**CA-2 GENAI**

**NAME- ISHA JAISWAL**

**PRN- 21070521031**

**QUESTION 1-**

Generate a model for Covid 19 with symptoms of parameters like fever, cold, shivering,

weight loss, generate 100 model data with random values for each parameter and order by

parameter lowest to highest in display based on the input parameter.

**CODE-**

import pandas as pd

import numpy as np

# Define the number of samples to generate

num\_samples = 100

# Define the columns for the dataset

columns = ['Fever', 'Cold', 'Shivering', 'Weight Loss']

# Generate random data for each column

data = {

'Fever': np.random.uniform(97, 104, num\_samples), # Fever in Fahrenheit

'Cold': np.random.uniform(0, 1, num\_samples), # Cold as binary 0 or 1 (no/yes)

'Shivering': np.random.uniform(0, 1, num\_samples), # Shivering as binary 0 or 1 (no/yes)

'Weight Loss': np.random.uniform(0, 10, num\_samples) # Weight loss in kg

}

# Create a DataFrame

df = pd.DataFrame(data)

# Sort function to order by a chosen parameter

def sort\_by\_parameter(df, parameter):

return df.sort\_values(by=parameter)

# Sort the dataset by 'Fever' (example sorting parameter)

sorted\_df = sort\_by\_parameter(df, 'Fever')

# Display the sorted data

print("Sorted Data (Top 5 rows):")

print(sorted\_df.head())

print("\nSummary Statistics of the Data:")

print(sorted\_df.describe())

**CODE EXPLAINATION-**

We import pandas for data handling and numpy for generating random numbers.

We set num\_samples to 100, meaning we'll generate 100 records of synthetic data.

We generate random data for four parameters:

Fever: Uniform random values between 97°F and 104°F.

Cold: Binary values (0 or 1) indicating presence of cold symptoms.

Shivering: Binary values (0 or 1) indicating presence of shivering.

Weight Loss: Random values between 0 and 10 kg.

We create a pandas DataFrame to store the generated data in a structured table format.

A function sort\_by\_parameter() is defined to sort the dataset by a chosen column. Here, the parameter passed is 'Fever'.

The DataFrame is sorted by Fever values (from lowest to highest).

head() shows the first 5 rows, and describe() provides a summary of statistical properties of the data.

**OUTPUT-**

Sorted Data (Top 5 rows):

Fever Cold Shivering Weight Loss

8 97.001629 0.218449 0.722177 3.541235

32 97.051532 0.410664 0.296925 1.083407

3 97.064044 0.396493 0.003084 3.597368

77 97.169584 0.909320 0.579737 1.062768

58 97.356408 0.132326 0.365276 7.536396

Summary Statistics of the Data:

Fever Cold Shivering Weight Loss

count 100.000000 100.000000 100.000000 100.000000

mean 100.336568 0.501186 0.493066 5.087025

std 1.952164 0.292190 0.289867 3.014353

min 97.001629 0.026664 0.002857 0.008774

25% 98.841729 0.249049 0.286121 2.397912

50% 100.373264 0.506590 0.483131 5.405215

75% 101.907945 0.753720 0.753236 7.874418

max 103.948364 0.988782 0.980619 9.975576

**OUTPUT EXPLAINATION-**

This shows the first 5 rows of the sorted data, ordered by Fever values, starting from 97.00°F and increasing.

Each row includes values for Cold, Shivering, and Weight Loss.

Fever: The average fever is ~100.34°F, with values ranging from ~97°F to ~104°F.

Cold & Shivering: These binary symptoms are roughly evenly distributed (mean near 0.5).

Weight Loss: Average weight loss is ~5.09 kg, with a minimum of ~0.01 kg and a maximum of ~9.98 kg.

This output provides an overview of simulated COVID-19 symptom data, showing individual variability and key statistics.

**QUESTION 2-**

Generate a model in Python to represent a Housing loan scheme and create a chart to

display the Emi based on rate of interest and reducing balance for a given period. If a customer

wishes to close the loan earlier, print the interest lost distributed over the remaining no. Of

months. Assume suitable data and inputs as necessary.

**CODE-**

import numpy as np

import matplotlib.pyplot as plt

# Function to calculate EMI

def calculate\_emi(principal, annual\_rate, tenure\_years):

monthly\_rate = annual\_rate / (12 \* 100) # Convert annual rate to monthly and percentage to decimal

n = tenure\_years \* 12 # Total number of months

emi = (principal \* monthly\_rate \* (1 + monthly\_rate) \*\* n) / ((1 + monthly\_rate) \*\* n - 1)

return emi

# Function to calculate outstanding balance and interest lost if loan closed early

def early\_closure(principal, annual\_rate, tenure\_years, months\_paid):

monthly\_rate = annual\_rate / (12 \* 100)

total\_months = tenure\_years \* 12

emi = calculate\_emi(principal, annual\_rate, tenure\_years)

# Calculate outstanding balance using reducing balance formula

outstanding\_balance = principal

for month in range(months\_paid):

interest = outstanding\_balance \* monthly\_rate

principal\_repaid = emi - interest

outstanding\_balance -= principal\_repaid

# Calculate total interest saved if loan is closed early

total\_interest\_remaining = 0

for month in range(months\_paid, total\_months):

interest = outstanding\_balance \* monthly\_rate

total\_interest\_remaining += interest

principal\_repaid = emi - interest

outstanding\_balance -= principal\_repaid

return emi, total\_interest\_remaining

# Plot EMI vs. interest rate

def plot\_emi\_vs\_interest(principal, tenure\_years):

interest\_rates = np.linspace(5, 15, 100) # Interest rates from 5% to 15%

emis = [calculate\_emi(principal, rate, tenure\_years) for rate in interest\_rates]

plt.figure(figsize=(10, 6))

plt.plot(interest\_rates, emis, label='EMI')

plt.title('EMI vs Interest Rate')

plt.xlabel('Interest Rate (%)')

plt.ylabel('EMI')

plt.grid(True)

plt.show()

# Example scenario

principal = 500000 # Loan amount

annual\_rate = 8 # Annual interest rate

tenure\_years = 20 # Loan tenure in years

months\_paid = 60 # Loan closed after 60 months (5 years)

# Calculate EMI and interest saved if closed early

emi, interest\_saved = early\_closure(principal, annual\_rate, tenure\_years, months\_paid)

# Display results

emi\_output = f"EMI: {emi:.2f}"

interest\_lost\_output = f"Interest Lost if Closed Early: {interest\_saved:.2f}"

# Plot the EMI vs Interest Rate graph

plot\_emi\_vs\_interest(principal, tenure\_years)

emi\_output, interest\_lost\_output

**CODE EXPLAINATION-**

EMI Calculation:

Formula: EMI is calculated using the loan principal, interest rate, and tenure. The formula accounts for the reducing balance each month as part of the principal is repaid.

For a ₹500,000 loan at an 8% interest rate over 20 years, the calculated EMI is ₹4182.20.

Early Closure:

If the borrower closes the loan after 5 years (60 months), the outstanding balance is calculated. The remaining interest the bank would have earned on the reducing balance is ₹315,168.14, which is considered "interest lost."

Visualization:

A chart is generated to show how EMI changes with different interest rates. As the interest rate increases from 5% to 15%, the EMI increases accordingly, reflecting the impact of higher rates on monthly payments.

**OUTPUT-**

EMI: ₹4182.20

Interest Lost if Closed Early: ₹315,168.14

**OUTPUT EXPLAINATION-**

This shows:

The calculated EMI based on the loan parameters.

The interest lost if the loan is closed early after 5 years.

Here’s the output based on the code:

EMI: ₹4182.20

This is the monthly installment for a ₹500,000 loan at an 8% annual interest rate over a 20-year period.

Interest Lost if Closed Early: ₹315,168.14

If the loan is closed after 5 years (60 months), the total interest that would have been paid over the remaining loan period is ₹315,168.14. This amount represents the interest the bank will lose due to early loan closure.