Data Mining and Analysis

Lab Assignment

***Submitted by***

**MAYANK RATHORE**

***24MCA10158***

******

**Submited to**

**Dr. Sandeep Monga**

**VIT Bhopal University**

**Bhopal, Madhya Pradesh**

**May, 2025**

|  |  |  |
| --- | --- | --- |
| **Table of Contents** | | |
| **SL.NO** | **Name of the experiment** | **DATE:** |
| **1.** | **Write a function in python that takes a DataFrame and generates a complete statistical**  **summary for numerical variables: count, mean, std, min, max, quartiles with out using**  **any inbuilt library/functions** | **29/05/25** |
| **2** | **Write a Python program that implements the Apriori algorithm to find frequent itemsets from transaction data. Input: List of transactions (each as a list of items) andminimum support threshold. Output: All frequent itemsets with their support counts** | **29/05/25** |
| **3** | **Write a python program to implement various evaluation metrics without using scikit-learn, such as Confusion matrix Precision, Recall, F1-score** | **29/05/25** |
| **4** | **Write a python program to implement Euclidean, Manhattan distance metrics without using inbuilt function** | **29/05/25** |
| **5** | **Write a python program to calculate R2, MAE, RMSE, MSE without using Sklearn**  **library** | **29/05/25** |
| **6** | **Write a program in python to convert non-stationarity data into stationarity** | **29/05/25** |
| **7** | **You are given a dataset consisting of 2D points. Your task is to implement the K-Means Clustering algorithm (i.e., without using libraries like scikit-learn) and group the data**  **into k clusters.** | **29/05/25** |
| **8** | **Write a Python program to decompose a time series into Trend, Seasonality, and Residual components using: Additive model, Multiplicative model Input: Monthly sales data (e.g., 3–5 years) Frequency = 12 (for monthly seasonality)**  **Output: Plot of the original series, trend, seasonal,and residual components** | **29/05/25** |

|  |  |
| --- | --- |
| **Date: 29/05/25** | **Data Mining and Analysis** |
| **Exp. No:1** | Write a function in python that takes a DataFrame and generates a complete statistical summary for numerical variables: count, mean, std, min, max, quartiles with out using any inbuilt library/functions |

Code

def read\_air\_passengers(file\_path):

    months = []

    passengers = []

    with open(file\_path, 'r') as f:

        next(f)  *# Skip header*

        for line in f:

            month, count = line.strip().split(',')

            months.append(month)

            passengers.append(int(count))

    return {'Month': months, 'Passengers': passengers}

def statistical\_summary(data\_dict, column\_name):

    data = data\_dict[column\_name]

    n = len(data)

*# Count*

    count = n

*# Mean*

    total = 0

    for val in data:

        total += val

    mean = total / count

*# Standard Deviation*

    variance = sum((val - mean) \*\* 2 for val in data) / count

    std\_dev = variance \*\* 0.5

*# Min and Max*

    min\_val = data[0]

    max\_val = data[0]

    for val in data:

        if val < min\_val:

            min\_val = val

        if val > max\_val:

            max\_val = val

*# Sort data for quartiles*

    sorted\_data = sorted(data)

    def get\_percentile(data, p):

        k = (len(data) - 1) \* p

        f = int(k)

        c = f + 1

        if c >= len(data):

            return data[f]

        return data[f] + (data[c] - data[f]) \* (k - f)

    q1 = get\_percentile(sorted\_data, 0.25)

    q2 = get\_percentile(sorted\_data, 0.50)

    q3 = get\_percentile(sorted\_data, 0.75)

*# Display summary*

    print(f"Statistical Summary for '{column\_name}':")

    print(f"Count     : {count}")

    print(f"Mean      : {round(mean, 2)}")

    print(f"Std Dev   : {round(std\_dev, 2)}")

    print(f"Min       : {min\_val}")

    print(f"25% (Q1)  : {q1}")

    print(f"50% (Q2)  : {q2}")

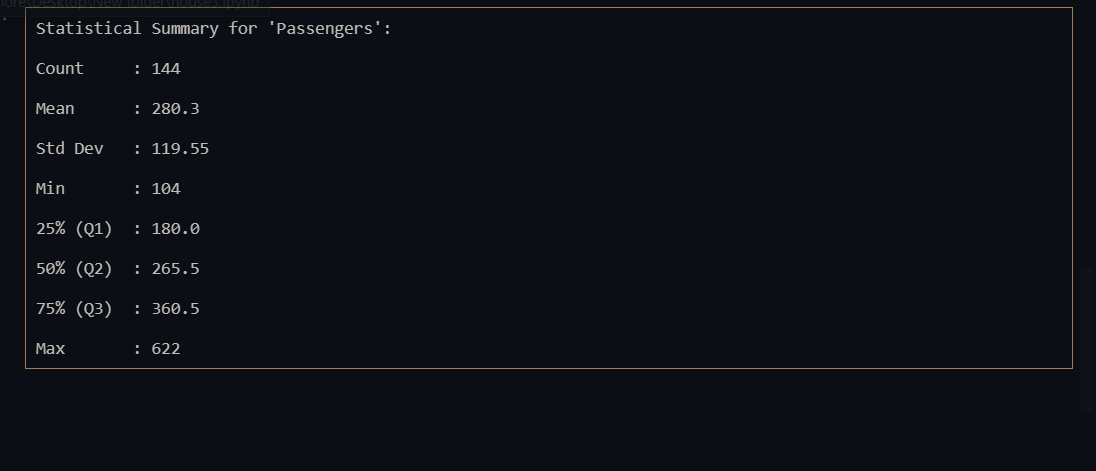
    print(f"75% (Q3)  : {q3}")

    print(f"Max       : {max\_val}")

data = read\_air\_passengers("AirPassengers.csv")

statistical\_summary(data, "Passengers")

Ouput:



|  |  |
| --- | --- |
| **Date: 29/05/25** | **Data Mining and Analysis** |
| **Exp. No:2** | Write a Python program that implements the Apriori algorithm to find frequent itemsets from transaction data. Input: List of transactions (each as a list of items) and minimum support threshold. Output: All frequent itemsets with their support counts |

Code: def get\_frequent\_itemsets(transactions, min\_support):

    from itertools import combinations

    def get\_support(itemset, transactions):

        count = 0

        for transaction in transactions:

            if set(itemset).issubset(set(transaction)):

                count += 1

        return count

    def generate\_candidates(prev\_frequent, length):

        candidates = []

        prev\_frequent = list(prev\_frequent)

        for i in range(len(prev\_frequent)):

            for j in range(i + 1, len(prev\_frequent)):

                candidate = tuple(sorted(set(prev\_frequent[i]) | set(prev\_frequent[j])))

                if len(candidate) == length and candidate not in candidates:

                    candidates.append(candidate)

        return candidates

    itemsets = []

    support\_data = {}

*# Get unique items*

    items = set()

    for transaction in transactions:

        for item in transaction:

            items.add(item)

*# Level 1: single items*

    L1 = []

    for item in sorted(items):

        support = get\_support([item], transactions)

        if support >= min\_support:

            L1.append((item,))

            support\_data[(item,)] = support

    current\_L = L1

    k = 2

    while current\_L:

        itemsets.extend(current\_L)

        candidates = generate\_candidates(current\_L, k)

        current\_L = []

        for candidate in candidates:

            support = get\_support(candidate, transactions)

            if support >= min\_support:

                current\_L.append(candidate)

                support\_data[candidate] = support

        k += 1

    return support\_data

transactions = [

    ['milk', 'bread', 'eggs'],

    ['milk', 'bread'],

    ['milk', 'eggs'],

    ['bread', 'eggs'],

    ['milk', 'bread', 'eggs'],

    ['bread']

]

min\_support = 3

frequent\_itemsets = get\_frequent\_itemsets(transactions, min\_support)

print("Frequent Itemsets with Support Counts:")

for itemset, support in frequent\_itemsets.items():

    print(f"{itemset} : {support}")

Output:



|  |  |
| --- | --- |
| **Date: 29/05/25** | **Data Mining and Analysis** |
| **Exp. No:3** | **Write a python program to implement various evaluation metrics without using scikit-**  **learn, such as Confusion matrix Precision, Recall, F1-score** |

Code:

import pandas as pd

import random

import matplotlib.pyplot as plt

import seaborn as sns

*# Load the data*

df = pd.read\_csv('Iris.csv')

*# Extract true labels*

true\_labels = df['Species'].tolist()

*# Step 1: Get unique classes*

classes = sorted(list(set(true\_labels)))  *# ['Iris-setosa', 'Iris-versicolor', 'Iris-virginica']*

*# Step 2: Generate mock predictions*

def generate\_mock\_predictions(true\_labels):

    predicted = []

    for label in true\_labels:

        if random.random() < 0.85:

            predicted.append(label)

        else:

            predicted.append(random.choice(classes))

    return predicted

*# Step 3: Confusion Matrix*

def compute\_confusion\_matrix(true\_labels, pred\_labels, classes):

    matrix = {actual: {pred: 0 for pred in classes} for actual in classes}

    for t, p in zip(true\_labels, pred\_labels):

        matrix[t][p] += 1

    return matrix

*# Step 4: Precision, Recall, F1*

def compute\_metrics(conf\_matrix, classes):

    metrics = {}

    for *cls* in classes:

        tp = conf\_matrix[*cls*][*cls*]

        fp = sum(conf\_matrix[other][*cls*] for other in classes if other != *cls*)

        fn = sum(conf\_matrix[*cls*][other] for other in classes if other != *cls*)

        precision = tp / (tp + fp) if (tp + fp) else 0

        recall = tp / (tp + fn) if (tp + fn) else 0

        f1 = 2 \* precision \* recall / (precision + recall) if (precision + recall) else 0

        metrics[*cls*] = {

            'Precision': round(precision, 2),

            'Recall': round(recall, 2),

            'F1-Score': round(f1, 2)

        }

    return metrics

*# Run evaluation*

pred\_labels = generate\_mock\_predictions(true\_labels)

conf\_matrix = compute\_confusion\_matrix(true\_labels, pred\_labels, classes)

metrics = compute\_metrics(conf\_matrix, classes)

*# Print confusion matrix*

print("🔹 Confusion Matrix:")

for actual in classes:

    print(f"{actual}: {conf\_matrix[actual]}")

*# Print metrics*

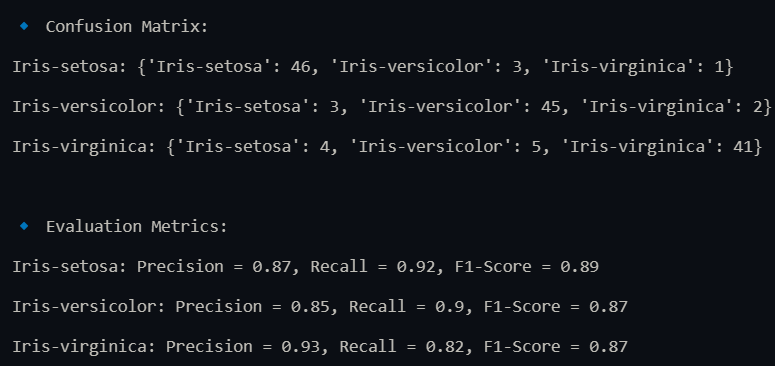
print("\n🔹 Evaluation Metrics:")

for *cls* in metrics:

    m = metrics[*cls*]

    print(f"{*cls*}: Precision = {m['Precision']}, Recall = {m['Recall']}, F1-Score = {m['F1-Score']}")

Output:



|  |  |
| --- | --- |
| **Date: 29/05/25** | **Data Mining and Analysis** |
| **Exp. No:4** | **Write a python program to implement Euclidean, Manhattan distance metrics without**  **using inbuilt function** |

Code:

def euclidean\_distance(point1, point2):

    sum\_sq = 0

    for i in range(len(point1)):

        diff = point1[i] - point2[i]

        sum\_sq += diff \* diff

    x = sum\_sq

    guess = x / 2.0

    for \_ in range(20):

        guess = (guess + x / guess) / 2

    return guess

def manhattan\_distance(point1, point2):

    total = 0

    for i in range(len(point1)):

        total += abs(point1[i] - point2[i])

    return total

p1 = [2, 4, 5]

p2 = [5, 1, 7]

print("Euclidean Distance:", round(euclidean\_distance(p1, p2), 3))

print("Manhattan Distance:", manhattan\_distance(p1, p2))

Output:

Euclidean Distance: 4.69

Manhattan Distance: 8

|  |  |
| --- | --- |
| **Date: 29/05/25** | **Data Mining and Analysis** |
| **Exp. No:5** | **Write a python program to calculate R2, MAE, RMSE, MSE without using Sklearn library** |

Code:

def mean(values):

    return sum(values) / len(values)

def mse(y\_true, y\_pred):

    n = len(y\_true)

    error\_sum = 0

    for yt, yp in zip(y\_true, y\_pred):

        error\_sum += (yt - yp) \*\* 2

    return error\_sum / n

def rmse(y\_true, y\_pred):

    return sqrt(mse(y\_true, y\_pred))

def mae(y\_true, y\_pred):

    n = len(y\_true)

    error\_sum = 0

    for yt, yp in zip(y\_true, y\_pred):

        error\_sum += abs(yt - yp)

    return error\_sum / n

def r2\_score(y\_true, y\_pred):

    mean\_y = mean(y\_true)

    ss\_tot = 0

    ss\_res = 0

    for yt, yp in zip(y\_true, y\_pred):

        ss\_tot += (yt - mean\_y) \*\* 2

        ss\_res += (yt - yp) \*\* 2

    return 1 - (ss\_res / ss\_tot)

def sqrt(x):

    guess = x / 2.0 if x > 1 else 1

    for \_ in range(20):

        guess = (guess + x / guess) / 2

    return guess

y\_true = [3, -0.5, 2, 7]

y\_pred = [2.5, 0.0, 2, 8]

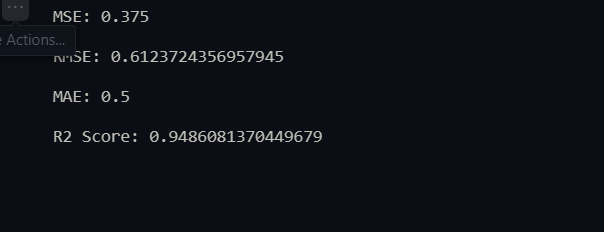
print("MSE:", mse(y\_true, y\_pred))

print("RMSE:", rmse(y\_true, y\_pred))

print("MAE:", mae(y\_true, y\_pred))

print("R2 Score:", r2\_score(y\_true, y\_pred))

Output:



|  |  |
| --- | --- |
| **Date: 29/05/25** | **Data Mining and Analysis** |
| **Exp. No:6** | **Write a program in python to convert non-stationarity data into stationarity** |

Code:

import pandas as pd

import matplotlib.pyplot as plt

df = pd.read\_csv('AirPassengers.csv')

data = df['#Passengers']

plt.figure(figsize=(12,4))

plt.plot(data)

plt.title('Original Air Passenger Data')

plt.show()

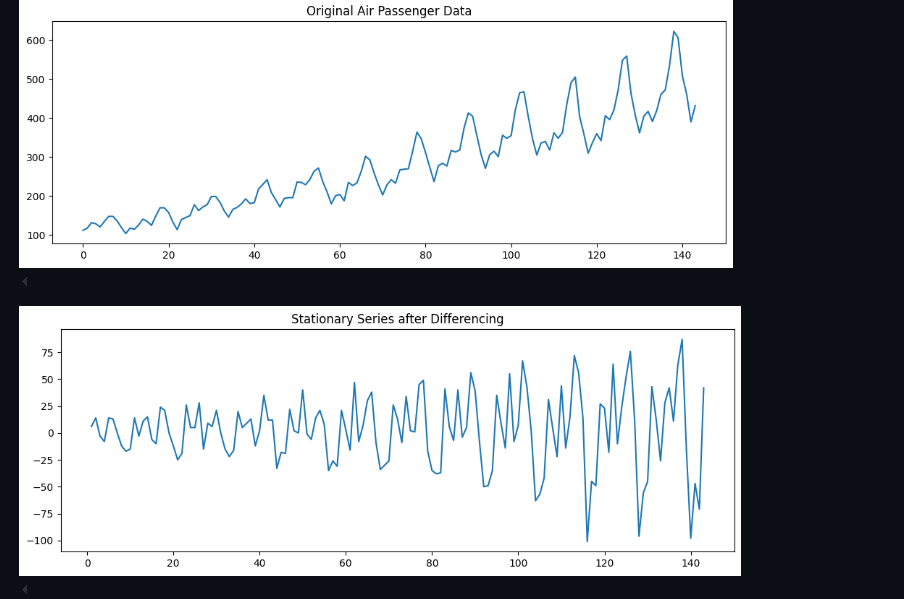
diff\_data = data.diff().dropna()

plt.figure(figsize=(12,4))

plt.plot(diff\_data)

plt.title('Stationary Series after Differencing')

plt.show()

Output:

|  |  |
| --- | --- |
| **Date: 29/05/25** | **Data Mining and Analysis** |
| **Exp. No:7** | **You are given a dataset consisting of 2D points. Your task is to implement the K-Means Clustering algorithm (i.e., without using libraries like scikit-learn) and group the data into k clusters.** |

Code:

import pandas as pd

import random

import matplotlib.pyplot as plt

df = pd.read\_csv('Iris.csv')

*# print(df.head())*

data = df[['SepalLengthCm', 'SepalWidthCm']].values.tolist()

*# Step 2: Euclidean Distance*

def euclidean(p1, p2):

    return sum((x - y) \*\* 2 for x, y in zip(p1, p2)) \*\* 0.5

*# Step 3: Initialize centroids randomly*

def initialize\_centroids(data, k):

    return random.sample(data, k)

*# Step 4: Assign clusters*

def assign\_clusters(data, centroids):

    clusters = [[] for \_ in centroids]

    for point in data:

        distances = [euclidean(point, centroid) for centroid in centroids]

        cluster\_idx = distances.index(min(distances))

        clusters[cluster\_idx].append(point)

    return clusters

*# Step 5: Update centroids*

def update\_centroids(clusters):

    new\_centroids = []

    for cluster in clusters:

        mean = [sum(dim) / len(cluster) for dim in zip(\*cluster)]

        new\_centroids.append(mean)

    return new\_centroids

def has\_converged(old, new):

    return all(euclidean(o, n) < 1e-4 for o, n in zip(old, new))

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

    centroids = initialize\_centroids(data, k)

    for \_ in range(max\_iter):

        clusters = assign\_clusters(data, centroids)

        new\_centroids = update\_centroids(clusters)

        if has\_converged(centroids, new\_centroids):

            break

        centroids = new\_centroids

    return centroids, clusters

k = 3

centroids, clusters = kmeans(data, k)

colors = ['r', 'g', 'b']

plt.figure(figsize=(8, 6))

for idx, cluster in enumerate(clusters):

    x, y = zip(\*cluster)

    plt.scatter(x, y, color=colors[idx], label=f'Cluster {idx+1}')

cx, cy = zip(\*centroids)

plt.scatter(cx, cy, color='black', marker='x', s=100, label='Centroids')

plt.xlabel('Sepal Length')

plt.ylabel('Sepal Width')

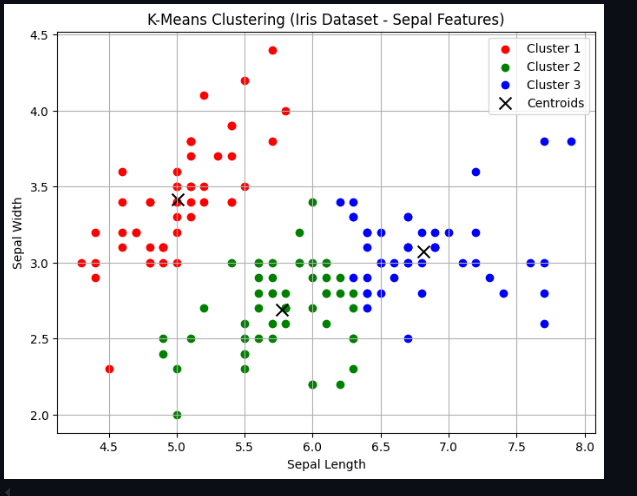
plt.title('K-Means Clustering (Iris Dataset - Sepal Features)')

plt.legend()

plt.grid(True)

plt.show()

Output:



|  |  |
| --- | --- |
| **Date: 29/05/25** | **Data Mining and Analysis** |
| **Exp. No:8** | **Write a Python program to decompose a time series into Trend, Seasonality, and Residual components using: Additive model, Multiplicative model Input: Monthly sales data (e.g., 3–5 years) Frequency = 12 (for monthly seasonality).**  **Output: Plot of the original series, trend, seasonal, and residual components** |

Code:

import pandas as pd

import matplotlib.pyplot as plt

from statsmodels.tsa.seasonal import seasonal\_decompose

*# Step 1: Load the dataset*

df = pd.read\_csv('AirPassengers.csv', parse\_dates=['Month'], index\_col='Month')

data = df['#Passengers']

*# Step 2: Additive Decomposition*

add\_decomp = seasonal\_decompose(data, model='additive', period=12)

*# Step 3: Multiplicative Decomposition*

mult\_decomp = seasonal\_decompose(data, model='multiplicative', period=12)

*# Step 4: Plot Additive Decomposition*

plt.figure(figsize=(12, 8))

plt.suptitle('Additive Decomposition of Air Passenger Data', fontsize=16)

add\_decomp.plot()

plt.tight\_layout()

plt.show()

*# Step 5: Plot Multiplicative Decomposition*

plt.figure(figsize=(12, 8))

plt.suptitle('Multiplicative Decomposition of Air Passenger Data', fontsize=16)

mult\_decomp.plot()

plt.tight\_layout()

plt.show()

Output:

