Pandas DataFrame Operations Report

1. DataFrame Creation and Indexing

Task 1: Create a DataFrame with 4 columns and 6 rows filled with random integers.

```
import pandas as pd
import numpy as np
# Create a Pandas DataFrame with 4 columns and 6 rows filled with random integers
df = pd.DataFrame(np.random.randint(1, 100, size=(6, 4)), columns=['A', 'B', 'C', 'D'])
# Set the index of the DataFrame to column 'A'
df.set_index('A', inplace=True)
```

```
Original DataFrame:
A B C D
0 52 42 45 9
1 83 16 6 59
2 41 7 6 69
3 70 97 42 32
4 50 47 56 22
5 90 14 85 22
DataFrame with new index:
B C D
A
52 42 45 9
83 16 6 59
41 7 6 69
70 97 42 32
50 47 56 22
90 14 85 22
```

Task 2: Create a DataFrame with specified columns and index, and access an element.

```
# Create a Pandas DataFrame with specified columns and index
df = pd.DataFrame(np.random.randint(1, 100, size=(3, 3)), columns=['A', 'B', 'C'],
index=['X', 'Y', 'Z'])
# Access the element at row 'Y' and column 'B'
element = df.at['Y', 'B'] # .at is used for fast access to a single value
```

```
Original DataFrame:
A B C
X 97 97 39
Y 74 25 8
Z 68 38 32
Element at row 'Y' and column 'B': 25
```

2. DataFrame Operations

Task 1: Create a DataFrame with 3 columns and 5 rows filled with random integers and add a new column.

```
# Create a Pandas DataFrame with 3 columns and 5 rows filled with random integers
df = pd.DataFrame(np.random.randint(1, 100, size=(5, 3)), columns=['A', 'B', 'C'])
# Add a new column 'D' which is the product of columns 'A' and 'B'
df['D'] = df['A'] * df['B'] # Element-wise multiplication
```

```
Original DataFrame:
A B C
0 66 14 24
1 36 5 91
2 16 94 19
3 66 11 52
4 13 10 74
```

```
DataFrame with new column:
A B C D
0 66 14 24 924
1 36 5 91 180
2 16 94 19 1504
3 66 11 52 726
4 13 10 74 130
```

Task 2: Create a DataFrame with 3 columns and 4 rows filled with random integers and compute sums.

```
# Create a Pandas DataFrame with 3 columns and 4 rows filled with random integers
df = pd.DataFrame(np.random.randint(1, 100, size=(4, 3)), columns=['A', 'B', 'C'])
# Compute the row-wise and column-wise sum
row_sum = df.sum(axis=1) # Sum across rows (axis=1)
column_sum = df.sum() # Sum across columns (default axis=0)
```

```
Original DataFrame:
A B C
0 42 67 11
1 90 56 91
2 55 15 20
3 83 15 88
Row-wise sum:
0 120
1 237
2 90
3 186
dtype: int64
Column-wise sum:
A 270
B 153
C 210
dtype: int64
```

3. Data Cleaning

Task 1: Create a DataFrame with NaN values and fill them with the mean of the respective columns.

```
# Create a Pandas DataFrame with 3 columns and 5 rows filled with random integers
df = pd.DataFrame(np.random.randint(1, 100, size=(5, 3)), columns=['A', 'B', 'C'])
# Introduce NaN values
df.iloc[0, 1] = np.nan
df.iloc[2, 2] = np.nan
df.iloc[4, 0] = np.nan
# Fill the NaN values with the mean of the respective columns
df.fillna(df.mean(), inplace=True) # Fill NaN with column means
```

```
DataFrame with NaN values filled:
A B C
0 89.0 47.0 62.0
1 25.0 30.0 68.0
2 54.0 62.0 67.0
3 92.0 39.0 54.0
4 65.0 57.0 84.0
```

Task 2: Create a DataFrame with NaN values and drop the rows with any NaN values.

```
# Create a Pandas DataFrame with 4 columns and 6 rows filled with random integers
df = pd.DataFrame(np.random.randint(1, 100, size=(6, 4)), columns=['A', 'B', 'C', 'D'])
# Introduce NaN values
df.iloc[1, 2] = np.nan
df.iloc[3, 0] = np.nan
df.iloc[5, 1] = np.nan
# Drop the rows with any NaN values
df.dropna(inplace=True) # Remove rows with NaN values
```

```
DataFrame with NaN values dropped:
A B C D
```

```
0 37.0 87.0 78.0 59
2 38.0 6.0 46.0 93
4 13.0 72.0 95.0 55
```

4. Data Aggregation

Task 1: Create a DataFrame with 'Category' and 'Value', and compute sum and mean.

```
# Create a Pandas DataFrame with 'Category' and 'Value'
df = pd.DataFrame({'Category': np.random.choice(['A', 'B', 'C'], size=10), 'Value':
np.random.randint(1, 100, size=10)})
# Group the DataFrame by 'Category' and compute the sum and mean of 'Value' for each
category
grouped = df.groupby('Category')['Value'].agg(['sum', 'mean']) # Aggregate functions on
grouped data

Grouped DataFrame:
sum mean
Category
A 140 35.000000
B 64 21.333333
C 176 58.666667
```

Task 2: Create a DataFrame with 'Product', 'Category', and 'Sales', and compute total sales.

```
# Create a Pandas DataFrame with 'Product', 'Category', and 'Sales'
df = pd.DataFrame({'Product': np.random.choice(['Prod1', 'Prod2', 'Prod3'], size=10),
'Category': np.random.choice(['A', 'B', 'C'], size=10), 'Sales': np.random.randint(1,
100, size=10)})
# Group the DataFrame by 'Category' and compute the total sales for each category
grouped = df.groupby('Category')['Sales'].sum() # Sum sales by category

Grouped DataFrame:
Category
A 125
B 270
C 162
Name: Sales, dtype: int32
```

5. Merging DataFrames

Task 1: Create two DataFrames with a common column and merge them.

```
# Create two Pandas DataFrames with a common column
df1 = pd.DataFrame({'Key': ['A', 'B', 'C', 'D'], 'Value1': np.random.randint(1, 100,
size=4)})
df2 = pd.DataFrame({'Key': ['A', 'B', 'C', 'E'], 'Value2': np.random.randint(1, 100,
size=4)})
# Merge the DataFrames using the common column
merged = pd.merge(df1, df2, on='Key') # Merge on 'Key' column

DataFrame 1:
Key Value1
0 A 2
1 B 95
2 C 92
3 D 41
DataFrame 2:
Key Value2
0 A 5
1 B 69
2 C 35
3 E 26
Merged DataFrame:
Key Value1 Value2
```

```
0 A 2 5
1 B 95 69
2 C 92 35
```

Task 2: Create two DataFrames with different columns and concatenate them.

```
# Create two Pandas DataFrames with different columns
df1 = pd.DataFrame({'A': np.random.randint(1, 100, size=3), 'B': np.random.randint(1,
100, size=3}})
df2 = pd.DataFrame({'C': np.random.randint(1, 100, size=3), 'D': np.random.randint(1,
100, size=3}})
# Concatenate the DataFrames along the rows
concat_rows = pd.concat([df1, df2], axis=0) # Stack DataFrames vertically
# Concatenate the DataFrames along the columns
concat_columns = pd.concat([df1, df2], axis=1) # Join DataFrames side by side
```

```
Concatenated DataFrame (rows):
A B C D
0 78.0 81.0 NaN NaN
1 21.0 4.0 NaN NaN
2 25.0 7.0 NaN NaN
0 NaN NaN 11.0 21.0
1 NaN NaN 45.0 94.0
2 NaN NaN 79.0 72.0
Concatenated DataFrame (columns):
A B C D
0 78 81 11 21
1 21 4 45 94
2 25 7 79 72
```

6. Time Series Analysis

Task 1: Create a DataFrame with a datetime index and compute the monthly mean.

```
# Create a Pandas DataFrame with a datetime index and one column filled with random
integers
date_rng = pd.date_range(start='2022-01-01', end='2022-12-31', freq='D')
df = pd.DataFrame(date_rng, columns=['date'])
df['data'] = np.random.randint(0, 100, size=(len(date_rng)))
df.set_index('date', inplace=True) # Set the date as the index
# Resample the DataFrame to compute the monthly mean of the values
monthly_mean = df.resample('M').mean() # Resample by month and calculate mean
```

```
Monthly mean DataFrame:
data
date
2022-01-31 48.483871
2022-02-28 47.250000
2022-03-31 54.032258
2022-04-30 44.366667
2022-05-31 52.419355
2022-06-30 49.900000
2022-07-31 49.741935
2022-08-31 52.806452
2022-09-30 53.833333
2022-10-31 48.838710
2022-11-30 51.033333
2022-12-31 55.612903
```

Task 2: Create a DataFrame with a datetime index and compute the rolling mean.

```
# Create a Pandas DataFrame with a datetime index ranging from '2021-01-01' to
'2021-12-31'
date_rng = pd.date_range(start='2021-01-01', end='2021-12-31', freq='D')
df = pd.DataFrame(date_rng, columns=['date'])
df['data'] = np.random.randint(0, 100, size=(len(date_rng)))
df.set_index('date', inplace=True) # Set the date as the index
# Compute the rolling mean with a window of 7 days
```

```
rolling_mean = df.rolling(window=7).mean() # Calculate rolling mean over a 7-day window
```

```
Rolling mean DataFrame:
data
date
2021-01-01 NaN
2021-01-02 NaN
2021-01-03 NaN
2021-01-04 NaN
2021-01-05 NaN
... ...
2021-12-27 40.571429
2021-12-28 38.571429
2021-12-29 44.000000
2021-12-30 49.571429
2021-12-31 51.285714
```

7. MultiIndex DataFrame

Task 1: Create a MultiIndex DataFrame and perform indexing.

```
# Create a Pandas DataFrame with a MultiIndex (hierarchical index)
arrays = [['A', 'A', 'B', 'B'], ['one', 'two', 'one', 'two']]
index = pd.MultiIndex.from_arrays(arrays, names=('Category', 'SubCategory'))
df = pd.DataFrame(np.random.randint(1, 100, size=(4, 3)), index=index,
columns=['Value1', 'Value2', 'Value3'])
# Basic indexing and slicing operations
df.loc['A'] # Access all rows with Category 'A'
df.loc[('B', 'two')] # Access specific row with Category 'B' and SubCategory 'two'
```

```
MultiIndex DataFrame:
Value1 Value2 Value3
Category SubCategory
A one 12 97 2
two 29 87 33
B one 59 75 35
two 96 28 67
Indexing at Category 'A':
Value1 Value2 Value3
SubCategory
one 12 97 2
two 29 87 33
Slicing at Category 'B' and SubCategory 'two':
Value1 96
Value2 28
Value3 67
Name: (B, two), dtype: int32
```

Task 2: Create a MultiIndex DataFrame and compute the sum of values.

```
# Create a Pandas DataFrame with MultiIndex consisting of 'Category' and 'SubCategory'
arrays = [['A', 'A', 'B', 'B', 'C', 'C'], ['one', 'two', 'one', 'two', 'one', 'two']]
index = pd.MultiIndex.from_arrays(arrays, names=('Category', 'SubCategory'))
df = pd.DataFrame(np.random.randint(1, 100, size=(6, 3)), index=index,
columns=['Value1', 'Value2', 'Value3'])
# Compute the sum of values for each 'Category' and 'SubCategory'
sum_values = df.groupby(['Category', 'SubCategory']).sum() # Group by both indices and
sum
```

```
Sum of values:
Value1 Value2 Value3
Category SubCategory
A one 50 63 87
two 84 21 39
B one 71 69 5
two 13 55 1
C one 50 31 88
two 48 13 28
```

8. Pivot Tables

Task 1: Create a pivot table to compute the sum of 'Value' for each 'Category' by 'Date'.

```
# Create a Pandas DataFrame with columns 'Date', 'Category', and 'Value'
date_rng = pd.date_range(start='2022-01-01', end='2022-01-10', freq='D')
df = pd.DataFrame({'Date': np.random.choice(date_rng, size=20), 'Category':
np.random.choice(['A', 'B', 'C'], size=20), 'Value': np.random.randint(1, 100,
size=20)})
# Create a pivot table to compute the sum of 'Value' for each 'Category' by 'Date'
pivot_table = df.pivot_table(values='Value', index='Date', columns='Category',
aggfunc='sum') # Pivot table with sum aggregation
```

```
Pivot Table:
Category A B C
Date
2022-01-02 22.0 7.0 95.0
2022-01-03 NaN 130.0 47.0
2022-01-04 206.0 NaN 31.0
2022-01-05 NaN NaN 12.0
2022-01-06 NaN 42.0 98.0
2022-01-08 26.0 16.0 NaN
2022-01-09 66.0 NaN NaN
2022-01-10 95.0 NaN NaN
```

Task 2: Create a pivot table to compute the mean 'Revenue' for each 'Quarter' by 'Year'.

```
# Create a Pandas DataFrame with columns 'Year', 'Quarter', and 'Revenue'
df = pd.DataFrame({'Year': np.random.choice([2020, 2021, 2022], size=12), 'Quarter':
np.random.choice(['Q1', 'Q2', 'Q3', 'Q4'], size=12), 'Revenue': np.random.randint(1,
1000, size=12)})
# Create a pivot table to compute the mean 'Revenue' for each 'Quarter' by 'Year'
pivot_table = df.pivot_table(values='Revenue', index='Year', columns='Quarter',
aggfunc='mean') # Mean revenue by year and quarter
```

```
Pivot Table:
Quarter Q2 Q3 Q4
Year
2020 517.0 806.0 478.5
2021 561.5 757.0 561.0
2022 NaN NaN 991.0
```

9. Applying Functions

Task 1: Apply a function that doubles the values of the DataFrame.

```
# Create a Pandas DataFrame with 3 columns and 5 rows filled with random integers
df = pd.DataFrame(np.random.randint(1, 100, size=(5, 3)), columns=['A', 'B', 'C'])
print('Original DataFrame:')
print(df)
# Apply a function that doubles the values of the DataFrame
df_doubled = df.applymap(lambda x: x * 2) # Apply function to each element
print('Doubled DataFrame:')
print(df_doubled)
```

```
Original DataFrame:
A B C
0 46 15 33
1 7 58 23
2 58 80 32
3 34 81 86
4 37 85 73
Doubled DataFrame:
A B C
```

```
0 92 30 66
1 14 116 46
2 116 160 64
3 68 162 172
4 74 170 146
```

Task 2: Apply a lambda function to create a new column that is the sum of the existing columns.

```
# Create a Pandas DataFrame with 3 columns and 6 rows filled with random integers
df = pd.DataFrame(np.random.randint(1, 100, size=(6, 3)), columns=['A', 'B', 'C'])
print('Original DataFrame:')
print(df)
# Apply a lambda function to create a new column 'D' that is the sum of the existing
columns
df['D'] = df.apply(lambda row: row['A'] + row['B'] + row['C'], axis=1) # Sum across rows
print('DataFrame with new column D:')
print(df)
```

```
Original DataFrame:
A B C
0 45 67 23
1 12 34 56
2 78 90 12
3 23 45 67
4 89 12 34
5 56 78 90
DataFrame with new column D:
A B C D
0 45 67 23 135
1 12 34 56 102
2 78 90 12 180
3 23 45 67 135
4 89 12 34 135
5 56 78 90 224
```

10. Working with Text Data

Task 1: Create a Pandas Series with 5 random text strings. Convert all the strings to uppercase.

```
# Create a Pandas Series with 5 random text strings
text_data = pd.Series(['apple', 'banana', 'cherry', 'date', 'elderberry'])
print('Original Series:')
print(text_data)
# Convert all the strings to uppercase
uppercase_data = text_data.str.upper() # Convert strings to uppercase
print('Uppercase Series:')
print(uppercase_data)
```

```
Original Series:
0 apple
1 banana
2 cherry
3 date
4 elderberry
dtype: object
Uppercase Series:
0 APPLE
1 BANANA
2 CHERRY
3 DATE
4 ELDERBERRY
dtype: object
```

Task 2: Create a Pandas Series with 5 random text strings. Extract the first three characters of each string.

```
# Create a Pandas Series with 5 random text strings
text_data = pd.Series(['apple', 'banana', 'cherry', 'date', 'elderberry'])
```

```
print('Original Series:')
print(text_data)
# Extract the first three characters of each string
first_three_chars = text_data.str[:3] # Slice the first three characters
print('First three characters:')
print(first_three_chars)
```

```
Original Series:
0 apple
1 banana
2 cherry
3 date
4 elderberry
dtype: object
First three characters:
0 app
1 ban
2 che
3 dat
4 eld
dtype: object
```