Loan Predictor

T.A: Verena Nashaat		
Team members		
Omar Khaled	**20201700534**	**Sec. 19**
Aya Osama	**20201700170**	**Sec. 7**
Karim Mohamed	**20201700608**	**Sec. 21**
Bassant Hossam	**20201701232**	**Sec. 8**
Usama Alaa	**20201700105**	**Sec. 4**
Hosam Hatem	**20201701235**	**Sec. 9**

Table of Contents

Introduction

I. State topic

Banks' primary business is lending. The main source of profit is the interest on the loan.

After an extensive verification and validation process, the loan companies grant a loan, but they do not have assurance that the applicant will be able to repay the loan without difficulty.

Thus, our loan predictor has been built to help loan companies in deciding whether to accept or reject a requested loan from an applicant. The loan predictor can achieve this as it applies AI concepts, data analysis techniques, in addition to performing a comparison between four Algorithms - Logistic Regression, Support Vector Machine, Decision Tree ID3 - KNN to achieve the best accuracy overall.

II. Project goals

- The project's purpose is to forecast if an applicant can take out a loan or not based on the available features such as Loan amount, Material status, Gender, Number of dependents, and Credit history.

III. Directions

1) Data Preprocessing.
   - A. Data Cleansing.
     1. Handling Nulls.
     2. Categorical Values.
     3. Remove Extreme Values.
2) Machine Learning Models Comparison.
   - a. Model Building.
   - b. Model Evaluation.
   - c. Model Optimization.

Data definition

I. Dataset

We used a dataset with **614** records of applicant s and **13** features for each one of them, which will be discussed in the next part.

Dataset Sample:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status
0	LP001002	Male	No	0	Graduate	No	5849	0.0	NaN	360.0	1.0	Urban	Y
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.0	360.0	1.0	Rural	N
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66.0	360.0	1.0	Urban	Y
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	360.0	1.0	Urban	Y
4	LP001008	Male	No	0	Graduate	No	6000	0.0	141.0	360.0	1.0	Urban	Y

Dataset Info:

```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Loan_ID            614 non-null    object
 1   Gender             601 non-null    object
 2   Married            611 non-null    object
 3   Dependents         599 non-null    object
 4   Education          614 non-null    object
 5   Self_Employed      582 non-null    object
 6   ApplicantIncome    614 non-null    int64
 7   CoapplicantIncome  614 non-null    float64
 8   LoanAmount         592 non-null    float64
 9   Loan_Amount_Term   600 non-null    float64
 10  Credit_History     564 non-null    float64
 11  Property_Area      614 non-null    object
 12  Loan_Status        614 non-null    object
dtypes: float64(4), int64(1), object(8)
memory usage: 62.5+ KB
```

II. Attributes

- Loan_ID
  This column is filled with the Loan ID.
  Data Type: String - Categorical (Nominal).
  Unique Values: […] **614** Values.

- Gender
  This column is filled with the applicant 's gender.
  Data Type: String - Categorical (Nominal).
  Unique Values: [Male, Female] **2** Values.

- Married
  This column is filled with the applicant 's material states.
  Data Type: String - Categorical (Nominal).
  Unique Values: [Yes, No] **2** Values.

- Dependents
  This column determines if the applicant has dependents.
  Data Type: Integer - Quantitative (Discrete)
  Unique Values: [0, …, 3+] **4** Values.

- Education

  This column is filled with the applicant 's educational states.
  Data Type: String - Categorical (Nominal).
  Unique Values: [Not Graduate, Graduate] **2** Values.

- Self-Employed
  This column is filled with the applicant 's Self-Employment states.
  Data Type: String - Categorical (Nominal).
  Unique Values: [Yes, No] **2** Values.

- Applicant Income
  This column determines the applicant income.
  Data Type: Integer - Quantitative (Discrete)
  Unique Values: [150, …, 81000] **505** Values.

- Coapplicant Income
  This column determines the Coapplicant income.
  Data Type: Integer - Quantitative (Discrete)
  Unique Values: [0, …, 41667] **287** Values.

- Loan Amount
  This column determines the Loan Amount.
  Data Type: Integer - Quantitative (Discrete)
  Unique Values: [9, …, 700] **204** Values.

- Loan Amount Term
  This column determines the Loan Amount Term.
  Data Type: Integer - Quantitative (Discrete)
  Unique Values: [12, …, 480] **11** Values.

- Credit History
  This column determines the Credit History of the applicant.
  Data Type: Integer - Quantitative (Discrete)
  Unique Values: [0,1] **2** Values.

- Property Area
  This column determines the Property Area.
  Data Type: String - Categorical (Nominal).
  Unique Values: [Urban, Rural, Semiurban] **3** Values.

- Loan Status
  This column determines the Loan Status.
  Data Type: String - Categorical (Nominal).
  Unique Values: [Yes, No] **2** Values.

Data Cleansing

I. Mapping String Values

The features should be in numerical form to apply formulas, descriptive analysis, and Machine Learning Techniques, therefore converting strings (categorical variables) into integer values with one label encoding approach will prepare them for the next procedures.

It's worth mentioning that those numbers don't have any mathematical meaning.

II. Handling Missing Values

Variables having missing values, or NULL values, must be dealt with either eliminating them or using mean/median/mode, or another advanced technique to assign a numeric value to them.
We didn’t have to implement a code to handle the nulls as the Label encoder can handle them.

Data before label encoding

```
train.info()
train.isnull().sum()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Loan_ID            614 non-null    object
 1   Gender             601 non-null    object
 2   Married            611 non-null    object
 3   Dependents         599 non-null    object
 4   Education          614 non-null    object
 5   Self_Employed      582 non-null    object
 6   ApplicantIncome    614 non-null    int64
 7   CoapplicantIncome  614 non-null    float64
 8   LoanAmount         592 non-null    float64
 9   Loan_Amount_Term   600 non-null    float64
 10  Credit_History     564 non-null    float64
 11  Property_Area      614 non-null    object
 12  Loan_Status        614 non-null    object
dtypes: float64(4), int64(1), object(8)
memory usage: 62.5+ KB
Loan_ID               0
Gender               13
Married               3
Dependents           15
Education             0
Self_Employed        32
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount           22
Loan_Amount_Term     14
Credit_History       50
Property_Area         0
Loan_Status           0
dtype: int64
```

Data after label encoding

```
train.info()
train.isnull().sum()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Loan_ID            614 non-null    int64
 1   Gender             614 non-null    int64
 2   Married            614 non-null    int64
 3   Dependents         614 non-null    int64
 4   Education          614 non-null    int64
 5   Self_Employed      614 non-null    int64
 6   ApplicantIncome    614 non-null    int64
 7   CoapplicantIncome  614 non-null    int64
 8   LoanAmount         614 non-null    int64
 9   Loan_Amount_Term   614 non-null    int64
 10  Credit_History     614 non-null    int64
 11  Property_Area      614 non-null    int64
 12  Loan_Status        614 non-null    int64
dtypes: int64(13)
memory usage: 62.5 KB
Loan_ID              0
Gender               0
Married              0
Dependents           0
Education            0
Self_Employed        0
ApplicantIncome      0
CoapplicantIncome    0
LoanAmount           0
Loan_Amount_Term     0
Credit_History       0
Property_Area        0
Loan_Status          0
dtype: int64
```

III. Dropping loan ID

Note that as the loan ID changes for each applicant and doesn’t relate to the loan status in any way, dropping it will not affect the prediction accuracy.

IV. Remove duplicated records

Overfitting happens when a model learns the detail and noise in the training data to the extent that it negatively impacts the performance of the model on new data, what happens when there are duplicate records in the dataset. So, to avoid model overfitting, we will check if there are any duplicated records, and if we found duplicates, we will remove all of them from the dataset.

V. Remove Extreme Values

The range and distribution of attribute values are important to machine learning algorithms. Outliers and extremes in data can poison and mislead the training process, resulting in longer training times, less accurate models, and, ultimately, weaker results.
An Extreme can easily be detected using the visualization technique via Box Plot where any point above or below the whiskers represents an outlier.

Conducting this technique on the Loan Amount Term, Loan Amount, Coapplicant Income, Applicant Income, and columns show that there aren't any extreme values on these features.

Reaching this point, now the cleansing phase has been finished and the dataset is ready for the next phase.

Used algorithms

Support Vector Machine (SVM)

is a relatively simple **Supervised Machine Learning Algorithm** used for classification.

SVM CODE:

```
[ ] support=svm.SVC(kernel='linear', C=1)
    support.fit(xtrain, ytrain)
    #print(support.score(xtrain,ytrain))
    yprd=support.predict(xtest)
    print("Accuracy : ", metrics.accuracy_score(ytest, yprd))
    print("precision : ", metrics.precision_score(ytest, yprd))
    print("Recall : ", metrics.recall_score(ytest, yprd))
```

Achieved Accuracy: 0.8378378378378378
Achieved precision: 0.8238993710691824
Achieved Recall: 0.9849624060150376

Logistic Regression

Logistic Regression is a “Supervised machine learning” algorithm that can be used to model the probability of a certain class or event.

Logistic Regression Code:

```
[ ] model = LogisticRegression(solver='liblinear', C=10, random_state=0)
    model.fit(xtrain, ytrain)
    yprd = model.predict(xtest)
    #print(model.score(xtest,ytest))

    print("Accuracy : ", metrics.accuracy_score(ytest, yprd))
    print("precision : ", metrics.precision_score(ytest, yprd))
    print("Recall : ", metrics.recall_score(ytest, yprd))
    print(confusion_matrix(ytest, yprd))
```

Achieved Accuracy: 0.8378378378378378
Achieved precision: 0.8238993710691824
Achieved Recall: 0.9849624060150376

ID3

In decision tree learning, ID3 (Iterative Dichotomiser 3) is an algorithm used to generate a decision tree from a dataset.

ID3 Code:

ID3

```
[ ] dt=tree.DecisionTreeClassifier(max_depth=2)
    dt.fit(xtrain, ytrain)
    yprd = dt.predict(xtest)
    # print(dt.score(xtest, ytest))
    print("Accuracy : ", metrics.accuracy_score(ytest, yprd))
    print("precision : ", metrics.precision_score(ytest, yprd))
    print("Recall : ", metrics.recall_score(ytest, yprd))
```

Accuracy: 0.8378378378378378
precision: 0.8238993710691824
Recall: 0.9849624060150376

KNN

KNN also called K- nearest neighbour is a supervised machine learning algorithm that can be used for classification and regression problems.

CODE:

KNN

```
[ ]  knn = KNeighborsClassifier(n_neighbors=7)
     knn.fit(xtrain, ytrain)
     yprd = knn.predict(xtest)
     print("Accuracy : ", metrics.accuracy_score(ytest, yprd))
     print("precision : ", metrics.precision_score(ytest, yprd))
     print("Recall : ", metrics.recall_score(ytest, yprd))
```

Accuracy: 0.772972972972973
precision: 0.7861635220125787
Recall: 0.9398496240601504