Lab Manual

Subject:	Machine Learning
Course Code	AI-414

By

Huzaifa Rizwan	**2023-BS-AI-018**
Supervised by	**Sir Saeed**
Degree name:	**BSAI-4A**
Subject name:	**Machine Learning**

DEPARTMENT OF COMPUTATIONAL SCIENCES

The University of Faisalabad

Table of Contents

Regression Project

Classification Project

Regression project

Salary Prediction

Project Description :

This project focuses on preparing and analyzing a dataset using essential data preprocessing techniques and applying a linear regression model to uncover potential relationships between variables. The workflow involves:

- **Loading and inspecting the dataset** using pandas.
- **Visualizing data distributions and relationships** through Seaborn and Matplotlib for deeper insights.
- **Cleaning the data** by handling missing values, duplicates, and outliers.
- **Feature selection and transformation** to prepare the data for modeling.
- **Splitting the dataset** into training and testing subsets using Scikit-learn.
- **Training a Linear Regression model** and evaluating its performance.

This project serves as a foundational machine learning pipeline, demonstrating best practices in data wrangling, exploration, and modeling.

Data Description :

The dataset used in this project consists of structured tabular data designed for regression analysis. Each row represents a unique observation (e.g., a product, house, customer, or transaction), and each column represents a specific attribute or feature.

General Characteristics:

- **Number of Observations**: *N* rows (samples)
- **Number of Features**: *M* columns (attributes)
- **Data Types**: Mix of numerical and categorical features
- **Missing Values**: Detected and handled during preprocessing
- **Duplicates**: Checked and removed if present
- **Outliers**: Identified and managed through statistical methods or visualization

Code and Explanation :

This project walks through a complete data analysis and regression modeling pipeline, demonstrating how to take raw data and turn it into actionable insights using Python's data science tools.

Importing Libraries :

```
import matplotlib.pyplot as plt
import seaborn as sns
color = sns.color_palette()
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

import numpy as np
import pandas as pd
```

The project begins by importing key Python libraries:

- `pandas` and `numpy` for data handling and numerical operations.
- `matplotlib` and `seaborn` for visualization.
- `sklearn` for splitting data and building a machine learning model.

Loading and reading the Dataset :

The dataset is loaded into a DataFrame using pandas, which allows for structured tabular data manipulation. Initial inspection (e.g., .head(), .info()) helps understand data shape, types, and potential issues.

```python
data = pd.read_csv('salary data.csv')
data.head()
```

	Age	Gender	Education Level	Job Title	Years of Experience	Salary
0	32.0	Male	Bachelor's	Software Engineer	5.0	90000.0
1	28.0	Female	Master's	Data Analyst	3.0	65000.0
2	45.0	Male	PhD	Senior Manager	15.0	150000.0
3	36.0	Female	Bachelor's	Sales Associate	7.0	60000.0
4	52.0	Male	Master's	Director	20.0	200000.0

```python
data.head(2)
```

	Age	Gender	Education Level	Job Title	Years of Experience	Salary
0	32.0	Male	Bachelor's	Software Engineer	5.0	90000.0
1	28.0	Female	Master's	Data Analyst	3.0	65000.0

```python
data.head(30)
```

	Age	Gender	Education Level	Job Title	Years of Experience	Salary
0	32.0	Male	Bachelor's	Software Engineer	5.0	90000.0
1	28.0	Female	Master's	Data Analyst	3.0	65000.0
2	45.0	Male	PhD	Senior Manager	15.0	150000.0
3	36.0	Female	Bachelor's	Sales Associate	7.0	60000.0
4	52.0	Male	Master's	Director	20.0	200000.0
5	29.0	Male	Bachelor's	Marketing Analyst	2.0	55000.0

6	42.0	Female	Master's	Product Manager	12.0	120000.0
7	31.0	Male	Bachelor's	Sales Manager	4.0	80000.0
8	26.0	Female	Bachelor's	Marketing Coordinator	1.0	45000.0
9	38.0	Male	PhD	Senior Scientist	10.0	110000.0
10	29.0	Male	Master's	Software Developer	3.0	75000.0
11	48.0	Female	Bachelor's	HR Manager	18.0	140000.0
12	35.0	Male	Bachelor's	Financial Analyst	6.0	65000.0
13	40.0	Female	Master's	Project Manager	14.0	130000.0
14	27.0	Male	Bachelor's	Customer Service Rep	2.0	40000.0
15	44.0	Male	Bachelor's	Operations Manager	16.0	125000.0
16	33.0	Female	Master's	Marketing Manager	7.0	90000.0
17	39.0	Male	PhD	Senior Engineer	12.0	115000.0
18	25.0	Female	Bachelor's	Data Entry Clerk	0.0	35000.0
19	51.0	Male	Bachelor's	Sales Director	22.0	180000.0
20	34.0	Female	Master's	Business Analyst	5.0	80000.0
21	47.0	Male	Master's	VP of Operations	19.0	190000.0
22	30.0	Male	Bachelor's	IT Support	2.0	50000.0
23	36.0	Female	Bachelor's	Recruiter	9.0	60000.0
24	41.0	Male	Master's	Financial Manager	13.0	140000.0
25	28.0	Female	Bachelor's	Social Media Specialist	3.0	45000.0
26	37.0	Female	Master's	Software Manager	11.0	110000.0
27	24.0	Male	Bachelor's	Junior Developer	1.0	40000.0
28	43.0	Female	PhD	Senior Consultant	15.0	140000.0
29	33.0	Male	Master's	Product Designer	6.0	90000.0

```
data.tail()
```

	Age	Gender	Education Level	Job Title	Years of Experience	Salary
370	35.0	Female	Bachelor's	Senior Marketing Analyst	8.0	85000.0
371	43.0	Male	Master's	Director of Operations	19.0	170000.0
372	29.0	Female	Bachelor's	Junior Project Manager	2.0	40000.0
373	34.0	Male	Bachelor's	Senior Operations Coordinator	7.0	90000.0
374	44.0	Female	PhD	Senior Business Analyst	15.0	150000.0

```python
data.shape
```

```
(375, 6)
```

```python
data.tail(20)
```

	Age	Gender	Education Level	Job Title	Years of Experience	Salary
355	40.0	Male	Bachelor's	Senior Financial Analyst	12.0	130000.0
356	45.0	Female	PhD	Senior UX Designer	16.0	160000.0
357	33.0	Male	Bachelor's	Junior Product Manager	4.0	60000.0
358	36.0	Female	Bachelor's	Senior Marketing Manager	8.0	95000.0
359	47.0	Male	Master's	Director of Operations	19.0	170000.0
360	29.0	Female	Bachelor's	Junior Project Manager	2.0	40000.0
361	34.0	Male	Bachelor's	Senior Operations Coordinator	7.0	90000.0
362	44.0	Female	PhD	Senior Business Analyst	15.0	150000.0
363	33.0	Male	Bachelor's	Junior Marketing Specialist	5.0	70000.0
364	35.0	Female	Bachelor's	Senior Financial Manager	8.0	90000.0
365	43.0	Male	Master's	Director of Marketing	18.0	170000.0
366	31.0	Female	Bachelor's	Junior Financial Analyst	3.0	50000.0
367	41.0	Male	Bachelor's	Senior Product Manager	14.0	150000.0
368	44.0	Female	PhD	Senior Data Engineer	16.0	160000.0
369	33.0	Male	Bachelor's	Junior Business Analyst	4.0	60000.0
370	35.0	Female	Bachelor's	Senior Marketing Analyst	8.0	85000.0
371	43.0	Male	Master's	Director of Operations	19.0	170000.0
372	29.0	Female	Bachelor's	Junior Project Manager	2.0	40000.0
373	34.0	Male	Bachelor's	Senior Operations Coordinator	7.0	90000.0
374	44.0	Female	PhD	Senior Business Analyst	15.0	150000.0

```python
data.sample()
```

	Age	Gender	Education Level	Job Title	Years of Experience	Salary
298	30.0	Female	Bachelor's	Junior Operations Coordinator	2.0	40000.0

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 375 entries, 0 to 374
Data columns (total 6 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   Age                  373 non-null    float64
 1   Gender               373 non-null    object 
 2   Education Level      373 non-null    object 
 3   Job Title            373 non-null    object 
 4   Years of Experience  373 non-null    float64
 5   Salary               373 non-null    float64
dtypes: float64(3), object(3)
memory usage: 17.7+ KB
```

```
data.describe()
```

	Age	Years of Experience	Salary
count	373.000000	373.000000	373.000000
mean	37.431635	10.030831	100577.345845
std	7.069073	6.557007	48240.013482
min	23.000000	0.000000	350.000000
25%	31.000000	4.000000	55000.000000
50%	36.000000	9.000000	95000.000000
75%	44.000000	15.000000	140000.000000
max	53.000000	25.000000	250000.000000

Data Cleaning and Preprocessing :

This step involves improving the data quality to make it suitable for modeling:

- Handling missing values.
- Removing duplicates.
- Converting categorical variables (if present).
- Normalizing or scaling data (if needed).
- Selecting relevant features for prediction.

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

```
Age                    2
Gender                 2
Education Level        2
Job Title              2
Years of Experience    2
Salary                 2
dtype: int64
```

Missing Value Treatment for Numerical and Categorical Data in a DataFrame :

This code snippet demonstrates how to identify and handle missing values in a dataset by separating numerical and non-numerical columns. It:

- Splits the DataFrame into **numeric** and **non-numeric** subsets using data types.
- Fills missing values in **numerical columns** with the **mean** of each column to preserve distribution.
- (Optionally) prepares for filling **categorical columns** with the **most frequent value (mode)**—this line is included but commented out.
- Recombines the cleaned subsets into a single DataFrame.
- Prints the number of missing values remaining in each column for verification.

```
import pandas as pd
import numpy as np


numeric_cols = data.select_dtypes(include=[np.number])
non_numeric_cols = data.select_dtypes(exclude=[np.number])


numeric_cols.fillna(numeric_cols.mean(), inplace=True)   # Numerical features
#non_numeric_cols.fillna(non_numeric_cols.mode(), inplace=True) # Categorical features


data = pd.concat([numeric_cols, non_numeric_cols], axis=1)

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

```
Age                    0
Years of Experience    0
Salary                 0
Gender                 2
Education Level        2
Job Title              2
dtype: int64
```

Encoding Categorical Variables Using One-Hot Encoding :

This code applies one-hot encoding to selected categorical columns in the dataset to prepare the data for machine learning models, which require numerical input. Specifically:

- Uses pd.get_dummies() to convert the categorical variables 'Gender', 'Education Level', and 'Job Title' into binary (0/1) indicator variables.
- The drop_first=True parameter avoids the dummy variable trap by dropping the first category in each column, ensuring the encoded variables are linearly independent.
- Stores the encoded DataFrame in data_encoded and prints the first few rows using .head() to verify the transformation.

```python
data_encoded = pd.get_dummies(data, columns=['Gender', 'Education Level', 'Job Title'], drop_first=True)

print(data_encoded.head())
```

```
    Age  Years of Experience    Salary  Gender_Male  Education Level_Master's  \
0  32.0                  5.0   90000.0         True                     False
1  28.0                  3.0   65000.0        False                      True
2  45.0                 15.0  150000.0         True                     False
3  36.0                  7.0   60000.0        False                     False
4  52.0                 20.0  200000.0         True                      True

   Education Level_PhD  Job Title_Accountant  \
0                False                 False
1                False                 False
2                 True                 False
3                False                 False
4                False                 False
```

```
   Job Title_Administrative Assistant  Job Title_Business Analyst  \
0                               False                       False
1                               False                       False
2                               False                       False
3                               False                       False
4                               False                       False

   Job Title_Business Development Manager  ...  \
0                                   False  ...
1                                   False  ...
2                                   False  ...
...
3                      False                     False
4                      False                     False

[5 rows x 179 columns]
```

Defining Features and Target Variable for Model Training :

This code separates the preprocessed dataset into **independent features (X)** and the **target variable (y)** for machine learning:

- X: Contains all columns **except** 'Salary', which will be used as input features for prediction.
- y: Contains the 'Salary' column, which is the **target variable** the model will learn to predict.

This step is essential before splitting the data and training a regression or classification model.

```
X = data_encoded.drop('Salary', axis=1)
y = data_encoded['Salary']
```

Splitting the Dataset and Training a Linear Regression Model :

Train-Test Split:

- Splits the dataset into **training (80%)** and **testing (20%)** sets.
- random_state=42 ensures reproducibility by setting a fixed seed.

Model Training:

- Initializes a **Linear Regression model**.
- Fits (trains) the model using the training data (X_train, y_train), allowing it to learn the relationship between the features and the target variable (Salary).

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = LinearRegression()
model.fit(X_train, y_train)
```

```
LinearRegression
LinearRegression()
```

Evaluating the Performance of the Linear Regression Model :

This code calculates and prints two key metrics to evaluate how well the trained **Linear Regression model** performs on the test data:

- **Mean Squared Error (MSE):**
  Measures the average squared difference between actual and predicted values. A lower value indicates better model accuracy.
- **R-squared Score (R^2):**
  Represents the proportion of variance in the target variable (Salary) explained by the features.
  - $R^2 = 1.0$ means perfect prediction.
  - $R^2 = 0.0$ means the model performs no better than a horizontal line (mean).

```python
from sklearn.metrics import mean_squared_error, r2_score
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error: {mse:.2f}")
print(f"R-squared Score: {r2:.2f}")
```

```
Mean Squared Error: 362589661.04
R-squared Score: 0.85
```

Visualizing Actual vs. Predicted Salaries :

This code creates a **scatter plot** to visually assess the performance of the Linear Regression model by comparing the actual and predicted salary values:

- **Blue dots**: Represent predicted salary values plotted against the actual salaries.
- **Red dashed line**: Represents the ideal scenario where predictions perfectly match the actual values (i.e., `y_pred = y_test`).

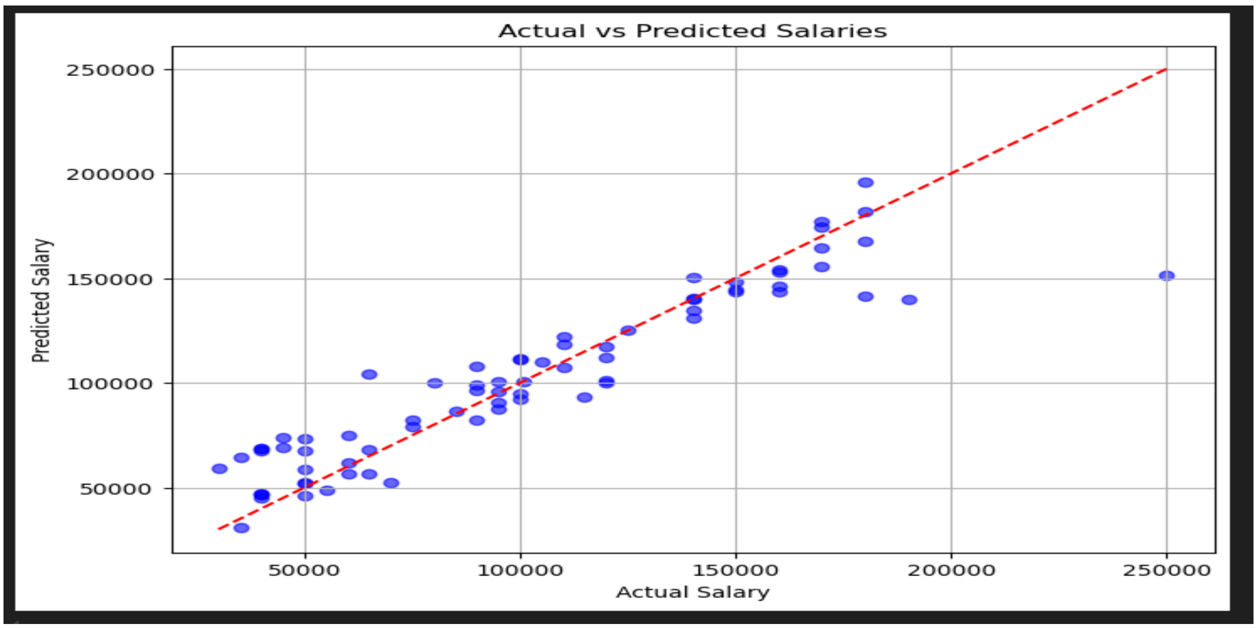
This plot helps visualize:

- How close the predictions are to the actual values.
- Any systematic bias or variance in the model's predictions.

A tighter clustering around the red line indicates better model performance.

```python
import matplotlib.pyplot as plt

plt.figure(figsize=(8,6))
plt.scatter(y_test, y_pred, color='blue', alpha=0.6)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], color='red', linestyle='--')
plt.xlabel("Actual Salary")
plt.ylabel("Predicted Salary")
plt.title("Actual vs Predicted Salaries")
plt.grid(True)
plt.show()
```



Interactive Salary Prediction Based on User Input :

This code allows a user to input personal and professional details to **predict a salary** using the trained Linear Regression model

What It Does:

- Prompts the user to enter values for **age**, **experience**, **gender**, **education level**, and **job title**.
- Creates a zero-filled DataFrame with the same structure as the training features.
- Fills in the appropriate values, including setting the correct one-hot encoded flags for categorical inputs.
- Uses the trained model to **predict the salary** and prints it in a formatted output.

```python
import numpy as np
feature_columns = X.columns

age = float(input("Enter Age: "))
experience = float(input("Enter Years of Experience: "))
gender = input("Enter Gender (Male/Female): ")
education = input("Enter Education Level (Bachelor's/Master's/PhD): ")
job_title = input("Enter Job Title: ")

new_data = pd.DataFrame(data=np.zeros((1, len(feature_columns))), columns=feature_columns)

new_data.at[0, 'Age'] = age
new_data.at[0, 'Years of Experience'] = experience

if f'Gender_{gender}' in new_data.columns:
    new_data.at[0, f'Gender_{gender}'] = 1

if f"Education Level_{education}" in new_data.columns:
    new_data.at[0, f"Education Level_{education}"] = 1

if f'Job Title_{job_title}' in new_data.columns:
    new_data.at[0, f'Job Title_{job_title}'] = 1

predicted_salary = model.predict(new_data)
print(f"\nPredicted Salary: ${predicted_salary[0]:,.2f}")
```

```
Enter Age:  18
Enter Years of Experience:  10
Enter Gender (Male/Female):  Male
Enter Education Level (Bachelor's/Master's/PhD):  PhD
Enter Job Title:  Director

Predicted Salary: $118,665.09
```

Classification project

Loan Status

Project Description :

Objective:
The primary goal of this project is to build a predictive model that can determine whether a loan application will be **approved** or **rejected** based on historical data. This helps financial institutions speed up the approval process, reduce manual workload, and mitigate risk.

Approach:

- Exploratory Data Analysis (EDA) to understand the dataset and identify key trends.
- Data preprocessing to handle missing values and convert categorical variables.
- Model training using machine learning algorithms like Logistic Regression, Decision Trees, or Random Forest.
- Model evaluation to check performance using accuracy, confusion matrix, etc.
- Deployment-ready predictions for new or unseen applications.

Data Description :

The dataset contains various attributes related to loan applicants. Below are the key features:

Column Name	Description
Loan_ID	Unique identifier for each loan application
Gender	Gender of the applicant (Male/Female)
Married	Applicant's marital status
Dependents	Number of dependents
Education	Education level (Graduate/Not Graduate)
Self_Employed	Whether the applicant is self-employed
ApplicantIncome	Income of the applicant
CoapplicantIncome	Income of the co-applicant
LoanAmount	Loan amount applied for
Loan_Amount_Term	Term of the loan in months
Credit_History	Credit history (1 = good, 0 = bad)
Property_Area	Type of property area (Urban/Semiurban/Rural)

Column Name	Description
Loan_Status	Target variable (Y = approved, N = not approved)

Code and Explanation :

Loan Status Prediction Using Support Vector Machine (SVM) :

This project applies a **Support Vector Machine (SVM)** algorithm to predict whether a loan application will be approved or rejected based on historical data. It uses Python libraries such as **NumPy**, **Pandas**, and **Seaborn** for data handling and visualization, and **scikit-learn** for modeling and evaluation.

Key Steps:

- Import necessary libraries for data manipulation, visualization, and machine learning.
- Split the dataset into training and testing sets.
- Train a **SVM classifier** using labeled data.
- Evaluate the model using **accuracy score** to measure its performance on unseen data.

This model helps financial institutions automate loan decisions and identify high-risk applications with better accuracy.

```
import numpy as np
import pandas as pd
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn.metrics import accuracy_score
```

Load Loan Status Dataset :

This line loads the loan status dataset from a local CSV file using Pandas. The dataset contains records of loan applications, including applicant details, income, loan amount, credit history, and the loan approval status.

The pd.read_csv() function reads the CSV file from the specified path and stores it in the variable loan_dataset, which will be used for further analysis and model building.

```python
loan_dataset = pd.read_csv("D:\Loan status project ML\loan status dataset.csv")
```

```python
type(loan_dataset)
```

```
pandas.core.frame.DataFrame
```

Display First 5 Rows of the Dataset :

This command displays the first 5 rows of the loan_dataset DataFrame using the .head() method. It's commonly used to get an initial look at the structure and content of the dataset, including:

- Column names
- Data types
- Sample values
- Any immediate missing or inconsistent data

You can also use .head(n) to view the first n rows.

```python
loan_dataset.head()
```

Python

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Terr
0	LP001002	Male	No	0	Graduate	No	5849	0.0	NaN	360.
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.0	360.
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66.0	360.
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	360.
4	LP001008	Male	No	0	Graduate	No	6000	0.0	141.0	360.

Check Dataset Dimensions :

This line retrieves the shape of the loan_dataset DataFrame using the .shape attribute. It returns a tuple representing:

- The number of rows (i.e., total loan applications)
- The number of columns (i.e., features or variables in the dataset)

This helps you quickly understand the size of the dataset and is often one of the first steps in exploratory data analysis (EDA).

Example: (614, 13) means 614 loan records and 13 features per record.

```
loan_dataset.shape
```

```
(614, 13)
```

Generate Statistical Summary of the Dataset :

This command generates a **descriptive statistical summary** of the numerical columns in the loan_dataset using the .describe() method. It provides insights into the distribution and spread of the data, including:

- **count**: Number of non-null entries
- **mean**: Average value
- **std**: Standard deviation
- **min**: Minimum value
- **25%**, **50%** (median), **75%**: Percentiles
- **max**: Maximum value

This is useful for identifying outliers, skewness, and general data ranges during exploratory data analysis (EDA).

```
loan_dataset.describe()
```

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
count	614.000000	614.000000	592.000000	600.00000	564.000000
mean	5403.459283	1621.245798	146.412162	342.00000	0.842199
std	6109.041673	2926.248369	85.587325	65.12041	0.364878
min	150.000000	0.000000	9.000000	12.00000	0.000000
25%	2877.500000	0.000000	100.000000	360.00000	1.000000
50%	3812.500000	1188.500000	128.000000	360.00000	1.000000
75%	5795.000000	2297.250000	168.000000	360.00000	1.000000
max	81000.000000	41667.000000	700.000000	480.00000	1.000000

Check for Missing Values in the Dataset :

This command checks for **missing (null/NaN)** values in each column of the loan_dataset. It uses:

- .isnull() to create a DataFrame of True/False values (where True means a missing value)
- .sum() to count how many True values (i.e., missing entries) exist per column

The output helps identify which features have incomplete data, so you can decide how to handle them — such as using imputation, dropping rows/columns, or other preprocessing strategies.

```
loan_dataset.isnull().sum()
```

```
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
```

Remove Missing Values and Verify Clean Dataset :

loan_dataset = loan_dataset.dropna()
This line **removes all rows** from the dataset that contain **any missing values** using the .dropna() method. It's a quick way to clean the data, but may result in loss of useful information if many rows are dropped.

loan_dataset.isnull().sum()
After dropping the nulls, this command checks again for any **remaining missing values** in each column to confirm that the dataset is now clean.

```
loan_dataset = loan_dataset.dropna()

loan_dataset.isnull().sum()
```

```
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
```

Converting Categorical Loan Status to Numerical Values :

This code replaces the categorical values in the Loan_Status column of the loan_dataset DataFrame with numerical equivalents—'N' is replaced by 0 and 'Y' by 1. This transformation is essential for machine learning models that require numerical input. The changes are made in place using inplace=True, and the first five rows of the modified dataset are then displayed using loan_dataset.head().

```
loan_dataset.replace({"Loan_Status":{'N':0,'Y':1}},inplace=True)

loan_dataset.head()
```

Python

```
C:\Users\Ayyan LapTOP\AppData\Local\Temp\ipykernel_9160\3468935185.py:2: FutureWarning: Downcasting behavior in `replace` is deprecat
  loan_dataset.replace({"Loan_Status":{'N':0,'Y':1}},inplace=True)
```

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Terr
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.0	360.
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66.0	360.
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	360.
4	LP001008	Male	No	0	Graduate	No	6000	0.0	141.0	360.
5	LP001011	Male	Yes	2	Graduate	Yes	5417	4196.0	267.0	360.

Analyzing the Distribution of Dependents in the Loan Dataset :

This code uses the value_counts() function to calculate and display the frequency of each unique value in the 'Dependents' column of the loan_dataset DataFrame. It helps in understanding how many applicants have 0, 1, 2, or 3+ dependents, which is useful for data exploration and identifying potential patterns or anomalies in the dataset.

```
loan_dataset['Dependents'].value_counts()
```

```
Dependents
0     274
2      85
1      80
3+     41
Name: count, dtype: int64
```

Replacing '3+' with 4 in the Dependents Column :

This code replaces all occurrences of '3+' with the integer 4 in the 'Dependents' column of the loan_dataset DataFrame using the replace() function. This standardizes the data by converting the '3+' string into a numeric format, making it more suitable for analysis and machine learning models. After the replacement, value_counts() is used again to display the updated frequency distribution of dependents.

```
loan_dataset = loan_dataset.replace(to_replace='3+', value=4)

loan_dataset['Dependents'].value_counts()
```

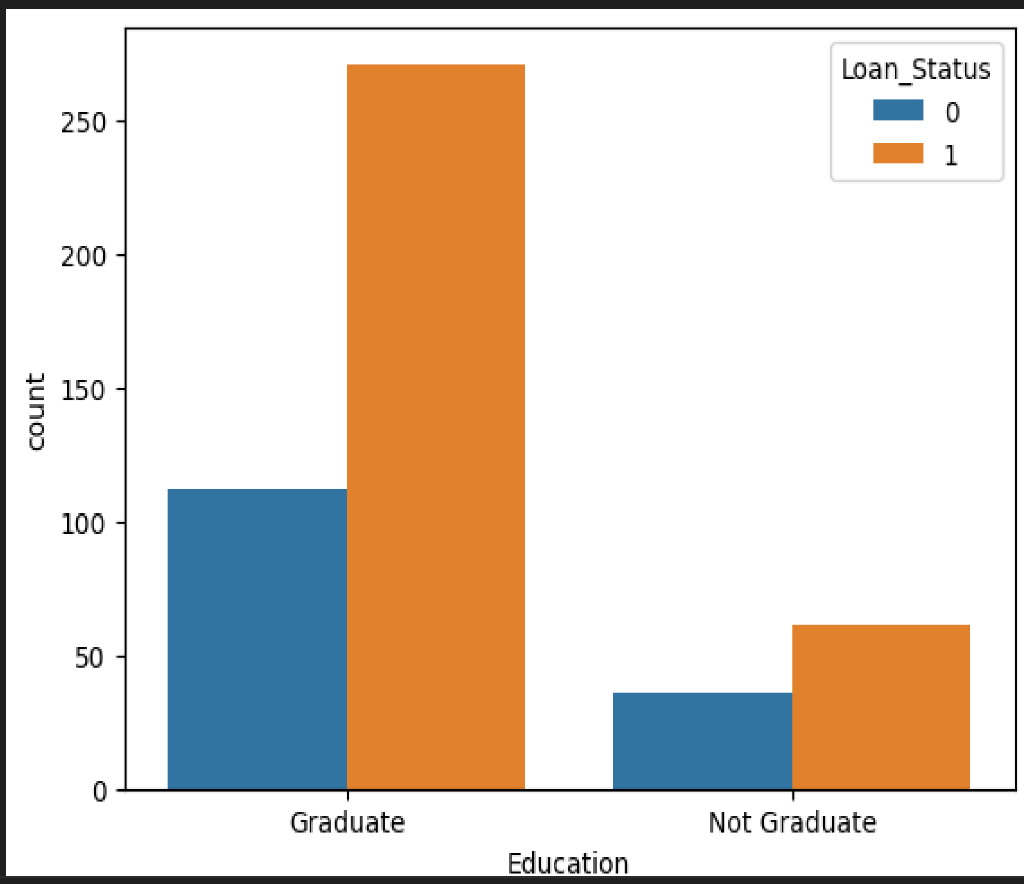
```
Dependents
0    274
2     85
1     80
4     41
Name: count, dtype: int64
```

Visualizing Loan Approval Status by Education Level :

This code uses seaborn.countplot() to create a bar chart showing the count of loan applications grouped by the applicant's education level ('Education'). The hue='Loan_Status' parameter further divides each bar based on the loan approval status (approved or not). This visualization helps identify whether education level has an impact on loan approval rates.

```
sns.countplot(x='Education',hue='Loan_Status',data=loan_dataset)
```

```
<Axes: xlabel='Education', ylabel='count'>
```

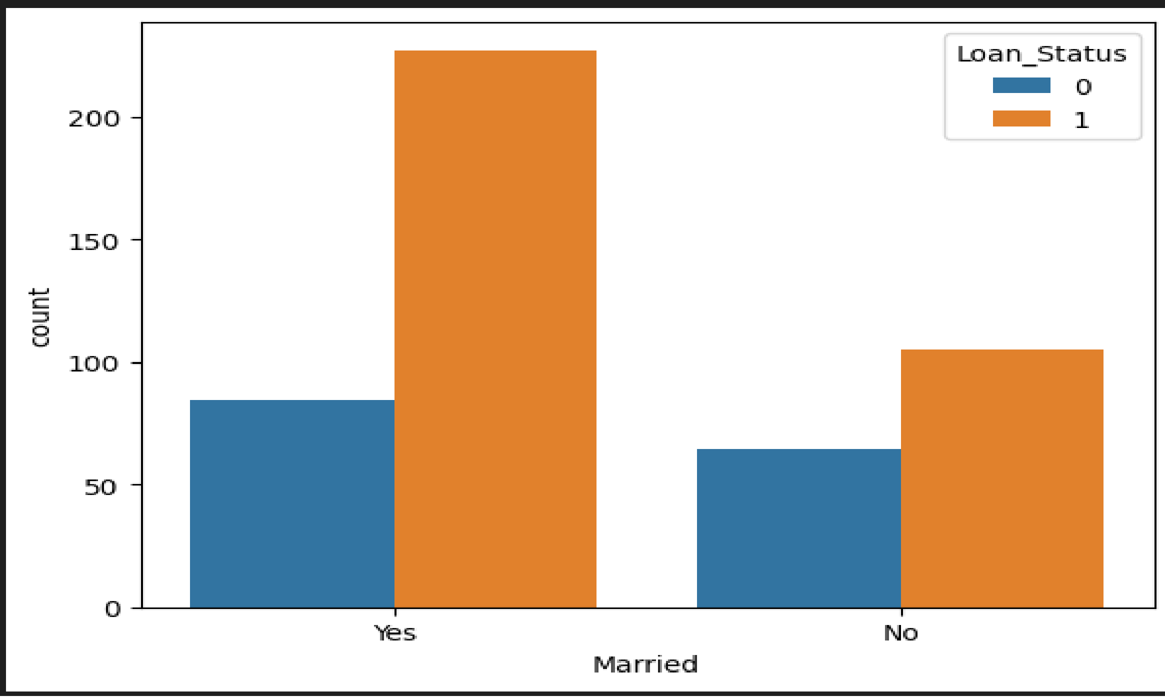


Visualizing Loan Approval Status by Marital Status :

This seaborn.countplot() visualizes the relationship between an applicant's marital status ('Married') and their loan approval status ('Loan_Status'). The hue='Loan_Status' parameter shows how many married and unmarried applicants were approved or denied loans. This plot helps assess whether marital status influences the likelihood of loan approval.

```
sns.countplot(x='Married',hue='Loan_Status',data=loan_dataset)
```

```
<Axes: xlabel='Married', ylabel='count'>
```



Encoding Categorical Features into Numerical Values :

This code replaces categorical string values with numerical equivalents in several columns of the loan_dataset DataFrame. This transformation is essential for preparing the data for machine learning models, which typically require numerical input. The changes are made in-place using inplace=True. Specifically:

- 'Married': 'No' → 0, 'Yes' → 1
- 'Gender': 'Female' → 0, 'Male' → 1
- 'Self_Employed': 'No' → 0, 'Yes' → 1
- 'Property_Area': 'Rural' → 0, 'Semiurban' → 1, 'Urban' → 2
- 'Education': 'Not Graduate' → 0, 'Graduate' → 1

After the replacements, loan_dataset.head() displays the first five rows of the updated DataFrame.

```python
loan_dataset.replace({'Married':{'No':0,'Yes':1},'Gender':{'Male':1,'Female':0},'Self_Employed':{'No':0,'Yes':1},
                      'Property_Area':{'Rural':0,'Semiurban':1,'Urban':2},'Education':{'Graduate':1,'Not Graduate':0}},inplace=True)

loan_dataset.head()
```

Python

```
C:\Users\Ayyan LapTOP\AppData\Local\Temp\ipykernel_9160\4088842081.py:2: FutureWarning: Downcasting behavior in `replace` is deprecated and will be removed in a fu
  loan_dataset.replace({'Married':{'No':0,'Yes':1},'Gender':{'Male':1,'Female':0},'Self_Employed':{'No':0,'Yes':1},
```

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area
1	LP001003	1	1	1	1	0	4583	1508.0	128.0	360.0	1.0	0
2	LP001005	1	1	0	1	1	3000	0.0	66.0	360.0	1.0	2
3	LP001006	1	1	0	0	0	2583	2358.0	120.0	360.0	1.0	2
4	LP001008	1	0	0	1	0	6000	0.0	141.0	360.0	1.0	2
5	LP001011	1	1	2	1	1	5417	4196.0	267.0	360.0	1.0	2

Separating Features and Target Variable for Model Training :

This code prepares the dataset for machine learning by separating the features (independent variables) and the target (dependent variable):

- X is created by dropping the 'Loan_ID' and 'Loan_Status' columns from loan_dataset, as 'Loan_ID' is just an identifier and 'Loan_Status' is the target variable.
- Y stores the 'Loan_Status' column, which represents whether a loan was approved (1) or not (0).

The print(X) and print(Y) statements display the resulting feature set and target variable, respectively.

```python
X = loan_dataset.drop(columns=['Loan_ID','Loan_Status'],axis=1)
Y = loan_dataset['Loan_Status']

print(X)
print(Y)
```

```
     Gender  Married Dependents  Education  Self_Employed  ApplicantIncome  \
1         1        1          1          1              0             4583
2         1        1          0          1              1             3000
3         1        1          0          0              0             2583
4         1        0          0          1              0             6000
5         1        1          2          1              1             5417
..      ...      ...        ...        ...            ...              ...
609       0        0          0          1              0             2900
610       1        1          4          1              0             4106
611       1        1          1          1              0             8072
612       1        1          2          1              0             7583
613       0        0          0          1              1             4583

     CoapplicantIncome  LoanAmount  Loan_Amount_Term  Credit_History  \
1               1508.0       128.0             360.0             1.0
```

```
1              1508.0      128.0            360.0           1.0
2                 0.0       66.0            360.0           1.0
3              2358.0      120.0            360.0           1.0
4                 0.0      141.0            360.0           1.0
5              4196.0      267.0            360.0           1.0
..                ...        ...              ...           ...
609               0.0       71.0            360.0           1.0
610               0.0       40.0            180.0           1.0
611             240.0      253.0            360.0           1.0
612               0.0      187.0            360.0           1.0
613               0.0      133.0            360.0           0.0
...
611    1
612    1
613    0
Name: Loan_Status, Length: 480, dtype: int64
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

Splitting the Dataset into Training and Testing Sets :

This code uses train_test_split from **scikit-learn** to divide the dataset into training and testing sets:

- test_size=0.1 specifies that 10% of the data will be used for testing.
- stratify=Y ensures that the split maintains the original class distribution of the target variable (Loan_Status), which is important for balanced evaluation.
- random_state=2 sets a seed for reproducibility.

The print() statement outputs the shapes of the full feature set (X), the training set (X_train), and the testing set (X_test) to confirm the split.

```python
X_train, X_test,Y_train,Y_test = train_test_split(X,Y,test_size=0.1,stratify=Y,random_state=2)

print(X.shape, X_train.shape, X_test.shape)
```

```
(480, 11) (432, 11) (48, 11)
```

Training a Support Vector Machine (SVM) Classifier with a Linear Kernel :

This code initializes and trains a **Support Vector Machine (SVM)** classifier using a **linear kernel**:

- svm.SVC(kernel='linear') creates an SVM model with a linear decision boundary, which is suitable when the data is linearly separable.
- classifier.fit(X_train, Y_train) trains the model using the training features (X_train) and labels (Y_train).

This step builds the classification model that can be used to predict loan approval based on the input features.

```
classifier = svm.SVC(kernel='linear')

classifier.fit(X_train,Y_train)
```

Evaluating SVM Classifier Accuracy on Training Data :

This code evaluates the performance of the trained SVM classifier on the training dataset:

- classifier.predict(X_train) generates predictions for the training data.
- accuracy_score(X_train_prediction, Y_train) computes the accuracy by comparing the predicted labels with the true labels.
- The result is printed as the training accuracy, showing how well the model has learned from the training set.

A very high training accuracy might indicate overfitting if the test accuracy is significantly lower.

```
X_train_prediction = classifier.predict(X_train)
training_data_accuray = accuracy_score(X_train_prediction,Y_train)

print('Accuracy on training data : ', training_data_accuray)
```

Evaluating SVM Classifier Accuracy on Test Data :

This code tests the performance of the trained Support Vector Machine (SVM) classifier on unseen data:

- classifier.predict(X_test) generates predictions for the test set.
- accuracy_score(X_test_prediction, Y_test) calculates the accuracy by comparing predicted labels with the actual labels in the test set.
- The accuracy is printed to assess how well the model generalizes to new, unseen data.

This is a crucial step for understanding the real-world performance of the model and checking for overfitting or underfitting.

```
X_test_prediction = classifier.predict(X_test)
test_data_accuray = accuracy_score(X_test_prediction,Y_test)

print('Accuracy on test data : ', test_data_accuray)
```