**Assignment 2**  
**### \*\*Exercise 1: Creating DataFrame from Scratch\*\*  
#1. Create a DataFrame with the following columns: `"Product"`, `"Category"`, `"Price"`, and `"Quantity"`. Use the following 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]`  
#2. Print the DataFrame.**  
import pandas as pd  
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]  
}  
df = pd.DataFrame(data)  
print(df)  
  
**### \*\*Exercise 2: Basic DataFrame Operations\*\*  
#1. Display the first 3 rows of the DataFrame.  
#2. Display the column names and index of the DataFrame.  
#3. Display a summary of statistics (mean, min, max, etc.) for the numeric columns in the DataFrame.**  
print(df.head(3))  
  
print("Column Names:", df.columns)  
print("Index:", df.index)  
  
print(df.describe())  
  
**### \*\*Exercise 3: Selecting Data\*\*  
#1. Select and display the `"Product"` and `"Price"` columns.  
#2. Select rows where the `"Category"` is `"Electronics"` and print them.**product\_price\_columns = df[['Product', 'Price']]  
print(product\_price\_columns)  
  
electronics\_rows = df[df['Category'] == 'Electronics']  
print(electronics\_rows)  
  
**### \*\*Exercise 4: Filtering Data\*\*  
#1. Filter the DataFrame to display only the products with a price greater than `10,000`.  
#2. Filter the DataFrame to show only products that belong to the `"Accessories"` category** **and have a quantity greater than `50`.**  
price\_greater = df[df['Price'] > 10000]  
print(price\_greater)  
  
accessories\_df = df[(df['Category'] == 'Accessories') & (df['Quantity'] > 50)]  
print(accessories\_df)  
  
**### \*\*Exercise 5: Adding and Removing Columns\*\*  
#1. Add a new column `"Total Value"` which is calculated by multiplying `"Price"` and `"Quantity"`.  
#2. Drop the `"Category"` column from the DataFrame and print the updated DataFrame.  
# 1. Add a new column "Total Value" which is calculated by multiplying "Price" and "Quantity"**  
df['Total Value'] = df['Price'] \* df['Quantity']  
  
df = df.drop(columns=['Category'])  
print(df)  
  
**### \*\*Exercise 6: Sorting Data\*\*  
#1. Sort the DataFrame by `"Price"` in descending order.  
#2. Sort the DataFrame by `"Quantity"` in ascending order, then by `"Price"` in descending order (multi-level sorting).**df\_sorted\_price= df.sort\_values(by='Price', ascending=False)  
print(df\_sorted\_price)  
  
df\_sorted\_quantity\_and\_price = df.sort\_values(by=['Quantity', 'Price'], ascending=[True, False])  
print(df\_sorted\_quantity\_and\_price)  
  
**### \*\*Exercise 7: Grouping Data\*\*  
#1. Group the DataFrame by `"Category"` and calculate the total quantity for each category.  
#2. Group by `"Category"` and calculate the average price for each category.**df["Category"]=["Electronics", "Accessories", "Electronics", "Accessories", "Electronics"]  
total\_quantity\_by\_category = df.groupby('Category')['Quantity'].sum()  
print(total\_quantity\_by\_category)  
  
average\_price\_by\_category = df.groupby('Category')['Price'].mean()  
print(average\_price\_by\_category)  
 **### \*\*Exercise 8: Handling Missing Data\*\*  
#1. Introduce some missing values in the `"Price"` column by assigning `None` to two rows.  
#2. Fill the missing values with the mean price of the available products.  
#3. Drop any rows where the `"Quantity"` is less than `50`.**  
df.loc[1, 'Price'] = None  
df.loc[3, 'Price'] = None  
print(df)  
  
mean\_price = df['Price'].mean()  
df['Price'] = df['Price'].apply(lambda x: mean\_price if x is None else x)  
print(df)  
  
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%.  
#2. Create a new column `"Discounted Price"` that reduces the original price by 10%.**df['Price'] = df['Price'].apply(lambda x: x \* 1.05)  
print(df)  
  
df['Discounted Price'] = df['Price'] \* 0.90  
**print(df)  
### \*\*Exercise 10: Merging DataFrames\*\*  
#1. Create another DataFrame with columns `"Product"` and `"Supplier"`, and merge it with the original DataFrame based on the `"Product"` column.**  
supplier\_data = {  
 "Product": ['Laptop', 'Mouse', 'Monitor', 'Keyboard', 'Phone'],  
 "Supplier": ['Supplier A', 'Supplier B', 'Supplier C', 'Supplier D', 'Supplier E']  
}  
supplier\_df = pd.DataFrame(supplier\_data)  
print("New DataFrame with 'Product' and 'Supplier':")  
print(supplier\_df)  
  
merged\_df = pd.merge(df, supplier\_df, on='Product')  
print("\nMerged DataFrame:")  
print(merged\_df)  
 **### \*\*Exercise 11: Pivot Tables\*\*  
#1. 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 :")  
print(pivot\_table)  
 **### \*\*Exercise 12: Concatenating DataFrames\*\*  
#1. Create two separate DataFrames for two different stores with the same columns (`"Product"`, `"Price"`, `"Quantity"`).  
#2. Concatenate these DataFrames to create a combined inventory list.**  
store1\_data = {  
 "Product": ['Laptop', 'Mouse', 'Monitor'],  
 "Price": [80000, 1500, 20000],  
 "Quantity": [5, 50, 20]  
}  
store1\_df = pd.DataFrame(store1\_data)  
  
store2\_data = {  
 "Product": ['Keyboard', 'Phone', 'Monitor'],  
 "Price": [3000, 40000, 21000],  
 "Quantity": [25, 15, 30]  
}  
store2\_df = pd.DataFrame(store2\_data)  
  
combined\_inventory\_df = pd.concat([store1\_df, store2\_df], ignore\_index=True)  
print("\nCombined Inventory DataFrame:")  
print(combined\_inventory\_df)  
 **### \*\*Exercise 13: Working with Dates\*\*  
#1. Create a DataFrame with a `"Date"` column that contains the last 5 days starting from today.  
#2. Add a column `"Sales"` with random values for each day.  
#3. Find the total sales for all days combined.**import pandas as pd  
import random  
from datetime import datetime, timedelta  
  
today = datetime.now()  
dates = [today - timedelta(days=i) for i in range(5)]  
dates\_df = pd.DataFrame({"Date": dates})  
  
dates\_df['Sales'] = [random.randint(100, 1000) for \_ in range(5)]  
print("DataFrame with 'Date' and 'Sales' columns:")  
print(dates\_df)  
  
total\_sales = dates\_df['Sales'].sum()  
print("\nTotal sales for all days combined:")  
print(total\_sales)  
  
**### \*\*Exercise 14: Reshaping Data with Melt\*\*  
#1. Create a DataFrame with columns `"Product"`, `"Region"`, `"Q1\_Sales"`, `"Q2\_Sales"`.  
#2. Use `pd.melt()` to reshape the DataFrame so that it has columns `"Product"`, `"Region"`, `"Quarter"`, and `"Sales"`.**import pandas as pd  
  
data = {  
 "Product": ['Laptop', 'Mouse', 'Monitor'],  
 "Region": ['North', 'South', 'East'],  
 "Q1\_Sales": [15000, 12000, 20000],  
 "Q2\_Sales": [18000, 11000, 22000]  
}  
df = pd.DataFrame(data)  
print("Original DataFrame:")  
print(df)  
  
melted\_df = pd.melt(df, id\_vars=['Product', 'Region'],  
 value\_vars=['Q1\_Sales', 'Q2\_Sales'],  
 var\_name='Quarter', value\_name='Sales')  
print("\nReshaped DataFrame:")  
print(melted\_df)  
 **### \*\*Exercise 15: Reading and Writing Data\*\*  
#1. Read the data from a CSV file named `products.csv` into a DataFrame.  
#2. After performing some operations (e.g., adding a new column or modifying values), write the DataFrame back to a new CSV file named `updated\_products.csv`.**import pandas as pd  
  
df = pd.read\_csv('products.csv')  
print("Original DataFrame:")  
print(df)  
  
df['Discounted\_Price'] = df['Price'] \* 0.90  
print("\nDataFrame after adding 'Discounted\_Price' column:")  
print(df)  
  
df.to\_csv('updated\_products.csv', index=False)  
print("\nDataFrame has been written to 'updated\_products.csv'.")  
  
**### \*\*Exercise 16: Renaming Columns\*\*  
#1. Given a DataFrame with columns `"Prod"`, `"Cat"`, `"Price"`, `"Qty"`, rename the columns to `"Product"`, `"Category"`, `"Price"`, and `"Quantity"`.  
#2. Print the renamed DataFrame.**import pandas as pd  
data = {  
 "Prod": ['Laptop', 'Mouse', 'Monitor', 'Keyboard', 'Phone'],  
 "Cat": ['Electronics', 'Accessories', 'Electronics', 'Accessories', 'Electronics'],  
 "Price": [80000, 1500, 20000, 3000, 40000],  
 "Qty": [10, 100, 50, 75, 30]  
}  
df = pd.DataFrame(data)  
  
df.rename(columns={"Prod": "Product", "Cat": "Category", "Qty": "Quantity"}, inplace=True)  
print(df)  
  
**### \*\*Exercise 17: Creating a MultiIndex DataFrame\*\*  
#1. Create a DataFrame using a MultiIndex (hierarchical index) with two levels: `"Store"` and `"Product"`. The DataFrame should have columns `"Price"` and `"Quantity"`, representing the price and quantity of products in different stores.  
#2. Print the MultiIndex DataFrame.**import pandas as pd  
index = pd.MultiIndex.from\_tuples([  
 ('Store\_A', 'Laptop'),  
 ('Store\_A', 'Mouse'),  
 ('Store\_A', 'Monitor'),  
 ('Store\_B', 'Keyboard'),  
 ('Store\_B', 'Phone'),  
 ('Store\_B', 'Mouse')], names=['Store', 'Product'])  
data = {  
 'Price': [85000, 1600, 21000, 3100, 41000, 1550],  
 'Quantity': [12, 90, 45, 70, 25, 95]  
}  
df = pd.DataFrame(data, index=index)  
print(df)  
  
**### \*\*Exercise 18: Resample Time-Series Data\*\*  
#1. Create a DataFrame with a `"Date"` column containing a range of dates for the past 30 days and a `"Sales"` column with random values.  
#2. Resample the data to show the total sales by week.**import pandas as pd  
import random  
date\_range = pd.date\_range(end=pd.Timestamp.today(), periods=30)  
sales\_data = [random.randint(100, 1000) for \_ in range(len(date\_range))]  
  
df = pd.DataFrame({  
 'Date': date\_range,  
 'Sales': sales\_data  
})  
  
df.set\_index('Date', inplace=True)  
  
weekly\_sales = df.resample('W').sum()  
  
print(weekly\_sales)  
  
**### \*\*Exercise 19: Handling Duplicates\*\*  
#1. Given a DataFrame with duplicate rows, identify and remove the duplicate rows.  
#2. Print the cleaned DataFrame.**import pandas as pd  
  
data = {  
 'Product': ['Laptop', 'Mouse', 'Monitor', 'Mouse', 'Phone', 'Monitor'],  
 'Category': ['Electronics', 'Accessories', 'Electronics', 'Accessories', 'Electronics', 'Electronics'],  
 'Price': [80000, 1500, 20000, 1500, 40000, 20000],  
 'Quantity': [10, 100, 50, 100, 30, 50]  
}  
df = pd.DataFrame(data)  
  
df\_cleaned = df.drop\_duplicates()  
print("Cleaned data:")  
print(df\_cleaned)  
  
**### \*\*Exercise 20: Correlation Matrix\*\*  
#1. Create a DataFrame with numeric data representing different features (e.g., `"Height"`, `"Weight"`, `"Age"`, `"Income"`).  
#2. Compute the correlation matrix for the DataFrame.  
#3. Print the correlation matrix.**import pandas as pd  
  
data = {  
 'Height': [170, 180, 160, 175, 165],  
 'Weight': [70, 80, 60, 75, 65],  
 'Age': [25, 30, 22, 28, 24],  
 'Income': [50000, 60000, 40000, 55000, 45000]  
}  
df = pd.DataFrame(data)  
  
correlation\_matrix = df.corr()  
print("correlation matrix:")  
print(correlation\_matrix)  
  
**### \*\*Exercise 21: Cumulative Sum and Rolling Windows\*\*  
#1. Create a DataFrame with random sales data for each day over the last 30 days.  
#2. Calculate the cumulative sum of the sales and add it as a new column `"Cumulative Sales"`.  
#3. Calculate the rolling average of sales over the past 7 days and add it as a new column `"Rolling Avg"`.**import pandas as pd  
import random  
  
date\_range = pd.date\_range(end=pd.Timestamp.today(), periods=30)  
  
sales\_data = [random.randint(100, 1000) for \_ in range(len(date\_range))]  
  
df = pd.DataFrame({  
 'Date': date\_range,  
 'Sales': sales\_data  
})  
  
df['Cumulative Sales'] = df['Sales'].cumsum()  
  
df['Rolling Avg'] = df['Sales'].rolling(window=7).mean()  
print(df)  
 **### \*\*Exercise 22: String Operations\*\*  
#1. Create a DataFrame with a column `"Names"` containing values like `"John Doe"`, `"Jane Smith"`, `"Sam Brown"`.  
#2. Split the `"Names"` column into two separate columns: `"First Name"` and `"Last Name"`.  
#3. Convert the `"First Name"` column to uppercase.**import pandas as pd  
data = {  
 'Names': ['John Doe', 'Jane Smith', 'Sam Brown']  
}  
df = pd.DataFrame(data)  
  
df[['First Name', 'Last Name']] = df['Names'].str.split(' ', expand=True)  
  
df['First Name'] = df['First Name'].str.upper()  
print(df)  
  
**### \*\*Exercise 23: Conditional Selections with `np.where`\*\*  
#1. Create a DataFrame with columns `"Employee"`, `"Age"`, and `"Department"`.  
#2. Create a new column `"Status"` that assigns `"Senior"` to employees aged 40 or above and `"Junior"` to employees below 40 using `np.where()`.**import pandas as pd  
  
# Create a DataFrame with 'Employee', 'Age', and 'Department' columns  
data = {  
 'Employee': ['Alice', 'Bob', 'Charlie', 'David', 'Eve'],  
 'Age': [25, 45, 35, 50, 28],  
 'Department': ['HR', 'Finance', 'IT', 'Marketing', 'Sales']  
}  
df = pd.DataFrame(data)  
df['Status'] = df['Age'].apply(lambda x: 'Senior' if x >= 40 else 'Junior')  
print(df)  
  
**### \*\*Exercise 24: Slicing DataFrames\*\*  
#1. Given a DataFrame with data on `"Products"`, `"Category"`, `"Sales"`, and `"Profit"`, slice the DataFrame to display:  
# - The first 10 rows.  
# - All rows where the `"Category"` is `"Electronics"`.  
# - Only the `"Sales"` and `"Profit"` columns for products with sales greater than 50,000.  
import pandas as pd  
  
# Sample DataFrame with 'Products', 'Category', 'Sales', and 'Profit' columns**data = {  
 'Products': ['Laptop', 'Mouse', 'Monitor', 'Keyboard', 'Phone', 'Tablet', 'Printer', 'Camera', 'Speaker', 'TV'],  
 'Category': ['Electronics', 'Accessories', 'Electronics', 'Accessories', 'Electronics', 'Electronics', 'Accessories', 'Electronics', 'Accessories', 'Electronics'],  
 'Sales': [80000, 1500, 20000, 3000, 40000, 35000, 2500, 45000, 4000, 60000],  
 'Profit': [15000, 300, 5000, 600, 8000, 7000, 450, 9000, 800, 12000]  
}  
df = pd.DataFrame(data)  
  
first\_10\_rows = df.head(10)  
print("First 10 rows:")  
print(first\_10\_rows)  
  
electronics\_category = df[df['Category'] == 'Electronics']  
print("\nRows where Category is 'Electronics':")  
print(electronics\_category)  
  
high\_sales = df.loc[df['Sales'] > 50000, ['Sales', 'Profit']]  
print("\nproducts with sales > 50,000:")  
print(high\_sales)  
  
**### \*\*Exercise 25: Concatenating DataFrames Vertically and Horizontally\*\*  
#1. Create two DataFrames with identical columns `"Employee"`, `"Age"`, `"Salary"`, but different rows (e.g., one for employees in `"Store A"` and one for employees in `"Store B"`).  
#2. Concatenate the DataFrames vertically to create a combined DataFrame.  
#3. Now create two DataFrames with different columns (e.g., `"Employee"`, `"Department"` and `"Employee"`, `"Salary"`) and concatenate them horizontally based on the common `"Employee"` column.**import pandas as pd  
  
data\_store\_a = {  
 'Employee': ['Alice', 'Bob', 'Charlie'],  
 'Age': [28, 34, 29],  
 'Salary': [70000, 80000, 75000]  
}  
data\_store\_b = {  
 'Employee': ['David', 'Eve', 'Frank'],  
 'Age': [45, 36, 30],  
 'Salary': [90000, 85000, 78000]  
}  
df\_store\_a = pd.DataFrame(data\_store\_a)  
df\_store\_b = pd.DataFrame(data\_store\_b)  
  
combined\_df = pd.concat([df\_store\_a, df\_store\_b], ignore\_index=True)  
print("Combined DataFrame (Vertical Concatenation):")  
print(combined\_df)  
  
data\_dept = {  
 'Employee': ['Alice', 'Bob', 'Charlie'],  
 'Department': ['HR', 'Finance', 'IT']  
}  
  
data\_salary = {  
 'Employee': ['Alice', 'Bob', 'Charlie'],  
 'Salary': [70000, 80000, 75000]  
}  
  
df\_dept = pd.DataFrame(data\_dept)  
df\_salary = pd.DataFrame(data\_salary)  
  
combined\_horizontal\_df = pd.merge(df\_dept, df\_salary, on='Employee')  
print("\nCombined DataFrame (Horizontal Concatenation):")  
print(combined\_horizontal\_df)  
 **### \*\*Exercise 26: Exploding Lists in DataFrame Columns\*\*  
#1. Create a DataFrame with a column `"Product"` and a column `"Features"` where each feature is a list (e.g., `["Feature1", "Feature2"]`).  
#2. Use the `explode()` method to create a new row for each feature in the list, so each product-feature pair has its own row.**  
import pandas as pd  
  
data = {  
 'Product': ['Laptop', 'Mouse', 'Monitor'],  
 'Features': [['Intel i7', '16GB RAM', '512GB SSD'], ['Wireless', 'Ergonomic'], ['4K Resolution', '27-inch', 'High Refresh Rate']]  
}  
df = pd.DataFrame(data)  
df\_exploded = df.explode('Features')  
print(df\_exploded)  
  
**### \*\*Exercise 27: Using `.map()` and `.applymap()`\*\*  
#1. Given a DataFrame with columns `"Product"`, `"Price"`, and `"Quantity"`, use `.map()` to apply a custom function to increase `"Price"` by 10% for each row.  
#2. Use `.applymap()` to format the numeric values in the DataFrame to two decimal places.**import pandas as pd  
  
data = {  
 'Product': ['Laptop', 'Mouse', 'Monitor'],  
 'Price': [80000, 1500, 20000],  
 'Quantity': [10, 100, 50]  
}  
df = pd.DataFrame(data)  
  
df['Price'] = df['Price'].map(lambda x: x \* 1.10)  
  
df\_formatted = df.applymap(lambda x: f"{x:.2f}" if isinstance(x, (int, float)) else x)  
print("Updated DataFrame :")  
print(df\_formatted)  
  
**### \*\*Exercise 28: Combining `groupby()` with `apply()`\*\*  
#1. Create a DataFrame with `"City"`, `"Product"`, `"Sales"`, and `"Profit"`.  
#2. Group by `"City"` and apply a custom function to calculate the profit margin (Profit/Sales) for each city.**import pandas as pd  
  
data = {  
 'City': ['New York', 'Los Angeles', 'New York', 'Chicago', 'Los Angeles', 'Chicago'],  
 'Product': ['Laptop', 'Mouse', 'Monitor', 'Keyboard', 'Phone', 'Tablet'],  
 'Sales': [80000, 1500, 20000, 3000, 40000, 35000],  
 'Profit': [15000, 300, 5000, 600, 8000, 7000]  
}  
df = pd.DataFrame(data)  
  
profit\_margin\_by\_city = df.groupby('City').apply(lambda x: x['Profit'].sum() / x['Sales'].sum()).reset\_index(name='Profit Margin')  
print(profit\_margin\_by\_city)  
 **### \*\*Exercise 29: Creating a DataFrame from Multiple Sources\*\*  
#1. Create three different DataFrames from different sources (e.g., CSV, JSON, and a Python dictionary).  
#2. Merge the DataFrames based on a common column and create a consolidated report.**  
import pandas as pd  
  
data\_dict = {  
 'ID': [1, 2, 3],  
 'Name': ['Alice', 'Bob', 'Charlie'],  
 'Age': [25, 30, 35]  
}  
df\_dict = pd.DataFrame(data\_dict)  
df\_csv = pd.read\_csv('data\_csv.csv')  
df\_json = pd.read\_json('data\_json.json')  
  
df\_merged = pd.merge(df\_dict, df\_csv, on='ID')  
df\_merged = pd.merge(df\_merged, df\_json, on='ID')  
print("Consolidated Report:")  
print(df\_merged)  
  
**### \*\*Exercise 30: Dealing with Large Datasets\*\*  
#1. Create a large DataFrame with 1 million rows, representing data on `"Transaction ID"`, `"Customer"`, `"Product"`, `"Amount"`, and `"Date"`.  
#2. Split the DataFrame into smaller chunks (e.g., 100,000 rows each), perform a simple analysis on each chunk (e.g., total sales), and combine the results.**import pandas as pd  
from datetime import datetime, timedelta  
import random  
  
# Create a large DataFrame with 1 million rows  
num\_rows = 1000000  
data = {  
 'Transaction ID': range(1, num\_rows + 1),  
 'Customer': [random.choice(['Alice', 'Bob', 'Charlie', 'David', 'Eva']) for \_ in range(num\_rows)],  
 'Product': [random.choice(['Laptop', 'Mouse', 'Monitor', 'Keyboard', 'Phone']) for \_ in range(num\_rows)],  
 'Amount': [random.uniform(10, 1000) for \_ in range(num\_rows)],  
 'Date': [datetime.now() - timedelta(days=random.randint(0, 365)) for \_ in range(num\_rows)]  
}  
  
df\_large = pd.DataFrame(data)  
print("Large DataFrame created .")  
print(df\_large)  
#---