|  |  |
| --- | --- |
| https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcQdAoy8HAYiGwgqBMeq0EtfxKBc5i9totR_W7yOXHEosxAvJ3u6k6r7MVNRo_M3U1QAJMc&usqp=CAU | **E.G.S. Pillay Engineering College**  An Autonomous Institution Affiliated to Anna University, Chennai | Approved by AICTE, New Delhi  Accredited by NBA T1 (B.E. – CIVIL, CSE, ECE, EEE, MECH & B.Tech – IT)  Accredited by NAAC with A++ Grade | One among Top 300 Engineering Institutions in India (NIRF-24)  Old Nagore Road, Thethi, Nagore Village, Nagapattinam – 611002, Tamil Nadu, India |

**Department of Artificial Intelligence & Data Science**

Academic Year: 2025-26 | Odd Semester | 05th Semester

**2302AS551 - PROGRAMMING WITH LARGE DATASETS LABORATORY**

**LAB MANUAL**

|  |  |
| --- | --- |
| **LIST OF EXPERIMENTS:** | |
| 1. Write a map reduce program to compute descriptive statistics such as mean, median, mode, standard deviation from a large dataset. |
| 1. Write a map-reduce program to compute box-plots and histograms of all the numerical variables in a large multi-variate dataset. |
| 1. Write a map-reduce program to compute correlation metrics between pairs of all the numerical variables in a large multi-variate dataset. |
| 1. Write a map-reduce program to perform clustering of a large multi-variate dataset. Measure the runtime and study its scaling behaviour as more nodes are added to the cluster. |
| 1. Write a map-reduce program to perform classification of a large multi-variate dataset into two or more classes. |
| 1. Write a spark program to compute box-plots and histograms of all the numerical variables in a large dataset. |
| 1. Write a spark program to perform classification in a large dataset. Measure the runtime and study its scaling behaviour as more nodes are added to the cluster. |
| 1. Write a spark program to perform regression in a large dataset. Measure the runtime and study its scaling behaviour as more nodes are added to the cluster. |

# COMPUTE THE PRODUCT OF LIST ELEMENTS AND FILTER EVEN NUMBERS AIM:

To implement a map reduce program to compute the product of list elements and filter even numbers for the given list.

# ALGORITHM:

1. Start the program
2. Define a function to double the numbers and define a list of numbers
3. Perform map () function to double each element in the list
4. Import the functools module
5. Use the reduce() and lambda function to compute the product of list elements
6. Use the filter() function to filter out even numbers
7. Convert the result to a list and print it
8. Stop the program

# PROGRAM

def double(n): return n\*2

numbers = [5,6,7,8]

result = map(double,numbers) print(list(result))

import functools

# define a list of numbers numbers = [1, 2, 3, 4]

# use reduce to compute the product of list elements product = functools.reduce(lambda x, y: x \* y, numbers) print("Product of list numbers:", product)

# define a function to check if a number is even defis\_even(n):

return n%2==0 #define a list of numbers

numbers =[1,2,3,4,5,6,7,8,9,10,11,12]

# use filter to filter out even numbers even\_numbers = filter(is\_even,numbers) print("Even numbers:",list(even\_numbers))

# OUTPUT :

[10, 12, 14, 16]

Product of list numbers: 24

Even numbers: [2, 4, 6, 8, 10, 12]

# COUNT WORD FREQUENCY IN TEXT DATA

**AIM**

To implement a Python program that uses the MapReduce approach to count the frequency of each word in a given text data.

# ALGORITHM

1. Start the program
2. Prepare a list of text lines to be processed
3. Define map function to split the line into individual words
4. Define reduce function to initialize a dictionary to store the word count
5. Accumulate and return the final word counts for each unique word
6. Print the word frequency by iterating through the final dictionary
7. Stop the program

# PROGRAM

from collections import default dict # Sample text data

text\_data = [ "Hello world",

"Hello from the other side", "Hello there",

"Hello world from ChatGPT",

]

# Map function defmap\_function(line):

# Splitting the line into words words = line.split()

# Emitting a key-value pair (word, 1) for each word return [(word, 1) for word in words]

# Reduce function defreduce\_function(mapped\_data):

# Using a dictionary to accumulate word counts word\_count = defaultdict(int)

for word, count in mapped\_data: word\_count[word] += count returnword\_count

# Step 1: Map step mapped\_data = [] for line in text\_data:

mapped\_data.extend(map\_function(line))

# Step 2: Reduce step

word\_count = reduce\_function(mapped\_data) # Output the word count

print("Word Count:")

for word, count in word\_count.items(): print(f"{word}: {count}")

# OUTPUT :

Word Count: Hello: 4

world: 2

from: 2

the: 1

other: 1

side: 1

there: 1

ChatGPT: 1

# SUM OF NUMBERS

**AIM**:

To implement a MapReduce program in Python that calculates the sum of numbers in a given list by using map and reduce functions.

# ALGORITHM:

1. Start the program
2. Initialize a map function that takes a number as input
3. Initialize a reduce function that takes the mapped data as input
4. Iterate through each key-value pair in the mapped data
5. If the key is "number," add the corresponding value to the assigned variable
6. For each number in the input list, call the map function and add the result
7. Call the reduce function to calculate the sum
8. Print the total sum obtained from the reduce step
9. Stop the program

# PROGRAM

# Sample list of numbers numbers = [10, 20, 30, 40, 50]

# Map function defmap\_function(number):

# Emit a key-value pair ("number", number) for each number return [("number", number)]

# Reduce function defreduce\_function(mapped\_data):

# Initialize a counter for the sum of numbers total\_sum = 0

for key, value in mapped\_data:

if key == "number":

total\_sum += value returntotal\_sum

# Step 1: Map step mapped\_data = []

for number in numbers: mapped\_data.extend(map\_function(number)) # Step 2: Reduce step

total\_sum = reduce\_function(mapped\_data) # Output the total sum

print(f"Total Sum: {total\_sum}")

# OUTPUT :

Total Sum: 150

# Count Lines in Text Data

**AIM**

To implement a MapReduce program in Python that counts the total number of lines in a given set of text data.

# ALGORITHM

1. Start the program
2. Create a list where each element represents a line of text
3. Define the map function that takes a line as input and returns a key-value pair
4. Define the reduce function that takes the mapped data as input
5. Iterate through each key-value pair in the mapped data
6. If the key is "line", add the value to linecount
7. Apply the reduce function to calculate the total line count
8. Print the line count
9. Stop the program

# PROGRAM

from collections import default dict

# Sample text data (each item represents a line) text\_data = [

"Hello world",

"Hello from the other side", "Hello there",

"Hello world from ChatGPT",

]

# Map function defmap\_function(line):

# Emit a key-value pair ("line", 1) for each line return [("line", 1)]

# Reduce function defreduce\_function(mapped\_data):

# Initialize a counter for the number of lines line\_count = 0

for key, count in mapped\_data: if key == "line":

line\_count += count

returnline\_count

# Step 1: Map step mapped\_data = [] for line in text\_data:

mapped\_data.extend(map\_function(line)) # Step 2: Reduce step

line\_count = reduce\_function(mapped\_data) # Output the line count

print(f"Line Count: {line\_count}")

# OUTPUT :

Line Count: 4

# COMPUTE DESCRIPTIVE STATISTICS

**AIM**

To write a MapReduce program in Python to compute descriptive statistics (mean, median, mode, variance, and standard deviation) for a given dataset of numbers.

# ALGORITHM

1. Start the program
2. Create a map function that emits key-value pairs to store the numbers for calculating statistics like mean, median, mode, etc
3. Create a reduce function that
   * Calculates the **mean** by dividing total\_sum by total\_count.
   * Calculates the **median** using the median() function on number\_list.
   * Determines the **mode** by finding the highest frequency in frequency.
   * Calculates the **variance** and **standard deviation** using variance() and stdev() functions.
4. Execute map and reduce phase
5. Print the computed values for mean, median, mode, variance, and standard deviation
6. Stop the program

# PROGRAM

from collections import default dict

from statistics import median, mode, variance, stdev

# Sample list of numbers

numbers = [10, 20, 20, 30, 40, 50, 50, 50, 60, 70]

# Map function defmap\_function(number):

# Emit key-value pairs for each number

return [("sum", number), ("count", 1), ("number", number)]

# Reduce function defreduce\_function(mapped\_data):

# Initialize variables for sum, count, and a list of numbers for median/mode total\_sum = 0

total\_count = 0 number\_list = []

frequency = defaultdict(int)

# Aggregate data

for key, value in mapped\_data:

if key == "sum":

total\_sum += value elif key == "count":

total\_count += 1

elif key == "number":

number\_list.append(value) frequency[value] += 1

# Calculate mean

mean\_value = total\_sum / total\_count if total\_count != 0 else 0 # Calculate median

median\_value = median(number\_list) if number\_list else None # Calculate mode

mode\_value = max(frequency, key=frequency.get) if frequency else None

# Calculate variance and standard deviation

variance\_value = variance(number\_list) if len(number\_list) > 1 else 0 stddev\_value = stdev(number\_list) if len(number\_list) > 1 else 0

returnmean\_value, median\_value, mode\_value, variance\_value, stddev\_value

# Step 1: Map step mapped\_data = []

for number in numbers: mapped\_data.extend(map\_function(number))

# Step 2: Reduce step

mean\_value, median\_value, mode\_value, variance\_value, stddev\_value = reduce\_function(mapped\_data)

# Output the results print(f"Mean: {mean\_value}")

print(f"Median: {median\_value}")

print(f"Mode: {mode\_value}") print(f"Variance: {variance\_value}") print(f"Standard Deviation: {stddev\_value}") **OUTPUT** :

Mean: 40.0

Median: 45.0

Mode: 50

Variance: 377.77777777777777

Standard Deviation: 19.436506316151

# BOX-PLOTS AND HISTOGRAM FOR NUMERICAL VARIABLES

**AIM:**

To write a map reduce program in python to compute box-plots and histograms of all the numerical variables in a large multi-variate dataset.

# ALGORITHM:

1. Start the program
2. Import required libraries such as, pandas, numpy, and matplotlib.pyplot for data manipulation and visualization.
3. Load the dataset using pandas.read\_csv().
4. Define mapper function for each numerical column in the dataset
5. Calculate and store the mean, standard deviation, minimum, maximum, first quartile (Q1), median, and third quartile (Q3) for each column.
6. Define reducer function to combine the mapped statistics into a single DataFrame
7. Transpose the resulting DataFrame to have each row represent a numerical column with its corresponding statistics.
8. For each numerical column, plot a boxplot and histogram to visually assess the distribution and outliers.
9. Call the mapper function to compute statistics for each numerical column.
10. Pass the mapped data to the reducer function to aggregate statistics.
11. Print the computed summary statistics.
12. Call the plotting function to display the visual representations of each column.
13. Stop the program

# PROGRAM

import pandas as pd import numpy as np

import matplotlib.pyplot as plt

# Generate a synthetic multivariate dataset np.random.seed(0)

data = pd.read\_csv(r"C:\Users\Student\Downloads\tested.csv") # Define MapReduce-like functions

def mapper(data):

"""Map step: Computes statistics for each numerical column.""" stats = {}

for column in data.select\_dtypes(include=[np.number]).columns: stats[column] = {

'mean': data[column].mean(),

'std': data[column].std(),

'min': data[column].min(),

'max': data[column].max(),

'q1': data[column].quantile(0.25), 'median': data[column].median(), 'q3': data[column].quantile(0.75),

}

return stats

def reducer(stats):

"""Reduce step: Combine statistics into a single DataFrame.""" returnpd.DataFrame(stats).T

defplot\_data(data):

"""Function to plot histograms and boxplots for each numerical column.""" for column in data.select\_dtypes(include=[np.number]).columns: plt.figure(figsize=(12, 5))

# Boxplot plt.subplot(1, 2, 1) plt.boxplot(data[column])

plt.title(f'Boxplot of {column}') # Histogram

plt.subplot(1, 2, 2)

plt.hist(data[column], bins=30, alpha=0.7) plt.title(f'Histogram of {column}')

plt.show()

# Main execution

if name == ' main ': # Step 1: Map

stats = mapper(data) # Step 2: Reduce result = reducer(stats)

# Print summary statistics print("Summary Statistics:") print(result)

# Step 3: Plot data plot\_data(data)

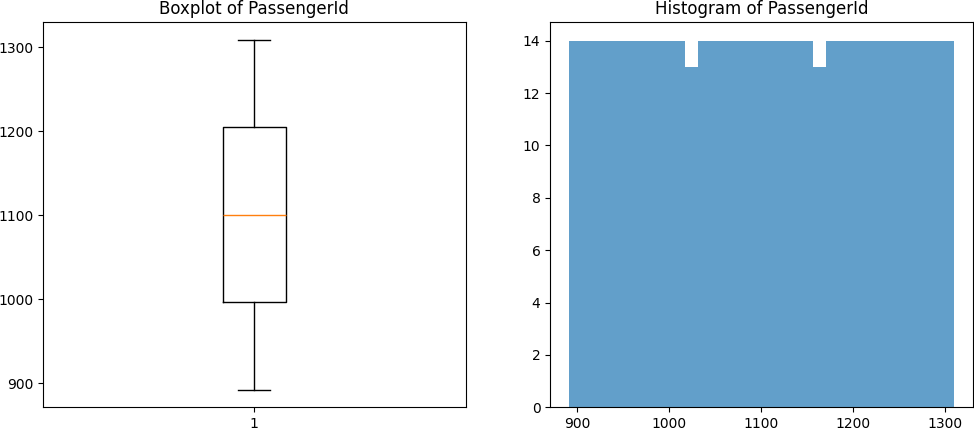
# OUTPUT :

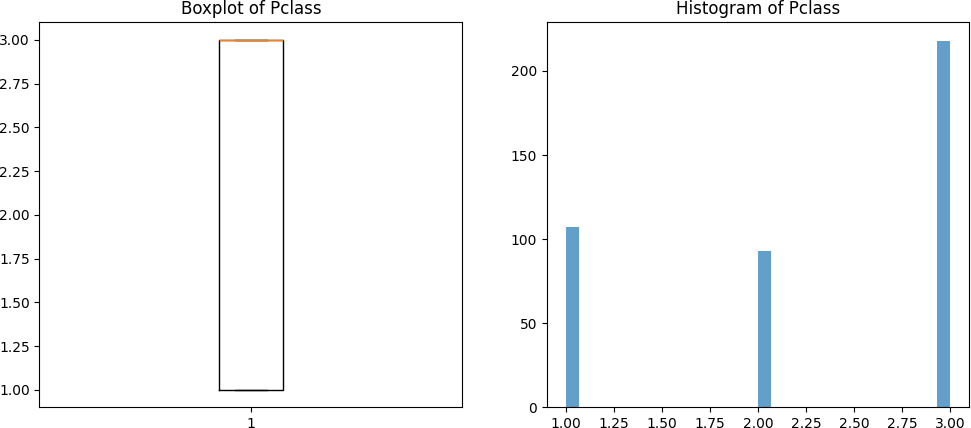
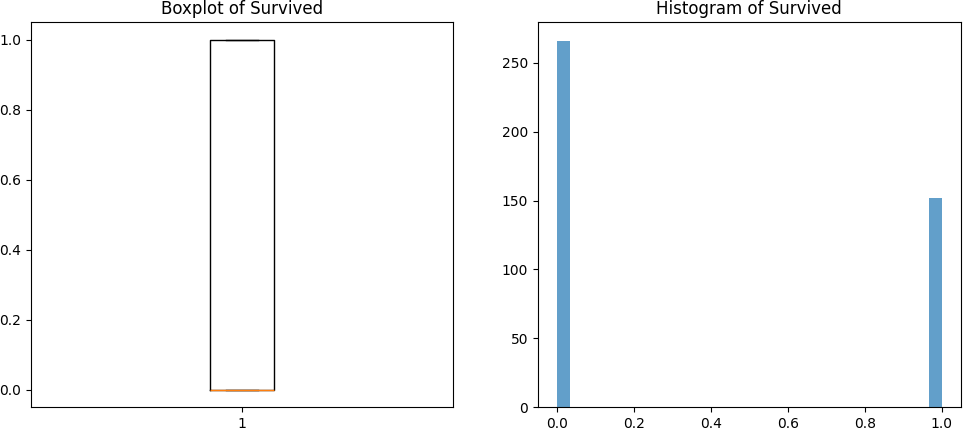
Summary Statistics:

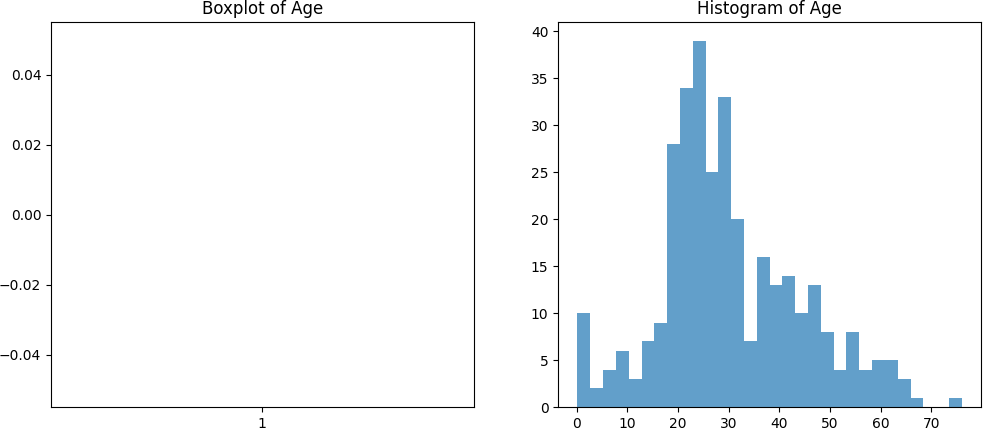
meanstd min max q1 median \

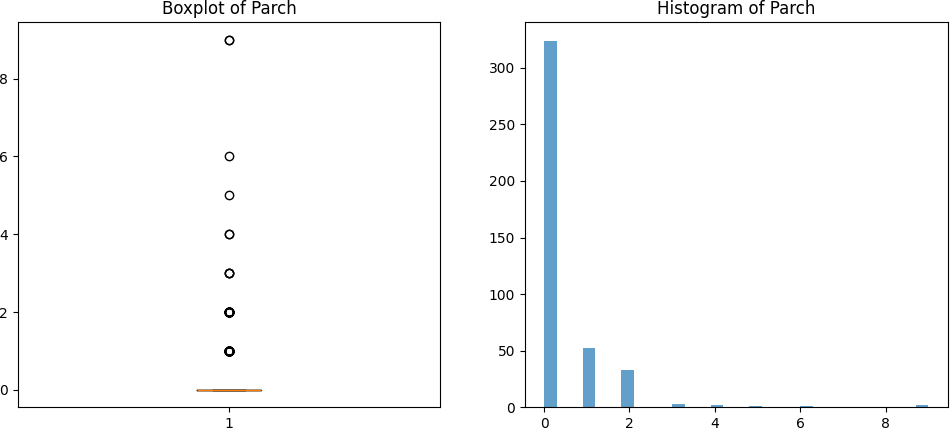
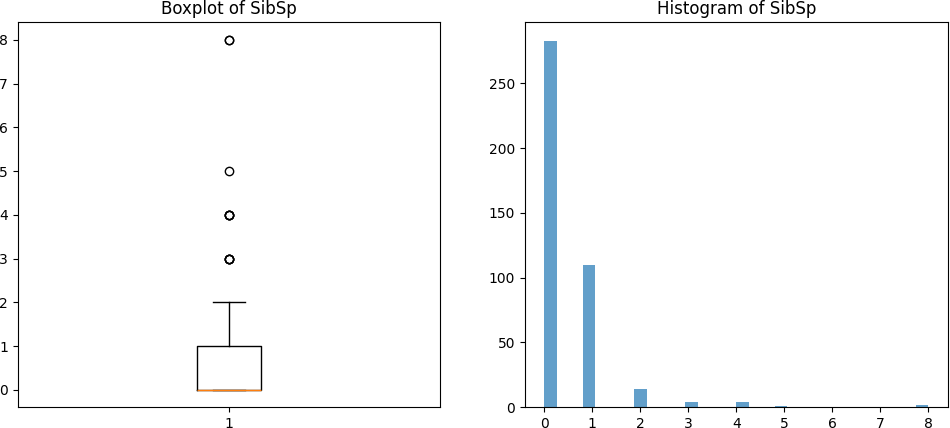
PassengerId 1100.500000 120.810458 892.00 1309.0000 996.2500 1100.5000

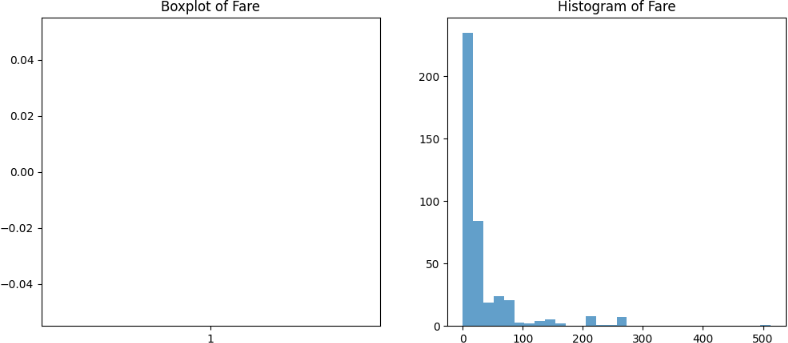
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Survived | 0.363636 0.481622 0.00 | | | 1.0000 0.0000 0.0000 | |
| Pclass | 2.265550 | 0.841838 | 1.00 | 3.0000 1.0000 | 3.0000 |
| Age | 30.272590 | 14.181209 | 0.17 | 76.0000 21.0000 | 27.0000 |
| SibSp | 0.447368 | 0.896760 | 0.00 | 8.0000 0.0000 | 0.0000 |
| Parch | 0.392344 | 0.981429 | 0.00 | 9.0000 0.0000 | 0.0000 |
| Fare | 35.627188 | 55.907576 | 0.00 | 512.3292 7.8958 | 14.4542 |
|  | q3 |  |  |  |  |
| Passenger Survived | Id 1204.75  1.00 | | | | |
| Pclass | 3.00 | | | | |
| Age | 39.00 | | | | |
| SibSp | 1.00 | | | | |
| Parch | 0.00 | | | | |
| Fare | 31.50 | | | | |











# CORRELATION METRICS

**AIM**:

To write a map-reduce program in python to compute correlation metrics, such as, Pearson, Spearman, and Kendall between pairs of all the numerical variables in a large multi-variate dataset **ALGORITHM:**

1. Start the program
2. Import necessary libraries such as pandas, numpy and itertools.
3. Load the dataset from a CSV file into a pandas DataFrame.
4. Define mapper function to select only numerical columns from the dataset.
5. Compute the Pearson, Spearman, and Kendall correlation coefficients for each pair of numerical columns
6. Define reducer function to return the correlation results
7. Convert the correlation results into a DataFrame for a structured, readable output
8. Display the DataFrame with column headers
9. Stop the program

# PROGRAM

From itertools import combinations import pandas as pd

importnumpy as np

# Load the dataset from a CSV file

dataset = pd.read\_csv(r"C:\Users\Student\Downloads\archive (3)\CC GENERAL.csv") # Mapper function to calculate correlation metrics

def mapper(dataset):

# Get only numerical columns from the dataset

numerical\_columns = dataset.select\_dtypes(include=[np.number]) # Store correlation results for all combinations

correlation\_results = []

# Calculate Pearson, Spearman, and Kendall correlations for each pair for col1, col2 in combinations(numerical\_columns.columns, 2):

pearson\_corr = numerical\_columns[col1].corr(numerical\_columns[col2], method='pearson') spearman\_corr = numerical\_columns[col1].corr(numerical\_columns[col2], method='spearman') kendall\_corr = numerical\_columns[col1].corr(numerical\_columns[col2], method='kendall')

# Append the results as a tuple

correlation\_results.append((col1, col2, pearson\_corr, spearman\_corr, kendall\_corr)) returncorrelation\_results

# Reducer function to simply return the mapped data (no further aggregation needed) def reducer(mapped\_data):

# Since the mapper already computes the correlations, the reducer can just return the results returnmapped\_data

# Step 3: Run MapReduce # Map step

mapped\_data = mapper(dataset) # Reduce step

correlation\_results = reducer(mapped\_data)

# Convert the correlation results to a DataFrame for better readability

correlation\_df = pd.DataFrame(correlation\_results, columns=['Variable 1', 'Variable 2', 'Pearson', 'Spearman', 'Kendall'])

# Display the correlation results print(correlation\_df)

# OUTPUT :

Variable 1 Variable 2 Pearson Spearman Kendall

|  |  |  |
| --- | --- | --- |
| 0 | BALANCE | BALANCE\_FREQUENCY 0.322412 0.544981 0.429865 |
| 1 | BALANCE | PURCHASES 0.181261 0.006473 0.022283 |
| 2 | BALANCE | ONEOFF\_PURCHASES 0.164350 0.145667 0.107438 |
| 3 | BALANCE | INSTALLMENTS\_PURCHASES 0.126469 -0.089979 -0.056408 |
| 4 | BALANCE | CASH\_ADVANCE 0.496692 0.565888 0.427766 |

.. ... ... ... ... ...

131 PAYMENTS PRC\_FULL\_PAYMENT 0.112138 0.187302 0.141349

132 PAYMENTS TENURE 0.106136 0.205412 0.163455

133 MINIMUM\_PAYMENTS PRC\_FULL\_PAYMENT -0.140379 -0.478365 -0.358310

134 MINIMUM\_PAYMENTS TENURE 0.059444 0.136618 0.108984

135 PRC\_FULL\_PAYMENT TENURE -0.016486 0.020244 0.01805

# CLUSTERING OF A LARGE MULTI-VARIATE DATASET

**AIM:**

To write a map reduce program in Python to perform clustering of a large multi-variate dataset.

# ALGORITHM:

1. Start the program
2. Import necessary libraries like numpy and csv
3. Define load\_data function to read the dataset from a CSV file.
4. Convert numeric data to a NumPy array, handling non-numeric data by skipping invalid rows
5. Initialize centroid function to randomly select kkk data points as initial centroids from the dataset
6. Perform map function to assign each data point to the nearest centroid by calculating the Euclidean distance
7. Perform reduce function to calculate the mean of data points in each cluster to update the centroids
8. Repeat the map and reduce functions until centroids do not change or the maximum number of iterations is reached
9. Print final centroids and the points in each cluster
10. Stop the program

# PROGRAM

Import numpy as np Import csv

file\_path =r"C:\Users\Student\Downloads\Dataset.csv" defload\_data(file\_path):

""" Load dataset from a CSV file into a NumPy array, skipping the header. """ with open(file\_path, 'r') as file:

reader = csv.reader(file)

header = next(reader) # Skip the header data = []

for row in reader:

try:

# Convert numeric columns to float; assuming the first column is an ID

numeric\_row = [float(value) for value in row[1:]] # Adjust indexing based on your data data.append(numeric\_row)

exceptValueError:

continue # Skip rows with non-numeric values

returnnp.array(data) definitialize\_centroids(data, k):

""" Randomly initialize centroids from the dataset. """ indices = np.random.choice(data.shape[0], k, replace=False) return data[indices]

defmap\_phase(data, centroids):

""" Map phase: Assign each point to the nearest centroid. """ clusters = {i: [] for i 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 defreduce\_phase(clusters):

""" Reduce phase: Calculate new centroids from the clusters. """ new\_centroids = []

forcluster\_points in clusters.values():

ifcluster\_points: # Check if the cluster is not empty new\_centroid = np.mean(cluster\_points, axis=0) new\_centroids.append(new\_centroid)

else:

new\_centroids.append(np.random.rand(len(cluster\_points[0]))) # Random initialization returnnp.array(new\_centroids)

defkmeans(file\_path, k, max\_iterations=100):

""" K-Means algorithm using a MapReduce-like approach. """ data = load\_data(file\_path)

centroids = initialize\_centroids(data, k) for iteration in range(max\_iterations):

clusters = map\_phase(data, centroids) new\_centroids = reduce\_phase(clusters) ifnp.all(centroids == new\_centroids):

break

centroids = new\_centroids return centroids, clusters

if name == ' main ':

# Path to your dataset

file\_path = r"C:\Users\Student\Downloads\Data set (1).csv"

k = 3 # Number of clusters (you can adjust this based on your needs) final\_centroids, final\_clusters = kmeans(file\_path, k)

print("Final Centroids:") print(final\_centroids) print("Clusters:")

forcluster\_id, points in final\_clusters.items(): print(f"Cluster {cluster\_id}:")

print(np.array(points)) # Convert to NumPy array for better readability

# OUTPUT :

Final Centroids:

[[4.83301557e+03 9.14620970e-01 1.05307798e+04 7.42990541e+03 3.10103977e+03 5.43050951e+03 7.42298917e-01 5.86638699e-01

5.97387970e-01 2.30828677e-01 9.25563910e+00 8.44135338e+01

1.24890977e+04 1.79058694e+04 2.47407345e+03 3.29256820e-01

1.18421053e+01]

[3.42346862e+03 9.46718838e-01 1.60033818e+03 9.97705933e+02 6.02950444e+02 2.10615966e+03 5.44120884e-01 3.18911864e-01

3.89734613e-01 2.11820479e-01 5.79111111e+00 2.24164444e+01

8.98909091e+03 2.92089034e+03 1.58035601e+03 1.36138769e-01

1.17417778e+01]

[8.76792372e+02 8.76021309e-01 6.16390905e+02 3.18393259e+02 2.98309285e+02 4.99693396e+02 4.73446067e-01 1.57149088e-01

3.56433236e-01 1.08916299e-01 2.29617783e+00 1.09008476e+01

2.74528604e+03 1.03266735e+03 5.72410795e+02 1.64024103e-01

1.14532225e+01]]

Clusters:

Cluster 0:

[[1.41160223e+03 4.54545000e-01 9.63240000e+02 7.19948985e+02

3.33333000e-01 1.20000000e+01]

[2.99042219e+03 9.09091000e-01 4.52327000e+03 1.10703080e+03

4.54545000e-01 1.20000000e+01]

[2.39291774e+03 1.00000000e+00 1.24624400e+04 2.00099092e+03

0.00000000e+00 1.20000000e+01]

# CLASSIFICATION OF A LARGE MULTI-VARIATE DATASET

**AIM:**

To write a map-reduce program in Python to perform classification of a large multi-variate dataset

# ALGORITHM:

1. Start the program
2. Import necessary libraries like NumPy and Pandas
3. Load the Iris dataset from the specified CSV file
4. Calculate the Euclidean distance between two data points
5. For each training data point:
   * Calculate the distance from the test point
   * Store the distance along with the label of the training point
   * Sort the distances in ascending order
   * Return the top k nearest neighbors.
6. For each test point in the test dataset:
   * Call map function to find the k nearest neighbors
   * Call reduce function to predict the class of the test point based on its nearest neighbors
   * Store all predictions
7. Split the data into training (80%) and testing (20%) sets
8. Execute the specified function to get predictions for the test set
9. Print the predictions and the actual classes from the test set for comparison
10. Stop the program

# PROGRAM

Import numpy as np import pandas as pd # Path to your dataset

file\_path =r"C:\Users\Student\Downloads\iris[1].csv" defload\_data(file\_path):

"""Load the Iris dataset from a CSV file.""" returnpd.read\_csv(file\_path) defeuclidean\_distance(point1, point2):

"""Calculate the Euclidean distance between two points.""" returnnp.sqrt(np.sum((point1 - point2) \*\* 2)) defmap\_phase(training\_data, test\_point, k):

"""Map phase: Calculate distances from the test point to all training points.""" distances = []

for index, row in training\_data.iterrows():

distance = euclidean\_distance(row[:-1], test\_point) # Exclude label distances.append((distance, row[-1])) # Store distance and label distances.sort(key=lambda x: x[0]) # Sort by distance

return distances[:k] # Return the k nearest neighbors defreduce\_phase(nearest\_neighbors):

"""Reduce phase: Determine the most common class among the nearest neighbors.""" classes = {}

for \_, label in nearest\_neighbors:

classes[label] = classes.get(label, 0) + 1

return max(classes, key=classes.get) # Return the class with the highest count defknn\_classify(training\_data, test\_data, k):

"""KNN classification using a MapReduce-like approach.""" predictions = []

for index, test\_point in test\_data.iterrows():

nearest\_neighbors = map\_phase(training\_data, test\_point[:-1], k) predicted\_class = reduce\_phase(nearest\_neighbors) predictions.append(predicted\_class)

return predictions

if name == ' main ': # Load the dataset

iris\_data = load\_data(file\_path)

# Split the data into training and test sets

training\_data = iris\_data.sample(frac=0.8, random\_state=42) # 80% for training test\_data = iris\_data.drop(training\_data.index) # 20% for testing

# Number of neighbors k = 3

# Classify test data

predictions = knn\_classify(training\_data, test\_data, k) # Display predictions

print("Predictions for the test set:") print(predictions)

# Optionally, you can also check the actual classes print("Actual classes:")

print(test\_data.iloc[:, -1].values) # Assuming the last column is the label

# OUTPUT:

Predictions for the test set:

['setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'versicolor', 'versicolor', 'versicolor', 'versicolor', 'versicolor', 'versicolor', 'versicolor', 'versicolor', 'versicolor', 'versicolor', 'versicolor', 'virginica', 'virginica', 'versicolor', 'virginica', 'virginica', 'virginica', 'virginica', 'virginica', 'virginica', 'virginica', 'virginica', 'virginica']

Actual classes:

['setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'versicolor' 'versicolor' 'versicolor' 'versicolor' 'versicolor' 'versicolor' 'versicolor' 'versicolor' 'versicolor' 'versicolor' 'versicolor' 'virginica' 'virginica' 'virginica' 'virginica' 'virginica' 'virginica' 'virginica' 'virginica' 'virginica' 'virginica' 'virginica' 'virginica']