1. Importing Libraries, Data and Data Cleaining

Download the files here: https://www.kaggle.com/datasets/rikdifos/credit-card-approval-prediction/download?datasetVersionNumber=3

application_record.csv contains a set of 18 features about **438,557** credit applicants. Some of which are duplicate records or can contain missing values. As shown in the steps below, after removing duplicates and records with missing data, the table is reduced to **304,317** applicants.

credit_record.csv contains historical payment data, identified as follows:

These two tables can be connected by the common column **ID**. However, not all applicants have data in the credit_record table. So leaving the intersect of the two datasets down to (xxx).

In [ ]:
```
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import KFold, cross_val_score
from sklearn.metrics import confusion_matrix
```

In [ ]:
```
application_record = pd.read_csv("application_record.csv")
credit_record = pd.read_csv("credit_record.csv")
```

1.1 Removing duplicate records and missing values

In [ ]:
```
# Removing duplicate records
print("Before removing duplicate records")
print("Number of rows and columns:", application_record.shape)
print("Number of duplicated records:", application_record.duplicated(subset=

application_record = application_record.drop_duplicates(subset=["ID"])
print("\nAfter removing duplicate records")
print("Number of rows and columns:", application_record.shape)
print("Number of duplicated records:", application_record.duplicated(subset=
print(application_record.nunique())
```

```
Before removing duplicate records
Number of rows and columns: (438557, 18)
Number of duplicated records: 47

After removing duplicate records
Number of rows and columns: (438510, 18)
Number of duplicated records: 0
ID                     438510
CODE_GENDER                 2
FLAG_OWN_CAR                2
FLAG_OWN_REALTY             2
CNT_CHILDREN               12
AMT_INCOME_TOTAL          866
NAME_INCOME_TYPE            5
NAME_EDUCATION_TYPE         5
NAME_FAMILY_STATUS          5
NAME_HOUSING_TYPE           6
DAYS_BIRTH              16379
DAYS_EMPLOYED            9406
FLAG_MOBIL                  1
FLAG_WORK_PHONE             2
FLAG_PHONE                  2
FLAG_EMAIL                  2
OCCUPATION_TYPE            18
CNT_FAM_MEMBERS            13
dtype: int64
```

In [ ]:
```
# Dealing with missing records
print(application_record.isna().sum())
```

```
ID                          0
CODE_GENDER                 0
FLAG_OWN_CAR                0
FLAG_OWN_REALTY             0
CNT_CHILDREN                0
AMT_INCOME_TOTAL            0
NAME_INCOME_TYPE            0
NAME_EDUCATION_TYPE         0
NAME_FAMILY_STATUS          0
NAME_HOUSING_TYPE           0
DAYS_BIRTH                  0
DAYS_EMPLOYED               0
FLAG_MOBIL                  0
FLAG_WORK_PHONE             0
FLAG_PHONE                  0
FLAG_EMAIL                  0
OCCUPATION_TYPE        134193
CNT_FAM_MEMBERS             0
dtype: int64
```

As shown above, the only column in the application_record table that contains missing data is OCCUPATION_TYPE, with a total of 134,193 records. The data dictionary does not provide any information on why this data missing or if there is any reason behind it, for example, being unemployed. Therefore, instead of removing all those records from the dataset, let's replace those missing values with a 'Not disclosed' label.

In [ ]:
```
application_record["OCCUPATION_TYPE"].fillna(value='Not disclosed', inplace=
print(application_record['OCCUPATION_TYPE'].value_counts(dropna=False))
```

```
Not disclosed           134193
Laborers                 78231
Core staff               43000
Sales staff              41094
Managers                 35481
Drivers                  26090
High skill tech staff    17285
Accountants              15983
Medicine staff           13518
Cooking staff             8076
Security staff            7993
Cleaning staff            5843
Private service staff     3455
Low-skill Laborers        2140
Secretaries               2044
Waiters/barmen staff      1665
Realty agents             1041
HR staff                   774
IT staff                   604
Name: OCCUPATION_TYPE, dtype: int64
```

As each client normally uses the credit card for several months, duplicate values in the ID column of the *credit_record* table are expected. And as shown below, there are no missing data in this table.

In [ ]:
```
# Checking for duplicate and missing values in the credit_record data
print(credit_record.shape)
print(credit_record.nunique())
credit_record.isna().sum()
```

```
(1048575, 3)
ID                45985
MONTHS_BALANCE       61
STATUS                8
dtype: int64
```

Out[ ]:
```
ID                0
MONTHS_BALANCE    0
STATUS            0
dtype: int64
```

Looking at the difference between the number of IDs on the credit_record table (45,985) and the application_record (438,510), we note that not all applicants have a credit history. Although the reason behind this is not disclosed in the data source, 3 possible reasons could explain the difference: (1) Some records could be data from new accounts, that haven't had any closed statements as of the date when the data was extracted; (2) Closed accounts for which the last closed statement was outside the data credit_record time frame; Or (3) all the bank's client data could be stored in the same datasource and not all clients might have credit cards associated with their accounts.

Even if these explanations are only especulative, not having a credit record won't allow a client to be classified as good or bad creditor, and therefore, these individuals are not part of this exercise and must be removed from the dataset. This step will be addressed in the client classification variable

In [ ]:
```
# Finding the intersect between the two tables.
len(set(application_record['ID']).intersection(set(credit_record['ID']))) #
```

Out[ ]:
```
36457
```

1.2 Creating the client classification variable

The **credit_record** table contains historical payment data identified as follows:

- 0: 1-29 days past due
- 1: 30-59 days past due
- 2: 60-89 days overdue
- 3: 90-119 days overdue
- 4: 120-149 days overdue
- 5: Overdue or bad debts, write-offs for more than 150 days
- C: paid off that month
- X: No loan for the month

To construct the classifier label, creditors that have no overdue or bad debts, will be classified with a label 'Good'. On the contrary, i.e. if there are any payments identified with the number 5 in the status column, they will be classified as 'Bad'.

In [ ]:
```
# classifying good or bad creditors
classification = []
for client in application_record["ID"]:
    client_history = credit_record["STATUS"][credit_record["ID"] == client]
    if len(client_history) > 0:
        flag = 'Good'
        for payment_status in client_history:
            if payment_status == '5':
                flag = 'Bad'
                break
    else:
        flag = 'No data'
    classification.append(flag)

application_record['CLASSIFICATION'] = (classification)
```

In [ ]:
```
application_record['CLASSIFICATION'].value_counts()
```

Out[ ]:
```
No data    402053
Good        36277
Bad           180
Name: CLASSIFICATION, dtype: int64
```

In [ ]:
```
# Review classification results
common_rows = pd.merge(application_record, credit_record, on='ID')
common_rows.drop_duplicates(subset=["ID"], inplace=True)
num_common_rows = len(common_rows)
print(num_common_rows)

# Expected number of good and bad creditors, vs no data
print(36277 + 180)

# application_record.to_csv("check.csv")
```

```
36457
36457
```

In [ ]:
```
# Removing creditors that have no credit history
df = application_record[application_record['CLASSIFICATION'] != 'No data']
print(df['CLASSIFICATION'].value_counts())
```

```
Good    36277
Bad       180
Name: CLASSIFICATION, dtype: int64
```

1.3 Removing variables

ID - The ID variable can be removed because it's a unique number assigned to each applicant and will not add value to the analysis.

FLAG_MOBIL - Checking the variable value counts, we can see that the variable FLAG_MOBIL, which is a boolean variable that identifies if there is a mobile phone registered in the client's account, only has true values and therefore, it can be removed from our analysis.

In [ ]:
```
df.FLAG_MOBIL.value_counts() # only one category
df.drop(columns=['ID', 'FLAG_MOBIL'], inplace=True)
```

```
/var/folders/yg/5ld4yk153p7419g8frs0z3qh0000gn/T/ipykernel_61239/1174668487
.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs
/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df.drop(columns=['ID', 'FLAG_MOBIL'], inplace=True)
```

As shown below, after cleaning the data, the dataset contains 36,457 rows and 17 columns.

In [ ]:
```
# Final dataset shape and data types
print(df.shape)
```

```
(36457, 17)
```

In [ ]:
```
# Defining X and y
X = df.drop(columns='CLASSIFICATION')
y = df['CLASSIFICATION']
#print(X.dtypes)
#print(y.dtypes)
```

1.4 Converting data types

In [ ]:
```
y.dtype
```

Out[ ]:
```
dtype('O')
```

In [ ]:
```
y = y.replace({'Good': 1, 'Bad': 0})
X = pd.get_dummies(X)
```

In [ ]:
```
# X.CODE_GENDER = X.CODE_GENDER.astype('category')
# X.FLAG_OWN_CAR = X.FLAG_OWN_CAR.astype('category')
# X.FLAG_OWN_REALTY = X.FLAG_OWN_REALTY.astype('category')

# X.NAME_INCOME_TYPE = X.NAME_INCOME_TYPE.astype('category')
# X.NAME_EDUCATION_TYPE = X.NAME_EDUCATION_TYPE.astype('category')
# X.NAME_FAMILY_STATUS = X.NAME_FAMILY_STATUS.astype('category')
# X.NAME_HOUSING_TYPE = X.NAME_HOUSING_TYPE.astype('category')

# X.FLAG_WORK_PHONE = X.FLAG_WORK_PHONE.astype('bool')
# X.FLAG_PHONE = X.FLAG_PHONE.astype('bool')
# X.FLAG_EMAIL = X.FLAG_EMAIL.astype('bool')

# X.OCCUPATION_TYPE = X.OCCUPATION_TYPE.astype('category')
# X.CNT_FAM_MEMBERS = X.CNT_FAM_MEMBERS.astype('int')
```

In [ ]:
```
print(X.dtypes)
```

```
CNT_CHILDREN                                          int64
AMT_INCOME_TOTAL                                    float64
DAYS_BIRTH                                            int64
DAYS_EMPLOYED                                         int64
FLAG_WORK_PHONE                                       int64
FLAG_PHONE                                            int64
FLAG_EMAIL                                            int64
CNT_FAM_MEMBERS                                     float64
CODE_GENDER_F                                         uint8
CODE_GENDER_M                                         uint8
FLAG_OWN_CAR_N                                        uint8
FLAG_OWN_CAR_Y                                        uint8
FLAG_OWN_REALTY_N                                     uint8
FLAG_OWN_REALTY_Y                                     uint8
NAME_INCOME_TYPE_Commercial associate                 uint8
NAME_INCOME_TYPE_Pensioner                            uint8
NAME_INCOME_TYPE_State servant                        uint8
NAME_INCOME_TYPE_Student                              uint8
NAME_INCOME_TYPE_Working                              uint8
NAME_EDUCATION_TYPE_Academic degree                   uint8
NAME_EDUCATION_TYPE_Higher education                  uint8
NAME_EDUCATION_TYPE_Incomplete higher                 uint8
NAME_EDUCATION_TYPE_Lower secondary                   uint8
NAME_EDUCATION_TYPE_Secondary / secondary special     uint8
NAME_FAMILY_STATUS_Civil marriage                     uint8
NAME_FAMILY_STATUS_Married                            uint8
NAME_FAMILY_STATUS_Separated                          uint8
NAME_FAMILY_STATUS_Single / not married               uint8
NAME_FAMILY_STATUS_Widow                              uint8
NAME_HOUSING_TYPE_Co-op apartment                     uint8
NAME_HOUSING_TYPE_House / apartment                   uint8
```

```
NAME_HOUSING_TYPE_Municipal apartment                    uint8
NAME_HOUSING_TYPE_Office apartment                       uint8
NAME_HOUSING_TYPE_Rented apartment                       uint8
NAME_HOUSING_TYPE_With parents                           uint8
OCCUPATION_TYPE_Accountants                              uint8
OCCUPATION_TYPE_Cleaning staff                           uint8
OCCUPATION_TYPE_Cooking staff                            uint8
OCCUPATION_TYPE_Core staff                               uint8
OCCUPATION_TYPE_Drivers                                  uint8
OCCUPATION_TYPE_HR staff                                 uint8
OCCUPATION_TYPE_High skill tech staff                    uint8
OCCUPATION_TYPE_IT staff                                 uint8
OCCUPATION_TYPE_Laborers                                 uint8
OCCUPATION_TYPE_Low-skill Laborers                       uint8
OCCUPATION_TYPE_Managers                                 uint8
OCCUPATION_TYPE_Medicine staff                           uint8
OCCUPATION_TYPE_Not disclosed                            uint8
OCCUPATION_TYPE_Private service staff                    uint8
OCCUPATION_TYPE_Realty agents                            uint8
OCCUPATION_TYPE_Sales staff                              uint8
OCCUPATION_TYPE_Secretaries                              uint8
OCCUPATION_TYPE_Security staff                           uint8
OCCUPATION_TYPE_Waiters/barmen staff                     uint8
dtype: object
```

2. Cross validation

In [ ]:

```python
# Creating a 10-Fold cross-validation object
kf = KFold(n_splits=10, shuffle=True, random_state=1)

knn = KNeighborsClassifier()
logreg = LogisticRegression()
dt = DecisionTreeClassifier()

# Performing 10-fold cross-validation for each model
knn_scores = cross_val_score(knn, X, y, cv=kf)
logreg_scores = cross_val_score(logreg, X, y, cv=kf)
dt_scores = cross_val_score(dt, X, y, cv=kf)

# print mean and standard deviation of each model's scores
print("KNN Accuracy: %0.2f (+/- %0.2f)" % (knn_scores.mean(), knn_scores.std
print("Logistic Regression Accuracy: %0.2f (+/- %0.2f)" % (logreg_scores.mea
print("Decision Tree Accuracy: %0.2f (+/- %0.2f)" % (dt_scores.mean(), dt_sc
```

```
KNN Accuracy: 0.99 (+/- 0.00)
Logistic Regression Accuracy: 1.00 (+/- 0.00)
Decision Tree Accuracy: 0.99 (+/- 0.00)
```