Import Libraries

**Import the usual libraries for pandas and plotting.

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

Get the Data

In [2]:

```
bank=pd.read_csv('Retail data.csv', delimiter=';', skiprows=0, low_memory=False)
```

Analyzing and preparing dataset for prediction

Check out the info(), head()

In [3]:

```
bank.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 23983 entries, 0 to 23982
Data columns (total 14 columns):
Cocunut                    23983 non-null int64
Mortgage_YN                23983 non-null object
AGE_AT_ORIGINATION         306 non-null float64
AGE                        23983 non-null int64
YEARS_WITH_BANK            23983 non-null int64
MARTIAL_STATUS             23983 non-null object
EDUCATION                  23983 non-null object
EMPLOYMENT                 23983 non-null object
GENDER                     23983 non-null object
CUST_INCOME                23983 non-null object
CURRENT_ADDRESS_DATE       23983 non-null object
CURRENT_JOB_DATE           23983 non-null object
CURRENT_WITH_BANK_DATE     23983 non-null object
CURRENT_BALANCE_EUR        23983 non-null object
dtypes: float64(1), int64(3), object(10)
memory usage: 2.6+ MB
```

In [4]:

```
bank.head()
```

Out[4]:

	Cocunut	Mortgage_YN	AGE_AT_ORIGINATION	AGE	YEARS_WITH_BANK	MARTIAL_STATUS	EDUCATION	EMPLOYMENT
0	1	Y	50.0	52	13	M	HGH	PVE
1	9	Y	48.0	49	11	M	HGH	SFE
2	11	Y	53.0	55	14	M	BCR	STE
3	12	Y	64.0	66	10	M	BCR	OTH
4	18	Y	46.0	47	9	S	MAS	PVE

Check is there NA values on data.

In [5]:

```
bank.isna().any()
```

Out[5]:

```
Cocunut                   False
Mortgage_YN               False
AGE_AT_ORIGINATION         True
AGE                       False
YEARS_WITH_BANK           False
MARTIAL_STATUS            False
EDUCATION                 False
EMPLOYMENT                False
GENDER                    False
CUST_INCOME               False
CURRENT_ADDRESS_DATE      False
CURRENT_JOB_DATE          False
CURRENT_WITH_BANK_DATE    False
CURRENT_BALANCE_EUR       False
dtype: bool
```

Counting NA values on dataset

In [6]:

```
bank.isna().sum()
```

Out[6]:

```
Cocunut                       0
Mortgage_YN                   0
AGE_AT_ORIGINATION        23677
AGE                           0
YEARS_WITH_BANK               0
MARTIAL_STATUS                0
EDUCATION                     0
EMPLOYMENT                    0
GENDER                        0
CUST_INCOME                   0
CURRENT_ADDRESS_DATE          0
CURRENT_JOB_DATE              0
CURRENT_WITH_BANK_DATE        0
CURRENT_BALANCE_EUR           0
dtype: int64
```

Removing column "AGE_AT_ORIGINATION", because there is a lot of NA values and it is bad for model

Defining new dataset "loans" without that column

In [7]:

```
loans=bank.drop(['AGE_AT_ORIGINATION'], axis=1)
```

In [8]:

```
loans.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 23983 entries, 0 to 23982
Data columns (total 13 columns):
Cocunut                   23983 non-null int64
Mortgage_YN               23983 non-null object
AGE                       23983 non-null int64
YEARS_WITH_BANK           23983 non-null int64
MARTIAL_STATUS            23983 non-null object
EDUCATION                 23983 non-null object
```

```
EMPLOYMENT                  23983 non-null object
GENDER                      23983 non-null object
CUST_INCOME                 23983 non-null object
CURRENT_ADDRESS_DATE        23983 non-null object
CURRENT_JOB_DATE            23983 non-null object
CURRENT_WITH_BANK_DATE      23983 non-null object
CURRENT_BALANCE_EUR         23983 non-null object
dtypes: int64(3), object(10)
memory usage: 2.4+ MB
```

Transforming values of variable 'Mortgage_YN', so we could use it, to make a model on this variable

In [9]:

```
loans['Mortgage_YN'].value_counts()
```

Out[9]:

```
N    23677
Y      306
Name: Mortgage_YN, dtype: int64
```

In [10]:

```
loans['Mortgage_YN']=loans['Mortgage_YN'].replace({'N': 0, 'Y': 1}).astype(int)
```

In [11]:

```
loans.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 23983 entries, 0 to 23982
Data columns (total 13 columns):
Cocunut                     23983 non-null int64
Mortgage_YN                 23983 non-null int32
AGE                         23983 non-null int64
YEARS_WITH_BANK             23983 non-null int64
MARTIAL_STATUS              23983 non-null object
EDUCATION                   23983 non-null object
EMPLOYMENT                  23983 non-null object
GENDER                      23983 non-null object
CUST_INCOME                 23983 non-null object
CURRENT_ADDRESS_DATE        23983 non-null object
CURRENT_JOB_DATE            23983 non-null object
CURRENT_WITH_BANK_DATE      23983 non-null object
CURRENT_BALANCE_EUR         23983 non-null object
dtypes: int32(1), int64(3), object(9)
memory usage: 2.3+ MB
```

In [12]:

```
loans['Mortgage_YN'].value_counts()
```

Out[12]:

```
0    23677
1      306
Name: Mortgage_YN, dtype: int64
```

Recoding other variables, so we could use it, to make a model on this variable

In [13]:

```
loans['MARTIAL_STATUS'].value_counts()
```

Out[13]:

```
M          17024
S           4223
D           1364
W           1329
*noval*       43
```

```
"noval"        43
Name: MARTIAL_STATUS, dtype: int64
```

Removing 'noval' from 'MARTIAL STATUS'

In [14]:

```
loans=loans[(loans['MARTIAL_STATUS'] == 'M') | (loans['MARTIAL_STATUS'] == 'S') | (loans
['MARTIAL_STATUS'] == 'D') | (loans['MARTIAL_STATUS'] == 'W')]
```

In [15]:

```
loans.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 23940 entries, 0 to 23982
Data columns (total 13 columns):
Cocunut                    23940 non-null int64
Mortgage_YN                23940 non-null int32
AGE                        23940 non-null int64
YEARS_WITH_BANK            23940 non-null int64
MARTIAL_STATUS             23940 non-null object
EDUCATION                  23940 non-null object
EMPLOYMENT                 23940 non-null object
GENDER                     23940 non-null object
CUST_INCOME                23940 non-null object
CURRENT_ADDRESS_DATE       23940 non-null object
CURRENT_JOB_DATE           23940 non-null object
CURRENT_WITH_BANK_DATE     23940 non-null object
CURRENT_BALANCE_EUR        23940 non-null object
dtypes: int32(1), int64(3), object(9)
memory usage: 2.5+ MB
```

In [16]:

```
loans['MARTIAL_STATUS']=loans['MARTIAL_STATUS'].replace({'M': 0, 'S': 1, 'D': 2, 'W': 3}
).astype(int)
```

In [17]:

```
loans.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 23940 entries, 0 to 23982
Data columns (total 13 columns):
Cocunut                    23940 non-null int64
Mortgage_YN                23940 non-null int32
AGE                        23940 non-null int64
YEARS_WITH_BANK            23940 non-null int64
MARTIAL_STATUS             23940 non-null int32
EDUCATION                  23940 non-null object
EMPLOYMENT                 23940 non-null object
GENDER                     23940 non-null object
CUST_INCOME                23940 non-null object
CURRENT_ADDRESS_DATE       23940 non-null object
CURRENT_JOB_DATE           23940 non-null object
CURRENT_WITH_BANK_DATE     23940 non-null object
CURRENT_BALANCE_EUR        23940 non-null object
dtypes: int32(2), int64(3), object(8)
memory usage: 2.4+ MB
```

In [18]:

```
loans['EDUCATION'].value_counts()
```

Out[18]:

```
HGH    15957
BCR     6619
PRS      633
SEC      414
MAS      193
```

```
PHD       117
PRI         6
OTH         1
Name: EDUCATION, dtype: int64
```

In [19]:

```
loans['EDUCATION']=loans['EDUCATION'].replace({'HGH': 0, 'BCR': 1, 'PRS': 2, 'SEC': 3, 'MAS': 4, 'PHD':5, 'PRI':6, 'OTH':7}).astype(int)
```

In [20]:

```
loans['EMPLOYMENT'].value_counts()
```

Out[20]:

```
PVE    10725
STE     6699
RET     6096
SFE      264
OTH      156
Name: EMPLOYMENT, dtype: int64
```

In [21]:

```
loans['EMPLOYMENT']=loans['EMPLOYMENT'].replace({'PVE': 0, 'STE': 1, 'RET': 2, 'SFE': 3, 'OTH':4}).astype(int)
```

In [22]:

```
loans['GENDER'].value_counts()
```

Out[22]:

```
F    12073
M    11867
Name: GENDER, dtype: int64
```

In [23]:

```
loans['GENDER']=loans['GENDER'].replace({'F': 0, 'M': 1}).astype(int)
```

In [24]:

```
loans['CUST_INCOME']=loans['CUST_INCOME'].apply(lambda x: x.replace(',', '.')).astype('float')
```

In [25]:

```
loans['CURRENT_BALANCE_EUR']=loans['CURRENT_BALANCE_EUR'].apply(lambda x: x.replace(',', '.')).astype('float')
```

In [26]:

```
loans.tail()
```

Out[26]:

	Cocunut	Mortgage_YN	AGE	YEARS_WITH_BANK	MARTIAL_STATUS	EDUCATION	EMPLOYMENT	GENDER	CUST_I
23978	79979	0	67	3	0	1	2	1	179
23979	79982	0	59	13	0	0	0	0	690
23980	79983	0	68	7	3	2	2	1	255
23981	79985	0	59	2	0	1	1	1	459
23982	79998	0	47	1	0	0	1	0	528

In [27]:

```
loans.isna().sum()
```

Out[27]:

```
Cocunut                 0
Mortgage_YN             0
AGE                     0
YEARS_WITH_BANK         0
MARTIAL_STATUS          0
EDUCATION               0
EMPLOYMENT              0
GENDER                  0
CUST_INCOME             0
CURRENT_ADDRESS_DATE    0
CURRENT_JOB_DATE        0
CURRENT_WITH_BANK_DATE  0
CURRENT_BALANCE_EUR     0
dtype: int64
```

Categorical Features

In [28]:

```
loans.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 23940 entries, 0 to 23982
Data columns (total 13 columns):
Cocunut                   23940 non-null int64
Mortgage_YN               23940 non-null int32
AGE                       23940 non-null int64
YEARS_WITH_BANK           23940 non-null int64
MARTIAL_STATUS            23940 non-null int32
EDUCATION                 23940 non-null int32
EMPLOYMENT                23940 non-null int32
GENDER                    23940 non-null int32
CUST_INCOME               23940 non-null float64
CURRENT_ADDRESS_DATE      23940 non-null object
CURRENT_JOB_DATE          23940 non-null object
CURRENT_WITH_BANK_DATE    23940 non-null object
CURRENT_BALANCE_EUR       23940 non-null float64
dtypes: float64(2), int32(5), int64(3), object(3)
memory usage: 2.1+ MB
```

Delete date variables, so we can use sklearn

In [29]:

```
loans2=loans.drop(['CURRENT_WITH_BANK_DATE','CURRENT_JOB_DATE','CURRENT_ADDRESS_DATE'], a
xis=1)
loans2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 23940 entries, 0 to 23982
Data columns (total 10 columns):
Cocunut                23940 non-null int64
Mortgage_YN            23940 non-null int32
AGE                    23940 non-null int64
YEARS_WITH_BANK        23940 non-null int64
MARTIAL_STATUS         23940 non-null int32
EDUCATION              23940 non-null int32
EMPLOYMENT             23940 non-null int32
GENDER                 23940 non-null int32
CUST_INCOME            23940 non-null float64
CURRENT_BALANCE_EUR    23940 non-null float64
dtypes: float64(2), int32(5), int64(3)
memory usage: 1.6 MB
```

Delete date variables, so we can use sklearn

In [30]:

```
loans2.head()
```

Out[30]:

	Cocunut	Mortgage_YN	AGE	YEARS_WITH_BANK	MARTIAL_STATUS	EDUCATION	EMPLOYMENT	GENDER	CUST_INCO
0	1	1	52	13	0	0	0	1	909.501
1	9	1	49	11	0	0	3	1	288.461
2	11	1	55	14	0	1	1	1	1280.528
3	12	1	66	10	0	1	4	0	620.959
4	18	1	47	9	1	4	0	0	2239.853

In [31]:

```
loans2['Mortgage_YN'].value_counts()
```

Out[31]:

```
0    23636
1      304
Name: Mortgage_YN, dtype: int64
```

Train Test Split

Now its time to split our data into a training set and a testing set!

In [32]:

```
from sklearn.model_selection import train_test_split
```

In [33]:

```
X = loans2.drop('Mortgage_YN',axis=1)
y = loans2['Mortgage_YN']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.05, random_state=4
2)
```

Training a Decision Tree Model

Let's start by training a single decision tree first!

Import DecisionTreeClassifier

In [34]:

```
from sklearn.tree import DecisionTreeClassifier
```

Create an instance of DecisionTreeClassifier() called dtree and fit it to the training data.

In [35]:

```
dtree = DecisionTreeClassifier()
```

In [36]:

```
dtree.fit(X_train,y_train)
```

Out[36]:

```
DecisionTreeClassifier()
```

Predictions and Evaluation of Decision Tree

Create predictions from the test set and create a classification report and a confusion matrix.

In [37]:

```
predictions = dtree.predict(X_test)
```

In [38]:

```
from sklearn.metrics import classification_report,confusion_matrix
```

In [39]:

```
print(classification_report(y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      1182
           1       1.00      1.00      1.00        15

    accuracy                           1.00      1197
   macro avg       1.00      1.00      1.00      1197
weighted avg       1.00      1.00      1.00      1197
```

In [40]:

```
print(confusion_matrix(y_test,predictions))
```

```
[[1182    0]
 [   0   15]]
```

Training the Random Forest model

Now its time to train our model!

Create an instance of the RandomForestClassifier class and fit it to our training data from the previous step.

In [41]:

```
from sklearn.ensemble import RandomForestClassifier
```

In [42]:

```
rfc = RandomForestClassifier(n_estimators=600)
```

In [43]:

```
rfc.fit(X_train,y_train)
```

Out[43]:

```
RandomForestClassifier(n_estimators=600)
```

Predictions and Evaluation

Let's predict off the y_test values and evaluate our model.

Predict the class of not.fully.paid for the X_test data.

In [44]:

```
predictions = rfc.predict(X_test)
```

Now create a classification report from the results

In [45]:

```
from sklearn.metrics import classification_report,confusion_matrix
```

In [46]:

```
print(classification_report(y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      1182
           1       1.00      1.00      1.00        15

    accuracy                           1.00      1197
   macro avg       1.00      1.00      1.00      1197
weighted avg       1.00      1.00      1.00      1197
```

In [47]:

```
print(confusion_matrix(y_test,predictions))
```

```
[[1182    0]
 [   0   15]]
```

LogMODEL

Now it's time to do a train test split, and train for logmodel!

In [48]:

```
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
```

In [49]:

```
LR=loans.drop(['CURRENT_WITH_BANK_DATE','CURRENT_JOB_DATE','CURRENT_ADDRESS_DATE'], axis=
1)
LR.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 23940 entries, 0 to 23982
Data columns (total 10 columns):
Cocunut                 23940 non-null int64
Mortgage_YN             23940 non-null int32
AGE                     23940 non-null int64
YEARS_WITH_BANK         23940 non-null int64
MARTIAL_STATUS          23940 non-null int32
EDUCATION               23940 non-null int32
EMPLOYMENT              23940 non-null int32
GENDER                  23940 non-null int32
CUST_INCOME             23940 non-null float64
CURRENT_BALANCE_EUR     23940 non-null float64
dtypes: float64(2), int32(5), int64(3)
memory usage: 1.6 MB
```

In [50]:

```
LR.head()
```

Out[50]:

	Cocunut	Mortgage_YN	AGE	YEARS_WITH_BANK	MARTIAL_STATUS	EDUCATION	EMPLOYMENT	GENDER	CUST_INCO
0	1	1	52	13	0	0	0	1	909.501

	Cocunut	Mortgage_YN	AGE	YEARS_WITH_BANK	MARTIAL_STATUS	EDUCATION	EMPLOYMENT	GENDER	CUST_INCO
1	9	1	49	11	0	0	3	1	288.461
2	11	1	55	14	0	1	1	1	1280.528
3	12	1	66	10	0	1	4	0	620.959
4	18	1	47	9	1	4	0	0	2239.853

In [51]:

```
X = LR[['AGE', 'YEARS_WITH_BANK', 'MARTIAL_STATUS','EDUCATION', 'EMPLOYMENT','GENDER','C
UST_INCOME','CURRENT_BALANCE_EUR','Cocunut']]
y = LR['Mortgage_YN']
```

In [52]:

```
X.head()
```

Out[52]:

	AGE	YEARS_WITH_BANK	MARTIAL_STATUS	EDUCATION	EMPLOYMENT	GENDER	CUST_INCOME	CURRENT_BALANCE
0	52	13	0	0	0	1	909.501308	7648.3
1	49	11	0	0	3	1	288.461539	30189.9
2	55	14	0	1	1	1	1280.528692	50553.1
3	66	10	0	1	4	0	620.959769	15907.2
4	47	9	1	4	0	0	2239.853846	27916.1

Train and fit a logistic regression model on the training set.

In [53]:

```
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
```

In [54]:

```
X, y = make_classification(random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)
logmodel = make_pipeline(StandardScaler(), LogisticRegression())
logmodel.fit(X_train, y_train)
```

Out[54]:

```
Pipeline(steps=[('standardscaler', StandardScaler()),
                ('logisticregression', LogisticRegression())])
```

Predictions and Evaluations

Now predict values for the testing data.

In [55]:

```
predictions = logmodel.predict(X_test)
```

In [56]:

```
from sklearn.metrics import classification_report
```

In [57]:

```
print(classification_report(y_test,predictions))
```

```
               precision    recall  f1-score   support

           0       1.00      0.93      0.97        15
           1       0.91      1.00      0.95        10

    accuracy                           0.96        25
   macro avg       0.95      0.97      0.96        25
weighted avg       0.96      0.96      0.96        25
```

CONCLUSION: Logmodel is the best for this data

In [ ]: