1. Apply the descriptive statistical functions in Python/R to find the minimum value, maximum value, mean value, median value, mode value, quantile, standard deviation, variance, and summary of the above dataset by considering appropriate features.

2. Demonstrate the above dataset's different data visualization techniques such as Scatter Plot, Horizontal Bar Chart, Histogram, and Line Graph by considering appropriate features.

CODE:

```
import pandas as pd

import matplotlib.pyplot as plt

# Soumya Ranjan Jena, 21BDS0173

df = pd.read_csv("Cars93.csv")

dataframe = df.dropna()

dataframe['Cylinders'] =
pd.to_numeric(dataframe['Cylinders'], errors='coerce')

dataframe = dataframe[dataframe['Cylinders'].notnull()]

row_headers = ['Min.Price', 'Price', 'Max.Price', 'MPG.city',
'MPG.highway', 'Cylinders',

'EngineSize', 'Horsepower', 'RPM', 'Rev.per.mile',
'Fuel.tank.capacity', 'Passengers',

'Length', 'Wheelbase', 'Width', 'Turn.circle',
'Rear.seat.room', 'Luggage.room', 'Weight']

def plot_visualizations(column):

plt.figure(figsize=(10, 6))

plt.subplot(2, 2, 1)

plt.scatter(dataframe.index, dataframe[column], color='b',
alpha=0.7)

plt.xlabel('Index')

plt.ylabel(column)

plt.title(f"Scatter Plot of {column}")

plt.subplot(2, 2, 2)

plt.barh(dataframe.index, dataframe[column],
color='skyblue')

plt.xlabel(column)

plt.ylabel('Index')

plt.title(f"Horizontal Bar Chart of {column}")

plt.subplot(2, 2, 3)

.hist(dataframe[column], bins=10, edgecolor='black')

plt.xlabel(column)

plt.ylabel('Frequency')

plt.title(f"Histogram of {column}")

plt.subplot(2, 2, 4)

plt.plot(dataframe.index, dataframe[column], color='r',
marker='o')

plt.xlabel('Index')

plt.ylabel(column)

plt.title(f"Line Graph of {column}")

plt.tight_layout()

plt.show()

for column in row_headers:

print("\nSummary of:", column)

print("Minimum value =", dataframe[column].min())

print("Maximum value =", dataframe[column].max())

print("Mean value =", dataframe[column].mean())

print("Median value =", dataframe[column].median())

print("Mode value =", dataframe[column].mode().values)

print("Quantile value =",
dataframe[column].quantile([0.25, 0.5, 0.75]))

print("Standard Deviation value =",
dataframe[column].std())

print("Variance value =", dataframe[column].var())

print("Summary:", dataframe[column].describe())

plot_visualizations(column)
```

The following data has the students mark from a unit test Marks = {
'Name':['Raman','Raman','Raman','Raman','Zuhaire','Zuhaire','Zuhaire'
,'Zuhaire','Ashravy','Ashravy','Ashravy','Ashravy','Mishti','Mishti', 'Mishti','Mishti'],
'UT':[1,2,3,4,1,2,3,4,1,2,3,4,1,2,3,4],
'Maths':[22,21,14,np.NaN,20,23,22,19,23,24,12,15,15,18,17,14],
'Science':[21,20,19,np.NaN,17,15,18,20,19,22,25,20,22,21,18,20],
'S.St':[18,17,15,19,22,21,19,17,20,24,19,20,25,25,20,19],
'Hindi':[20,22,24,18,24,25,23,21,
15,17,21,20,22,24,25,20],
'Eng':[21,24,23,np.NaN,19,15,13,16,22,21,23,17,22,23,20,18] } a. Find the attributes which have missing values. b. Find out the total count of the missing values in the data. c. Handle the missing values using following two ways: i. Replace the missing values by a value before that. ii. Remove the rows having missing values from the original dataset d. Go to the original data and fill the nan by using mode. e. find the percentage of marks scored by Raman in hindi before and after handling missing data f. Replace the missing values with linear interpolation imputation
methods.

CODE:

```
import numpy as np

import pandas as pd

Marks = {

'Name': ['Raman', 'Raman', 'Raman', 'Raman', 'Zuhaire', 'Zuhaire',

'Zuhaire', 'Zuhaire', 'Ashravy', 'Ashravy', 'Ashravy', 'Mishti', 'Mishti',

'Mishti'],

'Unit Test': [1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4, 1, 2],

'Mathematics': [22, 21, 14, np.nan, 20, 23, 22, 19, 23, 24, 12, 15, 18,

17],

'Science': [21, 20, 19, np.nan, 17, 15, 18, 20, 19, 22, 25, 20, 22, 21],

'Social Studies': [18, 17, 15, 19, 22, 21, 19, 17, 20, 24, 19, 20, 25,

25],

'Hindi': [20, 22, 24, 18, 24, 25, 23, 21, 15, 17, 21, 20, 22, 24],

'English': [21, 24, 23, np.nan, 19, 15, 13, 16, 22, 21, 23, 17, 22, 23]

}

df = pd.DataFrame(Marks)

# a. Find the attributes which have missing values. missing_values = df.isnull().sum()

print("Attributes with missing values:")

print(missing_values[missing_values > 0])

# b. Find out the total count of the missing values in the data.

total_missing = df.isnull().sum().sum()

print("\nTotal count of missing values in the data:", total_missing)

# c. Handle the missing values using two methods: # i. Replace the missing values by a value before that.

df_ffill = df.fillna(method='ffill')

 ("\nData after filling missing values using the previous value:")

print(df_ffill)

# ii. Remove the rows having missing values from the original dataset

df_dropped = df.dropna()

print("\nData after removing rows with missing values:")

print(df_dropped)

# d. Fill the NaN in the original data using mode.

df_mode_fill = df.copy() for column in df_mode_fill.columns:

df_mode_fill[column].fillna(df_mode_fill[column].mode()[0], inplace=True)

print("\nData after filling missing values with mode:")

print(df_mode_fill)

# e. Find the percentage of marks scored by Raman in Hindi before and after handling missing data.

raman_hindi_before = df[df['Name'] == 'Raman']['Hindi'].sum()

raman_hindi_after = df_mode_fill[df_mode_fill['Name'] ==

'Raman']['Hindi'].sum()

percentage_before = (raman_hindi_before / df['Hindi'].sum()) * 100

percentage_after = (raman_hindi_after / df_mode_fill['Hindi'].sum()) * 100

print(f"\nPercentage of marks scored by Raman in Hindi before handling missing

data: {percentage_before:.2f}%")

print(f"Percentage of marks scored by Raman in Hindi after handling missing

data: {percentage_after:.2f}%")

# f. Replace the missing values with linear interpolation imputation methods. df_interpolated = df.interpolate()
```

print("\nData after replacing missing values using linear interpolation:")

print(df_interpolated)

1. Find frequent itemsets.

CODE:

```
import pandas as pd

import numpy as np

import itertools

# Define the transactional data

data = [

['I1', 'I2', 'I5'],

['I2', 'I4'],

['I2', 'I3'],

['I1', 'I2', 'I4'],

['I1', 'I3'],

['I2', 'I3'],

['I1', 'I3'],

['I1', 'I2', 'I3', 'I5'],

['I1', 'I2', 'I3']

]

# Convert transactions to a DataFrame with binary values

all_items = sorted(set(itertools.chain.from_iterable(data)))

binary_df = pd.DataFrame(np.zeros((len(data), len(all_items))), columns=all_items)

for idx, transaction in enumerate(data):

for item in transaction:

binary_df.at[idx, item] = 1

# Find frequent itemsets with a minimum support count of 2

min_support_threshold = 2

frequent_sets = []

total_data_points = len(data)

for size in range(1, len(all_items) + 1):

for combo in itertools.combinations(all_items, size):
```

2. 2. Generate association rules.

```
combo_support_count = binary_df[list(combo)].all(axis=1).sum()

combo_support = combo_support_count / total_data_points # Convert to decimal

if combo_support_count >= min_support_threshold:

frequent_sets.append((combo, combo_support))

# Generate association rules with a minimum confidence of 60%

min_conf = 0.6

association_rules = []

for itemset, support in frequent_sets:

if len(itemset) > 1:

for size in range(1, len(itemset)):

for antecedent_part in itertools.combinations(itemset, size):

consequent_part = tuple(set(itemset) - set(antecedent_part))

antecedent_support = binary_df[list(antecedent_part)].all(axis=1).sum() /

total_data_points

rule_confidence = support / antecedent_support

if rule_confidence >= min_conf:

association_rules.append((antecedent_part, consequent_part,

rule_confidence))

# Display the results

print("Frequent Itemsets:")

for itemset, support in frequent_sets:

print(f"Itemset: {itemset}, Support: {support:.2f}")

print("\nAssociation Rules:")

for antecedent, consequent, confidence in association_rules:
```

```
print(f"Rule: {antecedent} -> {consequent}, Confidence:
{confidence:.2f}")
```

1. Construct a Decision Tree using the Gini Index measure.

CODE:

```
import pandas as pd

from sklearn.tree import DecisionTreeClassifier

from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score,
classification_report

from sklearn.tree import plot_tree

import matplotlib.pyplot as plt

# Load the dataset

file_path =
'C:/Users/bapun/Downloads/mushrooms.csv'

data = pd.read_csv(file_path)

# Display basic information about the dataset

print("Dataset Information:")

print(data.info())

print("\nFirst few rows of the dataset:")

print(data.head())

# Prepare the data for training

# Convert categorical variables to numeric using one-
hot encoding

X = pd.get_dummies(data.drop('class', axis=1))

y = data['class'].apply(lambda x: 1 if x == 'p' else 0) #
Convert class to

binary (1 for 'p', 0 for 'e')

# Split the dataset into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3,

random_state=42)

# Create and train the Decision Tree using Gini Index

clf = DecisionTreeClassifier(criterion='gini',
random_state=42)

clf.fit(X_train, y_train)

# Predict on the test set

y_pred = clf.predict(X_test)

# Calculate accuracy

accuracy = accuracy_score(y_test, y_pred)

print(f"\nAccuracy: {accuracy}")

# Print classification report

print("\nClassification Report:")

print(classification_report(y_test, y_pred,
target_names=['e', 'p']))

# Print author details

print("\nSoumya Ranjan Jena | 21BDS0173")

# Visualize the decision tree

plt.figure(figsize=(20,10))

plot_tree(clf, filled=True, feature_names=X.columns,
class_names=['e', 'p'])

plt.show()
```

2. Construct a Naïve Bayes classifier and display the performance metrics such as Accuracy, Precision, Recall, and F1 score.

CODE:

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.feature_extraction.text import CountVectorizer

from sklearn.naive_bayes import MultinomialNB

from sklearn.metrics import accuracy_score, precision_score,
recall_score,

f1_score, classification_report

import matplotlib.pyplot as plt

import numpy as np

# Load the dataset

file_path = 'C:/Users/bapun/Downloads/amazon_alexa.tsv'

data = pd.read_csv(file_path, sep=

'\t')

# Prepare the data for training

# Using 'verified_reviews' as the feature and 'feedback' as the
label

X = data['verified_reviews']

y = data['feedback']

# Convert text data to numerical data using CountVectorizer

vectorizer = CountVectorizer()

X_vectorized = vectorizer.fit_transform(X)

# Split the dataset into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X_vectorized, y,

test_size=0.3, random_state=42)

# Create and train the Naive Bayes classifier

nb_classifier = MultinomialNB()

nb_classifier.fit(X_train, y_train)

# Predict on the test set

y_pred = nb_classifier.predict(X_test)

# Calculate performance metrics

accuracy = accuracy_score(y_test, y_pred)

precision = precision_score(y_test, y_pred)

recall = recall_score(y_test, y_pred)

f1 = f1_score(y_test, y_pred)

# Print performance metrics

print(f"Accuracy: {accuracy}")

print(f"Precision: {precision}")

print(f"Recall: {recall}")

SOUMYA RANJAN JENA 21BDS0173print(f"F1 Score: {f1}")

# Print detailed classification report

print("\nClassification Report:")

print(classification_report(y_test, y_pred,
target_names=['Negative Feedback',

'Positive Feedback']))

# Plotting the bar graph for the performance metrics

metrics = {

'Accuracy': accuracy,

'Precision': precision,

'Recall': recall,

'F1 Score': f1

}

plt.figure(figsize=(10, 6))

plt.bar(metrics.keys(), metrics.values(), color=['blue', 'green',
'orange',

'red'])

plt.ylim(0, 1) # Set y-axis limit from 0 to 1

plt.ylabel('Score')

plt.title('Performance Metrics of Naive Bayes Classifier')

plt.show()

# Additional plot: Classification report as a bar chart

report = classification_report(y_test, y_pred,
target_names=['Negative

Feedback', 'Positive Feedback'], output_dict=True)

# Extracting precision, recall, and f1-score for both classes

classes = ['Negative Feedback', 'Positive Feedback']

precision_values = [report[cls]['precision'] for cls in classes]

recall_values = [report[cls]['recall'] for cls in classes]
```

```
f1_values = [report[cls]['f1-score'] for cls in classes]

# Plotting the classification report metrics

x = np.arange(len(classes)) # the label locations

width = 0.2 # the width of the bars

fig, ax = plt.subplots(figsize=(10, 6))

rects1 = ax.bar(x - width, precision_values, width,
label='Precision',

color='green')

rects2 = ax.bar(x, recall_values, width, label='Recall',
color='orange')

rects3 = ax.bar(x + width, f1_values, width, label='F1 Score',
color='red')

# Add some text for labels, title and custom x-axis tick labels, etc.

ax.set_ylabel('Scores')

ax.set_title('Precision, Recall, and F1 Score by class')

ax.set_xticks(x)

ax.set_xticklabels(classes)

ax.legend()

plt.ylim(0, 1) # Set y-axis limit from 0 to 1

plt.show()

# Print author details

print("\nSoumya Ranjan Jena | 21BDS0173"
```

Consider any dataset and implement the following algorithms.

1. K-medoid clustering algorithm

2. Agglomera<ve Clustering

3. Divisive clustering

CODE:

```
import pandas as pd

from sklearn.preprocessing import StandardScaler

from sklearn.impute import SimpleImputer

from sklearn_extra.cluster import KMedoids

from sklearn.cluster import AgglomerativeClustering

from sklearn.metrics import silhouette_score

import matplotlib.pyplot as plt

# Load dataset

data = pd.read_csv('/Users/venom/Downloads/CC
GENERAL.csv')

# Preprocessing: Handling missing values and scaling the
data

imputer = SimpleImputer(strategy='mean')

data_imputed =
imputer.fit_transform(data.drop(columns=['CUST_ID']))

scaler = StandardScaler()

data_scaled = scaler.fit_transform(data_imputed)

# K-Medoids Clustering

kmedoids = KMedoids(n_clusters=3, random_state=42)

kmedoids_labels = kmedoids.fit_predict(data_scaled)

kmedoids_silhouette = silhouette_score(data_scaled,
kmedoids_labels)

# Plot K-Medoids Clustering

plt.scatter(data_scaled[:, 0], data_scaled[:, 1],
c=kmedoids_labels, cmap='viridis', s=10)

plt.title(f'K-medoids Clustering (Silhouette Score:
{kmedoids_silhouette:.2f})')

plt.xlabel('Feature 1')

plt.ylabel('Feature 2')

plt.show()

# Agglomerative Clustering

agg_clustering = AgglomerativeClustering(n_clusters=3)

agg_labels = agg_clustering.fit_predict(data_scaled)

agg_silhouette = silhouette_score(data_scaled,
agg_labels)

# Plot Agglomerative Clustering

plt.scatter(data_scaled[:, 0], data_scaled[:, 1],
c=agg_labels, cmap='plasma', s=10)

plt.title(f'Agglomerative Clustering (Silhouette Score:
{agg_silhouette:.2f})')

plt.xlabel('Feature 1')plt.ylabel('Feature 2')
```

plt.show()

 # Divisive Clustering (Approximated by Agglomerative Clustering with a different linkage)

div_clustering = AgglomerativeClustering(n_clusters=3, linkage='ward')

div_labels = div_clustering.fit_predict(data_scaled)

div_silhouette = silhouette_score(data_scaled, div_labels)

 # Plot Divisive Clustering (Hierarchical approximation)

plt.scatter(data_scaled[:, 0], data_scaled[:, 1], c=div_labels, cmap='cool', s=10)

plt.title(**f**'Divisive Clustering (Silhouette Score: {div_silhouette**:.2f**})')

plt.xlabel('Feature 1')

plt.ylabel('Feature 2')

plt.show()

1. Simulate Page Rank algorithm in Python.

CODE:

import numpy as np

def create_transition_matrix(graph, num_nodes):

transition_matrix = np.zeros((num_nodes, num_nodes))

for node in graph:

neighbors = graph[node]

if len(neighbors) == 0:

transition_matrix[:, node] = 1 / num_nodes

else:

for neighbor in neighbors:

transition_matrix[neighbor, node] = 1 / len(neighbors)

return transition_matrix

def simulate_page_rank(graph, num_nodes, damping_factor=0.85, max_iters=100, tol=1e-6):

transition_matrix = create_transition_matrix(graph, num_nodes)

rank = np.ones(num_nodes) / num_nodes

damping_adjustment = np.ones(num_nodes) * (1 - damping_factor) / num_nodes

for i in range(max_iters):

new_rank = damping_factor * np.dot(transition_matrix, rank) + damping_adjustment

if np.linalg.norm(new_rank - rank, ord=1) < tol:

print(**f**"PageRank converged after {i+1} iterations.")

break

rank = new_rank

return rank

if __name__ == "__main__":

graph = {

0: [1, 2],

1: [2],

2: [0],

3: [0, 2]

}

num_nodes = 4

damping_factor = 0.85

page_ranks = simulate_page_rank(graph, num_nodes, damping_factor)

print("Final PageRank Scores:")

for i, score in enumerate(page_ranks):

print(**f**"Node {i}: {score**:.4f**}")

2. Write a Python program to measure document similarity.

CODE:

```
import PyPDF2

import os

from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.metrics.pairwise import cosine_similarity

import numpy as np

# Function to extract text from a PDF

def extract_text_from_pdf(pdf_path):

with open(pdf_path, 'rb') as file:

reader = PyPDF2.PdfReader(file)

text = ''

for page_num in range(len(reader.pages)):

page = reader.pages[page_num]

text += page.extract_text()

return text

# Function to compute cosine similarity between documents

def compute_similarity(documents):

vectorizer = TfidfVectorizer()

tfidf_matrix = vectorizer.fit_transform(documents)

return cosine_similarity(tfidf_matrix)

# Main function to read PDFs, compute similarities, and display the matrix

def main():

# Predefined paths of the PDF documents

pdf_paths = [

"/Users/venom/Desktop/Data_Mining/wildlife_1.pdf",

"/Users/venom/Desktop/Data_Mining/wildlife_2.pdf",

"/Users/venom/Desktop/Data_Mining/wildlife_3.pdf",

"/Users/venom/Desktop/Data_Mining/wildlife_4.pdf"

]

# Extract text from each PDF

documents = [extract_text_from_pdf(pdf) for pdf in pdf_paths]

# Compute the similarity matrix

similarity_matrix = compute_similarity(documents)

# Display the similarity matrix

num_docs = len(pdf_paths)

print("\nPairwise Document Similarity (Cosine Similarity Matrix):\n")

print(" " * 15 + "\t"

.join([f"Document {i + 1}" for i in range(num_docs)]))

SOUMYA RANJAN JENA 21BDS0173for i in range(num_docs):

similarity_row = "\t"

.join([f"{similarity_matrix[i][j]:.6f}" for j in range(num_docs)])

print(f"Document {i + 1}:\t{similarity_row}")

if __name__ == "__main__":

main()
```