**Q1.  Loading and Inspecting Data:**

·         Load various data formats (CSV, Excel, JSON) into Pandas DataFrames.

·         Explore DataFrame attributes like shape, columns, dtypes, head, tail, info, describe.

·         Practice selecting columns and rows using different methods (indexing, slicing, loc, iloc).

import pandas as pd

# Load a CSV file into a DataFrame

df\_csv = pd.read\_csv('path\_to\_file.csv')

# Inspect the DataFrame

print(df\_csv.head()) # Display first 5 rows

# Load an Excel file into a DataFrame (specify the sheet name if there are multiple sheets)

df\_excel = pd.read\_excel('path\_to\_file.xlsx', sheet\_name='Sheet1')

# Inspect the DataFrame

print(df\_excel.head()) # Display first 5 rows

# Load a JSON file into a DataFrame

df\_json = pd.read\_json('path\_to\_file.json')

# Inspect the DataFrame

print(df\_json.head()) # Display first 5 rows

# Get the shape of the DataFrame (rows, columns)

print(df\_csv.shape) # Example output: (100, 5) - 100 rows, 5 columns

# Get the column names of the DataFrame

print(df\_csv.columns)

# Get the data types of each column

print(df\_csv.dtypes)

# Display the first 5 rows

print(df\_csv.head())

# Display the last 5 rows

print(df\_csv.tail())

# Get concise summary of the DataFrame

print(df\_csv.info())

# Get summary statistics for numerical columns

print(df\_csv.describe())

# Select a single column (returns a Series)

print(df\_csv['column\_name'])

# Select multiple columns (returns a DataFrame)

print(df\_csv[['column\_name1', 'column\_name2']])

# Select a specific row by index position (e.g., row at index 0)

print(df\_csv.iloc[0])

# Select multiple rows (e.g., rows 0 to 4)

print(df\_csv.iloc[0:5])

# Select a row by its label (e.g., if the index is labeled)

print(df\_csv.loc[0])

# Select a range of rows by label

print(df\_csv.loc[0:4]) # Select rows from label 0 to label 4

# Select rows by label and specific columns

print(df\_csv.loc[0:4, ['column\_name1', 'column\_name2']])

# Select the first 5 rows and the first 2 columns

print(df\_csv.iloc[0:5, 0:2])

# Load the data

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

# Explore the DataFrame

print("Shape of DataFrame:", df.shape) # Output: (4, 4)

print("Columns:", df.columns) # Output: Index(['ID', 'Name', 'Age', 'City'], dtype='object')

print("Data types:", df.dtypes) # Output: ID int64, Name object, Age int64, City object

print("First 2 rows:", df.head(2)) # Output: First two rows

# Select a column

print("Name column:", df['Name'])

# Select multiple columns

print("Name and Age columns:", df[['Name', 'Age']])

# Select a specific row by index

print("Row at index 2:", df.iloc[2])

# Select rows 0 to 2 and columns 1 and 2

print("Rows 0-2, columns 1 and 2:", df.iloc[0:3, 1:3])

**Q2. Data Cleaning and Preparation**

* Identify missing values using isnull and isna.
* Handle missing values using fillna, dropna, interpolation.
* Apply scaling techniques (min-max, z-score) to numerical columns.
* Create dummy variables for categorical columns.

import pandas as pd

# Example DataFrame with missing values

data = {'A': [1, 2, None, 4], 'B': [None, 2, 3, 4], 'C': [1, None, 3, None]}

df = pd.DataFrame(data)

# Identify missing values using isnull()

print(df.isnull())

# Alternatively, you can use isna() (it works the same as isnull())

print(df.isna())

# Fill missing values with a specific value

df\_filled = df.fillna(0) # Replace NaN with 0

print(df\_filled)

# Fill missing values with the mean of each column

df\_filled\_mean = df.fillna(df.mean()) # Fill NaN with the mean of each column

print(df\_filled\_mean)

# Drop rows with missing values

df\_dropped = df.dropna() # Drops rows with any NaN values

print(df\_dropped)

# Drop columns with missing values

df\_dropped\_columns = df.dropna(axis=1) # Drops columns with any NaN values

print(df\_dropped\_columns)

from sklearn.preprocessing import MinMaxScaler

# Example DataFrame

df = pd.DataFrame({'A': [1, 2, 3, 4], 'B': [5, 6, 7, 8]})

# Initialize MinMaxScaler

scaler = MinMaxScaler()

# Apply Min-Max scaling to numerical columns

df\_scaled = pd.DataFrame(scaler.fit\_transform(df), columns=df.columns)

print(df\_scaled)

from sklearn.preprocessing import StandardScaler

# Example DataFrame

df = pd.DataFrame({'A': [1, 2, 3, 4], 'B': [5, 6, 7, 8]})

# Initialize StandardScaler

scaler = StandardScaler()

# Apply Z-Score scaling to numerical columns

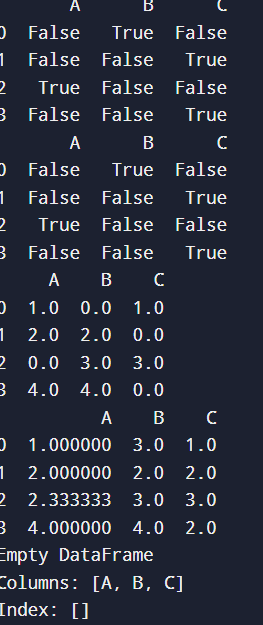
df\_standardized = pd.DataFrame(scaler.fit\_transform(df), columns=df.columns)

print(df\_standardized)

# Example DataFrame with categorical data

df = pd.DataFrame({'City': ['New York', 'Los Angeles', 'Chicago', 'New York']})

# Create dummy variables for the 'City' column

df\_dummies = pd.get\_dummies(df, columns=['City'])

print(df\_dummies

**Aggregation and Grouping:**

Calculate summary statistics (mean, median, count, etc.) using groupby.

Create pivot tables for data summarization.

Combine DataFrames using concat, merge, and join.

Practice different join types (inner, outer, left, right).

import pandas as pd

# Example DataFrame

data = {

'City': ['New York', 'Los Angeles', 'Chicago', 'New York', 'Chicago', 'Los Angeles'],

'Temperature': [75, 85, 60, 70, 65, 80],

'Rainfall': [0.1, 0.2, 0.3, 0.1, 0.4, 0.2]

}

df = pd.DataFrame(data)

# Group by 'City' and calculate mean, median, and count for numerical columns

grouped = df.groupby('City').agg({

'Temperature': ['mean', 'median', 'count'],

'Rainfall': ['mean', 'median', 'count']

})print(grouped)

