Background

Use supervised learning techniques to automate the credit card approval process for banks.

- Preproccess the data and apply supervised learning techniques to find the best model and parameters for the job. Save the accuracy score from your best model as a numeric variable, `best_score`. Aim for an accuracy score of at least `0.75`. The target variable is the last column of the DataFrame.

Commercial banks receive *a lot* of applications for credit cards. Many of them get rejected for many reasons, like high loan balances, low income levels, or too many inquiries on an individual's credit report, for example. Manually analyzing these applications is mundane, error-prone, and time-consuming (and time is money!). Luckily, this task can be automated with the power of machine learning and pretty much every commercial bank does so nowadays. In this workbook, you will build an automatic credit card approval predictor using machine learning techniques, just like real banks do.

The data is a small subset of the Credit Card Approval dataset from the UCI Machine Learning Repository showing the credit card applications a bank receives. This dataset has been loaded as a `pandas` DataFrame called `cc_apps`. The last column in the dataset is the target value.

In [4]:
```
# Import necessary libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import GridSearchCV

# Load the dataset
cc_apps = pd.read_csv("../data_raw/creditcard_approvals.data", header=None)
cc_apps
```

Out[4]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	b	30.83	0.000	u	g	w	v	1.25	t	t	1	g	0	+
1	a	58.67	4.460	u	g	q	h	3.04	t	t	6	g	560	+
2	a	24.50	0.500	u	g	q	h	1.50	t	f	0	g	824	+
3	b	27.83	1.540	u	g	w	v	3.75	t	t	5	g	3	+
4	b	20.17	5.625	u	g	w	v	1.71	t	f	0	s	0	+
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
685	b	21.08	10.085	y	p	e	h	1.25	f	f	0	g	0	-
686	a	22.67	0.750	u	g	c	v	2.00	f	t	2	g	394	-
687	a	25.25	13.500	y	p	ff	ff	2.00	f	t	1	g	1	-
688	b	17.92	0.205	u	g	aa	v	0.04	f	f	0	g	750	-
689	b	35.00	3.375	u	g	c	h	8.29	f	f	0	g	0	-

690 rows × 14 columns

In [5]:
```
# Replace the '?'s with NaN in dataset
cc_apps_nans_replaced = cc_apps.replace("?", np.nan)

# Create a copy of the NaN replacement DataFrame
cc_apps_imputed = cc_apps_nans_replaced.copy()

# Iterate over each column of cc_apps_nans_replaced and impute the most frequent value for object data types and the mean for numeric data types
for col in cc_apps_imputed.columns:
    # Check if the column is of object type
    if cc_apps_imputed[col].dtypes == "object":
        # Impute with the most frequent value
        cc_apps_imputed[col] = cc_apps_imputed[col].fillna(
            cc_apps_imputed[col].value_counts().index[0]
        )
    else:
        cc_apps_imputed[col] = cc_apps_imputed[col].fillna(cc_apps_imputed[col].mean())

# Dummify the categorical features
cc_apps_encoded = pd.get_dummies(cc_apps_imputed, drop_first=True)

# Extract the last column as your target variable
X = cc_apps_encoded.iloc[:, :-1].values
y = cc_apps_encoded.iloc[:, [-1]].values

# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)

# Instantiate StandardScaler and use it to rescale X_train and X_test
scaler = StandardScaler()
rescaledX_train = scaler.fit_transform(X_train)
rescaledX_test = scaler.transform(X_test)

# Instantiate a LogisticRegression classifier with default parameter values
logreg = LogisticRegression()

# Fit logreg to the train set
logreg.fit(rescaledX_train, y_train.ravel())

# Use logreg to predict instances from the training set
y_train_pred = logreg.predict(rescaledX_train)

# Print the confusion matrix of the logreg model
print(confusion_matrix(y_train, y_train_pred))
```

```
[[203   1]
 [  1 257]]
```

In [6]:
```
# Define the grid of values for tol and max_iter
tol = [0.01, 0.001, 0.0001]
max_iter = [100, 150, 200]

# Create a dictionary where tol and max_iter are keys and the lists of their values are the corresponding values
param_grid = dict(tol=tol, max_iter=max_iter)

# Instantiate GridSearchCV with the required parameters
grid_model = GridSearchCV(estimator=logreg, param_grid=param_grid, cv=5)

# Fit grid_model to the data
grid_model_result = grid_model.fit(rescaledX_train, y_train.ravel())

# Summarize results
best_train_score, best_train_params = grid_model_result.best_score_, grid_model_result.best_params_
print("Best: %f using %s" % (best_train_score, best_train_params))

# Extract the best model and evaluate it on the test set
best_model = grid_model_result.best_estimator_
best_score =  best_model.score(rescaledX_test, y_test)

print("Accuracy of logistic regression classifier: ", best_score)
```

```
Best: 0.818256 using {'max_iter': 100, 'tol': 0.01}
Accuracy of logistic regression classifier:  0.7982456140350878
```