Loan Eligibility Prediction

Libraries used

1- Pandas
2- Numpy
3- Seaborn
4- Sklearn
5- Matplotlib
6- Scipy

1-Data preprocessing

- **Major problems of data:**

1- NULL values
2- outliers
3- costing features

As we go through the preprocessing code we will be handling those main issues as well as checking for duplicated rows, replacing categorical values to numerical values (encoding) and normalization of data.

A) Converting categorical values to numerical

Many machine learning algorithms are designed to work with numerical data. By converting categorical variables into numerical representations, we can ensure compatibility with a wide range of algorithms.

As well as using mathematical operations that will require all values to be numeric, such as calculating outliers, plotting correlation and normalization.

using data.head() we can notice that some columns have categorical data such as 'married' and 'self-employed', we will take each distinct value and replace it with a numeric values starting from 0 to number of distinct values-1, using data['column name'].replace("old_value",0) .

We will apply the same changes to columns: married, gender, self employed, education and loan status.

-Using encoding:

pd.get_dummies(data, columns=['Property_Area', 'Dependents'])

This function call is creating dummy variables for the columns specified in the `columns` parameter, which are `'Property_Area'` and `'Dependents'`. Dummy variables are binary variables that represent different categories or levels of a categorical variable.

By creating these dummy variables, you can represent categorical data in a format that machine learning algorithms can more easily process, as they typically require numerical input.

B) Make sure there are no duplicated records

Using function data.duplicated() to check for duplicated records, and view their sum using sum(), we notice that there are no duplications.

C) Checking for NULLS

```
print (data.isnull().sum())
```

Using this code we find the presence of NULL values in some rows which are:
Gender, Married, Self Employed, LoanAmount, Loan_Amount_Term and Credit_History.

NULLs percentage is very small compared to column size, it will not be efficient to drop columns, so we will be replacing NULL values with mean or mode.

Using data.describe() and data[‘column name’].mode

We can view some statistics such as mean, the mode value for this column respectively.

To double check we can use:

```
print (data.isnull().sum())
```

Shows that the sum of nulls in all columns is zero.

```
data.info()
```

This function view each column with its datatype, to make sure all data types is int or float and convert object type to a numerical data type if necessary using

data[‘column name’]=data[‘column name’].astype(int)

D) Costing features

Loan_id will not benefit the model in predicting, also it will cause a high computational cost, so the best solution is to drop it using:

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

E) Feature scaling normalization

In statistics and machine learning, min-max normalization of data is a process of converting the original range of data to the range between 0 and 1.

The resulting normalized values represent the original data on 0-1 scale.

This will allow us to compare multiple features together and get more relevant information since now all the data will be on the same scale.

In min-max normalization, for every feature, its minimum value gets transformed into 0 and its maximum value gets transformed into 1.

All values in-between get scaled to be within 0-1 range based on the original value relative to minimum and maximum values of the feature.

```
#to take an object from class MinMaxScaler
scaler = MinMaxScaler()
#this code normalizes values in dataframe , but it returns an array
normalized_data = scaler.fit_transform(data)
#converting array to dataframe
df = pd.DataFrame(normalized_data, columns=data.columns)
#now we have df as a normalized data frame with all values in range [0,1]
```

F) Handling outliers

Outliers are data points that significantly deviate from the general pattern or distribution of the rest of the data.Outliers can have a significant influence on statistical measures and analysis. leading to biased estimations.

To calculate an outlier: it's any value > IQR*1.5+Q3 or <IQR*1.5-Q1

```
sns.boxplot(data=data, palette='rainbow', orient='h')
```

We noticed the presence of outliers in "ApplicantIncome" and "CoapplicantIncome" columns. we will be replacing outliers with non-outlier maximum or minimum values as needed.

```
#calculating maximum non-outlier

z_scores = stats.z score(data['ApplicantIncome'])

outliers = (z_scores > 3) | (z_scores < -3)

max_non_outlier = np.max(data['ApplicantIncome'][~outliers])
```

```
print(max_non_outlier)
```

2-Data plotting & visualization

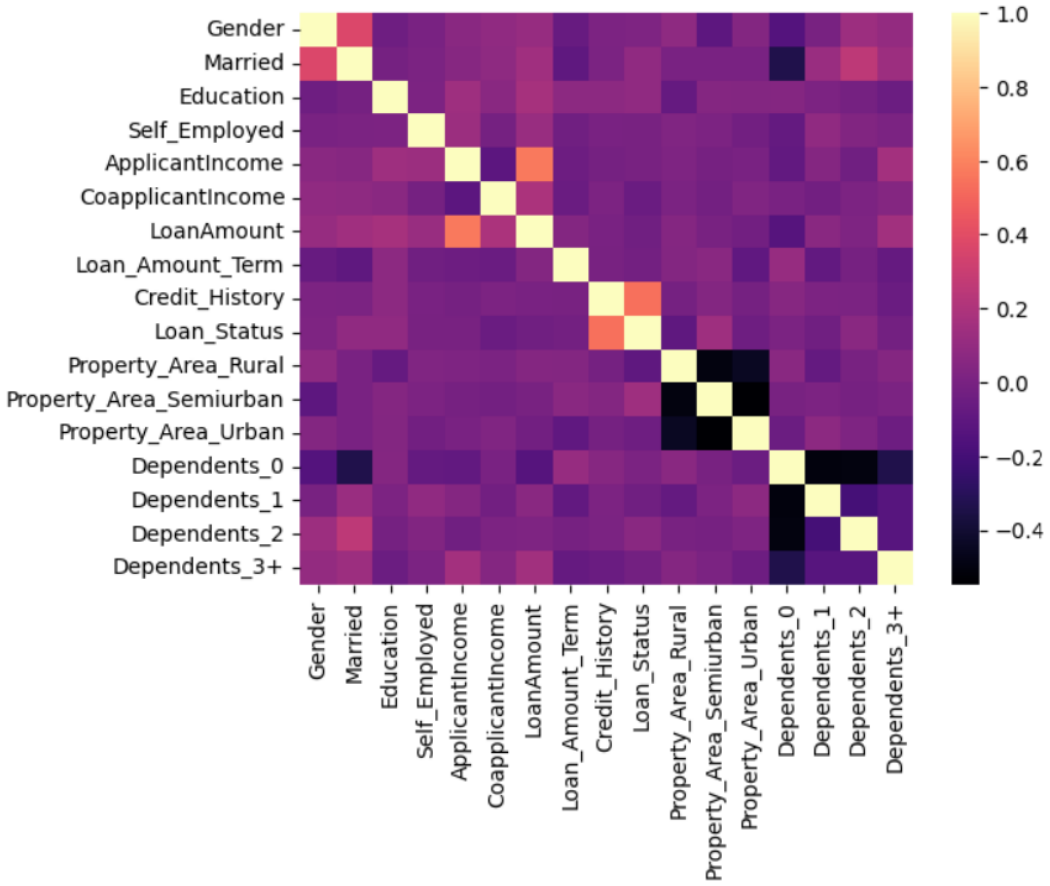
A) Visualize correlation:

To visualize correlation between variables, we can use the `seaborn` library, which is built on top of `matplotlib` and provides additional functionality for data visualization.

We calculate the correlation matrix using the `corr()` method of the DataFrame. Finally, we create a heatmap of the correlation matrix using the `heatmap()` function from `seaborn`. The `annot=True` parameter adds the correlation coefficient values to each cell of the heatmap, and the `cmap` parameter specifies the color map to be used.

To create a correlation table we can use the `corr()` method of a Pandas DataFrame to calculate the correlation coefficients between all pairs of columns. The resulting correlation table is stored in the `corr_table` variable. Finally, we display the correlation table using the `print()` function.

The output will be a table that shows the correlation coefficients between all pairs of columns. The diagonal of the table will contain 1's since the correlation coefficient between a column and itself is always 1. The correlation coefficients range from -1 to 1, where -1 indicates a perfect negative correlation, 0 indicates no correlation, and 1 indicates a perfect positive correlation.

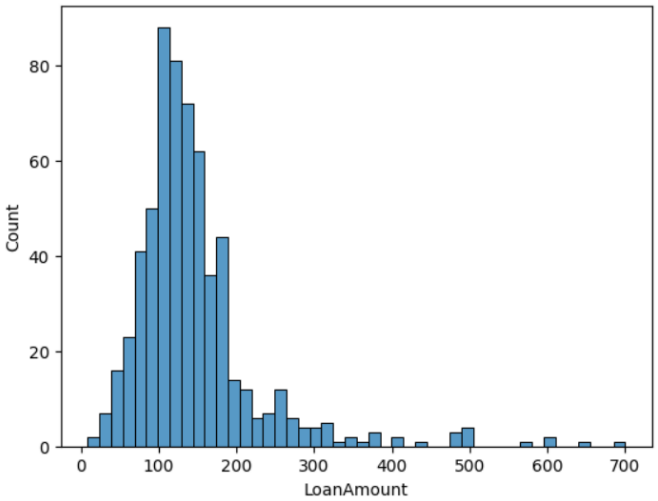


B) Another visualizations on data set:

- **sns.histplot(data['LoanAmount'])**

-Used to create a histogram plot of the 'LoanAmount' variable using the Seaborn library.

-Creates a histogram plot of the 'LoanAmount' variable using the data from the 'LoanAmount' column in the 'data' DataFrame. The histogram will display the distribution of loan amounts, with the x-axis representing the range of loan amounts and the y-axis representing the frequency or count of occurrences within each bin.



- **sns.barplot(x='Gender', y='LoanAmount', data=data)**

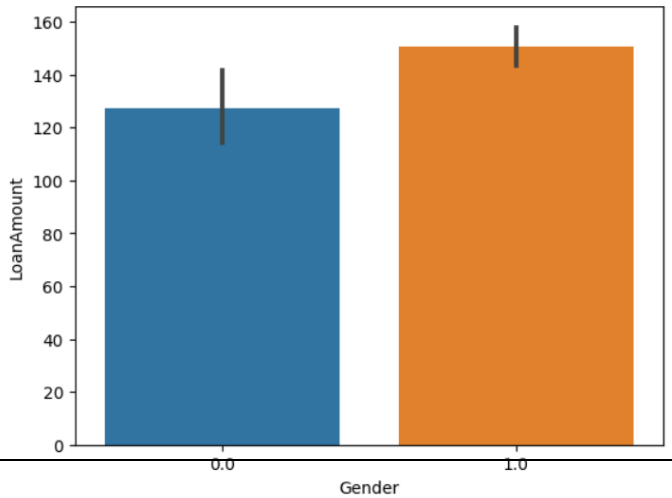
-Used to create a bar plot using the Seaborn library.

-Create a bar plot where the x-axis represents the 'Gender' variable and the y-axis represents the 'LoanAmount' variable. Each bar in the plot represents the mean or aggregated value of the 'LoanAmount' for each category of 'Gender'.

-The height of each bar indicates the value of the 'LoanAmount' variable, and you can compare the values across different categories of 'Gender' visually. The bar plot is useful for comparing and visualizing the distribution or average values of a numeric variable across different categories.

0(blue): Female

1(orange): Male



3-Data modeling

A) Logistic Regression:

Logistic regression is a statistical modeling technique used to predict the probability of an event occurring based on given input variables. It is a type of supervised learning algorithm that is particularly useful for binary classification problems, where the outcome variable can take one of two possible values.

Steps:

1. Importing Required Libraries:

The code begins by importing the necessary libraries. The "sklearn.metrics" module is imported to access the accuracy_score function, and the "sklearn.linear_model" module is imported to use the LogisticRegression class.

2. Initializing the Model:

The code creates an instance of the LogisticRegression class by calling LogisticRegression() and assigns it to the variable "model". This step initializes the logistic regression model with default settings.

3. Training the Model:

To train the logistic regression model, the code calls the fit() method on the model instance, passing the training data (x_train and y_train) as arguments. This step fits the model to the provided training data.

4. Making Predictions:

The code uses the trained model to make predictions on the test data by calling the predict() method on the model instance and passing the test data (x_test) as an argument. The predicted values are assigned to the variable "y_predict".

5. Evaluating Accuracy:

To evaluate the accuracy of the model's predictions, the code calls the accuracy_score() function from the sklearn.metrics module. It takes two arguments: the true labels (y_test) and the predicted labels (y_predict). The accuracy_score() function computes the accuracy of the predicted labels compared to the true labels.

6. Calculate the confusion matrix:

The confusion_matrix function is applied to the true target values y_test and the predicted values y_predict to compute the confusion matrix, which represents the performance of the classification model.

7. Generate a classification report:

The classification_report function is used to generate a detailed report containing various metrics such as precision, recall, F1-score, and support for each class.

8. Calculate the mean squared error:

The mean_squared_error function is used to calculate the mean squared error between the true target values y_test and the predicted values y_predict.

B) SVM:

Support Vector Machine is a machine learning algorithm used for solving classification and regression problems.

Steps:

1. **Importing required libraries:**
   Import the SVM module from SKlearn library.

2. **Splitting the dataset into training and testing sets:**
   The **train_test_split()** function randomly splits the X and y arrays into two sets - **X_train** and **y_train** are the training sets, and **X_test** and **y_test** are the testing sets.
   The **test_size** parameter specifies the proportion of the dataset to include in the test set, and **random_state** is used to ensure that the same random split is generated each time the code is run.

3. **Initializing the model:**
   Create an instance of the SVM classifier. The **SVC** constructor is used to create a support vector machine with a linear kernel.

4. **Training the model:**
   The **fit()** function takes the training data **X_train** and **y_train** and fits the SVM classifier to the data.

5. **Making predictions:**
   The **predict()** function takes the test data **X_test** as input and returns the predict class labels.

C) Decision Tree:

The decision tree classifier is a simple and intuitive model that can handle both categorical and numerical data. It is also easy to interpret and visualize, making it a popular choice for exploratory data analysis. The decision tree algorithm can handle missing values and outliers and can also handle nonlinear relationships between the input variables and the output variable.

Steps:

1. **Importing required libraries:**

Import DecisionTreeClassifier for creating a decision tree classifier.

2. **Splitting the dataset into training and testing sets:**

The function **splitdataset(loan_data)** takes a dataset **loan_data** as input and splits it into training and testing sets using the **train_test_split()** function. The function returns the original dataset X and its corresponding class labels y, as well as the training and testing sets **X_train**, **X_test**, **y_train**, and **y_test**.

3. **Training using entropy:**

The function **train_using_entropy()** takes the training and testing sets **X_train**, **X_test**, and **y_train** as input, creates a decision tree classifier using the **DecisionTreeClassifier()** function with entropy as the splitting criterion, and trains the classifier using the training data. The criterion parameter is set to "entropy" to use the information gain (entropy) as a measure of the quality of a split.
The **random_state** parameter is set to 100 to ensure that the results are reproducible.
The **max_depth** parameter is set to 1 to limit the depth of the tree to 1. This helps prevent overfitting and improves the generalization performance of the classifier. The function returns the predicted class labels **y_pred**.

Adjusting random state

Random state: hyper parameter used to control any randomness, used to obtain the same train/test splits or random variations in the model. The value is a random number that helps reproduce the same result.

Adjusting random state:

Using loop of range from 1 to 100

We checked accuracy with respect to changing the random state value and we choose the best accuracy using the if statement.

Conclusion

We found that logistic regression model and SVM model gave the best accuracy which equals 88.6%

With random state = 8 .