PYARI SRIVASTAVA.P

Analytical Day - 1

1. Write a program that defines a function euclidean\_distance that takes two Pandas Series as input and computes their Euclidean distance. It first converts the Series to NumPy arrays for faster computation and then applies the Euclidean distance formula

CODE:-

import pandas as pd

import numpy as np

def euclidean\_distance(series1, series2):

# Convert the Pandas Series to NumPy arrays for faster computation

array1 = series1.to\_numpy()

array2 = series2.to\_numpy()

# Compute the Euclidean distance using the formula

distance = np.sqrt(np.sum((array1 - array2) \*\* 2))

return distance

# Example usage

series1 = pd.Series([1, 2, 3])

series2 = pd.Series([4, 5, 6])

distance = euclidean\_distance(series1, series2)

print(f"The Euclidean distance is: {distance}")

OUTPUT:

The Euclidean distance is: 5.196152422706632

1. Write a program that defines a function find\_positions that takes a Pandas Series as input and returns the positions of the values neighbored by smaller values on both sides. It iterates through the series and checks if the current value is greater than both its previous and next values. If so, it adds the position of that value to the result list.

CODE:

import pandas as pd

def find\_positions(series):

# Initialize an empty list to store the positions of the values

positions = []

# Iterate through the series starting from the second element to the second last element

for i in range(1, len(series) - 1):

# Check if the current value is greater than both its previous and next values

if series[i] > series[i - 1] and series[i] > series[i + 1]:

positions.append(i)

return positions

# Example usage

series = pd.Series([1, 3, 2, 5, 4, 6, 7, 6])

positions = find\_positions(series)

print(f"Positions of values neighbored by smaller values: {positions}")

OUTPUT:

Positions of values neighbored by smaller values: [1, 3, 6]

1. Write a program that defines a function replace\_missing\_spaces that takes a string as input and replaces any missing white spaces with the least frequent character found in the string. It first counts the frequency of each character using a Pandas Series, then identifies the least frequent character. Finally, it replaces missing white spaces in the input string with the least frequent character and returns the modified string.

CODE:

import pandas as pd

def replace\_missing\_spaces(input\_string):

# Remove any existing spaces from the string for accurate frequency counting

string\_without\_spaces = input\_string.replace(" ", "")

# Count the frequency of each character using a Pandas Series

char\_series = pd.Series(list(string\_without\_spaces))

freq = char\_series.value\_counts()

# Identify the least frequent character

least\_frequent\_char = freq.idxmin()

# Replace missing white spaces with the least frequent character

modified\_string = input\_string.replace(" ", least\_frequent\_char)

return modified\_string

# Example usage

input\_string = "This is a test string"

modified\_string = replace\_missing\_spaces(input\_string)

print(f"Modified string: {modified\_string}")

OUTPUT:

Modified string: ThisTisTaTtestTstring

1. Write a program that defines a function compute\_autocorrelations that takes a numeric series and a lag as input and computes the autocorrelation at the specified lag using the autocorr method provided by Pandas.

CODE:

import pandas as pd

def compute\_autocorrelations(series, lag):

# Compute the autocorrelation using the autocorr method in Pandas

autocorrelation = series.autocorr(lag=lag)

return autocorrelation

# Example usage

series = pd.Series([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])

lag = 3

autocorrelation = compute\_autocorrelations(series, lag)

print(f"Autocorrelation at lag {lag}: {autocorrelation}")

OUTPUT:

Autocorrelation at lag 3: 1.0

1. Write a program that defines a function sundays\_of\_year that takes a year as input and returns a Pandas DatetimeIndex containing all the Sundays of that year. It generates a date range for the entire year, then filters out only the Sundays using the dayofweek attribute of the dates.

CODE:

import pandas as pd

def sundays\_of\_year(year):

# Generate a date range for the entire year

start\_date = f"{year}-01-01"

end\_date = f"{year}-12-31"

date\_range = pd.date\_range(start=start\_date, end=end\_date, freq='D')

# Filter out only Sundays using the dayofweek attribute

sundays = date\_range[date\_range.dayofweek == 6]

return sundays

# Example usage

year = 2024

sundays = sundays\_of\_year(year)

print(f"Sundays of {year}:")

print(sundays)

OUTPUT:

Sundays of 2024:

DatetimeIndex(['2024-01-07', '2024-01-14', '2024-01-21', '2024-01-28',

'2024-02-04', '2024-02-11', '2024-02-18', '2024-02-25',

'2024-03-03', '2024-03-10', '2024-03-17', '2024-03-24',

'2024-03-31', '2024-04-07', '2024-04-14', '2024-04-21',

'2024-04-28', '2024-05-05', '2024-05-12', '2024-05-19',

'2024-05-26', '2024-06-02', '2024-06-09', '2024-06-16',

'2024-06-23', '2024-06-30', '2024-07-07', '2024-07-14',

'2024-07-21', '2024-07-28', '2024-08-04', '2024-08-11',

'2024-08-18', '2024-08-25', '2024-09-01', '2024-09-08',

'2024-09-15', '2024-09-22', '2024-09-29', '2024-10-06',

'2024-10-13', '2024-10-20', '2024-10-27', '2024-11-03',

'2024-11-10', '2024-11-17', '2024-11-24', '2024-12-01',

'2024-12-08', '2024-12-15', '2024-12-22', '2024-12-29'],

dtype='datetime64[ns]', freq=None)

1. Write a program that defines to convert the series into a DataFrame using the reset\_index() method, which moves the index to a column and resets the index to default integer index.

CODE:

import pandas as pd

def series\_to\_dataframe(series):

# Use reset\_index to move the index to a column and reset the index

df = series.reset\_index()

return df

# Example usage

series = pd.Series([10, 20, 30, 40, 50], index=['a', 'b', 'c', 'd', 'e'])

df = series\_to\_dataframe(series)

print(df)

OUTPUT:

index 0

0 a 10

1 b 20

2 c 30

3 d 40

4 e 50

1. Write a program that that utilizes the pd.concat() function to stack the given series vertically and horizontally.

CODE:

import pandas as pd

def concat\_series(series1, series2):

# Stack the series vertically (along rows)

vertical\_concat = pd.concat([series1, series2], axis=0)

# Stack the series horizontally (along columns)

horizontal\_concat = pd.concat([series1, series2], axis=1)

return vertical\_concat, horizontal\_concat

# Example usage

series1 = pd.Series([10, 20, 30], name="A")

series2 = pd.Series([40, 50, 60], name="B")

# Perform vertical and horizontal stacking

vertical, horizontal = concat\_series(series1, series2)

print("Vertical Concatenation:")

print(vertical)

print("\nHorizontal Concatenation:")

print(horizontal)

OUTPUT:

Vertical Concatenation:

0 10

1 20

2 30

0 40

1 50

2 60

dtype: int64

Horizontal Concatenation:

A B

0 10 40

1 20 50

2 30 60

1. Suppose x consists of 3 variables – 6, 8, 10 and y consists of corresponding three variables: 12, 10, and 20. Find the Pearson’s coefficient rxy.

CODE:

import numpy as np

# Given data

x = np.array([6, 8, 10])

y = np.array([12, 10, 20])

# Calculate the Pearson correlation coefficient using the formula

def pearson\_correlation(x, y):

# Calculate the necessary summations

n = len(x)

sum\_x = np.sum(x)

sum\_y = np.sum(y)

sum\_xx = np.sum(x\*\*2)

sum\_yy = np.sum(y\*\*2)

sum\_xy = np.sum(x \* y)

# Calculate the Pearson correlation coefficient

numerator = (n \* sum\_xy) - (sum\_x \* sum\_y)

denominator = np.sqrt((n \* sum\_xx - sum\_x\*\*2) \* (n \* sum\_yy - sum\_y\*\*2))

r = numerator / denominator

return r

# Calculate Pearson's correlation coefficient

r\_xy = pearson\_correlation(x, y)

print(f"Pearson's correlation coefficient rxy: {r\_xy:.3f}")

OUTPUT:

Pearson's correlation coefficient rxy: 0.756

1. A dataset of students’ study hours and their corresponding exam scores is given: Study Hours=[2,4,6,8,10] Exam Scores=[50,55,65,75,85] Find the Spearman Rank Correlation between study hours and exam scores to assess whether the two variables are monotonically related.

CODE:

import numpy as np

from scipy.stats import spearmanr

# Given data

study\_hours = np.array([2, 4, 6, 8, 10])

exam\_scores = np.array([50, 55, 65, 75, 85])

# Calculate Spearman Rank Correlation using scipy

rho, p\_value = spearmanr(study\_hours, exam\_scores)

print(f"Spearman's Rank Correlation Coefficient: {rho:.3f}")

print(f"P-value: {p\_value:.3f}")

OUTPUT:

Spearman's Rank Correlation Coefficient: 1.000

P-value: 0.000

10. You are given two datasets containing customer names, but there may be spelling variations and typos between the two lists. Using the FuzzyWuzzy library, how would you compare customer names from both datasets to identify possible matches and calculate the similarity score between the names?

Scenario: You have two lists of customer names:

['John Doe', 'Jane Smith', 'Mary Jane', 'William Brown']

['Jon Doe', 'Jane Smtih', 'Marie Jane', 'William Browne']

You need to identify matching names from both datasets and calculate their similarity scores using the FuzzyWuzzy library.

CODE:

from fuzzywuzzy import fuzz

from fuzzywuzzy import process

# Lists of customer names

list\_1 = ['John Doe', 'Jane Smith', 'Mary Jane', 'William Brown']

list\_2 = ['Jon Doe', 'Jane Smtih', 'Marie Jane', 'William Browne']

# Function to compare names and calculate similarity scores

def compare\_names(list\_1, list\_2):

matches = []

for name1 in list\_1:

# Compare name1 with all names in list\_2

best\_match = process.extractOne(name1, list\_2, scorer=fuzz.ratio)

matches.append((name1, best\_match[0], best\_match[1])) # name1, best match, similarity score

return matches

# Find matches and similarity scores

matches = compare\_names(list\_1, list\_2)

# Print the results

for match in matches:

print(f"Name from List 1: {match[0]}")

print(f"Best match from List 2: {match[1]}")

print(f"Similarity Score: {match[2]}")

print("-" \* 30)

OUTPUT:  
Name from List 1: John Doe

Best match from List 2: Jon Doe

Similarity Score: 93

------------------------------

Name from List 1: Jane Smith

Best match from List 2: Jane Smtih

Similarity Score: 90

------------------------------

Name from List 1: Mary Jane

Best match from List 2: Marie Jane

Similarity Score: 84

------------------------------

Name from List 1: William Brown

Best match from List 2: William Browne

Similarity Score: 96

------------------------------