Supervised Machine Learning, Classification

Summary

The dataset for this project concerns about the german credit loan history. It describes various parameters that make it possible to characterize a loan request, as well as its final eligibility.
In this case, the task will be to predict the Creditability of the loan request using the given data. Therefore, focusing will be given on the interpretability of the model and the accuracy of the prediction.

The dataset has 1000 instances and 21 attributes.

Let's take a more detailed look at the attributes:

- 'Creditability' <- the label to be predicted
- 'Account Balance'
- 'Duration of Credit (month)'
- 'Payment Status of Previous Credit'
- 'Purpose'
- 'Credit Amount'
- 'Value Savings/Stocks'
- 'Length of current employment'
- 'Instalment per cent'
- 'Sex & Marital Status'
- 'Guarantors'
- 'Duration in Current address'
- 'Most valuable available asset'
- 'Age (years)'
- 'Concurrent Credits'
- 'Type of apartment'
- 'No of Credits at this Bank'
- 'Occupation'
- 'No of dependents'
- 'Telephone'
- 'Foreign Worker'

Since the goal of this project is to predict the final eligibility of the loan request, the 'Creditability' column will be the label of the dataset.

In [ ]:
```
# Import libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import KFold, cross_val_predict, train_test_split, Grid
from sklearn.pipeline import Pipeline
from sklearn.inspection import permutation_importance
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier,export_text
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, accuracy_score, precision_recall_
from sklearn.metrics import recall_score, roc_auc_score, auc, precision_recall_fscor
from imblearn.over_sampling import SMOTE
```

In [ ]:

```
# Mute the warnings
import warnings
warnings.filterwarnings('ignore')
```

In [ ]:

```
data = pd.read_csv('./german_credit.csv', sep=',')
data.shape
```

Out[ ]:

```
(1000, 21)
```

In [ ]:

```
data.head()
```

Out[ ]:

	Creditability	Account Balance	Duration of Credit (month)	Payment Status of Previous Credit	Purpose	Credit Amount	Value Savings/Stocks	Length of current employment	Insta p
0	1	1	18	4	2	1049	1	2	
1	1	1	9	4	0	2799	1	3	
2	1	2	12	2	9	841	2	4	
3	1	1	12	4	0	2122	1	3	
4	1	1	12	4	0	2171	1	3	

5 rows × 21 columns

In [ ]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 21 columns):
 #   Column                             Non-Null Count  Dtype
---  ------                             --------------  -----
 0   Creditability                      1000 non-null   int64
 1   Account Balance                    1000 non-null   int64
 2   Duration of Credit (month)         1000 non-null   int64
 3   Payment Status of Previous Credit  1000 non-null   int64
 4   Purpose                            1000 non-null   int64
 5   Credit Amount                      1000 non-null   int64
 6   Value Savings/Stocks               1000 non-null   int64
 7   Length of current employment       1000 non-null   int64
 8   Instalment per cent                1000 non-null   int64
 9   Sex & Marital Status               1000 non-null   int64
 10  Guarantors                         1000 non-null   int64
 11  Duration in Current address        1000 non-null   int64
 12  Most valuable available asset      1000 non-null   int64
 13  Age (years)                        1000 non-null   int64
 14  Concurrent Credits                 1000 non-null   int64
```

```
 15  Type of apartment                 1000 non-null   int64
 16  No of Credits at this Bank        1000 non-null   int64
 17  Occupation                        1000 non-null   int64
 18  No of dependents                  1000 non-null   int64
 19  Telephone                         1000 non-null   int64
 20  Foreign Worker                    1000 non-null   int64
dtypes: int64(21)
memory usage: 164.2 KB
```

In [ ]:
```
# No null values
data.isnull().sum()
```

Out[ ]:
```
Creditability                          0
Account Balance                        0
Duration of Credit (month)             0
Payment Status of Previous Credit      0
Purpose                                0
Credit Amount                          0
Value Savings/Stocks                   0
Length of current employment           0
Instalment per cent                    0
Sex & Marital Status                   0
Guarantors                             0
Duration in Current address            0
Most valuable available asset          0
Age (years)                            0
Concurrent Credits                     0
Type of apartment                      0
No of Credits at this Bank             0
Occupation                             0
No of dependents                       0
Telephone                              0
Foreign Worker                         0
dtype: int64
```

There are no null values and every feature is numerical (int64)

In [ ]:
```
data.nunique()
```

Out[ ]:
```
Creditability                          2
Account Balance                        4
Duration of Credit (month)            33
Payment Status of Previous Credit      5
Purpose                               10
Credit Amount                        923
Value Savings/Stocks                   5
Length of current employment           5
Instalment per cent                    4
Sex & Marital Status                   4
Guarantors                             3
Duration in Current address            4
Most valuable available asset          4
Age (years)                           53
Concurrent Credits                     3
Type of apartment                      3
No of Credits at this Bank             4
Occupation                             4
No of dependents                       2
Telephone                              2
Foreign Worker                         2
dtype: int64
```

In [ ]:
```
for col in data:
    print(col + ': ')
    print(data[col].unique())
```

```
Creditability:
[1 0]
Account Balance:
[1 2 4 3]
Duration of Credit (month):
[18  9 12 10  8  6 24 11 30 48 36 15 42 21 27 33 28  4 47 14 39 60  5 22
 54 13 16  7 20 26 45 72 40]
Payment Status of Previous Credit:
[4 2 3 0 1]
Purpose:
[ 2  0  9  3  1 10  5  4  6  8]
Credit Amount:
[ 1049  2799   841  2122  2171  2241  3398  1361  1098  3758  3905  6187
  1957  7582  1936  2647  3939  3213  2337  7228  3676  3124  2384  1424
  4716  4771   652  1154  3556  4796  3017  3535  6614  1376  1721   860
  1495  1934  3378  3868   996  1755  1028  2825  1239  1216  1258  1864
  1474  1382   640  3919  1224  2331  6313   385  1655  1053  3160  3079
  1163  2679  3578 10875  1344  1237  3077  2284  1567  2032  2745  1867
  2299   929  3399  2030  3275  1940  1602  1979  2022  3342  5866  2360
  1520  3651  2346  4454   666  1965  1995  2991  4221  1364  6361  4526
  3573  4455  2136  5954  3777   806  4712  7432  1851  1393  1412  1473
  1533  2012  3959   428  2366   763  3976  6260  1919  2603   936  3062
  4795  5842  2063  1459  1213  5103   874  2978  1820  2872  1925  2515
  2116  1453  1543  1318  2325   932  3148  3835  3832  5084  2406  2394
  2476  2964  1262  1542  1743   409  8858  3512  1158  2684  1498  6416
  3617  1291  1275  3972  3343   392  2134  5771  2788  5848  1228  1297
  1552  1963  3235  4139  1804  1950 12749  1236  1055  8072  2831  1449
  5742  2390  3430  2273  2923  1901  3711  8487  2255  7253  6761  1817
  2141  3609  2333  7824  1445  7721  3763  4439  1107  1444 12169  2753
  1494  2828  2483  1299  1549  3949  2901   709 10722  1287  3656  4679
  8613  2659  1516  4380   802  1572  3566  1278   426  8588  3857   685
  1603   601  2569  1316 10366  1568   629  1750  3488  1800  4151  2631
  5248  2899  6204   804  3595  5711  2687  3643  2146  2315  3448  2708
  1313  1493  2675  2118   909  1569  7678   660  2835  2670  3447  3568
  3652  3660  1126   683  2251  4675  2353  3357   672   338  2697  2507
  1478  3565  2221  1898   960  8133  2301   983  2320   339  5152  3749
  3074   745  1469  1374   783  2606  9436   930  2751   250  1201   662
  1300  1559  3016  1360  1204  1597  2073  2142  2132  1546  1418  1343
  2662  6070  1927  2404  1554  1283   717  1747  1288  1038  2848  1413
  3632  3229  3577   682  1924   727   781  2121   701  2069  1525  7629
  3499  1346 10477  2924  1231  1961  5045  1255  1858  1221  1388  2279
  2759  1410  1403  3021  6568  2578  7758   343  1591  3416  1108  5965
  1514  6742  3650  3599 13756   276  4041   458   918  7393  1225  2812
  3029  1480  1047  1471  5511  1206  6403   707  1503  6078  2528  1037
  1352  3181  4594  5381  4657  1391  1913  7166  1409   976  2375   522
  2743  5804  1169   776  1322  1175  2133  1829 11760  1501  1200  3195
  4530  1555  2326  1887  1264   846  1532   935  2442  3590  2288  5117
 14179  1386   618  1574   700   886  4686   790   766  2212  7308  5743
  3973  7418  2629  1941  2445  6468  7374  3812  4006  7472  2028  5324
  2323  1984   999  7409  2186  4473   937  3422  3105  2748  3872  5190
  3001  3863  5801  1592  1185  3780  3612  1076  3527  2051  3331  3104
  2611  1311  2108  4042   926  1680  1249  2463  1595  2058  7814  1740
  1240  6842  5150  1203  2080  1538  3878  3186  2896  6967  1819  5943
  7127  3349 10974   518  1860  9566  2930  1505  2238  2197  1881  1880
  2389  1967  3380  1455   730  3244  1670  3979  1922  1295  1544   907
  1715  1347  1007  1402  2002  2096  1101   894  1577  2764  8358  5433
  3485  3850  7408  1377  4272  1553  9857   362  1935 10222  1330  9055
  7966  3496  6948 12204  3446   684  4281  7174  2359  3621   741  7865
  2910  5302  3620  3509  1657  1164  6229  1193  4583  5371   708   571
  2522  5179  8229  1289  2712   975  1050   609  4788  3069   836  2577
  1620  1845  6579  1893 10623  2249  3108   958  9277  6314  1526  6615
  1872  2859  1582  1238  1433  7882  4169  3249  3149  2246  1797  2957
  2348  6289  6419  6143 15857  2223  7238  2503  2622  4351   368   754
  2424  6681  2427   753  2576   590  1414  1103   585  1068   713  1092
  2329   882   866  2415  2101  1301  1113   760   625  1323  1138  1795
  2728   484  1048  1155  7057  1537  2214  1585  1521  3990  3049  1282
 10144  1168   454  3594  1768 15653  2247  4576  8335  5800  8471  3622
  2181  7685  6110  3757  3394  6304  1244  3518  2613  7476  4591  5595
  6224  1905  2993  8947  4020  2779  2782  1884 11054  9157  9283  6527
```

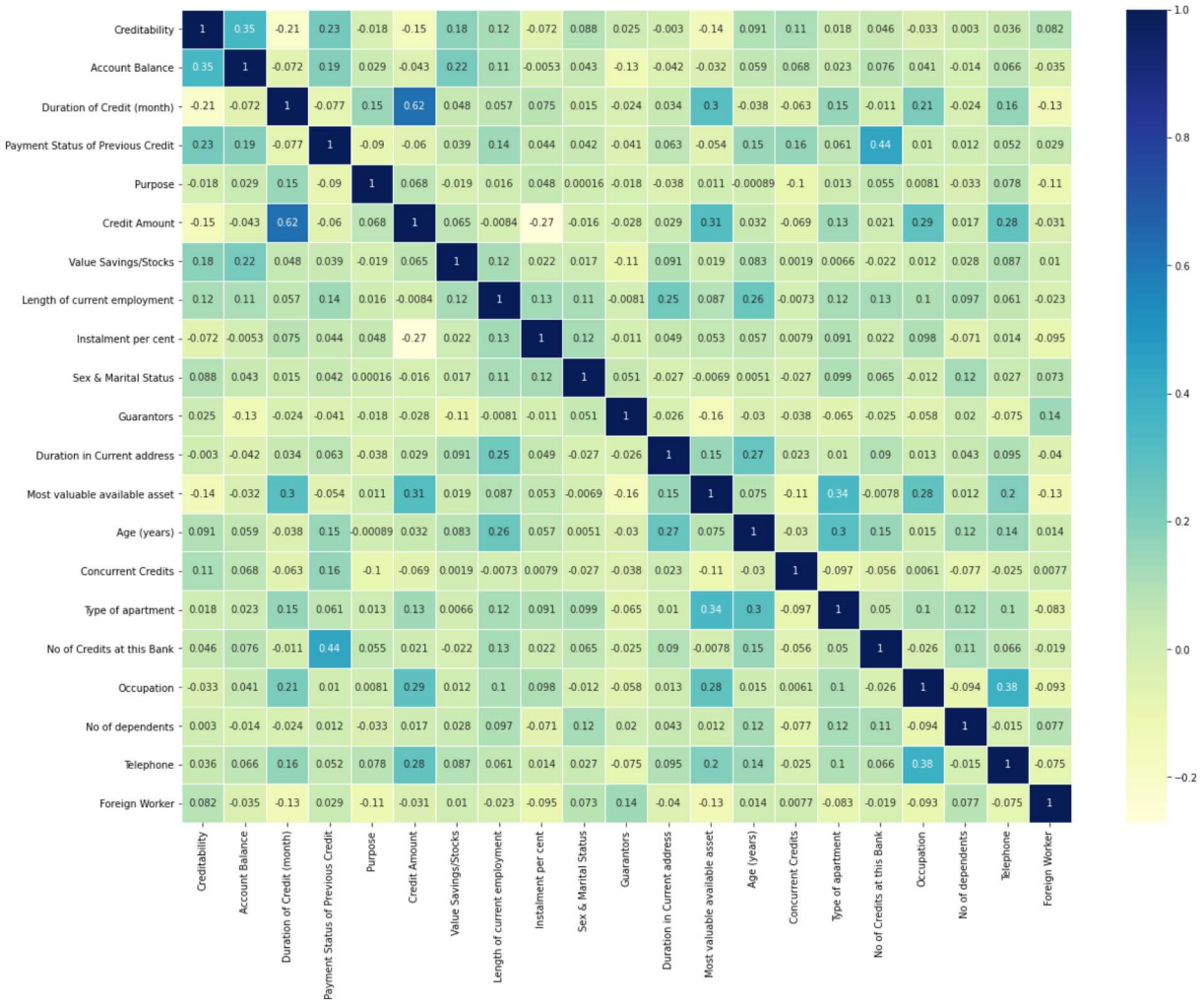
```
  3368  2511  5493  1338  1082  1149  1308  6148  1736  3059  2996  7596
  4811  1766  2760  5507  1199  2892  2862   654  1136  4113 14555   950
  2150  2820  3060  2600  5003  6288  2538  4933  1530  1437  1823  1422
  1217  9271  2145  1842  4297  3384  1245  4623  8386  1024 14318   433
  2149  2397   931  1512  4241  4736  1778  2327  6872   795  1908  1953
  2864  2319   915   947  1381  1285  1371  1042   900  1207  2278  6836
  3345  1198 15672  7297  1943  3190  5129  1808   759  1980 10961  6887
  1938  1835  1659  1209  3844  4843   639  5951  3804  4463  7980  4210
  4611 11560  4165  4057  6458  1977  1928  1123 11328 11938  2520 14782
  2671 12612  3031   626  3931  2302  3965  3914  4308  1534  2775  5998
  1271  9398   951  1355  3051  7855  9572  1837  4249  5234  6758  1366
  1358  2473  1337  7763  6560  3123  8065  2439  9034 14027  9629  1484
  1131  2064 12976  2580  2570  3915  1309  4817  2579  2225  4153  3114
  2124  1333  7119  4870   691  4370  2746  4110  2462  2969  4605  6331
  3552   697  1442  5293  3414  2039  3161   902 10297 14421  1056  1274
  1223  1372  2625  2235   959   884  1246  8086 10127   888   719 12389
  6850  2210  7485   797  4746   939  1188 11590  1190  2767  3441  4280
  3092  1331 15945  3234  9960  8648  1345  1647  4844  8318  2100 11816
   448 11998 18424 14896  2762  3386  2169  5096  1882  6999  2292  8978
   674  2718   750 12579  7511  3966  6199  1987  2303 12680  6350]
Value Savings/Stocks:
[1 2 3 5 4]
Length of current employment:
[2 3 4 1 5]
Instalment per cent:
[4 2 3 1]
Sex & Marital Status:
[2 3 4 1]
Guarantors:
[1 3 2]
Duration in Current address:
[4 2 3 1]
Most valuable available asset:
[2 1 3 4]
Age (years):
[21 36 23 39 38 48 40 65 24 31 44 25 37 49 33 26 51 29 56 47 34 28 41 58
 61 30 63 27 45 43 52 22 60 32 35 42 59 54 64 46 74 50 20 55 53 19 57 66
 68 70 67 75 62]
Concurrent Credits:
[3 1 2]
Type of apartment:
[1 2 3]
No of Credits at this Bank:
[1 2 3 4]
Occupation:
[3 2 1 4]
No of dependents:
[1 2]
Telephone:
[1 2]
Foreign Worker:
[1 2]
```

In [ ]:

```
corr = data.corr(method='pearson')
fig = plt.subplots(figsize = (20, 15))
sns.heatmap(corr,
            xticklabels=corr.columns,
            yticklabels=corr.columns,
            cmap='YlGnBu',
            annot=True,
            linewidth=0.5)
```

Out[ ]:

```
<AxesSubplot:>
```



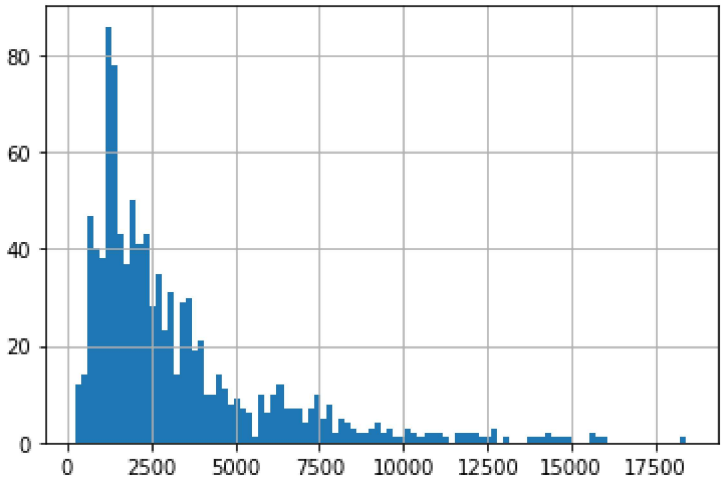
The only remarkable linear correlation is between Credit Amount and Duration of the Credit (month)

In [ ]:
```
print("min: " + str(data['Credit Amount'].min()) + " , max: " + str(data['Credit Amo
```

```
min: 250 , max: 18424
```

In [ ]:
```
data['Credit Amount'].hist(bins = 100)
```

Out[ ]:
```
<AxesSubplot:>
```



Since Credit Amount has a big number of unique value, it is best to discretize it in several bins.

In [ ]:

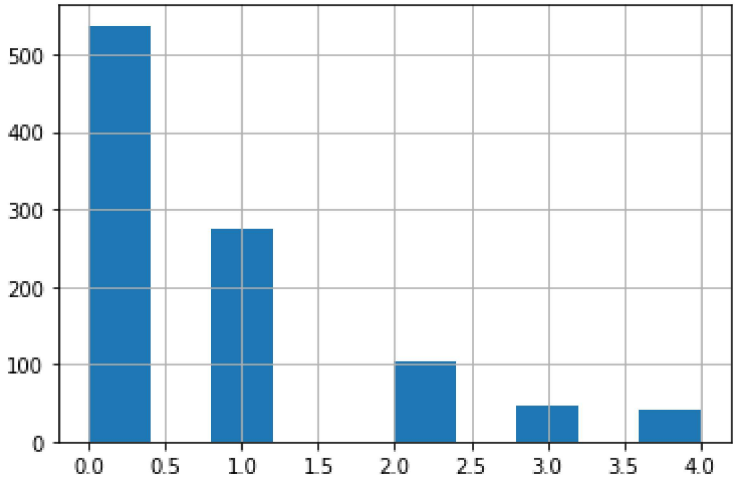
```
asd = pd.cut( x = data['Credit Amount'], bins=[0,2500,5000,7500,10000,100000], label
```

In [ ]:
```
asd.unique()
```

Out[ ]:
```
[0, 1, 2, 3, 4]
Categories (5, int64): [0 < 1 < 2 < 3 < 4]
```

In [ ]:
```
asd.hist(bins = 10)
```

Out[ ]:
```
<AxesSubplot:>
```



In [ ]:
```
data["Amount"] = asd
```

Train-Test Split

In this phase, the dataset is splitted into two parts: one for the training and one for the test.

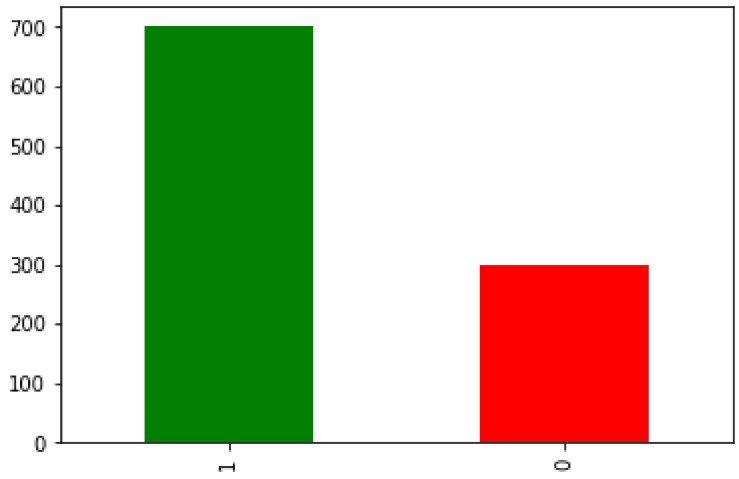
In [ ]:
```
data['Creditability'].value_counts()
```

Out[ ]:
```
1    700
0    300
Name: Creditability, dtype: int64
```

We have a case of unbalanced classes

In [ ]:
```
# Visualize the count for each class
from cProfile import label


data['Creditability'].value_counts().plot.bar(color=['green', 'red'])
```

Out[ ]:
```
<AxesSubplot:>
```



In [ ]:
```
x = data.drop(['Creditability'], axis = 1)
y = data.Creditability

# stratified on Creditability
X_train,X_test,y_train,y_test = train_test_split(x, y, test_size = 0.25,stratify=dat

# Show the results of the split
print("Training set has {} samples.".format(X_train.shape[0]))
print("Testing set has {} samples.".format(X_test.shape[0]))
```

```
Training set has 750 samples.
Testing set has 250 samples.
```

In [ ]:
```
y_train.value_counts()/750
```

Out[ ]:
```
1    0.7
0    0.3
Name: Creditability, dtype: float64
```

In [ ]:
```
y_test.value_counts()/250
```

Out[ ]:
```
1    0.7
0    0.3
Name: Creditability, dtype: float64
```

Train Models

In this paragraph we will train various Machine Learning models: LogisticRegression,K Nearest Neighbors, Decision Tree, Random Forest.
On each of these models will then be done the tuning of the hyperparameters, which will allow to find the best values that will be th input for the training of the best estimators. At the end, the results will be compared using several metrics, like Accuracy, Precision, Recall, FScore and AUC.

In [ ]:
```
list_metrics = []
```

Logistic Regression

In [ ]:
```
params = {"C": np.logspace(-4, 4, 20),
          "solver": ["newton-cg", "lbfgs", "liblinear", "sag", "saga"],
          "penalty": ['l2']}
```

```
lr_clf = LogisticRegression()

lr_cv = GridSearchCV(lr_clf, params, scoring="accuracy", n_jobs=-1, verbose=1, cv=5)
lr_cv.fit(X_train, y_train)
best_params = lr_cv.best_params_
print(f"Best parameters: {best_params}")
lr_clf = LogisticRegression(**best_params)
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
Best parameters: {'C': 1.623776739188721, 'penalty': 'l2', 'solver': 'newton-cg'}
```

In [ ]:

```
lr_clf.fit(X_train, y_train)

preds = lr_clf.predict(X_test)
```

In [ ]:

```
precision, recall, f_beta, support = precision_recall_fscore_support(y_test, preds,
auc = roc_auc_score(y_test, preds)
accuracy = accuracy_score(y_test, preds)

list_metrics.append(["Logistic Regression", accuracy, precision, recall, f_beta, auc

print(f"Accuracy is: {accuracy:.2f}")
print(f"Precision is: {precision:.2f}")
print(f"Recall is: {recall:.2f}")
print(f"Fscore is: {f_beta:.2f}")
print(f"AUC is: {auc:.2f}")
```

```
Accuracy is: 0.78
Precision is: 0.79
Recall is: 0.93
Fscore is: 0.92
AUC is: 0.67
```

Lets check the most important features

In [ ]:

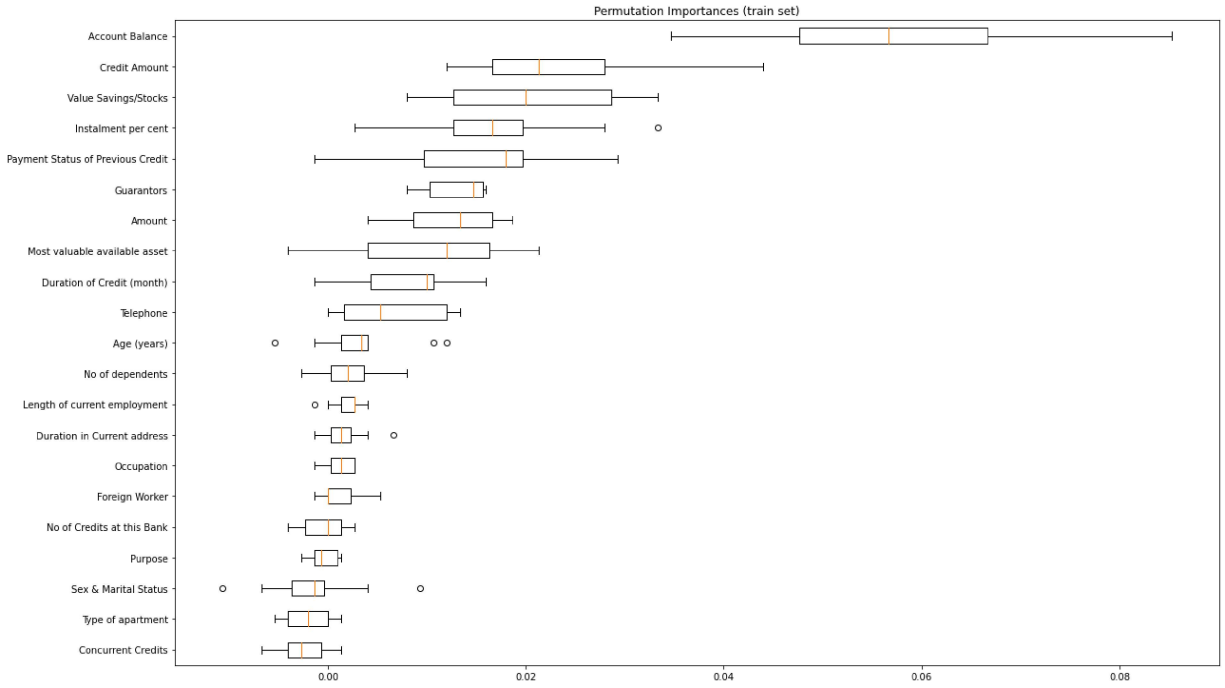
```
# Use permutation_importance to calculate permutation feature importances
feature_importances = permutation_importance(estimator=lr_clf, X = X_train, y = y_tr
                                  random_state=123, n_jobs=-1)
```

In [ ]:

```
def visualize_feature_importance(importance_array):
    # Sort the array based on mean value
    sorted_idx = importance_array.importances_mean.argsort()
    # Visualize the feature importances using boxplot
    fig, ax = plt.subplots()
    fig.set_figwidth(16)
    fig.set_figheight(10)
    fig.tight_layout()
    ax.boxplot(importance_array.importances[sorted_idx].T,
               vert=False, labels=X_train.columns[sorted_idx])
    ax.set_title("Permutation Importances (train set)")
    plt.show()
```

In [ ]:

```
visualize_feature_importance(feature_importances)
```



As we can see, we have several features that have a very little importance.
In this case, we can think to eliminate some of them and see if there is room for improvements in the results.

In [ ]:
```
sorted_idx = feature_importances.importances_mean
sorted_idx
```

Out[ ]:
```
array([ 0.05786667,  0.008     ,  0.0156    , -0.0004    ,  0.0232    ,
        0.02066667,  0.00186667,  0.01706667, -0.00146667,  0.01306667,
        0.0016    ,  0.0104    ,  0.00333333, -0.00266667, -0.002     ,
       -0.0004    ,  0.00133333,  0.00253333,  0.0064    ,  0.00106667,
        0.0124    ])
```

In [ ]:
```
soglia = feature_importances.importances_mean[18]
soglia
```

Out[ ]:
```
0.006400000000000028
```

In [ ]:
```
sorted_idx = feature_importances.importances_mean <= soglia
sorted_idx
```

Out[ ]:
```
array([False, False, False,  True, False, False,  True, False,  True,
       False,  True, False,  True,  True,  True,  True,  True,  True,
        True,  True, False])
```

In [ ]:
```
X2 = feature_importances.importances[sorted_idx]
X2
```

Out[ ]:
```
array([[ 0.00133333, -0.00133333,  0.00133333,  0.        , -0.00133333,
         -0.00266667,  0.00133333, -0.00133333,  0.        , -0.00133333],
       [ 0.00133333,  0.00266667,  0.00266667,  0.00266667,  0.00266667,
         0.004     ,  0.00266667,  0.        ,  0.00133333, -0.00133333],
       [-0.00133333, -0.00266667,  0.004     , -0.00133333, -0.00666667,
         -0.01066667,  0.00933333, -0.004     ,  0.        , -0.00133333],
       [-0.00133333,  0.00133333,  0.00266667,  0.00666667,  0.004     ,
         0.00133333,  0.        ,  0.00133333,  0.00133333, -0.00133333],
       [ 0.004     ,  0.00133333,  0.01066667, -0.00533333,  0.004     ,
         0.012     ,  0.00266667,  0.004     ,  0.00133333, -0.00133333],
```

```
       [-0.004     , -0.00533333, -0.004     ,  0.        , -0.00266667,
        -0.00266667, -0.00266667, -0.00666667,  0.00133333,  0.        ],
       [ 0.00133333, -0.00266667, -0.004     ,  0.        ,  0.        ,
        -0.00533333, -0.00133333, -0.00533333,  0.00133333, -0.004     ],
       [-0.00266667, -0.00133333,  0.00266667,  0.        ,  0.        ,
         0.00266667,  0.00133333,  0.00133333, -0.004     , -0.004     ],
       [ 0.00266667, -0.00133333,  0.00266667,  0.00133333,  0.00266667,
         0.00266667,  0.        ,  0.00133333,  0.        ,  0.00133333],
       [-0.00266667,  0.008     ,  0.00133333,  0.00266667,  0.00133333,
         0.        ,  0.00266667,  0.008     ,  0.        ,  0.004     ],
       [ 0.        ,  0.012     ,  0.008     ,  0.012     ,  0.00266667,
         0.        ,  0.01333333,  0.012     ,  0.00133333,  0.00266667],
       [ 0.        ,  0.        , -0.00133333, -0.00133333,  0.00533333,
         0.        ,  0.00133333,  0.        ,  0.00266667,  0.004     ]])
```

In [ ]:
```
X3 = X_train.columns[sorted_idx]
X3
```

Out[ ]:
```
Index(['Purpose', 'Length of current employment', 'Sex & Marital Status',
       'Duration in Current address', 'Age (years)', 'Concurrent Credits',
       'Type of apartment', 'No of Credits at this Bank', 'Occupation',
       'No of dependents', 'Telephone', 'Foreign Worker'],
      dtype='object')
```

In [ ]:
```
X_train_drop = X_train.drop(X3, axis = 1)
X_test_drop = X_test.drop(X3, axis = 1)
```

In [ ]:
```
X_train_drop.head()
```

Out[ ]:

	Account Balance	Duration of Credit (month)	Payment Status of Previous Credit	Credit Amount	Value Savings/Stocks	Instalment per cent	Guarantors	Most valuable available asset	Amc
397	4	12	2	707	1	4	1	1	
592	1	36	4	6229	1	4	2	4	
189	3	21	2	2923	2	1	1	3	
443	4	15	2	1386	5	4	1	1	
193	4	24	2	2255	5	4	1	2	

In [ ]:
```
params = {"C": np.logspace(-4, 4, 20),
          "solver": ["newton-cg", "lbfgs", "liblinear", "sag", "saga"],
          "penalty": ['l2']}

lr_clf = LogisticRegression()

lr_cv = GridSearchCV(lr_clf, params, scoring="accuracy", n_jobs=-1, verbose=1, cv=5)
lr_cv.fit(X_train_drop, y_train)
best_params = lr_cv.best_params_
print(f"Best parameters: {best_params}")
lr_clf = LogisticRegression(**best_params)
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
Best parameters: {'C': 0.012742749857031334, 'penalty': 'l2', 'solver': 'newton-cg'}
```

In [ ]:
```
lr_clf.fit(X_train_drop, y_train)
```

```
preds = lr_clf.predict(X_test_drop)
```

In [ ]:
```
precision, recall, f_beta, support = precision_recall_fscore_support(y_test, preds,
auc = roc_auc_score(y_test, preds)
accuracy = accuracy_score(y_test, preds)

list_metrics.append(["Logistic Regression drop", accuracy, precision, recall, f_beta

print(f"Accuracy is: {accuracy:.2f}")
print(f"Precision is: {precision:.2f}")
print(f"Recall is: {recall:.2f}")
print(f"Fscore is: {f_beta:.2f}")
print(f"AUC is: {auc:.2f}")
```

```
Accuracy is: 0.76
Precision is: 0.76
Recall is: 0.95
Fscore is: 0.94
AUC is: 0.62
```

In [ ]:
```
df1 = pd.DataFrame(list_metrics,columns=['Algorithm','accuracy', 'precision', 'recal
df1
```

Out[ ]:

	Algorithm	accuracy	precision	recall	f_beta	auc
0	Logistic Regression	0.776	0.787440	0.931429	0.924924	0.672381
1	Logistic Regression drop	0.756	0.759091	0.954286	0.944940	0.623810

As we can see, we have a little improvement on recall and f_beta, but some worsening on auc, accuracy and precision. So, I decided to keep all the feature for the further analysis.

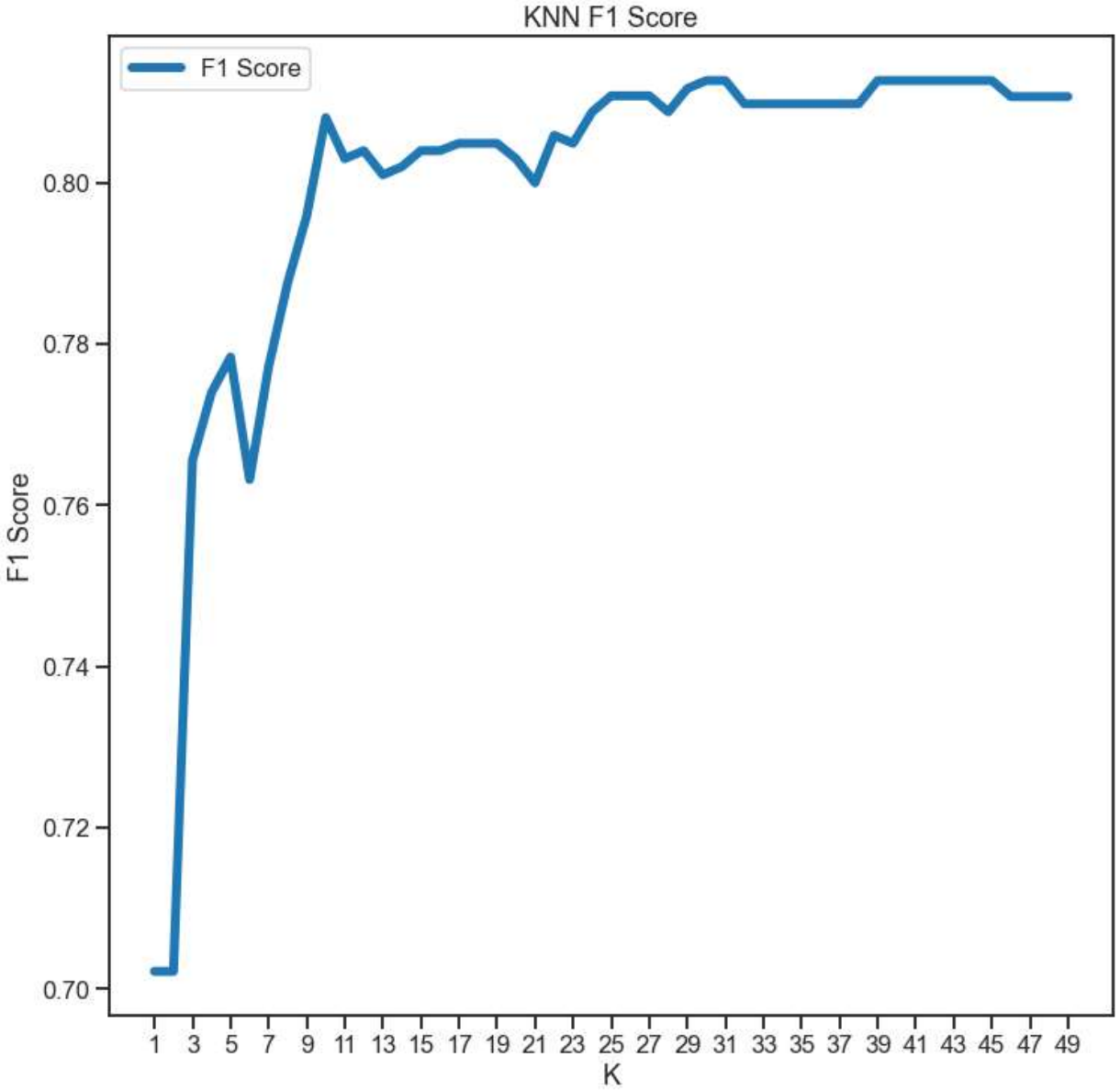
K-NEAREST NEIGHBORS

In [ ]:
```
### BEGIN SOLUTION
max_k = 50
f1_scores = list()
error_rates = list() # 1-accuracy

for k in range(1, max_k):

    knn = KNeighborsClassifier(n_neighbors=k, weights='distance')
    knn = knn.fit(X_train, y_train)

    y_pred = knn.predict(X_test)
    f1 = f1_score(y_pred, y_test)
    f1_scores.append((k, round(f1_score(y_test, y_pred), 4)))
    error = 1-round(accuracy_score(y_test, y_pred), 4)
    error_rates.append((k, error))

f1_results = pd.DataFrame(f1_scores, columns=['K', 'F1 Score'])
error_results = pd.DataFrame(error_rates, columns=['K', 'Error Rate'])
```

In [ ]:
```
# Plot F1 results
sns.set_context('talk')
sns.set_style('ticks')

plt.figure(dpi=300)
```

```
ax = f1_results.set_index('K').plot(figsize=(12, 12), linewidth=6)
ax.set(xlabel='K', ylabel='F1 Score')
ax.set_xticks(range(1, max_k, 2));
plt.title('KNN F1 Score')
plt.savefig('knn_f1.png')
```

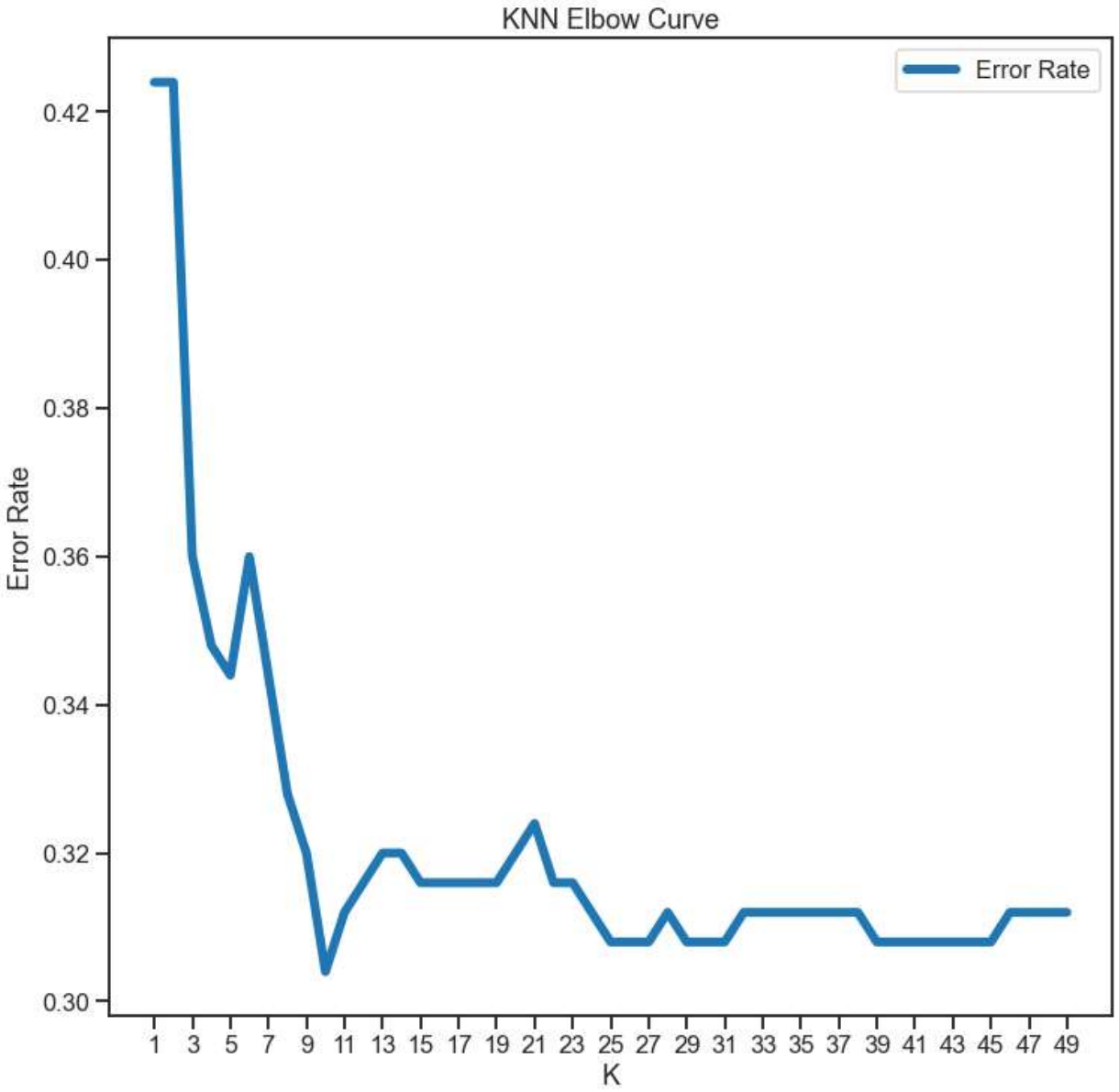
```
<Figure size 1800x1200 with 0 Axes>
```



In [ ]:

```
# Plot Accuracy (Error Rate) results
sns.set_context('talk')
sns.set_style('ticks')

plt.figure(dpi=300)
ax = error_results.set_index('K').plot(figsize=(12, 12), linewidth=6)
ax.set(xlabel='K', ylabel='Error Rate')
ax.set_xticks(range(1, max_k, 2))
plt.title('KNN Elbow Curve')
plt.savefig('knn_elbow.png')
```

```
<Figure size 1800x1200 with 0 Axes>
```



K = 10 is the best parameter

In [ ]:
```
knn = KNeighborsClassifier(n_neighbors=10, weights='distance')
knn = knn.fit(X_train, y_train)

y_pred = knn.predict(X_test)
```

In [ ]:
```
precision, recall, f_beta, support = precision_recall_fscore_support(y_test, y_pred,
auc = roc_auc_score(y_test, y_pred)
accuracy = accuracy_score(y_test, y_pred)

list_metrics.append(["KNN", accuracy, precision, recall, f_beta, auc])

print(f"Accuracy is: {accuracy:.2f}")
print(f"Precision is: {precision:.2f}")
print(f"Recall is: {recall:.2f}")
print(f"Fscore is: {f_beta:.2f}")
print(f"AUC is: {auc:.2f}")
```

```
Accuracy is: 0.70
Precision is: 0.72
Recall is: 0.91
Fscore is: 0.91
AUC is: 0.55
```

DECISION TREE

In [ ]:
```
params = {"criterion":("gini", "entropy"),
          "splitter":("best", "random"),
          "max_depth":(list(range(1, 20))),
          "min_samples_split":[2, 3, 4],
          "min_samples_leaf":list(range(1, 20))
          }

tree_clf = DecisionTreeClassifier(random_state=42)
tree_cv = GridSearchCV(tree_clf, params, scoring="accuracy", n_jobs=-1, verbose=1, c
tree_cv.fit(X_train, y_train)
best_params = tree_cv.best_params_
print(f'Best_params: {best_params}')
```

```
Fitting 5 folds for each of 4332 candidates, totalling 21660 fits
Best_params: {'criterion': 'gini', 'max_depth': 8, 'min_samples_leaf': 14, 'min_samp
les_split': 2, 'splitter': 'best'}
```

In [ ]:
```
tree_clf = DecisionTreeClassifier(**best_params)
tree_clf.fit(X_train, y_train)

y_pred = tree_clf.predict(X_test)
```

In [ ]:
```
precision, recall, f_beta, support = precision_recall_fscore_support(y_test, y_pred,
auc = roc_auc_score(y_test, y_pred)
accuracy = accuracy_score(y_test, y_pred)

list_metrics.append(["Decision Tree", accuracy, precision, recall, f_beta, auc])

print(f"Accuracy is: {accuracy:.2f}")
print(f"Precision is: {precision:.2f}")
print(f"Recall is: {recall:.2f}")
print(f"Fscore is: {f_beta:.2f}")
print(f"AUC is: {auc:.2f}")
```

```
Accuracy is: 0.75
Precision is: 0.79
Recall is: 0.87
Fscore is: 0.87
AUC is: 0.67
```

Printing the tree

In [ ]:
```
from sklearn.tree import export_text
tree_exp = export_text(tree_clf, feature_names=list(X_train.columns))
```

As we can see the first 3 most important choises are about the account balance, the Value savings and the Duration of the Credit

In [ ]:
```
print(tree_exp)
```

```
|--- Account Balance <= 2.50
|   |--- Value Savings/Stocks <= 2.50
|   |   |--- Duration of Credit (month) <= 22.50
|   |   |   |--- Payment Status of Previous Credit <= 1.50
|   |   |   |   |--- class: 0
|   |   |   |--- Payment Status of Previous Credit >  1.50
|   |   |   |   |--- Duration of Credit (month) <= 11.50
|   |   |   |   |   |--- Most valuable available asset <= 1.50
```

```
|   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |--- Most valuable available asset >  1.50
|   |   |   |   |   |   |--- class: 1
|   |   |   |   |--- Duration of Credit (month) >  11.50
|   |   |   |   |   |--- Credit Amount <= 1387.50
|   |   |   |   |   |   |--- Most valuable available asset <= 1.50
|   |   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |   |--- Most valuable available asset >  1.50
|   |   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |--- Credit Amount >  1387.50
|   |   |   |   |   |   |--- Age (years) <= 30.50
|   |   |   |   |   |   |   |--- Age (years) <= 26.50
|   |   |   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |   |   |--- Age (years) >  26.50
|   |   |   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |   |--- Age (years) >  30.50
|   |   |   |   |   |   |   |--- Length of current employment <= 4.50
|   |   |   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |   |   |--- Length of current employment >  4.50
|   |   |   |   |   |   |   |   |--- class: 0
|   |   |   |--- Duration of Credit (month) >  22.50
|   |   |   |   |--- Age (years) <= 29.50
|   |   |   |   |   |--- Duration of Credit (month) <= 27.00
|   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |--- Duration of Credit (month) >  27.00
|   |   |   |   |   |   |--- Credit Amount <= 5483.00
|   |   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |   |--- Credit Amount >  5483.00
|   |   |   |   |   |   |   |--- class: 0
|   |   |   |   |--- Age (years) >  29.50
|   |   |   |   |   |--- Duration of Credit (month) <= 47.50
|   |   |   |   |   |   |--- Credit Amount <= 2314.00
|   |   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |   |--- Credit Amount >  2314.00
|   |   |   |   |   |   |   |--- Purpose <= 1.50
|   |   |   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |   |   |--- Purpose >  1.50
|   |   |   |   |   |   |   |   |--- Telephone <= 1.50
|   |   |   |   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |   |   |   |--- Telephone >  1.50
|   |   |   |   |   |   |   |   |   |--- class: 0
|   |   |   |   |--- Duration of Credit (month) >  47.50
|   |   |   |   |   |--- class: 0
|   |   |--- Value Savings/Stocks >  2.50
|   |   |   |--- Account Balance <= 1.50
|   |   |   |   |--- Duration of Credit (month) <= 20.50
|   |   |   |   |   |--- class: 1
|   |   |   |   |--- Duration of Credit (month) >  20.50
|   |   |   |   |   |--- class: 0
|   |   |   |--- Account Balance >  1.50
|   |   |   |   |--- Duration in Current address <= 3.50
|   |   |   |   |   |--- Duration of Credit (month) <= 13.50
|   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |--- Duration of Credit (month) >  13.50
|   |   |   |   |   |   |--- class: 1
|   |   |   |   |--- Duration in Current address >  3.50
|   |   |   |   |   |--- class: 1
|--- Account Balance >  2.50
|   |--- Credit Amount <= 7839.50
|   |   |--- Account Balance <= 3.50
|   |   |   |--- Credit Amount <= 2343.50
|   |   |   |   |--- Age (years) <= 39.50
|   |   |   |   |   |--- class: 1
|   |   |   |   |--- Age (years) >  39.50
|   |   |   |   |   |--- class: 1
|   |   |   |--- Credit Amount >  2343.50
|   |   |   |   |--- class: 1
|   |   |--- Account Balance >  3.50
|   |   |   |--- Purpose <= 4.50
```

```
|   |   |   |   |--- Age (years) <= 23.50
|   |   |   |   |   |--- class: 1
|   |   |   |   |--- Age (years) >  23.50
|   |   |   |   |   |--- Most valuable available asset <= 1.50
|   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |--- Most valuable available asset >  1.50
|   |   |   |   |   |   |--- Occupation <= 2.50
|   |   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |   |--- Occupation >  2.50
|   |   |   |   |   |   |   |--- Credit Amount <= 3594.50
|   |   |   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |   |   |--- Credit Amount >  3594.50
|   |   |   |   |   |   |   |   |--- class: 1
|   |   |   |--- Purpose >  4.50
|   |   |   |   |--- Duration of Credit (month) <= 19.50
|   |   |   |   |   |--- Credit Amount <= 1548.50
|   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |--- Credit Amount >  1548.50
|   |   |   |   |   |   |--- class: 1
|   |   |   |   |--- Duration of Credit (month) >  19.50
|   |   |   |   |   |--- class: 1
|   |--- Credit Amount >  7839.50
|   |   |--- class: 1
```

RANDOM FOREST

In [ ]:

```
params_grid = {
    'bootstrap': [True],
    'max_depth': [80, 90, 100, 110],
    'max_features': [2, 3],
    'min_samples_leaf': [3, 4, 5],
    'min_samples_split': [8, 10, 12],
    'n_estimators': [100, 200, 300, 1000]
}

rf_clf = RandomForestClassifier(random_state=42)

rf_cv = GridSearchCV(rf_clf, params_grid, scoring="accuracy", cv=5, verbose=2, n_job


rf_cv.fit(X_train, y_train)
best_params = rf_cv.best_params_
print(f"Best parameters: {best_params}")
```

```
Fitting 5 folds for each of 288 candidates, totalling 1440 fits
Best parameters: {'bootstrap': True, 'max_depth': 80, 'max_features': 3, 'min_sample
s_leaf': 4, 'min_samples_split': 10, 'n_estimators': 200}
```

In [ ]:

```
rf_cv.best_params_
```

Out[ ]:

```
{'bootstrap': True,
 'max_depth': 80,
 'max_features': 3,
 'min_samples_leaf': 4,
 'min_samples_split': 10,
 'n_estimators': 200}
```

In [ ]:

```
rf_clf = RandomForestClassifier(**best_params)
rf_clf.fit(X_train, y_train)

y_pred = rf_clf.predict(X_test)
```

In [ ]:

```
precision, recall, f_beta, support = precision_recall_fscore_support(y_test, y_pred,
auc = roc_auc_score(y_test, y_pred)
accuracy = accuracy_score(y_test, y_pred)

list_metrics.append(["Random Forest", accuracy, precision, recall, f_beta, auc])

print(f"Accuracy is: {accuracy:.2f}")
print(f"Precision is: {precision:.2f}")
print(f"Recall is: {recall:.2f}")
print(f"Fscore is: {f_beta:.2f}")
print(f"AUC is: {auc:.2f}")
```

```
Accuracy is: 0.77
Precision is: 0.76
Recall is: 0.97
Fscore is: 0.96
AUC is: 0.64
```

Oversampling

One last test I would like to do is to see if there will be an improvement in performance with oversampling.

In [ ]:
```
# Resample training data using SMOTE
smote_sampler = SMOTE(random_state = 32)

X_smo, y_smo = smote_sampler.fit_resample(X_train, y_train)
```

In [ ]:
```
y_smo.value_counts()
```

Out[ ]:
```
0    525
1    525
Name: Creditability, dtype: int64
```

In [ ]:
```
rf_clf.fit(X_smo, y_smo)

y_pred = rf_clf.predict(X_test)
```

In [ ]:
```
precision, recall, f_beta, support = precision_recall_fscore_support(y_test, y_pred,
auc = roc_auc_score(y_test, y_pred)
accuracy = accuracy_score(y_test, y_pred)

list_metrics.append(["Decision Tree with Oversampling", accuracy, precision, recall,

print(f"Accuracy is: {accuracy:.2f}")
print(f"Precision is: {precision:.2f}")
print(f"Recall is: {recall:.2f}")
print(f"Fscore is: {f_beta:.2f}")
print(f"AUC is: {auc:.2f}")
```

```
Accuracy is: 0.73
Precision is: 0.80
Recall is: 0.82
Fscore is: 0.82
AUC is: 0.67
```

Conclusions

In [ ]:
```
df2 = pd.DataFrame(list_metrics,columns=['Algorithm','Accuracy', 'Precision', 'Recal
df2
```

Out[ ]:

	Algorithm	Accuracy	Precision	Recall	FScore	AUC
0	Logistic Regression	0.776	0.787440	0.931429	0.924924	0.672381
1	Logistic Regression drop	0.756	0.759091	0.954286	0.944940	0.623810
2	KNN	0.696	0.723982	0.914286	0.905135	0.550476
3	Decision Tree	0.752	0.792746	0.874286	0.870841	0.670476
4	Random Forest	0.768	0.764706	0.965714	0.956049	0.636190
5	Decision Tree with Oversampling	0.732	0.800000	0.822857	0.821954	0.671429

As can be seen, we have very close results for each of the algorithms tested.
The one that performs best, however, is the Random Forest, with good values for any of the metrics under consideration.

Looking further, surely it would be interesting to conduct further tests with the techniques for handling unbalanced classes, tested here only summarily.

Raffaele Pane