The University of Faisalabad

LAB MANUAL

Machine Learning

Submitted To:

Sir Saeed

Submitted By:

Taha Rehman

Registration no:

2023-BS-AI-90

Degree Program:

BS - Artificial Intelligence

Semester:

04

Table of Content

Project no 01

Bike Price Prediction using Regression

Description

This project focuses on predicting bike prices using regression analysis, a fundamental machine learning technique. The data set includes various attributes of bikes such as company, model, engine capacity, and more. The project follows a complete end-to-end machine learning pipeline starting from data loading, cleaning, exploratory data analysis (EDA), and feature engineering to building a predictive model.

The model is built using regression algorithms integrated within a pipeline that includes preprocessing steps like data standardization and encoding. Model performance is evaluated using metrics like R-squared and RMSE. Additionally, the project performs residual analysis and visualizes prediction results to ensure the model's accuracy and reliability. In the final stage, a predictive system is developed to forecast bike prices based on user inputs.

This project demonstrates how data science techniques can be applied to practical business problems, especially in the automotive domain, and highlights the importance of preprocessing, feature selection, and model evaluation in building effective predictive systems.

Import libraries

Essential Python libraries such as pandas, numpy, matplotlib, seaborn, and sklearn are imported to handle data, visualization, and modeling.

```python
[73]: import pandas as pd
      import numpy as np
      import matplotlib.pyplot as plt
      import seaborn as sns
```

```python
[75]: from sklearn.model_selection import train_test_split
      from sklearn.linear_model import LinearRegression
      from sklearn.metrics import mean_squared_error, r2_score
      from sklearn.preprocessing import OneHotEncoder
      from sklearn.compose import ColumnTransformer
      from sklearn.pipeline import Pipeline
```

1. LOAD DATASET

The dataset containing various bike features is loaded for analysis and model training.

```python
[5]: df = pd.read_csv('BIKE DETAILS.csv')
     print("Dataset loaded successfully.")
     print(f"Shape: {df.shape}")
     print("\nFirst 5 rows:")
     display(df.head())
```

```
Dataset loaded successfully.
Shape: (1061, 7)

First 5 rows:
```

	name	selling_price	year	seller_type	owner	km_driven	ex_showroom_price
0	Royal Enfield Classic 350	175000	2019	Individual	1st owner	350	NaN
1	Honda Dio	45000	2017	Individual	1st owner	5650	NaN
2	Royal Enfield Classic Gunmetal Grey	150000	2018	Individual	1st owner	12000	148114.0
3	Yamaha Fazer FI V 2.0 [2016-2018]	65000	2015	Individual	1st owner	23000	89643.0
4	Yamaha SZ [2013-2014]	20000	2011	Individual	2nd owner	21000	NaN

2. DATA INFORMATION

Summary and structure of the dataset are explored using functions like info() and describe() to understand data types and basic statistics.

```
[7]: print("\nDataset Info:")
     print(df.info())
```

```
Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1061 entries, 0 to 1060
Data columns (total 7 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   name               1061 non-null   object
 1   selling_price      1061 non-null   int64
 2   year               1061 non-null   int64
 3   seller_type        1061 non-null   object
 4   owner              1061 non-null   object
 5   km_driven          1061 non-null   int64
 6   ex_showroom_price  626 non-null    float64
dtypes: float64(1), int64(3), object(3)
memory usage: 58.2+ KB
None
```

3. CHECK MISSING VALUES

Null values are checked and handled to ensure the dataset is clean and ready for analysis.

```
[9]: missing_values = df.isnull().sum()
     print("\nMissing values in each column:")
     print(missing_values)
```

```
Missing values in each column:
name                   0
selling_price          0
year                   0
seller_type            0
owner                  0
km_driven              0
ex_showroom_price    435
dtype: int64
```

```
[10]: df = df.dropna()
      print(f"\nShape after dropping missing values: {df.shape}")
```

```
Shape after dropping missing values: (626, 7)
```

[12]:
```
print("\nSummary statistics:")
display(df.describe())
```

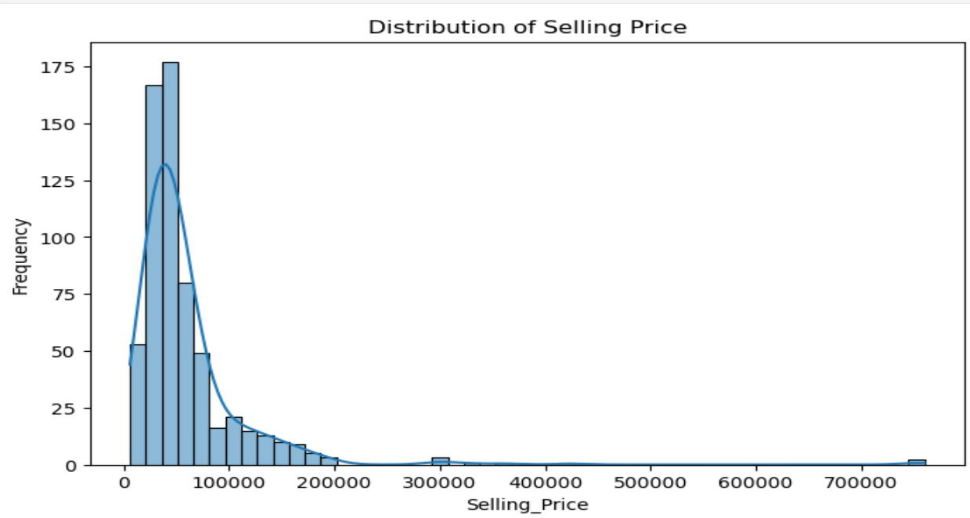
Summary statistics:

	selling_price	year	km_driven	ex_showroom_price
count	626.000000	626.000000	626.000000	6.260000e+02
mean	59445.164537	2014.800319	32671.576677	8.795871e+04
std	59904.350888	3.018885	45479.661039	7.749659e+04
min	6000.000000	2001.000000	380.000000	3.049000e+04
25%	30000.000000	2013.000000	13031.250000	5.485200e+04
50%	45000.000000	2015.000000	25000.000000	7.275250e+04
75%	65000.000000	2017.000000	40000.000000	8.703150e+04
max	760000.000000	2020.000000	585659.000000	1.278000e+06

4. EXPLORATORY DATA ANALYSIS (EDA)

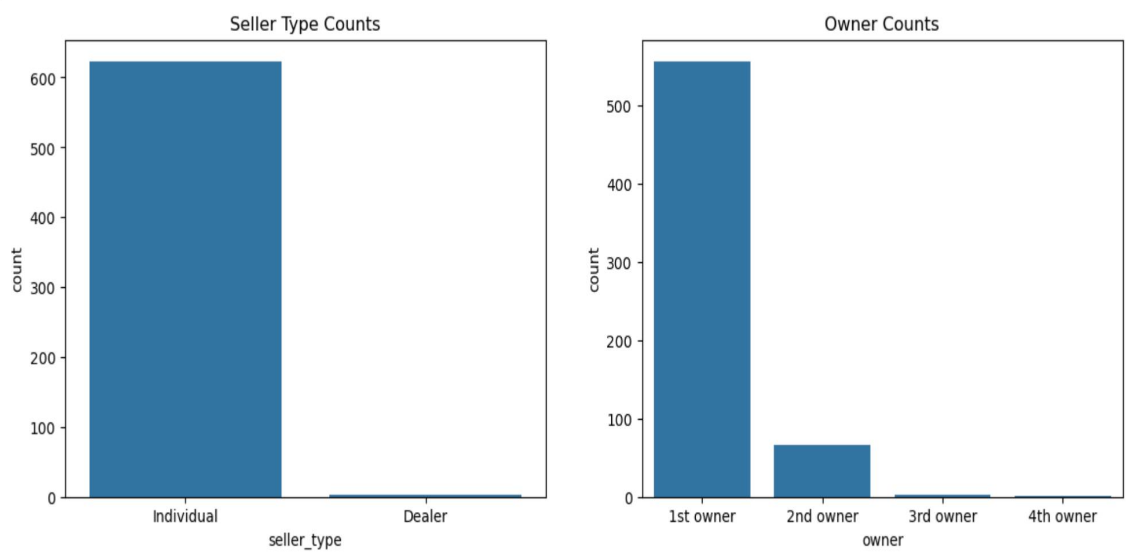
Data is visualized using plots and graphs to identify patterns, relationships, and outliers in features like engine size, mileage, brand, etc.

[14]:
```
plt.figure(figsize=(8,5))
sns.histplot(df['selling_price'], bins=50, kde=True)
plt.title('Distribution of Selling Price')
plt.xlabel('Selling_Price')
plt.ylabel('Frequency')
plt.show()
```

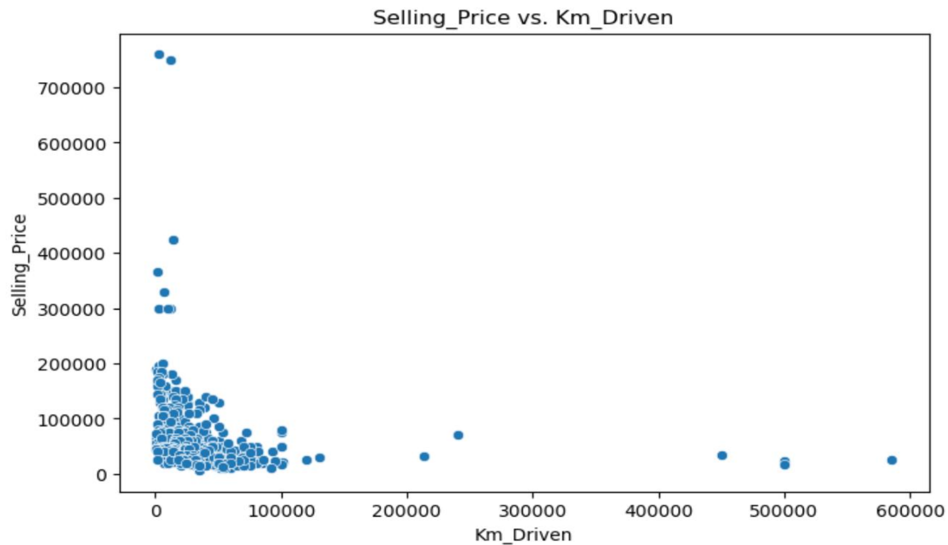


```
[15]: plt.figure(figsize=(6,4))
      sns.boxplot(y=df['selling_price'])
      plt.title('Boxplot of Selling Price')
      plt.show()
```

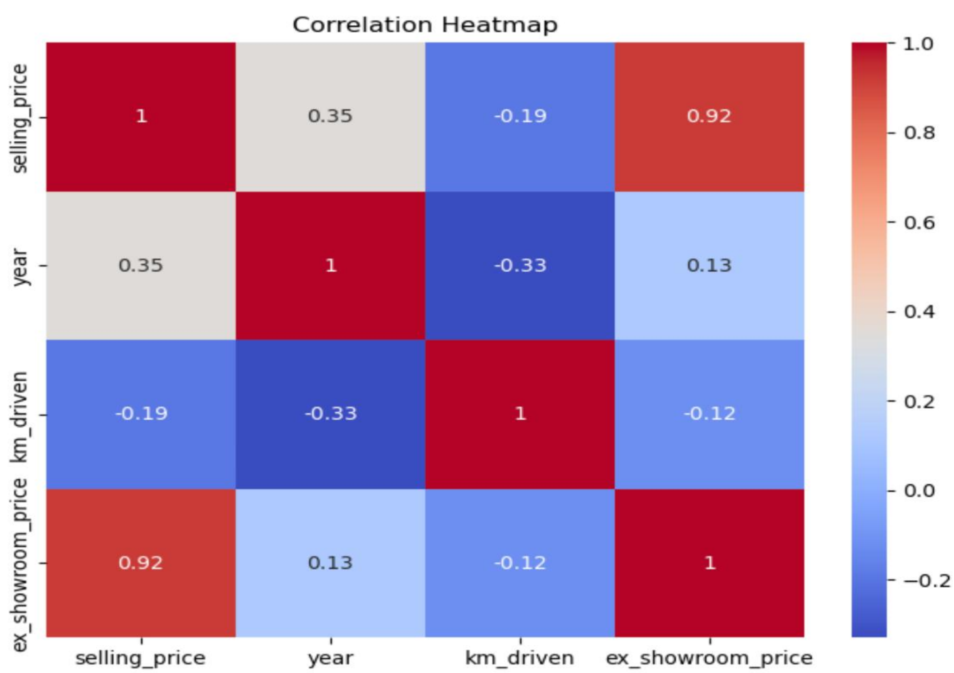

```
[16]: fig, axes = plt.subplots(1, 2, figsize=(14,5))
      sns.countplot(x='seller_type', data=df, ax=axes[0])
      axes[0].set_title('Seller Type Counts')
      sns.countplot(x='owner', data=df, ax=axes[1])
      axes[1].set_title('Owner Counts')
      plt.show()
```



```
[17]: plt.figure(figsize=(8,5))
sns.scatterplot(x='km_driven', y='selling_price', data=df)
plt.title('Selling_Price vs. Km_Driven')
plt.xlabel('Km_Driven')
plt.ylabel('Selling_Price')
plt.show()
```




```
[18]: num_cols = ['selling_price', 'year', 'km_driven', 'ex_showroom_price']
plt.figure(figsize=(8,6))
sns.heatmap(df[num_cols].corr(), annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```



5. FEATURE ENGINEERING

New features are created or transformed from existing ones to improve model accuracy. This may include encoding categorical variables and extracting useful information

```
[20]: current_year = 2025
      df['age'] = current_year - df['year']
```

```
[21]: df.drop(columns=['year'], inplace=True)
```

```
[22]: print("\nAdded 'age' feature and dropped 'year'.")

      Added 'age' feature and dropped 'year'.
```

6. DATA PREPARATION FOR MODELING

Features and target variables are selected, and data is prepared for input into the machine learning model.

```
[24]: X = df.drop(columns=['selling_price', 'name'])
      y = df['selling_price']
```

```
[25]: categorical_features = ['seller_type', 'owner']
      numerical_features = X.drop(columns=categorical_features).columns.tolist()
```

```
[26]: print(f"Numerical features: {numerical_features}")
      print(f"Categorical features: {categorical_features}")

      Numerical features: ['km_driven', 'ex_showroom_price', 'age']
      Categorical features: ['seller_type', 'owner']
```

7. TRAIN-TEST SPLIT

· The dataset is split into training and testing sets to evaluate model performance objectively.

```python
[28]: X_train, X_test, y_train, y_test = train_test_split(
          X, y, test_size=0.2, random_state=42
      )
      print(f"\nTraining data shape: {X_train.shape}")
      print(f"Testing data shape: {X_test.shape}")
```

```
Training data shape: (500, 5)
Testing data shape: (126, 5)
```

8. PIPELINE WITH PREPROCESSING AND MODELING

```python
[30]: from sklearn.preprocessing import OneHotEncoder
      from sklearn.compose import ColumnTransformer

      preprocessor = ColumnTransformer(
          transformers=[
              ('cat', OneHotEncoder(drop='first', sparse_output=False), categorical_features)
          ],
          remainder='passthrough'
      )
```

```python
[31]: model_pipeline = Pipeline([
          ('preprocessor', preprocessor),
          ('regressor', LinearRegression())
      ])
```

```python
[32]: # 10. TRAINING THE MODEL
```

```python
[33]: model_pipeline.fit(X_train, y_train)
      print("\nModel training completed.")
```

```
Model training completed.
```

9. PREDICTION ON TEST DATA

```
[35]: y_pred = model_pipeline.predict(X_test)
```

10. MODEL EVALUATION

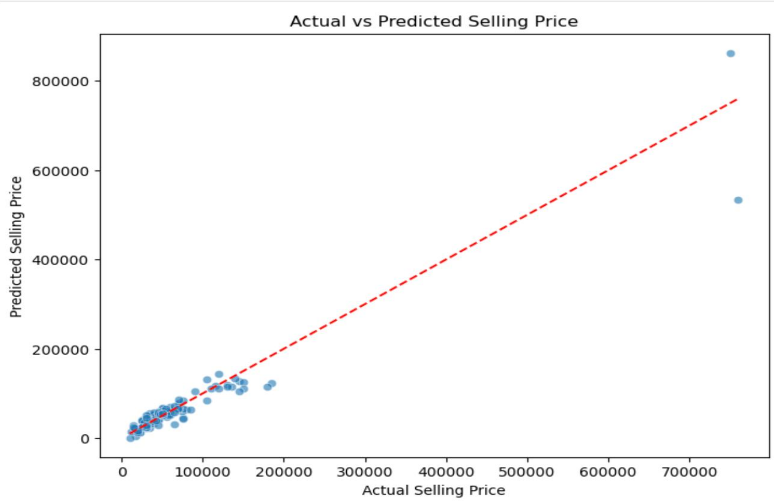
```
[37]: # Calculate evaluation metrics
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)
```

```
[38]: print(f"\nModel Evaluation Metrics:")
print(f"Mean Squared Error (MSE): {mse:.2f}")
print(f"Root Mean Squared Error (RMSE): {rmse:.2f}")
print(f"R-squared (R²): {r2:.2f}")
```

```
Model Evaluation Metrics:
Mean Squared Error (MSE): 720000517.44
Root Mean Squared Error (RMSE): 26832.83
R-squared (R²): 0.92
```

11. VISUALIZATION OF PREDICTIONS

```
[40]: plt.figure(figsize=(8,6))
sns.scatterplot(x=y_test, y=y_pred, alpha=0.6)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--')
plt.xlabel('Actual Selling Price')
plt.ylabel('Predicted Selling Price')
plt.title('Actual vs Predicted Selling Price')
plt.show()
```

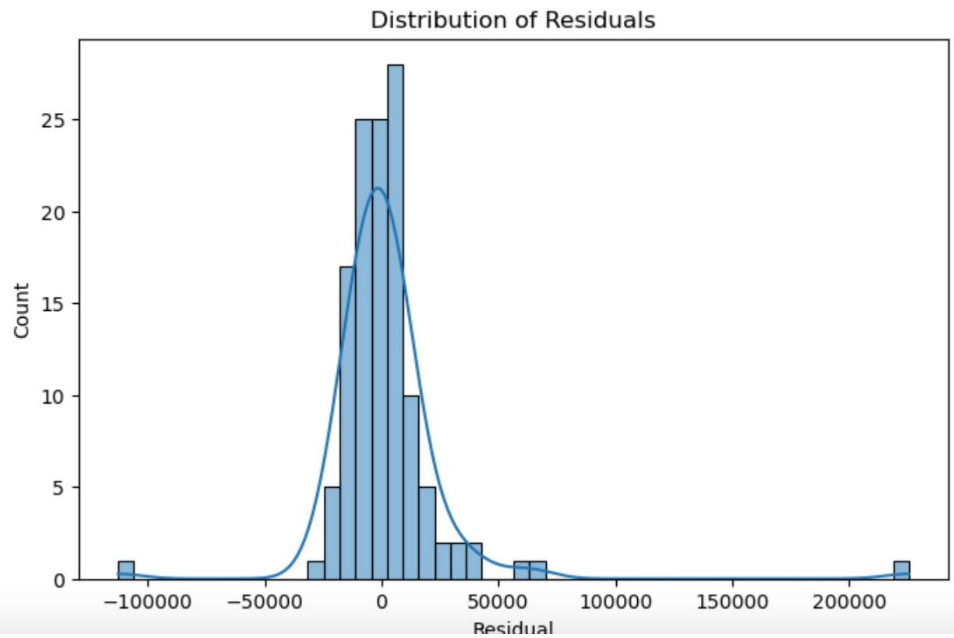


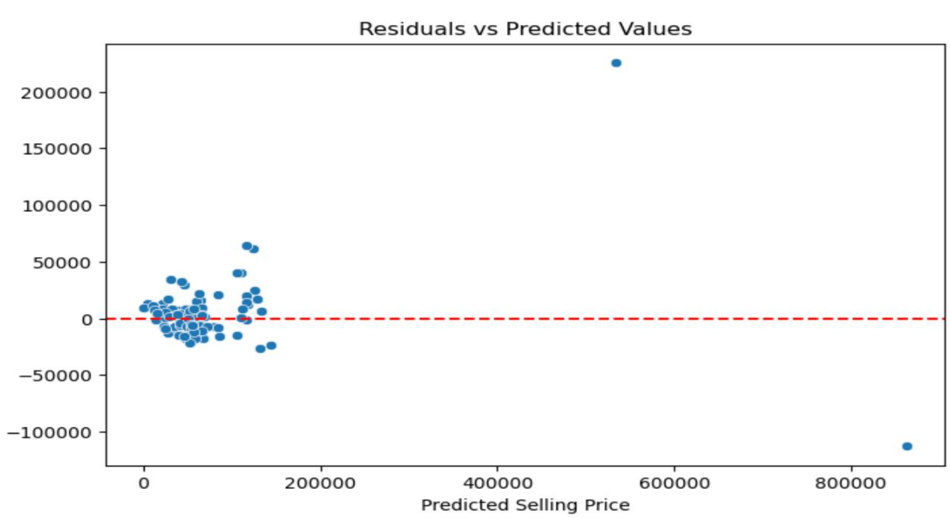
12. RESIDUAL ANALYSIS

[42]:

```
residuals = y_test - y_pred
plt.figure(figsize=(8,5))
sns.histplot(residuals, bins=50, kde=True)
plt.title('Distribution of Residuals')
plt.xlabel('Residual')
plt.show()

plt.figure(figsize=(8,5))
sns.scatterplot(x=y_pred, y=residuals)
plt.axhline(y=0, color='r', linestyle='--')
plt.xlabel('Predicted Selling Price')
plt.ylabel('Residual')
plt.title('Residuals vs Predicted Values')
plt.show()
```





Project no 2

Diabetes Prediction using Classification

Description:

This project aims to predict whether a person has diabetes or not based on various medical attributes using machine learning. We use a dataset that contains features like glucose level, blood pressure, insulin, BMI, age, etc. The project involves data preprocessing (handling missing values, encoding, scaling), training a machine learning model, and then testing its accuracy.

The goal is to create a smart system that can help in early detection of diabetes, which can be useful for doctors, clinics, and health researchers.

1. Import Required Libraries

This step brings in all the necessary Python libraries needed for data handling, visualization, and model building. These include pandas, numpy, matplotlib, seaborn, and sklearn.

[3]:
```
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn.metrics import accuracy_score
```

Data Collection and Analysis

PIMA Diabetes Dataset

2. Load the Dataset

We load the diabetes dataset using pandas. It contains information about patient health used to predict diabetes.

```
[ ]: diabetes_dataset = pd.read_csv('/content/diabetes.csv')
```

```
[ ]: pd.read_csv?
```

```
[ ]: # printing the first 5 rows of the dataset
     diabetes_dataset.head()
```

[4]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

3. Explore and Understand the Data

We look at the data using functions like head( ), shape, and info( ) to understand what kind of data we have and how many rows/columns there are.

```
[ ]: # number of rows and Columns in this dataset
     diabetes_dataset.shape
```

```
[5]: (768, 9)
```

```
[ ]: # getting the statistical measures of the data
     diabetes_dataset.describe()
```

[6]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

```
[ ]: diabetes_dataset['Outcome'].value_counts()
```

```
[7]: 0    500
     1    268
     Name: Outcome, dtype: int64
```

0 --> Non-Diabetic

1 --> Diabetic

4. Handle Missing Values

We check for any missing values in the dataset and fix them by either filling with a default value or removing the row/column to keep the data clean.

```
[ ]: diabetes_dataset.groupby('Outcome').mean()
```

[8]:

Outcome	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	3.298000	109.980000	68.184000	19.664000	68.792000	30.304200	0.429734	31.190000
1	4.865672	141.257463	70.824627	22.164179	100.335821	35.142537	0.550500	37.067164

```
[ ]: # separating the data and labels
     X = diabetes_dataset.drop(columns = 'Outcome', axis=1)
     Y = diabetes_dataset['Outcome']
```

```
[ ]: print(X)
```

```
     Pregnancies  Glucose  BloodPressure  ...   BMI  DiabetesPedigreeFunction  Age
0              6      148             72  ...  33.6                     0.627   50
1              1       85             66  ...  26.6                     0.351   31
2              8      183             64  ...  23.3                     0.672   32
3              1       89             66  ...  28.1                     0.167   21
4              0      137             40  ...  43.1                     2.288   33
..           ...      ...            ...  ...   ...                       ...  ...
763           10      101             76  ...  32.9                     0.171   63
764            2      122             70  ...  36.8                     0.340   27
765            5      121             72  ...  26.2                     0.245   30
766            1      126             60  ...  30.1                     0.349   47
767            1       93             70  ...  30.4                     0.315   23

[768 rows x 8 columns]
```

```
[ ]: print(Y)
```

```
0      1
1      0
2      1
3      0
4      1
      ..
763    0
764    0
765    0
766    1
767    0
Name: Outcome, Length: 768, dtype: int64
```

5. Encode Categorical Variables

Any column that contains text data (like gender or type) is converted into numbers using Label Encoding or One Hot Encoding so that the model can understand it.

```
[ ]: print(Y)
```

```
0      1
1      0
2      1
3      0
4      1
      ..
763    0
764    0
765    0
766    1
767    0
Name: Outcome, Length: 768, dtype: int64
```

6. Feature Scaling

This step makes sure that all numeric values are on the same scale (like between 0 and 1). It helps models train faster and better.

```
[ ]: scaler = StandardScaler()
```

```
[ ]: scaler.fit(X)
```

```
[13]: StandardScaler(copy=True, with_mean=True, with_std=True)
```

```
[ ]: standardized_data = scaler.transform(X)
```

```
[ ]: print(standardized_data)
```

```
[[ 0.63994726  0.84832379  0.14964075 ...  0.20401277  0.46849198
   1.4259954 ]
 [-0.84488505 -1.12339636 -0.16054575 ... -0.68442195 -0.36506078
  -0.19067191]
 [ 1.23388019  1.94372388 -0.26394125 ... -1.10325546  0.60439732
  -0.10558415]
 ...
 [ 0.3429808   0.00330087  0.14964075 ... -0.73518964 -0.68519336
  -0.27575966]
 [-0.84488505  0.1597866  -0.47073225 ... -0.24020459 -0.37110101
   1.17073215]
 [-0.84488505 -0.8730192   0.04624525 ... -0.20212881 -0.47378505
  -0.87137393]]
```

```
[ ]: X = standardized_data
     Y = diabetes_dataset['Outcome']
```

9. Split the Data set

We split our dataset into training and testing parts. Training data is used to build the model and testing data is used to check how good it performs.

```
Y = diabetes_dataset['Outcome']

[ ]: print(X)
     print(Y)

[[ 0.63994726  0.84832379  0.14964075 ...  0.20401277  0.46849198
   1.4259954 ]
 [-0.84488505 -1.12339636 -0.16054575 ... -0.68442195 -0.36506078
  -0.19067191]
 [ 1.23388019  1.94372388 -0.26394125 ... -1.10325546  0.60439732
  -0.10558415]
 ...
 [ 0.3429808   0.00330087  0.14964075 ... -0.73518964 -0.68519336
  -0.27575966]
 [-0.84488505  0.1597866  -0.47073225 ... -0.24020459 -0.37110101
   1.17073215]
 [-0.84488505 -0.8730192   0.04624525 ... -0.20212881 -0.47378505
  -0.87137393]]
0      1
1      0
2      1
3      0
4      1
      ..
763    0
764    0
765    0
766    1
767    0
Name: Outcome, Length: 768, dtype: int64
```

8. Train the Model

We use a machine learning model (like Logistic Regression or Random Forest) and train it using the training data so it can learn the patterns.

```
[ ]: X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size = 0.2, stratify=Y, random_state=2)

[ ]: print(X.shape, X_train.shape, X_test.shape)

(768, 8) (614, 8) (154, 8)
```

9. Make Predictions

Now we use the trained model to make predictions on test data to see if it can correctly guess whether a person has diabetes or not.

```
[ ]: input_data = (5,166,72,19,175,25.8,0.587,51)

     # changing the input_data to numpy array
     input_data_as_numpy_array = np.asarray(input_data)

     # reshape the array as we are predicting for one instance
     input_data_reshaped = input_data_as_numpy_array.reshape(1,-1)

     # standardize the input data
     std_data = scaler.transform(input_data_reshaped)
     print(std_data)

     prediction = classifier.predict(std_data)
     print(prediction)

     if (prediction[0] == 0):
       print('The person is not diabetic')
     else:
       print('The person is diabetic')
```

```
[[ 0.3429808   1.41167241  0.14964075 -0.09637905  0.82661621 -0.78595734
   0.34768723  1.51108316]]
[1]
The person is diabetic
```

10. Evaluate the Model

We check the model's performance using accuracy, confusion matrix, precision, and recall to see how well it predicts and where it makes mistakes.

```
[ ]: # accuracy score on the training data
     X_train_prediction = classifier.predict(X_train)
     training_data_accuracy = accuracy_score(X_train_prediction, Y_train)
```

```
[ ]: print('Accuracy score of the training data : ', training_data_accuracy)
```

```
Accuracy score of the training data :  0.7866449511400652
```

```
[ ]: # accuracy score on the test data
     X_test_prediction = classifier.predict(X_test)
     test_data_accuracy = accuracy_score(X_test_prediction, Y_test)
```

```
[ ]: print('Accuracy score of the test data : ', test_data_accuracy)
```

```
Accuracy score of the test data :  0.7727272727272727
```

Making a Predictive System