ASSIGNMENT - 11(Pandas)

Solution/Ans by - Pranav Rode(29)

1. Define the Pandas/Python pandas?

In [ ]:
```
Definition of Pandas/Python pandas:

- Pandas is an open-source data manipulation and analysis library for Python.
- It provides easy-to-use data structures and functions for working with structured data,
such as spreadsheets or SQL tables.
- Pandas is built on top of the NumPy library and is particularly well-suited
for tasks like data cleaning, transformation, and analysis.
```

2. What are the different types of Data Structures in Pandas?

In [ ]:
```
Different Types of Data Structures in Pandas:

Pandas primarily offers two main data structures:

a. Series: A one-dimensional labeled array capable of holding any data type.
           It is similar to a column in a spreadsheet or a single column in a SQL table.

b. DataFrame: A two-dimensional labeled data structure with columns of
              potentially different data types. It is like a spreadsheet or
              SQL table, where you have rows and columns.
```

3. Explain Series and DataFrame In Pandas

In [ ]:
```
Series and DataFrame in Pandas:

Series: A Series is essentially a one-dimensional array with labels (index)
        for each element. It can hold data of various types (integers, strings, floats, etc.).
        You can think of it as a column in a spreadsheet or a single column from a SQL table.

DataFrame: A DataFrame is a two-dimensional tabular data structure where data is organized
           in rows and columns. Each column can be of a different data type.
           It's similar to a spreadsheet or SQL table.
           You can perform various data manipulations like filtering, joining, grouping,
           and aggregating on DataFrames.
```

4. How Can You Create An Empty DataFrame and series in Pandas?

In [1]:
```
# Empty DataFrame:

import pandas as pd
df = pd.DataFrame()  # Empty dataframe 'df'
```

In [2]:
```
# Empty Series:

import pandas as pd
series = pd.Series()  # Empty Series 'series'
```

```
C:\Users\prana\AppData\Local\Temp\ipykernel_140\620040212.py:4: FutureWarning: The default dtype for empty Series will be 'objec
t' instead of 'float64' in a future version. Specify a dtype explicitly to silence this warning.
  series = pd.Series()  # Empty Series 'series'
```

5. How to check an empty DataFrame?

In [ ]:
```
To check if a DataFrame is empty, you can use the empty attribute of the DataFrame,
which returns a boolean value indicating whether the DataFrame has no data (i.e., it's empty) or not.
```

In [3]:
```
# Example:
import pandas as pd
# Create an empty DataFrame
df = pd.DataFrame()
# Check if the DataFrame is empty
df.empty
```

Out[3]:
```
True
```

6. What Are The Most Important Features Of The Pandas Library?

In [ ]:
```
Most Important Features of the Pandas Library:

Pandas is a versatile library with several important features, some of which include:

- Data Structures: Pandas provides powerful data structures like Series and DataFrame,
```

```
                  which make it easy to work with structured data.

- Data Cleaning and Preprocessing: It offers functions for handling missing data, filtering, and
                                   transforming data, making data cleaning and preprocessing efficient.

- Indexing and Selection: Pandas allows for intuitive indexing and selection of
                          data elements using labels or positional indexing.

- Aggregation and Grouping: You can easily aggregate and group data using Pandas, which is
                            crucial for summary statistics and data analysis.

- Merging and Joining: Pandas provides various ways to combine and merge data from
                       different sources, similar to SQL joins.

- Time Series Analysis: It has excellent support for time series data,
                        making it suitable for financial and temporal data analysis.

- Input/Output: Pandas supports reading and writing data in various formats,
                including CSV, Excel, SQL databases, and more.

- Plotting: It offers basic data visualization capabilities through integration with Matplotlib.

- Flexibility: You can handle data of different types and shapes,
               making Pandas flexible for various data analysis tasks.
```

7. How Will You Explain Reindexing In Pandas?

In [ ]:
```
Reindexing in Pandas refers to the process of modifying the index (row labels)
of a DataFrame or Series to match a new set of labels.
It is a fundamental operation in Pandas that allows you to realign
the data to a different index, which can be useful for various purposes, such as:

- Changing the Order of Rows: You can reorder the rows of your DataFrame
                              by specifying a new order for the index.

- Introducing Missing Data: If you have a new set of labels, some of the labels in your
                            DataFrame may not exist in the new index. Reindexing can introduce
                            missing values (NaN) for these labels or handle them based on your specifications.

- Alignment: Reindexing can be used to align multiple DataFrames or Series objects
             based on a common set of labels. This is particularly useful when
             performing operations on multiple data sources with potentially different indexes.

The primary method for reindexing in Pandas is the reindex() method.
```

In [8]:
```
# Here's a basic explanation of how to use it:

import pandas as pd

# Create a sample DataFrame
data = {'A': [1, 2, 3], 'B': [4, 5, 6]}
df = pd.DataFrame(data, index=['row1', 'row2', 'row3'])

# Define a new index
new_index = ['row3', 'row2', 'row4']

# Reindex the DataFrame
df_reindexed = df.reindex(new_index)

df_reindexed
```

Out[8]:

	A	B
row3	3.0	6.0
row2	2.0	5.0
row4	NaN	NaN

8. What are the different ways of creating DataFrame in pandas? Explain with examples.

In [ ]:
```
Different Ways of Creating a DataFrame in Pandas:

There are several ways to create a DataFrame in Pandas:
```

In [10]:
```
# (A) From a Dictionary:
import pandas as pd

data = {'Name': ['Alice', 'Bob', 'Charlie'],
        'Age': [25, 30, 35]}

df = pd.DataFrame(data)
df
```

Out[10]:

	Name	Age
0	Alice	25
1	Bob	30
2	Charlie	35

In [11]:
```
# (B) From a List of Lists:

import pandas as pd

data = [['Alice', 25], ['Bob', 30], ['Charlie', 35]]
df = pd.DataFrame(data, columns=['Name', 'Age'])
df
```

Out[11]:

	Name	Age
0	Alice	25
1	Bob	30
2	Charlie	35

In [ ]:
```
# (C) From a CSV File:
import pandas as pd

df = pd.read_csv('data.csv')
```

In [12]:
```
# (D) From a NumPy Array:

import pandas as pd
import numpy as np

data = np.array([[1, 2, 3], [4, 5, 6]])
df = pd.DataFrame(data, columns=['A', 'B', 'C'])
df
```

Out[12]:

	A	B	C
0	1	2	3
1	4	5	6

9. Create a DataFrame using List.

In [13]:
```
# Here's an example of creating a DataFrame using a list:
import pandas as pd

data = [['Alice', 25], ['Bob', 30], ['Charlie', 35]]
df = pd.DataFrame(data, columns=['Name', 'Age'])
df
```

Out[13]:

	Name	Age
0	Alice	25
1	Bob	30
2	Charlie	35

10. Create a DataFrame using Numpy Functions.

In [16]:
```
# Creating a DataFrame Using NumPy Functions:
import pandas as pd
import numpy as np

data = np.random.randint(0, 100, size=(3, 4))  # Creating a 3x4 array of random integers
df = pd.DataFrame(data, columns=['A', 'B', 'C', 'D'])
df
```

Out[16]:

	A	B	C	D
0	90	65	28	65
1	95	74	72	60
2	91	18	52	69

11. How to convert a NumPy array to a DataFrame of a given shape?

In [17]:
```
# You can convert a NumPy array to a DataFrame with a given shape using
# the reshape() method from NumPy to reshape the array to the desired shape
# and then create a DataFrame from it.
# Here's an example:
```

```
import pandas as pd
import numpy as np

# Create a NumPy array
numpy_array = np.array([1, 2, 3, 4, 5, 6])  # Example array

# Reshape the NumPy array to the desired shape, e.g., (2, 3)
reshaped_array = numpy_array.reshape(2, 3)

# Create a DataFrame from the reshaped array
df = pd.DataFrame(reshaped_array, columns=['A', 'B', 'C'])
df
```

Out[17]:

	A	B	C
0	1	2	3
1	4	5	6

12. Create a DataFrame using Dictionary with a list and arrays

In [ ]:
```
You can create a DataFrame using a dictionary where
the values are lists or arrays. Each key-value pair in
the dictionary represents a column in the DataFrame.
```

In [25]:
```
# Here's an example:

import pandas as pd

data = { 'Name': ['Alice', 'Bob', 'Charlie'],
         'Age': [25, 30, 35],
         'Scores': [90, 85, 88]      }

df = pd.DataFrame(data)
df
```

Out[25]:

	Name	Age	Scores
0	Alice	25	90
1	Bob	30	85
2	Charlie	35	88

13. How To Create A Copy Of The Series and DataFrame in Pandas?

In [ ]:
```
To create a copy of a Series or DataFrame in Pandas, you can use the copy() method.
This method creates a deep copy, ensuring that changes made to the original Series
or DataFrame do not affect the copy, and vice versa.
```

In [26]:
```
# Here's how to use it:

import pandas as pd

# Create a Series
original_series = pd.Series([1, 2, 3, 4, 5])

# Create a copy of the Series
copied_series = original_series.copy()

# Modify the original Series
original_series[0] = 10

# Check the copied Series
copied_series
```

Out[26]:
```
0    1
1    2
2    3
3    4
4    5
dtype: int64
```

In [27]:
```
import pandas as pd

# Create a DataFrame
data = {'A': [1, 2, 3], 'B': [4, 5, 6]}
original_df = pd.DataFrame(data)

# Create a copy of the DataFrame
copied_df = original_df.copy()

# Modify the original DataFrame
original_df['A'][0] = 10

# Check the copied DataFrame
copied_df
```

Out[27]:

	A	B
0	1	4
1	2	5
2	3	6

14. How Will You Add An Index, Row, Or Column To A DataFrame In Pandas?

In [ ]:
```
1. Adding an Index: You typically don't need to explicitly add an index to a DataFrame,
                    as Pandas automatically assigns a default integer-based index (0, 1, 2, ...)
                    when you create a DataFrame. However, you can set a specific column as the
                    index using the set_index() method:
```

In [28]:
```
import pandas as pd

# Create a DataFrame
data = {'Name': ['Alice', 'Bob', 'Charlie'],
        'Age': [25, 30, 35]}

df = pd.DataFrame(data)

# Set the 'Name' column as the index
df = df.set_index('Name')
df
```

Out[28]:

	Age
Name	
Alice	25
Bob	30
Charlie	35

In [ ]:
```
2. Adding a Row: To add a new row to a DataFrame, you can use the append() method.
                 You need to provide a dictionary with values for each column in the row:
```

In [29]:
```
import pandas as pd

# Create a DataFrame
data = {'Name': ['Alice', 'Bob', 'Charlie'],
        'Age': [25, 30, 35]}

df = pd.DataFrame(data)

# Add a new row
new_row = {'Name': 'David', 'Age': 28}
df = df.append(new_row, ignore_index=True)
df
```

```
C:\Users\prana\AppData\Local\Temp\ipykernel_140\3993540971.py:11: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.
  df = df.append(new_row, ignore_index=True)
```

Out[29]:

	Name	Age
0	Alice	25
1	Bob	30
2	Charlie	35
3	David	28

In [ ]:
```
3. Adding a Column: To add a new column to a DataFrame, you can simply
                    assign a new Series or list to it. For example:
```

In [31]:
```
import pandas as pd

# Create a DataFrame
data = {'Name': ['Alice', 'Bob', 'Charlie'],
        'Age': [25, 30, 35]}

df = pd.DataFrame(data)

# Add a new 'City' column
df['City'] = ['New York', 'San Francisco', 'Los Angeles']
df
```

Out[31]:

	Name	Age	City
0	Alice	25	New York
1	Bob	30	San Francisco
2	Charlie	35	Los Angeles

15. What Method Will You Use To Rename The Index Or Columns Of Pandas DataFrame?

In [ ]:
```
To rename the index or columns of a Pandas DataFrame, you can use the rename() method.
This method allows you to provide new names for the index or columns,
and you can specify whether you want to rename the index, columns, or both.
Here's how to use it:
```

In [ ]:
```
1. Renaming Index: To rename the index of a DataFrame, you can use the
                   rename() method with the index parameter.
```

In [34]:
```
# For example:
import pandas as pd

# Create a DataFrame with an existing index
data = {'A': [1, 2, 3], 'B': [4, 5, 6]}
df = pd.DataFrame(data, index=['X', 'Y', 'Z'])

# Rename the index
df = df.rename(index={'X': 'Row1', 'Y': 'Row2', 'Z': 'Row3'})
df
```

Out[34]:

	A	B
Row1	1	4
Row2	2	5
Row3	3	6

In [ ]:
```
2. Renaming Columns: To rename columns in a DataFrame, you can use the
                     rename() method with the columns parameter.
```

In [35]:
```
# For example:
import pandas as pd

# Create a DataFrame with existing column names
data = {'A': [1, 2, 3], 'B': [4, 5, 6]}
df = pd.DataFrame(data)

# Rename the columns
df = df.rename(columns={'A': 'Column1', 'B': 'Column2'})
df
```

Out[35]:

	Column1	Column2
0	1	4
1	2	5
2	3	6

In [ ]:
```
3. Renaming Both Index and Columns: You can also rename both the index and columns
                                    simultaneously by providing both index and columns
                                    parameters to the rename() method:
```

In [10]:
```
# For example:
import pandas as pd

# Create a DataFrame with an existing index and columns
data = {'A': [1, 2, 3], 'B': [4, 5, 6]}
df = pd.DataFrame(data, index=['X', 'Y', 'Z'])

# Rename both index and columns
df = df.rename(index={'X': 'Row1', 'Y': 'Row2', 'Z': 'Row3'},
               columns={'A': 'Column1', 'B': 'Column2'})

df
```

Out[10]:

	Column1	Column2
Row1	1	4
Row2	2	5
Row3	3	6

16. How Can You Iterate Over DataFrame In Pandas?

In [19]:
```
import pandas as pd

# Create a sample DataFrame
data = {'Name': ['Alice', 'Bob', 'Charlie'],
        'Age': [25, 30, 35]}

df = pd.DataFrame(data)
df
```

Out[19]:

	Name	Age
0	Alice	25
1	Bob	30
2	Charlie	35

In [27]:
```
# Using iterrows()
for index, row in df.iterrows():
    # Access data using row['column_name']
    print(index)
    print(row.Name, row.Age)
```

```
0
Alice 25
1
Bob 30
2
Charlie 35
```

In [40]:
```
# Using itertuples()
for row in df.itertuples():
    # Access data using row.column_name
    print(row.Name, row.Age)
```

```
Alice 25
Bob 30
Charlie 35
```

In [43]:
```
# Using items() or iteritems()
for column, values in df.items():
    # Access data using values
    print(column)
    print(values)
```

```
Name
0      Alice
1        Bob
2    Charlie
Name: Name, dtype: object
Age
0    25
1    30
2    35
Name: Age, dtype: int64
```

In [47]:
```
# Using List Comprehension
[value for value in df['Name']]  # Iterate over a column
```

Out[47]:
```
['Alice', 'Bob', 'Charlie']
```

In [48]:
```
# Using filter()
df.filter(items=['Name','Age'])  # Select specific columns
```

Out[48]:

	Name	Age
0	Alice	25
1	Bob	30
2	Charlie	35

In [ ]:
```
apply(): Applies a function along either the rows (axis=1) or columns (axis=0) of the DataFrame.
df.apply(function, axis=1)  # Apply function to each row
df.apply(function, axis=0)  # Apply function to each column
```

In [ ]:
```
applymap(): Applies a function element-wise to all elements in the DataFrame.
df.applymap(function)  # Apply function to each element
```

17. How to create an array from DataFrame

In [10]:
```
import pandas as pd

# Create a sample DataFrame
data = pd.DataFrame( {'Name': ['Alice', 'Bob', 'Charlie'],
                      'Age': [25, 30, 35]} )

df = data
arr = df.to_numpy()
print('\nNumpy Array\n----------\n', arr)
print(type(arr))
```

```
Numpy Array
----------
 [['Alice' 25]
 ['Bob' 30]
 ['Charlie' 35]]
<class 'numpy.ndarray'>
```

In [11]:
```
import pandas as pd

# initialize a dataframe
df = pd.DataFrame(
    [[1, 2, 3],
     [4, 5, 6],
     [7, 8, 9],
     [10, 11, 12]],
    columns=['a', 'b', 'c'])

# convert dataframe to numpy array
arr = df.to_numpy()

print('\nNumpy Array\n----------\n', arr)
print(type(arr))
```

```
Numpy Array
----------
 [[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]]
<class 'numpy.ndarray'>
```

18. How to create a list from DataFrame

In [13]:
```
import pandas as pd

df = pd.DataFrame({'product': ['Tablet', 'Printer', 'Laptop', 'Monitor'],
        'price': [250, 100, 1200, 300]
        })

products_list = df.values.tolist()
products_list
```

Out[13]:
```
[['Tablet', 250], ['Printer', 100], ['Laptop', 1200], ['Monitor', 300]]
```

19. How to Reset the dataframes index?

In [18]:
```
import pandas as pd

# Create own index
index = ['a', 'b', 'c', 'd', 'e']

# Convert the dictionary into DataFrame
# Make Own Index and Removing Default index
df = pd.DataFrame({'Name':['Jai', 'Princi', 'Gaurav', 'Anuj', 'Geeku'],
                   'Age':[27, 24, 22, 32, 15],
                   'Address':['Delhi', 'Kanpur', 'Allahabad', 'Kannauj', 'Noida'],
                   'Qualification':['Msc', 'MA', 'MCA', 'Phd', '10th'] },
                    index)
df.reset_index(inplace = True, drop = True)
df
```

Out[18]:

	Name	Age	Address	Qualification
0	Jai	27	Delhi	Msc
1	Princi	24	Kanpur	MA
2	Gaurav	22	Allahabad	MCA
3	Anuj	32	Kannauj	Phd
4	Geeku	15	Noida	10th

20. Write a Pandas program to count the number of rows and columns of a DataFrame

In [39]:
```
# Using shape()
import pandas as pd
df = pd.DataFrame({'Name':['Jai', 'Princi', 'Gaurav', 'Anuj', 'Geeku'],
                   'Age':[27, 24, 22, 32, 15],
                   'Address':['Delhi', 'Kanpur', 'Allahabad', 'Kannauj', 'Noida'],
                   'Qualification':['Msc', 'MA', 'MCA', 'Phd', '10th'] }
                    )
print("Number of rows:",df.shape[0])
print("Number of columns:",df.shape[1])
```

```
Number of rows: 5
Number of columns: 4
```

In [40]:
```
# Using len() and index()
import pandas as pd
df = pd.DataFrame({'Name':['Jai', 'Princi', 'Gaurav', 'Anuj', 'Geeku'],
                   'Age':[27, 24, 22, 32, 15],
                   'Address':['Delhi', 'Kanpur', 'Allahabad', 'Kannauj', 'Noida'],
                   'Qualification':['Msc', 'MA', 'MCA', 'Phd', '10th'] }
                    )
```

```
rows = len(df.index)
columns = len(df.columns)

print("Number of rows:", rows)
print("Number of columns:", columns)
```

```
Number of rows: 5
Number of columns: 4
```

21. Write a Pandas program to add, subtract, multiple, and divide two Pandas Series

In [61]:
```
import pandas as pd
import numpy as np
df1 = pd.Series([np.random.randint(1,35,5)])
df2 = pd.Series([np.random.randint(1,35,5)])
print("df1",df1)
print("df2",df2)

print("Addition:",df1+df2)
print("Subtraction:",df1-df2)
print("Multiplication:",df1*df2)
print("Division:",df1/df2)
```

```
df1 0    [13, 17, 28, 6, 25]
dtype: object
df2 0    [27, 25, 34, 4, 23]
dtype: object
Addition: 0    [40, 42, 62, 10, 48]
dtype: object
Subtraction: 0    [-14, -8, -6, 2, 2]
dtype: object
Multiplication: 0    [351, 425, 952, 24, 575]
dtype: object
Division: 0    [0.48148148148148145, 0.68, 0.8235294117647058...
dtype: object
```

22. Write a Pandas program to import excel data into a Pandas dataframe

In [8]:
```
import pandas as pd
# Use pandas.read_excel() to read the Excel file into a DataFrame
# You can specify the sheet_name if your Excel file has multiple sheets.
# By default, it reads the first sheet.
sheet_1 = "Emp_Records"
sheet_2 = "emp_salary"
sheet_3 = "emp_names"
df = pd.read_excel(r"C:\Users\prana\DS - Python\datasets\Emp_Records.xlsx",  sheet_name=sheet_1 )

# Now, you can work with the 'df' DataFrame as needed
# For example, you can display the first few rows of the DataFrame
df.head()
```

Out[8]:

	Emp ID	First Name	Age in Yrs	Weight in Kgs	Age in Company	Salary	City
0	677509	Lois	36.36	60	13.68	168251	Denver
1	940761	Brenda	47.02	60	9.01	51063	Stonewall
2	428945	Joe	54.15	68	0.98	50155	Michigantown
3	408351	Diane	39.67	51	18.30	180294	Hydetown
4	193819	Benjamin	40.31	58	4.01	117642	Fremont

23. Write a Pandas program to read specific columns from a given excel file

In [9]:
```
import pandas as pd
# Use pandas.read_excel() to read the Excel file into a DataFrame
# You can specify the sheet_name if your Excel file has multiple sheets.
# By default, it reads the first sheet.
sheet_1 = "Emp_Records"
sheet_2 = "emp_salary"
sheet_3 = "emp_names"

# Use pandas.read_excel() with the usecols parameter to read specific columns
df = pd.read_excel(r"C:\Users\prana\DS - Python\datasets\Emp_Records.xlsx",  sheet_name=sheet_1,
                   usecols=["First Name", "Salary"] )

# Now, you can work with the 'df' DataFrame as needed
# For example, you can display the first few rows of the DataFrame
df.head()
```

Out[9]:

	First Name	Salary
0	Lois	168251
1	Brenda	51063
2	Joe	50155
3	Diane	180294
4	Benjamin	117642

24. Write a Pandas program to find the sum, mean, max, min value of of a dataset.

In [12]:
```
import pandas as pd

# Create a sample DataFrame (replace this with your actual dataset)
data = {
    'Col1': [10, 15, 20, 25, 30],
    'Col2': [5, 12, 18, 27, 8],
    'Col3': [22, 17, 12, 31, 5]
}

df = pd.DataFrame(data)

# Calculate the sum, mean, max, and min values for each column
sum_values = df.sum()
mean_values = df.mean()
max_values = df.max()
min_values = df.min()

# Print the results
print("Sum of values:")
print(sum_values)
print("\nMean of values:")
print(mean_values)
print("\nMax values:")
print(max_values)
print("\nMin values:")
print(min_values)
```

```
Sum of values:
Col1    100
Col2     70
Col3     87
dtype: int64

Mean of values:
Col1    20.0
Col2    14.0
Col3    17.4
dtype: float64

Max values:
Col1    30
Col2    27
Col3    31
dtype: int64

Min values:
Col1    10
Col2     5
Col3     5
dtype: int64
```

25. How Can A DataFrame Be Converted To An Excel File and CSV file?

In [16]:
```
import pandas as pd

# Create a sample DataFrame (replace this with your actual DataFrame)
df = pd.DataFrame({'Column1': [1, 2, 3, 4, 5],
                   'Column2': ['A', 'B', 'C', 'D', 'E']})

# Specify the Excel file name (e.g., 'my_data.xlsx')
file_name = 'my_data.xlsx'

# Save the DataFrame to an Excel file
df.to_excel(file_name, index=False)  # Set index=False to exclude row numbers in the Excel file
print(f'DataFrame saved to {file_name}')
```

```
DataFrame saved to my_data.xlsx
```

In [17]:
```
import pandas as pd

# Create a sample DataFrame (replace this with your actual DataFrame)
df = pd.DataFrame({'Column1': [1, 2, 3, 4, 5],
                   'Column2': ['A', 'B', 'C', 'D', 'E']})

# Specify the csv file name (e.g., 'my_data.csv')
file_name = 'my_data.csv'
```

```
# Save the DataFrame to an Excel file
df.to_csv(file_name, index=False)  # Set index=False to exclude row numbers in the csv file
print(f'DataFrame saved to {file_name}')
```

```
DataFrame saved to my_data.csv
```

26. What is Groupby Function In Pandas? Explain with example

In [ ]:
```
The groupby() function in Pandas is a powerful tool for grouping and
aggregating data in a DataFrame based on one or more columns.
It allows you to split your data into groups based on specified
criteria and perform various operations within each group.
This function is often used in conjunction with aggregation functions
like sum(), mean(), count(), etc., to summarize data within each group.

Here's an explanation of the groupby() function with an example:
```

In [25]:
```
import pandas as pd

# Sample sales dataframe
df = pd.DataFrame({ 'Category': ['Electronics', 'Clothing', 'Electronics', 'Clothing', 'Electronics'],
                    'Product': ['Laptop', 'T-Shirt', 'Smartphone', 'Jeans', 'Tablet'],
                    'Price': [1000, 200, 800, 500, 300],
                    'Quantity': [3, 5, 2, 4, 1]
                  })

# Grouping by 'Category' and calculating total sales and average price within each group
category_summary = df.groupby('Category').agg({"Price": 'mean', "Quantity": 'sum'}).reset_index()
print(category_summary)
```

```
      Category  Price  Quantity
0     Clothing  350.0         9
1  Electronics  700.0         6
```

27. What is the use of apply function in pandas?

In [ ]:
```
The apply() function in Pandas is a versatile method that is used to apply a
custom function to one or more elements, rows, or columns of a DataFrame or Series.
It provides a flexible way to transform data in your DataFrame by applying a
user-defined function to each element or a specified axis.

Here are some common use cases and scenarios where the apply() function is valuable:
```

In [28]:
```
# Applying a Function Element-wise: You can use apply() to apply a
# custom function to each element in a DataFrame or Series.
import pandas as pd

df = pd.DataFrame({'A': [1, 2, 3, 4]})

# Applying a custom function to double each element
df['A'] = df['A'].apply(lambda x: x * 2)
```

In [32]:
```
# Applying a Function to Rows or Columns
import pandas as pd

df = pd.DataFrame({'A': [1, 2, 3], 'B': [4, 5, 6]})

# Applying a custom function to calculate the sum of each row
df['RowSum'] = df.apply(lambda row: row.sum(), axis=1)
df
```

Out[32]:

	A	B	RowSum
0	1	4	5
1	2	5	7
2	3	6	9

In [33]:
```
# Complex Data Transformations
import pandas as pd

df = pd.DataFrame({'A': [1, 2, 3], 'B': [4, 5, 6]})

# Applying a custom function for more complex transformations
def custom_function(row):
    return row['A'] * 2 + row['B']

df['NewColumn'] = df.apply(custom_function, axis=1)
df
```

Out[33]:

	A	B	NewColumn
0	1	4	6
1	2	5	9
2	3	6	12

In [35]:
```
# Handling Missing Data
import pandas as pd
import numpy as np

df = pd.DataFrame({'A': [1, np.nan, 3, 4]})

# Applying a custom function to replace missing values with 0
df['A'] = df['A'].apply(lambda x: x if not pd.isna(x) else 100)
df
```

Out[35]:

	A
0	1.0
1	100.0
2	3.0
3	4.0

In [37]:
```
# String Manipulations
import pandas as pd

df = pd.DataFrame({'Text': ['apple', 'banana', 'cherry']})

# Applying a custom function to convert text to uppercase
df['Text'] = df['Text'].apply(lambda x: x.upper())
df
```

Out[37]:

	Text
0	APPLE
1	BANANA
2	CHERRY

28. How to use apply() with lambda function?

In [44]:
```
# Using apply() with a Lambda Function on a Series:
import pandas as pd

# Create a sample Series
series = pd.Series([1, 2, 3, 4, 5])

# Apply a lambda function to double each element
doubled_series = series.apply(lambda x: x * 2)

doubled_series
```

Out[44]:
```
0     2
1     4
2     6
3     8
4    10
dtype: int64
```

In [45]:
```
# Applying lambda on speciic roow and column
import pandas as pd

# Create a sample DataFrame
data = {'A': [1, 2, 3], 'B': [4, 5, 6]}
df = pd.DataFrame(data)

# Apply a lambda function to a specific column ('A' in this case)
df['A'] = df['A'].apply(lambda x: x ** 2)

# Apply a lambda function to each row to calculate the sum of values in each row
df['RowSum'] = df.apply(lambda row: row['A'] + row['B'], axis=1)
df
```

Out[45]:

	A	B	RowSum
0	1	4	5
1	4	5	9
2	9	6	15

29. Explain is the use of info, describe, head, head, and tail functions

In [ ]:
```
The info(), describe(), head(), and tail() functions are commonly used methods
in Pandas to quickly inspect, summarize, and view the contents of a DataFrame.
Here's an explanation of each:

1) info() Function:
Use: The info() function is used to get a concise summary of the DataFrame,
     including the data types of each column, the number of non-null values, and memory usage.
Output: This will display information about the DataFrame, including column names, data types,
```

```
        and non-null counts, which is helpful for understanding the dataset's
        structure and identifying missing values.

2) describe() Function:
Use: The describe() function generates basic statistical summaries of
     the numerical columns in the DataFrame. It provides statistics like
     count, mean, standard deviation, minimum, 25th percentile, median (50th percentile),
     75th percentile, and maximum values.

3) head() Function:
Use: The head() function is used to display the first few rows of the DataFrame.
     By default, it shows the first 5 rows, but you can specify the number
     of rows you want to see by passing an argument.
Output: This displays the specified number of rows (default is 5) from
        the top of the DataFrame, allowing you to quickly inspect the dataset's structure and content.

4) tail() Function:
Use: The tail() function is similar to head(), but it displays the last few
     rows of the DataFrame. Like head(), you can specify the number of rows to display.
Output: This displays the specified number of rows (default is 5) from the
        bottom of the DataFrame, which can be helpful for checking the end of the dataset.
```

In [47]:
```
import pandas as pd
df = pd.read_csv(r'C:\Users\prana\DS - Python\datasets\medical_insurance.csv')
```

In [48]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1338 entries, 0 to 1337
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   age       1338 non-null   int64  
 1   sex       1338 non-null   object 
 2   bmi       1338 non-null   float64
 3   children  1338 non-null   int64  
 4   smoker    1338 non-null   object 
 5   region    1338 non-null   object 
 6   charges   1338 non-null   float64
dtypes: float64(2), int64(2), object(3)
memory usage: 73.3+ KB
```

In [49]:
```
df.describe()
```

Out[49]:

	age	bmi	children	charges
count	1338.000000	1338.000000	1338.000000	1338.000000
mean	39.207025	30.663397	1.094918	13270.422265
std	14.049960	6.098187	1.205493	12110.011237
min	18.000000	15.960000	0.000000	1121.873900
25%	27.000000	26.296250	0.000000	4740.287150
50%	39.000000	30.400000	1.000000	9382.033000
75%	51.000000	34.693750	2.000000	16639.912515
max	64.000000	53.130000	5.000000	63770.428010

In [50]:
```
df.head()
```

Out[50]:

	age	sex	bmi	children	smoker	region	charges
0	19	female	27.900	0	yes	southwest	16884.92400
1	18	male	33.770	1	no	southeast	1725.55230
2	28	male	33.000	3	no	southeast	4449.46200
3	33	male	22.705	0	no	northwest	21984.47061
4	32	male	28.880	0	no	northwest	3866.85520

In [51]:
```
df.tail()
```

Out[51]:

	age	sex	bmi	children	smoker	region	charges
1333	50	male	30.97	3	no	northwest	10600.5483
1334	18	female	31.92	0	no	northeast	2205.9808
1335	18	female	36.85	0	no	southeast	1629.8335
1336	21	female	25.80	0	no	southwest	2007.9450
1337	61	female	29.07	0	yes	northwest	29141.3603

30. Write a Pandas program to check whether only a title case is present in a given column of a DataFrame.

In [58]:
```
import pandas as pd

# Create a DataFrame
df = pd.DataFrame({'Name': ['John Doe', 'Jane Doe', 'peter Smith', 'Mary Johnson']})

# Check if the values in the 'Name' column are all title case
df['Name'].str.istitle()
```

Out[58]:
```
0     True
1     True
2    False
3     True
Name: Name, dtype: bool
```

31. What is the map Function In Pandas?

In [62]:
```
# Mapping Values to a New Column:
# You can create a new column based on the values of an existing column using the map() function.

# Replacing Values:
# You can replace specific values in a column using the map() function.
import pandas as pd

df = pd.DataFrame({'Status': ['Active', 'Inactive', 'Active', 'Inactive']})

# Replace 'Active' with 1 and 'Inactive' with 0
df['Status_Code'] = df['Status'].map({'Active': 1, 'Inactive': 0})
df
```

Out[62]:

	Status	Status_Code
0	Active	1
1	Inactive	0
2	Active	1
3	Inactive	0

In [65]:
```
# Applying a Function:
# You can apply a custom function to transform values in a column.
import pandas as pd

df = pd.DataFrame({'Numbers': [2, 3, 4]})

# Define a function to square each number
def square(x):
    return x ** 2

# Square the values in the 'Numbers' column
df['Squared'] = df['Numbers'].map(square)
df
```

Out[65]:

	Numbers	Squared
0	2	4
1	3	9
2	4	16

32. How will you add a column to a pandas DataFrame?

In [67]:
```
import pandas as pd

# Create a sample DataFrame
df = pd.DataFrame( {'Column1': [1, 2, 3, 4],
                    'Column2': ['A', 'B', 'C', 'D'] } )

# Adding a new column 'NewColumn' with values
df['NewColumn'] = [5, 6, 7, 8]

# Alternatively, you can add a new column with a constant value
df['ConstantColumn'] = 'X'
df
```

Out[67]:

	Column1	Column2	NewColumn	ConstantColumn
0	1	A	5	X
1	2	B	6	X
2	3	C	7	X
3	4	D	8	X

33. How will you add a column at a specific index to a pandas DataFrame?

In [68]:
```
# The insert() method allows you to specify the desired position (index)
# for the new column. Here's how you can do it:

import pandas as pd

# Create a sample DataFrame
df = pd.DataFrame( {'Column1': [1, 2, 3, 4],
                    'Column2': ['A', 'B', 'C', 'D'] } )

# Adding a new column 'NewColumn' with values at index 1
new_column_values = [5, 6, 7, 8]
df.insert(1, 'NewColumn', new_column_values)  # 1 is Index where the new column should be inserted
df
```

Out[68]:

	Column1	NewColumn	Column2
0	1	5	A
1	2	6	B
2	3	7	C
3	4	8	D

34. How to Delete Indices, Rows, or Columns From a Pandas DataFrame?

In [81]:
```
# Deleting Rows:

import pandas as pd
df = pd.DataFrame({'A': [1, 2, 3, 4, 5], 'B': [4, 5, 6, 7, 8]})

# Delete row with index 1
df = df.drop(1)

# Delete multiple rows with indices 0 and 2
df = df.drop([0, 2])
df
```

Out[81]:

	A	B
3	4	7
4	5	8

In [82]:
```
import pandas as pd
df = pd.DataFrame({'A': [1, 2, 3, 4, 5], 'B': [4, 5, 6, 7, 8]})

# Delete rows where column 'A' is greater than 2
df = df[df['A'] <= 2]
df
```

Out[82]:

	A	B
0	1	4
1	2	5

In [83]:
```
# Deleting Columns: Using drop() Method with axis=1:

import pandas as pd
df = pd.DataFrame({'A': [1, 2, 3], 'B': [4, 5, 6],  'C': [7, 8, 9], 'D': [10, 11, 12]})

# Delete column 'B'
df = df.drop('B', axis=1)

# Delete multiple columns 'A' and 'B'
df = df.drop(['A', 'C'], axis=1)
df
```

Out[83]:

	D
0	10
1	11
2	12

In [84]:
```
# Using del Statement:
import pandas as pd
df = pd.DataFrame({'A': [1, 2, 3], 'B': [4, 5, 6]})

# Delete column 'B' in-place
del df['B']
df
```

Out[84]:

	A
0	1
1	2
2	3

35. How to get the items which are not common to both series A and series B?

In [86]:
```
import pandas as pd

# Create two series
sr1 = pd.Series([1, 2, 3, 4, 5])
sr2 = pd.Series([3, 4, 5, 6, 7])

# Find the union of the two series
union = pd.Series(np.union1d(sr1, sr2))

# Find the intersection of the two series
intersection = pd.Series(np.intersect1d(sr1, sr2))

# Find the difference between the union and the intersection
difference = union[~union.isin(intersection)]
difference
```

Out[86]:
```
0    1
1    2
5    6
6    7
dtype: int64
```

36. How to get the minimum, 25th percentile, median, 75th, and max of a numeric series?

In [8]:
```
# Using Pandas describe()

import pandas as pd

# Create a numeric Series (replace this with your actual data)
data = pd.Series([10, 15, 20, 25, 30, 35, 40, 45, 50])

# Use the describe() method to get summary statistics
summary_stats = data.describe(percentiles=[.25, .5, .75])

# Extract the specific statistics
minimum = summary_stats['min']
q1 = summary_stats['25%']
median = summary_stats['50%']
q3 = summary_stats['75%']
maximum = summary_stats['max']

print("Minimum:", minimum)
print("25th Percentile (Q1):", q1)
print("Median (50th Percentile or Q2):", median)
print("75th Percentile (Q3):", q3)
print("Maximum:", maximum)
```

```
Minimum: 10.0
25th Percentile (Q1): 20.0
Median (50th Percentile or Q2): 30.0
75th Percentile (Q3): 40.0
Maximum: 50.0
```

In [3]:
```
# Using Numpy functions, percentile(), median(), min(), max()

import pandas as pd
import numpy as np

# Create a numeric Series (replace this with your actual data)
data = pd.Series([10, 15, 20, 25, 30, 35, 40, 45, 50])

# Calculate the minimum
minimum = np.min(data)

# Calculate the 25th percentile (Q1)
q1 = np.percentile(data, 25)

# Calculate the median (50th percentile or Q2)
median = np.median(data)

# Calculate the 75th percentile (Q3)
q3 = np.percentile(data, 75)

# Calculate the maximum
maximum = np.max(data)

print("Minimum:", minimum)
print("25th Percentile (Q1):", q1)
print("Median (50th Percentile or Q2):", median)
```

```
print("75th Percentile (Q3):", q3)
print("Maximum:", maximum)
```

```
Minimum: 10
25th Percentile (Q1): 20.0
Median (50th Percentile or Q2): 30.0
75th Percentile (Q3): 40.0
Maximum: 50
```

37. How can we sort the DataFrame?

In [3]:
```
import pandas as pd

# Create a sample DataFrame
df = pd.DataFrame({'A': [3, 1, 2, 4],
                   'B': [10, 8, 9, 7]})

# Sort by a single column
df_sorted = df.sort_values(by='A')  # Sort by column 'A' in ascending order
# To sort in descending order, you can use df.sort_values(by='A', ascending=False)
df_sorted
```

Out[3]:

	A	B
1	1	8
2	2	9
0	3	10
3	4	7

38. How to drop duplicate rows from DataFrame?

In [24]:
```
import pandas as pd

# Create a sample DataFrame with duplicate rows
df = pd.DataFrame({'A': [1, 2, 2, 3, 4],
                   'B': ['foo', 'bar', 'bar', 'baz', 'far']})

# Drop duplicate rows based on all columns
df_no_duplicates = df.drop_duplicates()

# Display the resulting DataFrames
print("DataFrame with all columns:")
df_no_duplicates
```

```
DataFrame with all columns:
```

Out[24]:

	A	B
0	1	foo
1	2	bar
3	3	baz
4	4	far

In [25]:
```
import pandas as pd

# Create a sample DataFrame with duplicate rows
df = pd.DataFrame({'A': [1, 2, 2, 3, 4],
                   'B': ['foo', 'bar', 'baz', 'baz', 'far']})

# Drop duplicate rows based on a subset of columns
df_no_duplicates_subset = df.drop_duplicates(subset=['B'])

# Display the resulting DataFrames
print("\nDataFrame with duplicates based on subset of columns:")
df_no_duplicates_subset
```

```
DataFrame with duplicates based on subset of columns:
```

Out[25]:

	A	B
0	1	foo
1	2	bar
2	2	baz
4	4	far

39. How to drop duplicate columns from DataFrame?

In [29]:
```
import pandas as pd

# Create a sample DataFrame with duplicate columns
df = pd.DataFrame( {'A': [1, 2, 3],
                    'B': [4, 5, 6],
```

```
                    'C': [7, 8, 9],
                    'D': [1, 2, 3],
                    'E': [4, 5, 6]} )

# Identify and drop duplicate columns based on column values
df_no_duplicate_columns = df.T.drop_duplicates().T

# Display the resulting DataFrame
df_no_duplicate_columns
```

Out[29]:

	A	B	C
0	1	4	7
1	2	5	8
2	3	6	9

40. Write a Pandas program to split a string of a column of a given DataFrame into multiple columns(Split Name and Surname)

In [30]:

```
import pandas as pd

# Create a sample DataFrame with a 'Full Name' column
data = {'Full Name': ['John Doe', 'Jane Smith', 'Alice Johnson']}
df = pd.DataFrame(data)

# Split the 'Full Name' column into 'First Name' and 'Last Name' columns
df[['First Name', 'Last Name']] = df['Full Name'].str.split(' ', n=1, expand=True)

# Display the resulting DataFrame
df
```

Out[30]:

	Full Name	First Name	Last Name
0	John Doe	John	Doe
1	Jane Smith	Jane	Smith
2	Alice Johnson	Alice	Johnson