ASSIGNMENT-2

from random import random
import pandas as pd
import numpy as np
from datetime import datetime

Exercise 1: Creating DataFrame from Scratch
1. Create a DataFrame with the following columns: `"Product"`, `"Category"`, `"Price"`, and `"Quantity"`. Use the following data:

Program:

```
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)
# 2. Print the DataFrame.
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.info())
# 3. Display a summary of statistics (mean, min, max, etc.) for the numeric columns in the DataFrame
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"` and print them.
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 calculated by multiplying `"Price"` and `"Quantity"`.

df["Total Value"] = df["Price"] * df["Quantity"]
print(df)

# 2. Drop the `"Category"` column from the DataFrame and print the updated DataFrame
df_dropped_cat = df.drop(columns = ['Category'])
print(df_dropped_cat)

Exercise 6: Sorting Data**
1. Sort the DataFrame by `"Price"` in descending order.

print(df.sort_values(by="Price",ascending=False))
# 2. Sort the DataFrame by `"Quantity"` in ascending order, then by `"Price"` in descending order (multi-level sorting).
print(df.sort_values(by=["Quantity","Price"],ascending=[True,False]))

# Exercise 7: Grouping Data**
# 1. Group the DataFrame by `"Category"` and calculate the total quantity for each category.
df_Cat_group = df.groupby("Category")["Quantity"].sum()
print(df_Cat_group)

# 2. Group by `"Category"` and calculate the average price for each category.
df_catAvg_group = df.groupby("Category")["Price"].mean()
print(df_catAvg_group)

Exercise 8: Handling Missing Data**
1. Introduce some missing values in the `"Price"` column by assigning `None` to two rows.

df.loc[0:1,"Price"] = None
print(df)
# 2. Fill the missing values with the mean price of the available products.
mean_price = df["Price"].mean()
df["Price"] = df["Price"].fillna(mean_price)
print(df)

# 3. Drop any rows where the `"Quantity"` is less than `50`.
df_quant = df[df["Quantity"] < 50].index

```
df_drop_quantity = df.drop(df_quant)
print(df_drop_quantity)

# 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"].apply(lambda x: x*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.
df_2 = pd.DataFrame({
      "Product" : ["Laptop","Mouse",'Monitor', 'Keyboard', 'Phone'],
      "Supplier" : ["HP","Logitech","Dell","Samsung","Apple"]
})
df_2_merged = pd.merge(df,df_2,on="Product",how="left")
print(df_2_merged)

# Exercise 11: Pivot Tables**
# 1. Create a pivot table that shows the total quantity of products for each category and product
combination.
df_pivot =
df.pivot_table(values="Quantity",index="Category",columns="Product",aggfunc="sum")
print(df_pivot)

# Exercise 12: Concatenating DataFrames**
# 1. Create two separate DataFrames for two different stores with the same columns
(`"Product"`, `"Price"`, `"Quantity"`).
```

Program:

```
df_1 = pd.DataFrame({
      "Product": ["Laptop","Keyboard","Mouse"],
      "Price": [50000,2500,1500],
      "Quantity":[50,30,40]
})

df2 = pd.DataFrame({
      "Product": ["Phone","Camera","GPU"],
      "Price": [25000,30000,40000],
      "Quantity":[10,30,20]
})
```

# 2. Concatenate these DataFrames to create a combined inventory list.
inventry_df = pd.concat([df_1,df2],ignore_index=True)
print(inventry_df)

Exercise 13: Working with Dates**
1. Create a DataFrame with a `"Date"` column that contains the last 5 days starting from today.
Program:
today = datetime.today()
five_dates = pd.date_range(end=today,periods=5)

df_dates = pd.DataFrame({
    "Date" : five_dates
})

2. Add a column `"Sales"` with random values for each day.
df_dates["Sales"] = np.random.randint(100,500,size = 5)

# 3. Find the total sales for all days combined.
print(df["Sales"].sum())

Exercise 14: Reshaping Data with Melt**
1. Create a DataFrame with columns `"Product"`, `"Region"`, `"Q1_Sales"`, `"Q2_Sales"`.
Program:
df_melt = pd.DataFrame({
    "Product": ["Laptop","Camera","Phone"],
    "Region": ["TN","Kerala","Andhra"],
    "Q1_Sales": [100,200,150],
    "Q2_Sales": [500,600,700]
})

# 2. Use `pd.melt()` to reshape the DataFrame so that it has columns `"Product"`, `"Region"`, `"Quarter"`, and `"Sales"`.
df_melted = pd.melt(df_melt, id_vars=["Product","Region"], var_name="Quarter", value_name="Sales")
print(df_melted)

Exercise 15: Reading and Writing Data**
1. Read the data from a CSV file named `products.csv` into a DataFrame.
Program:
df_products = pd.read_csv("products.csv")
print(df_products)

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`.

```
df_products["Category"] = ["Electronic","Accessories","Electronic","Electronic"]
print(df_products)

df_products["Price"] = df_products["Price"] - 1000
print(df_products)

df_products.to_csv("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"`.

Program:

```
df_to_rename = pd.DataFrame({
      "Prod" : ["Laptop","Mobile"],
      "Cat" : ["Electronics","Electronics"],
      "Price": [50000,25000],
      "Qty": [25,10]
})

df_rename =
df_to_rename.rename(columns={"Prod":"Product","Cat":"Category","Qty":"Quantity"})
```

2. Print the renamed DataFrame.

```
print(df_rename)
```

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.

Program:

```
multi_index = pd.MultiIndex.from_tuples(
[("Store_A","Laptop"),
       ("Store_A","Mouse"),
       ("Store_A","Monitor"),
       ("Store_B", "Laptop"),
       ("Store_B", "Mouse"),
       ("Store_B", "Monitor"),
       ],
      names=["Store","Product"]
```

```
)

data_multi = {
    "Price" : [50000,1500,20000,40000,3000,15000],
    "Quantity": [15,20,25,15,10,30]
}

df_multi_index = pd.DataFrame(data_multi,index=multi_index)

# 2. Print the MultiIndex DataFrame.
print(df_multi_index)

# 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.
```
Program:
```
today_date = datetime.today()
dates = pd.date_range(end=today_date,periods=30)

sales = np.random.randint(100,1000,size=30)

df_sales = pd.DataFrame({
    "Date" : dates,
    "Sales": sales
})
#print(df_sales)

# 2. Resample the data to show the total sales by week.
df_sales.set_index("Date",inplace=True)

weekly_sales = df_sales.resample("W").sum()
print(weekly_sales)

# Exercise 19: Handling Duplicates**
# 1. Given a DataFrame with duplicate rows, identify and remove the duplicate rows.
```
Program:
```
df_duplicate = pd.DataFrame({
    "Product" : ["Laptop","Mouse","Phone","Laptop","Mouse"],
    "Price" : [50000,2500,30000,50000,2500]
})

duplicates = df_duplicate.duplicated()
print(df_duplicate[duplicates])
```

```
df_cleaned = df_duplicate.drop_duplicates()

# 2. Print the cleaned DataFrame.
print(df_cleaned)

# Exercise 20: Correlation Matrix**
# 1. Create a DataFrame with numeric data representing different features (e.g., `"Height"`,
`"Weight"`, `"Age"`, `"Income"`).
```
Program:
```
df_correlation = pd.DataFrame({
    'Height': [160, 175, 168, 180, 155],
    'Weight': [60, 70, 65, 85, 50],
    'Age': [25, 32, 28, 40, 22],
    'Income': [50000, 60000, 58000, 80000, 45000]
})

# 2. Compute the correlation matrix for the DataFrame.
correlation_matrix_df = df_correlation.corr()

# 3. Print the correlation matrix.
print(correlation_matrix_df)

# Exercise 21: Cumulative Sum and Rolling Windows**
# 1. Create a DataFrame with random sales data for each day over the last 30 days.
```
Program:
```
print(df_sales)

# 2. Calculate the cumulative sum of the sales and add it as a new column `"Cumulative Sales"`.

df_sales["Cumulative Sales"] = df_sales["Sales"].cumsum()

# 3. Calculate the rolling average of sales over the past 7 days and add it as a new column
`"Rolling Avg"`.
df_sales["Rolling Average"] = df_sales["Sales"].rolling(window=7).mean()
print(df_sales)

# Exercise 22: String Operations**
# 1. Create a DataFrame with a column `"Names"` containing values like `"John Doe"`, `"Jane
Smith"`, `"Sam Brown"`.
```
Program:
```
df_string = pd.DataFrame({
    "Names" : ["John Doe", "Jane Smith", "Sam Brown"],
```

```
})

# 2. Split the `"Names"` column into two separate columns: `"First Name"` and `"Last Name"`.

df_string[["First Name", "Last Name"]] = df_string["Names"].str.split(' ', expand=True)

# 3. Convert the `"First Name"` column to uppercase.
df_string["First Name"] = df_string["First Name"].str.upper()
print(df_string)

# Exercise 23: Conditional Selections with `np.where`**
# 1. Create a DataFrame with columns `"Employee"`, `"Age"`, and `"Department"`.
```
Program:
```
df_np_where = pd.DataFrame({
      "Employee" : ["Sai", "Subash"],
      "Age" : [45,30],
      "Department" : ["IT", "Finance"]
})

# 2. Create a new column `"Status"` that assigns `"Senior"` to employees aged 40 or above and `"Junior"` to employees below 40 using `np.where()`.

df_np_where["Status"] = np.where(df_np_where["Age"] >= 40, "Senior", "Junior")
print(df_np_where)

# 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.
```
Program:
```
df_slice = pd.DataFrame({
      'Product': ['Laptop', 'Mouse', 'Monitor', 'Keyboard', 'Phone', 'Tablet', 'Printer', 'Webcam', 'Speaker', 'Headphones', 'Charger', 'Case', 'Dock'],
      'Category': ['Electronics', 'Accessories', 'Electronics', 'Accessories', 'Electronics', 'Electronics', 'Accessories', 'Electronics', 'Accessories', 'Electronics', 'Accessories', 'Accessories', 'Electronics'],
      'Sales': [80000, 1500, 20000, 3000, 40000, 12000, 2500, 70000, 5000, 18000, 1500, 2000, 3000],
      'Profit': [20000, 500, 7000, 800, 10000, 3000, 400, 15000, 800, 6000, 400, 600, 700]
})
```

```
print(df_slice.head(10))

print(df_slice[df_slice["Category"]=="Electronics"])

print(df_slice[df_slice["Sales"]>50000][["Sales","Profit"]])
```

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"`).

Program:

```
df_store_a = pd.DataFrame({
    "Employee" : ["Sai","Subash"],
    "Age" : [30,45],
    "Salary" : [50000,60000]
})

df_store_b = pd.DataFrame({
    "Employee" : ["Chandra","Akash"],
    "Age" : [31,46],
    "Salary" : [55000,66000]
})
```

2. Concatenate the DataFrames vertically to create a combined DataFrame.

```
df_verti = pd.concat([df_store_a,df_store_b])
print(df_verti)
```

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.

Program:

```
df_horizontal1 = pd.DataFrame({
    "Employee" : ["Sai", "Subash"],
    "Department" : ["HR", "IT"]
})

df_horizontal2 = pd.DataFrame({
    "Employee" : ["Sai", "Subash"],
    "Salary" : [50000,60000]
})

df_horizontal = pd.merge(df_horizontal1,df_horizontal2, on="Employee")
print(df_horizontal)
```

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"]`).
Program:
```
df_explode = pd.DataFrame({
    'Product': ['Laptop', 'Smartphone', 'Tablet'],
    'Features': [['Touchscreen', '8GB RAM', '256GB SSD'],
                 ['5G', '64GB Storage', '12MP Camera'],
                 ['10.5 inch Screen', '4GB RAM']]
})
```

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.

```
df_exploded = df_explode.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.
```
df_1["Price"] = df_1["Price"].map(lambda x : x*1.10)
print(df_1)
```

2. Use `.applymap()` to format the numeric values in the DataFrame to two decimal places.
```
df_1_format = df_1.applymap(lambda x: f"{x:.2f}" if isinstance(x, (int,float)) else x)
print(df_1_format)
```

Exercise 28: Combining `groupby()` with `apply()`**
1. Create a DataFrame with `"City"`, `"Product"`, `"Sales"`, and `"Profit"`.
Program:
```
df_city = pd.DataFrame({
    "City" : ["Chennai","Bangalore", "Hyderabad", "Chennai", "Bangalore", "Mumbai", "Hyderabad"],
    "Product" : ["Laptop","Mouse","Keyboard","Phone","Tablet","Monitor","CPU"],
    "Sales" : [80000, 1500, 20000, 3000, 40000, 12000, 10000],
    "Profit" : [20000, 500, 7000, 800, 10000, 3000, 500]
})
```

2. Group by `"City"` and apply a custom function to calculate the profit margin (Profit/Sales) for each city.
Program:
```
def profit_margin(data):
```

```
    data["Profit Margin"] = data["Profit"] / data["Sales"]
    return data
df_profit_margin = df_city.groupby("City").apply(profit_margin)
print(df_profit_margin)

# Exercise 29: Creating a DataFrame from Multiple Sources**
# 1. Create three different DataFrames from different sources (e.g., CSV, JSON, and a Python dictionary).
```
Program:
```
df_csv = pd.read_csv("data_merge.csv")
df_json = pd.read_json("data_merge.json")

data_dict = pd.DataFrame({
    'ID': [1, 2, 3],
    'Location': ['Chennai', 'Bangalore', 'Mumbai']
})

# 2. Merge the DataFrames based on a common column and create a consolidated report.

df_merged_multiple = pd.merge(df_csv,df_json, on="ID")
df_merged_multiple = pd.merge(df_merged_multiple, data_dict, on="ID")

print(df_merged_multiple)

# 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"`.
```
Program:
```
num_rows = 1000000
transaction_id = np.arange(1, num_rows+1)
customers = np.random.choice(["Sai","Subash","Chandra","Akash","Rex","Alen","Induja"], num_rows)
products = np.random.choice(["Laptop","Mouse","Monitor","Keyboard","Phone"], num_rows)
amounts = np.random.uniform(10,1000,num_rows)
start_date = pd.Timestamp('2023-01-01')
end_date = pd.Timestamp('2024-01-01')
date_range = pd.date_range(start=start_date, end=end_date)
random_dates = np.random.choice(date_range, size=num_rows, replace=True)

df_large = pd.DataFrame({
    "Transaction ID" : transaction_id,
    "Customer" : customers,
    "Product" : products,
```

```
    "Amount" : amounts,
    "Date" : random_dates
})

print(df_large.head())
```

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.

Program:

```
def analyze_chunk(chunk):
    total_sales = chunk["Amount"].sum()
    return total_sales

chunk_size = 100000
num_chunks = num_rows // chunk_size

chunks = np.array_split(df_large,num_chunks)

results = [analyze_chunk(chunk) for chunk in chunks]

total_sales_large = sum(results)

print(total_sales_large)
```