Pandas Exam Paper 2 - (Total Marks 30 Questions - 2 Marks Each)

Section A: Data Manipulation (7 Questions)

1. **Applying Functions to Columns**
   Apply a function to double the values of the 'Price' column using `apply()` .

   Answer: import pandas as pd

   ```
   # Sample DataFrame
   data = {
   Product': ['A', 'B', 'C'], 'Price': [10, 20, 30]
   }
   df = pd.DataFrame(data)

   # Applying function to double the 'Price' column
   df['Price'] = df['Price'].apply(lambda x: x * 2)

   print(df)
   Output:   Product  Price
   0      A     20
   1      B     40
   2      C     60
   ```

2. **Mapping Values in Series**
   Use `map()` to replace all occurrences of 'Yes' in the 'Passed' column with `True` and 'No' with `False` .

   Answer: import pandas as pd

```
# Sample DataFrame
data = {'Student': ['Alice', 'Bob', 'Charlie'], 'Passed': ['Yes', 'No', 'Yes']}
df = pd.DataFrame(data)

# Mapping values in 'Passed' column
df['Passed'] = df['Passed'].map({'Yes': True, 'No': False})

print(df)
```

Output :

```
    Student  Passed
0   Alice    True
1     Bob   False
2  Charlie    True
```

3. **Lowercase Strings**
   Convert all strings in the 'Names' column to lowercase.

Answer:

```
import pandas as pd

# Sample DataFrame
data = {'Names': ['Alice', 'BOB', 'ChArLie']}
df = pd.DataFrame(data)

# Converting all names to lowercase
df['Names'] = df['Names'].str.lower()

print(df)
```

Output :

```
     Names
0   alice
1     bob
2  charlie
```

4. **Uppercase Strings**
   Convert the 'City' column to uppercase.

   Answer:
```
import pandas as pd


# Sample DataFrame

data = {'City': ['New York', 'los angeles', 'chicago']}

df = pd.DataFrame(data)


# Converting all city names to uppercase

df['City'] = df['City'].str.upper()


print(df)
```

Output :
```
        City
0     NEW YORK
1  LOS ANGELES
2      CHICAGO
```

5. **Splitting Strings**
   Split the 'FullName' column into 'FirstName' and 'LastName' using a space as the delimiter.

   Answer:
```
import pandas as pd


# Sample DataFrame

data = {'FullName': ['Alice Johnson', 'Bob Smith', 'Charlie Brown']}

df = pd.DataFrame(data)


# Splitting 'FullName' into 'FirstName' and 'LastName'

df[['FirstName', 'LastName']] = df['FullName'].str.split(' ', expand=True)


print(df)
```

Output :

```
          FullName FirstName LastName
0  Alice Johnson     Alice  Johnson
1     Bob Smith       Bob    Smith
2  Charlie Brown   Charlie   Brown
```

6. **String Contains**
   Filter rows where the 'Email' column contains '@gmail.com'.

   Answer: import pandas as pd

```
# Sample DataFrame
data = {'Name': ['Alice', 'Bob', 'Charlie'],
        'Email': ['alice@gmail.com', 'bob@yahoo.com', 'charlie@gmail.com']}
df = pd.DataFrame(data)

# Filtering rows where 'Email' contains '@gmail.com'
gmail_df = df[df['Email'].str.contains('@gmail.com')]

print(gmail_df)
```

Output :

```
      Name            Email
0   Alice  alice@gmail.com
2  Charlie  charlie@gmail.com
```

7. **Replacing String Patterns**
   Use `str.replace()` to replace the domain in all emails from '@example.com' to '@newdomain.com'.

   Answer: import pandas as pd

```
# Sample DataFrame
data = {'Name': ['Alice', 'Bob', 'Charlie'],
        'Email': ['alice@example.com', 'bob@example.com', 'charlie@example.com']}
df = pd.DataFrame(data)

# Replacing '@example.com' with '@newdomain.com'
```

```
df['Email'] = df['Email'].str.replace('@example.com', '@newdomain.com', regex=False)


print(df)
```

Output :

```
     Name              Email
0   Alice  alice@newdomain.com
1     Bob    bob@newdomain.com
2  Charlie  charlie@newdomain.com
```

Section B: Grouping and Aggregation (8 Questions)

8. **Grouping Data**
   Group the DataFrame by the 'Department' column and calculate the mean salary for each department.

   Answer:

```
import pandas as pd


# Sample DataFrame
data = {'Employee': ['Alice', 'Bob', 'Charlie', 'David', 'Eve'],
        'Department': ['HR', 'IT', 'IT', 'HR', 'Finance'],
        'Salary': [50000, 60000, 70000, 55000, 65000]}
df = pd.DataFrame(data)


# Grouping by 'Department' and calculating mean salary
dept_salary_mean = df.groupby('Department')['Salary'].mean()


print(dept_salary_mean)
```

Output :

```
Department
Finance    65000.0
HR         52500.0
IT         65000.0
```

Name: Salary, dtype: float64

9. **Aggregating Data**
   Apply multiple aggregate functions (mean, max) to the 'Sales' column using `agg()` .

   Answer:

```
import pandas as pd

# Sample DataFrame
data = {'Employee': ['Alice', 'Bob', 'Charlie', 'David', 'Eve'],
        'Sales': [1000, 1500, 1200, 1800, 1300]}
df = pd.DataFrame(data)

# Applying multiple aggregate functions to the 'Sales' column
sales_agg = df['Sales'].agg(['mean', 'max'])

print(sales_agg)
```

Output :

```
mean    1360.0
max     1800.0
Name: Sales, dtype: float64
```

10. **Aggregate Multiple Functions**
    Use `aggregate()` to calculate both the sum and count of the 'Marks' column.

    Answer:

```
import pandas as pd

# Sample DataFrame
data = {'Student': ['Alice', 'Bob', 'Charlie', 'David', 'Eve'],
    'Marks': [85, 90, 78, 92, 88]}
df = pd.DataFrame(data)
```

```
# Applying multiple aggregate functions (sum and count) to the 'Marks' column

marks_agg = df['Marks'].agg(['sum', 'count'])


print(marks_agg)
```

```
Output :   sum      433

count      5

Name: Marks, dtype: int64
```

11. **Filtering with `isin()`**
    Filter rows where the 'City' column is either 'New York' or 'Los Angeles' using `isin()` .

    Answer: import pandas as pd

```
# Sample DataFrame
data = {'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eve'],
        'City': ['New York', 'Los Angeles', 'Chicago', 'New York', 'Houston']}
df = pd.DataFrame(data)

# Filtering rows where 'City' is 'New York' or 'Los Angeles'
filtered_df = df[df['City'].isin(['New York', 'Los Angeles'])]

print(filtered_df)
```

Output :

```
    Name        City
0  Alice    New York
1    Bob  Los Angeles
3  David    New York
```

12. **Grouping and Aggregating**
    Group the DataFrame by 'Gender' and calculate the sum of the 'Marks' column for each group.

    Answer: import pandas as pd

```
# Sample DataFrame
data = {'Student': ['Alice', 'Bob', 'Charlie', 'David', 'Eve'],
        'Gender': ['Female', 'Male', 'Male', 'Male', 'Female'],
        'Marks': [85, 90, 78, 92, 88]}
df = pd.DataFrame(data)

# Grouping by 'Gender' and calculating the sum of 'Marks'
marks_sum = df.groupby('Gender')['Marks'].sum()

print(marks_sum)
```

```
Output : Gender

Female    173

Male      260

Name: Marks, dtype: int64
```

13. **Multiple Aggregations on Multiple Columns**
Perform multiple aggregations (min, max, mean) on the 'Age' and 'Salary' columns.

Answer:
```
import pandas as pd


# Sample DataFrame

data = {'Employee': ['Alice', 'Bob', 'Charlie', 'David', 'Eve'],

        'Age': [25, 30, 35, 40, 28],

        'Salary': [50000, 60000, 70000, 80000, 55000]}

df = pd.DataFrame(data)


# Applying multiple aggregations to 'Age' and 'Salary' columns

agg_results = df[['Age', 'Salary']].agg(['min', 'max', 'mean'])


print(agg_results)
```

```
Output :           Age   Salary

min      25.00  50000.0

max      40.00  80000.0

mean     31.60  63000.0
```

14. **Grouping and Counting**
Group by 'City' and count the number of entries in each city.

Answer:
```
import pandas as pd


# Sample DataFrame

data = {'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eve', 'Frank'],

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

```
df = pd.DataFrame(data)


# Grouping by 'City' and counting the number of entries
city_counts = df.groupby('City')['Name'].count()


print(city_counts)
Output : City
Chicago        2
Los Angeles    2
New York       2
Name: Name, dtype: int64
```

15. **Using `apply()` with Groupby**
   Apply a custom function to find the range (max-min) of the 'Salary' column for each department.

   Answer:
```
import pandas as pd


# Sample DataFrame
data = {'Employee': ['Alice', 'Bob', 'Charlie', 'David', 'Eve', 'Frank'],
        'Department': ['HR', 'IT', 'IT', 'HR', 'Finance', 'Finance'],
        'Salary': [50000, 60000, 70000, 55000, 65000, 75000]}
df = pd.DataFrame(data)


# Defining a custom function to calculate the range
def salary_range(x):
    return x.max() - x.min()


# Applying the function using groupby and apply
salary_range_by_dept = df.groupby('Department')['Salary'].apply(salary_range)


print(salary_range_by_dept)
```

Output : Department

```
Finance    10000
HR          5000
IT         10000
Name: Salary, dtype: int64
```

Section C: Merging, Joining, and Concatenating (5 Questions)

16. **Concatenating DataFrames**
    Concatenate two DataFrames `df1` and `df2` along rows.

Answer: import pandas as pd

```
# Creating first DataFrame
df1 = pd.DataFrame({'ID': [1, 2, 3],
                    'Name': ['Alice', 'Bob', 'Charlie']})

# Creating second DataFrame
df2 = pd.DataFrame({'ID': [4, 5],
                    'Name': ['David', 'Eve']})

# Concatenating along rows
df_combined = pd.concat([df1, df2], axis=0, ignore_index=True)

print(df_combined)
```

Output :

```
   ID     Name
0   1    Alice
1   2      Bob
2   3  Charlie
3   4    David
4   5      Eve
```

17. **Merging DataFrames**

    Merge two DataFrames `df1` and `df2` on the 'ID' column.

    Answer: import pandas as pd

```
# Creating first DataFrame

df1 = pd.DataFrame({'ID': [1, 2, 3],

                    'Name': ['Alice', 'Bob', 'Charlie']})


# Creating second DataFrame

df2 = pd.DataFrame({'ID': [1, 2, 3],

                    'Salary': [50000, 60000, 70000]})


# Merging on 'ID' column

df_merged = pd.merge(df1, df2, on='ID')


print(df_merged)
```

Output :

```
   ID   Name  Salary
```

```
0   1  Alice  50000
1   2    Bob  60000
2   3 Charlie  70000
```

18. **Merging with Different Keys**
Merge DataFrames on different column names: 'df1' has 'EmployeeID' and 'df2' has 'ID'.

Answer:
```
import pandas as pd


# Creating first DataFrame
df1 = pd.DataFrame({'EmployeeID': [1, 2, 3],
                    'Name': ['Alice', 'Bob', 'Charlie']})


# Creating second DataFrame
df2 = pd.DataFrame({'ID': [1, 2, 3],
                    'Salary': [50000, 60000, 70000]})


# Merging on different column names
df_merged = pd.merge(df1, df2, left_on='EmployeeID', right_on='ID')


print(df_merged)
```

Output :
```
   EmployeeID    Name  ID  Salary
0           1   Alice   1  50000
1           2     Bob   2  60000
2           3 Charlie   3  70000
```

19. **Concatenating Along Columns**
Concatenate two DataFrames `df1` and `df2` along columns.

Answer:
```
import pandas as pd
```

```
# Creating first DataFrame

df1 = pd.DataFrame({'ID': [1, 2, 3],

                    'Name': ['Alice', 'Bob', 'Charlie']})


# Creating second DataFrame

df2 = pd.DataFrame({'Age': [25, 30, 35],

                    'Salary': [50000, 60000, 70000]})


# Concatenating along columns

df_combined = pd.concat([df1, df2], axis=1)


print(df_combined)
```

```
Output :    ID    Name  Age  Salary

0   1  Alice   25  50000

1   2    Bob   30  60000

2   3 Charlie   35  70000
```

20. **Joining DataFrames**
Join `df1` and `df2` on the 'ID' column with an outer join.

Answer: import pandas as pd

```
# Creating first DataFrame

df1 = pd.DataFrame({'ID': [1, 2, 3],

                    'Name': ['Alice', 'Bob', 'Charlie']})


# Creating second DataFrame

df2 = pd.DataFrame({'ID': [2, 3, 4],

                    'Salary': [60000, 70000, 80000]})


# Performing an outer join on 'ID'
```

```
df_joined = pd.merge(df1, df2, on='ID', how='outer')


print(df_joined)
```

Output :

```
   ID    Name  Salary
0   1  Alice     NaN
1   2    Bob  60000.0
2   3 Charlie  70000.0
3   4    NaN  80000.0
```

Section D: Reshaping and Input/Output (10 Questions)

21. **Transposing DataFrames**
    Transpose the rows and columns of the DataFrame `df` .

    Answer:

```
import pandas as pd


# Sample DataFrame
df = pd.DataFrame({'ID': [1, 2, 3],
        'Name': ['Alice', 'Bob', 'Charlie'],
        'Salary': [50000, 60000, 70000]})


# Transposing the DataFrame
df_transposed = df.T


print(df_transposed)
```

Output :

```
        0     1       2
ID      1     2       3
Name  Alice   Bob  Charlie
```

Salary 50000 60000   70000

22. **Using `T` Attribute**
Use the `T` attribute to transpose the DataFrame `df`.

Answer: import pandas as pd

```
# Sample DataFrame
df = pd.DataFrame({'ID': [1, 2, 3],
                   'Name': ['Alice', 'Bob', 'Charlie'],
                   'Salary': [50000, 60000, 70000]})

# Transposing the DataFrame using .T
df_transposed = df.T

print(df_transposed)
```

```
Output :        0      1        2
ID       1      2        3
Name  Alice    Bob  Charlie
Salary 50000  60000   70000
```

23. **Writing to CSV**
Save the DataFrame `df` to a file called `output.csv`.

Answer: import pandas as pd

```
# Sample DataFrame
df = pd.DataFrame({'ID': [1, 2, 3],
                   'Name': ['Alice', 'Bob', 'Charlie'],
                   'Salary': [50000, 60000, 70000]})

# Saving DataFrame to CSV
df.to_csv('output.csv', index=False)

print("DataFrame saved to output.csv")
```

24. **Writing to Excel**
Export the DataFrame `df` to an Excel file named `output.xlsx`.

Answer:
```
import pandas as pd

# Sample DataFrame
df = pd.DataFrame({'ID': [1, 2, 3],
                   'Name': ['Alice', 'Bob', 'Charlie'],
                   'Salary': [50000, 60000, 70000]})

# Saving DataFrame to Excel
df.to_excel('output.xlsx', index=False, engine='openpyxl')

print("DataFrame saved to output.xlsx")
```

25. **Writing to JSON**
Convert the DataFrame `df` to a JSON file named `output.json`.

Answer:
```
import pandas as pd

# Sample DataFrame
df = pd.DataFrame({'ID': [1, 2, 3],
                   'Name': ['Alice', 'Bob', 'Charlie'],
                   'Salary': [50000, 60000, 70000]})

# Saving DataFrame to JSON
df.to_json('output.json', orient='records', indent=4)

print("DataFrame saved to output.json")
```

26. **Rendering DataFrame as HTML**
Convert the DataFrame `df` to an HTML table and save it as `output.html`.

Answer:
```
import pandas as pd
```

```
# Sample DataFrame

df = pd.DataFrame({'ID': [1, 2, 3],

                   'Name': ['Alice', 'Bob', 'Charlie'],

                   'Salary': [50000, 60000, 70000]
```

27. **Loading CSV File**
    Load a CSV file named `student_data.csv` into a DataFrame.

    Answer:

```
import pandas as pd


# Loading the CSV file into a DataFrame

df = pd.read_csv('student_data.csv')


# Display the first few rows

print(df.head())


df = pd.read_csv('/path/to/student_data.csv')
```

29. **Saving a DataFrame as**

28. **Loading Excel File**

Load an Excel file named `sales_data.xlsx` into a DataFrame.

Answer: import pandas as pd

# Loading the Excel file into a DataFrame

df = pd.read_excel('sales_data.xlsx', engine='openpyxl')

# Display the first few rows

print(df.head())

df = pd.read_excel('sales_data.xlsx', sheet_name='Sheet1', engine='openpyxl')

df = pd.read_excel('/path/to/sales_data.xlsx', engine='openpyxl')

29. **Saving a DataFrame as CSV**

Save the DataFrame `df` to a CSV file called `employees.csv`, including only the 'Name' and 'Salary' columns.

Answer: import pandas as pd

```
# Sample DataFrame

df = pd.DataFrame({'ID': [1, 2, 3],

             'Name': ['Alice', 'Bob', 'Charlie'],

             'Salary': [50000, 60000, 70000],

             'Department': ['HR', 'IT', 'Finance']})


# Saving only 'Name' and 'Salary' columns to CSV

df[['Name', 'Salary']].to_csv('employees.csv', index=False)


print("DataFrame saved to employees.csv")
```

Output : Name,Salary

```
Alice,50000

Bob,60000
```

Charlie,70000

30. **Saving a DataFrame as JSON with Specific Columns**
    Save the DataFrame `df` as a JSON file, but only include the 'Name' and 'Department' columns.

    Answer: import pandas as pd

```
# Sample DataFrame
df = pd.DataFrame({'ID': [1, 2, 3],
                   'Name': ['Alice', 'Bob', 'Charlie'],
                   'Salary': [50000, 60000, 70000],
                   'Department': ['HR', 'IT', 'Finance']})

# Saving only 'Name' and 'Department' columns to JSON
df[['Name', 'Department']].to_json('employees.json', orient='records', indent=4)

print("DataFrame saved to employees.json")
```