# Q1. List any five functions of the pandas library with execution.

1. read\_csv: This function is used to read a CSV file and create a pandas DataFrame.

import pandas as pd

# Reading a CSV file

df = pd.read\_csv('data.csv')

print(df)

1. head: This function is used to display the first few rows of a DataFrame.

import pandas as pd

# Creating a DataFrame

data = {'Name': ['Tanishq', 'Riya', 'Vasuki', 'Krishna', 'Ram'],

'Age': [25, 30, 35, 28, 32],

'City': ['Pune', 'Kashmir', 'Assam', 'Delhi', 'chennai']}

df = pd.DataFrame(data)

# Displaying the first 3 rows

print(df.head(3))

1. info: This function provides a concise summary of a DataFrame, including the column names, data types, and non-null values.

import pandas as pd

# Creating a DataFrame

data = {'Name': ['John', 'Jane', 'Mike'],

'Age': [25, 30, None],

'City': ['New York', 'London', 'Paris']}

df = pd.DataFrame(data)

# Displaying the summary information

print(df.info())

1. describe: This function generates descriptive statistics of a DataFrame, such as count, mean, standard deviation, minimum, maximum, and quartile values.

import pandas as pd

# Creating a DataFrame

data = {'Name': ['John', 'Jane', 'Mike', 'Lisa', 'Mark'],

'Age': [25, 30, 35, 28, 32],

'Salary': [50000, 60000, 70000, 55000, 65000]}

df = pd.DataFrame(data)

# Generating descriptive statistics

print(df.describe())

1. groupby: This function is used to group rows of a DataFrame based on one or more columns, enabling further aggregation or analysis.

import pandas as pd

# Creating a DataFrame

data = {'Name': ['Shudh', 'Swaraj', 'Kunal', 'Lisa', 'Mark'],

'Age': [25, 30, 35, 28, 32],

'City': ['New York', 'London', 'Paris', 'Sydney', 'Tokyo'],

'Salary': [50000, 60000, 70000, 55000, 65000]}

df = pd.DataFrame(data)

# Grouping by 'City' column and calculating average salary

grouped\_df = df.groupby('City')['Salary'].mean()

print(grouped\_df)

# Q2. Given a Pandas DataFrame df with columns 'A', 'B', and 'C', write a Python function to re-index the DataFrame with a new index that starts from 1 and increments by 2 for each row.

Python function that re-indexes a Pandas DataFrame with a new index starting from 1 and incrementing by 2 for each row:

import pandas as pd

def reindex\_dataframe(df):

new\_index = pd.Index(range(1, len(df)\*2, 2))

df = df.set\_index(new\_index)

return df

The function with an example DataFrame:

import pandas as pd

# Example DataFrame

df = pd.DataFrame({'A': [10, 20, 30, 40], 'B': [50, 60, 70, 80], 'C': [90, 100, 110, 120]})

# Reindex the DataFrame

df\_reindexed = reindex\_dataframe(df)

# Print the reindexed DataFrame

print(df\_reindexed)

The output will be a new DataFrame with the same columns ('A', 'B', 'C'), but with a new index starting from 1 and incrementing by 2 for each row:

A B C

1 10 50 90

3 20 60 100

5 30 70 110

7 40 80 120

The original index of the DataFrame is replaced with the new index starting from 1.

# Q3. You have a Pandas DataFrame df with a column named 'Values'. Write a Python function that iterates over the DataFrame and calculates the sum of the first three values in the 'Values' column. The function should print the sum to the console. For example, if the 'Values' column of df contains the values [10, 20, 30, 40, 50], your function should calculate and print the sum of the first three values, which is 60.

Python function that iterates over a Pandas DataFrame and calculates the sum of the first three values in the 'Values' column:

import pandas as pd

def calculate\_sum\_of\_first\_three(df):

sum\_of\_first\_three = df['Values'].head(3).sum()

print("Sum of the first three values:", sum\_of\_first\_three)

# Example usage

df = pd.DataFrame({'Values': [10, 20, 30, 40, 50]})

calculate\_sum\_of\_first\_three(df)

sql

Sum of the first three values: 60

# Q4. Given a Pandas DataFrame df with a column 'Text', write a Python function to create a new column Word\_Count' that contains the number of words in each row of the 'Text' column.

The apply function in Pandas along with the split method to count the number of words in each row of the 'Text' column. Here's an example Python function that achieves this:

import pandas as pd

def add\_word\_count(df):

df['Word\_Count'] = df['Text'].apply(lambda x: len(str(x).split()))

return df

The apply method on the 'Text' column to apply a lambda function to each row. The lambda function splits the text into a list of words using the split() method and calculates the length of the list using len(). The resulting word count is assigned to the 'Word\_Count' column.

This function by passing your DataFrame df as an argument:

df = pd.DataFrame({'Text': ['Hello, how are you?', 'I am doing great!', 'Python is awesome']})

df\_with\_word\_count = add\_word\_count(df)

print(df\_with\_word\_count)

Output:

Text Word\_Count

0 Hello, how are you? 4

1 I am doing great! 4

2 Python is awesome 3

The function adds a new column 'Word\_Count' to the original DataFrame with the respective word counts for each row in the 'Text' column.

# Q5. How are DataFrame.size() and DataFrame.shape() different?

In pandas, the DataFrame.size and DataFrame.shape methods provide different information about the structure and size of a DataFrame.

1. DataFrame.size: This method returns the total number of elements in the DataFrame, which is calculated as the product of the number of rows and columns. It gives you the total count of all elements in the DataFrame, including missing or NaN values. The size attribute returns an integer value.

Example:

import pandas as pd

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

print(df.size) # Output: 6

1. DataFrame.shape: This method returns a tuple that represents the dimensions of the DataFrame. It returns the number of rows and columns in the DataFrame as (rows, columns). The shape attribute provides a convenient way to quickly check the size of the DataFrame.

Example:

import pandas as pd

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

print(df.shape) # Output: (3, 2)

DataFrame.size gives the total count of all elements in the DataFrame, while DataFrame.shape provides the dimensions of the DataFrame in terms of rows and columns.

# Q6. Which function of pandas do we use to read an excel file?

In pandas, we can use the read\_excel() function to read an Excel file. This function allows use to read data from an Excel file into a pandas DataFrame.

Here's an example of how to use it:

import pandas as pd

# Read an Excel file

df = pd.read\_excel('filename.xlsx')

# Display the DataFrame

print(df)

In the example above, 'filename.xlsx' should be replaced with the actual path and name of the Excel file we want to read. The read\_excel() function will automatically detect the data and headers in the Excel file and load them into the DataFrame. By default, it reads the first sheet of the Excel file, but we can also specify a specific sheet by passing the sheet name or index as an argument to the function.

# Q7. You have a Pandas DataFrame df that contains a column named 'Email' that contains email

# addresses in the format 'username@domain.com'. Write a Python function that creates a new column

# 'Username' in df that contains only the username part of each email address.

# The username is the part of the email address that appears before the '@' symbol. For example, if the

# email address is 'john.doe@example.com', the 'Username' column should contain 'john.doe'. Your

# function should extract the username from each email address and store it in the new 'Username'

# column.

The apply() function in Pandas along with the split() function to extract the username from each email address and create a new column. Here's a Python function that accomplishes this:

import pandas as pd

def extract\_username(df):

df['Username'] = df['Email'].apply(lambda x: x.split('@')[0])

return df

In this function, we use the apply() function to apply a lambda function to each value in the 'Email' column. The lambda function splits each email address using the '@' symbol as the separator and returns the first part, which is the username. This username is then stored in the new 'Username' column of the DataFrame.

We can call this function with your DataFrame as an argument to create the new 'Username' column:

df = extract\_username(df)

After calling this function, the DataFrame df will have a new column named 'Username' containing the extracted usernames from the 'Email' column.

# Q8. You have a Pandas DataFrame df with columns 'A', 'B', and 'C'. Write a Python function that selects all rows where the value in column 'A' is greater than 5 and the value in column 'B' is less than 10. The function should return a new DataFrame that contains only the selected rows. For example, if df contains the following values:

# A B C

# 0 3 5 1

# 1 8 2 7

# 2 6 9 4

# 3 2 3 5

# 4 9 1 2

# Data Science Masters Your function should select the following rows: A B C

# 1 8 2 7

# 4 9 1 2

# The function should return a new DataFrame that contains only the selected rows.

Python function to select the rows based on the conditions and return a new DataFrame:

import pandas as pd

def select\_rows(df):

selected\_rows = df[(df['A'] > 5) & (df['B'] < 10)]

return selected\_rows

Here's how we can use this function with the given example DataFrame:

# Create the example DataFrame

data = {'A': [3, 8, 6, 2, 9],

'B': [5, 2, 9, 3, 1],

'C': [1, 7, 4, 5, 2]}

df = pd.DataFrame(data)

# Call the function to select rows

selected\_df = select\_rows(df)

# Print the selected DataFrame

print(selected\_df)

Output:

A B C

1 8 2 7

4 9 1 2

The select\_rows function uses boolean indexing to select the rows that satisfy the given conditions: column 'A' greater than 5 and column 'B' less than 10. It returns a new DataFrame containing only the selected rows.

# Q9. Given a Pandas DataFrame df with a column 'Values', write a Python function to calculate the mean, median, and standard deviation of the values in the 'Values' column.

A Python function that calculates the mean, median, and standard deviation of a column named 'Values' in a Pandas DataFrame:

import pandas as pd

def calculate\_statistics(df):

# Calculate mean

mean = df['Values'].mean()

# Calculate median

median = df['Values'].median()

# Calculate standard deviation

std\_dev = df['Values'].std()

return mean, median, std\_dev

To use this function, we can pass your DataFrame df as an argument, and it will return a tuple containing the mean, median, and standard deviation values.

Here's an example usage:

import pandas as pd

# Create a sample DataFrame

data = {'Values': [10, 20, 30, 40, 50]}

df = pd.DataFrame(data)

# Calculate statistics

mean, median, std\_dev = calculate\_statistics(df)

# Print the results

print("Mean:", mean)

print("Median:", median)

print("Standard Deviation:", std\_dev)

Output:

Mean: 30.0

Median: 30.0

Standard Deviation: 15.811388300841896

In this example, the 'Values' column contains the numbers [10, 20, 30, 40, 50]. The mean is 30.0, the median is 30.0, and the standard deviation is approximately 15.81.

# Q10. Given a Pandas DataFrame df with a column 'Sales' and a column 'Date', write a Python function to create a new column 'MovingAverage' that contains the moving average of the sales for the past 7 days for each row in the DataFrame. The moving average should be calculated using a window of size 7 and should include the current day.

The rolling function in Pandas to calculate the moving average over a specified window size. Here's an example Python function that takes a DataFrame df with columns 'Sales' and 'Date' and adds a new column 'MovingAverage' with the moving average of sales for the past 7 days:

import pandas as pd

def calculate\_moving\_average(df):

df['MovingAverage'] = df['Sales'].rolling(window=7, min\_periods=1).mean()

return df

# Example usage:

# df = ... # Your DataFrame with 'Sales' and 'Date' columns

# df = calculate\_moving\_average(df)

In this function, df['Sales'].rolling(window=7, min\_periods=1) creates a rolling window object with a window size of 7 and a minimum of 1 period, ensuring that even if there are missing values at the beginning of the DataFrame, it will still calculate the moving average. Then, .mean() calculates the mean of each window and assigns the values to the 'MovingAverage' column in the DataFrame.

# Q11. You have a Pandas DataFrame df with a column 'Date'. Write a Python function that creates a new

# column 'Weekday' in the DataFrame. The 'Weekday' column should contain the weekday name (e.g.

# Monday, Tuesday) corresponding to each date in the 'Date' column.

# For example, if df contains the following values:

# Date

# 0 2023-01-01

# 1 2023-01-02

# 2 2023-01-03

# 3 2023-01-04

# 4 2023-01-05

# Your function should create the following DataFrame:

# Date Weekday

# 0 2023-01-01 Sunday

# 1 2023-01-02 Monday

# 2 2023-01-03 Tuesday

# 3 2023-01-04 Wednesday

# 4 2023-01-05 Thursday

# The function should return the modified DataFrame.

To create a new column 'Weekday' in a Pandas DataFrame based on the 'Date' column, we can use the dt.weekday\_name attribute of the datetime objects. Here's a Python function that achieves this:

import pandas as pd

def add\_weekday\_column(df):

df['Date'] = pd.to\_datetime(df['Date']) # Convert 'Date' column to datetime if it's not already

df['Weekday'] = df['Date'].dt.weekday\_name # Create 'Weekday' column based on 'Date' column

return df

This function by passing your DataFrame df as an argument, and it will modify the DataFrame by adding the 'Weekday' column. Here's an example usage:

import pandas as pd

# Example DataFrame

df = pd.DataFrame({'Date': ['2023-01-01', '2023-01-02', '2023-01-03', '2023-01-04', '2023-01-05']})

# Call the function to add the 'Weekday' column

df = add\_weekday\_column(df)

# Print the modified DataFrame

print(df)

Output:

Date Weekday

0 2023-01-01 Sunday

1 2023-01-02 Monday

2 2023-01-03 Tuesday

3 2023-01-04 Wednesday

4 2023-01-05 Thursday

# Q12. Given a Pandas DataFrame df with a column 'Date' that contains timestamps, write a Python function to select all rows where the date is between '2023-01-01' and '2023-01-31'.

import pandas as pd

def select\_rows\_between\_dates(df):

df['Date'] = pd.to\_datetime(df['Date']) # Convert 'Date' column to datetime if not already

mask = (df['Date'] >= '2023-01-01') & (df['Date'] <= '2023-01-31') # Create a boolean mask

selected\_rows = df[mask] # Apply the mask to select rows

return selected\_rows

# Q13. To use the basic functions of pandas, what is the first and foremost necessary library that needs to be imported?

To use the basic functions of pandas, the first and foremost library that needs to be imported is the pandas library itself. The pandas library is a powerful data manipulation and analysis tool for Python.

To import the pandas library, you can use the following import statement at the beginning of your Python script or notebook:

import pandas as pd