AIM:- Data Loading, Data preprocessing, Data exploration, and Data preparation.

1. **Data Loading:**

- Load a dataset (Iris dataset: https://www.kaggle.com/datasets/uciml/iris) into your preferred ML environment (Python).

```
import pandas as pd
df = pd.read_csv("D:\\5th Sem\\LAB\\ML\\Iris.csv")
```

- Display the first few rows of the dataset to inspect its structure and content.

```
print("First 5 rows of the Iris dataset:-\n", df.head())
```

Output:-

```
First 5 rows of the Iris dataset:-
   Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm      Species
0   1            5.1           3.5            1.4           0.2  Iris-setosa
1   2            4.9           3.0            1.4           0.2  Iris-setosa
2   3            4.7           3.2            1.3           0.2  Iris-setosa
3   4            4.6           3.1            1.5           0.2  Iris-setosa
4   5            5.0           3.6            1.4           0.2  Iris-setosa
```

- Check the dimensions of the dataset (number of rows and columns).

```
print("Dimension of the dataset: ", df.shape)
```

Output:-

```
Dimension of the dataset:  (150, 6)
```

- Identify the data types of each column (numeric, categorical, text, etc.).

```
print("Data types of each column:\n", df.dtypes)
```

Output:-

```
Data types of each column:
 Id                 int64
SepalLengthCm    float64
SepalWidthCm     float64
PetalLengthCm    float64
PetalWidthCm     float64
Species           object
dtype: object
```

2. Data Exploration:

- Calculate basic summary statistics for the numeric columns (mean, median, min, max, standard deviation).

```
import matplotlib.pyplot as plt
import seaborn as sns
from DataLoading import df
numeric_columns = ['SepalLengthCm', 'SepalWidthCm',
'PetalLengthCm', 'PetalWidthCm']
print("Basic Statistics for the numeric columns:-\n",
df[numeric_columns].describe())
```

Output:-

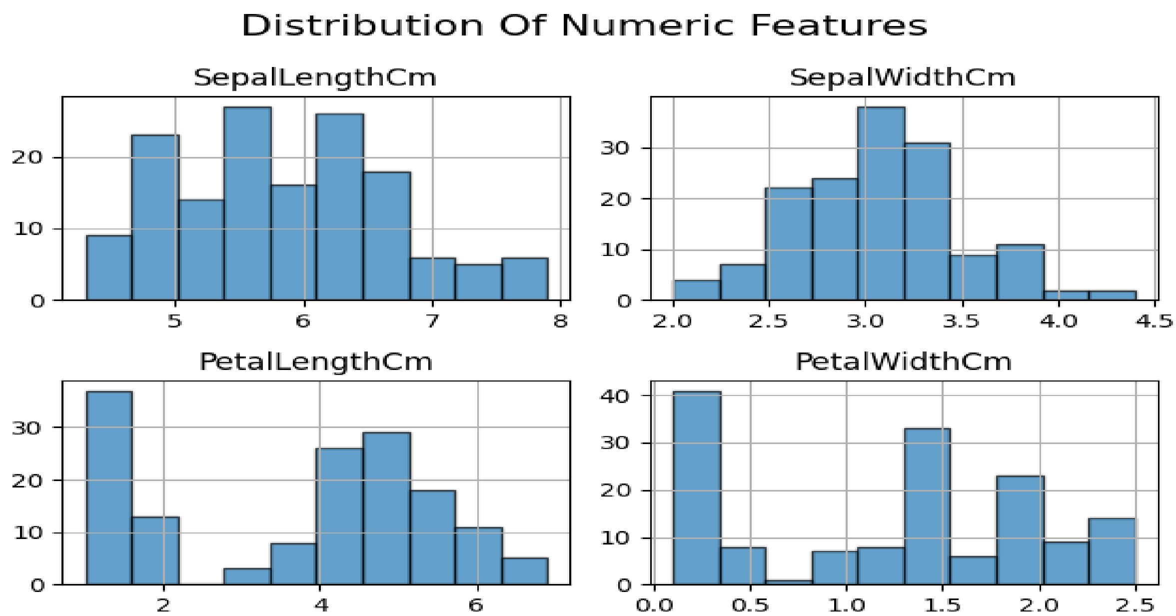
Basic Statistics for the numeric columns:-

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

- Visualize the distribution of numeric features using histograms or box plots.

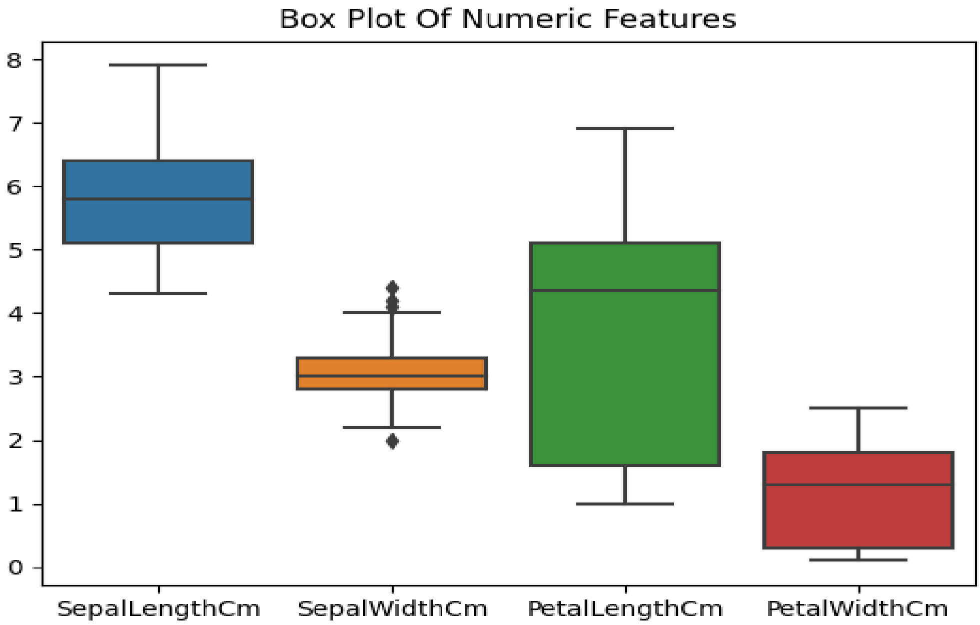
```
# Creating histograms for numeric features
df[numeric_columns].hist(edgecolor='black', alpha=0.7)
plt.suptitle("Distribution Of Numeric Features", fontsize=16)
plt.tight_layout()
plt.show()
```

Output:-



```
# Creating box plots for numeric features
sns.boxplot(data=df[numeric_columns])
plt.title("Box Plot Of Numeric Features")
plt.show()
```

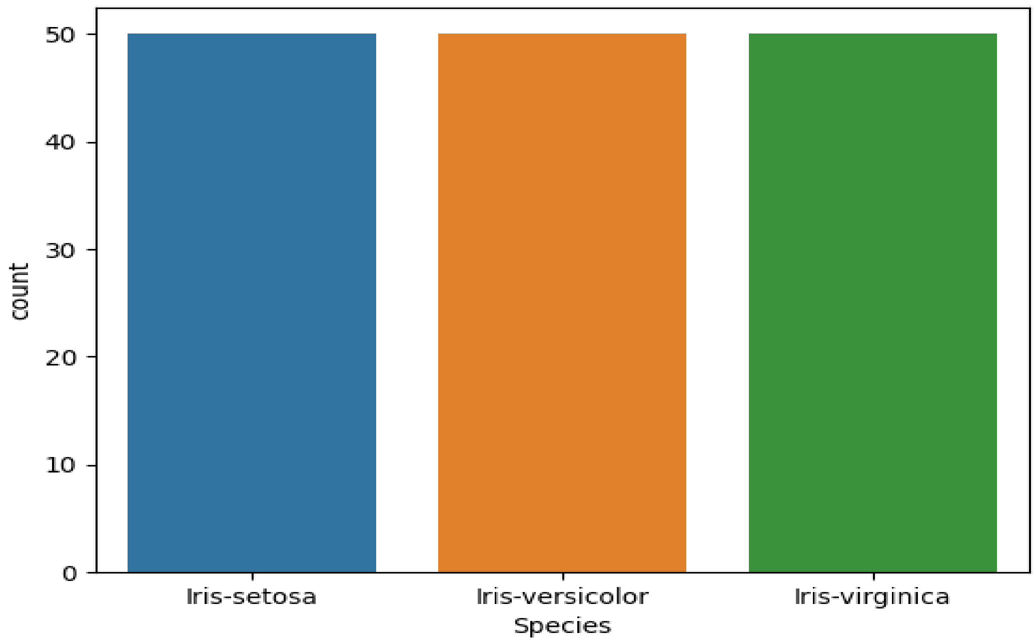
Output:-



- Explore the frequency distribution of categorical features using bar plots.

```
sns.countplot(data=df, x='Species')
plt.show()
```

Output:-



3. Data Preprocessing:

- Handle missing values: Identify and handle any missing values in the dataset (e.g., imputation, removal).

```
from DataLoading import df
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler, MinMaxScaler
# Display the number of missing values in each column
missingValues = df.isnull().sum()
print("Missing values per column:-")
print(missingValues)
```

Output:-

```
Missing values per column:-
Id               0
SepalLengthCm    0
SepalWidthCm     0
PetalLengthCm    0
PetalWidthCm     0
Species          0
dtype: int64
```

- Display the number of missing values after imputation.

```
# Fill the missing values with the mean of each column except
column4
df_imputed = df.fillna(df.drop('Species', axis=1).mean())
missingValues_after_imputation = df_imputed.isnull().sum()
print("Missing Values after imputation:-")
print(missingValues_after_imputation)
```

Output:-

```
Missing Values after imputation:-
Id               0
SepalLengthCm    0
SepalWidthCm     0
PetalLengthCm    0
PetalWidthCm     0
Species          0
dtype: int64
```

- Encode categorical variables: Convert categorical variables into numerical form (e.g., one-hot encoding, label encoding).

```python
# Categorical Column
categorical_column = 'Species'
# One-Hot Encoding (Creating Dummy Variables)
data_encoded_onehot = pd.get_dummies(df,
columns=[categorical_column], prefix=[categorical_column])
# Display the first few rows of the encoded data
print("One-Hot Encoded Data:")
print(data_encoded_onehot.head())
# Label Encoding
data_encoded_label = df.copy()
label_encoder = LabelEncoder()
data_encoded_label[categorical_column] =
label_encoder.fit_transform(data_encoded_label[categorical_column])
# Displays the first few rows of the encoded data
print("\nLabel Encoded Data:")
print(data_encoded_label.head())
# Reverse Label Encoding(for demonstration purposes)
reverse_encoded_labels =
label_encoder.inverse_transform(data_encoded_label[categorical_col
umn])
data_encoded_label[categorical_column] = reverse_encoded_labels
# Display the first few rows of the data with reversed level encoding
print("\nData with Reversed Label Encoding:")
print(data_encoded_label.head())
```

Output:-

```
One-Hot Encoded Data:
   Id  SepalLengthCm  ...  Species_Iris-versicolor  Species_Iris-virginica
0   1            5.1  ...                    False                   False
1   2            4.9  ...                    False                   False
2   3            4.7  ...                    False                   False
3   4            4.6  ...                    False                   False
4   5            5.0  ...                    False                   False

[5 rows x 8 columns]

Label Encoded Data:
   Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm  Species
0   1            5.1           3.5            1.4           0.2        0
1   2            4.9           3.0            1.4           0.2        0
2   3            4.7           3.2            1.3           0.2        0
3   4            4.6           3.1            1.5           0.2        0
4   5            5.0           3.6            1.4           0.2        0
```

```
Data with Reversed Label Encoding:
   Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm      Species
0   1            5.1           3.5            1.4           0.2  Iris-setosa
1   2            4.9           3.0            1.4           0.2  Iris-setosa
2   3            4.7           3.2            1.3           0.2  Iris-setosa
3   4            4.6           3.1            1.5           0.2  Iris-setosa
4   5            5.0           3.6            1.4           0.2  Iris-setosa
```

- Feature scaling: Normalize or standardize numeric features to bring them to a similar scale.

```
numeric_columns = ['SepalLengthCm', 'SepalWidthCm',
'PetalLengthCm', 'PetalWidthCm']
# Standardization
scaler_standard = StandardScaler()
data_standardized =
pd.DataFrame(scaler_standard.fit_transform(df[numeric_columns]),
columns=numeric_columns)
# Display the first few rows of standardized data
print("Standardized Data:")
print(data_standardized.head())

# Normalization(MinMax Scaling)
scaler_minmax = MinMaxScaler()
data_normalized =
pd.DataFrame(scaler_minmax.fit_transform(df[numeric_columns]),
columns=numeric_columns)
# Display the first few rows of normalized data
print("\nNormalized Data:")
print(data_normalized.head())
```

Output:-

```
Standardized Data:
   SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm
0      -0.900681      1.032057      -1.341272     -1.312977
1      -1.143017     -0.124958      -1.341272     -1.312977
2      -1.385353      0.337848      -1.398138     -1.312977
3      -1.506521      0.106445      -1.284407     -1.312977
4      -1.021849      1.263460      -1.341272     -1.312977

Normalized Data:
   SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm
0       0.222222      0.625000       0.067797      0.041667
1       0.166667      0.416667       0.067797      0.041667
2       0.111111      0.500000       0.050847      0.041667
3       0.083333      0.458333       0.084746      0.041667
4       0.194444      0.666667       0.067797      0.041667

Process finished with exit code 0
```

4. Data Preparation for ML:

- Split the dataset into training and testing sets (e.g., 80% for training, 20% for testing).

```
from DataLoading import df
from sklearn.model_selection import train_test_split
# Features and target variable
X = df.drop(columns=['Species'])  # Features
Y = df['Species']  # Target Variable
# Split the dataset into training and testing sets(80% training, 20%testing)
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2,
random_state=42)
# Display the shape of the training and testing sets
print("Shape of X_train:", X_train.shape)
print("Shape of X_test:", X_test.shape)
print("Shape of Y_train:", Y_train.shape)
print("Shape of Y_test:", Y_test.shape)
```

Output:-

```
Shape of X_train: (120, 5)
Shape of X_test: (30, 5)
Shape of Y_train: (120,)
Shape of Y_test: (30,)
```

- Ensure the data is in the appropriate format for the ML algorithms (e.g., arrays, matrices).

```
# Convert the data to arrays or matrices
X_train_array = X_train.values
X_test_array = X_test.values
Y_train_array = Y_train.values
Y_test_array = Y_test.values
# Display the type and shape of the array
print("Type of X_train_array:", type(X_train_array))
print("Shape of X_train_array:", X_train_array.shape)
print("Type of Y_train_array:", type(Y_train_array))
print("Shape of Y_train_array:", X_train_array.shape)
```

Output:-

```
Type of X_train_array: <class 'numpy.ndarray'>
Shape of X_train_array: (120, 5)
Type of Y_train_array: <class 'numpy.ndarray'>
Shape of Y_train_array: (120, 5)
```

Submitted By,
Name- Shankar Singh Mahanty
Regd. No.- 2101020758
Roll No.- CSE21238
Group- 3
Sem- 5th Branch- CSE