**Lab Experiment - 1**

1. **A study was conducted to understand the effect of number of hours the students spend studying on their performance in the final exam. Write a code to plot line chart with number of hours studying on X Axis and score in final exam on the Y Axis. Use a red star as the point character; label the axis and give the plot a title.**

**Program:**

import matplotlib.pyplot as plt

hours = [1,2,3,4,5,6,7,8,9,10]

scores = [50,54,60,54,70,52,90,100,80,60]

plt.plot(hours,scores,marker = '\*', color = 'red', linestyle = '-', markersize =10)

plt.xlabel('Number of Hours spent Studing')

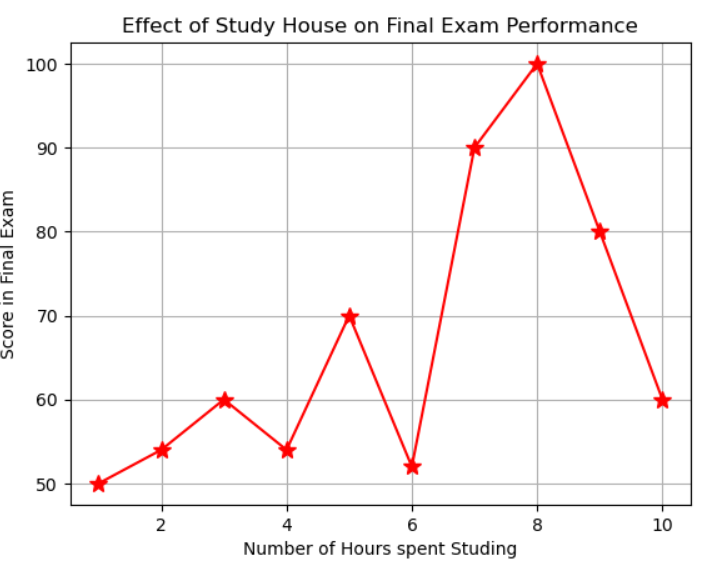
plt.ylabel('Score in Final Exam')

plt.title('Effect of Study House on Final Exam Performance')

plt.grid(True)

plt.show()

**Output:**



**Lab Experiment - 2**

1. **For the given dataset 'mtcars.csv' plot a histogram to check the frequency distribution of the variable 'mpg'.**

**Program:**

import pandas as pd

import matplotlib.pyplot as plt

data **=** pd**.**read\_csv('mtcars.csv')

plt**.**hist(data['mpg'], color **=** 'lightblue', edgecolor **=** 'black')

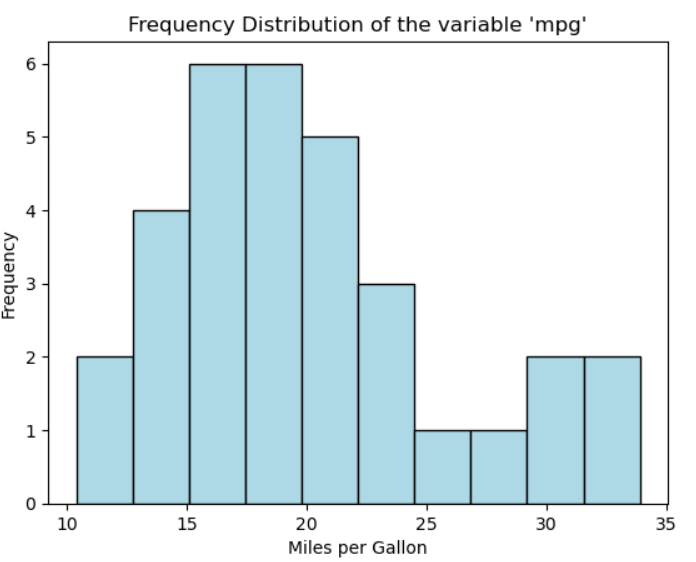
plt**.**xlabel('Miles per Gallon')

plt**.**ylabel('Frequency')

plt**.**title("Frequency Distribution of the variable 'mpg'")

plt**.**show()

**Output:**



**Lab Experiment - 3**

1. **Consider the books dataset BL-Flickr-Images-Book.csv from Kaggle which contains information about books. Write a program to demonstrate the following:  
   Program:**
   1. **Importing the data into a DataFrame**
   2. **Find and drop the columns which are irrelevant for the book information**
   3. **Change the index of the DataFrame**
   4. **Tidy up the fields in the data such as date of publication with the help of simple regular experession**
   5. **Combine str methods with NumPy to clean columns**

**Program:**

import pandas as pd

import numpy as np

# Importing the data into a DataFrame

df = pd.read\_csv("BL-Flickr-Images-Book.csv")

# Find and drop the columns which are irrelevant for the book information

icols = ['Edition Statement','Contributors','Corporate Author','Corporate Contributors','Former owner','Engraver','Flickr URL','Shelfmarks','Issuance type']

df.drop(columns = icols, inplace = True)

# Change the index of the DataFrame

df.set\_index('Identifier', inplace=True)

# Tidy up the fields in the data such as date of publication with the help of simple regular experession

df['Date of Publication'] = df['Date of Publication'].str.extract(r'(\d{4})')

# Combine str methods with NumPy to clean columns

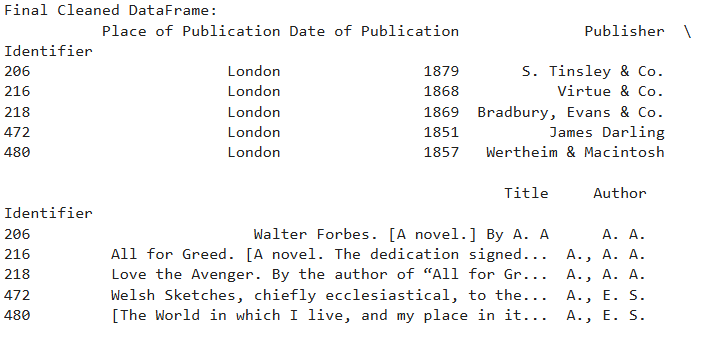
df['Place of Publication'] = np.where(df['Place of Publication'].str.contains('London'), 'London', df['Place of Publication'])

df['Place of Publication'] = np.where(df['Place of Publication'].str.contains('Oxford'), 'Oxford', df['Place of Publication'])

print("\nFinal Cleaned DataFrame:")

print(df.head())

**Output:**



**Lab Experiment - 4**

1. **Implement K-means clustering using MapReduce.**

**Program:**

import numpy as np

def initialize\_centroids(data, k):

indices = np.random.choice(len(data), size=k, replace=False)

return data[indices]

def assign\_clusters(data, centroids):

clusters = [[] for \_ in range(len(centroids))]

for point in data:

distances = np.linalg.norm(point - centroids, axis=1)

closest\_centroid = np.argmin(distances)

clusters[closest\_centroid].append(point)

return clusters

def update\_centroids(clusters):

new\_centroids = []

for cluster in clusters:

if cluster:

new\_centroids.append(np.mean(cluster, axis=0))

return np.array(new\_centroids)

def kmeans(data, k, max\_iters=100):

centroids = initialize\_centroids(data, k)

for \_ in range(max\_iters):

clusters = assign\_clusters(data, centroids) # Map step

new\_centroids = update\_centroids(clusters) # Reduce step

if np.allclose(centroids, new\_centroids, atol=1e-6):

break

centroids = new\_centroids

return centroids, clusters

if \_\_name\_\_ == "\_\_main\_\_":

np.random.seed(42)

data = np.random.rand(100, 2)

k = 3

centroids, clusters = kmeans(data, k)

print("Final centroids:")

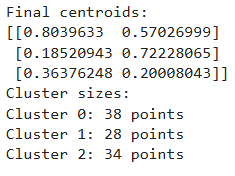
print(centroids)

print("Cluster sizes:")

for i, cluster in enumerate(clusters):

print(f"Cluster {i}: {len(cluster)} points")

**Output:**



**Lab Experiment - 5**

1. **Give a MapReduce Program to calculate the frequency of a given word in a given file.**

**Program:**

with open('lorem.txt', 'r') as f:

text = f.read()

target\_word = "lorem"

def mapper(line):

words = line.strip().split()

return [(word, 1) for word in words if word == target\_word]

def reducer(counts):

total = 0

for count in counts:

total += count

return total

mapped = [pair for line in text.split('\n') for pair in mapper(line)]

counts = [count for word, count in mapped]

frequency = reducer(counts)

print(f"The word '{target\_word}' appears {frequency} time(s) in the given text.")

**Output:**



**Lab Experiment - 6**

1. **Develop a Map Reduce program to calculate the maximum, minimum, and mean values of sepal length, sepal width, petal length, and petal width from the iris flower dataset. Visualize the relationship between sepal length and sepal width and also between petal length and petal width.**

**Program:**

import numpy as np

from sklearn.datasets import load\_iris

import matplotlib.pyplot as plt

iris = load\_iris()

X = iris.data

def map\_function(data):

return {

'sepal\_length': data[0],

'sepal\_width': data[1],

'petal\_length': data[2],

'petal\_width': data[3]

}

def reduced\_function(results):

stats = {}

for feature in ['sepal\_length', 'sepal\_width', 'petal\_length', 'petal\_width']:

values = [result[feature] for result in results]

stats[feature] = {

'min': np.min(values),

'max': np.max(values),

'mean': np.mean(values)

}

return stats

mapped\_results = [map\_function(data) for data in X]

min\_max\_mean\_values = reduced\_function(mapped\_results)

for feature, values in min\_max\_mean\_values.items():

print(f"{feature}: Min= {values['min']}, Max= {values['max']}, Mean= {values['mean']}")

plt.figure(figsize=(12,5))

plt.subplot(1,2,1)

plt.scatter(X[:,0], X[:, 1], c='blue', label='Iris data')

plt.grid(True)

plt.title("Sepal Length vs Sepal Width")

plt.xlabel("Sepal Length")

plt.ylabel("Sepal Width")

plt.legend()

plt.subplot(1,2,2)

plt.scatter(X[:,2], X[:, 3], c='red', label='Iris data')

plt.grid(True)

plt.title("Petal Length vs Petal Width")

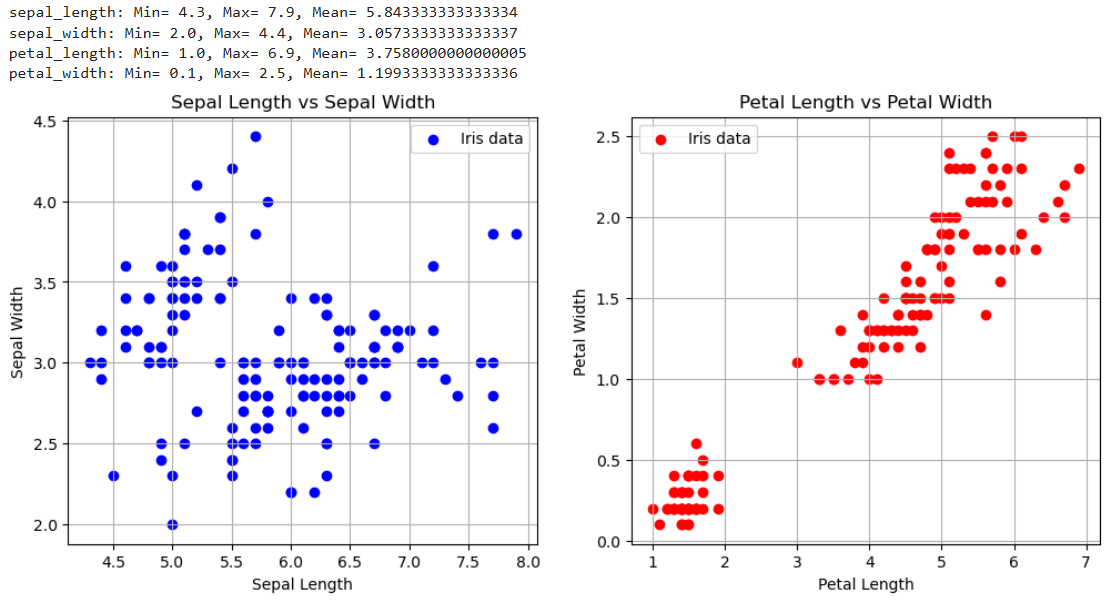
plt.xlabel("Petal Length")

plt.ylabel("Petal Width")

plt.legend()

plt.show()

**Output:**

****

**Lab Experiment - 7**

1. **Implement K-Means and Hierarchical Clustering technique using Spark.**

**Program:**

from pyspark.sql import SparkSession

from pyspark.ml.clustering import KMeans,BisectingKMeans

from pyspark.ml.evaluation import ClusteringEvaluator

spark = SparkSession.builder.appName("KMeansExample").getOrCreate()

dataset = spark.read.format("libsvm").load("kmeans\_data.txt")

kmeans = KMeans().setK(2).setSeed(1)

model = kmeans.fit(dataset)

predictions = model.transform(dataset)

predictions.show()

evaluator = ClusteringEvaluator()

silhouette = evaluator.evaluate(predictions)

print(f"Silhouette with squared euclidean distance = {silhouette}")

centers = model.clusterCenters()

print("Cluster Centers: ")

for center in centers:

print(center)

bkm = BisectingKMeans().setK(2).setSeed(1)

model = bkm.fit(dataset)

predictions = model.transform(dataset)

predictions.show()

evaluator = ClusteringEvaluator()

silhouette = evaluator.evaluate(predictions)

print(f"Silhouette with squared euclidean distance = {silhouette}")

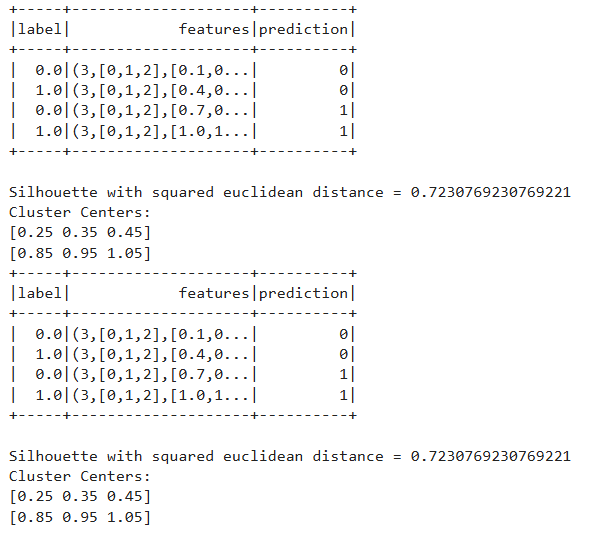
centers = model.clusterCenters()

print("Cluster Centers: ")

for center in centers:

print(center)

**Output:**



**Lab Experiment - 8**

1. **Develop a Java application to find the maximum temperature using Spark.**

**Program:**

import org.apache.spark.api.java.JavaRDD;

import org.apache.spark.api.java.JavaSparkContext;

import org.apache.spark.api.java.function.Function;

import org.apache.spark.SparkConf;

import org.apache.log4j.Level;

import org.apache.log4j.Logger;

public class MaxTemperature {

public static void main(String[] args) {

Logger.getLogger("org.apache.spark").setLevel(Level.WARN);

SparkConf conf = new SparkConf().setAppName("Max Temperature").setMaster("local");

JavaSparkContext sc = new JavaSparkContext(conf);

String inputFile = args[0];

JavaRDD<String> lines = sc.textFile(inputFile);

JavaRDD<Double> temperatures = lines.map(new Function<String, Double>() {

@Override

public Double call(String line) {

String[] parts = line.split(",");

return Double.parseDouble(parts[1]);

}

});

Double maxTemperature = temperatures.reduce((a, b) -> Math.max(a, b));

System.out.println("Maximum Temperature: " + maxTemperature);

sc.stop();

}

}

**Output:**

