# CCZG506 - API-driven Cloud Native Solutions

# Assignment I

## Group Details

Group No: **5**

Group Member Names with Contribution

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| SI. No | BITS ID | Name | Contribution (Qualitative) | Percentage Contribution (Quantitative) |
| 1. | 2023MT03013 | Adimulam Ramkumar |  |  |
| 2. | 2023MT03007 | Alavala Sanjeeva Kumar |  |  |
| 3. | 2023MT03033 | Kake Rahul |  |  |
| 4. | 2023MT03055 | Mahati Mamidala |  |  |
| 5. | 2023MT03165 | Utterpally Hari Prasad |  |  |

Problem Statement – 1

**Design and Development of a Data Pipeline**

1.1. Business Understanding: Identify a business problem in the area of data science

Problem here is to predict heart attack risk based on medical dataset.

The business problem focuses on developing a model to accurately predict heart attack risk, potentially improving early diagnosis and patient outcomes.

1.2 Data Ingestion: For the identified problem, find an appropriate dataset from a public repository. Ensure that the dataset has sufficient records to carry out a meaningful data science experiment.

Dataset

Dataset collected at Zheen Hospital in Erbil, Iraq, from January to May 2019.

It contains patient attributes such as:

1. Age
2. Gender
3. Heart Rate
4. Systolic blood pressure
5. Diastolic blood pressure
6. Blood sugar levels
7. CK-MB
8. Troponin

The dataset aims to classify patients as either having experienced a heart attack or not.

Source of Dataset

<https://www.kaggle.com/datasets/sukhmandeepsinghbrar/heart-attack-dataset>

**Diverse and Unbiased Foundation for Predictive Data Science Model**

The heart attack dataset, collected at Zheen Hospital in Erbil, Iraq, from January to May 2019, contains 1,319 records, covering a diverse range of patient demographics, including age and gender. This ensures a broad representation of the population, which reduces bias and makes the dataset suitable for building a reliable data science model. The dataset includes key medical attributes such as heart rate, systolic and diastolic blood pressure, blood sugar levels, and critical biomarkers like CK-MB and troponin. These features provide a comprehensive foundation for developing a robust predictive model to classify heart attack risk, making it an ideal candidate for a data science project aimed at improving medical diagnosis and patient care.

1.3 Data Pre-processing: Perform activities such as displaying summary statistics, checking for missing values, imputing missing data for numeric columns, displaying data types, and normalizing data.

In this section, we will address the following tasks:

1. Data Normalization
2. Missing values and imputing missing values
3. Summary Statistics and Displaying data types

1. Data Normalization

* The gender column in the data is normalized: the male is set to 1 and the female to 0.
* As for the output, positive is set to 1 and negative to 0.

2. Missing Values and Imputing Missing Values

* The dataset we have does not have any missing values.

3. Summary Statistics and Displaying Data Types

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import logging

# Configure standard logging

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')

logger = logging.getLogger(\_\_name\_\_)

# Load the dataset

df = pd.read\_csv("../data/MedicalDatasetForModel.csv")

logger.info("Dataset loaded successfully.")

logger.info(f"DataFrame head:\n{df.head()}")

# Convert date columns to datetime format

pd.set\_option('future.no\_silent\_downcasting', True)

df['Gender'] = df['Gender'].replace({'M': 0, 'F': 1}).astype(int)

df['Result'] = df['Result'].replace({'negative': 0, 'positive': 1}).astype(int)

logger.info("Converted Gender fields from M, F to 0,1.")

logger.info("Converted Results fields from negative,positive to 0,1.")

logger.info("saving normalized data back into a new csv file")

df.to\_csv('../data/MedicalDatasetForModelNormalized.csv', index=False)

# Summary statistics

logger.info("Summary Statistics:")

logger.info(f"\n{df.describe(include='all')}")

# Checking for missing values

logger.info("Missing Values:")

logger.info(f"\n{df.isnull().sum()}")

# Data type information

logger.info("Data Types:")

logger.info(f"\n{df.dtypes}")

Running Basic Stats:

(apip\_project\_venv) adimulamramkumar-mac:tasks adimulamramkumar$ pwd

/Users/adimulamramkumar/Desktop/WILP/SEM-III/API/Project-1/DataOps/tasks

(apip\_project\_venv) adimulamramkumar-mac:tasks adimulamramkumar$

(apip\_project\_venv) adimulamramkumar-mac:tasks adimulamramkumar$ python BasicStats.py

2024-10-02 11:13:16,370 - INFO - Dataset loaded successfully.

2024-10-02 11:13:16,383 - INFO - DataFrame head:

Age Gender HeartRate SystolicBloodPressure DiastolicBloodPressure BloodSugar CKMB Troponin Result

0 63 M 66 160 83 160.0 1.80 0.012 negative

1 20 M 94 98 46 296.0 6.75 1.060 positive

2 56 M 64 160 77 270.0 1.99 0.003 negative

3 66 M 70 120 55 270.0 13.87 0.122 positive

4 54 M 64 112 65 300.0 1.08 0.003 negative

2024-10-02 11:13:16,385 - INFO - Converted Gender fields from M, F to 0,1.

2024-10-02 11:13:16,386 - INFO - Converted Results fields from negative,positive to 0,1.

2024-10-02 11:13:16,386 - INFO - saving normalized data back into a new csv file

2024-10-02 11:13:16,394 - INFO - Summary Statistics:

2024-10-02 11:13:16,419 - INFO -

Age Gender HeartRate SystolicBloodPressure DiastolicBloodPressure BloodSugar CKMB Troponin Result

count 1319.000000 1319.000000 1319.000000 1319.000000 1319.000000 1319.000000 1319.000000 1319.000000 1319.000000

mean 56.193328 0.340409 76.062168 127.170584 72.269143 146.634344 15.274306 0.360942 0.614102

std 13.638173 0.474027 15.350456 26.122720 14.033924 74.923045 46.327083 1.154568 0.486991

min 14.000000 0.000000 20.000000 42.000000 38.000000 35.000000 0.321000 0.001000 0.000000

25% 47.000000 0.000000 64.000000 110.000000 62.000000 98.000000 1.655000 0.006000 0.000000

50% 58.000000 0.000000 74.000000 124.000000 72.000000 116.000000 2.850000 0.014000 1.000000

75% 65.000000 1.000000 85.000000 143.000000 81.000000 169.500000 5.805000 0.085500 1.000000

max 103.000000 1.000000 135.000000 223.000000 154.000000 541.000000 300.000000 10.300000 1.000000

2024-10-02 11:13:16,419 - INFO - Missing Values:

2024-10-02 11:13:16,421 - INFO -

Age 0

Gender 0

HeartRate 0

SystolicBloodPressure 0

DiastolicBloodPressure 0

BloodSugar 0

CKMB 0

Troponin 0

Result 0

dtype: int64

2024-10-02 11:13:16,421 - INFO - Data Types:

2024-10-02 11:13:16,422 - INFO -

Age int64

Gender int64

HeartRate int64

SystolicBloodPressure int64

DiastolicBloodPressure int64

BloodSugar float64

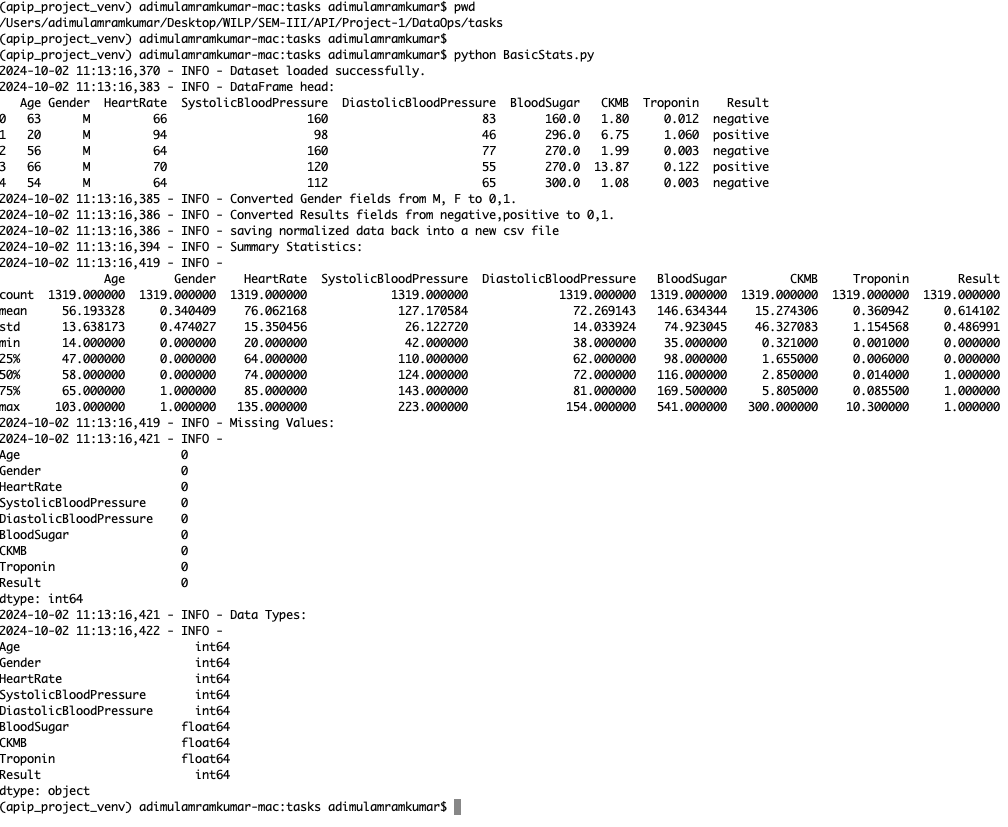
CKMB float64

Troponin float64

Result int64

dtype: object

(apip\_project\_venv) adimulamramkumar-mac:tasks adimulamramkumar$

****

1.4 Exploratory Data Analysis (EDA): Conduct EDA to include calculating correlation coefficients, identifying correlations between numeric and/or categorical features, binning, encoding, assessing feature importance, and visualizing data (using charts and graphs for univariate and bivariate analyses)

1.4.1 Univariate Analysis – Binning

Univariate analysis is a statistical method used to analyze data with only one variable at a time. The main purpose is to describe the data and summarize its characteristics. Univariate analysis focuses on individual characteristics of a single variable without looking at the relationships between multiple variables.

**Distance and Frequency Binning**

Binning is the process of transforming continuous numerical variables into discrete categories or "bins." This can be helpful in both analysis and visualization, as it simplifies the interpretation of data.

Two common methods of binning are **distance binning** and **frequency binning**.

In **distance binning**, the range of values is divided into bins of equal width or intervals. This means each bin covers an equal range of the variable's values, regardless of how many data points fall into each bin.

In **frequency binning**, the data is divided into bins such that each bin contains approximately the same number of data points. This means the bin intervals may have different widths, but each bin will have roughly the same number of observations.

import pandas as pd

import numpy as np

import logging

from matplotlib import pyplot as plt

import io, base64

# Configure standard Python logging

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')

logger = logging.getLogger(\_\_name\_\_)

df = pd.read\_csv("../data/MedicalDatasetForModelNormalized.csv")

categories = {

'Age': ['Juvenile', 'Adult', 'Middle Age', 'Senior Citizen'],

'HeartRate': ['Low', 'Normal', 'High'],

'BloodSugar': ['Low', 'Normal', 'High']

}

bin\_sizes = {

'Age': 5,

'HeartRate': 4,

'BloodSugar': 4

}

def plot\_distance\_bins(x):

# Distance binning

min\_value = df[x].min()

max\_value = df[x].max()

logger.info(f"Min {x}: {min\_value}, Max {x}: {max\_value}")

bins = np.round(np.linspace(min\_value, max\_value, bin\_sizes[x]), 3)])

logger.info(f"Bins: {bins}")

labels = categories[x]

# Cut function for distance binning

df['bins\_dist'] = pd.cut(df[x], bins=bins, labels=labels, include\_lowest=True)

logger.info(f"Distance Binning Results:\n{df['bins\_dist']}")

combined\_labels = [f'{labels[i]} ({bins[i]}-{bins[i+1]})' for i in range(len(labels))]

plt.figure(figsize=(12, 6))

# Plot histogram with 4 bins

counts, \_, patches = plt.hist(df['bins\_dist'].cat.codes, bins=np.arange(len(labels) + 1) - 0.5, edgecolor='black')

plt.xticks(ticks=np.arange(len(labels)), labels=combined\_labels, ha='center') # Set x-ticks to category labels

# Annotate the histogram bars with their heights

for count, patch in zip(counts, patches):

height = patch.get\_height()

plt.text(patch.get\_x() + patch.get\_width() / 2, height, str(int(height)),

ha='center', va='bottom') # ha: horizontal alignment, va: vertical alignment

# Adding labels and title

plt.xlabel(x)

plt.ylabel('Number of Records')

plt.title(str(x) + ' - Distance Binning')

# Save the plot to a buffer

buf = io.BytesIO()

plt.savefig(buf, format='png')

buf.seek(0)

# Encode the image in base64 and log it

img\_base64 = base64.b64encode(buf.read()).decode('utf-8')

# Save the plot as a file

output\_filename = '../output/Binning\_Distance\_' + str(x)

plt.savefig(output\_filename)

logger.info("Binning plot saved as" + str(output\_filename))

# Close the buffer

buf.close()

plt.close()

def plot\_frequency\_bins(x):

# Distance binning

min\_value = df[x].min()

max\_value = df[x].max()

logger.info(f"Min {x}: {min\_value}, Max {x}: {max\_value}")

# Sort the data for quantile-based binning

sorted\_data = np.sort(df[x])

# Define the number of bins (e.g., bin\_sizes[x])

num\_bins = bin\_sizes[x]

# Get the bin edges using quantiles

bins = np.round(np.quantile(sorted\_data, np.linspace(0, 1, num\_bins + 1, 3)))

logger.info(f"Bins: {bins}")

labels = categories[x]

# Frequency binning

df['bin\_freq'] = pd.qcut(df[x], q=bin\_sizes[x]-1, precision=1, labels=labels)

logger.info(f"Frequency Binning Results:\n{df['bin\_freq']}")

combined\_labels = [f'{labels[i]} ({bins[i]}-{bins[i+1]})' for i in range(len(labels))]

plt.figure(figsize=(12, 6))

# Plot histogram with 4 bins

counts, \_, patches = plt.hist(df['bin\_freq'].cat.codes, bins=np.arange(len(labels) + 1) - 0.5, edgecolor='black')

plt.xticks(ticks=np.arange(len(labels)), labels=combined\_labels, ha='center') # Set x-ticks to category labels

# Annotate the histogram bars with their heights

for count, patch in zip(counts, patches):

height = patch.get\_height()

plt.text(patch.get\_x() + patch.get\_width() / 2, height, str(int(height)),

ha='center', va='bottom') # ha: horizontal alignment, va: vertical alignment

# Adding labels and title

plt.xlabel(x)

plt.ylabel('Number of Records')

plt.title(str(x) + ' - Frequence Binning')

# Save the plot to a buffer

buf = io.BytesIO()

plt.savefig(buf, format='png')

buf.seek(0)

# Encode the image in base64 and log it

img\_base64 = base64.b64encode(buf.read()).decode('utf-8')

# Save the plot as a file

output\_filename = '../output/Binning\_Frequency\_' + str(x)

plt.savefig(output\_filename)

logger.info("Binning plot saved as" + str(output\_filename))

# Close the buffer

buf.close()

plt.close()

plot\_distance\_bins('Age')

plot\_frequency\_bins('Age')

plot\_distance\_bins('HeartRate')

plot\_frequency\_bins('HeartRate')

plot\_distance\_bins('BloodSugar')

plot\_frequency\_bins('BloodSugar')

(apip\_project\_venv) adimulamramkumar-mac:tasks adimulamramkumar$ python Binning.py

2024-10-02 12:14:48,493 - INFO - Dataset loaded for binning.

2024-10-02 12:14:48,494 - INFO - Min Age: 14, Max Age: 103

2024-10-02 12:14:48,494 - INFO - Bins: [ 14. 36.25 58.5 80.75 103. ]

2024-10-02 12:14:48,496 - INFO - Distance Binning Results:

0 Middle Age

1 Juvenile

2 Adult

3 Middle Age

4 Adult

...

1314 Adult

1315 Middle Age

1316 Adult

1317 Adult

1318 Adult

Name: bins\_dist, Length: 1319, dtype: category

Categories (4, object): ['Juvenile' < 'Adult' < 'Middle Age' < 'Senior Citizen']

2024-10-02 12:14:48,796 - INFO - Binning plot saved as../output/Binning\_Distance\_Age

2024-10-02 12:14:48,797 - INFO - Min Age: 14, Max Age: 103

2024-10-02 12:14:48,798 - INFO - Bins: [ 14. 45. 54. 60. 68. 103.]

2024-10-02 12:14:48,800 - INFO - Frequency Binning Results:

0 Middle Age

1 Juvenile

2 Adult

3 Senior Citizen

4 Adult

...

1314 Juvenile

1315 Senior Citizen

1316 Juvenile

1317 Adult

1318 Adult

Name: bin\_freq, Length: 1319, dtype: category

Categories (4, object): ['Juvenile' < 'Adult' < 'Middle Age' < 'Senior Citizen']

2024-10-02 12:14:49,000 - INFO - Binning plot saved as../output/Binning\_Frequency\_Age

2024-10-02 12:14:49,001 - INFO - Min HeartRate: 20, Max HeartRate: 135

2024-10-02 12:14:49,001 - INFO - Bins: [ 20. 58.333 96.667 135. ]

2024-10-02 12:14:49,003 - INFO - Distance Binning Results:

0 Normal

1 Normal

2 Normal

3 Normal

4 Normal

...

1314 Normal

1315 Normal

1316 Normal

1317 Low

1318 Normal

Name: bins\_dist, Length: 1319, dtype: category

Categories (3, object): ['Low' < 'Normal' < 'High']

2024-10-02 12:14:49,220 - INFO - Binning plot saved as../output/Binning\_Distance\_HeartRate

2024-10-02 12:14:49,221 - INFO - Min HeartRate: 20, Max HeartRate: 135

2024-10-02 12:14:49,221 - INFO - Bins: [ 20. 64. 74. 85. 135.]

2024-10-02 12:14:49,224 - INFO - Frequency Binning Results:

0 Low

1 High

2 Low

3 Normal

4 Low

...

1314 High

1315 High

1316 High

1317 Low

1318 High

Name: bin\_freq, Length: 1319, dtype: category

Categories (3, object): ['Low' < 'Normal' < 'High']

2024-10-02 12:14:49,401 - INFO - Binning plot saved as../output/Binning\_Frequency\_HeartRate

2024-10-02 12:14:49,402 - INFO - Min BloodSugar: 35.0, Max BloodSugar: 541.0

2024-10-02 12:14:49,402 - INFO - Bins: [ 35. 203.667 372.333 541. ]

2024-10-02 12:14:49,404 - INFO - Distance Binning Results:

0 Low

1 Normal

2 Normal

3 Normal

4 Normal

...

1314 Normal

1315 Low

1316 Low

1317 High

1318 Low

Name: bins\_dist, Length: 1319, dtype: category

Categories (3, object): ['Low' < 'Normal' < 'High']

2024-10-02 12:14:49,581 - INFO - Binning plot saved as../output/Binning\_Distance\_BloodSugar

2024-10-02 12:14:49,581 - INFO - Min BloodSugar: 35.0, Max BloodSugar: 541.0

2024-10-02 12:14:49,582 - INFO - Bins: [ 35. 98. 116. 170. 541.]

2024-10-02 12:14:49,584 - INFO - Frequency Binning Results:

0 High

1 High

2 High

3 High

4 High

...

1314 High

1315 High

1316 Low

1317 High

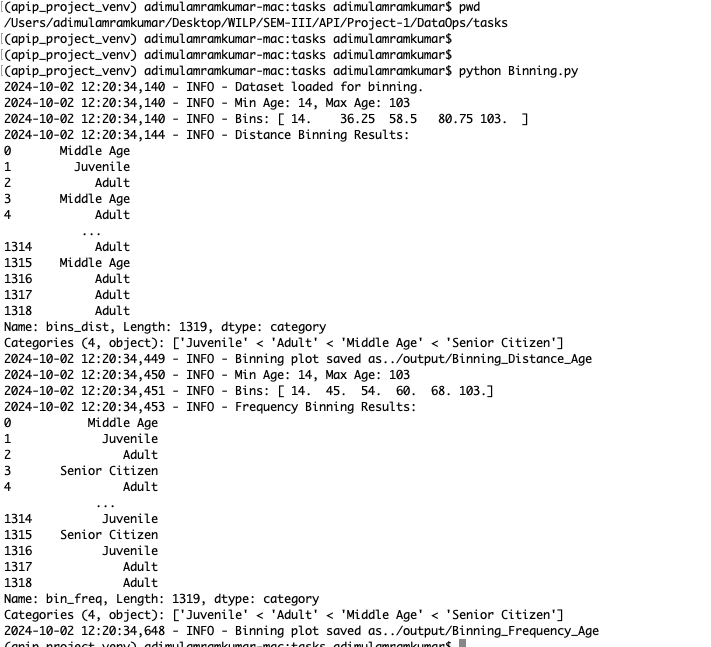
1318 Normal

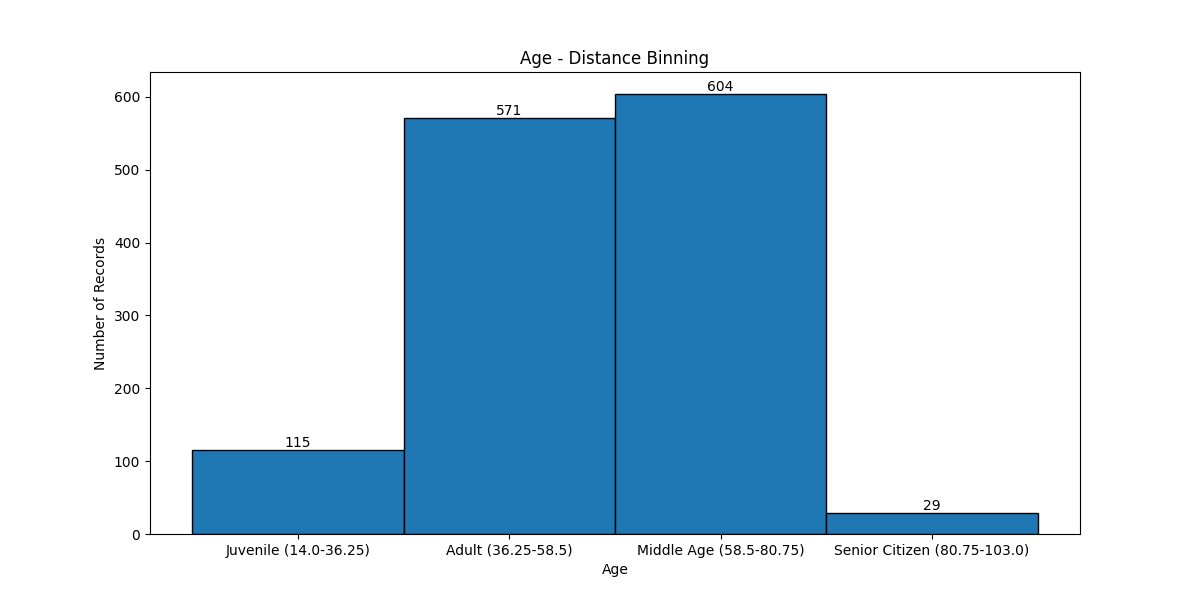
Name: bin\_freq, Length: 1319, dtype: category

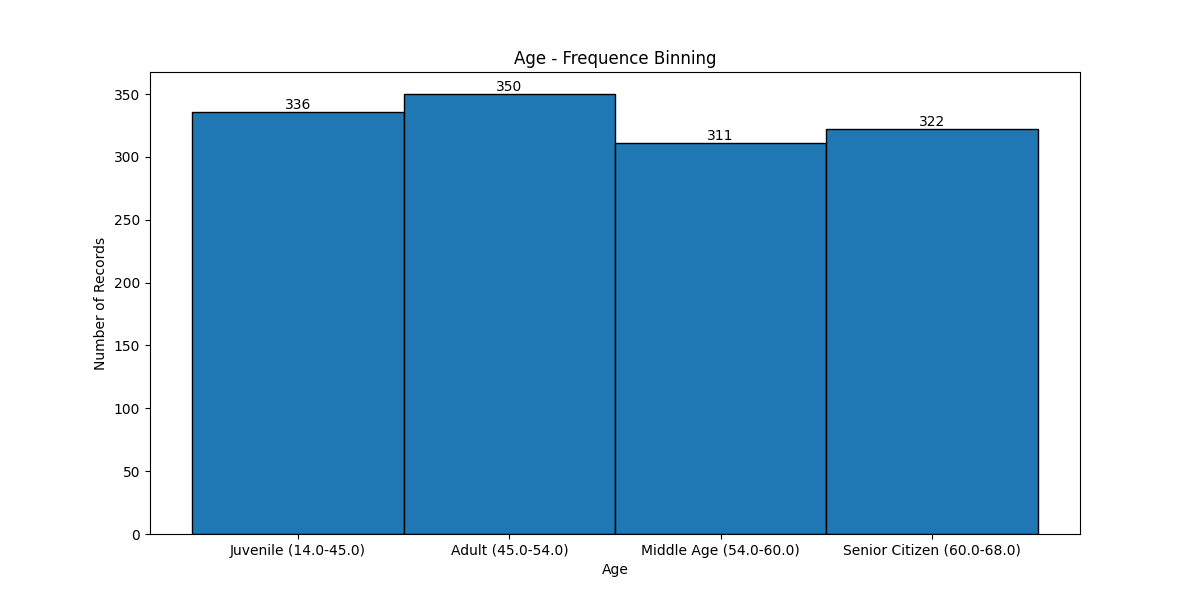
Categories (3, object): ['Low' < 'Normal' < 'High']

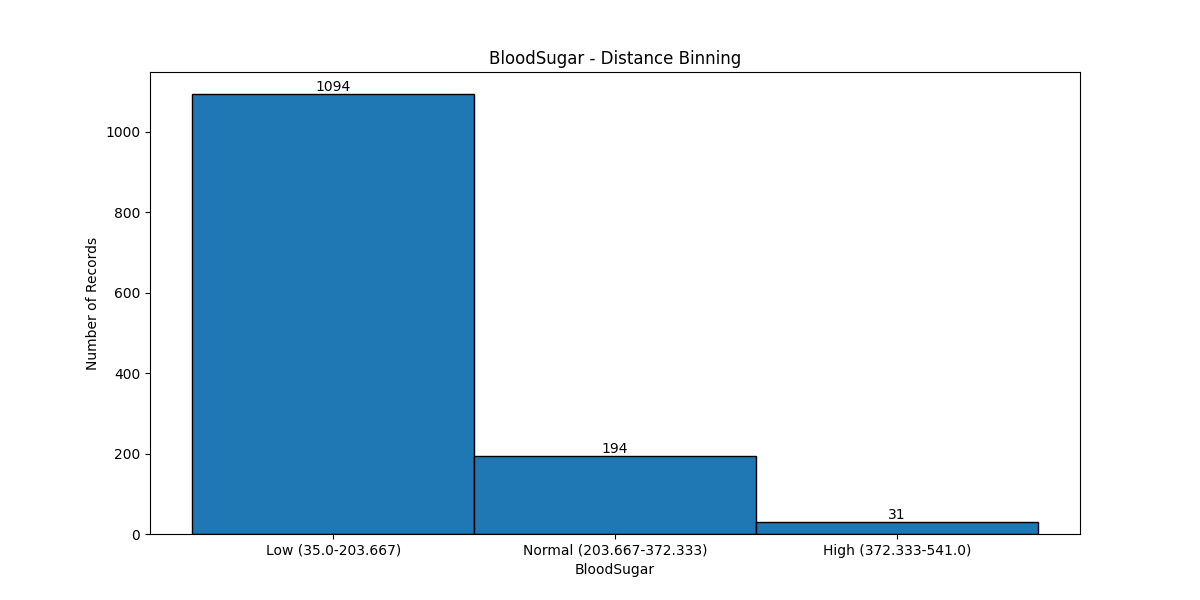
2024-10-02 12:14:49,770 - INFO - Binning plot saved as../output/Binning\_Frequency\_BloodSugar

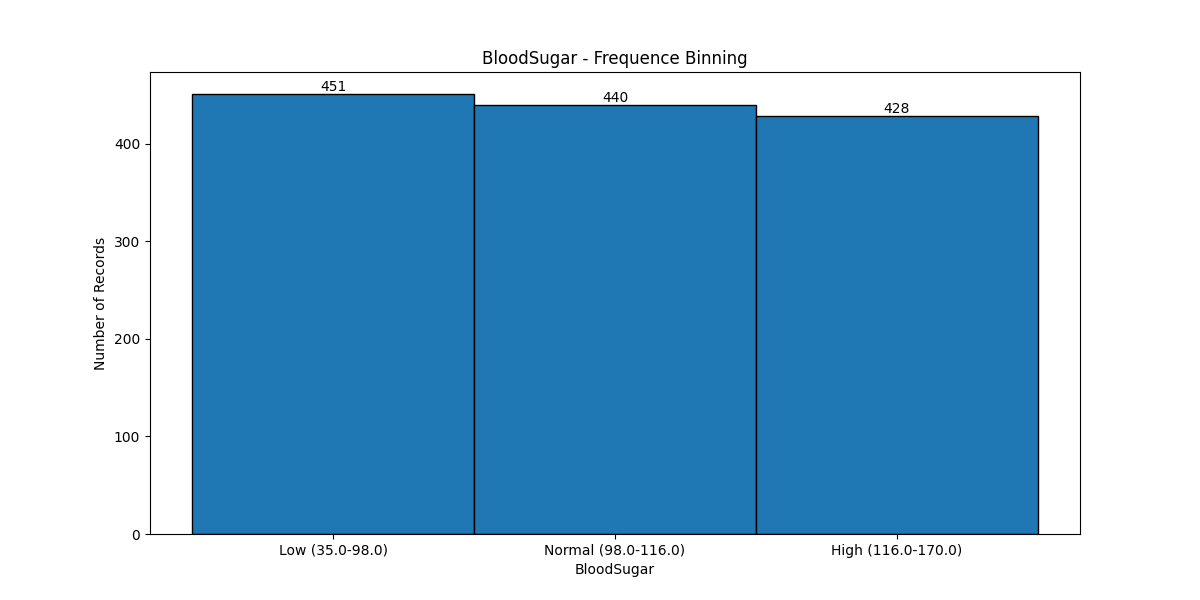
(apip\_project\_venv) adimulamramkumar-mac:tasks adimulamramkumar$

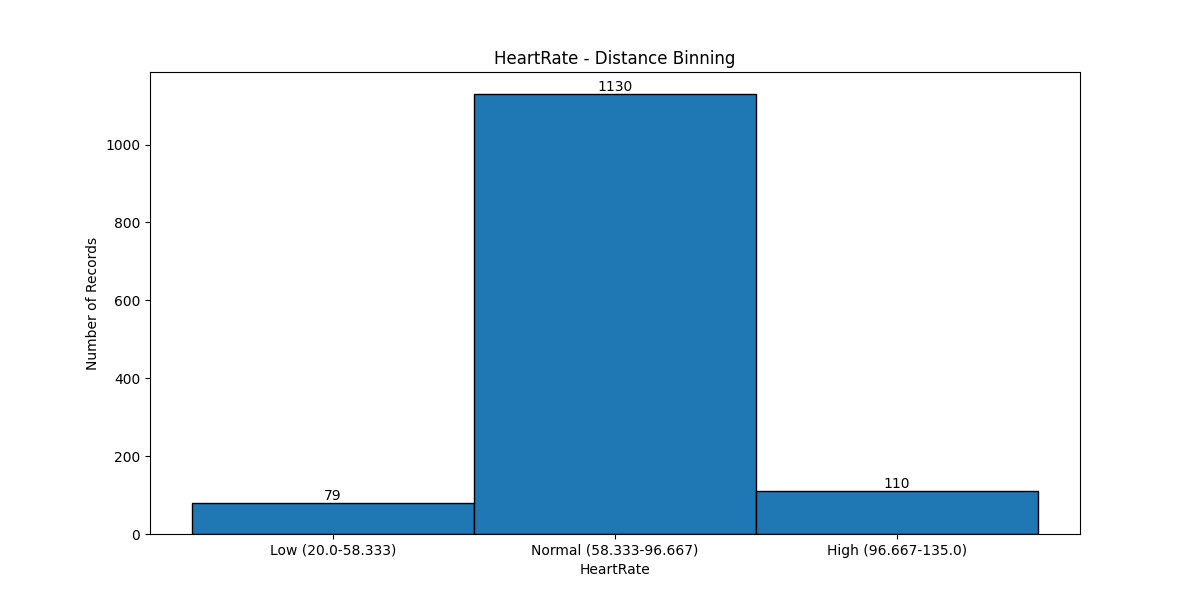
****

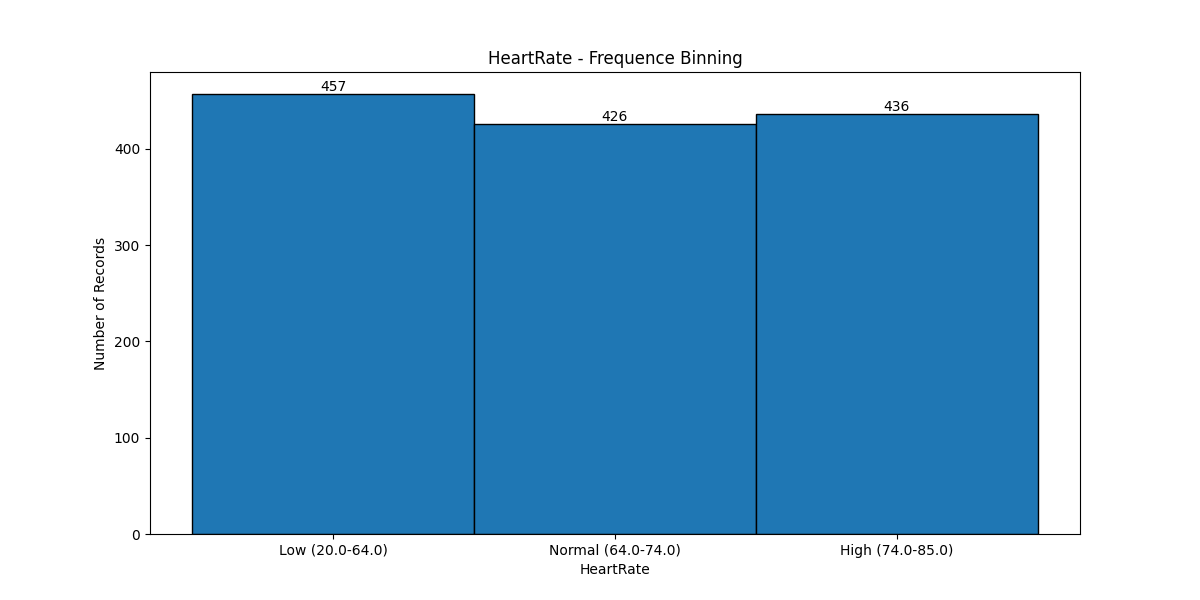
****

****

****

****

****

****

1.4.2 Bivariate Analysis – Pearson Correlation

Bivariate analysis refers to the statistical analysis of two variables to explore their relationship, association, or correlation. It examines how one variable changes concerning another and aims to understand the underlying patterns and associations between the two variables.

The **Pearson correlation coefficient** (often denoted as **r**) is a measure of the linear relationship between two numerical variables. It quantifies how strongly the two variables are linearly related and whether the relationship is positive or negative.

import pandas as pd

from scipy.stats import pearsonr

import logging

from matplotlib import pyplot as plt

import io

import base64

from tabulate import tabulate

# Configure standard Python logging

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')

logger = logging.getLogger(\_\_name\_\_)

# Import the data

df = pd.read\_csv("../data/MedicalDatasetForModelNormalized.csv")

logger.info("Dataset loaded for Pearson correlation.")

# store results

result\_data = []

def compute\_pearson\_correlation(x, y):

# Convert columns to series

list1 = df[x]

list2 = df[y]

# Compute Pearson correlation

corr, \_ = pearsonr(list1, list2)

logger.info(f'Pearson correlation between {x} and {y}: {corr:.3f}')

# Scatter plot

plt.scatter(list1, list2)

plt.xlabel(x)

plt.ylabel(y)

plt.title('Scatter plot of ' + str(x) + ' vs ' + str(y))

# Save the plot to a buffer

buf = io.BytesIO()

plt.savefig(buf, format='png')

buf.seek(0)

# Encode the image in base64 and log it

img\_base64 = base64.b64encode(buf.read()).decode('utf-8')

# Save the plot as a file

output\_filename = '../output/' + str(x) + '\_' + str(y)

plt.savefig(output\_filename)

logger.info("Scatter plot saved as" + str(output\_filename))

# Close the buffer

buf.close()

plt.close()

result\_data.append([x, y, corr])

compute\_pearson\_correlation('Age', 'HeartRate')

compute\_pearson\_correlation('Age', 'BloodSugar')

compute\_pearson\_correlation('Age', 'Troponin')

compute\_pearson\_correlation('Age', 'CKMB')

compute\_pearson\_correlation('BloodSugar', 'Troponin')

compute\_pearson\_correlation('SystolicBloodPressure', 'Troponin')

compute\_pearson\_correlation('DiastolicBloodPressure', 'Troponin')

compute\_pearson\_correlation('Age', 'Result')

compute\_pearson\_correlation('BloodSugar', 'Result')

compute\_pearson\_correlation('CKMB', 'Result')

compute\_pearson\_correlation('Troponin', 'Result')

headers = ["Variable1", "Variable2", "PearsonCorrelation"]

print(tabulate(result\_data, headers = headers, tablefmt = "fancy\_grid"))

(apip\_project\_venv) adimulamramkumar-mac:tasks adimulamramkumar$ python PearsonCorrelation.py

2024-10-02 13:41:03,676 - INFO - Dataset loaded for Pearson correlation.

2024-10-02 13:41:03,678 - INFO - Pearson correlation between Age and HeartRate: -0.001

2024-10-02 13:41:03,929 - INFO - Scatter plot saved as../output/Age\_HeartRate

2024-10-02 13:41:03,931 - INFO - Pearson correlation between Age and BloodSugar: -0.004

2024-10-02 13:41:04,062 - INFO - Scatter plot saved as../output/Age\_BloodSugar

2024-10-02 13:41:04,064 - INFO - Pearson correlation between Age and Troponin: 0.089

2024-10-02 13:41:04,192 - INFO - Scatter plot saved as../output/Age\_Troponin

2024-10-02 13:41:04,194 - INFO - Pearson correlation between Age and CKMB: 0.018

2024-10-02 13:41:04,323 - INFO - Scatter plot saved as../output/Age\_CKMB

2024-10-02 13:41:04,324 - INFO - Pearson correlation between BloodSugar and Troponin: 0.021

2024-10-02 13:41:04,454 - INFO - Scatter plot saved as../output/BloodSugar\_Troponin

2024-10-02 13:41:04,456 - INFO - Pearson correlation between SystolicBloodPressure and Troponin: 0.044

2024-10-02 13:41:04,603 - INFO - Scatter plot saved as../output/SystolicBloodPressure\_Troponin

2024-10-02 13:41:04,605 - INFO - Pearson correlation between DiastolicBloodPressure and Troponin: 0.043

2024-10-02 13:41:04,741 - INFO - Scatter plot saved as../output/DiastolicBloodPressure\_Troponin

2024-10-02 13:41:04,743 - INFO - Pearson correlation between Age and Result: 0.238

2024-10-02 13:41:04,864 - INFO - Scatter plot saved as../output/Age\_Result

2024-10-02 13:41:04,865 - INFO - Pearson correlation between BloodSugar and Result: -0.033

2024-10-02 13:41:05,022 - INFO - Scatter plot saved as../output/BloodSugar\_Result

2024-10-02 13:41:05,023 - INFO - Pearson correlation between CKMB and Result: 0.218

2024-10-02 13:41:05,155 - INFO - Scatter plot saved as../output/CKMB\_Result

2024-10-02 13:41:05,157 - INFO - Pearson correlation between Troponin and Result: 0.229

2024-10-02 13:41:05,294 - INFO - Scatter plot saved as../output/Troponin\_Result

╒════════════════════════╤═════════════╤═════════════

│ Variable1 │ Variable2 │ PearsonCorrelation │

╞════════════════════════╪═════════════╪═════════════

│ Age │ HeartRate │ -0.00135128 │

├────────────────────────┼─────────────┼──────────────────────┤

│ Age │ BloodSugar │ -0.00400739 │

├────────────────────────┼─────────────┼──────────────────────┤

│ Age │ Troponin │ 0.0885566 │

├────────────────────────┼─────────────┼──────────────────────┤

│ Age │ CKMB │ 0.0177001 │

├────────────────────────┼─────────────┼──────────────────────┤

│ BloodSugar │ Troponin │ 0.0210689 │

├────────────────────────┼─────────────┼──────────────────────┤

│ SystolicBloodPressure │ Troponin │ 0.0437288 │

├────────────────────────┼─────────────┼──────────────────────┤

│ DiastolicBloodPressure │ Troponin │ 0.04336 │

├────────────────────────┼─────────────┼──────────────────────┤

│ Age │ Result │ 0.238002 │

├────────────────────────┼─────────────┼──────────────────────┤

│ BloodSugar │ Result │ -0.0330594 │

├────────────────────────┼─────────────┼──────────────────────┤

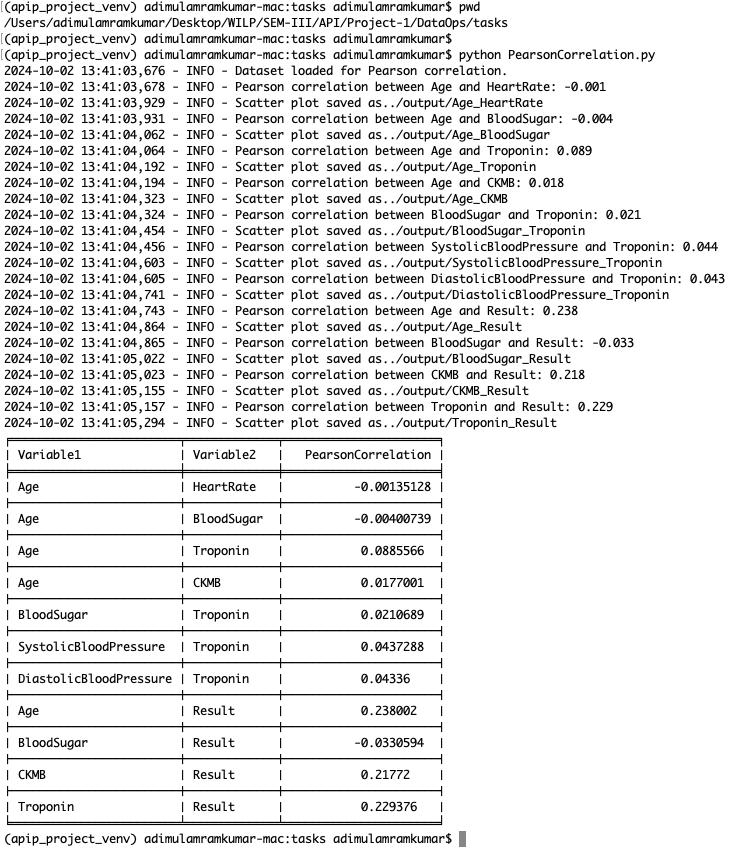
│ CKMB │ Result │ 0.21772 │

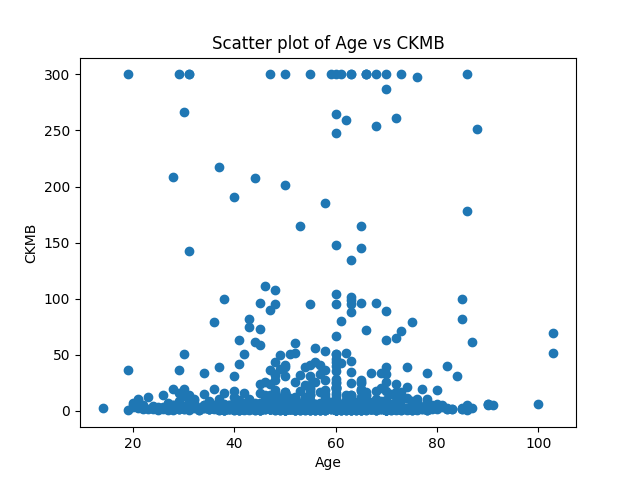
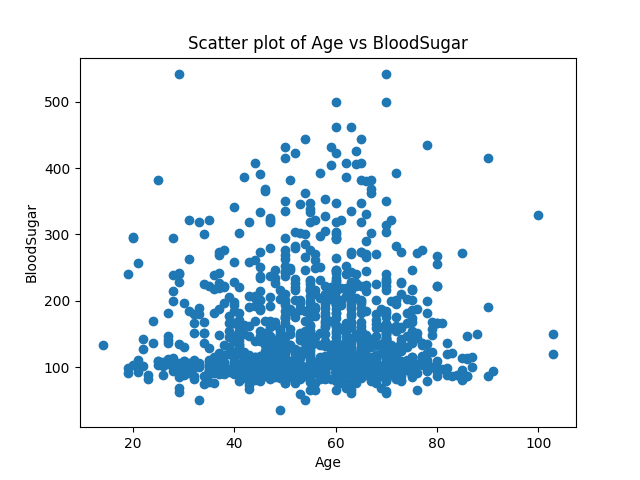
├────────────────────────┼─────────────┼──────────────────────┤

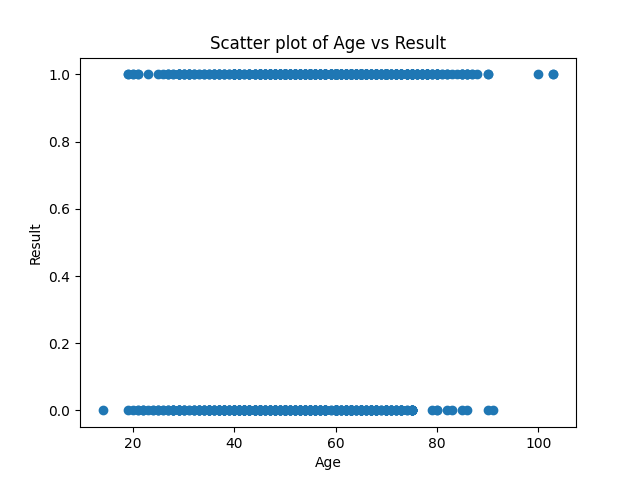
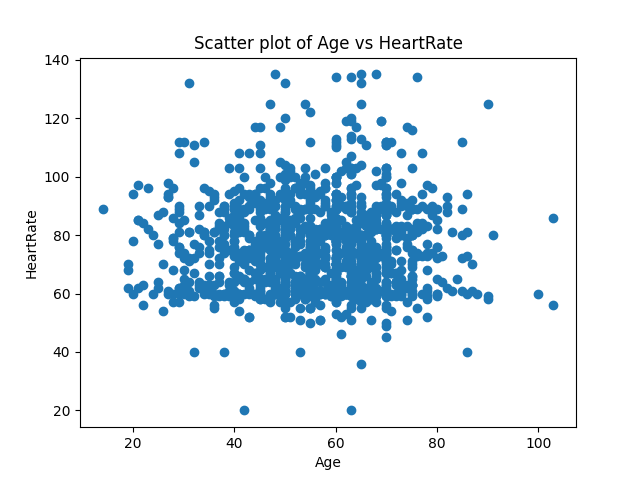
│ Troponin │ Result │ 0.229376 │

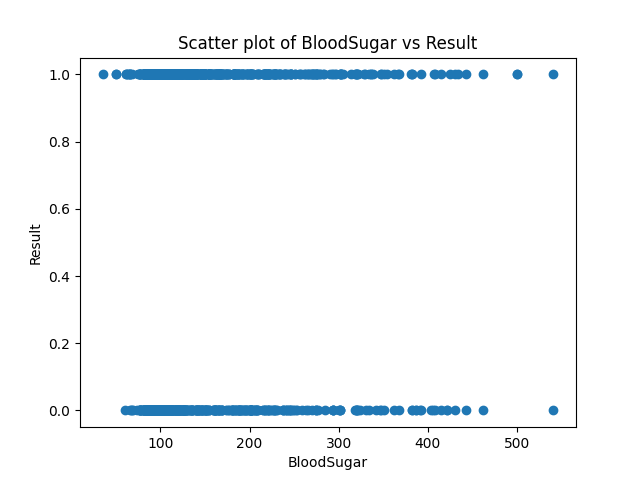
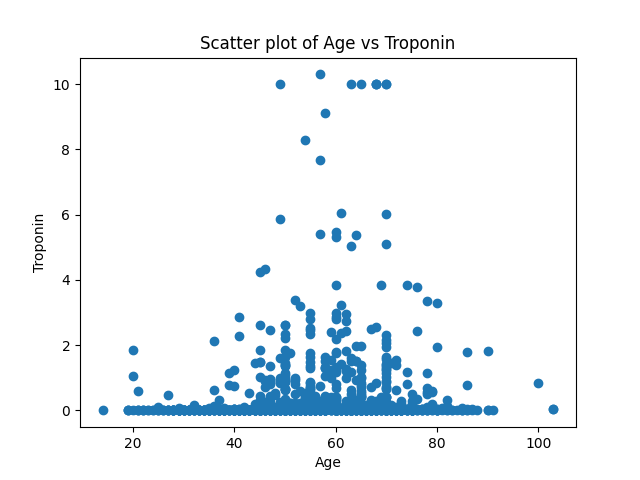
╘════════════════════════╧═════════════╧═════════════

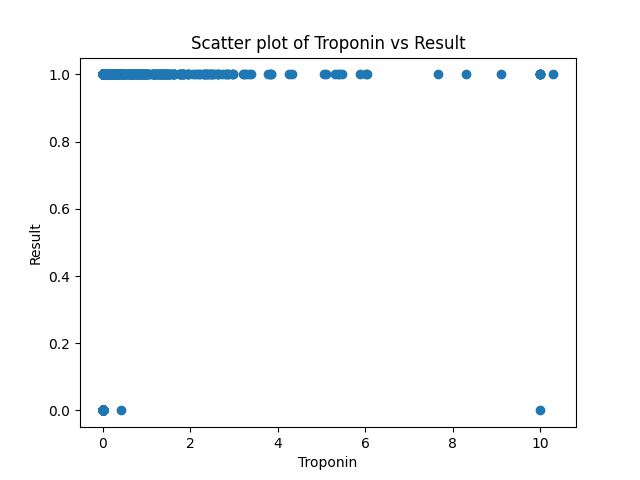
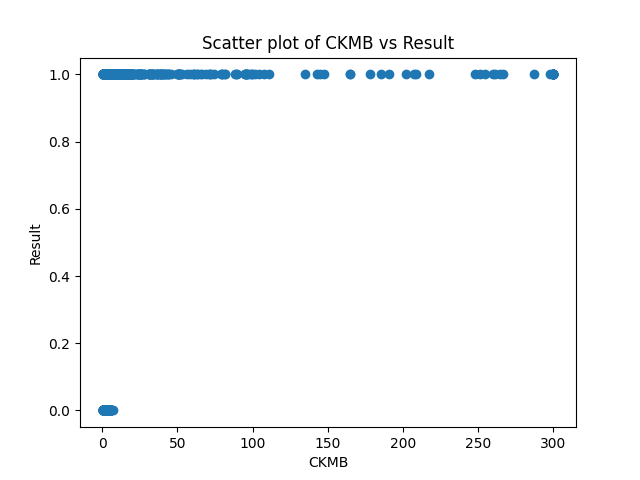
(apip\_project\_venv) adimulamramkumar-mac:tasks adimulamramkumar$

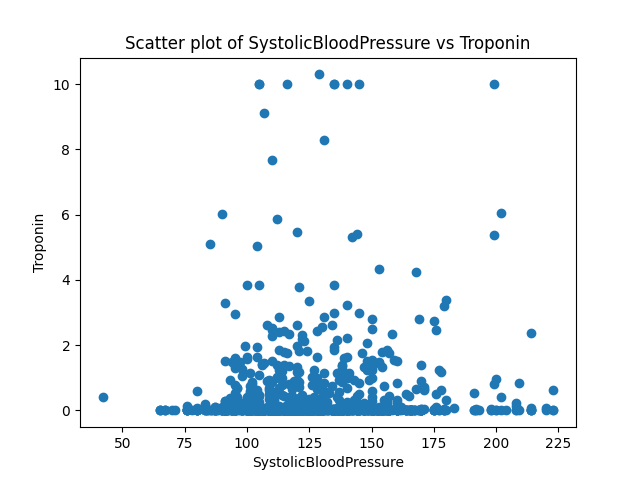
****

****

****

****

****

****

1.5 DataOps: Implement workflows to automate activities from steps 1.3 and 1.4 within a data pipeline. Schedule these workflows to run every 2 minutes, logging all activity details and displaying them on a Cloud dashboard.

1.5.1 Prefect Workflow

from prefect import flow, task, get\_run\_logger

import subprocess

import os

@task(log\_prints=True) # Ensures all prints are captured in logs

def run\_task(script\_name):

logger = get\_run\_logger()

# Define the full path to the script based on the project structure

script\_path = os.path.join(os.path.dirname(\_\_file\_\_), '../tasks', script\_name)

try:

# Run the external Python script using its full path

result = subprocess.run(['python', script\_path], capture\_output=True, text=True)

# Log both stdout and stderr

if result.returncode == 0:

logger.info(f"Successfully executed {script\_name}:\n{result.stdout}") # Log standard output

else:

logger.error(f"Error in {script\_name}: {result.stderr}") # Log standard error

# Always print both stdout and stderr, regardless of success or failure

print(result.stdout)

print(result.stderr)

except Exception as e:

logger.error(f"Failed to execute {script\_name}: {str(e)}")

return 0

@flow

def main\_flow():

# Run tasks sequentially and capture the results

data1 = run\_task("BasicStats.py")

data2 = run\_task("Binning.py", wait\_for=[data1])

data3 = run\_task("PearsonCorrelation.py", wait\_for=[data2])

# To run locally

if \_\_name\_\_ == "\_\_main\_\_":

main\_flow.serve(name="heart-attack-risk-workflow",

tags=["heart attack risk datascience project workflow"],

parameters={},

interval=120)

1.5.2 Running Prefect Workflow on Cloud

Pre-Requisites

1. Created a cloud-account in <https://www.prefect.io/> using BITS WILP mail ID.
2. Use ‘code’ received in email to login to prefect.io
3. Go to ‘My Profile’ -> API Keys -> Created a new API Key. Note down the API Key, I think we won’t be able to retrieve it again, so better save it somewhere. API Key starts with ‘pnu\_’.
4. Login using prefect command-line tool, list workspace, set the default workspace.

(apip\_project\_venv) adimulamramkumar-mac:DataOps adimulamramkumar$ prefect cloud login -k pnu\_4t3M9YNXs6TytZ5PNmzIM2gcicvXQL16BZBG

It looks like you're already authenticated on this profile.

? Would you like to reauthenticate? [y/N]: y

Authenticated with Prefect Cloud! Using workspace 'bitspilani-wilp/default'.

(apip\_project\_venv) adimulamramkumar-mac:DataOps adimulamramkumar$ prefect cloud workspace ls

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━┓

┃ Workspaces: ┃

┡━━━━━━━━━━━━━━━━━━━━━━━━━━━┩

│ \* bitspilani-wilp/default │

└───────────────────────────┘

\* active workspace

(apip\_project\_venv) adimulamramkumar-mac:DataOps adimulamramkumar$ prefect cloud workspace set --workspace bitspilani-wilp/default

Successfully set workspace to 'bitspilani-wilp/default' in profile 'local'.

(apip\_project\_venv) adimulamramkumar-mac:DataOps adimulamramkumar$

Running Workflow in the cloud

In one-terminal, run workflow.py, let the terminal be active as is:

(apip\_project\_venv) adimulamramkumar-mac:flows adimulamramkumar$ python workflow.py

Your flow 'main-flow' is being served and polling for scheduled runs!

To trigger a run for this flow, use the following command:

$ prefect deployment run 'main-flow/heart-attack-risk-workflow'

You can also run your flow via the Prefect UI: <https://app.prefect.cloud/account/447374eb-6251-4bfb-b0c7-18f7c4be91d0/workspace/e15c14f8-cea0-41dc-bd42-d4b2899de804/deployments/deployment/978ef9be-4a04-40c3-a7f0-693ae6af8569>

**In a second terminal - trigger a run:**

(apip\_project\_venv) adimulamramkumar-mac:flows adimulamramkumar$   
prefect deployment run 'main-flow/heart-attack-risk-workflow'

Creating flow run for deployment 'main-flow/heart-attack-risk-workflow'...

Created flow run 'benevolent-sawfish'.

└── UUID: 60130e7d-2d1d-4cbb-ae81-0968350c6fc1

└── Parameters: {}

└── Job Variables: {}

└── Scheduled start time: 2024-10-02 14:02:07 IST (now)

└── URL: https://app.prefect.cloud/account/447374eb-6251-4bfb-b0c7-18f7c4be91d0/workspace/e15c14f8-cea0-41dc-bd42-d4b2899de804/runs/flow-run/60130e7d-2d1d-4cbb-ae81-0968350c6fc1

(apip\_project\_venv) adimulamramkumar-mac:flows adimulamramkumar$

In the first terminal, we can see the DataOps flow activity and also we can check in the cloud:

**Logs seen in local terminal:**

==

14:16:08.228 | INFO | prefect.flow\_runs.runner - Runner 'heart-attack-risk-workflow' submitting flow run '81d551a8-17ca-45aa-8c8c-57807869e049'

14:16:08.984 | INFO | prefect.flow\_runs.runner - Opening process...

14:16:08.998 | INFO | prefect.flow\_runs.runner - Completed submission of flow run '81d551a8-17ca-45aa-8c8c-57807869e049'

14:16:11.994 | INFO | Flow run 'tricky-elephant' - Downloading flow code from storage at '.'

14:16:14.247 | INFO | Task run 'run\_task-f9a' - Created task run 'run\_task-f9a' for task 'run\_task'

14:16:16.079 | INFO | Task run 'run\_task-f9a' - Successfully executed BasicStats.py:

14:16:16.081 | INFO | Task run 'run\_task-f9a' -

14:16:16.081 | INFO | Task run 'run\_task-f9a' - 2024-10-02 14:16:15,863 - INFO - Dataset loaded successfully.

2024-10-02 14:16:15,874 - INFO - DataFrame head:

Age Gender HeartRate ... CKMB Troponin Result

0 63 M 66 ... 1.80 0.012 negative

1 20 M 94 ... 6.75 1.060 positive

2 56 M 64 ... 1.99 0.003 negative

3 66 M 70 ... 13.87 0.122 positive

4 54 M 64 ... 1.08 0.003 negative

[5 rows x 9 columns]

2024-10-02 14:16:15,876 - INFO - Converted Gender fields from M, F to 0,1.

2024-10-02 14:16:15,876 - INFO - Converted Results fields from negative,positive to 0,1.

2024-10-02 14:16:15,876 - INFO - saving normalized data back into a new csv file

2024-10-02 14:16:15,882 - INFO - Summary Statistics:

2024-10-02 14:16:15,903 - INFO -

Age Gender ... Troponin Result

count 1319.000000 1319.000000 ... 1319.000000 1319.000000

mean 56.193328 0.340409 ... 0.360942 0.614102

std 13.638173 0.474027 ... 1.154568 0.486991

min 14.000000 0.000000 ... 0.001000 0.000000

25% 47.000000 0.000000 ... 0.006000 0.000000

50% 58.000000 0.000000 ... 0.014000 1.000000

75% 65.000000 1.000000 ... 0.085500 1.000000

max 103.000000 1.000000 ... 10.300000 1.000000

[8 rows x 9 columns]

2024-10-02 14:16:15,903 - INFO - Missing Values:

2024-10-02 14:16:15,904 - INFO -

Age 0

Gender 0

HeartRate 0

SystolicBloodPressure 0

DiastolicBloodPressure 0

BloodSugar 0

CKMB 0

Troponin 0

Result 0

dtype: int64

2024-10-02 14:16:15,904 - INFO - Data Types:

2024-10-02 14:16:15,904 - INFO -

Age int64

Gender int64

HeartRate int64

SystolicBloodPressure int64

DiastolicBloodPressure int64

BloodSugar float64

CKMB float64

Troponin float64

Result int64

dtype: object

14:16:16.087 | INFO | Task run 'run\_task-f9a' - Finished in state Completed()

14:16:16.102 | INFO | Task run 'run\_task-210' - Created task run 'run\_task-210' for task 'run\_task'

14:16:17.743 | INFO | Task run 'run\_task-210' - Successfully executed Binning.py:

14:16:17.745 | INFO | Task run 'run\_task-210' -

14:16:17.746 | INFO | Task run 'run\_task-210' - 2024-10-02 14:16:17,070 - INFO - Dataset loaded for binning.

2024-10-02 14:16:17,070 - INFO - Min Age: 14, Max Age: 103

2024-10-02 14:16:17,070 - INFO - Bins: [ 14. 36.25 58.5 80.75 103. ]

2024-10-02 14:16:17,073 - INFO - Distance Binning Results:

0 Middle Age

1 Juvenile

2 Adult

3 Middle Age

4 Adult

...

1314 Adult

1315 Middle Age

1316 Adult

1317 Adult

1318 Adult

Name: bins\_dist, Length: 1319, dtype: category

Categories (4, object): ['Juvenile' < 'Adult' < 'Middle Age' < 'Senior Citizen']

2024-10-02 14:16:17,420 - INFO - Binning plot saved as../output/Binning\_Distance\_Age

2024-10-02 14:16:17,420 - INFO - Min Age: 14, Max Age: 103

2024-10-02 14:16:17,421 - INFO - Bins: [ 14. 45. 54. 60. 68. 103.]

2024-10-02 14:16:17,425 - INFO - Frequency Binning Results:

0 Middle Age

1 Juvenile

2 Adult

3 Senior Citizen

4 Adult

...

1314 Juvenile

1315 Senior Citizen

1316 Juvenile

1317 Adult

1318 Adult

Name: bin\_freq, Length: 1319, dtype: category

Categories (4, object): ['Juvenile' < 'Adult' < 'Middle Age' < 'Senior Citizen']

2024-10-02 14:16:17,634 - INFO - Binning plot saved as../output/Binning\_Frequency\_Age

14:16:17.751 | INFO | Task run 'run\_task-210' - Finished in state Completed()

14:16:17.768 | INFO | Task run 'run\_task-c8d' - Created task run 'run\_task-c8d' for task 'run\_task'

14:16:21.443 | INFO | Task run 'run\_task-c8d' - Successfully executed PearsonCorrelation.py:

╒════════════════════════╤═════════════╤═════════════

│ Variable1 │ Variable2 │ PearsonCorrelation │

╞════════════════════════╪═════════════╪═════════════

│ Age │ HeartRate │ -0.00135128 │

├────────────────────────┼─────────────┼──────────────────────┤

│ Age │ BloodSugar │ -0.00400739 │

├────────────────────────┼─────────────┼──────────────────────┤

│ Age │ Troponin │ 0.0885566 │

├────────────────────────┼─────────────┼──────────────────────┤

│ Age │ CKMB │ 0.0177001 │

├────────────────────────┼─────────────┼──────────────────────┤

│ BloodSugar │ Troponin │ 0.0210689 │

├────────────────────────┼─────────────┼──────────────────────┤

│ SystolicBloodPressure │ Troponin │ 0.0437288 │

├────────────────────────┼─────────────┼──────────────────────┤

│ DiastolicBloodPressure │ Troponin │ 0.04336 │

├────────────────────────┼─────────────┼──────────────────────┤

│ Age │ Result │ 0.238002 │

├────────────────────────┼─────────────┼──────────────────────┤

│ BloodSugar │ Result │ -0.0330594 │

├────────────────────────┼─────────────┼──────────────────────┤

│ CKMB │ Result │ 0.21772 │

├────────────────────────┼─────────────┼──────────────────────┤

│ Troponin │ Result │ 0.229376 │

╘════════════════════════╧═════════════╧═════════════

14:16:21.447 | INFO | Task run 'run\_task-c8d' - ╒════════════════════════╤═════════════╤═════════════

│ Variable1 │ Variable2 │ PearsonCorrelation │

╞════════════════════════╪═════════════╪═════════════

│ Age │ HeartRate │ -0.00135128 │

├────────────────────────┼─────────────┼──────────────────────┤

│ Age │ BloodSugar │ -0.00400739 │

├────────────────────────┼─────────────┼──────────────────────┤

│ Age │ Troponin │ 0.0885566 │

├────────────────────────┼─────────────┼──────────────────────┤

│ Age │ CKMB │ 0.0177001 │

├────────────────────────┼─────────────┼──────────────────────┤

│ BloodSugar │ Troponin │ 0.0210689 │

├────────────────────────┼─────────────┼──────────────────────┤

│ SystolicBloodPressure │ Troponin │ 0.0437288 │

├────────────────────────┼─────────────┼──────────────────────┤

│ DiastolicBloodPressure │ Troponin │ 0.04336 │

├────────────────────────┼─────────────┼──────────────────────┤

│ Age │ Result │ 0.238002 │

├────────────────────────┼─────────────┼──────────────────────┤

│ BloodSugar │ Result │ -0.0330594 │

├────────────────────────┼─────────────┼──────────────────────┤

│ CKMB │ Result │ 0.21772 │

├────────────────────────┼─────────────┼──────────────────────┤

│ Troponin │ Result │ 0.229376 │

╘════════════════════════╧═════════════╧═════════════

14:16:21.449 | INFO | Task run 'run\_task-c8d' - 2024-10-02 14:16:19,407 - INFO - Dataset loaded for Pearson correlation.

2024-10-02 14:16:19,409 - INFO - Pearson correlation between Age and HeartRate: -0.001

2024-10-02 14:16:19,697 - INFO - Scatter plot saved as../output/Age\_HeartRate

2024-10-02 14:16:19,699 - INFO - Pearson correlation between Age and BloodSugar: -0.004

2024-10-02 14:16:19,853 - INFO - Scatter plot saved as../output/Age\_BloodSugar

2024-10-02 14:16:19,855 - INFO - Pearson correlation between Age and Troponin: 0.089

2024-10-02 14:16:20,008 - INFO - Scatter plot saved as../output/Age\_Troponin

2024-10-02 14:16:20,009 - INFO - Pearson correlation between Age and CKMB: 0.018

2024-10-02 14:16:20,162 - INFO - Scatter plot saved as../output/Age\_CKMB

2024-10-02 14:16:20,163 - INFO - Pearson correlation between BloodSugar and Troponin: 0.021

2024-10-02 14:16:20,314 - INFO - Scatter plot saved as../output/BloodSugar\_Troponin

2024-10-02 14:16:20,316 - INFO - Pearson correlation between SystolicBloodPressure and Troponin: 0.044

2024-10-02 14:16:20,479 - INFO - Scatter plot saved as../output/SystolicBloodPressure\_Troponin

2024-10-02 14:16:20,481 - INFO - Pearson correlation between DiastolicBloodPressure and Troponin: 0.043

2024-10-02 14:16:20,642 - INFO - Scatter plot saved as../output/DiastolicBloodPressure\_Troponin

2024-10-02 14:16:20,643 - INFO - Pearson correlation between Age and Result: 0.238

2024-10-02 14:16:20,788 - INFO - Scatter plot saved as../output/Age\_Result

2024-10-02 14:16:20,790 - INFO - Pearson correlation between BloodSugar and Result: -0.033

2024-10-02 14:16:20,975 - INFO - Scatter plot saved as../output/BloodSugar\_Result

2024-10-02 14:16:20,977 - INFO - Pearson correlation between CKMB and Result: 0.218

2024-10-02 14:16:21,123 - INFO - Scatter plot saved as../output/CKMB\_Result

2024-10-02 14:16:21,125 - INFO - Pearson correlation between Troponin and Result: 0.229

2024-10-02 14:16:21,270 - INFO - Scatter plot saved as../output/Troponin\_Result

14:16:21.454 | INFO | Task run 'run\_task-c8d' - Finished in state Completed()

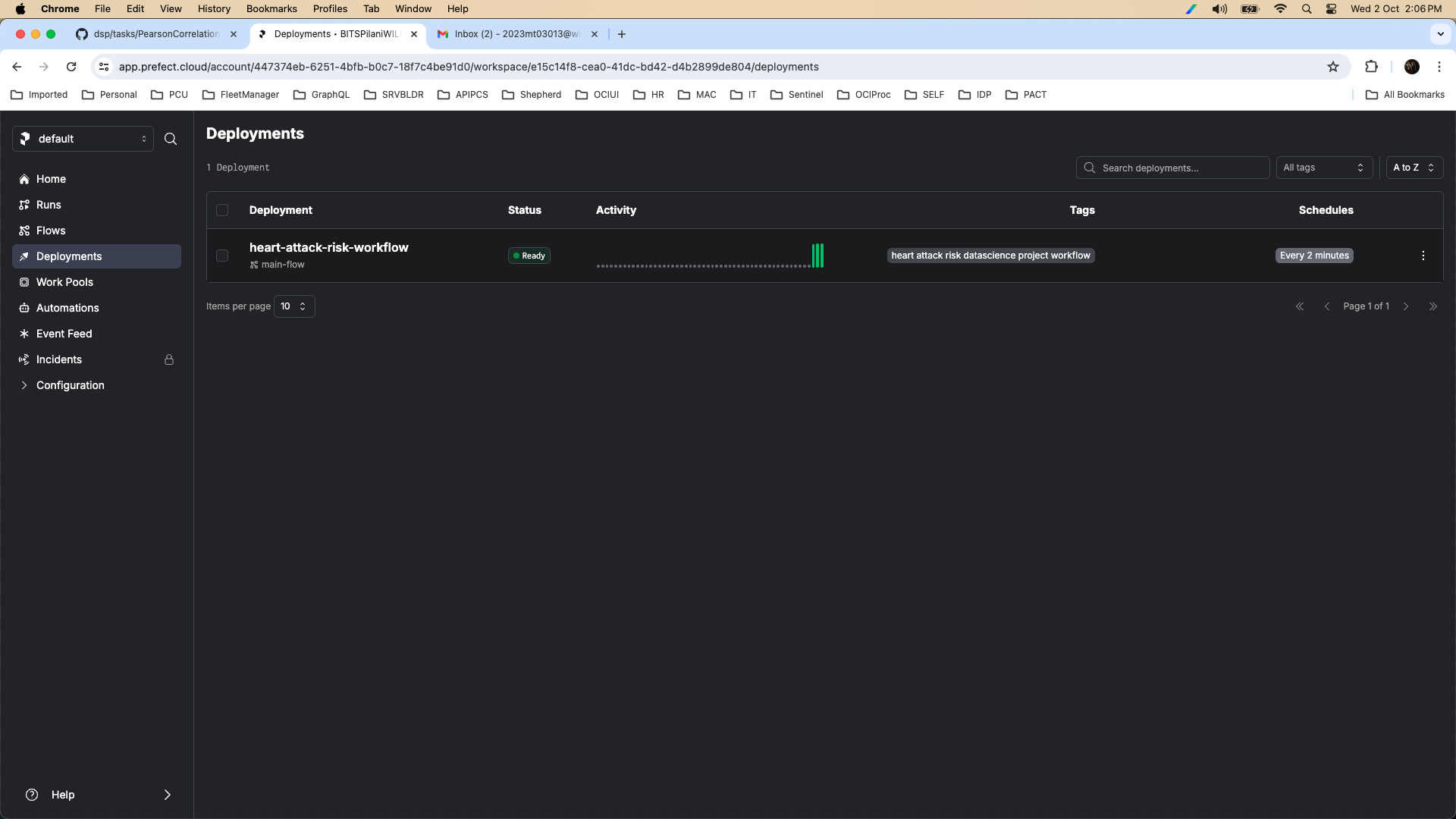
14:16:22.162 | INFO | Flow run 'tricky-elephant' - Finished in state Completed()

14:16:24.815 | INFO | prefect.flow\_runs.runner - Process for flow run 'tricky-elephant' exited cleanly.

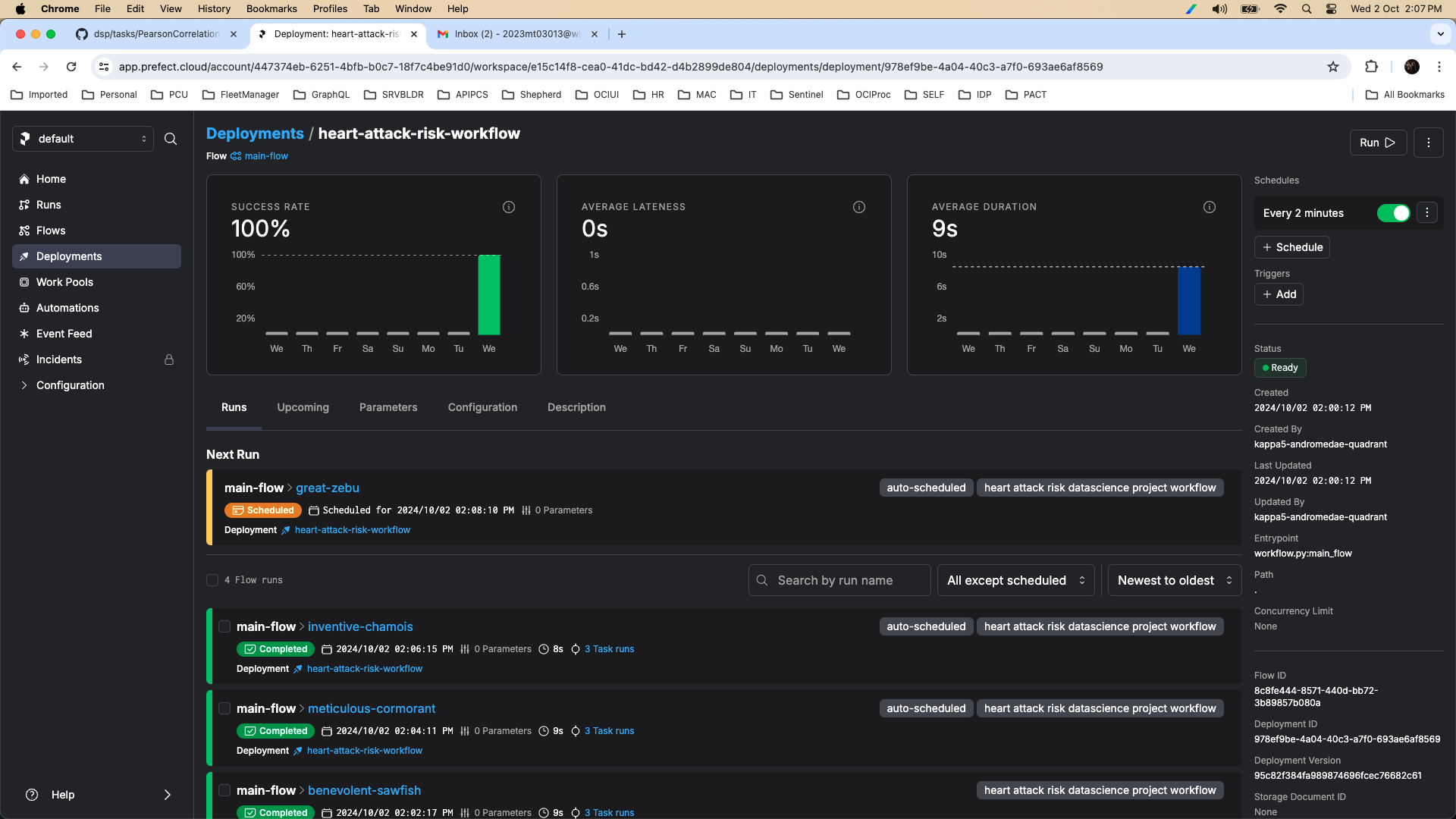
==

Deployment in Cloud

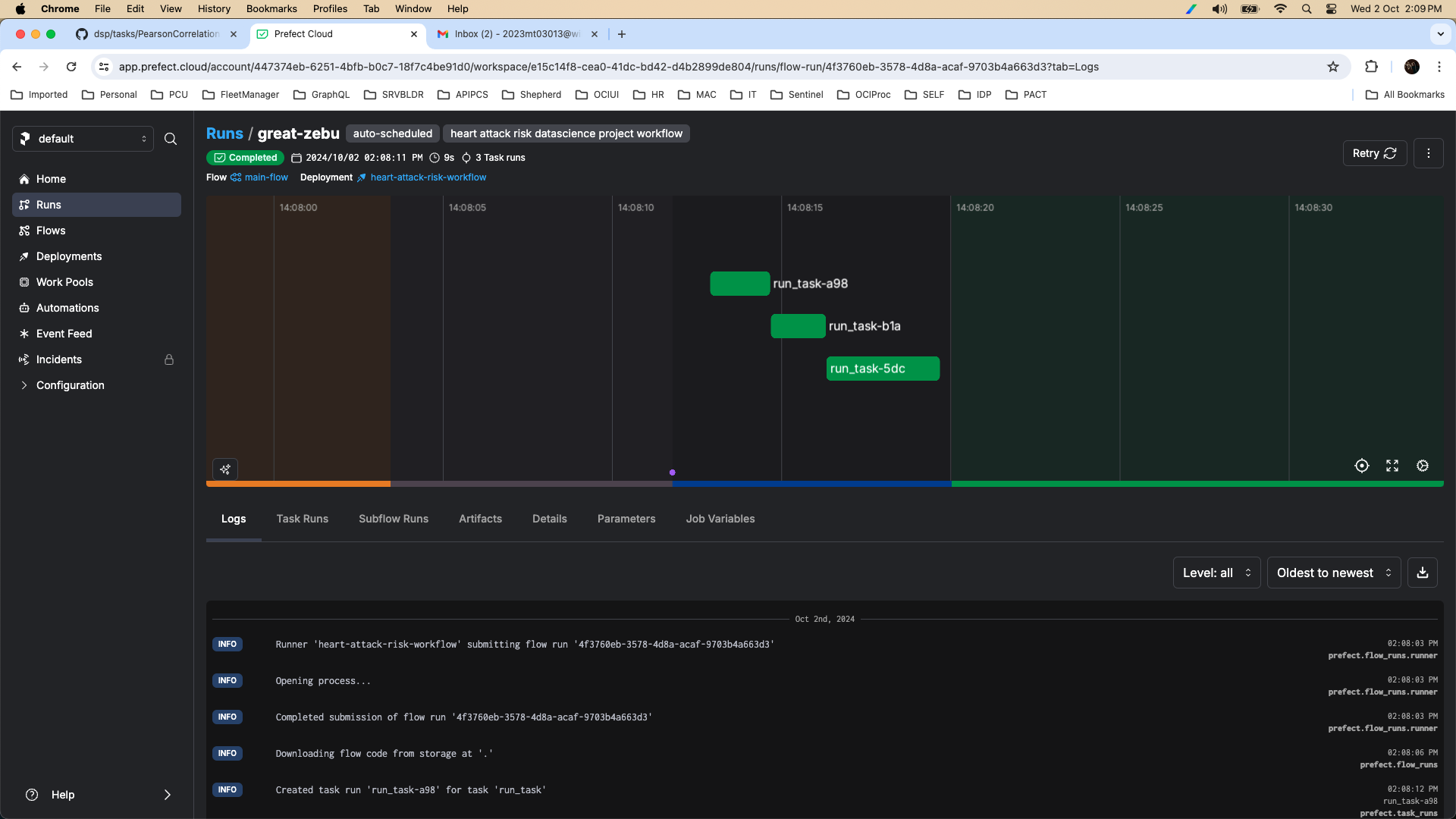
<https://app.prefect.cloud/account/447374eb-6251-4bfb-b0c7-18f7c4be91d0/workspace/e15c14f8-cea0-41dc-bd42-d4b2899de804/deployments/deployment/978ef9be-4a04-40c3-a7f0-693ae6af8569>

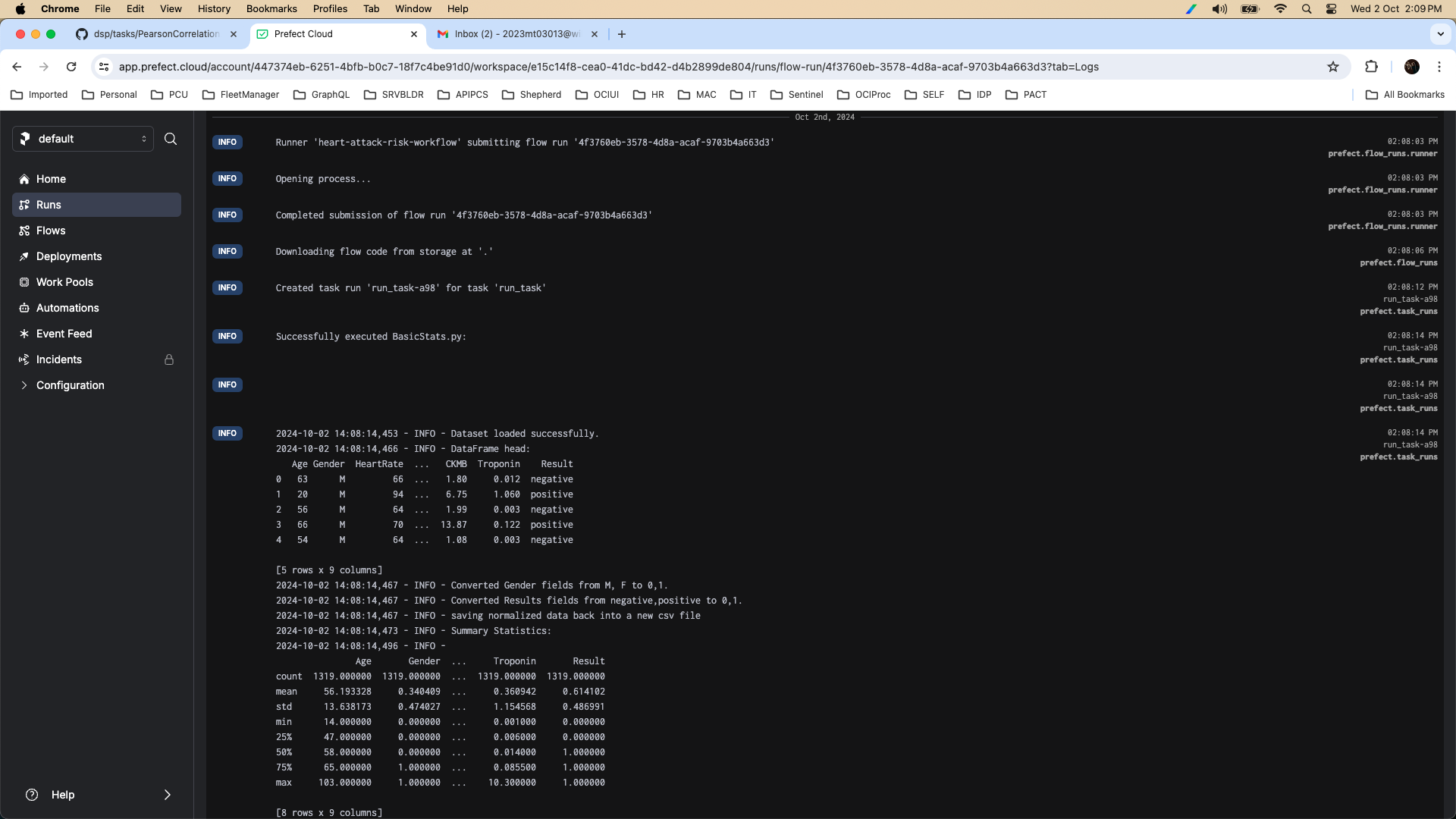


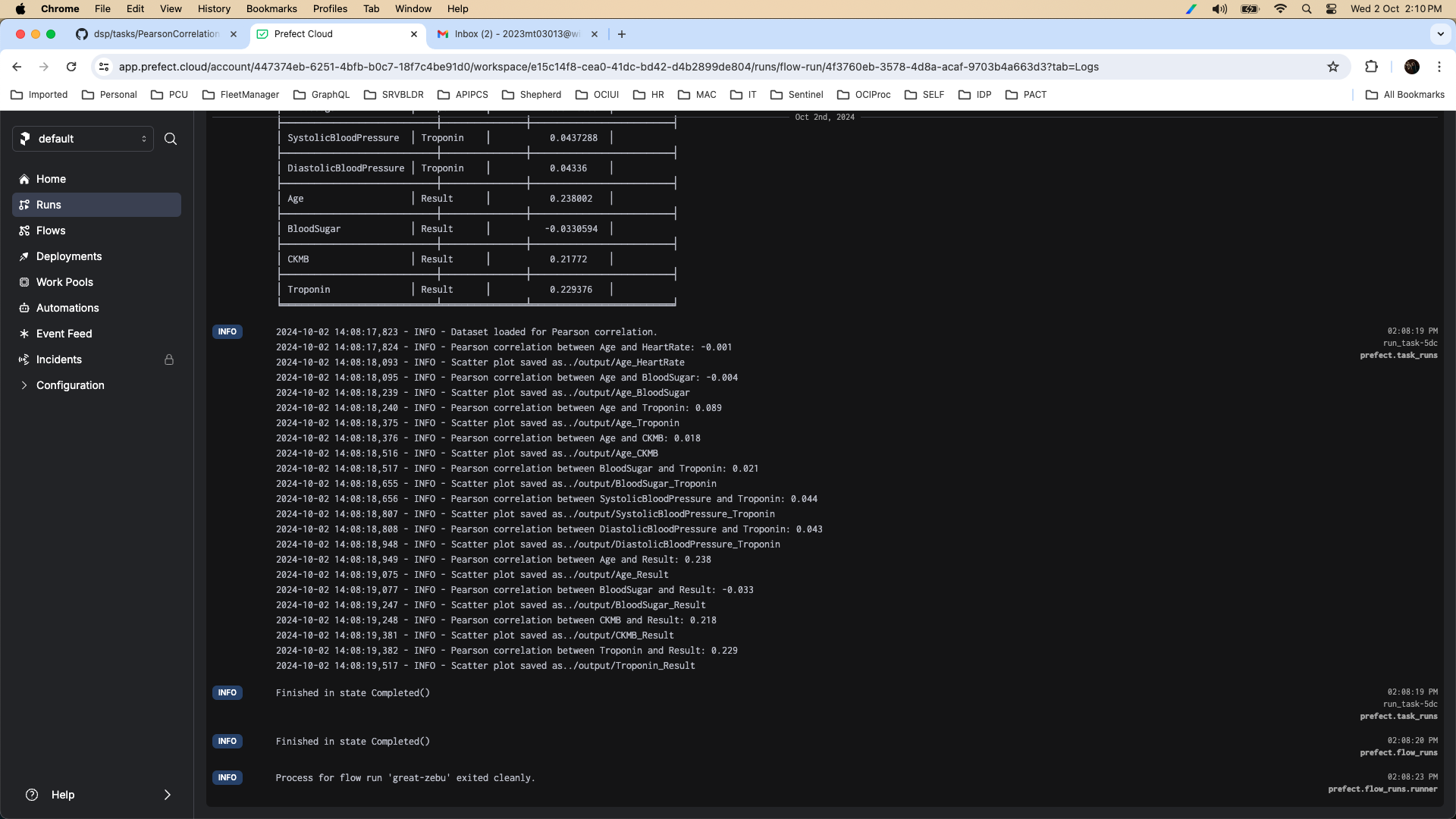
Flow Runs

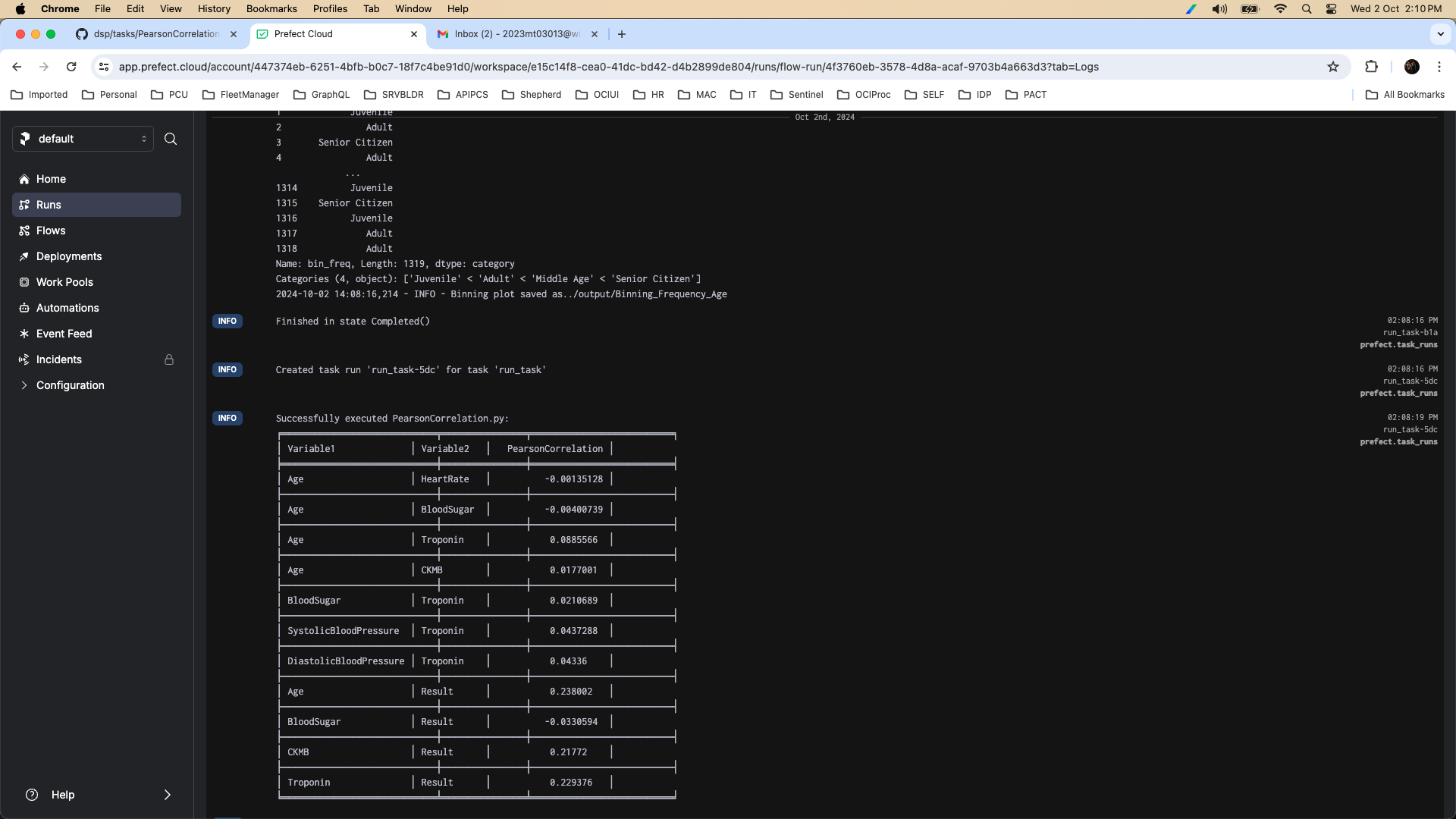


Run Logs

****

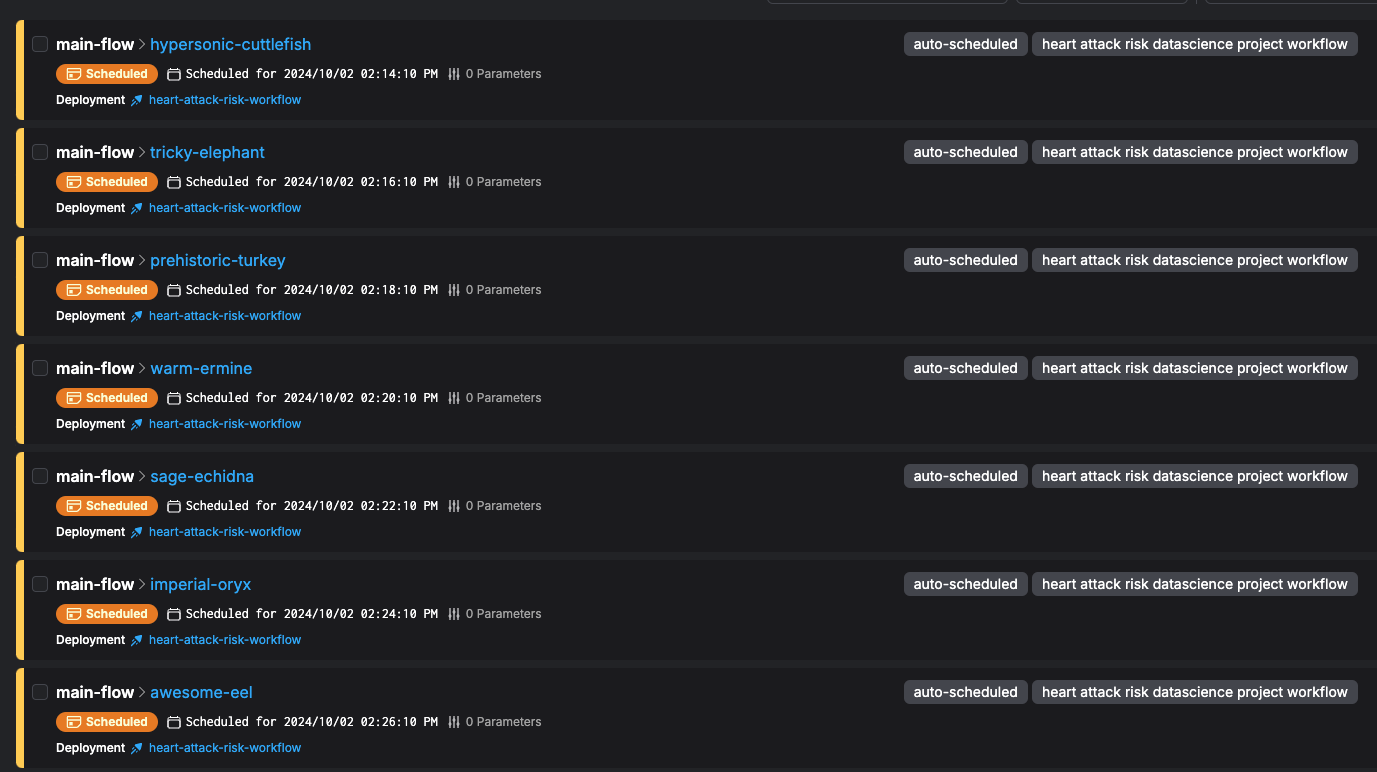
****

****

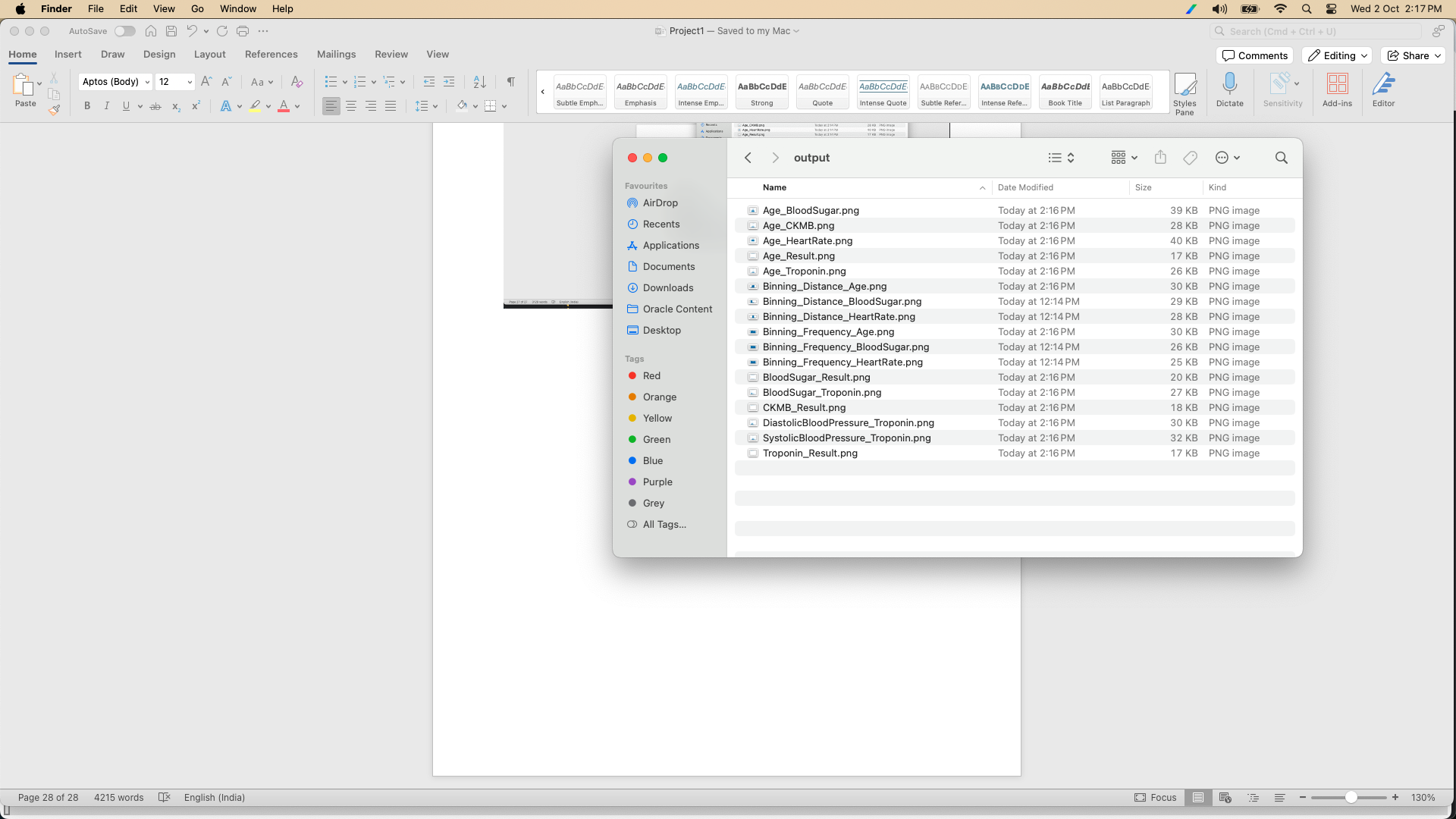
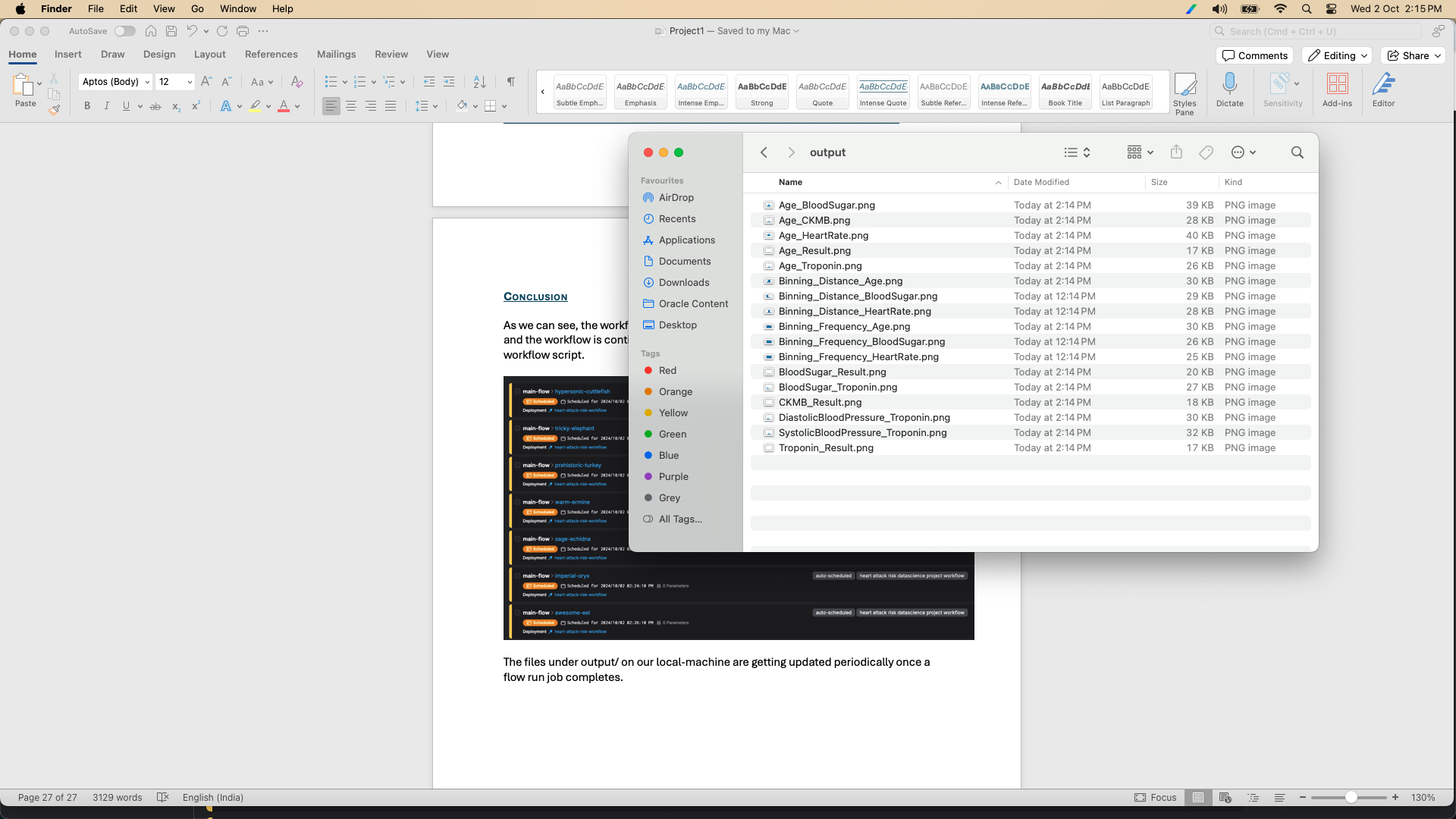
****

Conclusion

As we can see, the workflow ran all 3 tasks (BasicStats, Binning, PearsonCorrelation) and the workflow is continuously getting invoked for every 2mins as we configured in the workflow script.



The files under output/ on our local-machine are getting updated periodically once a flow run job completes.



Problem Statement – 2

**Design and Development of a Machine Learning Pipeline**

2.1. Model Preparation: Identify suitable machine learning algorithms for solving the business problem based on the dataset. Select any two algorithms

In our case, we are trying to build a model that can accurately predict risk of a heart-attack based on data-set.

So, our problem falls under Classification Problem, specifically a Binary Classification Problem.

We are selecting following two supervised machine learning algorithms for the same:

1. Random Forest Classifier
2. Logistic Regression

Logistic Regression is a straightforward and interpretable binary classifier, while Random Forest is an ensemble model built from multiple decision trees.

Both have their respective advantages and limitations, which we will explore in detail in the following sections.

2.2 Model Training: Split the dataset into training (70%) and testing (30%) sets and train the models.

2.2.1 Pre-Requisites

1. Install mlflow and psutil python libraries in the local machine.
2. Run ‘mlflow ui --host 0.0.0.0 --port 5000’ in one terminal and keep it alive

2.2.2 Random Forest Classifier

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score, classification\_report, precision\_score, recall\_score, f1\_score, confusion\_matrix

import mlflow

import mlflow.sklearn

import psutil

import time

# Set the MLflow tracking URI to 'http'

mlflow.set\_tracking\_uri("http://localhost:5000")

# Function for data preprocessing

def preprocess\_data(data):

pd.set\_option('future.no\_silent\_downcasting', True)

data['Gender'] = data['Gender'].replace({'M': 0, 'F': 1}).astype(int)

data['Result'] = data['Result'].replace({'negative': 0, 'positive': 1}).astype(int)

# Convert categorical variables to one-hot encoding

data = pd.get\_dummies(data, columns=['Gender'])

# Split data into X (features) and y (target)

X = data.drop('Result', axis=1)

y = data['Result']

return X, y

# Function for training the model

def train\_model(X\_train, y\_train, max\_depth=3, n\_estimators=100):

# Initialize the classifier

clf = RandomForestClassifier(max\_depth=max\_depth, n\_estimators=n\_estimators, random\_state=42)

# Train the model

clf.fit(X\_train, y\_train)

return clf

# Function to evaluate the model

def evaluate\_model(model, X\_test, y\_test):

# Make predictions

y\_pred = model.predict(X\_test)

# Evaluate accuracy

accuracy = accuracy\_score(y\_test, y\_pred)

print(f"Accuracy: {accuracy:.2f}")

# Display classification report

print("Classification Report:")

print(classification\_report(y\_test, y\_pred))

# Function to log model and system metrics to MLflow

def log\_to\_mlflow(model, X\_train, X\_test, y\_train, y\_test):

with mlflow.start\_run():

# Log hyper parameters using in Random Forest Algorithm

mlflow.log\_param("max\_depth", model.max\_depth)

mlflow.log\_param("n\_estimators", model.n\_estimators)

# Log model metrics

y\_pred = model.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

precision = precision\_score(y\_test, y\_pred, average='micro')

recall = recall\_score(y\_test, y\_pred, average='micro')

f1 = f1\_score(y\_test, y\_pred, average='micro')

confusion = confusion\_matrix(y\_test, y\_pred)

mlflow.log\_metric("accuracy", accuracy)

mlflow.log\_metric("precision", precision)

mlflow.log\_metric("recall", recall)

mlflow.log\_metric("f1-score", f1)

# Log confusion matrix

confusion\_dict = {

"true\_positive": confusion[1][1],

"false\_positive": confusion[0][1],

"true\_negative": confusion[0][0],

"false\_negative": confusion[1][0]

}

mlflow.log\_metrics(confusion\_dict)

# Log system metrics

# Example: CPU and Memory Usage

cpu\_usage = psutil.cpu\_percent(interval=1)

memory\_usage = psutil.virtual\_memory().percent

mlflow.log\_metric("system\_cpu\_usage", cpu\_usage)

mlflow.log\_metric("system\_memory\_usage", memory\_usage)

# Log execution time for training the model

execution\_time = {} # Dictionary to store execution times

# Example: Execution time for training the model

start\_time = time.time()

model = train\_model(X\_train, y\_train)

end\_time = time.time()

execution\_time["system\_model\_training"] = end\_time - start\_time

# Log execution time

mlflow.log\_metrics(execution\_time)

# Evaluate model and log metrics

evaluate\_model(model, X\_test, y\_test)

# Log model

mlflow.sklearn.log\_model(model, "model")

# Main function

def main():

# Load the dataset

data = pd.read\_csv("../data/MedicalDatasetForModel.csv")

# Preprocess the data

X, y = preprocess\_data(data)

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Train the model

model = train\_model(X\_train, y\_train)

# Evaluate and log to MLflow

log\_to\_mlflow(model, X\_train, X\_test, y\_train, y\_test)

if \_\_name\_\_ == "\_\_main\_\_":

main()

Running randomForest model locally

(apip\_project\_venv) adimulamramkumar-mac:code adimulamramkumar$ python RandomForestClassifier.py

Accuracy: 0.98

Classification Report:

precision recall f1-score support

0 0.97 0.97 0.97 155

1 0.98 0.98 0.98 241

accuracy 0.98 396

macro avg 0.98 0.98 0.98 396

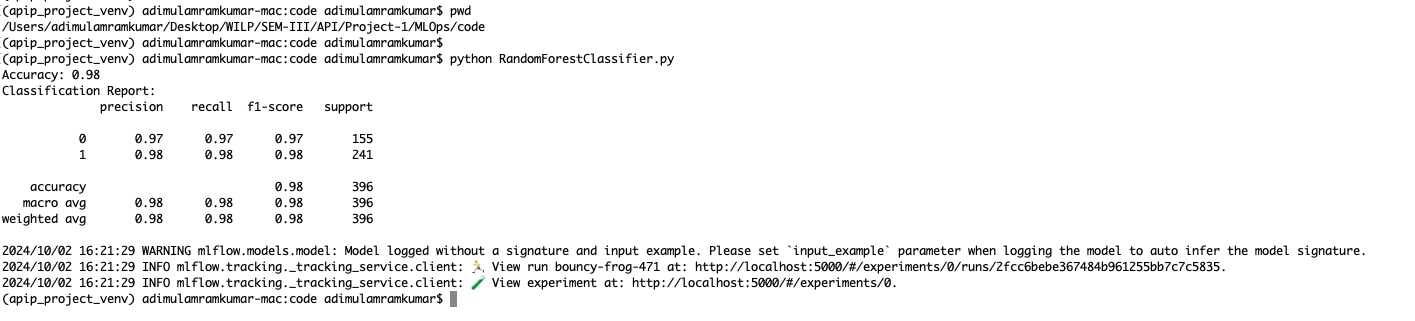
weighted avg 0.98 0.98 0.98 396

2024/10/02 16:21:29 WARNING mlflow.models.model: Model logged without a signature and input example. Please set `input\_example` parameter when logging the model to auto infer the model signature.

2024/10/02 16:21:29 INFO mlflow.tracking.\_tracking\_service.client: 🏃 View run bouncy-frog-471 at: http://localhost:5000/#/experiments/0/runs/2fcc6bebe367484b961255bb7c7c5835.

2024/10/02 16:21:29 INFO mlflow.tracking.\_tracking\_service.client: 🧪 View experiment at: http://localhost:5000/#/experiments/0.

(apip\_project\_venv) adimulamramkumar-mac:code adimulamramkumar$



2.2.3 Logistic Regression Classification

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score, classification\_report, precision\_score, recall\_score, f1\_score, confusion\_matrix

import mlflow

import mlflow.sklearn

import psutil

import time

# Set the MLflow tracking URI to 'http'

mlflow.set\_tracking\_uri("http://localhost:5000")

# Function for data preprocessing

def preprocess\_data(data):

pd.set\_option('future.no\_silent\_downcasting', True)

data['Gender'] = data['Gender'].replace({'M': 0, 'F': 1}).astype(int)

data['Result'] = data['Result'].replace({'negative': 0, 'positive': 1}).astype(int)

# Convert categorical variables to one-hot encoding

data = pd.get\_dummies(data, columns=['Gender'])

# Split data into X (features) and y (target)

X = data.drop('Result', axis=1)

y = data['Result']

return X, y

# Function for training the model

def train\_model(X\_train, y\_train):

# Initialize the classifier

clf = LogisticRegression(max\_iter=500, solver='liblinear')

# Train the model

clf.fit(X\_train, y\_train)

return clf

# Function to evaluate the model

def evaluate\_model(model, X\_test, y\_test):

# Make predictions

y\_pred = model.predict(X\_test)

# Evaluate accuracy

accuracy = accuracy\_score(y\_test, y\_pred)

print(f"Accuracy: {accuracy:.2f}")

# Display classification report

print("Classification Report:")

print(classification\_report(y\_test, y\_pred))

# Function to log model and system metrics to MLflow

def log\_to\_mlflow(model, X\_train, X\_test, y\_train, y\_test):

with mlflow.start\_run():

# Log hyper parameters using in Random Forest Algorithm

mlflow.log\_param("max\_iterations", model.max\_iter)

# Log model metrics

y\_pred = model.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

precision = precision\_score(y\_test, y\_pred, average='micro')

recall = recall\_score(y\_test, y\_pred, average='micro')

f1 = f1\_score(y\_test, y\_pred, average='micro')

confusion = confusion\_matrix(y\_test, y\_pred)

mlflow.log\_metric("accuracy", accuracy)

mlflow.log\_metric("precision", precision)

mlflow.log\_metric("recall", recall)

mlflow.log\_metric("f1-score", f1)

# Log confusion matrix

confusion\_dict = {

"true\_positive": confusion[1][1],

"false\_positive": confusion[0][1],

"true\_negative": confusion[0][0],

"false\_negative": confusion[1][0]

}

mlflow.log\_metrics(confusion\_dict)

# Log system metrics

# Example: CPU and Memory Usage

cpu\_usage = psutil.cpu\_percent(interval=1)

memory\_usage = psutil.virtual\_memory().percent

mlflow.log\_metric("system\_cpu\_usage", cpu\_usage)

mlflow.log\_metric("system\_memory\_usage", memory\_usage)

# Log execution time for training the model

execution\_time = {} # Dictionary to store execution times for different stages

# Example: Execution time for training the model

start\_time = time.time()

model = train\_model(X\_train, y\_train)

end\_time = time.time()

execution\_time["system\_model\_training"] = end\_time - start\_time

# Log execution time

mlflow.log\_metrics(execution\_time)

# Evaluate model and log metrics

evaluate\_model(model, X\_test, y\_test)

# Log model

mlflow.sklearn.log\_model(model, "model")

# Main function

def main():

# Load the dataset

data = pd.read\_csv("../data/MedicalDatasetForModel.csv")

# Preprocess the data

X, y = preprocess\_data(data)

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Train the model

model = train\_model(X\_train, y\_train)

# Evaluate and log to MLflow

log\_to\_mlflow(model, X\_train, X\_test, y\_train, y\_test)

if \_\_name\_\_ == "\_\_main\_\_":

main()

Running LogisticRegression model locally

(apip\_project\_venv) adimulamramkumar-mac:code adimulamramkumar$ python LogisticRegression.py

Accuracy: 0.79

Classification Report:

precision recall f1-score support

0 0.74 0.69 0.72 155

1 0.81 0.85 0.83 241

accuracy 0.79 396

macro avg 0.78 0.77 0.77 396

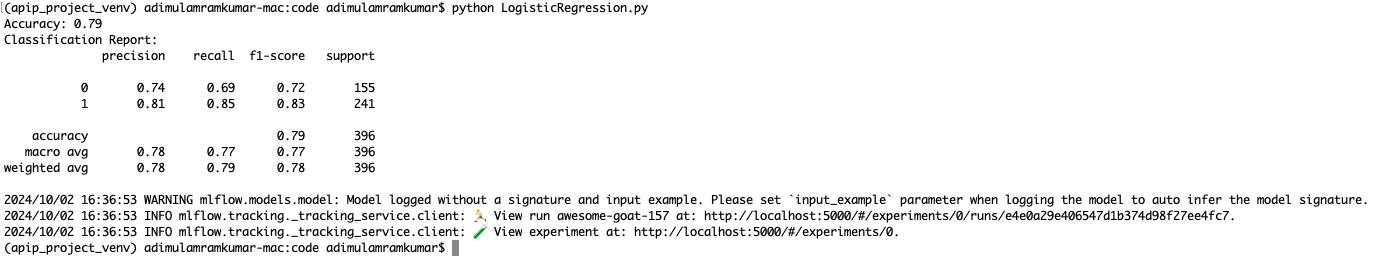
weighted avg 0.78 0.79 0.78 396

2024/10/02 16:36:53 WARNING mlflow.models.model: Model logged without a signature and input example. Please set `input\_example` parameter when logging the model to auto infer the model signature.

2024/10/02 16:36:53 INFO mlflow.tracking.\_tracking\_service.client: 🏃 View run awesome-goat-157 at: http://localhost:5000/#/experiments/0/runs/e4e0a29e406547d1b374d98f27ee4fc7.

2024/10/02 16:36:53 INFO mlflow.tracking.\_tracking\_service.client: 🧪 View experiment at: http://localhost:5000/#/experiments/0.

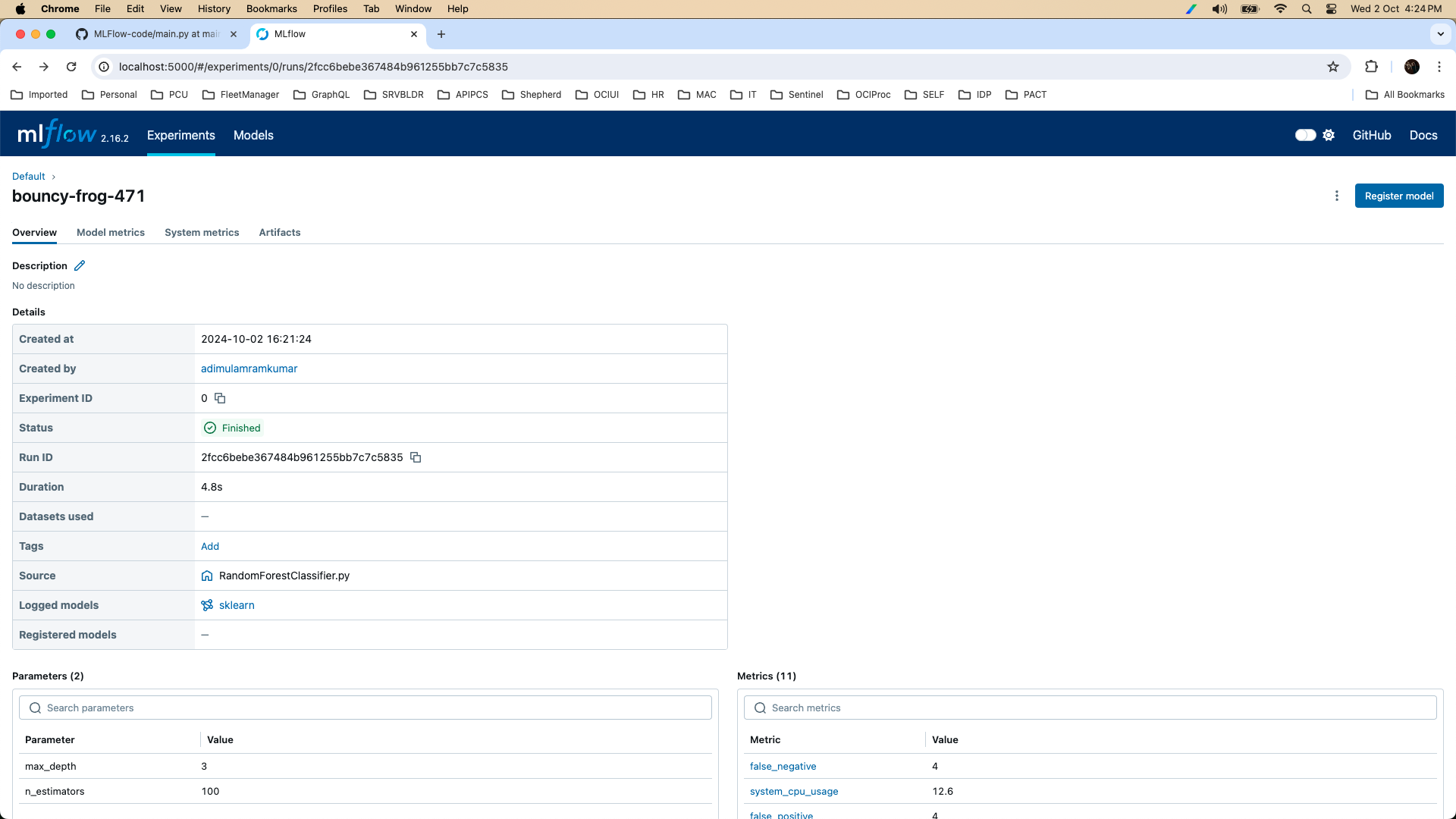
(apip\_project\_venv) adimulamramkumar-mac:code adimulamramkumar$



2.3 Model Evaluation: Evaluate the models using at least one metric (e.g., accuracy for classification models).

2.3.1 RandomForest

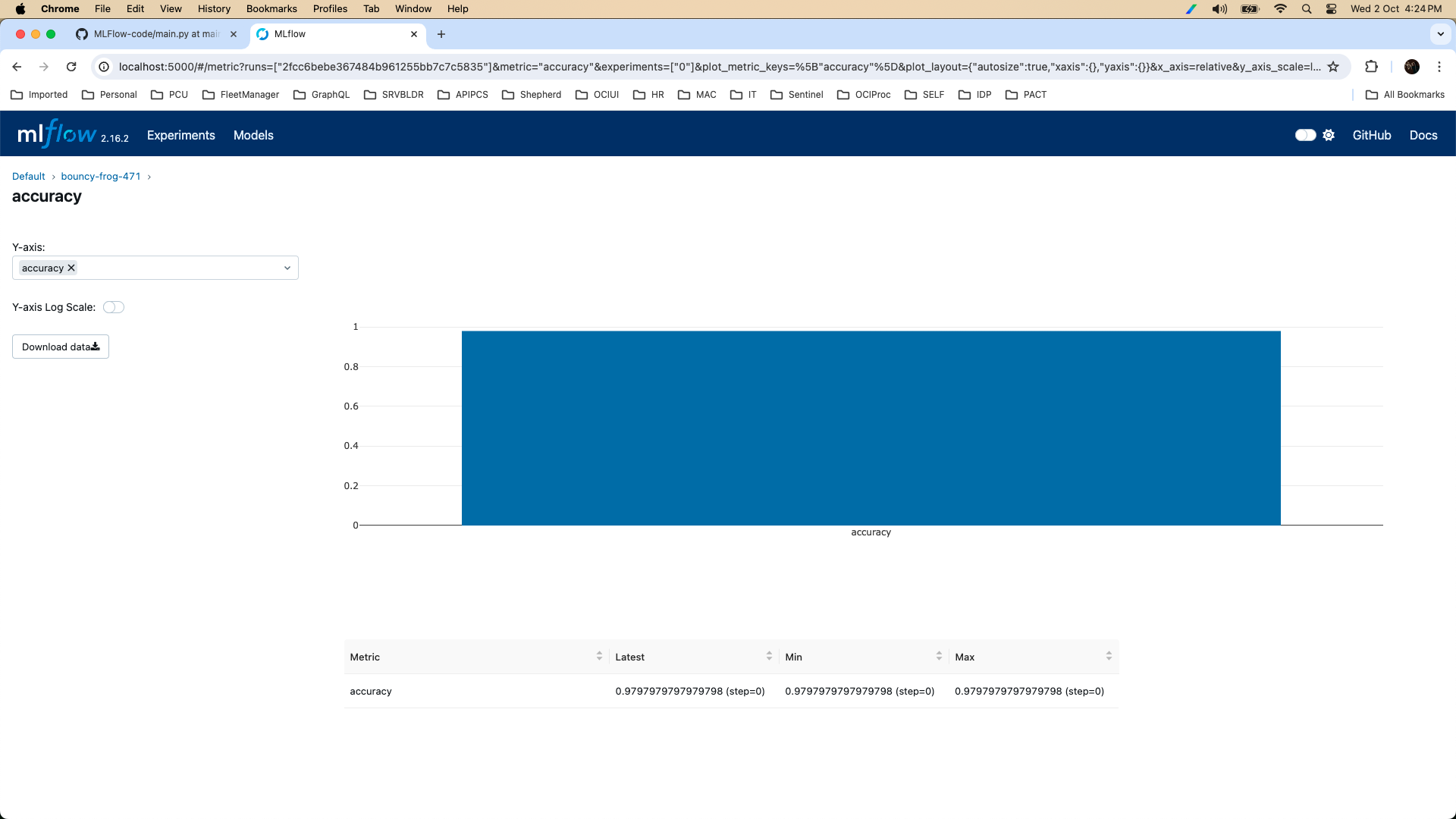
**RandomForest Model Run Basic Details:**



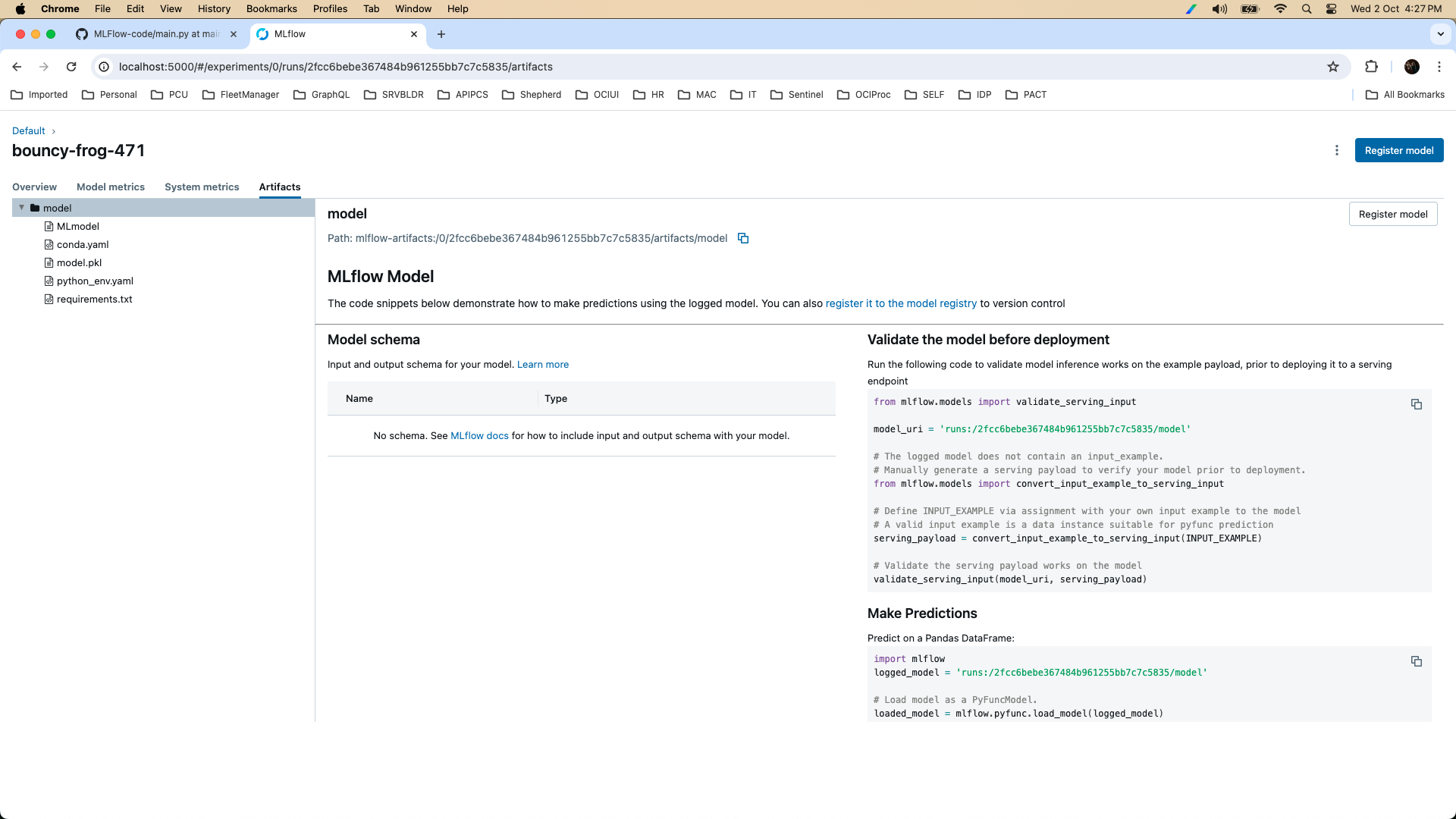
**RandomForest Model Parameters and Metrics:**



**RandomForest Model Accuracy:**

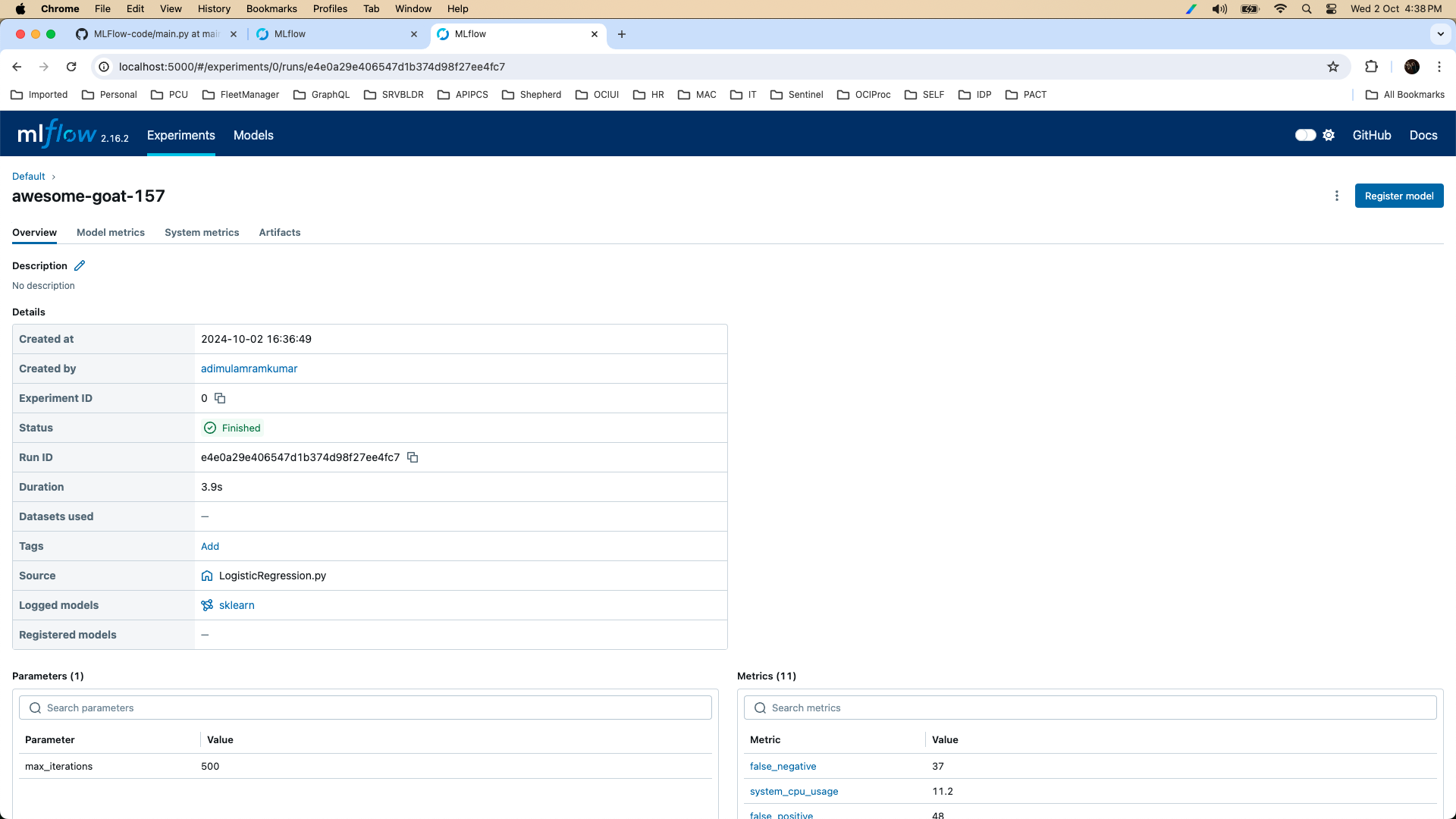


**RandomForest Model Artifacts:**

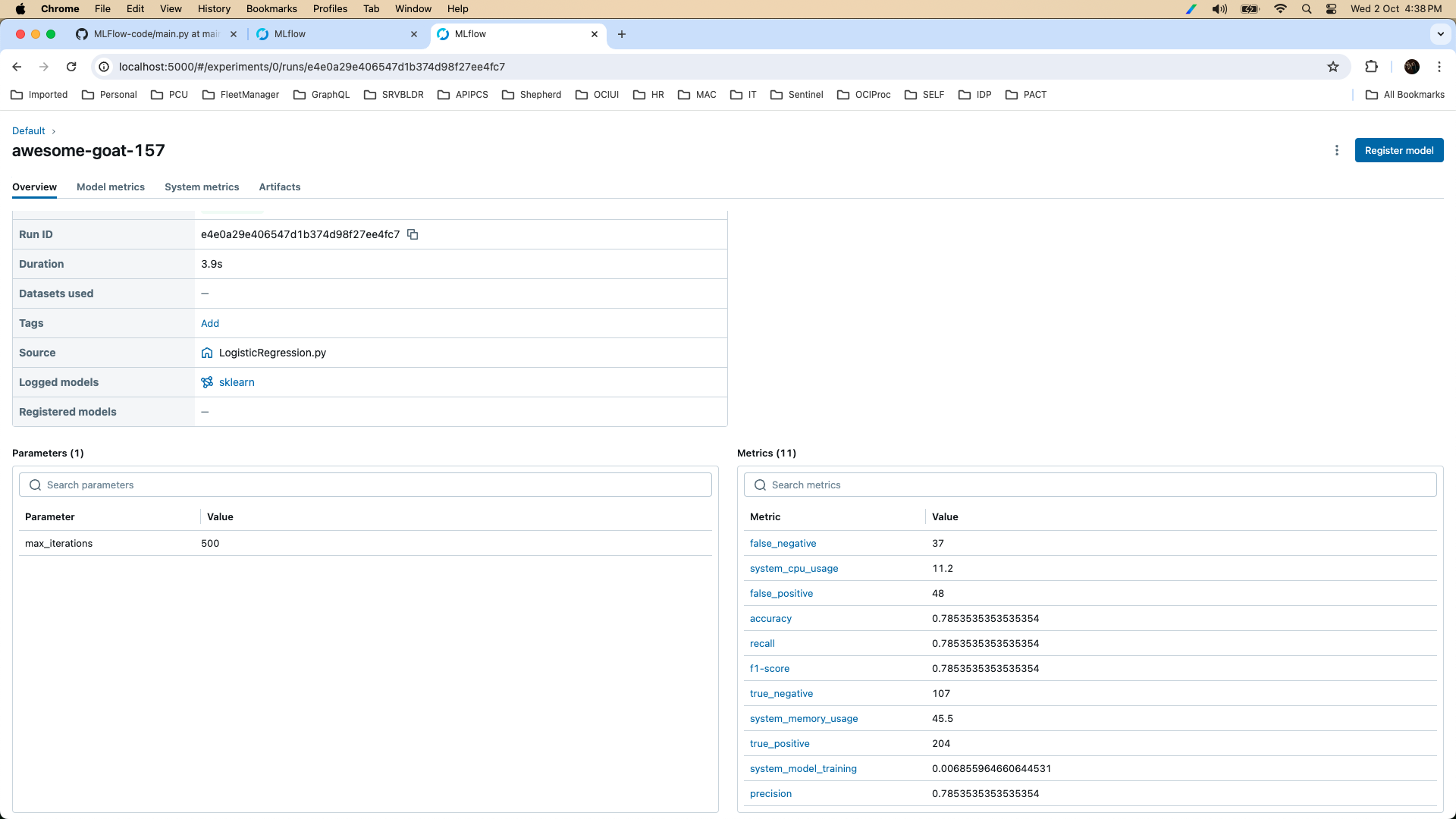
****

2.3.2 LogisticRegression

**Logistic Regression Model Run Basic Details:**



**Logistic Regression Model Parameters and Metrics:**



**Logistic Regression Model Accuracy:**

****

2.3.3 RandomForest vs LogisticRegression Accuracy

Random Forest

Accuracy: 0.98

Classification Report:

precision recall f1-score support

0 0.97 0.97 0.97 155

1 0.98 0.98 0.98 241

accuracy 0.98 396

macro avg 0.98 0.98 0.98 396

weighted avg 0.98 0.98 0.98 396

Logistic Regression

Accuracy: 0.79

Classification Report:

precision recall f1-score support

0 0.74 0.69 0.72 155

1 0.81 0.85 0.83 241

accuracy 0.79 396

macro avg 0.78 0.77 0.77 396

weighted avg 0.78 0.79 0.78 396

Analysis

1. **Random Forest**:
   * **Accuracy**: 0.98
   * **Precision, Recall, F1-score**: All are high (0.97-0.98), indicating strong predictive performance for both classes (0 = no heart attack, 1 = heart attack).
2. **Logistic Regression**:
   * **Accuracy**: 0.79
   * Precision, recall, and F1-score for class 0 (no heart attack) are lower compared to Random Forest, particularly with a recall of 0.69. Class 1 (heart attack) performs better but still not as well as Random Forest, with an F1-score of 0.83.

This suggests that Random Forest captures the patterns in the data better, likely due to its ability to handle complex interactions between features, while Logistic Regression may be more constrained by its linear nature.

2.4 MLOps: Monitor the model and log relevant metrics (at least four, such as accuracy, precision, recall, F1 score, etc.).

As seen in 2.2, 2.3, We have already logged metrics and shown them in mlflow-UI.

Random Forest

Accuracy: 0.98

Classification Report:

precision recall f1-score support

0 0.97 0.97 0.97 155

1 0.98 0.98 0.98 241

accuracy 0.98 396

macro avg 0.98 0.98 0.98 396

weighted avg 0.98 0.98 0.98 396

Logistic Regression

Accuracy: 0.79

Classification Report:

precision recall f1-score support

0 0.74 0.69 0.72 155

1 0.81 0.85 0.83 241

accuracy 0.79 396

macro avg 0.78 0.77 0.77 396

weighted avg 0.78 0.79 0.78 396

Problem Statement – 3

**API ACCESS**

3.1 Retrieve Key Application Details: Use Built-in APIs to access important application information (e.g., flow, deployment etc.)

3.1.1 Deployments

import requests

PREFECT\_API\_KEY="pnu\_4t3M9YNXs6TytZ5PNmzIM2gcicvXQL16BZBG"

ACCOUNT\_ID="447374eb-6251-4bfb-b0c7-18f7c4be91d0"

WORKSPACE\_ID="e15c14f8-cea0-41dc-bd42-d4b2899de804"

DEPLOYMENT\_ID="22e10c79-4d9b-4e0e-926c-f9979c089ea4"

PREFECT\_API\_URL = f"https://api.prefect.cloud/api/accounts/{ACCOUNT\_ID}/workspaces/{WORKSPACE\_ID}/deployments/{DEPLOYMENT\_ID}"

# Set up headers with Authorization

headers = {"Authorization": f"Bearer {PREFECT\_API\_KEY}"}

# Make the request using GET

response = requests.get(PREFECT\_API\_URL, headers=headers)

# Check the response status

if response.status\_code == 200:

deployment\_info = response.json()

print(deployment\_info)

else:

print(f"Error: Received status code {response.status\_code}")

print(f"Response content: {response.text}")

(apip\_project\_venv) adimulamramkumar-mac:api adimulamramkumar$ python deployment.py

{'id': '22e10c79-4d9b-4e0e-926c-f9979c089ea4', 'created': '2024-10-02T11:25:26.471884+00:00', 'updated': '2024-10-02T11:25:26.467224+00:00', 'infra\_overrides': {}, 'name': 'heart-attack-risk-workflow', 'version': '95c82f384fa989874696fcec76682c61', 'description': 'None', 'flow\_id': '10f32598-b16d-48da-8916-7537741c5a35', 'schedule': {'interval': 120.0, 'anchor\_date': '2024-10-02T16:55:25.626469+05:30', 'timezone': 'Asia/Kolkata'}, 'is\_schedule\_active': True, 'paused': False, 'disabled': False, 'schedules': [{'id': '52c39eb8-80da-44b4-9fc7-dafcb93f0d82', 'created': '2024-10-02T11:25:26.492177+00:00', 'updated': '2024-10-02T12:23:36.303297+00:00', 'deployment\_id': '22e10c79-4d9b-4e0e-926c-f9979c089ea4', 'schedule': {'interval': 120.0, 'anchor\_date': '2024-10-02T16:55:25.626469+05:30', 'timezone': 'Asia/Kolkata'}, 'active': True, 'last\_scheduled\_at': '2024-10-02T12:23:36.329771+00:00', 'soonest\_scheduled\_run': '2024-10-02T12:25:25.626469+00:00', 'latest\_scheduled\_run': '2024-10-02T14:25:25.626469+00:00', 'max\_active\_runs': None, 'max\_scheduled\_runs': None, 'catchup': False}], 'concurrency\_limit': None, 'global\_concurrency\_limit': None, 'concurrency\_options': None, 'job\_variables': {}, 'parameters': {}, 'tags': ['heart attack risk datascience project workflow'], 'labels': None, 'work\_queue\_name': None, 'last\_polled': '2024-10-02T12:46:25.064445+00:00', 'parameter\_openapi\_schema': {'type': 'object', 'title': 'Parameters', 'properties': {}}, 'path': '.', 'pull\_steps': [], 'entrypoint': 'workflow.py:main\_flow', 'manifest\_path': None, 'storage\_document\_id': None, 'infrastructure\_document\_id': None, 'created\_by': {'id': 'ea639eed-d04f-46a7-ae65-4576fa087022', 'type': 'USER', 'display\_value': 'kappa5-andromedae-quadrant'}, 'updated\_by': {'id': 'ea639eed-d04f-46a7-ae65-4576fa087022', 'type': 'USER', 'display\_value': 'kappa5-andromedae-quadrant'}, 'work\_pool\_name': None, 'status': 'READY', 'enforce\_parameter\_schema': True}

(apip\_project\_venv) adimulamramkumar-mac:api adimulamramkumar$

Pretty Format:  
{

"id": "22e10c79-4d9b-4e0e-926c-f9979c089ea4",

"created": "2024-10-02T11:25:26.471884+00:00",

"updated": "2024-10-02T11:25:26.467224+00:00",

"infra\_overrides": {},

"name": "heart-attack-risk-workflow",

"version": "95c82f384fa989874696fcec76682c61",

"description": "None",

"flow\_id": "10f32598-b16d-48da-8916-7537741c5a35",

"schedule": {

"interval": 120,

"anchor\_date": "2024-10-02T16:55:25.626469+05:30",

"timezone": "Asia/Kolkata"

},

"is\_schedule\_active": "True",

"paused": "False",

"disabled": "False",

"schedules": [

{

"id": "52c39eb8-80da-44b4-9fc7-dafcb93f0d82",

"created": "2024-10-02T11:25:26.492177+00:00",

"updated": "2024-10-02T12:23:36.303297+00:00",

"deployment\_id": "22e10c79-4d9b-4e0e-926c-f9979c089ea4",

"schedule": {

"interval": 120,

"anchor\_date": "2024-10-02T16:55:25.626469+05:30",

"timezone": "Asia/Kolkata"

},

"active": "True",

"last\_scheduled\_at": "2024-10-02T12:23:36.329771+00:00",

"soonest\_scheduled\_run": "2024-10-02T12:25:25.626469+00:00",

"latest\_scheduled\_run": "2024-10-02T14:25:25.626469+00:00",

"max\_active\_runs": "None",

"max\_scheduled\_runs": "None",

"catchup": "False"

}

],

"concurrency\_limit": "None",

"global\_concurrency\_limit": "None",

"concurrency\_options": "None",

"job\_variables": {},

"parameters": {},

"tags": [

"heart attack risk datascience project workflow"

],

"labels": "None",

"work\_queue\_name": "None",

"last\_polled": "2024-10-02T12:46:25.064445+00:00",

"parameter\_openapi\_schema": {

"type": "object",

"title": "Parameters",

"properties": {}

},

"path": ".",

"pull\_steps": [],

"entrypoint": "workflow.py:main\_flow",

"manifest\_path": "None",

"storage\_document\_id": "None",

"infrastructure\_document\_id": "None",

"created\_by": {

"id": "ea639eed-d04f-46a7-ae65-4576fa087022",

"type": "USER",

"display\_value": "kappa5-andromedae-quadrant"

},

"updated\_by": {

"id": "ea639eed-d04f-46a7-ae65-4576fa087022",

"type": "USER",

"display\_value": "kappa5-andromedae-quadrant"

},

"work\_pool\_name": "None",

"status": "READY",

"enforce\_parameter\_schema": "True"

}

3.1.2 Flows

import requests

PREFECT\_API\_KEY="pnu\_4t3M9YNXs6TytZ5PNmzIM2gcicvXQL16BZBG"

ACCOUNT\_ID="447374eb-6251-4bfb-b0c7-18f7c4be91d0"

WORKSPACE\_ID="e15c14f8-cea0-41dc-bd42-d4b2899de804"

DEPLOYMENT\_ID="22e10c79-4d9b-4e0e-926c-f9979c089ea4"

FLOW\_ID="10f32598-b16d-48da-8916-7537741c5a35"

# Correct API URL to get flow details

PREFECT\_API\_URL = f"https://api.prefect.cloud/api/accounts/{ACCOUNT\_ID}/workspaces/{WORKSPACE\_ID}/flows/{FLOW\_ID}"

# Set up headers with Authorization

headers = {"Authorization": f"Bearer {PREFECT\_API\_KEY}"}

# Make the request using GET

response = requests.get(PREFECT\_API\_URL, headers=headers)

# Check the response status

if response.status\_code == 200:

flow\_info = response.json()

print(flow\_info)

else:

print(f"Error: Received status code {response.status\_code}")

print(f"Response content: {response.text}")

(apip\_project\_venv) adimulamramkumar-mac:api adimulamramkumar$ python flows.py

{'id': '10f32598-b16d-48da-8916-7537741c5a35', 'created': '2024-10-02T11:25:26.075706+00:00', 'updated': '2024-10-02T11:25:26.075721+00:00', 'name': 'main-flow', 'tags': [], 'labels': {}}

(apip\_project\_venv) adimulamramkumar-mac:api adimulamramkumar$

{

"id": "10f32598-b16d-48da-8916-7537741c5a35",

"created": "2024-10-02T11:25:26.075706+00:00",

"updated": "2024-10-02T11:25:26.075721+00:00",

"name": "main-flow",

"tags": [],

"labels": {}

}

3.1.3 Download Run Logs

import requests

PREFECT\_API\_KEY="pnu\_4t3M9YNXs6TytZ5PNmzIM2gcicvXQL16BZBG"

ACCOUNT\_ID="447374eb-6251-4bfb-b0c7-18f7c4be91d0"

WORKSPACE\_ID="e15c14f8-cea0-41dc-bd42-d4b2899de804"

FLOW\_RUN\_ID="701b7c9f-e6ca-4ffa-a3aa-1290359606d2"

# Correct API URL to get flow details

PREFECT\_API\_URL = f"https://api.prefect.cloud/api/accounts/{ACCOUNT\_ID}/workspaces/{WORKSPACE\_ID}/flow\_runs/{FLOW\_RUN\_ID}/logs/download"

# Set up headers with Authorization

headers = {"Authorization": f"Bearer {PREFECT\_API\_KEY}"}

# Make the request using GET

response = requests.get(PREFECT\_API\_URL, headers=headers)

# Check the response status

if response.status\_code == 200:

with open('/tmp/response\_data.txt', 'wb') as file:

file.write(response.content)

else:

print(f"Error: Received status code {response.status\_code}")

print(f"Response content: {response.text}")

(apip\_project\_venv) adimulamramkumar-mac:api adimulamramkumar$ python downloadLogs.py

(apip\_project\_venv) adimulamramkumar-mac:api adimulamramkumar$ echo $?

0

(apip\_project\_venv) adimulamramkumar-mac:api adimulamramkumar$ ls -la /tmp/response\_data.txt

-rw-r--r-- 1 adimulamramkumar wheel 14956 Oct 2 18:38 /tmp/response\_data.txt

(apip\_project\_venv) adimulamramkumar-mac:api adimulamramkumar$

(apip\_project\_venv) adimulamramkumar-mac:api adimulamramkumar$ cat /tmp/response\_data.txt | tail -10

2024-10-02 18:33:32,080 - INFO - Scatter plot saved as../output/Age\_Result

2024-10-02 18:33:32,081 - INFO - Pearson correlation between BloodSugar and Result: -0.033

2024-10-02 18:33:32,255 - INFO - Scatter plot saved as../output/BloodSugar\_Result

2024-10-02 18:33:32,257 - INFO - Pearson correlation between CKMB and Result: 0.218

2024-10-02 18:33:32,388 - INFO - Scatter plot saved as../output/CKMB\_Result

2024-10-02 18:33:32,389 - INFO - Pearson correlation between Troponin and Result: 0.229

2024-10-02 18:33:32,520 - INFO - Scatter plot saved as../output/Troponin\_Result"

2024-10-02 13:03:32.679554+00:00,20,701b7c9f-e6ca-4ffa-a3aa-1290359606d2,5cd1dabc-5230-4535-ad15-3f3a322ae445,Finished in state Completed()

2024-10-02 13:03:33.046698+00:00,20,701b7c9f-e6ca-4ffa-a3aa-1290359606d2,,Finished in state Completed()

2024-10-02 13:03:35.996999+00:00,20,701b7c9f-e6ca-4ffa-a3aa-1290359606d2,,Process for flow run 'amaranth-falcon' exited cleanly.

(apip\_project\_venv) adimulamramkumar-mac:api adimulamramkumar$

3.2 Display Application Details: Present at least four application details retrieved via APIs.

import requests

from datetime import datetime, timedelta

import pytz

PREFECT\_API\_KEY="pnu\_4t3M9YNXs6TytZ5PNmzIM2gcicvXQL16BZBG"

ACCOUNT\_ID="447374eb-6251-4bfb-b0c7-18f7c4be91d0"

WORKSPACE\_ID="e15c14f8-cea0-41dc-bd42-d4b2899de804"

DEPLOYMENT\_ID="22e10c79-4d9b-4e0e-926c-f9979c089ea4"

FLOW\_ID="10f32598-b16d-48da-8916-7537741c5a35"

headers = { "Authorization": f"Bearer {PREFECT\_API\_KEY}" }

PREFECT\_API\_URL = f"https://api.prefect.cloud/api/accounts/{ACCOUNT\_ID}/workspaces/{WORKSPACE\_ID}"

# endpoint URLs

DEPLOYMENT\_ENDPOINT = f'{PREFECT\_API\_URL}/deployments/filter'

FLOW\_ENDPOINT = f'{PREFECT\_API\_URL}/flows/filter'

FLOW\_RUN\_ENDPOINT = f'{PREFECT\_API\_URL}/flow\_runs/filter'

TASK\_RUN\_ENDPOINT=f'{PREFECT\_API\_URL}/task\_runs/filter'

WORK\_QUEUES\_ENDPOINT=f'{PREFECT\_API\_URL}/work\_queues/filter'

SCHEDULES\_ENDPOINT=f'{PREFECT\_API\_URL}/deployments/get\_scheduled\_flow\_runs'

CONCURRENCY\_LIMIT\_ENDPOINT=f'{PREFECT\_API\_URL}/concurrency\_limits/filter'

# Get deployments

deployments = requests.post(DEPLOYMENT\_ENDPOINT, headers=headers)

deployments\_json = deployments.json()

# Get flow details

flows = requests.post(FLOW\_ENDPOINT, headers=headers)

flows\_json = flows.json()

# Get flow run details

flow\_runs = requests.post(FLOW\_RUN\_ENDPOINT, headers=headers)

flow\_runs\_json = flow\_runs.json()

flow\_runs\_sorted\_json = sorted(flow\_runs\_json, key=lambda x: x['start\_time'] or '', reverse=True)

# Get task run details

task\_runs = requests.post(TASK\_RUN\_ENDPOINT, headers=headers)

task\_runs\_json = task\_runs.json()

task\_runs\_sorted\_json = sorted(task\_runs\_json, key=lambda x: x['start\_time'] or '', reverse=True)

# Get schedules

new\_time = datetime.utcnow() + timedelta(minutes=8)

formatted\_time = new\_time.strftime('%Y-%m-%dT%H:%M:%SZ')

payload = {

'deployment\_ids': [ DEPLOYMENT\_ID ],

'scheduled\_before': formatted\_time

}

schedules = requests.post(SCHEDULES\_ENDPOINT, headers=headers, json=payload)

schedules\_json = schedules.json()

schedules\_json\_sorted = sorted(schedules\_json, key=lambda x: x['created'] or '', reverse=True)

# concurrency limits

concurrency\_limits = requests.post(CONCURRENCY\_LIMIT\_ENDPOINT, headers=headers)

concurrency\_limits\_json = concurrency\_limits.json()

# Displaying application details

print("Deployments:")

for deployment in deployments\_json:

print(f" ID: {deployment['id']}, Name: {deployment['name']}")

print("\n")

print("Flows:")

for flow in flows\_json:

print(f" ID: {flow['id']}, Name: {flow['name']}")

print("\n")

print("Flow Runs:")

for flow\_run in flow\_runs\_sorted\_json[:3]:

print(f"- ID: {flow\_run['id']}, State: {flow\_run['state']}")

print("\n")

print("Task Runs:")

for task\_run in task\_runs\_sorted\_json[:3]:

#print(task\_run)

print(f"- ID: {task\_run['id']}, Task Name: {task\_run['name']}, State: {task\_run['state']}")

print("\n")

print("Schedules:")

for schedule in schedules\_json\_sorted[:3]:

utc\_time = datetime.fromisoformat(schedule['expected\_start\_time'])

ist\_timezone = pytz.timezone('Asia/Kolkata')

ist\_time = utc\_time.astimezone(ist\_timezone)

print(f" ID: {schedule['id']}, Name: {schedule['name']}, ScheduledAt: {ist\_time}")

#print(schedule)

print("\n")

print("Concurrency Limit:")

print(concurrency\_limits\_json)

print("\n")

(apip\_project\_venv) adimulamramkumar-mac:api adimulamramkumar$ python application.py

Deployments:

ID: 22e10c79-4d9b-4e0e-926c-f9979c089ea4, Name: heart-attack-risk-workflow

Flows:

ID: 10f32598-b16d-48da-8916-7537741c5a35, Name: main-flow

Flow Runs:

- ID: bdc5d99c-e4d9-4120-92ee-b3e55fcad2cf, State: {'id': '5c807c7b-1f5c-4a1d-8d1e-7a51da9c6adb', 'type': 'COMPLETED', 'name': 'Completed', 'timestamp': '2024-10-02T16:35:35.192744+00:00', 'message': '', 'data': None, 'state\_details': {'flow\_run\_id': 'bdc5d99c-e4d9-4120-92ee-b3e55fcad2cf', 'task\_run\_id': None, 'child\_flow\_run\_id': None, 'scheduled\_time': None, 'cache\_key': None, 'cache\_expiration': None, 'deferred': None, 'untrackable\_result': True, 'pause\_timeout': None, 'pause\_reschedule': False, 'pause\_key': None, 'run\_input\_keyset': None, 'refresh\_cache': None, 'retriable': None, 'transition\_id': 'f5166882-d31f-4667-b5ef-45e42f126db1', 'task\_parameters\_id': None}}

- ID: 7fd2fedb-80c9-4ee0-ac39-e0b6de40ed80, State: {'id': '2ef5b109-767a-4963-9ffa-6534a0efb207', 'type': 'COMPLETED', 'name': 'Completed', 'timestamp': '2024-10-02T16:33:34.857689+00:00', 'message': '', 'data': None, 'state\_details': {'flow\_run\_id': '7fd2fedb-80c9-4ee0-ac39-e0b6de40ed80', 'task\_run\_id': None, 'child\_flow\_run\_id': None, 'scheduled\_time': None, 'cache\_key': None, 'cache\_expiration': None, 'deferred': None, 'untrackable\_result': True, 'pause\_timeout': None, 'pause\_reschedule': False, 'pause\_key': None, 'run\_input\_keyset': None, 'refresh\_cache': None, 'retriable': None, 'transition\_id': '4b9152d0-5f95-4a9c-a0b4-df88b74c2e33', 'task\_parameters\_id': None}}

- ID: c25c1630-bea5-4cc1-9476-4d9ef9423c38, State: {'id': '0a3afb72-1059-4912-bbae-558bd7bf2a90', 'type': 'COMPLETED', 'name': 'Completed', 'timestamp': '2024-10-02T16:31:41.504575+00:00', 'message': '', 'data': None, 'state\_details': {'flow\_run\_id': 'c25c1630-bea5-4cc1-9476-4d9ef9423c38', 'task\_run\_id': None, 'child\_flow\_run\_id': None, 'scheduled\_time': None, 'cache\_key': None, 'cache\_expiration': None, 'deferred': None, 'untrackable\_result': True, 'pause\_timeout': None, 'pause\_reschedule': False, 'pause\_key': None, 'run\_input\_keyset': None, 'refresh\_cache': None, 'retriable': None, 'transition\_id': '4e188426-65d4-4b76-b9b6-dcaa8381b6c5', 'task\_parameters\_id': None}}

Task Runs:

- ID: bd790d75-8102-484d-8a42-dfad58a341de, Task Name: run\_task-b74, State: {'id': '5cc60cfd-4fc6-43f5-9fc2-749bf3461583', 'type': 'COMPLETED', 'name': 'Completed', 'timestamp': '2024-10-02T16:35:31.612410+00:00', 'message': '', 'data': None, 'state\_details': {'flow\_run\_id': 'bdc5d99c-e4d9-4120-92ee-b3e55fcad2cf', 'task\_run\_id': 'bd790d75-8102-484d-8a42-dfad58a341de', 'child\_flow\_run\_id': None, 'scheduled\_time': None, 'cache\_key': None, 'cache\_expiration': None, 'deferred': None, 'untrackable\_result': True, 'pause\_timeout': None, 'pause\_reschedule': False, 'pause\_key': None, 'run\_input\_keyset': None, 'refresh\_cache': None, 'retriable': None, 'transition\_id': None, 'task\_parameters\_id': None}}

- ID: b0fb6e99-9043-452d-ba4d-5839967f59e2, Task Name: run\_task-2c4, State: {'id': '609bc287-f9ab-48f9-b4b3-adc50b9e53fc', 'type': 'COMPLETED', 'name': 'Completed', 'timestamp': '2024-10-02T16:33:34.199359+00:00', 'message': '', 'data': None, 'state\_details': {'flow\_run\_id': '7fd2fedb-80c9-4ee0-ac39-e0b6de40ed80', 'task\_run\_id': 'b0fb6e99-9043-452d-ba4d-5839967f59e2', 'child\_flow\_run\_id': None, 'scheduled\_time': None, 'cache\_key': None, 'cache\_expiration': None, 'deferred': None, 'untrackable\_result': True, 'pause\_timeout': None, 'pause\_reschedule': False, 'pause\_key': None, 'run\_input\_keyset': None, 'refresh\_cache': None, 'retriable': None, 'transition\_id': None, 'task\_parameters\_id': None}}

- ID: c3f1bde0-d193-4044-ab01-cd1166af8f30, Task Name: run\_task-8d1, State: {'id': '2e2c2d01-a52d-4f3c-bf00-e9747de32d83', 'type': 'COMPLETED', 'name': 'Completed', 'timestamp': '2024-10-02T16:33:30.816410+00:00', 'message': '', 'data': None, 'state\_details': {'flow\_run\_id': '7fd2fedb-80c9-4ee0-ac39-e0b6de40ed80', 'task\_run\_id': 'c3f1bde0-d193-4044-ab01-cd1166af8f30', 'child\_flow\_run\_id': None, 'scheduled\_time': None, 'cache\_key': None, 'cache\_expiration': None, 'deferred': None, 'untrackable\_result': True, 'pause\_timeout': None, 'pause\_reschedule': False, 'pause\_key': None, 'run\_input\_keyset': None, 'refresh\_cache': None, 'retriable': None, 'transition\_id': None, 'task\_parameters\_id': None}}

Schedules:

ID: 95501cb3-6ca3-4356-a035-f9d5303efe42, Name: colorful-okapi, ScheduledAt: 2024-10-02 22:13:25.626469+05:30

ID: eed31053-96c5-4ac5-b411-da57a72bf838, Name: eager-trout, ScheduledAt: 2024-10-02 22:11:25.626469+05:30

ID: 7af41aa5-9b93-49cc-8f61-3aee4567d062, Name: camouflaged-stork, ScheduledAt: 2024-10-02 22:09:25.626469+05:30

Concurrency Limit:

[]

(apip\_project\_venv) adimulamramkumar-mac:api adimulamramkumar$