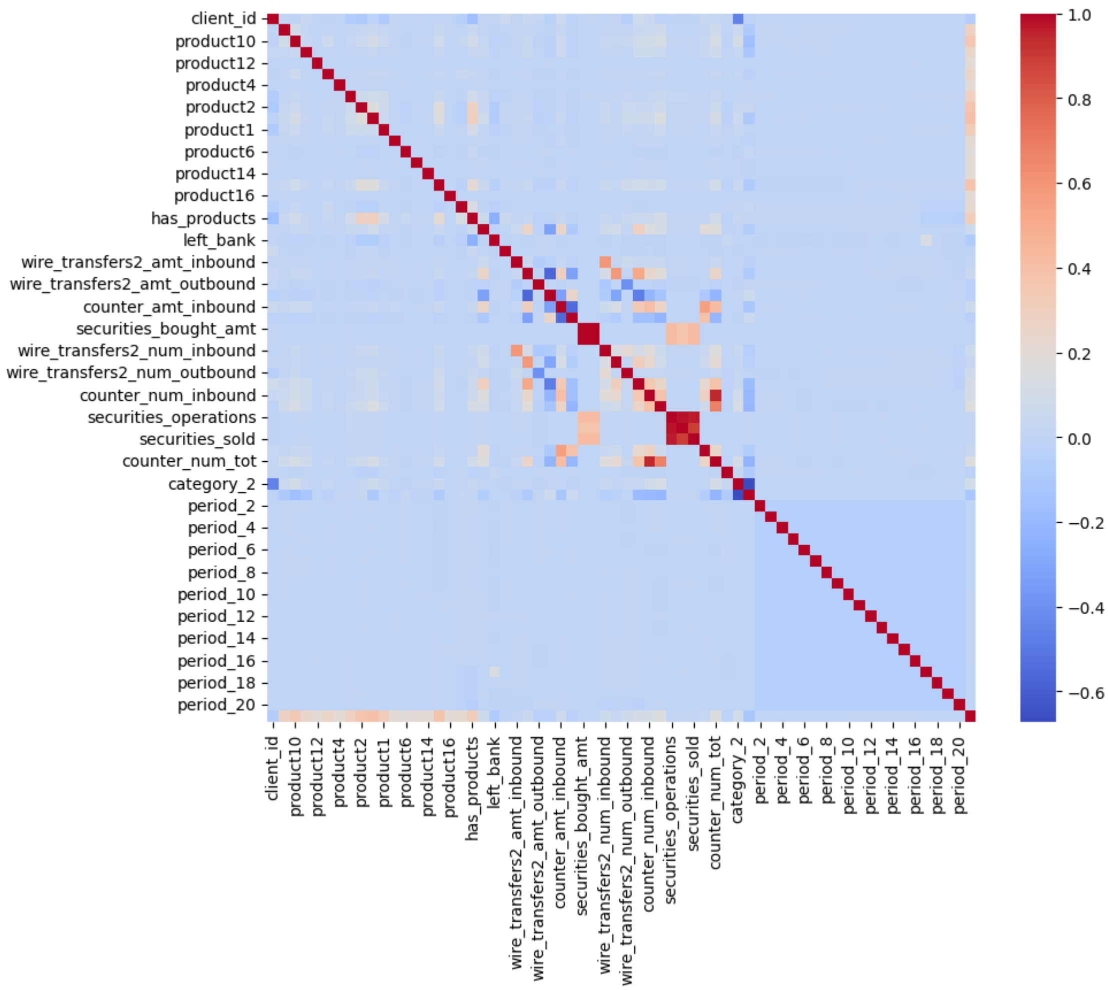
[17]:
```
means_by_products = [train.loc[train[cols[i]] == 1].repays_debt.mean()
                      for i in range(1, 18)]
plt.figure(figsize=(8,2))
plt.bar(cols[1:18], means_by_products)
plt.tick_params(axis='x', labelsize=8, pad=1)
plt.xticks(rotation=45, fontsize=10, ha='right')
plt.show()
```



[18]:
```python
corr_matrix = X_train.corr()
```

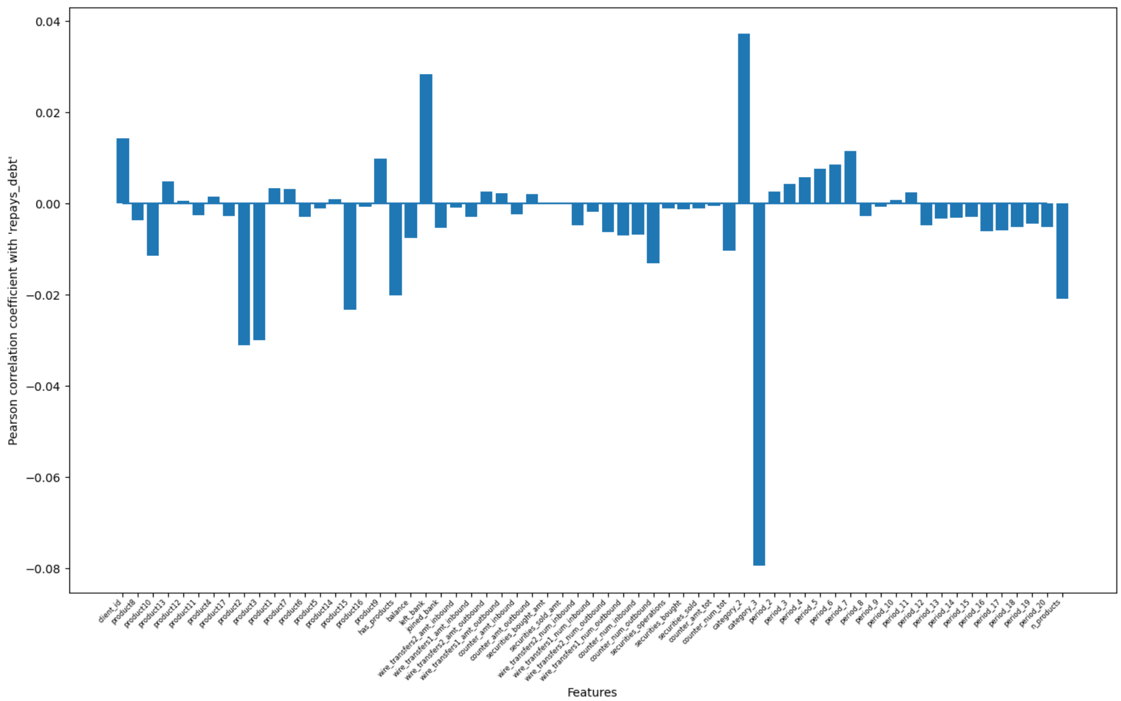
[19]:
```python
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, cmap='coolwarm',
            cbar=True, square=True)
```

[19]: <Axes: >



[20]:
```python
corrs = [np.corrcoef(X_train[col], X_train['repays_debt'])[0, 1]
 for col in X_train.columns if col != 'repays_debt']

plt.figure(figsize = (16, 9))
plt.hlines(0, 0, 61)
plt.bar(X_train.columns.drop(['repays_debt']), corrs)
plt.ylabel("Pearson correlation coefficient with 'repays_debt'")
plt.xlabel("Features")
```

```
plt.tick_params(axis='x', labelsize=8, pad=1)
plt.xticks(rotation=45, fontsize=6, ha='right')
plt.show()
```



1.1.3 Model selection

We are almost done with the preliminaries. For this task we are going to train and evaluate and select the following classification models: - Logistic Regression - SVM Classifier - KNN Classifier - Random Forest Classifier

Though, we need a resampling of the training set, because it is too large for the training, especially for KNN and RF, and furthermore is too unbalanced in the response columns, with a large proportion of zeroes. Outliers will be either removed or evaluated separately only if necessary.

Logistic Regression First of all `n_products` column we created previously is linear combination of other features, therefore the latter will be removed.

[21]:
```
dropped_cols = list(X_train.columns[:18]) + ['repays_debt']
dropped_cols
X_lr = X_train.drop(columns=dropped_cols)
y_lr = X_train['repays_debt']

rus = RandomUnderSampler(random_state=42)

X_under, y_under = rus.fit_resample(X_lr, y_lr)
```

```
X_lr_train, X_lr_test, y_lr_train, y_lr_test = train_test_split(X_under,␣
↪y_under, test_size=.2,stratify=y_under)
```

[22]:
```
lr = LogisticRegressionCV()
lr.fit(X_lr_train, y_lr_train)
```

```
c:\Users\giova\AppData\Local\Programs\Python\Python312\Lib\site-
packages\sklearn\linear_model\_logistic.py:469: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
c:\Users\giova\AppData\Local\Programs\Python\Python312\Lib\site-
packages\sklearn\linear_model\_logistic.py:469: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
c:\Users\giova\AppData\Local\Programs\Python\Python312\Lib\site-
packages\sklearn\linear_model\_logistic.py:469: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
c:\Users\giova\AppData\Local\Programs\Python\Python312\Lib\site-
packages\sklearn\linear_model\_logistic.py:469: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
```

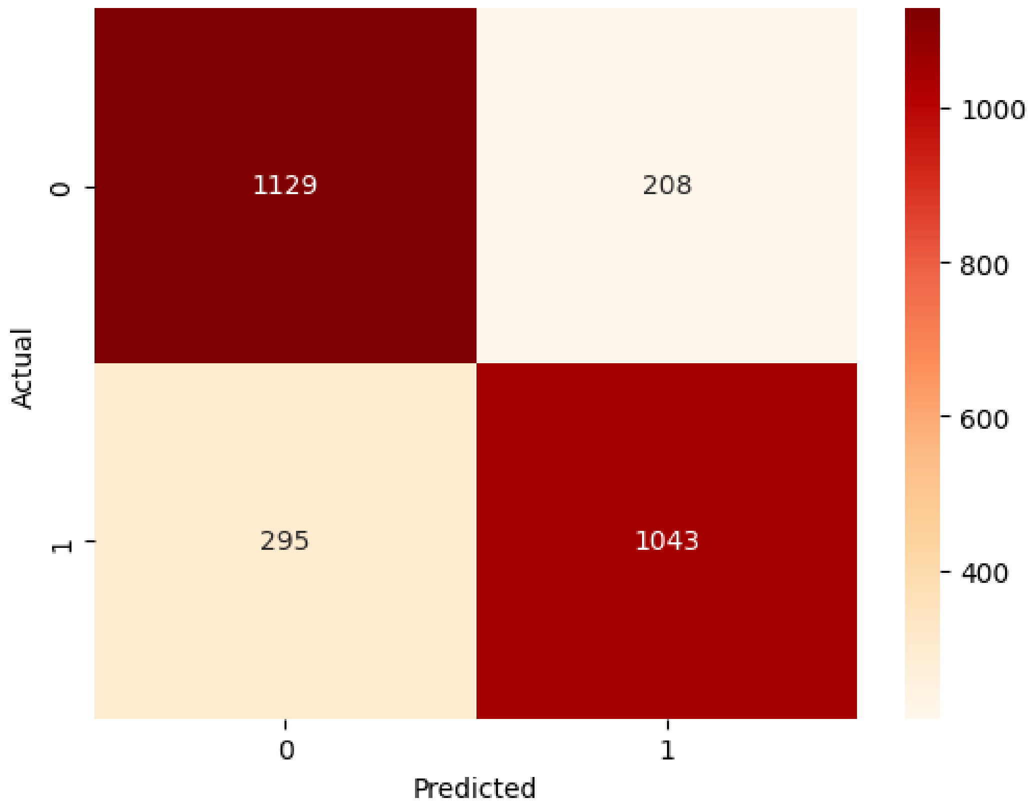
```
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
c:\Users\giova\AppData\Local\Programs\Python\Python312\Lib\site-
packages\sklearn\linear_model\_logistic.py:469: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
c:\Users\giova\AppData\Local\Programs\Python\Python312\Lib\site-
packages\sklearn\linear_model\_logistic.py:469: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

[22]:
```
LogisticRegressionCV()
```

[23]:
```python
y_lr_pred = lr.predict(X_lr_test)
best_f1_score = f1_score(y_lr_test, y_lr_pred)
print('f1 score:', best_f1_score)
cfm = confusion_matrix(y_lr_test, y_lr_pred)
sns.heatmap(cfm, cmap='OrRd', annot=True, fmt='.0f')
plt.xlabel('Predicted')
plt.ylabel('Actual')
```

```
f1 score: 0.805716492854384
```

[23]:
```
Text(50.722222222222214, 0.5, 'Actual')
```



We try to remove different features through "backward selection" and validate the results on the test set we previously extracted out of the undersampled train set.

[ ]:
```
features = list(X_lr_train.columns)
new_best_f1_score = best_f1_score
i = 1
while len(features) > 0 and new_best_f1_score >= best_f1_score:
    print("Iteration n.", i)
    best_f1_score = new_best_f1_score
    worst_feature = None
    for j, feature in enumerate(features):
        lr_tmp = LogisticRegressionCV(verbose=0)
        lr_tmp.fit(X_lr_train[features].drop(columns=feature), y_lr_train)
        y_pred_tmp = lr_tmp.predict(X_lr_test[features].drop(columns=feature))
        tmp_f1_score = f1_score(y_lr_test, y_pred_tmp)
        if tmp_f1_score >= new_best_f1_score or j == 0:
            worst_feature, new_best_f1_score = feature, tmp_f1_score
            print(f"Iteration n. {i}. Without feature '{feature}'"
                  f"-> f1 score = {tmp_f1_score}")
    if worst_feature in features and new_best_f1_score >= best_f1_score:
        features.remove(worst_feature)
```

```
    i += 1
```

[25]:
```
print(len(features))
np.array(features).tofile('features_lr.txt', sep='\n')
```

```
43
```

[26]:
```
X_lr_evaluation = X_test.drop(columns=dropped_cols)
y_lr_evaluation = X_test['repays_debt']

X_lr_submission = X_lr_evaluation.loc[X_test['repays_debt'] == '??']
y_lr_submission = y_lr_evaluation.loc[X_test['repays_debt'] == '??']
X_lr_evaluation = X_lr_evaluation.loc[X_test['repays_debt'] != '??']
y_lr_evaluation = y_lr_evaluation.loc[X_test['repays_debt'] != '??'].astype(int)
```

[27]:
```
reduced_lr = LogisticRegressionCV()
reduced_lr.fit(X_lr_train[features], y_lr_train)
y_lr_evaluation_pred = reduced_lr.predict(X_lr_evaluation[features])
print(
    "Classification Report:\n",
    classification_report(y_lr_evaluation, y_lr_evaluation_pred)
)
cfm = confusion_matrix(y_lr_evaluation, y_lr_evaluation_pred)
plt.title('Logistic Regression Confusion Matrix')
sns.heatmap(cfm, cmap='OrRd', annot=True, fmt='.0f')
plt.xlabel('Predicted')
plt.ylabel('Actual')
```

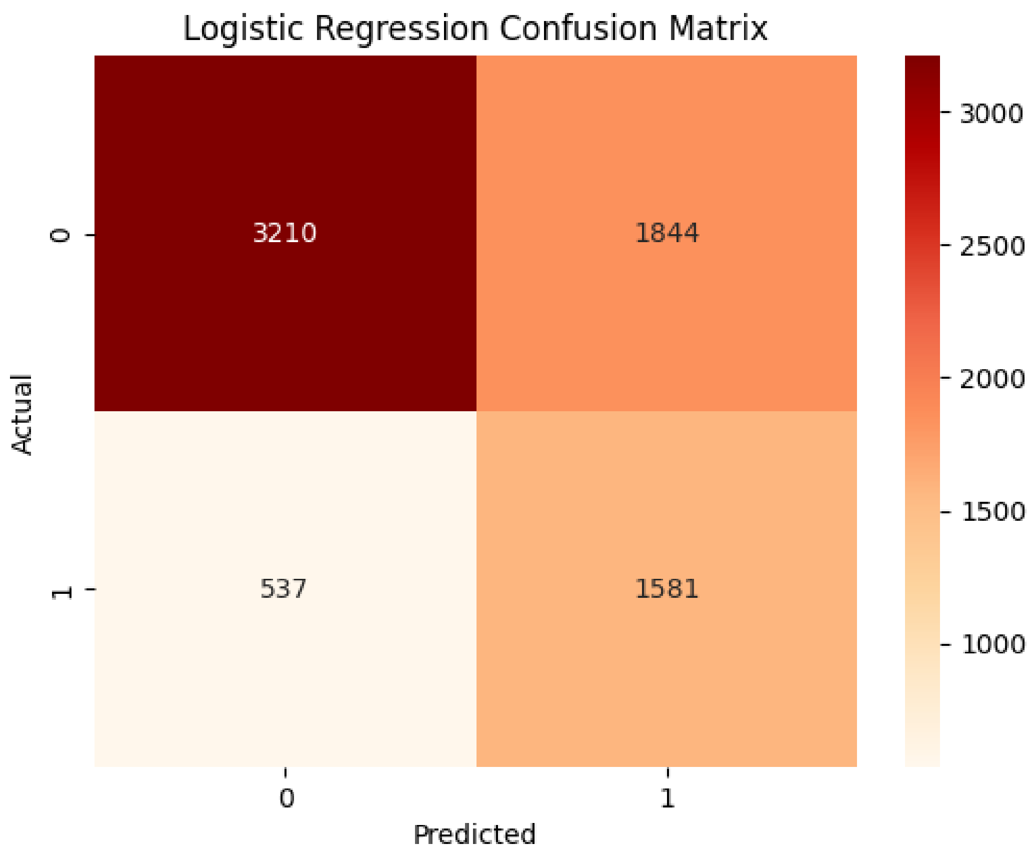
```
c:\Users\giova\AppData\Local\Programs\Python\Python312\Lib\site-
packages\sklearn\linear_model\_logistic.py:469: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
c:\Users\giova\AppData\Local\Programs\Python\Python312\Lib\site-
packages\sklearn\linear_model\_logistic.py:469: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
```

```
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
c:\Users\giova\AppData\Local\Programs\Python\Python312\Lib\site-
packages\sklearn\linear_model\_logistic.py:469: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
c:\Users\giova\AppData\Local\Programs\Python\Python312\Lib\site-
packages\sklearn\linear_model\_logistic.py:469: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
c:\Users\giova\AppData\Local\Programs\Python\Python312\Lib\site-
packages\sklearn\linear_model\_logistic.py:469: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.64      0.73      5054
           1       0.46      0.75      0.57      2118

    accuracy                           0.67      7172
   macro avg       0.66      0.69      0.65      7172
weighted avg       0.74      0.67      0.68      7172
```

```
c:\Users\giova\AppData\Local\Programs\Python\Python312\Lib\site-
packages\sklearn\linear_model\_logistic.py:469: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

[27]: Text(50.722222222222214, 0.5, 'Actual')



[28]:
```
y_lr_submission_pred = reduced_lr.predict(X_lr_submission[features])
y_lr_submission_pred.mean()
```

[28]: 0.6100278551532033

SVM Classifier We procede similarly for a Support Vector model and for the other models, but from now on we also standardize the data, whereas it was not useful for the binomial glm.

[29]:
```
X_svc = X_train.drop(columns=dropped_cols)
y_svc = X_train['repays_debt']

rus = RandomUnderSampler(random_state=42)

X_under, y_under = rus.fit_resample(X_svc, y_svc)
X_svc_train, X_svc_test, y_svc_train, y_svc_test = train_test_split(X_under,
 ↪y_under, test_size=.2,stratify=y_under)


# -- scaling data ----------------------------------------------------

scaler = StandardScaler()

X_svc_train = pd.DataFrame(scaler.fit_transform(X_svc_train), columns=X_svc.
 ↪columns)
X_svc_test = pd.DataFrame(scaler.fit_transform(X_svc_test), columns=X_svc.
 ↪columns)

X_svc_evaluation = pd.DataFrame(scaler.fit_transform(X_test.
 ↪drop(columns=dropped_cols)), columns=X_svc.columns)
y_svc_evaluation = X_test['repays_debt']

X_svc_submission = pd.DataFrame(scaler.fit_transform(X_svc_evaluation.
 ↪loc[X_test['repays_debt'] == '??']), columns=X_svc.columns)
y_svc_submission = y_svc_evaluation.loc[X_test['repays_debt'] == '??']
X_svc_evaluation = pd.DataFrame(scaler.fit_transform(X_svc_evaluation.
 ↪loc[X_test['repays_debt'] != '??']), columns=X_svc.columns)
y_svc_evaluation = y_svc_evaluation.loc[X_test['repays_debt'] != '??'].
 ↪astype(int)
#-------------------------------------------------------------------------------------
```

[30]:
```
svc = LinearSVC()
svc.fit(X_svc_train, y_svc_train)
```

```
c:\Users\giova\AppData\Local\Programs\Python\Python312\Lib\site-
packages\sklearn\svm\_classes.py:31: FutureWarning: The default value of `dual`
will change from `True` to `'auto'` in 1.5. Set the value of `dual` explicitly
to suppress the warning.
  warnings.warn(
c:\Users\giova\AppData\Local\Programs\Python\Python312\Lib\site-
packages\sklearn\svm\_base.py:1237: ConvergenceWarning: Liblinear failed to
converge, increase the number of iterations.
  warnings.warn(
```

[30]: LinearSVC()

[31]:
```
y_svc_pred = lr.predict(X_svc_test)
best_f1_score = f1_score(y_svc_test, y_svc_pred)
print('f1 score:', best_f1_score)
# cfm = confusion_matrix(y_lr_test, y_lr_pred)
# print(
#     "Classification Report:\n",
#     classification_report(y_svc_test, y_svc_pred)
# )
# sns.heatmap(cfm, cmap='OrRd', annot=True, fmt='.0f')
# plt.title('SVC Confusion Matrix')
# plt.xlabel('Predicted')
# plt.ylabel('Actual')
```

```
f1 score: 0.7832877782116361
```

[ ]:
```
# Backward selection

features = list(X_svc_train.columns)
new_best_f1_score = best_f1_score
i = 1
while len(features) > 0 and new_best_f1_score >= best_f1_score:
    print("Iteration n.", i)
    best_f1_score = new_best_f1_score
    worst_feature = None
    for j, feature in enumerate(features):
        svc_tmp = LinearSVC()
        svc_tmp.fit(X_svc_train[features].drop(columns=feature), y_svc_train)
        y_pred_tmp = svc_tmp.predict(X_svc_test[features].drop(columns=feature))
        tmp_f1_score = f1_score(y_svc_test, y_pred_tmp)
        if tmp_f1_score >= new_best_f1_score or j == 0:
            worst_feature, new_best_f1_score = feature, tmp_f1_score
            print(f"Iteration n. {i}. Without feature '{feature}'"
                  f"-> f1 score = {tmp_f1_score}")
    if worst_feature in features and new_best_f1_score >= best_f1_score:
        features.remove(worst_feature)
    i += 1
```

[33]:
```
print(len(features))
np.array(features).tofile('features_svc.txt', sep='\n')
```

```
29
```

[34]:
```
reduced_svc = LinearSVC()
reduced_svc.fit(X_svc_train[features], y_svc_train)
y_svc_evaluation_pred = reduced_svc.predict(X_svc_evaluation[features])
```

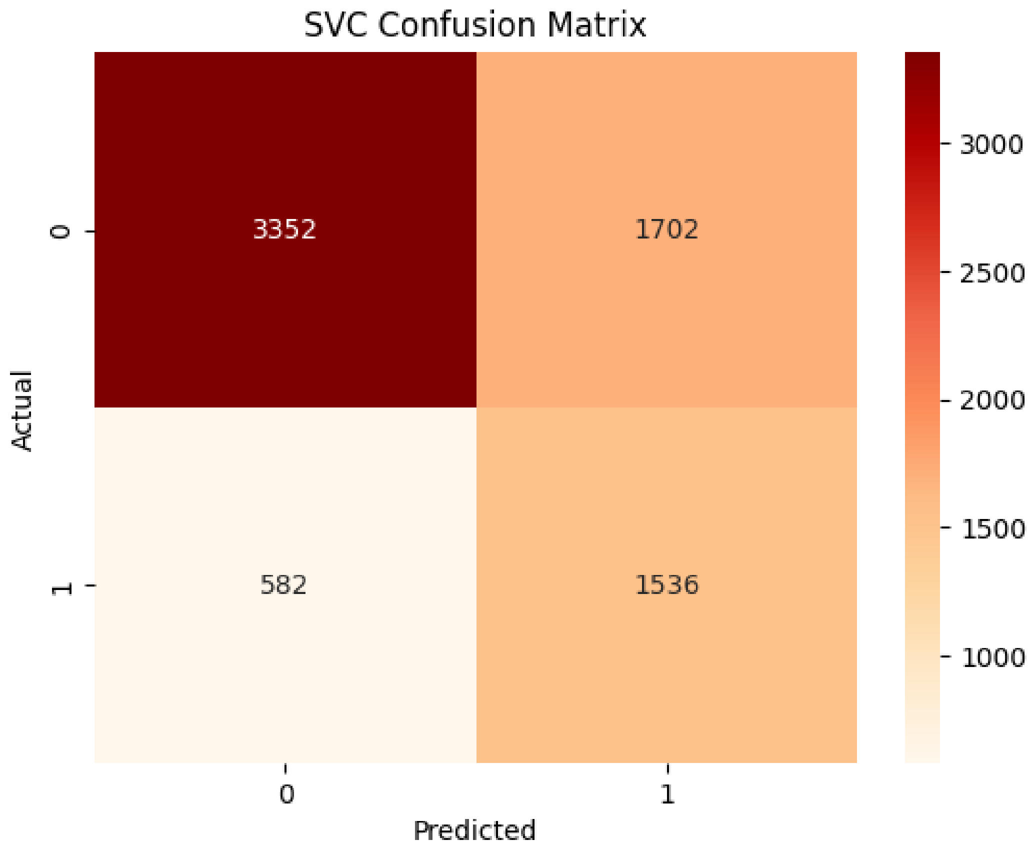
```python
print(
    "Classification Report:\n",
    classification_report(y_svc_evaluation, y_svc_evaluation_pred)
)
cfm = confusion_matrix(y_svc_evaluation, y_svc_evaluation_pred)
plt.title('SVC Confusion Matrix')
sns.heatmap(cfm, cmap='OrRd', annot=True, fmt='.0f')
plt.xlabel('Predicted')
plt.ylabel('Actual')
```

```
c:\Users\giova\AppData\Local\Programs\Python\Python312\Lib\site-
packages\sklearn\svm\_classes.py:31: FutureWarning: The default value of `dual`
will change from `True` to `'auto'` in 1.5. Set the value of `dual` explicitly
to suppress the warning.
  warnings.warn(
```

```
Classification Report:
               precision    recall  f1-score   support

           0       0.85      0.66      0.75      5054
           1       0.47      0.73      0.57      2118

    accuracy                           0.68      7172
   macro avg       0.66      0.69      0.66      7172
weighted avg       0.74      0.68      0.69      7172

```

```
c:\Users\giova\AppData\Local\Programs\Python\Python312\Lib\site-
packages\sklearn\svm\_base.py:1237: ConvergenceWarning: Liblinear failed to
converge, increase the number of iterations.
  warnings.warn(
```

[34]: Text(50.722222222222214, 0.5, 'Actual')



KNN Classifier

[35]:
```
X_knn = X_train.drop(columns=['client_id', 'repays_debt'])
y_knn = X_train['repays_debt']

rus = RandomUnderSampler(random_state=42)

X_under, y_under = rus.fit_resample(X_knn, y_knn)
X_knn_train, X_knn_test, y_knn_train, y_knn_test = train_test_split(X_under,␣
 ↪y_under, test_size=.2,stratify=y_under)


# -- scaling data -----------------------------------------------------

scaler = StandardScaler()

X_knn_train = pd.DataFrame(scaler.fit_transform(X_knn_train), columns=X_knn.
 ↪columns)
X_knn_test = pd.DataFrame(scaler.fit_transform(X_knn_test), columns=X_knn.
 ↪columns)
```

```
X_knn_evaluation = pd.DataFrame(scaler.fit_transform(X_test[X_knn.columns]),␣
↪columns=X_knn.columns)
y_knn_evaluation = X_test['repays_debt']

X_knn_submission = pd.DataFrame(scaler.fit_transform(X_knn_evaluation.
↪loc[X_test['repays_debt'] == '??']), columns=X_knn.columns)
y_knn_submission = y_knn_evaluation.loc[X_test['repays_debt'] == '??']
X_knn_evaluation = pd.DataFrame(scaler.fit_transform(X_knn_evaluation.
↪loc[X_test['repays_debt'] != '??']), columns=X_knn.columns)
y_knn_evaluation = y_knn_evaluation.loc[X_test['repays_debt'] != '??'].
↪astype(int)
#---------------------------------------------------------------------------------
```

[36]:
```
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_knn_train, y_knn_train)
```

[36]: KNeighborsClassifier()

[37]:
```
y_knn_pred = knn.predict(X_knn_test)
best_f1_score = f1_score(y_knn_test, y_knn_pred)
print('f1 score:', best_f1_score)
# cfm = confusion_matrix(y_knn_test, y_knn_pred)
# print(
#     "Classification Report:\n",
#     classification_report(y_knn_test, y_knn_pred)
# )
# sns.heatmap(cfm, cmap='OrRd', annot=True, fmt='.0f')
# plt.title('kNN Confusion Matrix')
# plt.xlabel('Predicted')
# plt.ylabel('Actual')
```

```
f1 score: 0.7656423546834505
```

[38]:
```
# Backward selection

features = list(X_knn_train.columns)
new_best_f1_score = best_f1_score
i = 1
while len(features) > 0 and new_best_f1_score >= best_f1_score:
    print("Iteration n.", i)
    best_f1_score = new_best_f1_score
    worst_feature = None
    for j, feature in enumerate(features):
        knn_tmp = KNeighborsClassifier(n_neighbors=5)
        knn_tmp.fit(X_knn_train[features].drop(columns=feature), y_knn_train)
        y_pred_tmp = knn_tmp.predict(X_knn_test[features].drop(columns=feature))
```

```
        tmp_f1_score = f1_score(y_knn_test, y_pred_tmp)
        if tmp_f1_score >= new_best_f1_score or j == 0:
            worst_feature, new_best_f1_score = feature, tmp_f1_score
            print(f"Iteration n. {i}. Without feature '{feature}'"
                  f"-> f1 score = {tmp_f1_score}")
    if worst_feature in features and new_best_f1_score >= best_f1_score:
        features.remove(worst_feature)
    i += 1
```

```
Iteration n. 1
Iteration n. 1. Without feature 'product8'-> f1 score = 0.7756906077348066
Iteration n. 1. Without feature 'product13'-> f1 score = 0.7771217712177122
Iteration n. 1. Without feature 'product12'-> f1 score = 0.7774502579218865
Iteration n. 2
Iteration n. 2. Without feature 'product8'-> f1 score = 0.7881918819188192
Iteration n. 3
Iteration n. 3. Without feature 'product10'-> f1 score = 0.7775330396475771
Iteration n. 3. Without feature 'product13'-> f1 score = 0.7851361295069904
Iteration n. 3. Without feature 'product17'-> f1 score = 0.7855297157622739
Iteration n. 3. Without feature 'product1'-> f1 score = 0.7899343544857768
Iteration n. 4
Iteration n. 4. Without feature 'product10'-> f1 score = 0.7922740524781341
Iteration n. 5
Iteration n. 5. Without feature 'product13'-> f1 score = 0.7895310796074155
Iteration n. 5. Without feature 'product11'-> f1 score = 0.7901687454145268
Iteration n. 5. Without feature 'product5'-> f1 score = 0.7906123945727906
Iteration n. 5. Without feature 'product14'-> f1 score = 0.7924253459577567
Iteration n. 5. Without feature 'counter_amt_inbound'-> f1 score =
0.7932920160408312
Iteration n. 5. Without feature 'counter_amt_tot'-> f1 score =
0.7932920160408312
Iteration n. 5. Without feature 'period_2'-> f1 score = 0.7950219619326501
Iteration n. 5. Without feature 'period_16'-> f1 score = 0.7963436928702011
Iteration n. 6
Iteration n. 6. Without feature 'product13'-> f1 score = 0.7928649435748089
Iteration n. 6. Without feature 'balance'-> f1 score = 0.7936857562408223
Iteration n. 6. Without feature 'wire_transfers2_amt_inbound'-> f1 score =
0.7960526315789473
Iteration n. 6. Without feature 'wire_transfers1_amt_inbound'-> f1 score =
0.7963436928702011
Iteration n. 6. Without feature 'wire_transfers2_amt_outbound'-> f1 score =
0.7963436928702011
Iteration n. 6. Without feature 'counter_amt_inbound'-> f1 score =
0.797366495976591
Iteration n. 6. Without feature 'counter_amt_tot'-> f1 score = 0.797366495976591
Iteration n. 7
Iteration n. 7. Without feature 'product13'-> f1 score = 0.7933042212518195
```

```
Iteration n. 7. Without feature 'balance'-> f1 score = 0.7948623853211009
Iteration n. 7. Without feature 'wire_transfers2_amt_inbound'-> f1 score =
0.7970749542961609
Iteration n. 7. Without feature 'wire_transfers1_amt_inbound'-> f1 score =
0.797366495976591
Iteration n. 7. Without feature 'wire_transfers2_amt_outbound'-> f1 score =
0.797366495976591
Iteration n. 7. Without feature 'counter_amt_inbound'-> f1 score =
0.7981014968966776
Iteration n. 8
Iteration n. 8. Without feature 'product13'-> f1 score = 0.7918486171761281
Iteration n. 8. Without feature 'balance'-> f1 score = 0.7950128346167951
Iteration n. 8. Without feature 'wire_transfers2_amt_inbound'-> f1 score =
0.7978102189781022
Iteration n. 8. Without feature 'wire_transfers1_amt_inbound'-> f1 score =
0.7981014968966776
Iteration n. 8. Without feature 'wire_transfers2_amt_outbound'-> f1 score =
0.7981014968966776
Iteration n. 8. Without feature 'securities_bought_amt'-> f1 score =
0.7981014968966776
Iteration n. 8. Without feature 'securities_sold_amt'-> f1 score =
0.7981014968966776
Iteration n. 8. Without feature 'securities_operations'-> f1 score =
0.7981014968966776
Iteration n. 8. Without feature 'securities_bought'-> f1 score =
0.7981014968966776
Iteration n. 8. Without feature 'securities_sold'-> f1 score =
0.7981014968966776
Iteration n. 9
Iteration n. 9. Without feature 'product13'-> f1 score = 0.7918486171761281
Iteration n. 9. Without feature 'balance'-> f1 score = 0.7950128346167951
Iteration n. 9. Without feature 'wire_transfers2_amt_inbound'-> f1 score =
0.7978102189781022
Iteration n. 9. Without feature 'wire_transfers1_amt_inbound'-> f1 score =
0.7981014968966776
Iteration n. 9. Without feature 'wire_transfers2_amt_outbound'-> f1 score =
0.7981014968966776
Iteration n. 9. Without feature 'securities_bought_amt'-> f1 score =
0.7981014968966776
Iteration n. 9. Without feature 'securities_sold_amt'-> f1 score =
0.7981014968966776
Iteration n. 9. Without feature 'securities_operations'-> f1 score =
0.7981014968966776
Iteration n. 9. Without feature 'securities_bought'-> f1 score =
0.7981014968966776
Iteration n. 10
Iteration n. 10. Without feature 'product13'-> f1 score = 0.7918486171761281
Iteration n. 10. Without feature 'balance'-> f1 score = 0.7950128346167951
```

```
Iteration n. 10. Without feature 'wire_transfers2_amt_inbound'-> f1 score =
0.7978102189781022
Iteration n. 10. Without feature 'wire_transfers1_amt_inbound'-> f1 score =
0.7981014968966776
Iteration n. 10. Without feature 'wire_transfers2_amt_outbound'-> f1 score =
0.7981014968966776
Iteration n. 10. Without feature 'securities_bought_amt'-> f1 score =
0.7981014968966776
Iteration n. 10. Without feature 'securities_sold_amt'-> f1 score =
0.7981014968966776
Iteration n. 10. Without feature 'securities_operations'-> f1 score =
0.7981014968966776
Iteration n. 11
Iteration n. 11. Without feature 'product13'-> f1 score = 0.7918486171761281
Iteration n. 11. Without feature 'balance'-> f1 score = 0.7950128346167951
Iteration n. 11. Without feature 'wire_transfers2_amt_inbound'-> f1 score =
0.7978102189781022
Iteration n. 11. Without feature 'wire_transfers1_amt_inbound'-> f1 score =
0.7981014968966776
Iteration n. 11. Without feature 'wire_transfers2_amt_outbound'-> f1 score =
0.7981014968966776
Iteration n. 11. Without feature 'securities_bought_amt'-> f1 score =
0.7981014968966776
Iteration n. 11. Without feature 'securities_sold_amt'-> f1 score =
0.7981014968966776
Iteration n. 12
Iteration n. 12. Without feature 'product13'-> f1 score = 0.7918486171761281
Iteration n. 12. Without feature 'balance'-> f1 score = 0.7950128346167951
Iteration n. 12. Without feature 'wire_transfers2_amt_inbound'-> f1 score =
0.7978102189781022
Iteration n. 12. Without feature 'wire_transfers1_amt_inbound'-> f1 score =
0.7981014968966776
Iteration n. 12. Without feature 'wire_transfers2_amt_outbound'-> f1 score =
0.7981014968966776
Iteration n. 13
Iteration n. 13. Without feature 'product13'-> f1 score = 0.7918486171761281
Iteration n. 13. Without feature 'balance'-> f1 score = 0.7950128346167951
Iteration n. 13. Without feature 'wire_transfers2_amt_inbound'-> f1 score =
0.7978102189781022
Iteration n. 13. Without feature 'wire_transfers1_amt_inbound'-> f1 score =
0.7981014968966776
Iteration n. 14
Iteration n. 14. Without feature 'product13'-> f1 score = 0.7921368765926465
Iteration n. 14. Without feature 'balance'-> f1 score = 0.7950128346167951
Iteration n. 14. Without feature 'wire_transfers2_amt_inbound'-> f1 score =
0.7978102189781022
Iteration n. 14. Without feature 'period_12'-> f1 score = 0.7979539641943734
```

[39]:
```
print(len(features))
np.array(features).tofile('features_knn.txt', sep='\n')
```

```
49
```

[40]:
```
# parameter 'n_neighbors' tuning

for n in range(3,20):
    knn_tmp = KNeighborsClassifier(n_neighbors=n)
    knn_tmp.fit(X_knn_train[features], y_knn_train)
    scores = cross_val_score(knn_tmp, X_knn_train[features], y_knn_train,
                             cv=5, scoring='f1')
    print(f"n_neighbors = {n}: cross-val f1 scores: {scores} (mean = {scores.
 ↪mean():.3f})")
```

```
n_neighbors = 3: cross-val f1 scores: [0.74104683 0.76049838 0.74187558
0.74728389 0.73471264] (mean = 0.745)
n_neighbors = 4: cross-val f1 scores: [0.71036585 0.72764435 0.7034914
0.72105263 0.70510204] (mean = 0.714)
n_neighbors = 5: cross-val f1 scores: [0.7588717  0.78244804 0.76851421
0.77599244 0.75428043] (mean = 0.768)
n_neighbors = 6: cross-val f1 scores: [0.73932253 0.76892822 0.74962519
0.74300254 0.73410115] (mean = 0.747)
n_neighbors = 7: cross-val f1 scores: [0.76586034 0.79172414 0.77808728
0.78497653 0.76658933] (mean = 0.777)
n_neighbors = 8: cross-val f1 scores: [0.75988428 0.7751938  0.76568627 0.766
0.74498287] (mean = 0.762)
n_neighbors = 9: cross-val f1 scores: [0.78169336 0.79212454 0.78737233
0.78457197 0.77262288] (mean = 0.784)
n_neighbors = 10: cross-val f1 scores: [0.76997113 0.7806049  0.77027683
0.77113198 0.75855422] (mean = 0.770)
n_neighbors = 11: cross-val f1 scores: [0.79212454 0.79559026 0.78591288
0.79063232 0.77413479] (mean = 0.788)
n_neighbors = 12: cross-val f1 scores: [0.78104265 0.78571429 0.77600387
0.77234568 0.77216397] (mean = 0.777)
n_neighbors = 13: cross-val f1 scores: [0.79890561 0.78949793 0.78422274
0.78770686 0.7826484 ] (mean = 0.789)
n_neighbors = 14: cross-val f1 scores: [0.7884797  0.78178368 0.77627772
0.77194703 0.77846009] (mean = 0.779)
n_neighbors = 15: cross-val f1 scores: [0.80018165 0.79207008 0.78731343
0.79849341 0.78229119] (mean = 0.792)
n_neighbors = 16: cross-val f1 scores: [0.79737336 0.78807947 0.7826506
0.78492025 0.78215962] (mean = 0.787)
n_neighbors = 17: cross-val f1 scores: [0.80346557 0.80183486 0.78475128
0.79607109 0.78467153] (mean = 0.794)
n_neighbors = 18: cross-val f1 scores: [0.79850397 0.79584121 0.773549
0.78457831 0.77610536] (mean = 0.786)
n_neighbors = 19: cross-val f1 scores: [0.79981802 0.80183908 0.78418605
```

```
0.79512424 0.78300594] (mean = 0.793)
```

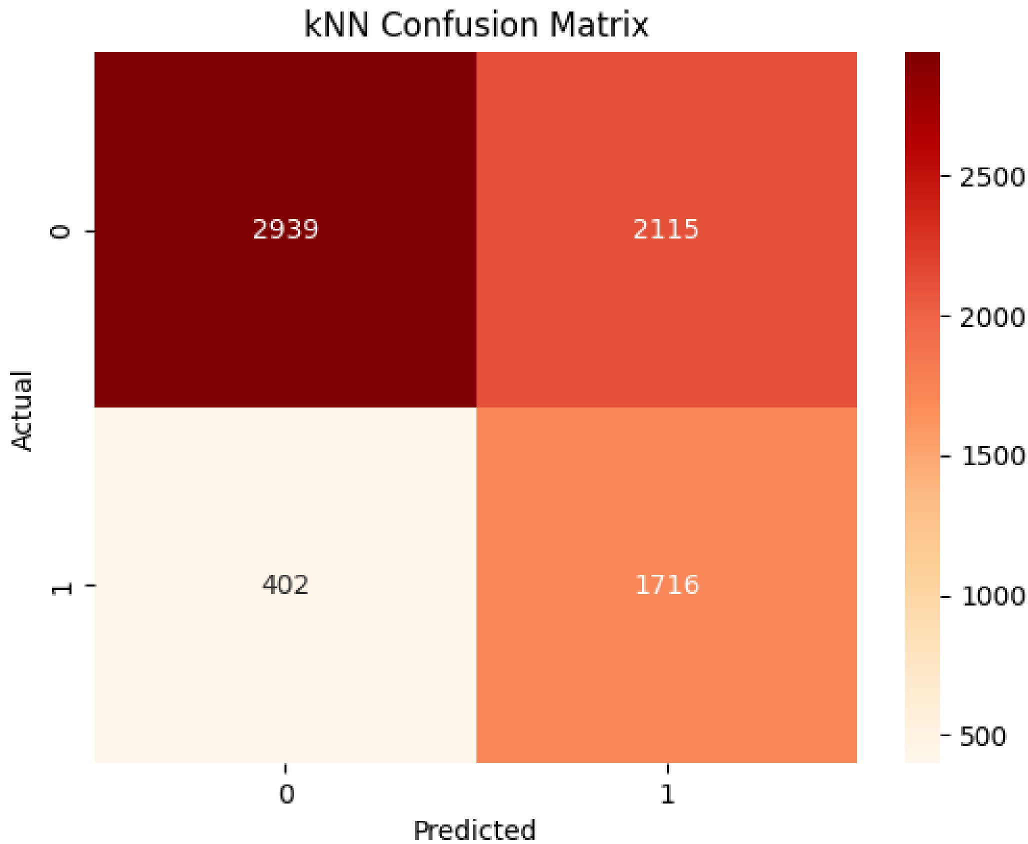
[41]:
```python
# 17 is the best parameter after cross validation in the train set
knn = KNeighborsClassifier(n_neighbors=17)
knn.fit(X_knn_train[features], y_knn_train)
y_knn_pred = knn.predict(X_knn_test[features])
best_f1_score = f1_score(y_knn_test, y_knn_pred)
print('f1 score:', best_f1_score)
```

```
f1 score: 0.8069919883466861
```

[42]:
```python
reduced_knn = KNeighborsClassifier(n_neighbors=17)
reduced_knn.fit(X_knn_train[features], y_knn_train)
y_knn_evaluation_pred = reduced_knn.predict(X_knn_evaluation[features])
print(
    "Classification Report:\n",
    classification_report(y_knn_evaluation, y_knn_evaluation_pred)
)
cfm = confusion_matrix(y_knn_evaluation, y_knn_evaluation_pred)
plt.title('kNN Confusion Matrix')
sns.heatmap(cfm, cmap='OrRd', annot=True, fmt='.0f')
plt.xlabel('Predicted')
plt.ylabel('Actual')
```

```
Classification Report:
               precision    recall  f1-score   support

           0       0.88      0.58      0.70      5054
           1       0.45      0.81      0.58      2118

    accuracy                           0.65      7172
   macro avg       0.66      0.70      0.64      7172
weighted avg       0.75      0.65      0.66      7172
```

[42]:
```
Text(50.722222222222214, 0.5, 'Actual')
```



Random Forest

[43]:
```
X_rf = X_train.drop(columns=['client_id', 'repays_debt'])
y_rf = X_train['repays_debt']

rus = RandomUnderSampler(random_state=42)

X_under, y_under = rus.fit_resample(X_rf, y_rf)
X_rf_train, X_rf_test, y_rf_train, y_rf_test = train_test_split(X_under,␣
↪y_under, test_size=.2,stratify=y_under)

X_rf_evaluation = X_test[X_rf.columns]
y_rf_evaluation = X_test['repays_debt']

X_rf_submission = X_rf_evaluation.loc[X_test['repays_debt'] == '??']
y_rf_submission = y_rf_evaluation.loc[X_test['repays_debt'] == '??']
X_rf_evaluation = X_rf_evaluation.loc[X_test['repays_debt'] != '??']
y_rf_evaluation = y_rf_evaluation.loc[X_test['repays_debt'] != '??'].astype(int)
```

[44]:
```
rf = RandomForestClassifier(n_estimators=101, random_state=42, n_jobs=-1)
rf.fit(X_rf_train, y_rf_train)
```

[44]:
```
RandomForestClassifier(n_estimators=101, n_jobs=-1, random_state=42)
```

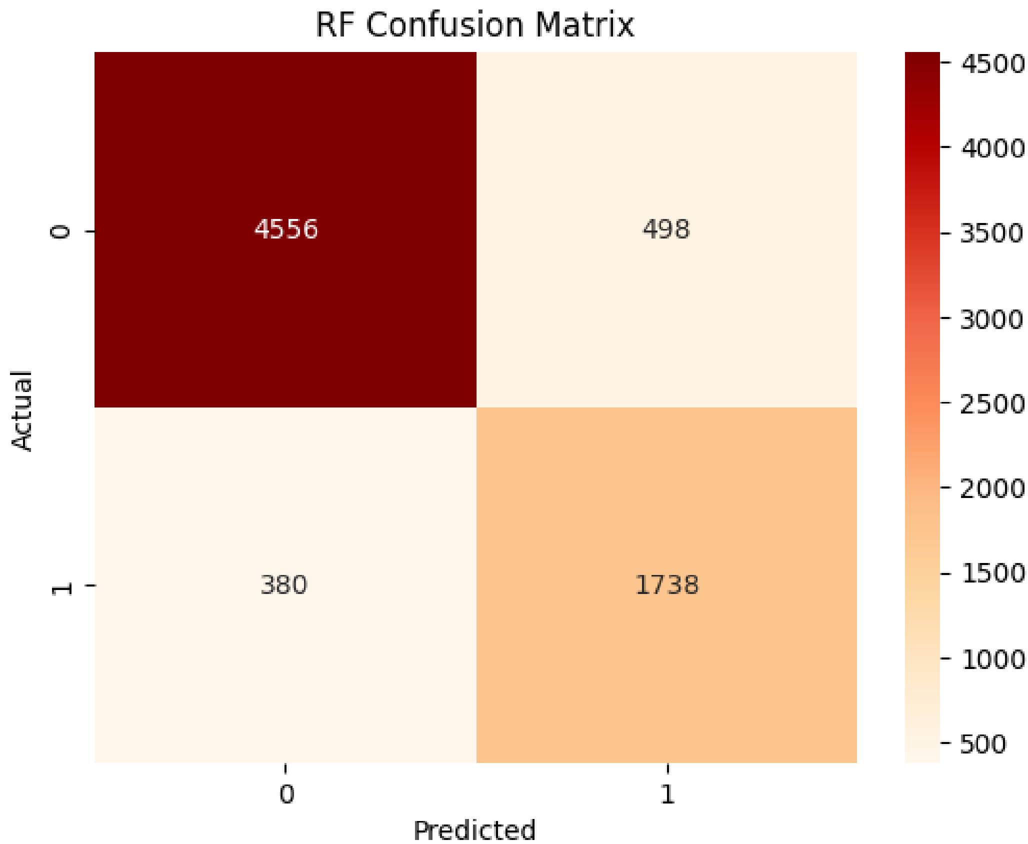
[45]:
```
y_rf_pred = rf.predict(X_rf_test)
best_f1_score = f1_score(y_rf_test, y_rf_pred)
print('f1 score:', best_f1_score)

# cfm = confusion_matrix(y_rf_test, y_rf_pred)
# print(
#     "Classification Report:\n",
#     classification_report(y_rf_test, y_rf_pred)
# )
# sns.heatmap(cfm, cmap='OrRd', annot=True, fmt='.0f')
# plt.title('RF Confusion Matrix')
# plt.xlabel('Predicted')
# plt.ylabel('Actual')
```

```
f1 score: 0.8538812785388128
```

[46]:
```
y_rf_evaluation_pred = rf.predict(X_rf_evaluation)
print(
    "Classification Report:\n",
    classification_report(y_rf_evaluation, y_rf_evaluation_pred)
)
cfm = confusion_matrix(y_rf_evaluation, y_rf_evaluation_pred)
plt.title('RF Confusion Matrix')
sns.heatmap(cfm, cmap='OrRd', annot=True, fmt='.0f')
plt.xlabel('Predicted')
plt.ylabel('Actual')
```

```
Classification Report:
               precision    recall  f1-score   support

           0       0.92      0.90      0.91      5054
           1       0.78      0.82      0.80      2118

    accuracy                           0.88      7172
   macro avg       0.85      0.86      0.86      7172
weighted avg       0.88      0.88      0.88      7172
```

[46]:
```
Text(50.722222222222214, 0.5, 'Actual')
```



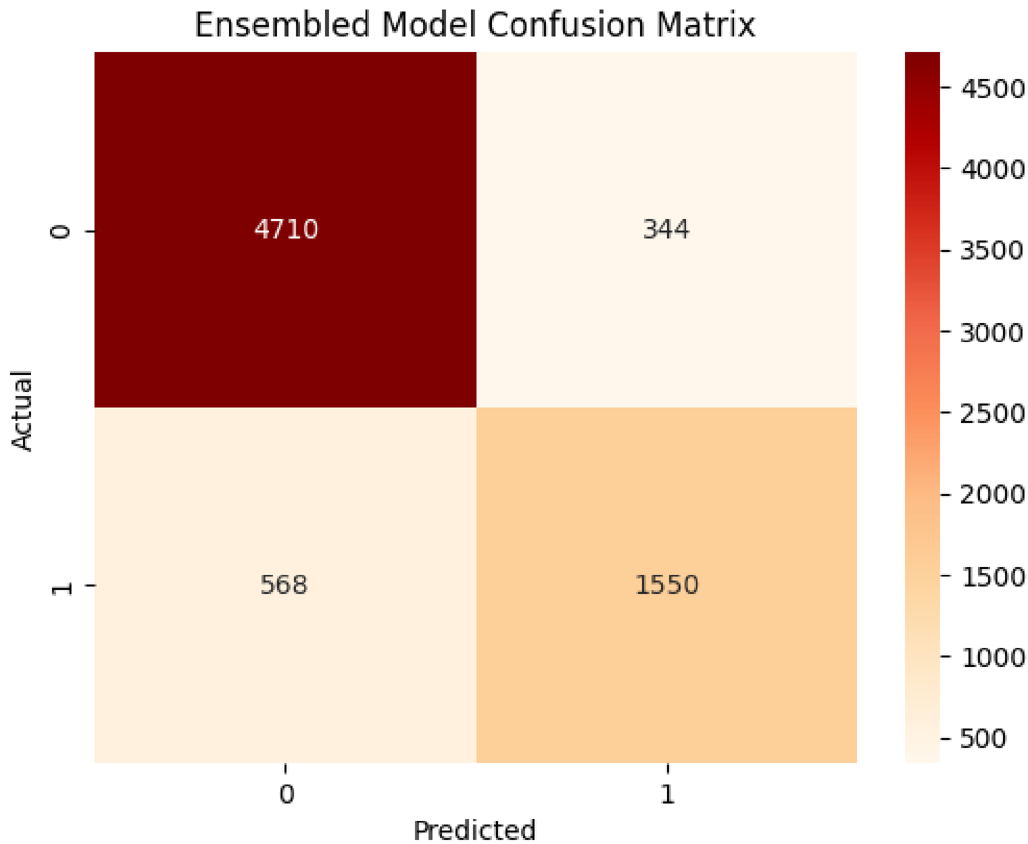
Ensembled model Random forest looks the model that gives better results, nevertheless it is good practice to include democratically all the models, through a weighted mean of all predictions.

[47]:
```
f1_scores = [
    f1_score(y_lr_test, y_lr_pred),
    f1_score(y_svc_test, y_svc_pred),
    f1_score(y_knn_test, y_knn_pred),
    f1_score(y_rf_test, y_rf_pred)
]

def ensemble(*y_preds, weights: list[float] = 1) -> np.array:
    if isinstance(weights, int):
        weights = np.array([weights]*len(y_preds))
    return (np.column_stack(y_preds) @ weights) / sum(weights)

test_preds = [y_lr_pred, y_svc_pred, y_knn_pred, y_rf_pred]
evaluation_preds = [y_lr_evaluation_pred, y_svc_evaluation_pred,
                    y_knn_evaluation_pred, y_rf_evaluation_pred]
```

[48]:
```
y_evaluation = y_rf_evaluation
# print(pd.DataFrame(
#     {'answer': ensemble(*evaluation_preds, weights=f1_scores)}).
 ↪value_counts().sort_index())
y_ensemble_evaluation_pred = ensemble(*evaluation_preds, weights=f1_scores) > 0.
 ↪75
print(
    "Classification Report:\n",
    classification_report(y_evaluation, y_ensemble_evaluation_pred)
)
cfm = confusion_matrix(y_evaluation, y_ensemble_evaluation_pred)
plt.title('Ensembled Model Confusion Matrix')
sns.heatmap(cfm, cmap='OrRd', annot=True, fmt='.0f')
plt.xlabel('Predicted')
plt.ylabel('Actual')
```

```
Classification Report:
               precision    recall  f1-score   support

           0       0.89      0.93      0.91      5054
           1       0.82      0.73      0.77      2118

    accuracy                           0.87      7172
   macro avg       0.86      0.83      0.84      7172
weighted avg       0.87      0.87      0.87      7172
```

[48]:
```
Text(50.722222222222214, 0.5, 'Actual')
```



we claim a costumer to be in the class '1' when all four models give that outcome, with pseudo-random exceptions of three voting models due to numerical machine calculation approximations (this is a very rough, yet deliberate approach). This choice drops dramatically the False positive rate, but tends to increase the number of false negatives.

1.1.4 Conclusion of Task 1

We adopted some pre-processing where necessary, but in the end the sole Random Forest Classification model would have performed relatively well, almost better than the mixed model, but the results with these basic ML models are quite satisfectory for the sake of the competition, although much improvable.