Labcycle 1. Install and set up Python and essential libraries like NumPy and pandas.

Installation of Python

Step 1: Search for Python

Type "Python download" in the Google search bar and press Enter key. In the list of links shown, select the

very link or click on the official website link: https://www.python.org/downloads/

Step 2: Select Version to Install Python

Choose the correct link for your device from the options provided: either Windows installer (64-bit) or

Windows installer (32-bit) and proceed to download the executable file.

Step 3: Downloading the Python Installer

Once you have downloaded the installer, open the .exe file, such as python-3.12.3-amd64.exe, by doubleclicking it to launch the Python installer. Choose the option to install the launcher for all users by checking the corresponding checkbox, so that all users of the computer can access the Python launcher application.

Enable users to run Python from the command line by checking the "Add python.exe to PATH" checkbox.

After Clicking the "Install Now" Button the setup will start installing Python on your Windows system.

Step 4: Verify the Python Installation in Windows

After successful installation of Python, close the installation window. You can check if the installation of Python was successful by using either the command line or the Integrated Development Environment (IDLE), which you may have installed. To access the command line, click on the Start menu and type "cmd" in the search bar.

Then click on Command Prompt, type the command "python --V" or "python --version". You can see installed version of Python on your system.

Go to Python Integrated Development Environment (IDLE). In Windows search bar, type IDLE and you can see "IDLE (Python 3.12.3- bit)". Open IDLE on the IDLE screen itself you can see version. This gives the conformation of successful installation of python.

Installation of essential packages Numpy and Pandas.

a) Install numpy package.

NumPy is an open-source Python library that facilitates efficient numerical operations on large quantities of data. There are a few functions that exist in NumPy that we use on pandas DataFrames. The most important part about NumPy is that pandas is built on top of it which means Numpy is required for operating the Pandas.

It is defined as a Python package used for performing the various numerical computations and processing of the multidimensional and single-dimensional array elements. The calculations using Numpy arrays are faster than the normal Python array. It is also capable of handling a vast amount of data and convenient with Matrix multiplication and data reshaping. Steps to install Numpy is,

Step 1: Open command prompt, CMD.

Step 2: Type the command,

C:\Users\Admin> py –m pip install numpy

Or

C:\Users\Admin>pip3 install numpy

Step 3: Upgrade the software by the command

C:\Users\Admin> py –m pip install - -upgrade pip

Or

C:\Users\Admin> pip3 install - -upgrade pip

b) Install pandas package.

Pandas is a very popular library for working with data (its goal is to be the most powerful and flexible opensource tool, and in our opinion, it has reached that goal). DataFrames are at the center of pandas. A DataFrame is structured like a table or spreadsheet. The rows and the columns both have indexes, and you can perform operations on rows or columns separately. It can perform five significant steps required for processing and analysis of data irrespective of the origin of the data, i.e., load, manipulate, prepare, model, and analyze.

Steps to install pandas is,

Step 1: Open command prompt, CMD.

Step 2: Type the command,

C:\Users\Admin> py –m pip install pandas

Or

C:\Users\Admin>pip3 install pandas

Step 3: Upgrade the software by the command

C:\Users\Admin> py –m pip install - -upgrade pip

Or

C:\Users\Admin> pip3 install - -upgrade pip

Upon successful installation, the version installed can be verified using the below program.

```
File  Edit  Format  Run  Options  Window  Help
import numpy
import pandas

print("Numpy version: ")
print(numpy.__version__)

print("Pandas version: ")
print(pandas.__version__)
```

Output:

```
Type "help", "copyright", "cr
>>>
=== RESTART: C:/Users/Admin/A
Numpy version:
1.24.1
Pandas version:
1.5.3
>>>
```

Labcycle 2. Introduce scikit-learn as a machine learning library

Scikit-learn is a widely-used Python library that provides a comprehensive suite of tools and functionalities for machine learning tasks.

1. **Versatility**: Scikit-learn offers a wide range of tools and functionalities for various machine learning tasks, including but not limited to:

• Supervised Learning: Classification, Regression. Almost all the popular supervised learning algorithms, like Linear Regression, Support Vector Machine (SVM), Decision Tree etc., are the part of scikit-learn

• Unsupervised Learning: Clustering, Dimensionality Reduction. It supports all the popular unsupervised learning algorithms from clustering, factor analysis, PCA (Principal Component Analysis) to unsupervised neural networks

• Model Selection and Evaluation: Cross-validation, Hyperparameter Tuning

• Preprocessing: Data cleaning, Feature Engineering

2. Consistent Interface: It provides a consistent and user-friendly API, making it easy to experiment with different algorithms and techniques without needing to learn new syntax for each.

3. Integration with Other Libraries: Scikit-learn seamlessly integrates with other Python libraries like NumPy, pandas, and Matplotlib, allowing smooth data manipulation, preprocessing, and visualization.

4. Ease of Learning: Its well-documented and straightforward interface makes it suitable for both beginners and experienced machine learning practitioners. It's often recommended for educational purposes due to its simplicity.

5. Performance and Scalability: While focusing on simplicity, scikit-learn also emphasizes performance. It's optimized for efficiency and scalability, making it suitable for handling large datasets and complex models.

6. Community and Development: As an open-source project, scikit-learn benefits from a vibrant community of developers and contributors. Regular updates, bug fixes, and enhancements ensure it stays relevant and up-to-date with the latest advancements in machine learning.

7. Application in Industry and Academia: Scikit-learn's robustness and ease of use have made it a go-to choose in various domains, including finance, healthcare, natural language processing, and more. It's widely used in research and production environments.

To access the command line, click on the Start menu and type "cmd" in the search bar. To install the scikit-learn module type the below command:

```
C:\Users\Admin>py -V
Python 3.10.11

C:\Users\Admin>py -m pip install scikit-learn
```

Output: dir command can be used to find out all the methods supported by sklearn

```
>>> import sklearn
>>> print(dir(sklearn))
['__SKLEARN_SETUP__', '__all__', '__builtins__', '__cached__', '__check_build',
'__doc__', '__file__', '__loader__', '__name__', '__package__', '__path__', '__s
pec__', '__version__', '_config', '_distributor_init', 'base', 'clone', 'config_
context', 'exceptions', 'externals', 'get_config', 'logger', 'logging', 'os', 'r
andom', 'set_config', 'setup_module', 'show_versions', 'sys', 'utils']
>>>
```

Example: linear_example.py

```
from sklearn.linear_model import LinearRegression

# Sample dataset

X_train = [[1], [2], [3], [4], [5]]  # Features

y_train = [2,  4,  6,  8,  10]  # Labels
```

 # Initialize the linear regression model

```
model = LinearRegression()
```

 # Train the model on the training data

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

 # Predict using the trained model

```
X_test = [[8], [9], [10]]  # New data to predict
predictions = model.predict(X_test)
```

 # Print predictions

```
print("Predictions:", predictions)
```

Output:

```
>>>
= RESTART: C:\Users\Admin\Downloads\supervisedandunsupervisedlearning\supervised
.py
Predictions: [16. 18. 20.]
>>>
```

Labcycle 3: Install and set up scikit-learn and other necessary tools.

- Scikit-learn (Sklearn) is the most useful and robust library for machine learning in Python.
- It provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction via a consistence interface in Python.
- This library, which is largely written in Python, is built upon NumPy, pandas,SciPy and Matplotlib.

Pip upgrade:

Using pip:

• Pip is Python's package manager.

• It usually comes installed with Python.

• Open a terminal/command prompt.

Py –m install pip - -upgrade pip

Procedure to install scikit-learn and other libraries:

a)Install NumPy :-

pip install numpy

press Enter. This command will download and install **NumPy**.

b)Install pandas:-

pip install pandas

press Enter. This command will download and install **pandas**.

c)Install matplotlib:-

pip install matplotlib

press Enter. This command will download and install **matplotlib**.

d)Install scipy:-

pip install scipy

press Enter. This command will download and install scipy

e)Install scikit-learn(sklearn):-

pip install scikit-learn

press Enter. This command will download and install scikit-learn

Verify installations:

Open a IDLE:

```
import numpy
import pandas
import scipy
import sklearn
import matplotlib

print("Numpy version: ")
print(numpy.__version__)
print("Numpy library is successfully installed ")
print("Pandas version: ")
print(pandas.__version__)
print("Pandas library is successfully installed ")
print("scipy version: ")
print(scipy.__version__)
print("scipy library is successfully installed ")
print("sklearn version: ")
print(sklearn.__version__)
```

```
print("sklearn library is successfully installed ")
print("matplotlib version: ")
print(matplotlib.__version__)
print("matplotlib library is successfully installed ")
```

Output:

```
Numpy version:
1.26.3
Numpy library is successfully installed
Pandas version:
2.2.0
Pandas library is successfully installed
scipy version:
1.12.0
scipy library is successfully installed
sklearn version:
1.4.0
sklearn library is successfully installed
matplotlib version:
3.8.2
matplotlib library is successfully installed
```

4. Write a program to Load and explore the dataset of .CVS and excel files using pandas.

```
import pandas as pd
def explore_dataset(file_path):
    if file_path.endswith('.csv'):
        df = pd.read_csv("C:/Prabha/BCA/Machine Learning/ML_LAB/iris.csv")
    elif file_path.endswith('.xlsx'):
        df = pd.read_excel("C:/Prabha/BCA/Machine Learning/ML_LAB/iris.xlsx")
    else:
        print("Unsupported file format.")
        return
    print("Dataset information:")
    print(df.info())
    print("\n first few rows of the data are")
    print(df.head())
    print("Summary statistics:")
    print(df.describe())
```

```
    print("Unique values are: ")
    for column in df.select_dtypes(include='object').columns:
        print(f"{column}: {df[column].unique()}")

file_path = 'iris.csv'
explore_dataset("F:\XXXXX\iris.csv")
```

Output:

Dataset information:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             150 non-null    int64
 1   SepalLengthCm  150 non-null    float64
 2   SepalWidthCm   150 non-null    float64
 3   PetalLengthCm  150 non-null    float64
 4   PetalWidthCm   150 non-null    float64
 5   Species        150 non-null    object
dtypes: float64(4), int64(1), object(1)
memory usage: 7.2+ KB
None
```

first few rows of the data are

	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

Summary statistics:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	75.500000	5.843333	3.054000	3.758667	1.198667
std	43.445368	0.828066	0.433594	1.764420	0.763161
min	1.000000	4.300000	2.000000	1.000000	0.100000
25%	38.250000	5.100000	2.800000	1.600000	0.300000
50%	75.500000	5.800000	3.000000	4.350000	1.300000
75%	112.750000	6.400000	3.300000	5.100000	1.800000
max	150.000000	7.900000	4.400000	6.900000	2.500000

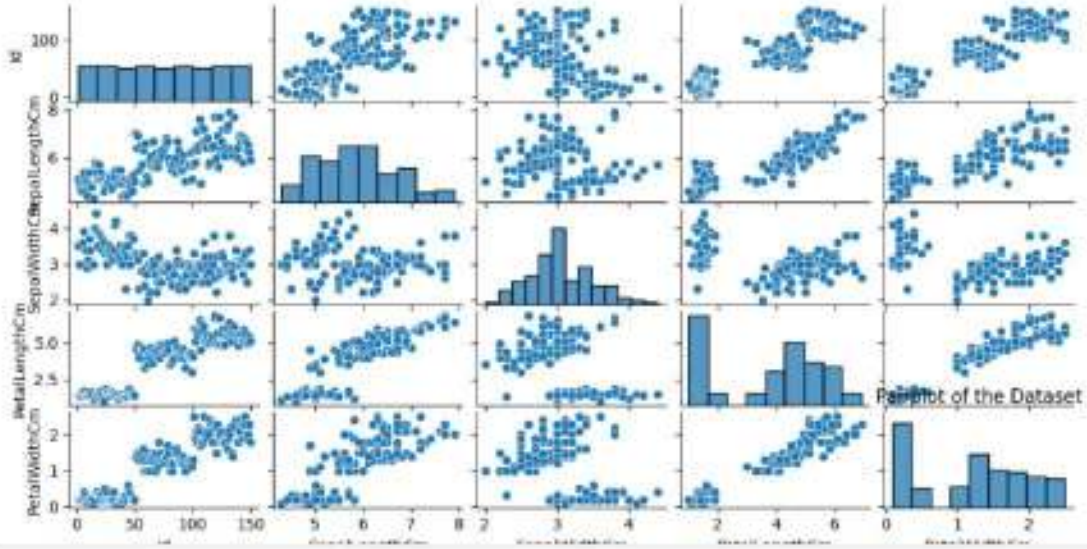
Unique values are:

```
Species: ['Iris-setosa' 'Iris-versicolor' 'Iris-virginica']
```

5. Write a program to Visualize the dataset to gain insights using Matplotlib or Seaborn by plotting scatter plots, bar charts

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

def visualize_dataset(file_path):
    df = pd.read_csv("C:/Prabha/BCA/Machine Learning/ML_LAB/iris.csv")
    sns.pairplot(df)
    plt.title("Pairplot of the Dataset")
    plt.show()
    if df.iloc[:, 0].dtype =='object':
        sns.countplot(x=df.columns[0], data=df)
        plt.title("Bar chart of categorical column")
        plt.xlabel(df.columns[0])
        plt.ylabel("count")
        plt.show()
    else:
        print("No categorical column found to plot bar chart")

file_path = 'iris.csv'
visualize_dataset("C:/Prabha/BCA/Machine Learning/ML_LAB/iris.csv")
```

Output:



6. Write a program to Handle missing data, encode categorical variables, and perform feature scaling

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder, StandardScaler

# Load Iris dataset
iris = load_iris()
iris_df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
iris_df['target'] = iris.target

def preprocess_dataset(df):
 df.iloc[::10, 0] = float('NaN')
 imputer = SimpleImputer(strategy='mean')
 df[df.columns] = imputer.fit_transform(df[df.columns])
 scaler = StandardScaler()
 df[df.columns[:-1]] = scaler.fit_transform(df[df.columns[:-1]])
 return df

preprocessed_df = preprocess_dataset(iris_df)
print("Preprocessed dataset:")
print(preprocessed_df.head())
```

Output:

```
Preprocessed dataset:
   sepal length (cm)  sepal width (cm)  ...  petal width (cm)  target
0       1.132764e-15          1.019004  ...         -1.315444     0.0
1      -1.196022e+00         -0.131979  ...         -1.315444     0.0
2      -1.451098e+00          0.328414  ...         -1.315444     0.0
3      -1.578636e+00          0.098217  ...         -1.315444     0.0
4      -1.068484e+00          1.249201  ...         -1.315444     0.0

[5 rows x 5 columns]
```

7. Write a program to implement a k-Nearest Neighbours (k-NN) classifier using scikit-learn and Train the classifier on the dataset and evaluate its performance.

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report
```

```
# Load Iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,random_state=42)
# Initialize the k-NN classifier
k = 3 # Number of neighbors
knn_classifier = KNeighborsClassifier(n_neighbors=k)

# Train the classifier
knn_classifier.fit(X_train, y_train)
# Make predictions on the testing set
y_pred = knn_classifier.predict(X_test)
# Evaluate the classifier's performance
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
# Display classification report
print("Classification Report:")
print(classification_report(y_test, y_pred, target_names=iris.target_names))
```

Output:

```
Accuracy: 1.0
Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        10
  versicolor       1.00      1.00      1.00         9
   virginica       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

8. Write a program to implement a linear regression model for regression tasks

and Train the model on a dataset with continuous target variables

```
import numpy as np
import pandas as pd
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
```

```
# Load California Housing dataset
california = fetch_california_housing()
X = california.data
y = california.target

# Convert the data to a pandas DataFrame for easier manipulation
california_df = pd.DataFrame(data=X, columns=california.feature_names)
california_df['target'] = y

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Initialize Linear Regression model
linear_regression = LinearRegression()

# Train the model
linear_regression.fit(X_train, y_train)

# Make predictions on the testing set
y_pred = linear_regression.predict(X_test)

# Evaluate the model's performance
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print("Mean Squared Error:", mse)
print("R-squared Score:", r2)
```

Output:

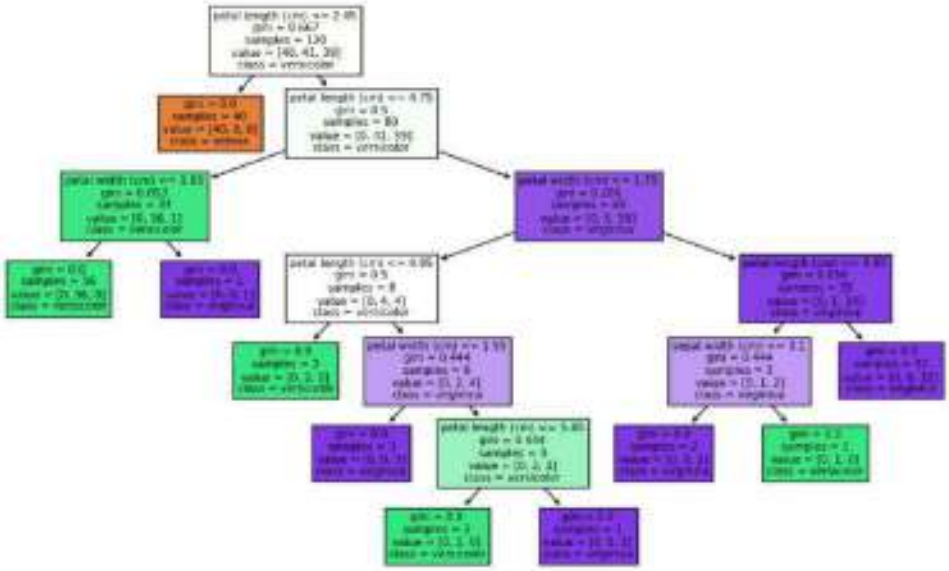
Mean Squared Error: 0.5558915986952425
R-squared Score: 0.5757877060324521

9. Write a program to implement a decision tree classifier using scikit-learn and visualize the decision tree and understand its splits.

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree
import matplotlib.pyplot as plt

# Load Iris dataset
```

```
iris = load_iris()
X = iris.data
y = iris.target

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize Decision Tree classifier
decision_tree = DecisionTreeClassifier()

# Train the classifier
decision_tree.fit(X_train, y_train)

# Visualize the decision tree
plt.figure(figsize=(12, 8))
plot_tree(decision_tree, feature_names=iris.feature_names,
class_names=iris.target_names, filled=True)
plt.show()
```

Output:

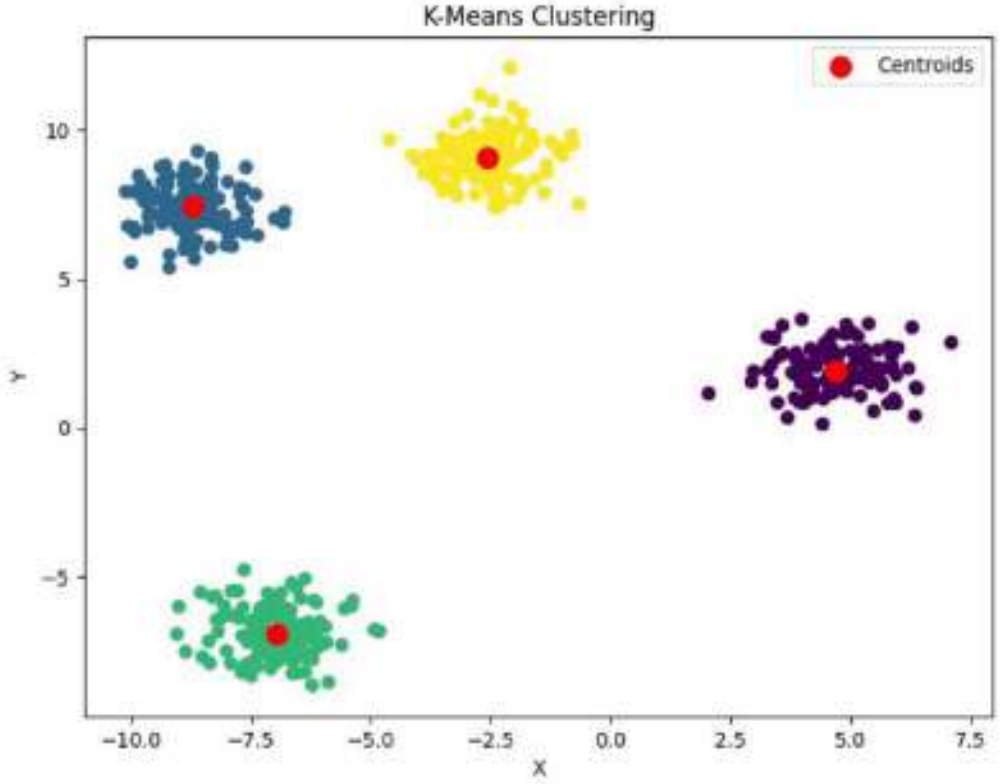


10. Write a program to Implement K-Means clustering and Visualize clusters.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
```

```
# Generate sample data
X, y = make_blobs(n_samples=500, centers=4, cluster_std=0.8, random_state=42)

# Create a K-Means clusterer with 4 clusters
kmeans = KMeans(n_clusters=4, random_state=42)

# Fit the data
kmeans.fit(X)

# Get cluster labels
labels = kmeans.labels_

# Plot the data with cluster labels
plt.figure(figsize=(8, 6))
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s=100,
c='red', label='Centroids')
plt.title('K-Means Clustering')
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
plt.show()
```

Output:


K-Means Clustering
Centroids
10
5
0
−5
Y
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
X