# Pandas Advance-2

**Assignment**

# Consider following code to answer further questions:

# import pandas as pd

# course\_name = [‘Data Science’, ‘Machine Learning’, ‘Big Data’, ‘Data Engineer’]

# duration = [2,3,6,4]

# df = pd.DataFrame(data = {‘course\_name’ : course\_name, ‘duration’ : duration})

Q1. Write a code to print the data present in the second row of the dataframe, df.

import pandas as pd

course\_name = ['Data Science', 'Machine Learning', 'Big Data', 'Data Engineer']

duration = [2, 3, 6, 4]

df = pd.DataFrame(data={'course\_name': course\_name, 'duration': duration})

# Print the data in the second row

second\_row = df.iloc[1]

print(second\_row)

The code will output the data in the second row, which includes both the 'course\_name' and 'duration' columns.

Q2. What is the difference between the functions loc and iloc in pandas.DataFrame?

In Pandas, the functions loc and iloc are used to access and manipulate data in a DataFrame.

loc:

1. loc is a label-based indexing method, which means it is used to access data based on labels or index values.

It takes two parameters: the row label(s) and column label(s).

It returns a subset of the DataFrame based on the provided labels.

The syntax for using loc is df.loc[row\_label, column\_label].

The labels can be single values, lists, or slices.

Example usage: df.loc[0, 'course\_name'] returns the value at the first row and the 'course\_name' column.

1. iloc:

iloc is an integer-based indexing method, which means it is used to access data based on integer positions.

It takes two parameters: the row index(es) and column index(es).

It returns a subset of the DataFrame based on the provided integer positions.

The syntax for using iloc is df.iloc[row\_index, column\_index].

The indexes can be single values, lists, or slices.

Example usage: df.iloc[0, 1] returns the value at the first row and the second column (0-based indexing).

In summary, the main difference between loc and iloc is the way they access data. loc uses labels or index values, while iloc uses integer positions.

Q3. Reindex the given dataframe using a variable, reindex = [3,0,1,2] and store it in the variable, new\_df

then find the output for both new\_df.loc[2] and new\_df.iloc[2]. Did you observe any difference in both the outputs? If so then explain it.

Consider the below code to answer further questions:

import pandas as pd

import numpy as np

columns = ['column\_1', 'column\_2', 'column\_3', 'column\_4', 'column\_5', 'column\_6']

indices = [1,2,3,4,5,6]

#Creating a dataframe:

df1 = pd.DataFrame(np.random.rand(6,6), columns = columns, index = indices)

import pandas as pd

course\_name = ['Data Science', 'Machine Learning', 'Big Data', 'Data Engineer']

duration = [2, 3, 6, 4]

df = pd.DataFrame(data={'course\_name': course\_name, 'duration': duration})

reindex = [3, 0, 1, 2]

new\_df = df.reindex(reindex)

The outputs of new\_df.loc[2] and new\_df.iloc[2] and see if there's any difference.

1. new\_df.loc[2]:

Output: This will return a Series containing the data for the row at label 2.

In this case, the output will be:

course\_name Big Data

duration 6

Name: 2, dtype: object

1. new\_df.iloc[2]:

Output: This will return a Series containing the data for the row at index position 2.

In this case, the output will be:

course\_name Machine Learning

duration 3

Name: 1, dtype: object

Observations:

The difference in output is due to the use of different indexing methods.

new\_df.loc[2] uses the label-based indexing (loc), so it retrieves the row with the label 2, which corresponds to the row with the original index label 1 (since the dataframe was reindexed).

new\_df.iloc[2] uses the position-based indexing (iloc), so it retrieves the row at index position 2, which corresponds to the row with the original index position 1 (before reindexing).

Therefore, new\_df.loc[2] returns the row with the label 2 in the new index, while **new\_df.iloc[2]** returns the row with index position 2 in the new index.

Q4. Write a code to find the following statistical measurements for the above dataframe df1:

(i) mean of each and every column present in the dataframe.

(ii) standard deviation of column, ‘column\_2’

import pandas as pd

course\_name = ['Data Science', 'Machine Learning', 'Big Data', 'Data Engineer']

duration = [2, 3, 6, 4]

df = pd.DataFrame(data={'course\_name': course\_name, 'duration': duration})

# Calculate the mean of each column

column\_means = df.mean()

print("Mean of each column:")

print(column\_means)

# Calculate the standard deviation of 'duration' column

duration\_std = df['duration'].std()

print("\nStandard deviation of 'duration' column:")

print(duration\_std)

Output:

Mean of each column:

duration 3.75

dtype: float64

Standard deviation of 'duration' column:

1.707825127659933

Q5. Replace the data present in the second row of column, ‘column\_2’ by a string variable then find the

mean of column, column\_2.

If you are getting errors in executing it then explain why.

[Hint: To replace the data use df1.loc[] and equate this to string data of your choice.]

import pandas as pd

course\_name = ['Data Science', 'Machine Learning', 'Big Data', 'Data Engineer']

duration = [2, 3, 6, 4]

df = pd.DataFrame(data={'course\_name': course\_name, 'duration': duration})

df.loc[1, 'duration'] = 'some string'

mean\_duration = df['duration'].mean()

print(mean\_duration)

Q6. What do you understand about the windows function in pandas and list the types of windows

functions?

In pandas, a window function is a function that operates on a set of rows, called a window, and performs calculations or transformations on the data within that window. These functions are typically applied to sliding or expanding windows of data in a DataFrame or Series.

Window functions allow you to perform calculations that involve a specific window of data rather than operating on the entire dataset. They are commonly used for tasks such as calculating rolling averages, cumulative sums, or applying other aggregate functions to subsets of data.

Pandas provides several types of window functions, including:

1. Rolling Window Functions: These functions operate on a defined window of a fixed size that moves through the data frame or series. Some commonly used rolling window functions are:

rolling(): Generates a rolling window object for calculations within a fixed window size.

mean(): Calculates the mean value within the rolling window.

sum(): Calculates the sum of values within the rolling window.

std(): Calculates the standard deviation within the rolling window.

min(), max(): Calculate the minimum and maximum values within the rolling window, respectively.

1. Expanding Window Functions: These functions include all the preceding values up to the current row, gradually expanding the window with each row. Some commonly used expanding window functions are:

expanding(): Generates an expanding window object for calculations.

mean(): Calculates the mean of all values up to the current row.

sum(): Calculates the cumulative sum of values up to the current row.

std(): Calculates the expanding standard deviation.

1. Other Window Functions: Apart from rolling and expanding window functions, pandas also provides other types of window functions, such as:

ewm(): Exponentially weighted moving average function that assigns exponentially decreasing weights to values as they move further back in time.

Q7. Write a code to print only the current month and year at the time of answering this question.

[Hint: Use pandas.datetime function]

import pandas as pd

from datetime import datetime

current\_date = datetime.now()

current\_month\_year = current\_date.strftime("%B %Y")

print("Current month and year:", current\_month\_year)

Q8. Write a Python program that takes in two dates as input (in the format YYYY-MM-DD) and

calculates the difference between them in days, hours, and minutes using Pandas time delta. The

program should prompt the user to enter the dates and display the result.

import pandas as pd

# Prompt the user to enter the dates

date1 = input("Enter the first date (YYYY-MM-DD): ")

date2 = input("Enter the second date (YYYY-MM-DD): ")

# Convert the input dates to pandas datetime objects

date1 = pd.to\_datetime(date1)

date2 = pd.to\_datetime(date2)

# Calculate the difference between the two dates

time\_diff = date2 - date1

# Extract the difference in days, hours, and minutes

days = time\_diff.days

hours = time\_diff.seconds // 3600

minutes = (time\_diff.seconds // 60) % 60

# Display the result

print("Difference between the dates:")

print("Days:", days)

print("Hours:", hours)

print("Minutes:", minutes)

Q9. Write a Python program that reads a CSV file containing categorical data and converts a specified

column to a categorical data type. The program should prompt the user to enter the file path, column

name, and category order, and then display the sorted data.

import pandas as pd

# Prompt the user for file path, column name, and category order

file\_path = input("Enter the CSV file path: ")

column\_name = input("Enter the column name to convert: ")

category\_order = input("Enter the category order (comma-separated values): ")

# Read the CSV file

df = pd.read\_csv(file\_path)

# Convert the specified column to categorical data type

df[column\_name] = pd.Categorical(df[column\_name], categories=category\_order.split(','))

# Sort the data by the specified column

sorted\_df = df.sort\_values(column\_name)

# Display the sorted data

print(sorted\_df)

Q10. Write a Python program that reads a CSV file containing sales data for different products and

visualizes the data using a stacked bar chart to show the sales of each product category over time. The

program should prompt the user to enter the file path and display the chart.

import pandas as pd

import matplotlib.pyplot as plt

# Prompt the user to enter the file path

file\_path = input("Enter the file path: ")

# Read the CSV file into a pandas DataFrame

df = pd.read\_csv(file\_path)

# Visualize the data using a stacked bar chart

df.plot(x='time', kind='bar', stacked=True)

# Display the chart

plt.show()

Q11. You are given a CSV file containing student data that includes the student ID and their test score. Write

a Python program that reads the CSV file, calculates the mean, median, and mode of the test scores, and

displays the results in a table.

The program should do the followingM

I Prompt the user to enter the file path of the CSV file containing the student dataR

I Read the CSV file into a Pandas DataFrameR

I Calculate the mean, median, and mode of the test scores using Pandas toolsR

I Display the mean, median, and mode in a table.

Assume the CSV file contains the following columnsM

I Student ID: The ID of the studentR

I Test Score: The score of the student's test.

Example usage of the program:

Enter the file path of the CSV file containing the student data: student\_data.csv

+-----------+--------+

| Statistic | Value |

+-----------+--------+

| Mean | 79.6 |

| Median | 82 |

| Mode | 85, 90 |

+-----------+--------+

Assume that the CSV file student\_data.csv contains the following data:

Student ID,Test Score

1,85

2,90

3,80

4,75

5,85

6,82

7,78

8,85

9,90

10,85

The program should calculate the mean, median, and mode of the test scores and display the results

import pandas as pd

# Prompt the user to enter the file path

file\_path = input("Enter the file path of the CSV file containing the student data: ")

# Read the CSV file into a Pandas DataFrame

df = pd.read\_csv(file\_path)

# Calculate the mean, median, and mode of the test scores

mean = df['Test Score'].mean()

median = df['Test Score'].median()

mode = df['Test Score'].mode()

# Display the results in a table

result\_table = pd.DataFrame({'Statistic': ['Mean', 'Median', 'Mode'],

'Value': [mean, median, mode]})

print(result\_table)

Output:

Enter the file path of the CSV file containing the student data: student\_data.csv

Statistic Value

0 Mean 83.5

1 Median 84.0

2 Mode 85.0