Assignment-02(29-08-24):

import pandas as pd

# Exercise-1: Creating dataframes from scratch

data={

'Product': ['Laptop', 'Mouse', 'Monitor', 'Keyboard', 'Phone'],

'Category': ['Electronics', 'Accessories', 'Electronics', 'Accessories', 'Electronics'],

'Price': [80000, 1500, 20000, 3000, 40000],

'Quantity': [10, 100, 50, 75, 30]

}

# Create a dataframe

df=pd.DataFrame(data)

print(df)

# Exercise-2: Basic dataframe operations

# 1. Display the first 3 rows of the DataFrame

print(df.head(3))

# 2. Display the column names and index of the DataFrame

print(df.columns)

print(df.index)

# 3. Display a summary of statistics for the numeric columns

print(df.describe())

# Exercise-3: Selecting Data

# 1. Select and display the "Product" and "Price" columns

print(df[['Product', 'Price']])

# 2. Select rows where the "Category" is "Electronics"

print(df[df['Category'] == 'Electronics'])

# Exercise-4: Filtering data

# 1. Filter the DataFrame to display only the products with a price greater than 10,000

print(df[df['Price'] > 10000])

# 2. Filter the DataFrame to show only products that belong to the "Accessories" category

# and have a quantity greater than 50

print(df[(df['Category'] == 'Accessories') & (df['Quantity'] > 50)])

# Exercise-5: Adding and removing columns

# 1. Add a new column "Total Value" which is Price \* Quantity

df['Total Value'] = df['Price'] \* df['Quantity']

print(df)

# 2. Drop the "Category" column from the DataFrame

df\_dropped = df.drop(columns=['Category'])

print(df\_dropped)

# Exercise-6: Sorting Data

# 1. Sort the DataFrame by "Price" in descending order

df\_sorted\_price = df.sort\_values(by='Price', ascending=False)

print(df\_sorted\_price)

# 2. Sort the DataFrame by "Quantity" in ascending order, then by "Price" in descending order

df\_sorted\_multi = df.sort\_values(by=['Quantity', 'Price'], ascending=[True, False])

print(df\_sorted\_multi)

# Exercise-7: Grouping data

# 1. Group by "Category" and calculate the total quantity for each category

category\_quantity = df.groupby('Category')['Quantity'].sum()

print(category\_quantity)

# 2. Group by "Category" and calculate the average price for each category

category\_avg\_price = df.groupby('Category')['Price'].mean()

print(category\_avg\_price)

# Exercise-8: Handling missing data

# 1. Introduce some missing values in the "Price" column

df.loc[2, 'Price'] = None

df.loc[4, 'Price'] = None

print(df)

# 2. Fill the missing values with the mean price of the available products

df['Price'].fillna(df['Price'].mean(), inplace=True)

print(df)

# 3. Drop any rows where the "Quantity" is less than 50

df\_filtered = df[df['Quantity'] >= 50]

print(df\_filtered)

# Exercise-9: Apply Custom Functions

# 1. Apply a custom function to the "Price" column that increases all prices by 5%

df['Price'] = df['Price'].apply(lambda x: x \* 1.05)

print(df)

# 2. Create a new column "Discounted Price" that reduces the original price by 10%

df['Discounted Price'] = df['Price'] \* 0.90

print(df)

# Exercise-10: Merging Dataframes

# Create another DataFrame

suppliers = pd.DataFrame({

'Product': ['Laptop', 'Mouse', 'Monitor', 'Keyboard', 'Phone'],

'Supplier': ['Supplier A', 'Supplier B', 'Supplier A', 'Supplier C', 'Supplier B']

})

# Merge with the original DataFrame

df\_merged = pd.merge(df, suppliers, on='Product')

print(df\_merged)

# Exercise-11: Pivot tables

# Create a pivot table that shows the total quantity of products

# for each category and product combination

pivot\_table = df.pivot\_table(values='Quantity', index='Category', columns='Product', aggfunc='sum')

print(pivot\_table)

# Exercise-12: Concatenating Dataframes

# Data for two stores

store1 = pd.DataFrame({

'Product': ['Laptop', 'Mouse', 'Monitor'],

'Price': [80000, 1500, 20000],

'Quantity': [10, 100, 50]

})

store2 = pd.DataFrame({

'Product': ['Keyboard', 'Phone', 'Tablet'],

'Price': [3000, 40000, 25000],

'Quantity': [75, 30, 20]

})

# Concatenate the DataFrames

combined\_inventory = pd.concat([store1, store2], ignore\_index=True)

print(combined\_inventory)

# Exercise-13: Working with dates

from datetime import datetime,timedelta

# Create a DataFrame with a "Date" column that contains the last 5 days starting from today

dates = [datetime.today() - timedelta(days=i) for i in range(5)]

sales = [2500, 3400, 4100, 2900, 4700]

df\_dates = pd.DataFrame({'Date': dates, 'Sales': sales})

print(df\_dates)

# Find the total sales for all days combined

total\_sales = df\_dates['Sales'].sum()

print(f'Total Sales: {total\_sales}')

# Exercise-14: Reshaping data with melt

# Create a dataframe

df\_sales=pd.DataFrame({

'Product': ['Laptop','Phone','Monitor'],

'Region': ['North', 'South', 'West'],

'Q1\_Sales': [15000, 30000, 25000],

'Q2\_Sales': [18000, 32000, 27000]

})

# Using pd.melt() to reshape the DataFrame

df\_melted = pd.melt(df\_sales,

id\_vars=['Product', 'Region'],

value\_vars=['Q1\_Sales', 'Q2\_Sales'],

var\_name='Quarter', value\_name='Sales')

print(df\_melted)

# Exercise-15- Reading and writing data

df\_csv=pd.read\_csv("Products.csv")

df\_csv['Discount']=df\_csv['Price']\*0.10

# write back to a new CSV file

df\_csv.to\_csv('updated\_products.csv', index=False)

# Exercise-16: Renaming columns

# Original DataFrame

df\_rename = pd.DataFrame({

'Prod': ['Laptop', 'Mouse', 'Monitor'],

'Cat': ['Electronics', 'Accessories', 'Electronics'],

'Price': [80000, 1500, 20000],

'Qty': [10, 100, 50]

})

# Rename columns

df\_rename.columns = ['Product', 'Category', 'Price', 'Quantity']

print(df\_rename)

# Exercise-17: Create a multi-index dataframe

# Create a MultiIndex DataFrame

index = pd.MultiIndex.from\_tuples([('Store A', 'Laptop'), ('Store A', 'Mouse'),

('Store B', 'Monitor'), ('Store B', 'Keyboard')],

names=['Store', 'Product'])

df\_multi = pd.DataFrame({

'Price': [80000, 1500, 20000, 3000],

'Quantity': [10, 100, 50, 75]

}, index=index)

print(df\_multi)

# Exercise-18: Resample Time-Series Data

# Create a DataFrame with a range of dates and sales values

dates = pd.date\_range(start='2024-08-01', end='2024-08-30')

sales = [250, 300, 150, 400, 500, 600, 700, 200, 450, 350,

800, 900, 500, 750, 650, 300, 550, 450, 600, 700,

800, 250, 500, 600, 700, 800, 900, 1000, 950, 850]

df\_time = pd.DataFrame({'Date': dates, 'Sales': sales})

# Resample the data to show total sales by week

df\_weekly\_sales = df\_time.set\_index('Date').resample('W').sum()

print(df\_weekly\_sales)

# Exercise-19- Handling\_Duplicates

# Original DataFrame with duplicates

df\_duplicates = pd.DataFrame({

'Product': ['Laptop', 'Mouse', 'Monitor', 'Laptop'],

'Category': ['Electronics', 'Accessories', 'Electronics', 'Electronics'],

'Price': [80000, 1500, 20000, 80000]

})

# Remove duplicate rows based on all columns

df\_no\_duplicates = df\_duplicates.drop\_duplicates()

print(df\_no\_duplicates)

# Remove duplicate rows based on the "Product" column

df\_no\_duplicates\_product = df\_duplicates.drop\_duplicates(subset=['Product'])

print(df\_no\_duplicates\_product)

# Exercise-20: Correlation Matrix

# Create a DataFrame with some numerical data

df\_corr = pd.DataFrame({

'Height': [170, 165, 180, 175],

'Weight': [70, 65, 80, 75],

'Age': [25, 30, 22, 28],

'Income': [50000, 60000, 55000, 62000]

})

# Compute the correlation matrix

corr\_matrix = df\_corr.corr()

print(corr\_matrix)

# Exercise-21: Cumulative Sum and Rolling Windows

# Create a DataFrame with random sales data

dates = pd.date\_range(start='2024-08-01', periods=30)

sales = [200, 300, 400, 250, 150, 350, 450, 500, 550, 600,

650, 700, 750, 800, 850, 900, 950, 1000, 1050, 1100,

1150, 1200, 1250, 1300, 1350, 1400, 1450, 1500, 1550, 1600]

df\_sales = pd.DataFrame({'Date': dates, 'Sales': sales})

# Calculate the cumulative sum

df\_sales['Cumulative Sales'] = df\_sales['Sales'].cumsum()

# Calculate the rolling average of sales over the past 7 days

df\_sales['Rolling Avg'] = df\_sales['Sales'].rolling(window=7).mean()

print(df\_sales)

# Exercise-22: String Operations

# Create a DataFrame with names

df\_names = pd.DataFrame({

'Names': ['Mitali Raj', 'Saina Nehwal', 'Jhulan Goswami']

})

# Split the "Names" column into two separate columns

df\_names[['First Name', 'Last Name']] = df\_names['Names'].str.split(' ', expand=True)

# Convert the "First Name" column to uppercase

df\_names['First Name'] = df\_names['First Name'].str.upper()

print(df\_names)

# Exercise-23: Conditional Selections with np.where

# Create a DataFrame with employee data

df\_employee = pd.DataFrame({

'Employee': ['John', 'Jane', 'Sam', 'Emily'],

'Age': [45, 34, 29, 41],

'Department': ['HR', 'Finance', 'IT', 'Marketing']

})

# Create a new column "Status" using conditional selections

df\_employee['Status'] = ['Senior' if age >= 40 else 'Junior' for age in df\_employee['Age']]

print(df\_employee)

# Exercise-24: Slicing Dataframes

# Create a DataFrame with product data

df\_products = pd.DataFrame({

'Product': ['Laptop', 'Mouse', 'Monitor', 'Keyboard', 'Phone'],

'Category': ['Electronics', 'Accessories', 'Electronics', 'Accessories', 'Electronics'],

'Sales': [100000, 20000, 50000, 15000, 40000],

'Profit': [25000, 5000, 12000, 3000, 10000]

})

# 1. The first 10 rows

print(df\_products.head(10))

# 2. Rows where the "Category" is "Electronics"

print(df\_products[df\_products['Category'] == 'Electronics'])

# 3. "Sales" and "Profit" columns for products with sales greater than 50,000

print(df\_products[df\_products['Sales'] > 50000][['Sales', 'Profit']])

# Exercise-25: Concatenating DataFrames Vertically and Horizontally

# Vertically concatenating DataFrames

df\_storeA = pd.DataFrame({

'Employee': ['John', 'Jane'],

'Age': [45, 34],

'Salary': [50000, 60000]

})

df\_storeB = pd.DataFrame({

'Employee': ['Sam', 'Emily'],

'Age': [29, 41],

'Salary': [55000, 62000]

})

df\_combined = pd.concat([df\_storeA, df\_storeB])

print(df\_combined)

# Horizontally concatenating DataFrames

df\_emp\_dept = pd.DataFrame({

'Employee': ['John', 'Jane', 'Sam', 'Emily'],

'Department': ['HR', 'Finance', 'IT', 'Marketing']

})

df\_emp\_salary = pd.DataFrame({

'Employee': ['John', 'Jane', 'Sam', 'Emily'],

'Salary': [50000, 60000, 55000, 62000]

})

df\_horizontal\_concat = pd.merge(df\_emp\_dept, df\_emp\_salary, on='Employee')

print(df\_horizontal\_concat)

# Exercise-26: Exploding Lists in DataFrame Columns

# Create a DataFrame with a column "Features" that contains lists

df\_features = pd.DataFrame({

'Product': ['Laptop', 'Phone'],

'Features': [['Feature1', 'Feature2'], ['Feature3', 'Feature4']]

})

# Use the explode() method to create a new row for each feature

df\_exploded = df\_features.explode('Features')

print(df\_exploded)

# Exercise-27: Using .map() and .applyMap()

# Create a DataFrame with product data

df\_map = pd.DataFrame({

'Product': ['Laptop', 'Mouse', 'Monitor'],

'Price': [80000, 1500, 20000],

'Quantity': [10, 100, 50]

})

# Use .map() to increase "Price" by 10%

df\_map['Price'] = df\_map['Price'].map(lambda x: x \* 1.10)

# Use .applyMap() to format the numeric values to two decimal places

df\_map = df\_map.applymap(lambda x: f'{x:.2f}' if isinstance(x, (int, float)) else x)

print(df\_map)

# Exercise-28: Combining groupBy() with apply()

# Create a DataFrame with sales data

df\_sales = pd.DataFrame({

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

'Product': ['Laptop', 'Laptop', 'Monitor', 'Monitor'],

'Sales': [100000, 120000, 50000, 55000],

'Profit': [25000, 30000, 12000, 14000]

})

# Group by "City" and apply a custom function to calculate the profit margin (Profit/Sales)

def profit\_margin(group):

return group['Profit'].sum() / group['Sales'].sum()

city\_profit\_margin = df\_sales.groupby('City').apply(profit\_margin)

print(city\_profit\_margin)

# Exercise-29: Creating a DataFrame from Multiple Sources

# DataFrame from CSV

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

# DataFrame from JSON

df\_json = pd.read\_json('products.json')

# DataFrame from a dictionary

data\_dict = {

'Product': ['Laptop', 'Phone', 'Monitor'],

'Category': ['Electronics', 'Electronics', 'Electronics'],

'Price': [80000, 40000, 20000]

}

df\_dict = pd.DataFrame(data\_dict)

# Merge the DataFrames based on a common column

df\_merged = pd.merge(pd.merge(df\_csv, df\_json, on='Product'), df\_dict, on='Product')

print(df\_merged)

# Exercise-30: Dealing with large datasets

import pandas as pd

# Define the size of the DataFrame

n = 1000000

# Creating a large DataFrame with 1 million rows

df\_large = pd.DataFrame({

'Transaction ID': range(n),

'Customer': ['Customer A', 'Customer B', 'Customer C'] \* (n // 3) + ['Customer A'] \* (n % 3),

'Product': ['Laptop', 'Mouse', 'Monitor', 'Keyboard'] \* (n // 4) + ['Laptop'] \* (n % 4),

'Amount': [(i % 5000) + 100 for i in range(n)],

'Date': pd.date\_range(start='2024-01-01', periods=n, freq='min')

})

# Split the DataFrame into chunks and perform analysis on each chunk

chunk\_size = 100000

chunks = [df\_large[i:i+chunk\_size] for i in range(0, n, chunk\_size)]

# Analysis of total sales per chunk

total\_sales = [chunk['Amount'].sum() for chunk in chunks]

combined\_sales = sum(total\_sales)

print(f'Total Sales: {combined\_sales}')