# Loan Prediction Project

**Members:**

1) Samy Samer
2) Mina Nabil
3) Marina Atef
4) Maria Medhat
5) Sandra Alfy
6) Michel Magdy

# 1) Pre-Processing:

First, we removed duplicates then replacing the string by numeric values.

```
#data cleaning

#drop duplicates
data.drop_duplicates()

#replacing strings by numeric values
data['Loan_ID'] = data['Loan_ID'].str.replace('LP', '')
data["Gender"]=data["Gender"].replace(['Female','Male'],[0,1])
data["Married"]=data["Married"].replace(['No','Yes'],[0,1])
data["Education"]=data["Education"].replace(['Not Graduate','Graduate'],[0,1])
data["Self_Employed"]=data["Self_Employed"].replace(['No','Yes'],[0,1])
data["Property_Area"]=data["Property_Area"].replace(['Urban','Rural','Semiurban'],[0,1,2])
data["Loan_Status"]=data["Loan_Status"].replace(['N','Y'],[0,1])
data["Dependents"] = data["Dependents"].replace('3+',3)
```

We found two columns with different data types.

```
data["Dependents"]=data["Dependents"].astype(float)
data['Loan_ID'] = data['Loan_ID'].astype(float)
```

Then calculated the correlation to know data which we need to drop.

```
#calculate the correlation
data.corr()
```

| | Loan_ID | Gender | Married | Dependents | Education | Self_Employed | ApplicantIncome | CoapplicantIncome | LoanAmount | Loan_Amount_Term | Credit_History | Property_Area | Loan_Status |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Loan_ID | 1.000000 | -0.031415 | -0.014705 | 0.054679 | -0.037547 | 0.033295 | 0.019331 | 0.039516 | 0.038447 | -0.030481 | -0.032910 | 0.072041 | 0.011958 |
| Gender | -0.031415 | 1.000000 | 0.369612 | 0.175970 | -0.049258 | -0.009829 | 0.053989 | 0.083946 | 0.106947 | -0.075117 | 0.016337 | -0.085306 | 0.019857 |
| Married | -0.014705 | 0.369612 | 1.000000 | 0.343417 | -0.014223 | 0.001909 | 0.051332 | 0.077770 | 0.149519 | -0.103810 | 0.004381 | 0.001875 | 0.089280 |
| Dependents | 0.054679 | 0.175970 | 0.343417 | 1.000000 | -0.059161 | 0.057867 | 0.118679 | 0.027259 | 0.163997 | -0.100484 | -0.050082 | -0.000813 | 0.006781 |
| Education | -0.037547 | -0.049258 | -0.014223 | -0.059161 | 1.000000 | 0.012333 | 0.140760 | 0.062290 | 0.171133 | 0.078784 | 0.081822 | 0.003592 | 0.085884 |
| Self_Employed | 0.033295 | -0.009829 | 0.001909 | 0.057867 | 0.012333 | 1.000000 | 0.140826 | -0.011152 | 0.123931 | -0.037069 | 0.003883 | 0.019688 | -0.002303 |
| ApplicantIncome | 0.019331 | 0.053989 | 0.051332 | 0.118679 | 0.140760 | 0.140826 | 1.000000 | -0.116605 | 0.570909 | -0.045306 | -0.014715 | -0.007894 | -0.004710 |
| CoapplicantIncome | 0.039516 | 0.083946 | 0.077770 | 0.027259 | 0.062290 | -0.011152 | -0.116605 | 1.000000 | 0.188619 | -0.059878 | -0.002056 | -0.028356 | -0.059187 |
| LoanAmount | 0.038447 | 0.106947 | 0.149519 | 0.163997 | 0.171133 | 0.123931 | 0.570909 | 0.188619 | 1.000000 | 0.039447 | -0.008433 | 0.014074 | -0.037318 |
| Loan_Amount_Term | -0.030481 | -0.075117 | -0.103810 | -0.100484 | 0.078784 | -0.037069 | -0.045306 | -0.059878 | 0.039447 | 1.000000 | 0.001470 | 0.090610 | -0.021268 |
| Credit_History | -0.032910 | 0.016337 | 0.004381 | -0.050082 | 0.081822 | 0.003883 | -0.014715 | -0.002056 | -0.008433 | 0.001470 | 1.000000 | 0.037822 | 0.561678 |
| Property_Area | 0.072041 | -0.085306 | 0.001875 | -0.000813 | 0.003592 | 0.019688 | -0.007894 | -0.028356 | 0.014074 | 0.090610 | 0.037822 | 1.000000 | 0.103253 |
| Loan_Status | 0.011958 | 0.019857 | 0.089280 | 0.006781 | 0.085884 | -0.002303 | -0.004710 | -0.059187 | -0.037318 | -0.021268 | 0.561678 | 0.103253 | 1.000000 |

According to correlation we dropped the useless features.

```
data.drop(['Loan_ID'],axis=1,inplace=True)
data.drop(['Gender'],axis=1,inplace=True)
```

Replace Null values in a column with mode of column values.

```
data['Married'].fillna(value=data['Married'].mode()[0], inplace=True)
data['Dependents'].fillna( data.Dependents.mode()[0] ,inplace = True )
data['Self_Employed'].fillna(value=data['Self_Employed'].mode()[0], inplace=True)
data['Loan_Amount_Term'].fillna(value=data['Loan_Amount_Term'].mode()[0], inplace=True)
data['LoanAmount'].fillna(value=data['LoanAmount'].mode()[0], inplace=True)
data['Credit_History'].fillna(value=data['Credit_History'].mode()[0], inplace=True)
```

separate array into input and output components.

```
Y = data["Loan_Status"]
X = data.drop("Loan_Status", axis=1)
#split our data to train and test
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.3,random_state=1)
```

Then we made data scaling to set range of values between 0 and 1 to make Y depends on weight of X only.

```
scaler = MinMaxScaler()
X_train = pd.DataFrame(scaler.fit_transform(X_train), columns=X_train.columns)
X_test = pd.DataFrame(scaler.fit_transform(X_test), columns=X_test.columns)
```

Principle components analysis (PCA) for feature extraction.

```
pca = PCA(n_components=3)
pca_train = pca.fit_transform(X_train)
pca_test = pca.fit_transform(X_test)
```

# 2) Algorithms:

## A) Logistic Regression:

Before Feature extraction:

We set

**Solver by Liblinear**: because it is a simple package for solving large-scale regularized linear classification, regression and outlier detection.

**C by 1**: as it's average value as by increasing value of C overfitting occurs and by decreasing value of C underfitting occurs.

```
#Logistic Regression before feature extraction
L_model = LogisticRegression(solver='liblinear',C=1.0,random_state=1)
L_model.fit(X_train,Y_train)
model_accuracy=L_model.score(X_train,Y_train)
print("the Logistic Regression accuracy before feature extraction is :",model_accuracy)
```

✓ 0.7s

```
the Logistic Regression accuracy before feature extraction is : 0.8181818181818182
```

After feature extraction:

```
#Logistic Regression after feature extraction
L_model.fit(pca_train,Y_train)
model_accuracy=L_model.score(pca_test,Y_test)
print("the Logistic Regression accuracy after feature extraction is :",model_accuracy)
```

✓ 0.8s

```
the Logistic Regression accuracy after feature extraction is : 0.6108108108108108
```

## B) SVM:

Before feature extraction:

We set:

**Kernel by poly**: we found it was the best choice for our data with best accuracy.

```
#SVM before feature extraction
S_model = svm.SVC(kernel='poly')
S_model.fit(X_train,Y_train)
model_accuracy=S_model.score(X_test,Y_test)
print("the SVM accuracy before feature extraction is :",model_accuracy)
```

✓ 0.1s

```
the SVM accuracy before feature extraction is : 0.7891891891891892
```

After feature extraction:

```
#SVM after feature extraction
S_model.fit(pca_train,Y_train)
model_accuracy=S_model.score(pca_test,Y_test)
print("the SVM accuracy after feature extraction is :",model_accuracy)
```

✓ 0.3s

```
the SVM accuracy after feature extraction is : 0.6756756756756757
```

## C) Decision tree:

Before feature extraction:

We used **max_depth**: it represents the number of levels of tree as we used an average number to avoid overfitting with best accuracy.

```
#Decision tree before feature extraction
D_model= tree.DecisionTreeClassifier(max_depth = 3)
D_model.fit(X_train,Y_train)
model_accuracy=D_model.score(X_test,Y_test)
print("the Decision tree accuracy before feature extraction is :",model_accuracy)
```

✓ 0.2s

```
the Decision tree accuracy before feature extraction is : 0.7783783783783784
```

After feature extraction:

```
#Decision tree after feature extraction
D_model.fit(pca_train,Y_train)
model_accuracy=D_model.score(pca_test,Y_test)
print("the Decision tree accuracy after feature extraction is :",model_accuracy)
```

✓ 0.1s

```
the Decision tree accuracy after feature extraction is : 0.6864864864864865
```

# Bonus Algorithms:

## A) Random Forest:

Before feature extraction:

we used:

**n_estimators**: as it represents number of trees in forest and we used the value 100 as it was best for our accuracy.

**Max_depth**: it represents the number of levels of forest as we used an average number to avoid overfitting with best accuracy.

```
#Random Forest before feature extraction
R_model=RandomForestClassifier(n_estimators=100,max_depth=5)
R_model.fit(X_train,Y_train)
model_accuracy=R_model.score(X_test,Y_test)
print("the Random Forest accuracy before feature extraction is :",model_accuracy)
```

✓ 0.6s

```
the Random Forest accuracy before feature extraction is : 0.7837837837837838
```

After feature extraction:

```
#Random Forest after feature extraction
R_model.fit(pca_train,Y_train)
model_accuracy=R_model.score(pca_test,Y_test)
print("the Random Forest accuracy after feature extraction is :",model_accuracy)
```

✓ 0.5s

```
the Random Forest accuracy after feature extraction is : 0.6324324324324324
```

## B) Naïve bayes:

Before feature extraction:

```
#Naive bayes before feature extraction
N_model= GaussianNB()
N_model.fit(X_train,Y_train)
model_accuracy=N_model.score(X_test,Y_test)
print("the Naive bayes accuracy before feature extraction is :",model_accuracy)
```

✓ 0.7s

```
the Naive bayes accuracy before feature extraction is : 0.772972972972973
```

After feature extraction:

```
#Naive bayes after feature extraction
N_model.fit(pca_train,Y_train)
model_accuracy=N_model.score(pca_test,Y_test)
print("the Naive bayes accuracy after feature extraction is :",model_accuracy)
```

✓ 0.1s

```
the Naive bayes accuracy after feature extraction is : 0.6216216216216216
```

## C) KNN:

Before feature extraction:

We used **n_neighbors = 5**: because it was a stable number for our accuracy.

```
#KNN before feature extraction
K_model= KNeighborsClassifier(n_neighbors=5)
K_model.fit(X_train, Y_train)
model_accuracy=K_model.score(X_test,Y_test)
print("the KNN accuracy before feature extraction is :",model_accuracy)
```

✓ 0.1s

```
the KNN accuracy before feature extraction is : 0.7513513513513513
```

After feature extraction:

```
#KNN after feature extraction
K_model.fit(pca_train, Y_train)
model_accuracy=K_model.score(pca_test,Y_test)
print("the KNN accuracy after feature extraction is :",model_accuracy)
```

✓ 0.1s

```
the KNN accuracy after feature extraction is : 0.7135135135135136
```