Loan Eligibility Prediction

20/5 -- 9:40-10:00

In the making of our Ai model, before data preprocessing we started with collecting some information on our dataset, like the number of rows and columns, the datatype of each column, some statistical information on our dataset and then displayed the values of the categorical data in the dataset, then we moved along with the data preprocessing step and divided it into the following phases**: -**

1. **Data cleansing:** to ensure that the data is consistent, accurate and free of errors, we followed the following steps**: -**
    - **Removing duplicates and irrelevant data from the dataset: -** Where we dropped the Loan_ID column as it has no duplicates and considered irrelevant.

```
[ ]  sum( df.duplicated(subset = 'Loan_ID') ) == 0

     True
```

```
     df.drop("Loan_ID", axis=1 , inplace=True)
```

    - **Handling missing data: -** Where we checked for nulls in each column and found out that we have nulls in 7 columns in our dataset (Gender, Self_Employed, Married, Dependents, Loan_Amount_Term, Credit_History, Loan_Amount)

```
[ ]  # check the missing values
     df.isnull().sum()

     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
```

and because dropping the nulls isn't the best way to deal with nulls, we filled the nulls with the mode for the categorical data (Gender, Self_Employed, Married, Dependents, Loan_Amount_Term, Credit_History) and with the mean for the continuous data (Loan_Amount).

```
missing =['Gender','Married','Dependents','Self_Employed','Loan_Amount_Term','Credit_History']
for i in missing[:7]:
  # fill the categorical data with the mode of the column
    df[i].fillna(df[i].mode()[0],inplace=True)
```

```
#fill the continuous data with the mean to avoid outliers
df['LoanAmount'].fillna(df['LoanAmount'].mean(), inplace=True)
```

- **Handling outliers: -**

With the help of the statistical analysis rules, we were able to identify the outliers in the dataset, visualize them using a box plot and then drop most of them for a better Ai model.

```
# check outliers
def print_ol():
 sns.boxplot(data=df , x = 'ApplicantIncome' , hue = 'Loan_Status')
 plt.show()
 sns.boxplot(data=df , x = 'CoapplicantIncome' , hue = 'Loan_Status')
 plt.show()
 sns.boxplot(data=df , x = 'LoanAmount' , hue = 'Loan_Status')
 plt.show()
print_ol()
```

```
columns = ['ApplicantIncome','CoapplicantIncome','LoanAmount']
for col in columns:
    # calculate interquartile range
    q25, q75 = np.percentile(df[col], 25), np.percentile(df[col], 75)
    iqr = q75 - q25
    lower= q25-(iqr * 1.5)
    upper= (iqr*1.5)+q75
    # identify outliers
    outliers=( ( df[col] < lower) | (df[col] > upper) )
    outlier_index= df[outliers].index
    df.drop(outlier_index,inplace=True)

print_ol()
```

2. **Data Encoding:** converting the categorical data into numbers using either LabelEncoder or OneHotEncoder, but in our model we applied the label encoding method and the replace method to each categorical variable in the dataset as shown in the following piece of code: -

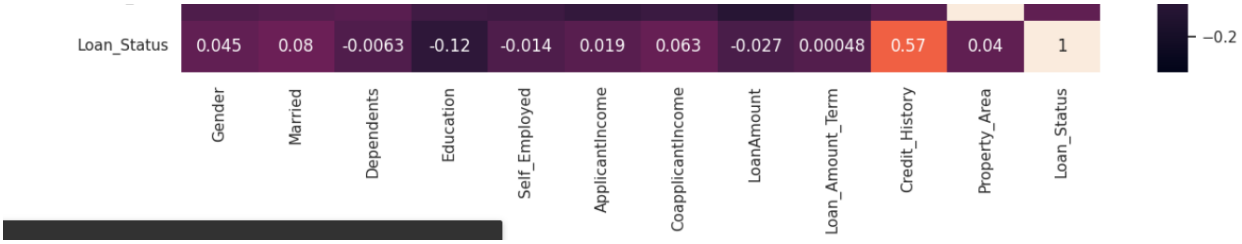
```
cols_to_encode = ['Education', 'Married','Gender','Self_Employed','Loan_Status','Loan_Amount_Term','Property_Area']

label_encoder = LabelEncoder()
for col in cols_to_encode:
    df[col] = label_encoder.fit_transform(df[col])
```

```
[ ] df['Dependents'] = df['Dependents'].replace(['3+'], [int('3')])
    df['Dependents'] = df['Dependents'].replace(['0'], [int('0')])
    df['Dependents'] = df['Dependents'].replace(['1'], [int('1')])
    df['Dependents'] = df['Dependents'].replace(['2'], [int('2')])
```

3. **Feature Selection and Extraction:** to reduce the input variable to your model by using only relevant data and getting rid of the data noise.
   In our model we used two techniques: -

   - **The correlation:** which describes the relationship between each 2 columns in our dataset in a range of [-1, 1].
     The independent column/variable with the lowest correlation with our target/dependent variable (Loan_Status), the less important the selected independent variable is to the model and vice versa.



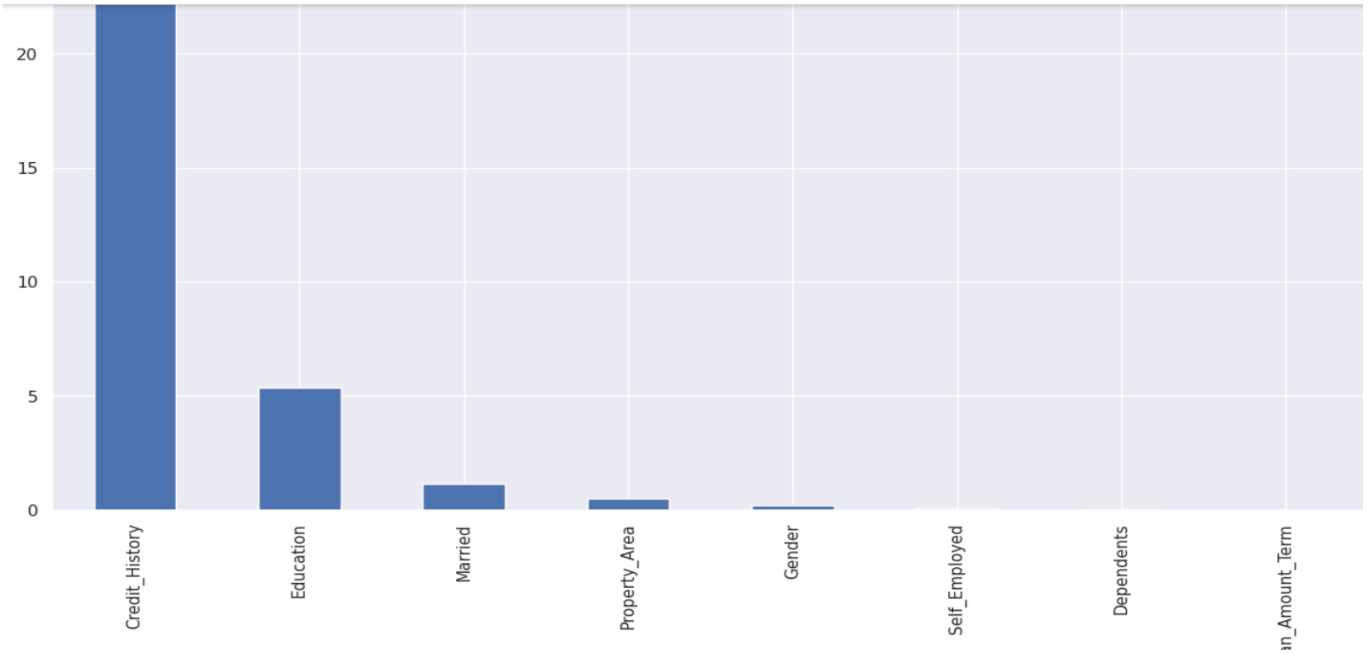
The calculated correlation shown above has shown us that Credit_History has the most effect on the dataset, while Dependents, Self_Employed and Loan_Amount_Term have the weakest 3 relationships with our target (Loan_Status).

- **Chi-squared:** measures the independence between categorical data only based on the null hypothesis test.

```
#using chi-squared
dfs=df[['Gender', 'Married', 'Dependents','Education', 'Self_Employed', 'Loan_Amount_Term',  'Credit_History',  'Property_Area', 'Loan_Status']]
x_temp=dfs.drop("Loan_Status",axis=1)
y_temp=dfs['Loan_Status']
chi_scores=chi2(x_temp,y_temp)
chi_scores
```

```
(array([1.99795896e-01, 1.16318823e+00, 2.85397523e-02, 5.36803727e+00,
        9.22679196e-02, 2.02821913e-05, 2.52103328e+01, 5.09129843e-01]),
 array([6.54885643e-01, 2.80805309e-01, 8.65846097e-01, 2.05090427e-02,
        7.61313525e-01, 9.96406679e-01, 5.14060438e-07, 4.75515783e-01]))
```

```
# the higher the chi value, the more important the feature
chi_values=pd.Series(chi_scores[0],index=x_temp.columns)
chi_values.sort_values(ascending=False,inplace=True)
chi_values.plot.bar()
```



From the graph plotted based on the chi-squared distribution, we also deduced that Self_Employed, Dependents, and Loan_Amount_Term are the most irrelevant data in the dataset with respect to the target (Loan_Status), hence we dropped them.

```
df.drop("Loan_Amount_Term",axis=1,inplace=True)
df.drop("Dependents",axis=1,inplace=True)
df.drop("Self_Employed",axis=1,inplace=True)
```

4. **Data Splitting:** this step is where we split the dataset into testing and training sets for the model to test and train on, the common ratios are 30%:70% and 20%:80%.

in this model we used testing set size= 0.3 (30%) and training set size= 0.7 (70%)

```
[ ]  X = df.iloc[:, 0:8].values
     Y = df.iloc[:, 8].values
```

```
[ ]  X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.3,random_state=0)
```

5. **Feature Scaling:** this step focuses on scaling our continuous data values so that they fit into a smaller range. It has two techniques: -
    - **Normalization:** converting continuous data values from their natural range into a standard range, usually 0 and 1 (or sometimes -1 to +1) and in this data model we used a range from -1 to +1.

```
[ ]  #Normalization
     scaler = MinMaxScaler()
     income_scaled = scaler.fit_transform(income.values.reshape(-1,1))
     lamount_scaled = scaler.fit_transform(lamount.values.reshape(-1,1))
```

We applied the normalization on the data using MinMaxScaler which is implemented using the following mathematical formula: -

$$x' = (x - x_{min})/(x_{max} - x_{min})$$

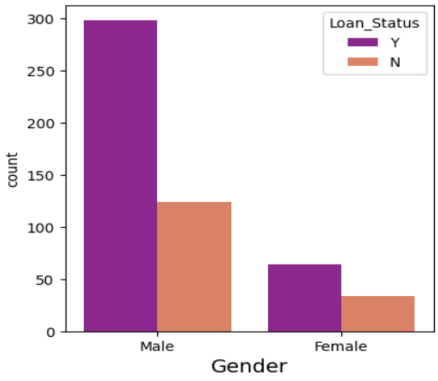
- **Standardization (Z-score Normalization):** this method makes the values of each feature in the data have a mean of zero and a unit-variance, and in the model, we used StandardScaler which is calculated through the following formula:

$$x' = \frac{x - \bar{x}}{\sigma}$$

```
[ ]  #Standardization
     scaler=StandardScaler()
     coincome_scaled=scaler.fit_transform(coincome.values.reshape(-1,1))
```

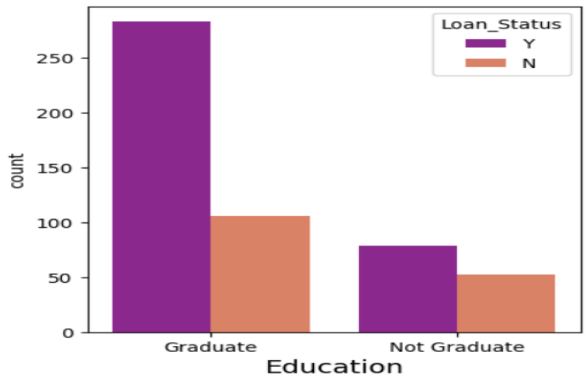
Now we need to visualize the data to determine the relationship between the independent variables with the target (Loan_Status).

The graph below shows that the loan acceptance rate for males is larger than that for the females.



The graph below shows that the loan acceptance rate for married people is larger than that for the not married.

The graph below shows that the loan acceptance rate for graduates is larger than that for the not graduates.



Now that we're done with data preprocessing and visualization, we can move to training the data, and we used 4 models to train on our data; Logistic Regression, Decision Tree, Support Vector Machine (SVM) and Extreme Gradient Boost (XG Boost).

1. **Logistic Regression:** a classification algorithm that predicts the probability that an instance belongs to a given class or not by transforming the linear regression output (continuous value) into a categorical value using the **sigmoid function**.

```
#Logistic Regression model
classifier = LogisticRegression(random_state=0)
classifier.fit(X_train, y_train)
```

The hyperparameter used in training the model is random_state, which sets seed to a random generator.

2. **SVM:** a classification algorithm that classifies the data based on its features, and its objective is finding the best decision boundary/ hyperplane with maximum margin from features.

```
# using SVM model
clf = SVC(kernel='linear', C=1)
clf.fit(X_train, y_train)
```

The hyperparameters used in this model are: -

1) **Kernel**: maps the data into a higher dimension, can be linear, poly, rbf or sigmoid, but in this model we used linear.
2) **C**: it's a penalty for misclassified points, the lower C value is, the better the regularization hence the better the model.

3. **Decision Tree:** a regression and binary classification algorithm that learns from data to approximate a sine curve with a set of if-then-else conditions.

```
# using Decision Tree model
DT = DecisionTreeClassifier()
DT.fit(X_train, y_train)
```

No hyperparameters used here, all hyperparameters are set to their default value, such as max_depth, min_samples_split, max_features, min_samples_leaf.

4. **XG Boost:** an algorithm provides a parallel tree boosting that solves many data science problems in a fast and accurate way.

```
# using XG boost model
model = XGBClassifier(max_depth=3, subsample=1, n_estimators=50, min_child_weight=1, random_state=5)
model.fit(X, Y)
```

Hyperparameters used: -

a. **Max_depth:** maximum tree depth.
b. **Subsample:** subsample ratio of training instance.
c. **N_estimators:** number of trees boosted.
d. **Min_child_weight:** minimum sum of instance weight needed in a child (internal node).
e. **Random_state.**

And now after training the data with the 4 models, we need to test the model to see which one gives better accuracy.

1. **Testing the logistic regression model:**

```
# Using Logistic Regression Model
y_pred = classifier.predict(X_test)
logisticAcc=accuracy_score(y_pred,y_test)*100
print('Classification report: \n',classification_report(y_test, y_pred))
print(f"Accuracy: {round(logisticAcc,2)}%")
print('Mean Squared Error : ', round(mean_squared_error(np.asarray(y_test), y_pred),2))
print('Confusion matrix :\n ', confusion_matrix(y_test,y_pred))
```

```
Classification report:
              precision    recall  f1-score   support

           0       0.90      0.52      0.66        50
           1       0.81      0.97      0.88       106

    accuracy                           0.83       156
   macro avg       0.85      0.75      0.77       156
weighted avg       0.84      0.83      0.81       156

Accuracy: 82.69%
Mean Squared Error :  0.17
Confusion matrix :
  [[ 26  24]
 [  3 103]]
```

2. Testing the SVM model:

```
# Using SVM Model
y_pred = clf.predict(X_test)
svm_acc = accuracy_score(y_test, y_pred)*100
print('Classification report: \n',classification_report(y_test, y_pred))
print(f"Accuracy: {round(svm_acc,2)}%")
print('Mean Squared Error : ', round(mean_squared_error( np.asarray(y_test), y_pred),2))
print('Confusion matrix : \n', confusion_matrix(y_test,y_pred))
```

```
Classification report:
               precision    recall  f1-score   support

           0       0.90      0.52      0.66        50
           1       0.81      0.97      0.88       106

    accuracy                           0.83       156
   macro avg       0.85      0.75      0.77       156
weighted avg       0.84      0.83      0.81       156

Accuracy: 82.69%
Mean Squared Error :  0.17
Confusion matrix :
 [[ 26  24]
 [  3 103]]
```

3. Testing the decision tree model:

```
# Using Decision Tree Model
y_predict = DT.predict(X_test)
decisionAcc = accuracy_score(y_predict,y_test)*100
print('Classification report: \n',classification_report(y_test, y_predict))
print(f"Accuracy: {round(decisionAcc,2)}%")
print('Mean Squared Error : ', round(mean_squared_error(np.asarray(y_test), y_predict),2))
print('Confusion matrix : \n', confusion_matrix(y_test,y_pred))
```

```
Classification report:
               precision    recall  f1-score   support

           0       0.60      0.66      0.63        50
           1       0.83      0.79      0.81       106

    accuracy                           0.75       156
   macro avg       0.72      0.73      0.72       156
weighted avg       0.76      0.75      0.75       156

Accuracy: 75.0%
Mean Squared Error :  0.25
Confusion matrix :
 [[ 26  24]
 [  3 103]]
```

4. Testing the XGB model:

```
# Using XG Boost Model
y_predict = model.predict(X_test)
xgbAcc=accuracy_score(y_test, y_predict)*100
print('Classification report:\n',classification_report(y_test, y_predict))
print(f"Accuracy: {round(xgbAcc,2)}%")
print('Mean Squared Error : ',round( mean_squared_error(np.asarray(y_test), y_predict),2))
print('Confusion matrix :\n', confusion_matrix(y_test,y_pred))
```

```
Classification report:
               precision    recall  f1-score   support

           0       0.94      0.64      0.76        50
           1       0.85      0.98      0.91       106

    accuracy                           0.87       156
   macro avg       0.90      0.81      0.84       156
weighted avg       0.88      0.87      0.86       156

Accuracy: 87.18%
Mean Squared Error :  0.13
Confusion matrix :
 [[ 26  24]
 [  3 103]]
```

Conclusion: -

```
score = [ round(xgbAcc,2), round(logisticAcc,2) , round(svm_acc,2) ,round(decisionAcc,2)]
Models = pd.DataFrame({'Algorithm': ["XG boost","Logistic Regression","SVM","Decision Tree"],'Accuracy': score})
Models.sort_values(by='Accuracy', ascending=False)
```

	Algorithm	Accuracy
0	XG boost	87.18
1	Logistic Regression	82.69
2	SVM	82.69
3	Decision Tree	73.08

As shown in the figure above, XG Boost model gives the best accuracy, and it's considered a good model as it's between 70-90% accuracy.