*1

```python
import pandas as pd

# Step 1: Load data from CSV
titanic_data = pd.read_csv("./Titanic.csv")

# Step 2: Indexing and Selecting Data
# Select first 5 rows and 'Name' and 'Age' columns
selected_data = titanic_data.loc[:4, ['Name', 'Age']]

# Step 3: Sorting Data
# Sort data by 'Age' in ascending order
sorted_data = titanic_data.sort_values(by='Age', ascending=True)

# Step 4: Describe attributes of data
# Summary statistics for numerical columns
data_description = titanic_data.describe()

# Step 5: Check data types of each column
data_types = titanic_data.dtypes

# Print the results (for testing purpose)
print("Selected Data:\n", selected_data)
print("\nSorted Data by Age:\n", sorted_data.head())
print("\nData Description:\n", data_description)
print("\nData Types:\n", data_types)
```

```python
import pandas as pd

# 1. Reading data from different formats
csv_file = "titanic.csv"  # Replace with your CSV file path
# xls_file = "titanic.xlsx"  # Replace with your Excel file path

# Load data
df_csv = pd.read_csv(csv_file)
# df_xls = pd.read_excel(xls_file)

# Display the first few rows of each dataset
print("Data from CSV:")
print(df_csv.head())

print("\nData from Excel:")
# print(df_xls.head())
# 2. Indexing and selecting data
# Select specific columns
selected_columns = df_csv[['Name', 'Age', 'Survived']]
print("\nSelected Columns:")
print(selected_columns.head())

# Select rows based on a condition (e.g., passengers who survived)
survived = df_csv[df_csv['Survived'] == 1]
print("\nPassengers who survived:")
print(survived.head())

# 3. Sorting data
sorted_data = df_csv.sort_values(by='Age', ascending=True)  # Sort by Age
print("\nSorted Data by Age:")
print(sorted_data.head())

# 4. Describe attributes of data
description = df_csv.describe()
print("\nSummary Statistics:")
print(description)

# 5. Check data types of each column
data_types = df_csv.dtypes
print("\nData Types:")
print(data_types)
```

*2.

```python
import pandas as pd

# Step 1: Load Telecom_Churn dataset
telecom_data = pd.read_csv('./Telecom_churn.csv')

# Step 2: Compute summary statistics for each feature

# Select only numeric columns
numeric_data = telecom_data.select_dtypes(include='number')

# 1. Minimum Value
min_values = numeric_data.min()  # Apply min() to numeric data only
print("Minimum values:\n", min_values)
```

```
# 2. Maximum Value
max_values = numeric_data.max()  # Apply max() to numeric data only
print("\nMaximum values:\n", max_values)

# 3. Mean Value
mean_values = numeric_data.mean()  # Mean for numeric columns
print("\nMean values:\n", mean_values)

# 4. Range (Max - Min) for each feature
range_values = max_values - min_values  # Range calculation for numeric columns
print("\nRange values:\n", range_values)

# 5. Standard Deviation
std_dev = numeric_data.std()  # Standard deviation for numeric columns
print("\nStandard Deviation:\n", std_dev)

# 6. Variance (Square of Standard Deviation)
variance = numeric_data.var()  # Variance for numeric columns
print("\nVariance:\n", variance)

# 7. Percentiles (25th, 50th, and 75th percentiles)
percentiles = numeric_data.quantile([0.25, 0.5, 0.75])  # Percentiles for numeric
columns
print("\nPercentiles (25th, 50th, 75th):\n", percentiles)
```

*3.

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

# Load the House_Price Prediction dataset
house_data = pd.read_csv('House_data.csv')  # Change the path if needed

# Check the first few rows
print(house_data.head())

# Standard Deviation for each numeric feature
std_dev = house_data.std(numeric_only=True)
print("Standard Deviation for each feature:\n", std_dev)

# Variance for each numeric feature
variance = house_data.var(numeric_only=True)
```

```python
print("\nVariance for each feature:\n", variance)

# Percentiles (25th, 50th, and 75th) for each numeric feature
percentiles = house_data.quantile([0.25, 0.5, 0.75], numeric_only=True)
print("\nPercentiles (25th, 50th, 75th) for each feature:\n", percentiles)

# Set up the plotting environment
plt.figure(figsize=(15, 12))

# Loop through each numeric feature and plot its histogram
for i, column in enumerate(house_data.select_dtypes(include='number').columns,
1):
    plt.subplot(4, 4, i)  # Adjust the number of rows and columns based on the
number of features
    sns.histplot(house_data[column], kde=True, bins=30)  # Histogram with kernel
density estimate
    plt.title(f'Histogram of {column}')
    plt.xlabel(column)
    plt.ylabel('Frequency')

# Display the histograms
plt.tight_layout()  # Adjust subplots to fit in the figure area
plt.show()
```

*4.

```python
import math
import pandas as pd

# Load the dataset
df = pd.read_csv('./Lipstick.csv')

# Preprocess Data (Convert categorical variables to numeric values)
df['Age'] = df['Age'].map({'<21': 0, '21-35': 1, '>35': 2})
df['Income'] = df['Income'].map({'Low': 0, 'Medium': 1, 'High': 2})
df['Gender'] = df['Gender'].map({'Male': 0, 'Female': 1})
df['Ms'] = df['Ms'].map({'Single': 0, 'Married': 1})
df['Buys'] = df['Buys'].map({'No': 0, 'Yes': 1})

# Convert DataFrame to list of dictionaries
data = df.to_dict(orient='records')

# Calculate entropy
def entropy(data, target_attr):
    values = [row[target_attr] for row in data]
    freq = {val: values.count(val) for val in set(values)}
    total = len(values)
    return -sum((freq[val] / total) * math.log2(freq[val] / total) for val in
freq)

# Calculate information gain
def info_gain(data, attr, target_attr):
    total_entropy = entropy(data, target_attr)
    attr_values = set(row[attr] for row in data)
    weighted_entropy = 0
    for val in attr_values:
        subset = [row for row in data if row[attr] == val]
        weighted_entropy += (len(subset) / len(data)) * entropy(subset,
target_attr)
    return total_entropy - weighted_entropy

# Find the root node
target = "Buys"
attributes = ["Age", "Income", "Gender", "Ms"]  # Match the exact column names
gains = {attr: info_gain(data, attr, target) for attr in attributes}
root_node = max(gains, key=gains.get)

print("Root Node:", root_node)
```

*5. *6. *7. *8.

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier

# Load the dataset
df = pd.read_csv('./Lipstick.csv')

# Preprocess Data (Convert categorical variables to numeric values)
df['Age'] = df['Age'].map({'<21': 0, '21-35': 1, '>35': 2})
df['Income'] = df['Income'].map({'Low': 0, 'Medium': 1, 'High': 2})
df['Gender'] = df['Gender'].map({'Male': 0, 'Female': 1})
df['Ms'] = df['Ms'].map({'Single': 0, 'Married': 1})
df['Buys'] = df['Buys'].map({'No': 0, 'Yes': 1})

# Define features and target
X = df[['Age', 'Income', 'Gender', 'Ms']]
```

```
y = df['Buys']

# Create and train the decision tree classifier
clf = DecisionTreeClassifier()
clf.fit(X, y)


# Test data for prediction: Age < 21, Income = Low, Gender = Female, Marital
Status = Married
test_data = [[0, 0, 1, 1]]  # Mapping for Age < 21, Income = Low, Gender =
Female, Ms = Married   //only change this line for diffrent practical

prediction = clf.predict(test_data)

# Print the prediction (0 = No, 1 = Yes)
if prediction[0] == 1:
    print("Prediction: Yes, the customer will buy the lipstick.")
else:
    print("Prediction: No, the customer will not buy the lipstick.")


#practical-6
# Test data for prediction: Age > 35, Income = Medium, Gender = Female, Marital
Status = Married
# test_data = [[2, 1, 1, 1]]  # Mapping for Age > 35, Income = Medium, Gender =
Female, Ms = Married


#practical-7
# Test data: [Age > 35, Income = Medium, Gender = Female, Marital Status =
Married]
# test_data = [[2, 1, 1, 1]]  # Corresponds to: Age > 35, Income = Medium, Gender
= Female, Marital Status = Married

#practical-8

# # Test data: [Age = 21-35, Income = Low, Gender = Male, Marital Status =
Married]
# test_data = [[1, 0, 0, 1]]  # Corresponds to: Age = 21-35, Income = Low, Gender
= Male, Marital Status = Married
```

*9. *10.

```
import numpy as np
```

```python
# Given points
points = {
    "P1": [0.1, 0.6],
    "P2": [0.15, 0.71],
    "P3": [0.08, 0.9],
    "P4": [0.16, 0.85],
    "P5": [0.2, 0.3],
    "P6": [0.25, 0.5],
    "P7": [0.24, 0.1],
    "P8": [0.3, 0.2],
}

# Initial centroids
m1 = np.array(points["P1"])  # Cluster 1
m2 = np.array(points["P8"])  # Cluster 2

# Function to compute Euclidean distance
def euclidean_distance(point, centroid):
    return np.sqrt(np.sum((point - centroid) ** 2))

# Assign points to clusters
clusters = {"C1": [], "C2": []}

for point_name, point_coords in points.items():
    point = np.array(point_coords)
    dist_to_m1 = euclidean_distance(point, m1)
    dist_to_m2 = euclidean_distance(point, m2)

    # Assign point to the nearest cluster
    if dist_to_m1 < dist_to_m2:
        clusters["C1"].append(point_name)
    else:
        clusters["C2"].append(point_name)

# Update centroids
def compute_centroid(cluster_points):
    cluster_coords = [points[point] for point in cluster_points]
    return np.mean(cluster_coords, axis=0)

new_m1 = compute_centroid(clusters["C1"])
new_m2 = compute_centroid(clusters["C2"])

# Answers to the questions
P6_cluster = "C1" if "P6" in clusters["C1"] else "C2"
```

```python
population_C2 = len(clusters["C2"])

# Print Results
print(f"Clusters: {clusters}")
print(f"P6 belongs to: {P6_cluster}")
print(f"Population of Cluster C2: {population_C2}")
print(f"Updated m1: {new_m1}")
print(f"Updated m2: {new_m2}")



# practical-10
import numpy as np

# Points
points = {
    "P1": [2, 10],
    "P2": [2, 5],
    "P3": [8, 4],
    "P4": [5, 8],
    "P5": [7, 5],
    "P6": [6, 4],
    "P7": [1, 2],
    "P8": [4, 9]
}

# Initial centroids
centroids = {
    "C1": points["P1"],  # m1
    "C2": points["P4"],  # m2
    "C3": points["P7"]   # m3
}

# Function to calculate Euclidean distance
def euclidean_distance(p1, p2):
    return np.sqrt(np.sum((np.array(p1) - np.array(p2))**2))

# Perform clustering
def k_means(points, centroids):
    clusters = {c: [] for c in centroids.keys()}

    # Step 1: Assign points to the closest centroid
    for point_name, point in points.items():
        distances = {c: euclidean_distance(point, centroids[c]) for c in
centroids.keys()}
```

```python
        closest_centroid = min(distances, key=distances.get)
        clusters[closest_centroid].append(point_name)

    # Step 2: Update centroids
    for c in centroids.keys():
        if clusters[c]:  # Avoid division by zero
            cluster_points = [points[p] for p in clusters[c]]
            centroids[c] = np.mean(cluster_points, axis=0).tolist()

    return clusters, centroids

# Perform one iteration of k-means
clusters, updated_centroids = k_means(points, centroids)

# Results
print("Cluster Assignments:", clusters)
print("Updated Centroids:", updated_centroids)

# Answers to the questions
# 1. Which cluster does P6 belong to?
p6_cluster = next(c for c, ps in clusters.items() if "P6" in ps)
print(f"1] P6 belongs to: {p6_cluster}")

# 2. Population of cluster around C3
c3_population = len(clusters["C3"])
print(f"2] Population around C3: {c3_population}")

# 3. Updated values of m1, m2, m3
print(f"3] Updated values:")
print(f"   m1 (C1): {updated_centroids['C1']}")
print(f"   m2 (C2): {updated_centroids['C2']}")
print(f"   m3 (C3): {updated_centroids['C3']}")


# other distance matrice used for k-mean
# 1 Euclidean Distance
# Measures the straight-line distance between points.

# 2 Manhattan Distance
# Measures the sum of absolute differences along each dimension.
#  ∑ |xi-yi|

# 3 Cosine Distance
# Measures the angle between two vectors (used as 1–cosine similarity).
# Good for text clustering or data with high dimensionality.
```

*11.

```python
import pandas as pd
import matplotlib.pyplot as plt

# Load the dataset
file_path = "IRIS.csv"  # Replace with the correct file path
iris_data = pd.read_csv(file_path)

# 1. List features and their types
print("Features and their Types:")
print(iris_data.dtypes)

# 2. Display the first few rows of the dataset
print("\nFirst few rows of the dataset:")
print(iris_data.head())

# 3. Create histograms for numeric features
numeric_features = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width']

# Set up the plotting environment
plt.figure(figsize=(12, 8))

for i, feature in enumerate(numeric_features, 1):
    plt.subplot(2, 2, i)  # Create a 2x2 grid of subplots
    plt.hist(iris_data[feature], bins=15, color='skyblue', edgecolor='black')
    plt.title(f'Histogram of {feature}')
    plt.xlabel(feature)
    plt.ylabel('Frequency')

plt.tight_layout()
plt.show()
```

*12.

```python
import pandas as pd
import matplotlib.pyplot as plt

# Load the dataset
file_path = "IRIS.csv"  # Replace with the correct file path
iris_data = pd.read_csv(file_path)

# 1. Create box plots for numeric features
numeric_features = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width']
```

```
# Set up the plotting environment
plt.figure(figsize=(12, 8))

for i, feature in enumerate(numeric_features, 1):
    plt.subplot(2, 2, i)  # Create a 2x2 grid of subplots
    plt.boxplot(iris_data[feature], vert=True, patch_artist=True,
boxprops=dict(facecolor='lightblue'))
    plt.title(f'Box Plot of {feature}')
    plt.ylabel(feature)

plt.tight_layout()
plt.show()

# 2. Identify potential outliers
print("\nPotential Outliers Analysis:")
for feature in numeric_features:
    q1 = iris_data[feature].quantile(0.25)  # 25th percentile
    q3 = iris_data[feature].quantile(0.75)  # 75th percentile
    iqr = q3 - q1  # Interquartile range
    lower_bound = q1 - 1.5 * iqr
    upper_bound = q3 + 1.5 * iqr
    outliers = iris_data[(iris_data[feature] < lower_bound) | (iris_data[feature]
> upper_bound)]
    print(f"{feature}: {len(outliers)} outliers detected.")
    if not outliers.empty:
        print(outliers[[feature]])
```

*13.

```
# Import necessary libraries
import pandas as pd

# Load the dataset
file_path = 'Covid Vaccine Statewise.csv'  # Path to the dataset
data = pd.read_csv(file_path)

# a. Describe the dataset
print("Dataset Information:")
print(data.info())  # Prints detailed info about columns, data types, and missing
values
print("\nFirst 5 rows of the dataset:")
print(data.head())  # Prints the first 5 rows of the dataset
```

```
# b. Number of persons state-wise vaccinated for the first dose
statewise_first_dose = data.groupby('State')['First Dose
Administered'].sum().dropna()
print("\nState-wise First Dose Administered:")
print(statewise_first_dose)

# c. Number of persons state-wise vaccinated for the second dose
statewise_second_dose = data.groupby('State')['Second Dose
Administered'].sum().dropna()
print("\nState-wise Second Dose Administered:")
print(statewise_second_dose)
```

*14.

```
# Import necessary libraries
import pandas as pd

# Load the dataset
file_path = 'Covid Vaccine Statewise.csv'  # Path to the dataset
data = pd.read_csv(file_path)

# A. Describe the dataset
print("Dataset Information:")
print(data.info())  # Prints detailed info about columns, data types, and missing
values
print("\nFirst 5 rows of the dataset:")
print(data.head())  # Prints the first 5 rows of the dataset

# B. Total number of males vaccinated
total_males_vaccinated = data['Male (Doses Administered)'].sum(skipna=True)
print("\nTotal number of males vaccinated:")
print(total_males_vaccinated)

# C. Total number of females vaccinated
total_females_vaccinated = data['Female (Doses Administered)'].sum(skipna=True)
print("\nTotal number of females vaccinated:")
print(total_females_vaccinated)
```

*15.

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

# Load the Titanic dataset
file_path = 'Titanic.csv'  # Replace with your file path
titanic_data = pd.read_csv(file_path)

# Display dataset information
print(titanic_data.info())
print(titanic_data.head())

# Visualize survival count by gender
plt.figure(figsize=(8, 5))
sns.countplot(data=titanic_data, x="Sex", hue="Survived", palette="Set2")
plt.title("Survival Count by Gender")
plt.ylabel("Count")
plt.xlabel("Gender")
plt.legend(title="Survived", loc="upper right")
plt.show()

# Visualize survival count by passenger class
plt.figure(figsize=(8, 5))
sns.countplot(data=titanic_data, x="Pclass", hue="Survived", palette="Set2")
plt.title("Survival Count by Passenger Class")
plt.ylabel("Count")
plt.xlabel("Passenger Class")
plt.legend(title="Survived", loc="upper right")
plt.show()

# Visualize survival count by embarkation point
plt.figure(figsize=(8, 5))
sns.countplot(data=titanic_data, x="Embarked", hue="Survived", palette="Set2")
plt.title("Survival Count by Embarkation Point")
plt.ylabel("Count")
plt.xlabel("Embarkation Point")
plt.legend(title="Survived", loc="upper right")
plt.show()

# Distribution of age by survival status
plt.figure(figsize=(8, 5))
sns.kdeplot(data=titanic_data, x="Age", hue="Survived", fill=True, alpha=0.5,
palette="Set2")
```

```
plt.title("Age Distribution by Survival Status")
plt.xlabel("Age")
plt.ylabel("Density")
plt.show()

# Boxplot of fare by passenger class and survival status
plt.figure(figsize=(8, 5))
sns.boxplot(data=titanic_data, x="Pclass", y="Fare", hue="Survived",
palette="Set2")
plt.title("Fare Distribution by Passenger Class and Survival Status")
plt.ylabel("Fare")
plt.xlabel("Passenger Class")
plt.legend(title="Survived", loc="upper right")
plt.show()
```

*16.

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

# Load the Titanic dataset from Seaborn's inbuilt dataset
titanic_data = sns.load_dataset('titanic')

# Check the first few rows to confirm the dataset is loaded correctly
print(titanic_data.head())

# Plot a histogram for the 'fare' column
plt.figure(figsize=(8, 5))
sns.histplot(titanic_data['fare'], bins=30, kde=True, color='blue')
plt.title("Distribution of Ticket Prices (Fare)")
plt.xlabel("Fare")
plt.ylabel("Frequency")
plt.show()
```

*17. *22.

```
# these values for pr no- 17
TP = 1  # True Positives
FP = 1  # False Positives
FN = 8  # False Negatives
TN = 90 # True Negatives
```

```
# these values for pr no- 22

# TP = 1  # True Positives
# FP = 1  # False Positives
# FN = 8  # False Negatives
# TN = 90 # True Negatives


# 1. Accuracy
accuracy = (TP + TN) / (TP + TN + FP + FN)

# 2. Error Rate
error_rate = (FP + FN) / (TP + TN + FP + FN)

# 3. Precision
precision = TP / (TP + FP)

# 4. Recall (Sensitivity)
recall = TP / (TP + FN)

# Print the results
print(f"Accuracy: {accuracy:.4f}")
print(f"Error Rate: {error_rate:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
```

*20. *21.

```
import pandas as pd
import numpy as np

# Load the dataset
iris = pd.read_csv("./IRIS.csv")

# Extract features
X = iris.iloc[:, :-1].values  # Excluding the species column

# Number of clusters
k = 3

# Number of clusters for practcal-21 (only this value is changed in pr-21)
# k = 4
```

```python
# Initialize cluster centers randomly
np.random.seed(42)
initial_indices = np.random.choice(len(X), k, replace=False)
centroids = X[initial_indices]

# Function to calculate Euclidean distance
def euclidean_distance(a, b):
    return np.sqrt(np.sum((a - b) ** 2, axis=1))

# Perform K-Means for at least 10 iterations
max_iterations = 10
for iteration in range(max_iterations):
    # Step 1: Assign points to the nearest cluster center
    clusters = [[] for _ in range(k)]
    for point in X:
        distances = [np.linalg.norm(point - centroid) for centroid in centroids]
        cluster_index = np.argmin(distances)
        clusters[cluster_index].append(point)

    # Step 2: Update the centroids
    new_centroids = []
    for cluster in clusters:
        new_centroids.append(np.mean(cluster, axis=0))
    new_centroids = np.array(new_centroids)

    # Check for convergence (if centroids do not change)
    if np.allclose(centroids, new_centroids, atol=1e-6):
        print(f"Convergence reached at iteration {iteration + 1}")
        break
    centroids = new_centroids

# Final centroids after 10 iterations
print("Final Cluster Centers:")
for i, centroid in enumerate(centroids):
    print(f"Cluster {i + 1} center: {centroid}")


# other distance matrice used for k-mean
# 1 Euclidean Distance
# Measures the straight-line distance between points.

# 2 Manhattan Distance
# Measures the sum of absolute differences along each dimension.
#  ∑ |xi-yi|
# 3 Cosine Distance
# Measures the angle between two vectors (used as 1-cosine similarity).
# Good for text clustering or data with high dimensionality.
```

*24.

```
import pandas as pd

# Load the Iris dataset from the CSV file
df = pd.read_csv('./IRIS.csv')

# Check for unique values in each column
print("Unique values in each column:")
print(df.nunique())

# Check data types of each column
print("\nData types of each column:")
print(df.dtypes)

# Convert data types (example: convert 'sepal_length' to float32)
df['sepal_length'] = df['sepal_length'].astype('float32')

# Data type of each column after conversion
print("\nData types after conversion:")
print(df.dtypes)

# Check for missing values
print("\nMissing values in each column:")
print(df.isnull().sum())

# Fill missing values with the mean of the numeric columns
numeric_cols = df.select_dtypes(include=['float64', 'float32']).columns
df[numeric_cols] = df[numeric_cols].fillna(df[numeric_cols].mean())

# Display the cleaned data
print("\nData after filling missing values:")
print(df.head())
```

*25.

```
import pandas as pd
```

```python
# Step 1: Load the Lipstick dataset
df = pd.read_csv('./Lipstick.csv')

# Step 2: Check the first few rows of the dataset
print("First few rows of the dataset:")
print(df.head())

# Step 3: Data Cleaning

# 3.1 Check for missing values
print("\nMissing values in the dataset:")
print(df.isnull().sum())

# If there are any missing values, we'll fill them (using the mode for
categorical variables)
df.fillna(df.mode().iloc[0], inplace=True)

# After filling missing values, let's check again
print("\nMissing values after cleaning:")
print(df.isnull().sum())

# 3.2 Check for duplicates
print("\nChecking for duplicate rows:")
print(f"Number of duplicates: {df.duplicated().sum()}")

# Drop duplicates if they exist
df.drop_duplicates(inplace=True)

# 3.3 Verify no duplicates are left
print("\nAfter dropping duplicates:")
print(f"Number of duplicates: {df.duplicated().sum()}")

# Step 4: Data Transformation (Convert categorical variables to numeric values)

df['Age'] = df['Age'].map({'<21': 0, '21-35': 1, '>35': 2})
df['Income'] = df['Income'].map({'Low': 0, 'Medium': 1, 'High': 2})
df['Gender'] = df['Gender'].map({'Male': 0, 'Female': 1})
df['Ms'] = df['Ms'].map({'Single': 0, 'Married': 1})
df['Buys'] = df['Buys'].map({'No': 0, 'Yes': 1})

# Step 5: Check the transformed dataset
print("\nTransformed dataset:")
print(df.head())

# Step 6: Basic Statistical Summary
```

```python
print("\nStatistical summary of the dataset:")
print(df.describe())

# Optional: Check for correlations (if relevant)
print("\nCorrelation between numeric variables:")
print(df.corr())

# Step 7: Save the cleaned and transformed dataset
df.to_csv('./Transformed_Lipstick.csv', index=False)
```

*18.

```python
import pandas as pd

# Load the dataset
house_data = pd.read_csv('./House_data.csv')

# Cleaning the 'price' column by removing non-numeric characters and converting
to numeric
house_data['price_cleaned'] = (
    house_data['price']
    .str.replace('[^0-9]', '', regex=True)
    .astype(float)
)

# Calculate each statistic separately
mean_price = house_data.groupby('district')['price_cleaned'].mean()
median_price = house_data.groupby('district')['price_cleaned'].median()
min_price = house_data.groupby('district')['price_cleaned'].min()
max_price = house_data.groupby('district')['price_cleaned'].max()
std_price = house_data.groupby('district')['price_cleaned'].std()

# Combine the statistics into a single DataFrame
summary_stats_separate = pd.DataFrame({
    'mean': mean_price,
    'median': median_price,
    'min': min_price,
    'max': max_price,
    'std': std_price
})

# Display the combined statistics
print(summary_stats_separate)
```

*19.

```
import pandas as pd

# Load the Iris dataset
iris_data = pd.read_csv('./IRIS.csv')  # Ensure the path to your dataset is
correct

# Check column names to verify the structure
print("Column names:", iris_data.columns)

# Calculate statistics grouped by the corrected column name 'species'
mean_values = iris_data.groupby('species').mean()
std_values = iris_data.groupby('species').std()
min_values = iris_data.groupby('species').min()
max_values = iris_data.groupby('species').max()
percentile_25 = iris_data.groupby('species').quantile(0.25)
median_values = iris_data.groupby('species').median()
percentile_75 = iris_data.groupby('species').quantile(0.75)

# Aggregate all statistics into a single DataFrame
species_stats = pd.concat(
    [mean_values, std_values, min_values, max_values, percentile_25,
median_values, percentile_75],
    keys=['Mean', 'Std Dev', 'Min', 'Max', '25th Percentile', 'Median', '75th
Percentile'],
    axis=1
)

# Display the combined statistics
print(species_stats)
```