# 

# ***AI-Enhanced-Fitness-Wellness-Analyzer-Project***

Contents

[***AI-Enhanced-Fitness-Wellness-Analyzer-Project*** 1](#_Toc154437010)

[**Data Analysis:** 1](#_Toc154437011)

[**1.** **Exploring and Preprocessing:** 1](#_Toc154437012)

[**2.** **Data Visualization:** 46](#_Toc154437013)

[**AI-Driven Recommendation:** 64](#_Toc154437014)

[**1.** **RecommendationSystem:** 64](#_Toc154437015)

[**Health Trend Analysis:** 89](#_Toc154437016)

[**1.** **HealthTrendAnalysis** 89](#_Toc154437017)

[**Goal Tracking and Achievement** 90](#_Toc154437018)

[**GoalTracking:** 90](#_Toc154437019)

[**User Profiling** 92](#_Toc154437020)

[**Machine Learning Model:** 92](#_Toc154437021)

[**User Interface Development** 106](#_Toc154437022)

[**Web\_app** 106](#_Toc154437023)

## **Data Analysis:**

### **Exploring and Preprocessing:**

#### **dailyActivity\_merged.ipynb:**

import pandas as pd

filePath = '/University/6th Semester/Sixth Semester/IDS-AIProject/FitabaseData4.12.16-5.12.16/dailyActivity\_merged.csv'

dfDailyActivity = pd.read\_csv(filePath)

# displaying the first few rows of our data

print(dfDailyActivity.head())

# Displaying the basic information about the dataset

print(dfDailyActivity.info())

# Displaying the summary statistics of numeric columns

print(dfDailyActivity.describe())

# Displaying the names of all columns

print(dfDailyActivity.columns)

# Checking for missing values in each column

print(dfDailyActivity.isnull().sum())

# Exploring unique values in a specific columns

print(dfDailyActivity['Id'].unique())

print(dfDailyActivity['ActivityDate'].unique())

print(dfDailyActivity['TotalSteps'].unique())

print(dfDailyActivity['TotalDistance'].unique())

print(dfDailyActivity['TrackerDistance'].unique())

print(dfDailyActivity['LoggedActivitiesDistance'].unique())

print(dfDailyActivity['VeryActiveDistance'].unique())

print(dfDailyActivity['ModeratelyActiveDistance'].unique())

print(dfDailyActivity['LightActiveDistance'].unique())

print(dfDailyActivity['SedentaryActiveDistance'].unique())

print(dfDailyActivity['VeryActiveMinutes'].unique())

print(dfDailyActivity['FairlyActiveMinutes'].unique())

print(dfDailyActivity['LightlyActiveMinutes'].unique())

print(dfDailyActivity['SedentaryMinutes'].unique())

print(dfDailyActivity['Calories'].unique())

# Checking for duplicate rows

duplicates = dfDailyActivity.duplicated()

print("Number of Duplicate Rows:", duplicates.sum())

duplicateRows = dfDailyActivity[duplicates]

print("\nAs No Duplicate Rows:")

print(duplicateRows)

# Removing the Duplicate Rows

dfDailyActivity = dfDailyActivity.drop\_duplicates()

# Verifying afterwards

print("\nNo Existing Duplicates:")

print("Total Number of Rows:", len(dfDailyActivity))

# Visualizing the features to identify the outliers.

import seaborn as sns

import matplotlib.pyplot as plt

plt.figure(figsize=(15, 10))

for i, column in enumerate(dfDailyActivity.columns):

    plt.subplot(3, 5, i+1)

    sns.boxplot(x=dfDailyActivity[column])

    plt.title(f'Box Plot of {column}')

plt.tight\_layout()

plt.show()

# We are applying capping

totalStepsUpperlimit = 20000

dfDailyActivity['TotalSteps'] = dfDailyActivity['TotalSteps'].clip(upper=totalStepsUpperlimit)

print("Capped TotalSteps:")

print(dfDailyActivity['TotalSteps'].describe())

totalDistanceUpperLimit = 15.0

# Applying capping for TotalDistance

dfDailyActivity['TotalDistance'] = dfDailyActivity['TotalDistance'].clip(upper=totalDistanceUpperLimit)

print("Capped TotalDistance:")

print(dfDailyActivity['TotalDistance'].describe())

# Setting the upper limit for TrackerDistance capping

trackerDistanceUpperLimit = 15.0

# Applying capping for TrackerDistance

dfDailyActivity['TrackerDistance'] = dfDailyActivity['TrackerDistance'].clip(upper=trackerDistanceUpperLimit)

# Checking the results

print("Capped TrackerDistance:")

print(dfDailyActivity['TrackerDistance'].describe())

#### **dailyCalories\_merged.ipynb:**

import pandas as pd

filePath = '/University/6th Semester/Sixth Semester/IDS-AIProject/FitabaseData4.12.16-5.12.16/dailyCalories\_merged.csv'

dfDailyCalories = pd.read\_csv(filePath)

# displaying the first few rows of our datset.

print(dfDailyCalories.head())

# Displaying the basic information about the dataset

print(dfDailyCalories.info())

# Displaying the summary statistics of numeric columns

print(dfDailyCalories.describe())

# Displaying the summary statistics of numeric columns

print(dfDailyCalories.describe())

# Checking for missing values in each column

print(dfDailyCalories.isnull().sum())

# Explore unique values in a specific columns

print(dfDailyCalories['Id'].unique())

print(dfDailyCalories['ActivityDay'].unique())

print(dfDailyCalories['Calories'].unique())

# Checking for duplicate rows

duplicates = dfDailyCalories.duplicated()

print("Number of Duplicate Rows:", duplicates.sum())

duplicate\_rows = dfDailyCalories[duplicates]

print("\nAs No Duplicate Rows:")

print(duplicate\_rows)

# Removing the Duplicate Rows

dfDailyCalories = dfDailyCalories.drop\_duplicates()

# Verifying afterwards

print("\nNo Existing Duplicates:")

print("Total Number of Rows:", len(dfDailyCalories))

# Visualizing the features to identify the outliers.

import seaborn as sns

import matplotlib.pyplot as plt

plt.figure(figsize=(15, 10))

for i, column in enumerate(dfDailyCalories.columns):

    plt.subplot(3, 5, i+1)

    sns.boxplot(x=dfDailyCalories[column])

    plt.title(f'Box Plot of {column}')

plt.tight\_layout()

plt.show()

# Setting limits for Calories capping

caloriesLowerLimit = 400

caloriesUpperLimit = 4100

# Apply capping for Calories

dfDailyCalories['Calories'] = dfDailyCalories['Calories'].clip(lower=caloriesLowerLimit, upper=caloriesUpperLimit)

# Checking results

print("Capped Calories:")

print(dfDailyCalories['Calories'].describe())

# Displaying summary statistics of all columns

print(dfDailyCalories.describe())

import seaborn as sns

import matplotlib.pyplot as plt

# Setting size of the plot

plt.figure(figsize=(16, 10))

for column in dfDailyCalories.columns:

    # Creating box plot for each feature

    plt.subplot(3, 5, dfDailyCalories.columns.get\_loc(column) + 1)  # Adjust the subplot grid as needed

    sns.boxplot(x=dfDailyCalories[column])

    plt.title(column)

plt.tight\_layout()

plt.show()

# Specifying the path to save the filtered dataset

filteredDatasetPath = '/University/6th Semester/Sixth Semester/IDS-AIProject/FilteredFitbaseData/dailyCalories\_merged\_Filtered.csv'

# Saving the DataFrame to a CSV file

dfDailyCalories.to\_csv(filteredDatasetPath, index=False)

print(f"Filtered dataset saved to: {filteredDatasetPath}")

#### **dailyIntensities\_merged.ipynb:**

import pandas as pd

filePath = '/University/6th Semester/Sixth Semester/IDS-AIProject/FitabaseData4.12.16-5.12.16/dailyIntensities\_merged.csv'

dfDailyIntensity = pd.read\_csv(filePath)

# displaying the first few rows of our datset.

print(dfDailyIntensity.head())

# Displaying the basic information about the dataset

print(dfDailyIntensity.info())

# Displaying the summary statistics of numeric columns

print(dfDailyIntensity.describe())

# Displaying the names of all columns

print(dfDailyIntensity.columns)

# Checking for missing values in each column

print(dfDailyIntensity.isnull().sum())

# Exploring unique values in a specific columns

print(dfDailyIntensity['Id'].unique())

print(dfDailyIntensity['ActivityDay'].unique())

print(dfDailyIntensity['SedentaryMinutes'].unique())

print(dfDailyIntensity['LightlyActiveMinutes'].unique())

print(dfDailyIntensity['FairlyActiveMinutes'].unique())

print(dfDailyIntensity['VeryActiveMinutes'].unique())

print(dfDailyIntensity['SedentaryActiveDistance'].unique())

print(dfDailyIntensity['LightActiveDistance'].unique())

print(dfDailyIntensity['ModeratelyActiveDistance'].unique())

print(dfDailyIntensity['VeryActiveDistance'].unique())

# Checking for duplicate rows

duplicates = dfDailyIntensity.duplicated()

print("Number of Duplicate Rows:", duplicates.sum())

duplicate\_rows = dfDailyIntensity[duplicates]

print("\nAs No Duplicate Rows:")

print(duplicate\_rows)

# Removing the Duplicate Rows

dfDailyIntensity = dfDailyIntensity.drop\_duplicates()

# Verifying afterwards

print("\nNo Existing Duplicates:")

print("Total Number of Rows:", len(dfDailyIntensity))

# Visualizing the features to identify the outliers.

import seaborn as sns

import matplotlib.pyplot as plt

plt.figure(figsize=(15, 10))

for i, column in enumerate(dfDailyIntensity.columns):

    plt.subplot(3, 5, i+1)

    sns.boxplot(x=dfDailyIntensity[column])

    plt.title(f'Box Plot of {column}')

plt.tight\_layout()

plt.show()

# We are applying capping

LightlyActiveMinutesUpperlimit = 450

dfDailyIntensity['LightlyActiveMinutes'] = dfDailyIntensity['LightlyActiveMinutes'].clip(upper=LightlyActiveMinutesUpperlimit)

print("Capped LightlyActiveMinutes:")

print(dfDailyIntensity['LightlyActiveMinutes'].describe())

FairlyActiveMinutesLimit = 50

# Applying capping for FairlyActiveMinutes

dfDailyIntensity['FairlyActiveMinutes'] = dfDailyIntensity['FairlyActiveMinutes'].clip(upper=FairlyActiveMinutesLimit)

print("Capped FairlyActiveMinutes:")

print(dfDailyIntensity['FairlyActiveMinutes'].describe())

# Setting the upper limit for VeryActiveMinutes capping

VeryActiveMinutesLimit = 75

# Applying capping for VeryActiveMinutes

dfDailyIntensity['VeryActiveMinutes'] = dfDailyIntensity['VeryActiveMinutes'].clip(upper=VeryActiveMinutesLimit)

# Checking the results

print("Capped VeryActiveMinutes:")

print(dfDailyIntensity['VeryActiveMinutes'].describe())

# Setting the upper limit for VeryActiveDistance capping

veryActiveDistanceUpperLimit = 5.0

# Applying capping for VeryActiveDistance

dfDailyIntensity['VeryActiveDistance'] = dfDailyIntensity['VeryActiveDistance'].clip(upper=veryActiveDistanceUpperLimit)

# Checking the results

print("Capped VeryActiveDistance:")

print(dfDailyIntensity['VeryActiveDistance'].describe())

# Setting the upper limit for moderatelyActiveDistance capping

moderatelyActiveDistanceUpperLimit = 2.0

# Applying capping for moderatelyActiveDistance

dfDailyIntensity['ModeratelyActiveDistance'] = dfDailyIntensity['ModeratelyActiveDistance'].clip(upper=moderatelyActiveDistanceUpperLimit)

# Checking the results

print("Capped ModeratelyActiveDistance:")

print(dfDailyIntensity['ModeratelyActiveDistance'].describe())

# Set the upper limit for lightActiveDistance capping

lightActiveDistanceUpperLimit = 9.0

# Apply capping for lightActiveDistance

dfDailyIntensity['LightActiveDistance'] = dfDailyIntensity['LightActiveDistance'].clip(upper=lightActiveDistanceUpperLimit)

# Check the results

print("Capped lightActiveDistance:")

print(dfDailyIntensity['LightActiveDistance'].describe())

# Set the upper limit for sedentaryActiveDistance capping

sedentaryActiveDistanceUpperLimit = 0.000

# Apply capping for sedentaryActiveDistance

dfDailyIntensity['SedentaryActiveDistance'] = dfDailyIntensity['SedentaryActiveDistance'].clip(upper=sedentaryActiveDistanceUpperLimit)

# Check the results

print("Capped SedentaryActiveDistance:")

print(dfDailyIntensity['SedentaryActiveDistance'].describe())

# Set the upper limit for VeryActiveMinutes capping

veryActiveMinutesUpperLimit = 75

# Apply capping for VeryActiveMinutes

dfDailyIntensity['VeryActiveMinutes'] = dfDailyIntensity['VeryActiveMinutes'].clip(upper=veryActiveMinutesUpperLimit)

# Check the results

print("Capped VeryActiveMinutes:")

print(dfDailyIntensity['VeryActiveMinutes'].describe())

# Displaying summary statistics of all columns

print(dfDailyIntensity.describe())

import seaborn as sns

import matplotlib.pyplot as plt

# Set the size of the plot

plt.figure(figsize=(16, 10))

for column in dfDailyIntensity.columns:

    # Create a box plot for each feature

    plt.subplot(3, 5, dfDailyIntensity.columns.get\_loc(column) + 1)  # Adjust the subplot grid as needed

    sns.boxplot(x=dfDailyIntensity[column])

    plt.title(column)

plt.tight\_layout()

plt.show()

# Specifying the path to save the filtered dataset

filteredDatasetPath = '/University/6th Semester/Sixth Semester/IDS-AIProject/FilteredFitbaseData/dailyIntensities\_merged\_Filtered.csv'

# Saving the DataFrame to a CSV file

dfDailyIntensity.to\_csv(filteredDatasetPath, index=False)

print(f"Filtered dataset saved to: {filteredDatasetPath}")

#### **dailySteps\_merged.ipynb:**

import pandas as pd

filePath = '/University/6th Semester/Sixth Semester/IDS-AIProject/FitabaseData4.12.16-5.12.16/dailySteps\_merged.csv'

dfDailySteps = pd.read\_csv(filePath)

# displaying the first few rows of our datset.

print(dfDailySteps.head())

# Displaying the basic information about the dataset

print(dfDailySteps.info())

# Displaying the summary statistics of numeric columns

print(dfDailySteps.describe())

# Displaying the names of all columns

print(dfDailySteps.columns)

# Checking for missing values in each column

print(dfDailySteps.isnull().sum())

# Explore unique values in a specific columns

print(dfDailySteps['Id'].unique())

print(dfDailySteps['ActivityDay'].unique())

print(dfDailySteps['StepTotal'].unique())

# Checking for duplicate rows

duplicates = dfDailySteps.duplicated()

print("Number of Duplicate Rows:", duplicates.sum())

duplicate\_rows = dfDailySteps[duplicates]

print("\nAs No Duplicate Rows:")

print(duplicate\_rows)

# Removing the Duplicate Rows

dfDailySteps = dfDailySteps.drop\_duplicates()

# Verifying afterwards

print("\nNo Existing Duplicates:")

print("Total Number of Rows:", len(dfDailySteps))

# Checking for duplicate rows

duplicates = dfDailySteps.duplicated()

print("Number of Duplicate Rows:", duplicates.sum())

duplicate\_rows = dfDailySteps[duplicates]

print("\nAs No Duplicate Rows:")

print(duplicate\_rows)

# Removing the Duplicate Rows

dfDailySteps = dfDailySteps.drop\_duplicates()

# Verifying afterwards

print("\nNo Existing Duplicates:")

print("Total Number of Rows:", len(dfDailySteps))

# We are applying capping

StepTotalUpperlimit = 21000

dfDailySteps['StepTotal'] = dfDailySteps['StepTotal'].clip(upper=StepTotalUpperlimit)

print("Capped TotalSteps:")

print(dfDailySteps['StepTotal'].describe())

# Displaying summary statistics of all columns

print(dfDailySteps.describe())

import seaborn as sns

import matplotlib.pyplot as plt

# Set the size of the plot

plt.figure(figsize=(16, 10))

for column in dfDailySteps.columns:

    # Create a box plot for each feature

    plt.subplot(3, 5, dfDailySteps.columns.get\_loc(column) + 1)  # Adjust the subplot grid as needed

    sns.boxplot(x=dfDailySteps[column])

    plt.title(column)

plt.tight\_layout()

plt.show()

# df\_daily\_activity is our DataFrame with outliers removed

# Specifying the path to save the filtered dataset

filteredDatasetPath = '/University/6th Semester/Sixth Semester/IDS-AIProject/FilteredFitbaseData/dailySteps\_merged\_Filtered.csv'

# Saving the DataFrame to a CSV file

dfDailySteps.to\_csv(filteredDatasetPath, index=False)

print(f"Filtered dataset saved to: {filteredDatasetPath}")

#### **heartrate\_seconds\_merged.ipynb:**

import pandas as pd

filePath = '/University/6th Semester/Sixth Semester/IDS-AIProject/FitabaseData4.12.16-5.12.16/heartrate\_seconds\_merged.csv'

dfHeartRateSeconds = pd.read\_csv(filePath)

# displaying the first few rows of our datset.

print(dfHeartRateSeconds.head())

# Displaying the basic information about the dataset

print(dfHeartRateSeconds.info())

# Displaying the summary statistics of numeric columns

print(dfHeartRateSeconds.describe())

# Displaying the names of all columns

print(dfHeartRateSeconds.columns)

# Checking for missing values in each column

print(dfHeartRateSeconds.isnull().sum())

# Explore unique values in a specific columns

print(dfHeartRateSeconds['Id'].unique())

print(dfHeartRateSeconds['Time'].unique())

print(dfHeartRateSeconds['Value'].unique())

# Checking for duplicate rows

duplicates = dfHeartRateSeconds.duplicated()

print("Number of Duplicate Rows:", duplicates.sum())

duplicate\_rows = dfHeartRateSeconds[duplicates]

print("\nAs No Duplicate Rows:")

print(duplicate\_rows)

# Removing the Duplicate Rows

dfHeartRateSeconds = dfHeartRateSeconds.drop\_duplicates()

# Verifying afterwards

print("\nNo Existing Duplicates:")

print("Total Number of Rows:", len(dfHeartRateSeconds))

# Visualizing the outliers for the "Value" column.

import seaborn as sns

import matplotlib.pyplot as plt

plt.figure(figsize=(8, 6))

sns.boxplot(x=dfHeartRateSeconds['Value'])

plt.title('Box Plot of Value')

plt.show()

# We are applying capping

ValueUpperlimit = 125

dfHeartRateSeconds['Value'] = dfHeartRateSeconds['Value'].clip(upper=ValueUpperlimit)

print("Capped Value:")

print(dfHeartRateSeconds['Value'].describe())

# Displaying summary statistics of all columns

print(dfHeartRateSeconds.describe())

# Visualizing the outliers for the "Value" column.

import seaborn as sns

import matplotlib.pyplot as plt

plt.figure(figsize=(8, 6))

sns.boxplot(x=dfHeartRateSeconds['Value'])

plt.title('Box Plot of Value')

plt.show()

# df\_daily\_activity is our DataFrame with outliers removed

# Specifying the path to save the filtered dataset

filteredDatasetPath = '/University/6th Semester/Sixth Semester/IDS-AIProject/FilteredFitbaseData/heartrate\_seconds\_merged\_Filtered.csv'

# Saving the DataFrame to a CSV file

dfHeartRateSeconds.to\_csv(filteredDatasetPath, index=False)

print(f"Filtered dataset saved to: {filteredDatasetPath}")

#### **hourlyCalories\_merged.ipynb:**

import pandas as pd

filePath = '/University/6th Semester/Sixth Semester/IDS-AIProject/FitabaseData4.12.16-5.12.16/hourlyCalories\_merged.csv'

dfHourlyCalories = pd.read\_csv(filePath)

# displaying the first few rows of our datset.

print(dfHourlyCalories.head())

# Displaying the basic information about the dataset

print(dfHourlyCalories.info())

# Displaying the summary statistics of numeric columns

print(dfHourlyCalories.describe())

# Displaying the names of all columns

print(dfHourlyCalories.columns)

# Checking for missing values in each column

print(dfHourlyCalories.isnull().sum())

# Explore unique values in a specific columns

print(dfHourlyCalories['Id'].unique())

print(dfHourlyCalories['ActivityHour'].unique())

print(dfHourlyCalories['Calories'].unique())

# Checking for duplicate rows

duplicates = dfHourlyCalories.duplicated()

print("Number of Duplicate Rows:", duplicates.sum())

duplicate\_rows = dfHourlyCalories[duplicates]

print("\nAs No Duplicate Rows:")

print(duplicate\_rows)

# Removing the Duplicate Rows

dfHourlyCalories = dfHourlyCalories.drop\_duplicates()

# Verifying afterwards

print("\nNo Existing Duplicates:")

print("Total Number of Rows:", len(dfHourlyCalories))

# Visualizing the features to identify the outliers.

import seaborn as sns

import matplotlib.pyplot as plt

plt.figure(figsize=(15, 10))

for i, column in enumerate(dfHourlyCalories.columns):

    plt.subplot(3, 5, i+1)

    sns.boxplot(x=dfHourlyCalories[column])

    plt.title(f'Box Plot of {column}')

plt.tight\_layout()

plt.show()

# We are applying capping

ValueUpperlimit = 175

dfHourlyCalories['Calories'] = dfHourlyCalories['Calories'].clip(upper=ValueUpperlimit)

print("Capped Calories:")

print(dfHourlyCalories['Calories'].describe())

# Displaying summary statistics of all columns

print(dfHourlyCalories.describe())

# Visualizing the features to identify the outliers.

import seaborn as sns

import matplotlib.pyplot as plt

plt.figure(figsize=(15, 10))

for i, column in enumerate(dfHourlyCalories.columns):

    plt.subplot(3, 5, i+1)

    sns.boxplot(x=dfHourlyCalories[column])

    plt.title(f'Box Plot of {column}')

plt.tight\_layout()

plt.show()

# df\_daily\_activity is our DataFrame with outliers removed

# Specifying the path to save the filtered dataset

filteredDatasetPath = '/University/6th Semester/Sixth Semester/IDS-AIProject/FilteredFitbaseData/hourlyCalories\_merged\_Filtered.csv'

# Saving the DataFrame to a CSV file

dfHourlyCalories.to\_csv(filteredDatasetPath, index=False)

print(f"Filtered dataset saved to: {filteredDatasetPath}")

#### **hourlyIntensities\_merged.ipynb:**

import pandas as pd

filePath = '/University/6th Semester/Sixth Semester/IDS-AIProject/FitabaseData4.12.16-5.12.16/hourlyIntensities\_merged.csv'

dfHourlyIntensity = pd.read\_csv(filePath)

# displaying the first few rows of our datset.

print(dfHourlyIntensity.head())

# Displaying the basic information about the dataset

print(dfHourlyIntensity.info())

# Displaying the summary statistics of numeric columns

print(dfHourlyIntensity.describe())

# Displaying the names of all columns

print(dfHourlyIntensity.columns)

# Checking for missing values in each column

print(dfHourlyIntensity.isnull().sum())

# Explore unique values in a specific columns

print(dfHourlyIntensity['Id'].unique())

print(dfHourlyIntensity['ActivityHour'].unique())

print(dfHourlyIntensity['TotalIntensity'].unique())

print(dfHourlyIntensity['AverageIntensity'].unique())

# Checking for duplicate rows

duplicates = dfHourlyIntensity.duplicated()

print("Number of Duplicate Rows:", duplicates.sum())

duplicate\_rows = dfHourlyIntensity[duplicates]

print("\nAs No Duplicate Rows:")

print(duplicate\_rows)

# Removing the Duplicate Rows

dfHourlyIntensity = dfHourlyIntensity.drop\_duplicates()

# Verifying afterwards

print("\nNo Existing Duplicates:")

print("Total Number of Rows:", len(dfHourlyIntensity))

# Visualizing the features to identify the outliers.

import seaborn as sns

import matplotlib.pyplot as plt

plt.figure(figsize=(15, 10))

for i, column in enumerate(dfHourlyIntensity.columns):

    plt.subplot(3, 5, i+1)

    sns.boxplot(x=dfHourlyIntensity[column])

    plt.title(f'Box Plot of {column}')

plt.tight\_layout()

plt.show()

# We are applying capping

TotalIntensityUpperLimit = 30

dfHourlyIntensity['TotalIntensity'] = dfHourlyIntensity['TotalIntensity'].clip(upper=TotalIntensityUpperLimit)

print("Capped TotalIntensity:")

print(dfHourlyIntensity['TotalIntensity'].describe())

AverageIntensityUpperLimit = 0.6

# Applying capping for TotalDistance

dfHourlyIntensity['AverageIntensity'] = dfHourlyIntensity['AverageIntensity'].clip(upper=AverageIntensityUpperLimit)

print("Capped AverageIntensity:")

print(dfHourlyIntensity['AverageIntensity'].describe())

# Displaying summary statistics of all columns

print(dfHourlyIntensity.describe())

import seaborn as sns

import matplotlib.pyplot as plt

# Set the size of the plot

plt.figure(figsize=(16, 10))

# Iterate through each column in the DataFrame

for column in dfHourlyIntensity.columns:

    # Create a box plot for each feature

    plt.subplot(3, 5, dfHourlyIntensity.columns.get\_loc(column) + 1)  # Adjust the subplot grid as needed

    sns.boxplot(x=dfHourlyIntensity[column])

    plt.title(column)

plt.tight\_layout()

plt.show()

# Specifying the path to save the filtered dataset

filteredDatasetPath = '/University/6th Semester/Sixth Semester/IDS-AIProject/FilteredFitbaseData/hourlyIntensities\_merged\_Filtered.csv'

# Saving the DataFrame to a CSV file

dfHourlyIntensity.to\_csv(filteredDatasetPath, index=False)

print(f"Filtered dataset saved to: {filteredDatasetPath}")

#### **hourlySteps\_merged.ipynb:**

import pandas as pd

filePath = '/University/6th Semester/Sixth Semester/IDS-AIProject/FitabaseData4.12.16-5.12.16/hourlySteps\_merged.csv'

dfHourlySteps = pd.read\_csv(filePath)

# displaying the first few rows of our datset.

print(dfHourlySteps.head())

# Displaying the basic information about the dataset

print(dfHourlySteps.info())

# Displaying the summary statistics of numeric columns

print(dfHourlySteps.describe())

# Displaying the names of all columns

print(dfHourlySteps.columns)

# Checking for missing values in each column

print(dfHourlySteps.isnull().sum())

# Explore unique values in a specific columns

print(dfHourlySteps['Id'].unique())

print(dfHourlySteps['ActivityHour'].unique())

print(dfHourlySteps['StepTotal'].unique())

# Checking for duplicate rows

duplicates = dfHourlySteps.duplicated()

print("Number of Duplicate Rows:", duplicates.sum())

duplicate\_rows = dfHourlySteps[duplicates]

print("\nAs No Duplicate Rows:")

print(duplicate\_rows)

# Removing the Duplicate Rows

dfHourlySteps = dfHourlySteps.drop\_duplicates()

# Verifying afterwards

print("\nNo Existing Duplicates:")

print("Total Number of Rows:", len(dfHourlySteps))

# Visualizing the features to identify the outliers.

import seaborn as sns

import matplotlib.pyplot as plt

plt.figure(figsize=(15, 10))

for i, column in enumerate(dfHourlySteps.columns):

    plt.subplot(3, 5, i+1)

    sns.boxplot(x=dfHourlySteps[column])

    plt.title(f'Box Plot of {column}')

plt.tight\_layout()

plt.show()

# We are applying capping

totalStepsUpperlimit = 2600

dfHourlySteps['StepTotal'] = dfHourlySteps['StepTotal'].clip(upper=totalStepsUpperlimit)

print("Capped StepTotal:")

print(dfHourlySteps['StepTotal'].describe())

# Displaying summary statistics of all columns

print(dfHourlySteps.describe())

import seaborn as sns

import matplotlib.pyplot as plt

# Set the size of the plot

plt.figure(figsize=(16, 10))

# Iterate through each column in the DataFrame

for column in dfHourlySteps.columns:

    # Create a box plot for each feature

    plt.subplot(3, 5, dfHourlySteps.columns.get\_loc(column) + 1)  # Adjust the subplot grid as needed

    sns.boxplot(x=dfHourlySteps[column])

    plt.title(column)

plt.tight\_layout()

plt.show()

# Specifying the path to save the filtered dataset

filteredDatasetPath = '/University/6th Semester/Sixth Semester/IDS-AIProject/FilteredFitbaseData/hourlySteps\_merged\_Filtered.csv'

# Saving the DataFrame to a CSV file

dfHourlySteps.to\_csv(filteredDatasetPath, index=False)

print(f"Filtered dataset saved to: {filteredDatasetPath}")

#### **minuteCaloriesNarrow\_merged.ipynb:**

import pandas as pd

filePath = '/University/6th Semester/Sixth Semester/IDS-AIProject/FitabaseData4.12.16-5.12.16/minuteCaloriesNarrow\_merged.csv'

dfMinuteCalories = pd.read\_csv(filePath)

# displaying the first few rows of our datset.

print(dfMinuteCalories.head())

# Displaying the basic information about the dataset

print(dfMinuteCalories.info())

# Displaying the summary statistics of numeric columns

print(dfMinuteCalories.describe())

# Displaying the names of all columns

print(dfMinuteCalories.columns)

# Checking for missing values in each column

print(dfMinuteCalories.isnull().sum())

# Explore unique values in a specific columns

print(dfMinuteCalories['Id'].unique())

print(dfMinuteCalories['ActivityMinute'].unique())

print(dfMinuteCalories['Calories'].unique())

# Checking for duplicate rows

duplicates = dfMinuteCalories.duplicated()

print("Number of Duplicate Rows:", duplicates.sum())

duplicate\_rows = dfMinuteCalories[duplicates]

print("\nAs No Duplicate Rows:")

print(duplicate\_rows)

# Removing the Duplicate Rows

dfMinuteCalories = dfMinuteCalories.drop\_duplicates()

# Verifying afterwards

print("\nNo Existing Duplicates:")

print("Total Number of Rows:", len(dfMinuteCalories))

# Visualizing the outliers for the "Calories" column.

import seaborn as sns

import matplotlib.pyplot as plt

plt.figure(figsize=(8, 6))

sns.boxplot(x=dfMinuteCalories['Calories'])

plt.title('Box Plot of Calories')

plt.show()

# Set the lower and upper limits for Calories capping

caloriesLowerLimit = 0.6

caloriesUpperLimit = 2.17

# Apply capping for Calories

dfMinuteCalories['Calories'] = dfMinuteCalories['Calories'].clip(lower=caloriesLowerLimit, upper=caloriesUpperLimit)

# Check the results

print("Capped Calories:")

print(dfMinuteCalories['Calories'].describe())

# Displaying summary statistics of all columns

print(dfMinuteCalories.describe())

# Visualizing the outliers for the "Calories" column.

import seaborn as sns

import matplotlib.pyplot as plt

plt.figure(figsize=(8, 6))

sns.boxplot(x=dfMinuteCalories['Calories'])

plt.title('Box Plot of Calories')

plt.show()

# Specifying the path to save the filtered dataset

filteredDatasetPath = '/University/6th Semester/Sixth Semester/IDS-AIProject/FilteredFitbaseData/minuteCaloriesNarrow\_merged\_Filtered.csv'

# Saving the DataFrame to a CSV file

dfMinuteCalories.to\_csv(filteredDatasetPath, index=False)

print(f"Filtered dataset saved to: {filteredDatasetPath}")

#### **minuteCaloriesWide\_merged.ipynb:**

import pandas as pd

filePath = '/University/6th Semester/Sixth Semester/IDS-AIProject/FitabaseData4.12.16-5.12.16/minuteCaloriesWide\_merged.csv'

dfMinuteCaloriesWide = pd.read\_csv(filePath)

# displaying the first few rows of our datset.

print(dfMinuteCaloriesWide.head())

# Displaying the basic information about the dataset

print(dfMinuteCaloriesWide.info())

# Displaying the summary statistics of numeric columns

print(dfMinuteCaloriesWide.describe())

# Displaying the names of all columns

print(dfMinuteCaloriesWide.columns)

# Checking for missing values in each column

print(dfMinuteCaloriesWide.isnull().sum())

# Explore unique values in a specific columns

print(dfMinuteCaloriesWide['Id'].unique())

print(dfMinuteCaloriesWide['ActivityHour'].unique())

print(dfMinuteCaloriesWide['Calories00'].unique())

print(dfMinuteCaloriesWide['Calories01'].unique())

print(dfMinuteCaloriesWide['Calories03'].unique())

print(dfMinuteCaloriesWide['Calories04'].unique())

print(dfMinuteCaloriesWide['Calories05'].unique())

print(dfMinuteCaloriesWide['Calories06'].unique())

print(dfMinuteCaloriesWide['Calories07'].unique())

print(dfMinuteCaloriesWide['Calories08'].unique())

print(dfMinuteCaloriesWide['Calories09'].unique())

print(dfMinuteCaloriesWide['Calories10'].unique())

print(dfMinuteCaloriesWide['Calories11'].unique())

print(dfMinuteCaloriesWide['Calories12'].unique())

print(dfMinuteCaloriesWide['Calories13'].unique())

print(dfMinuteCaloriesWide['Calories14'].unique())

print(dfMinuteCaloriesWide['Calories15'].unique())

print(dfMinuteCaloriesWide['Calories16'].unique())

print(dfMinuteCaloriesWide['Calories17'].unique())

print(dfMinuteCaloriesWide['Calories18'].unique())

print(dfMinuteCaloriesWide['Calories19'].unique())

print(dfMinuteCaloriesWide['Calories20'].unique())

print(dfMinuteCaloriesWide['Calories21'].unique())

print(dfMinuteCaloriesWide['Calories22'].unique())

print(dfMinuteCaloriesWide['Calories23'].unique())

print(dfMinuteCaloriesWide['Calories24'].unique())

print(dfMinuteCaloriesWide['Calories25'].unique())

print(dfMinuteCaloriesWide['Calories26'].unique())

print(dfMinuteCaloriesWide['Calories27'].unique())

print(dfMinuteCaloriesWide['Calories28'].unique())

print(dfMinuteCaloriesWide['Calories29'].unique())

print(dfMinuteCaloriesWide['Calories30'].unique())

print(dfMinuteCaloriesWide['Calories31'].unique())

print(dfMinuteCaloriesWide['Calories32'].unique())

print(dfMinuteCaloriesWide['Calories33'].unique())

print(dfMinuteCaloriesWide['Calories34'].unique())

print(dfMinuteCaloriesWide['Calories35'].unique())

print(dfMinuteCaloriesWide['Calories36'].unique())

print(dfMinuteCaloriesWide['Calories37'].unique())

print(dfMinuteCaloriesWide['Calories38'].unique())

print(dfMinuteCaloriesWide['Calories39'].unique())

print(dfMinuteCaloriesWide['Calories40'].unique())

print(dfMinuteCaloriesWide['Calories41'].unique())

print(dfMinuteCaloriesWide['Calories42'].unique())

print(dfMinuteCaloriesWide['Calories43'].unique())

print(dfMinuteCaloriesWide['Calories44'].unique())

print(dfMinuteCaloriesWide['Calories45'].unique())

print(dfMinuteCaloriesWide['Calories46'].unique())

print(dfMinuteCaloriesWide['Calories47'].unique())

print(dfMinuteCaloriesWide['Calories48'].unique())

print(dfMinuteCaloriesWide['Calories49'].unique())

print(dfMinuteCaloriesWide['Calories50'].unique())

print(dfMinuteCaloriesWide['Calories51'].unique())

print(dfMinuteCaloriesWide['Calories52'].unique())

print(dfMinuteCaloriesWide['Calories53'].unique())

print(dfMinuteCaloriesWide['Calories54'].unique())

print(dfMinuteCaloriesWide['Calories55'].unique())

print(dfMinuteCaloriesWide['Calories56'].unique())

print(dfMinuteCaloriesWide['Calories57'].unique())

print(dfMinuteCaloriesWide['Calories58'].unique())

print(dfMinuteCaloriesWide['Calories59'].unique())

# Checking for duplicate rows

duplicates = dfMinuteCaloriesWide.duplicated()

print("Number of Duplicate Rows:", duplicates.sum())

duplicate\_rows = dfMinuteCaloriesWide[duplicates]

print("\nAs No Duplicate Rows:")

print(duplicate\_rows)

# Removing the Duplicate Rows

dfMinuteCaloriesWide = dfMinuteCaloriesWide.drop\_duplicates()

# Verifying afterwards

print("\nNo Existing Duplicates:")

print("Total Number of Rows:", len(dfMinuteCaloriesWide))

# Visualizing the first 15 features to identify the outliers.

import seaborn as sns

import matplotlib.pyplot as plt

plt.figure(figsize=(15, 10))

# Iterate through the first 15 columns in the DataFrame

for i, column in enumerate(dfMinuteCaloriesWide.columns[:15]):

    plt.subplot(3, 5, i+1)  # Adjust the subplot grid as needed

    sns.boxplot(x=dfMinuteCaloriesWide[column])

    plt.title(f'Box Plot of {column}')

plt.tight\_layout()

plt.show()

# Visualizing the next 15 features (features 16 to 30) to identify the outliers.

import seaborn as sns

import matplotlib.pyplot as plt

plt.figure(figsize=(15, 10))

# Iterate through the next 15 columns in the DataFrame

for i, column in enumerate(dfMinuteCaloriesWide.columns[15:30]):

    plt.subplot(3, 5, i+1)

    sns.boxplot(x=dfMinuteCaloriesWide[column])

    plt.title(f'Box Plot of {column}')

plt.tight\_layout()

plt.show()

# Visualizing the next 15 features to identify the outliers.

import seaborn as sns

import matplotlib.pyplot as plt

plt.figure(figsize=(15, 10))

# Iterate through the next 15 columns in the DataFrame

for i, column in enumerate(dfMinuteCaloriesWide.columns[30:45]):

    plt.subplot(3, 5, i+1)

    sns.boxplot(x=dfMinuteCaloriesWide[column])

    plt.title(f'Box Plot of {column}')

plt.tight\_layout()

plt.show()

# Visualizing the next 15 features to identify the outliers.

import seaborn as sns

import matplotlib.pyplot as plt

plt.figure(figsize=(15, 10))

# Iterate through the next 15 columns in the DataFrame

for i, column in enumerate(dfMinuteCaloriesWide.columns[45:60]):

    plt.subplot(3, 5, i+1)

    sns.boxplot(x=dfMinuteCaloriesWide[column])

    plt.title(f'Box Plot of {column}')

plt.tight\_layout()

plt.show()

# Visualizing the last 2 features to identify the outliers.

import seaborn as sns

import matplotlib.pyplot as plt

plt.figure(figsize=(15, 10))

# Iterate through the next 15 columns in the DataFrame

for i, column in enumerate(dfMinuteCaloriesWide.columns[60:63]):

    plt.subplot(3, 5, i+1)

    sns.boxplot(x=dfMinuteCaloriesWide[column])

    plt.title(f'Box Plot of {column}')

plt.tight\_layout()

plt.show()

# We are applying capping

Calories00UpperLimit = 0.6

dfMinuteCaloriesWide['Calories00'] = dfMinuteCaloriesWide['Calories00'].clip(upper=Calories00UpperLimit)

print("Capped Calories00:")

print(dfMinuteCaloriesWide['Calories00'].describe())

# We are applying capping

Calories01UpperLimit = 0.6

dfMinuteCaloriesWide['Calories01'] = dfMinuteCaloriesWide['Calories01'].clip(upper=Calories01UpperLimit)

print("Capped Calories01:")

print(dfMinuteCaloriesWide['Calories01'].describe())

# We are applying capping

Calories02UpperLimit = 0.6

dfMinuteCaloriesWide['Calories02'] = dfMinuteCaloriesWide['Calories02'].clip(upper=Calories02UpperLimit)

print("Capped Calories02:")

print(dfMinuteCaloriesWide['Calories02'].describe())

# We are applying capping

Calories03UpperLimit = 0.6

dfMinuteCaloriesWide['Calories03'] = dfMinuteCaloriesWide['Calories03'].clip(upper=Calories03UpperLimit)

print("Capped Calories03:")

print(dfMinuteCaloriesWide['Calories03'].describe())

# We are applying capping

Calories04UpperLimit = 0.6

dfMinuteCaloriesWide['Calories04'] = dfMinuteCaloriesWide['Calories04'].clip(upper=Calories04UpperLimit)

print("Capped Calories04:")

print(dfMinuteCaloriesWide['Calories04'].describe())

# We are applying capping

Calories05UpperLimit = 0.6

dfMinuteCaloriesWide['Calories05'] = dfMinuteCaloriesWide['Calories05'].clip(upper=Calories05UpperLimit)

print("Capped Calories05:")

print(dfMinuteCaloriesWide['Calories05'].describe())

# We are applying capping

Calories06UpperLimit = 0.6

dfMinuteCaloriesWide['Calories06'] = dfMinuteCaloriesWide['Calories06'].clip(upper=Calories06UpperLimit)

print("Capped Calories06:")

print(dfMinuteCaloriesWide['Calories06'].describe())

# We are applying capping

Calories07UpperLimit = 0.6

dfMinuteCaloriesWide['Calories07'] = dfMinuteCaloriesWide['Calories07'].clip(upper=Calories07UpperLimit)

print("Capped Calories07:")

print(dfMinuteCaloriesWide['Calories07'].describe())

# We are applying capping

Calories08pperLimit = 0.6

dfMinuteCaloriesWide['Calories08'] = dfMinuteCaloriesWide['Calories08'].clip(upper=Calories08pperLimit)

print("Capped Calories08:")

print(dfMinuteCaloriesWide['Calories08'].describe())

# We are applying capping

Calories09UpperLimit = 0.6

dfMinuteCaloriesWide['Calories09'] = dfMinuteCaloriesWide['Calories09'].clip(upper=Calories09UpperLimit)

print("Capped Calories09:")

print(dfMinuteCaloriesWide['Calories09'].describe())

# We are applying capping

Calories10UpperLimit = 0.6

dfMinuteCaloriesWide['Calories10'] = dfMinuteCaloriesWide['Calories10'].clip(upper=Calories10UpperLimit)

print("Capped Calories10:")

print(dfMinuteCaloriesWide['Calories10'].describe())

# We are applying capping

Calories11UpperLimit = 0.6

dfMinuteCaloriesWide['Calories11'] = dfMinuteCaloriesWide['Calories11'].clip(upper=Calories11UpperLimit)

print("Capped Calories11:")

print(dfMinuteCaloriesWide['Calories11'].describe())

# We are applying capping

Calories12UpperLimit = 0.6

dfMinuteCaloriesWide['Calories12'] = dfMinuteCaloriesWide['Calories12'].clip(upper=Calories12UpperLimit)

print("Capped Calories12:")

print(dfMinuteCaloriesWide['Calories12'].describe())

# We are applying capping

Calories13UpperLimit = 0.6

dfMinuteCaloriesWide['Calories13'] = dfMinuteCaloriesWide['Calories13'].clip(upper=Calories13UpperLimit)

print("Capped Calories13:")

print(dfMinuteCaloriesWide['Calories13'].describe())

# We are applying capping

Calories14UpperLimit = 0.6

dfMinuteCaloriesWide['Calories14'] = dfMinuteCaloriesWide['Calories14'].clip(upper=Calories14UpperLimit)

print("Capped Calories14:")

print(dfMinuteCaloriesWide['Calories14'].describe())

# We are applying capping

Calories15UpperLimit = 0.6

dfMinuteCaloriesWide['Calories15'] = dfMinuteCaloriesWide['Calories15'].clip(upper=Calories15UpperLimit)

print("Capped Calories15:")

print(dfMinuteCaloriesWide['Calories15'].describe())

# We are applying capping

Calories16UpperLimit = 0.6

dfMinuteCaloriesWide['Calories16'] = dfMinuteCaloriesWide['Calories16'].clip(upper=Calories16UpperLimit)

print("Capped Calories16:")

print(dfMinuteCaloriesWide['Calories16'].describe())

# We are applying capping

Calories17UpperLimit = 0.6

dfMinuteCaloriesWide['Calories17'] = dfMinuteCaloriesWide['Calories17'].clip(upper=Calories17UpperLimit)

print("Capped Calories17:")

print(dfMinuteCaloriesWide['Calories17'].describe())

# We are applying capping

Calories18UpperLimit = 0.6

dfMinuteCaloriesWide['Calories18'] = dfMinuteCaloriesWide['Calories18'].clip(upper=Calories18UpperLimit)

print("Capped Calories18:")

print(dfMinuteCaloriesWide['Calories18'].describe())

# We are applying capping

Calories19UpperLimit = 0.6

dfMinuteCaloriesWide['Calories19'] = dfMinuteCaloriesWide['Calories19'].clip(upper=Calories19UpperLimit)

print("Capped Calories19:")

print(dfMinuteCaloriesWide['Calories19'].describe())

# We are applying capping

Calories20UpperLimit = 0.6

dfMinuteCaloriesWide['Calories20'] = dfMinuteCaloriesWide['Calories20'].clip(upper=Calories20UpperLimit)

print("Capped Calories20:")

print(dfMinuteCaloriesWide['Calories20'].describe())

# We are applying capping

Calories21UpperLimit = 0.6

dfMinuteCaloriesWide['Calories21'] = dfMinuteCaloriesWide['Calories21'].clip(upper=Calories21UpperLimit)

print("Capped Calories21:")

print(dfMinuteCaloriesWide['Calories21'].describe())

# We are applying capping

Calories22UpperLimit = 0.6

dfMinuteCaloriesWide['Calories22'] = dfMinuteCaloriesWide['Calories22'].clip(upper=Calories22UpperLimit)

print("Capped Calories22:")

print(dfMinuteCaloriesWide['Calories22'].describe())

# We are applying capping

Calories23UpperLimit = 0.6

dfMinuteCaloriesWide['Calories23'] = dfMinuteCaloriesWide['Calories23'].clip(upper=Calories23UpperLimit)

print("Capped Calories23:")

print(dfMinuteCaloriesWide['Calories23'].describe())

# We are applying capping

Calories24UpperLimit = 0.6

dfMinuteCaloriesWide['Calories24'] = dfMinuteCaloriesWide['Calories24'].clip(upper=Calories24UpperLimit)

print("Capped Calories24:")

print(dfMinuteCaloriesWide['Calories24'].describe())

# We are applying capping

Calories25UpperLimit = 0.6

dfMinuteCaloriesWide['Calories25'] = dfMinuteCaloriesWide['Calories25'].clip(upper=Calories25UpperLimit)

print("Capped Calories25:")

print(dfMinuteCaloriesWide['Calories25'].describe())

# We are applying capping

Calories26UpperLimit = 0.6

dfMinuteCaloriesWide['Calories26'] = dfMinuteCaloriesWide['Calories26'].clip(upper=Calories26UpperLimit)

print("Capped Calories26:")

print(dfMinuteCaloriesWide['Calories26'].describe())

# We are applying capping

Calories27UpperLimit = 0.6

dfMinuteCaloriesWide['Calories27'] = dfMinuteCaloriesWide['Calories27'].clip(upper=Calories27UpperLimit)

print("Capped Calories27:")

print(dfMinuteCaloriesWide['Calories27'].describe())

# We are applying capping

Calories28UpperLimit = 0.6

dfMinuteCaloriesWide['Calories28'] = dfMinuteCaloriesWide['Calories28'].clip(upper=Calories28UpperLimit)

print("Capped Calories28:")

print(dfMinuteCaloriesWide['Calories28'].describe())

# We are applying capping

Calories29UpperLimit = 0.6

dfMinuteCaloriesWide['Calories29'] = dfMinuteCaloriesWide['Calories29'].clip(upper=Calories29UpperLimit)

print("Capped Calories29:")

print(dfMinuteCaloriesWide['Calories29'].describe())

# We are applying capping

Calories30UpperLimit = 0.6

dfMinuteCaloriesWide['Calories30'] = dfMinuteCaloriesWide['Calories30'].clip(upper=Calories30UpperLimit)

print("Capped Calories30:")

print(dfMinuteCaloriesWide['Calories30'].describe())

# We are applying capping

Calories31UpperLimit = 0.6

dfMinuteCaloriesWide['Calories31'] = dfMinuteCaloriesWide['Calories31'].clip(upper=Calories31UpperLimit)

print("Capped Calories31:")

print(dfMinuteCaloriesWide['Calories31'].describe())

# We are applying capping

Calories32UpperLimit = 0.6

dfMinuteCaloriesWide['Calories32'] = dfMinuteCaloriesWide['Calories32'].clip(upper=Calories32UpperLimit)

print("Capped Calories32:")

print(dfMinuteCaloriesWide['Calories32'].describe())

# We are applying capping

Calories33UpperLimit = 0.6

dfMinuteCaloriesWide['Calories33'] = dfMinuteCaloriesWide['Calories33'].clip(upper=Calories33UpperLimit)

print("Capped Calories33:")

print(dfMinuteCaloriesWide['Calories33'].describe())

# We are applying capping

Calories34UpperLimit = 0.6

dfMinuteCaloriesWide['Calories34'] = dfMinuteCaloriesWide['Calories34'].clip(upper=Calories34UpperLimit)

print("Capped Calories34:")

print(dfMinuteCaloriesWide['Calories34'].describe())

# We are applying capping

Calories35UpperLimit = 0.6

dfMinuteCaloriesWide['Calories35'] = dfMinuteCaloriesWide['Calories35'].clip(upper=Calories35UpperLimit)

print("Capped Calories35:")

print(dfMinuteCaloriesWide['Calories35'].describe())

# We are applying capping

Calories36UpperLimit = 0.6

dfMinuteCaloriesWide['Calories36'] = dfMinuteCaloriesWide['Calories36'].clip(upper=Calories36UpperLimit)

print("Capped Calories36:")

print(dfMinuteCaloriesWide['Calories36'].describe())

# We are applying capping

Calories37UpperLimit = 0.6

dfMinuteCaloriesWide['Calories37'] = dfMinuteCaloriesWide['Calories37'].clip(upper=Calories37UpperLimit)

print("Capped Calories37:")

print(dfMinuteCaloriesWide['Calories37'].describe())

# We are applying capping

Calories38UpperLimit = 0.6

dfMinuteCaloriesWide['Calories38'] = dfMinuteCaloriesWide['Calories38'].clip(upper=Calories38UpperLimit)

print("Capped Calories38:")

print(dfMinuteCaloriesWide['Calories38'].describe())

# We are applying capping

Calories39UpperLimit = 0.6

dfMinuteCaloriesWide['Calories39'] = dfMinuteCaloriesWide['Calories39'].clip(upper=Calories39UpperLimit)

print("Capped Calories39:")

print(dfMinuteCaloriesWide['Calories39'].describe())

# We are applying capping

Calories40UpperLimit = 0.6

dfMinuteCaloriesWide['Calories40'] = dfMinuteCaloriesWide['Calories40'].clip(upper=Calories40UpperLimit)

print("Capped Calories40:")

print(dfMinuteCaloriesWide['Calories40'].describe())

# We are applying capping

Calories41UpperLimit = 0.6

dfMinuteCaloriesWide['Calories41'] = dfMinuteCaloriesWide['Calories41'].clip(upper=Calories41UpperLimit)

print("Capped Calories41:")

print(dfMinuteCaloriesWide['Calories41'].describe())

# We are applying capping

Calories42UpperLimit = 0.6

dfMinuteCaloriesWide['Calories42'] = dfMinuteCaloriesWide['Calories42'].clip(upper=Calories42UpperLimit)

print("Capped Calories42:")

print(dfMinuteCaloriesWide['Calories42'].describe())

# We are applying capping

Calories43UpperLimit = 0.6

dfMinuteCaloriesWide['Calories43'] = dfMinuteCaloriesWide['Calories43'].clip(upper=Calories43UpperLimit)

print("Capped Calories43:")

print(dfMinuteCaloriesWide['Calories43'].describe())

# We are applying capping

Calories44UpperLimit = 0.6

dfMinuteCaloriesWide['Calories44'] = dfMinuteCaloriesWide['Calories44'].clip(upper=Calories44UpperLimit)

print("Capped Calories44:")

print(dfMinuteCaloriesWide['Calories44'].describe())

# We are applying capping

Calories45UpperLimit = 0.6

dfMinuteCaloriesWide['Calories45'] = dfMinuteCaloriesWide['Calories45'].clip(upper=Calories45UpperLimit)

print("Capped Calories45:")

print(dfMinuteCaloriesWide['Calories45'].describe())

# We are applying capping

Calories46UpperLimit = 0.6

dfMinuteCaloriesWide['Calories46'] = dfMinuteCaloriesWide['Calories46'].clip(upper=Calories46UpperLimit)

print("Capped Calories46:")

print(dfMinuteCaloriesWide['Calories46'].describe())

# We are applying capping

Calories47UpperLimit = 0.6

dfMinuteCaloriesWide['Calories47'] = dfMinuteCaloriesWide['Calories47'].clip(upper=Calories47UpperLimit)

print("Capped Calories47:")

print(dfMinuteCaloriesWide['Calories47'].describe())

# We are applying capping

Calories48UpperLimit = 0.6

dfMinuteCaloriesWide['Calories48'] = dfMinuteCaloriesWide['Calories48'].clip(upper=Calories48UpperLimit)

print("Capped Calories48:")

print(dfMinuteCaloriesWide['Calories48'].describe())

# We are applying capping

Calories49UpperLimit = 0.6

dfMinuteCaloriesWide['Calories49'] = dfMinuteCaloriesWide['Calories49'].clip(upper=Calories49UpperLimit)

print("Capped Calories49:")

print(dfMinuteCaloriesWide['Calories49'].describe())

# We are applying capping

Calories50UpperLimit = 0.6

dfMinuteCaloriesWide['Calories50'] = dfMinuteCaloriesWide['Calories50'].clip(upper=Calories50UpperLimit)

print("Capped Calories50:")

print(dfMinuteCaloriesWide['Calories50'].describe())

# We are applying capping

Calories51UpperLimit = 0.6

dfMinuteCaloriesWide['Calories51'] = dfMinuteCaloriesWide['Calories51'].clip(upper=Calories51UpperLimit)

print("Capped Calories51:")

print(dfMinuteCaloriesWide['Calories51'].describe())

# We are applying capping

Calories52UpperLimit = 0.6

dfMinuteCaloriesWide['Calories52'] = dfMinuteCaloriesWide['Calories52'].clip(upper=Calories52UpperLimit)

print("Capped Calories52:")

print(dfMinuteCaloriesWide['Calories52'].describe())

# We are applying capping

Calories53UpperLimit = 0.6

dfMinuteCaloriesWide['Calories53'] = dfMinuteCaloriesWide['Calories53'].clip(upper=Calories53UpperLimit)

print("Capped Calories53:")

print(dfMinuteCaloriesWide['Calories53'].describe())

# We are applying capping

Calories54UpperLimit = 0.6

dfMinuteCaloriesWide['Calories54'] = dfMinuteCaloriesWide['Calories54'].clip(upper=Calories54UpperLimit)

print("Capped Calories54:")

print(dfMinuteCaloriesWide['Calories54'].describe())

# We are applying capping

Calories55UpperLimit = 0.6

dfMinuteCaloriesWide['Calories55'] = dfMinuteCaloriesWide['Calories55'].clip(upper=Calories55UpperLimit)

print("Capped Calories55:")

print(dfMinuteCaloriesWide['Calories55'].describe())

# We are applying capping

Calories56UpperLimit = 0.6

dfMinuteCaloriesWide['Calories56'] = dfMinuteCaloriesWide['Calories56'].clip(upper=Calories56UpperLimit)

print("Capped Calories56:")

print(dfMinuteCaloriesWide['Calories56'].describe())

# We are applying capping

Calories57UpperLimit = 0.6

dfMinuteCaloriesWide['Calories57'] = dfMinuteCaloriesWide['Calories57'].clip(upper=Calories57UpperLimit)

print("Capped Calories57:")

print(dfMinuteCaloriesWide['Calories57'].describe())

# We are applying capping

Calories58UpperLimit = 0.6

dfMinuteCaloriesWide['Calories58'] = dfMinuteCaloriesWide['Calories58'].clip(upper=Calories58UpperLimit)

print("Capped Calories58:")

print(dfMinuteCaloriesWide['Calories58'].describe())

# We are applying capping

Calories59UpperLimit = 0.6

dfMinuteCaloriesWide['Calories59'] = dfMinuteCaloriesWide['Calories59'].clip(upper=Calories59UpperLimit)

print("Capped Calories59:")

print(dfMinuteCaloriesWide['Calories59'].describe())

# Displaying summary statistics of all columns

print(dfMinuteCaloriesWide.describe())

# Visualizing the first 15 features to identify the outliers.

import seaborn as sns

import matplotlib.pyplot as plt

plt.figure(figsize=(15, 10))

# Iterate through the first 15 columns in the DataFrame

for i, column in enumerate(dfMinuteCaloriesWide.columns[:15]):

    plt.subplot(3, 5, i+1)

    sns.boxplot(x=dfMinuteCaloriesWide[column])

    plt.title(f'Box Plot of {column}')

plt.tight\_layout()

plt.show()

# Visualizing the next 15 features to identify the outliers.

import seaborn as sns

import matplotlib.pyplot as plt

plt.figure(figsize=(15, 10))

# Iterate through the next 15 columns in the DataFrame

for i, column in enumerate(dfMinuteCaloriesWide.columns[15:30]):

    plt.subplot(3, 5, i+1)

    sns.boxplot(x=dfMinuteCaloriesWide[column])

    plt.title(f'Box Plot of {column}')

plt.tight\_layout()

plt.show()

# Visualizing the next 15 features to identify the outliers.

import seaborn as sns

import matplotlib.pyplot as plt

plt.figure(figsize=(15, 10))

# Iterate through the next 15 columns in the DataFrame

for i, column in enumerate(dfMinuteCaloriesWide.columns[30:45]):

    plt.subplot(3, 5, i+1)

    sns.boxplot(x=dfMinuteCaloriesWide[column])

    plt.title(f'Box Plot of {column}')

plt.tight\_layout()

plt.show()

# Visualizing the next 15 features to identify the outliers.

import seaborn as sns

import matplotlib.pyplot as plt

plt.figure(figsize=(15, 10))

# Iterate through the next 15 columns in the DataFrame

for i, column in enumerate(dfMinuteCaloriesWide.columns[45:60]):

    plt.subplot(3, 5, i+1)

    sns.boxplot(x=dfMinuteCaloriesWide[column])

    plt.title(f'Box Plot of {column}')

plt.tight\_layout()

plt.show()

# Specifying the path to save the filtered dataset

filteredDatasetPath = '/University/6th Semester/Sixth Semester/IDS-AIProject/FilteredFitbaseData/minuteCaloriesWide\_merged\_Filtered.csv'

# Saving the DataFrame to a CSV file

dfMinuteCaloriesWide.to\_csv(filteredDatasetPath, index=False)

print(f"Filtered dataset saved to: {filteredDatasetPath}")

#### **minuteIntensitiesNarrow\_merged.ipynb:**

import pandas as pd

file\_path = '/University/6th Semester/Sixth Semester/IDS-AIProject/FitabaseData4.12.16-5.12.16/minuteIntensitiesNarrow\_merged.csv'

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

# Display the first few rows of the dataset

print(df\_minute\_intensities.head())

# Display basic information about the dataset

print(df\_minute\_intensities.info())

# Display summary statistics of numeric columns

print(df\_minute\_intensities.describe())

# Display the names of all columns

print(df\_minute\_intensities.columns)

# Checking for missing values in each column

print(df\_minute\_intensities.isnull().sum())

# Explore unique values in the 'Id', 'ActivityMinute', and 'Intensity' columns

print(df\_minute\_intensities['Id'].unique())

print(df\_minute\_intensities['ActivityMinute'].unique())

print(df\_minute\_intensities['Intensity'].unique())

# Checking for duplicate rows

duplicates\_minute = df\_minute\_intensities.duplicated()

print("Number of Duplicate Rows:", duplicates\_minute.sum())

#duplicate\_rows\_minute = df\_minute\_intensities[duplicates\_minute]

print("\nDuplicate Rows:")

print(duplicates\_minute)

# Removing duplicate rows

df\_minute\_intensities = df\_minute\_intensities.drop\_duplicates()

# Verifying afterwards

print("\nNo Existing Duplicates:")

print("Total Number of Rows:", len(df\_minute\_intensities))

# Visualizing the outliers for the "Intensity" column.

import seaborn as sns

import matplotlib.pyplot as plt

plt.figure(figsize=(8, 6))

sns.boxplot(x=df\_minute\_intensities['Intensity'])

plt.title('Box Plot of Intensity')

plt.show()

# We are applying capping

IntensityUpperlimit = 0.000

df\_minute\_intensities['Intensity'] = df\_minute\_intensities['Intensity'].clip(upper=IntensityUpperlimit)

print("Capped Intensity:")

print(df\_minute\_intensities['Intensity'].describe())

# Visualizing the outliers for the "Intensity" column.

import seaborn as sns

import matplotlib.pyplot as plt

plt.figure(figsize=(8, 6))

sns.boxplot(x=df\_minute\_intensities['Intensity'])

plt.title('Box Plot of Intensity')

plt.show()

column\_name = 'Id'

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

sns.boxplot(x=df\_minute\_intensities[column\_name])

plt.xlabel('Id')

plt.title(f'Boxplot of {column\_name} without outliers')

plt.show()

# Specifying the path to save the filtered dataset

filteredDatasetPath = '/University/6th Semester/Sixth Semester/IDS-AIProject/FilteredFitbaseData/minuteIntensitiesNarrow\_merged\_Filtered.csv'

# Saving the DataFrame to a CSV file

df\_minute\_intensities.to\_csv(filteredDatasetPath, index=False)

print(f"Filtered dataset saved to: {filteredDatasetPath}")

#### **minuteIntensitiesWide\_merged.ipynb:**

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

file\_path\_wide = '/University/6th Semester/Sixth Semester/IDS-AIProject/FitabaseData4.12.16-5.12.16/minuteIntensitiesWide\_merged.csv'

df\_minute\_intensities\_wide = pd.read\_csv(file\_path\_wide)

# Display the first few rows

print(df\_minute\_intensities\_wide.head())

# Display basic information about the dataset

print(df\_minute\_intensities\_wide.info())

# Checking for missing values in each column

print(df\_minute\_intensities\_wide.isnull().sum())

# Checking for missing values in each column

print(df\_minute\_intensities\_wide.isnull().sum())

# Handling missing values

# Checking for duplicate rows

duplicates\_wide = df\_minute\_intensities\_wide.duplicated()

print("Number of Duplicate Rows:", duplicates\_wide.sum())

duplicate\_rows\_wide = df\_minute\_intensities\_wide[duplicates\_wide]

print("\nAs No Duplicate Rows:")

print(duplicate\_rows\_wide)

# Removing the Duplicate Rows

df\_minute\_intensities\_wide = df\_minute\_intensities\_wide.drop\_duplicates()

# Verifying afterwards

print("\nNo Existing Duplicates:")

print("Total Number of Rows:", len(df\_minute\_intensities\_wide))

# Convert 'ActivityHour' to datetime format

df\_minute\_intensities\_wide['ActivityHour'] = pd.to\_datetime(df\_minute\_intensities\_wide['ActivityHour'])

# Visualization: Time series plot for 'Intensity' values

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

for i in range(2, len(df\_minute\_intensities\_wide.columns)):

    plt.plot(df\_minute\_intensities\_wide['ActivityHour'], df\_minute\_intensities\_wide.iloc[:, i], label=df\_minute\_intensities\_wide.columns[i])

plt.title('Time Series of Intensity Values')

plt.xlabel('Time')

plt.ylabel('Intensity')

plt.legend(loc='upper right')

plt.show()

# Box plot for 'Intensity' values

plt.figure(figsize=(16, 10))

for i, column in enumerate(df\_minute\_intensities\_wide.columns[2:]):

    plt.subplot(5, 12, i + 1)  # Adjust the subplot grid as needed

    sns.boxplot(x=df\_minute\_intensities\_wide[column])

    plt.title(f'Box Plot of {column}')

plt.tight\_layout()

plt.show()

def remove\_outliers\_iqr(data, column):

    Q1 = data[column].quantile(0.25)

    Q3 = data[column].quantile(0.75)

    IQR = Q3 - Q1

    lower\_bound = Q1 - 1.5 \* IQR

    upper\_bound = Q3 + 1.5 \* IQR

    outliers = data[(data[column] < lower\_bound) | (data[column] > upper\_bound)]

    data\_no\_outliers = data[(data[column] >= lower\_bound) & (data[column] <= upper\_bound)]

    return data\_no\_outliers

# Remove outliers in each column of df\_minute\_intensities\_wide

for column in df\_minute\_intensities\_wide.columns[2:]:

    df\_minute\_intensities\_wide = remove\_outliers\_iqr(df\_minute\_intensities\_wide, column)

# Box plot without outliers

plt.figure(figsize=(16, 10))

for i, column in enumerate(df\_minute\_intensities\_wide.columns[2:]):

    plt.subplot(5, 12, i + 1)

    sns.boxplot(x=df\_minute\_intensities\_wide[column])

    plt.title(f'{column}')

plt.tight\_layout()

plt.show()

# Specifying the path to save the filtered dataset

filteredDatasetPath = '/University/6th Semester/Sixth Semester/IDS-AIProject/FilteredFitbaseData/minuteIntensitiesWide\_merged\_Filtered.csv'

# Saving the DataFrame to a CSV file

df\_minute\_intensities\_wide.to\_csv(filteredDatasetPath, index=False)

print(f"Filtered dataset saved to: {filteredDatasetPath}")

#### **minuteMETsNarrow\_merged.ipynb:**

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

file\_path = '/University/6th Semester/Sixth Semester/IDS-AIProject/FitabaseData4.12.16-5.12.16/minuteMETsNarrow\_merged.csv'

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

# Display the first few rows of the dataset

print(df\_mets.head())

# Display basic information about the dataset

print(df\_mets.info())

# Check for missing values

print(df\_mets.isnull().sum())

# Handling missing values

# Drop rows with missing values

df\_mets = df\_mets.dropna()

# Explore unique values

print(df\_mets['Id'].unique())

print(df\_mets['ActivityMinute'].unique())

print(df\_mets['METs'].unique())

# Box plot

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

sns.boxplot(x='METs', data=df\_mets)

plt.title('Distribution of METs')

plt.xlabel('METs')

plt.show()

def remove\_outliers\_iqr(df, column):

    Q1 = df[column].quantile(0.25)

    Q3 = df[column].quantile(0.75)

    IQR = Q3 - Q1

    # lower and upper bounds for outliers

    lower\_bound = Q1 - 1.5 \* IQR

    upper\_bound = Q3 + 1.5 \* IQR

    # Identify and remove outliers

    outliers = df[(df[column] < lower\_bound) | (df[column] > upper\_bound)]

    df\_no\_outliers = df[(df[column] >= lower\_bound) & (df[column] <= upper\_bound)]

    return df\_no\_outliers, outliers

# Remove outliers for 'METs' column

df\_mets\_no\_outliers, outliers\_mets = remove\_outliers\_iqr(df\_mets, 'METs')

# Display the removed outliers

print("Outliers:")

print(outliers\_mets)

# Box plot without outliers

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

sns.boxplot(x='METs', data=df\_mets\_no\_outliers)

plt.title('Distribution of METs (No Outliers)')

plt.xlabel('METs')

plt.show()

# Specifying the path to save the filtered dataset

filteredDatasetPath = '/University/6th Semester/Sixth Semester/IDS-AIProject/FilteredFitbaseData/minuteMETsNarrow\_merged\_Filtered.csv'

# Saving the DataFrame to a CSV file

df\_mets\_no\_outliers.to\_csv(filteredDatasetPath, index=False)

print(f"Filtered dataset saved to: {filteredDatasetPath}")

#### **minuteSleep\_merged.ipynb:**

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

file\_path = '/University/6th Semester/Sixth Semester/IDS-AIProject/FitabaseData4.12.16-5.12.16/minuteSleep\_merged.csv'

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

# Display the first few rows of the dataset

print(df\_sleep.head())

# Display basic information about the dataset

print(df\_sleep.info())

# Check for missing values

print(df\_sleep.isnull().sum())

# Handling missing values, filling missing values with the median

df\_sleep['value'].fillna(df\_sleep['value'].median(), inplace=True)

# Convert 'date' to datetime format

df\_sleep['date'] = pd.to\_datetime(df\_sleep['date'])

# Remove duplicates if any

df\_sleep.drop\_duplicates(inplace=True)

# Time series plot

plt.figure(figsize=(15, 6))

sns.lineplot(x='date', y='value', data=df\_sleep)

plt.title('Sleep Value Over Time')

plt.xlabel('Date')

plt.ylabel('Sleep Value')

plt.show()

# Box plot for all columns in 'minuteSleep\_merged'

plt.figure(figsize=(20, 10))

for i, column in enumerate(df\_sleep.columns[1:]):  # Exclude 'Id' column

    plt.subplot(3, 4, i + 1)

    sns.boxplot(x=column, data=df\_sleep)

    plt.title(f'Box Plot of {column}')

plt.tight\_layout()

plt.show()

# We are applying capping

df\_sleepUpperLimit = 1.001

df\_sleep['value'] = df\_sleep['value'].clip(upper=df\_sleepUpperLimit)

print("Capped Value:")

print(df\_sleep['value'].describe())

# Box plot for all columns in 'minuteSleep\_merged'

plt.figure(figsize=(20, 10))

for i, column in enumerate(df\_sleep.columns[1:]):  # Exclude 'Id' column

    plt.subplot(3, 4, i + 1)

    sns.boxplot(x=column, data=df\_sleep)

    plt.title(f'Box Plot of {column}')

plt.tight\_layout()

plt.show()

# Specifying the path to save the filtered dataset

filteredDatasetPath = '/University/6th Semester/Sixth Semester/IDS-AIProject/FilteredFitbaseData/minuteSleep\_merged\_Filtered.csv'

# Saving the DataFrame to a CSV file

df\_sleep.to\_csv(filteredDatasetPath, index=False)

print(f"Filtered dataset saved to: {filteredDatasetPath}")

#### **minuteStepsNarrow\_merged.ipynb:**

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

file\_path = '/University/6th Semester/Sixth Semester/IDS-AIProject/FitabaseData4.12.16-5.12.16/minuteStepsNarrow\_merged.csv'

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

# Display the first few rows of the dataset

print(df\_steps.head())

# Display basic information about the dataset

print(df\_steps.info())

# Check for missing values

print(df\_steps.isnull().sum())

# Convert 'ActivityMinute' to datetime format

df\_steps['ActivityMinute'] = pd.to\_datetime(df\_steps['ActivityMinute'])

# Data Cleaning

# Remove duplicates

df\_steps.drop\_duplicates(inplace=True) ,

# Time series plot

plt.figure(figsize=(15, 6))

sns.lineplot(x='ActivityMinute', y='Steps', data=df\_steps)

plt.title('Steps Over Time')

plt.xlabel('Activity Minute')

plt.ylabel('Steps')

plt.show()

# Visualizing the outliers for the "Id" column.

import seaborn as sns

import matplotlib.pyplot as plt

plt.figure(figsize=(8, 6))

sns.boxplot(x=df\_steps['Id'])

plt.title('Box Plot of Id')

plt.show()

# Visualizing the outliers for the "Steps" column.

import seaborn as sns

import matplotlib.pyplot as plt

plt.figure(figsize=(8, 6))

sns.boxplot(x=df\_steps['Steps'])

plt.title('Box Plot of Steps')

plt.show()

# We are applying capping

StepsUpperlimit = 0.0001

df\_steps['Steps'] = df\_steps['Steps'].clip(upper=StepsUpperlimit)

print("Capped Steps:")

print(df\_steps['Steps'].describe())

# Visualizing the outliers for the "Steps" column.

import seaborn as sns

import matplotlib.pyplot as plt

plt.figure(figsize=(8, 6))

sns.boxplot(x=df\_steps['Steps'])

plt.title('Box Plot of Steps')

plt.show()

def remove\_outliers\_iqr(df, column):

    Q1 = df[column].quantile(0.25)

    Q3 = df[column].quantile(0.75)

    IQR = Q3 - Q1

    lower\_bound = Q1 - 1.5 \* IQR

    upper\_bound = Q3 + 1.5 \* IQR

    outliers = df[(df[column] < lower\_bound) | (df[column] > upper\_bound)]

    df\_no\_outliers = df[(df[column] >= lower\_bound) & (df[column] <= upper\_bound)]

    return df\_no\_outliers, outliers

# Remove outliers for 'Steps' column in 'minuteStepsNarrow\_merged'

df\_steps\_no\_outliers, outliers\_steps = remove\_outliers\_iqr(df\_steps, 'Steps')

# Display boxplot for 'Steps' column after removing outliers

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

sns.boxplot(x='Steps', data=df\_steps\_no\_outliers)

plt.title('Box Plot of Steps')

plt.xlabel('Steps')

plt.show()

# Specifying the path to save the filtered dataset

filteredDatasetPath = '/University/6th Semester/Sixth Semester/IDS-AIProject/FilteredFitbaseData/minuteStepsNarrow\_merged\_Filtered.csv'

# Saving the DataFrame to a CSV file

df\_steps.to\_csv(filteredDatasetPath, index=False)

print(f"Filtered dataset saved to: {filteredDatasetPath}")

#### **minuteStepsWide\_merged.ipynb:**

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

file\_path = '/University/6th Semester/Sixth Semester/IDS-AIProject/FitabaseData4.12.16-5.12.16/minuteStepsWide\_merged.csv'

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

# Display the first few rows of the dataset

print(df\_steps\_wide.head())

# Display basic information about the dataset

print(df\_steps\_wide.info())

# Check for missing values

print(df\_steps\_wide.isnull().sum())

# Convert 'ActivityHour' to datetime format

df\_steps\_wide['ActivityHour'] = pd.to\_datetime(df\_steps\_wide['ActivityHour'])

# Remove duplicates

df\_steps\_wide.drop\_duplicates(inplace=True)

# Time series plot

plt.figure(figsize=(15, 6))

sns.lineplot(x='ActivityHour', y='value', hue='variable',

             data=pd.melt(df\_steps\_wide, id\_vars=['ActivityHour'],

                          value\_vars=df\_steps\_wide.columns[2:]))

plt.title('Steps Over Time')

plt.xlabel('Activity Hour')

plt.ylabel('Steps')

plt.show()

# Box plot

plt.figure(figsize=(8, 6))

sns.boxplot(data=df\_steps\_wide.iloc[:, 2:])

plt.title('Distribution of Steps')

plt.xlabel('Hourly Steps')

plt.show()

# Melt the dataframe to long format for easy plotting

df\_melted = pd.melt(df\_steps\_wide, id\_vars=['Id', 'ActivityHour'], var\_name='Hour', value\_name='Steps')

# Box plot for each column

plt.figure(figsize=(20, 10))

sns.boxplot(x='Hour', y='Steps', data=df\_melted)

plt.title('Box Plot of Steps by Hour')

plt.xlabel('Hour')

plt.ylabel('Steps')

plt.xticks(rotation=90)

plt.show()

# Function to remove outliers using IQR method

def remove\_outliers\_iqr(df, column):

    Q1 = df[column].quantile(0.25)

    Q3 = df[column].quantile(0.75)

    IQR = Q3 - Q1

    lower\_bound = Q1 - 1.5 \* IQR

    upper\_bound = Q3 + 1.5 \* IQR

    outliers = df[(df[column] < lower\_bound) | (df[column] > upper\_bound)]

    df\_no\_outliers = df[(df[column] >= lower\_bound) & (df[column] <= upper\_bound)]

    return df\_no\_outliers, outliers

# Melt the dataframe to long format for easy plotting

df\_melted = pd.melt(df\_steps\_wide, id\_vars=['Id', 'ActivityHour'], var\_name='Hour', value\_name='Steps')

# Remove outliers for each hour

outliers\_dict = {}

for hour in df\_melted['Hour'].unique():

    df\_no\_outliers, outliers = remove\_outliers\_iqr(df\_melted[df\_melted['Hour'] == hour], 'Steps')

    df\_melted.loc[df\_melted['Hour'] == hour, 'Steps'] = df\_no\_outliers['Steps']

    outliers\_dict[hour] = outliers

# Box plot for each column without outliers

plt.figure(figsize=(20, 10))

sns.boxplot(x='Hour', y='Steps', data=df\_melted)

plt.title('Box Plot of Steps by Hour (Without Outliers)')

plt.xlabel('Hour')

plt.ylabel('Steps')

plt.xticks(rotation=90)

plt.show()

# Specifying the path to save the filtered dataset

filteredDatasetPath = '/University/6th Semester/Sixth Semester/IDS-AIProject/FilteredFitbaseData/minuteStepsWide\_merged\_Filtered.csv'

# Saving the DataFrame to a CSV file

df\_melted.to\_csv(filteredDatasetPath, index=False)

print(f"Filtered dataset saved to: {filteredDatasetPath}")

#### **sleepDay\_merged.ipynb:**

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

file\_path = '/University/6th Semester/Sixth Semester/IDS-AIProject/FitabaseData4.12.16-5.12.16/sleepDay\_merged.csv'

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

# Display the first few rows of the dataset

print(df\_sleep\_day.head())

# Check for missing values

print(df\_sleep\_day.isnull().sum())

# Check data types

print(df\_sleep\_day.dtypes)

# Descriptive statistics

print(df\_sleep\_day.describe())

# Convert 'SleepDay' to datetime format

df\_sleep\_day['SleepDay'] = pd.to\_datetime(df\_sleep\_day['SleepDay'])

# Handle missing values

df\_sleep\_day = df\_sleep\_day.dropna()

plt.figure(figsize=(15, 6))

sns.lineplot(x='SleepDay', y='TotalSleepRecords', data=df\_sleep\_day)

plt.title('Total Sleep Records Over Time')

plt.xlabel('Sleep Day')

plt.ylabel('Total Sleep Records')

plt.show()

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

sns.histplot(df\_sleep\_day['TotalMinutesAsleep'], bins=30, kde=True)

plt.title('Distribution of Total Minutes Asleep')

plt.xlabel('Total Minutes Asleep')

plt.ylabel('Frequency')

plt.show()

plt.figure(figsize=(10, 8))

sns.heatmap(df\_sleep\_day.corr(), annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.5)

plt.title('Correlation Heatmap')

plt.show()

# Box plot for all columns

plt.figure(figsize=(12, 8))

# Box plot for 'TotalSleepRecords' column

plt.subplot(3, 1, 1)

sns.boxplot(x='TotalSleepRecords', data=df\_sleep\_day)

plt.title('Box Plot of TotalSleepRecords')

plt.xlabel('TotalSleepRecords')

# Box plot for 'TotalMinutesAsleep' column

plt.subplot(3, 1, 2)

sns.boxplot(x='TotalMinutesAsleep', data=df\_sleep\_day)

plt.title('Box Plot of TotalMinutesAsleep')

plt.xlabel('TotalMinutesAsleep')

# Box plot for 'TotalTimeInBed' column

plt.subplot(3, 1, 3)

sns.boxplot(x='TotalTimeInBed', data=df\_sleep\_day)

plt.title('Box Plot of TotalTimeInBed')

plt.xlabel('TotalTimeInBed')

plt.tight\_layout()

plt.show()

def remove\_outliers\_iqr(df, column):

    Q1 = df[column].quantile(0.25)

    Q3 = df[column].quantile(0.75)

    IQR = Q3 - Q1

    lower\_bound = Q1 - 1.5 \* IQR

    upper\_bound = Q3 + 1.5 \* IQR

    df\_no\_outliers = df[(df[column] >= lower\_bound) & (df[column] <= upper\_bound)]

    outliers = df[(df[column] < lower\_bound) | (df[column] > upper\_bound)]

    return df\_no\_outliers, outliers

# Remove outliers for each column

df\_sleep\_day\_no\_outliers\_records, outliers\_records = remove\_outliers\_iqr(df\_sleep\_day, 'TotalSleepRecords')

df\_sleep\_day\_no\_outliers\_minutes, outliers\_minutes = remove\_outliers\_iqr(df\_sleep\_day, 'TotalMinutesAsleep')

df\_sleep\_day\_no\_outliers\_time, outliers\_time = remove\_outliers\_iqr(df\_sleep\_day, 'TotalTimeInBed')

# Box plots without outliers

plt.figure(figsize=(12, 8))

# Box plot for 'TotalSleepRecords' column

plt.subplot(3, 1, 1)

sns.boxplot(x='TotalSleepRecords', data=df\_sleep\_day\_no\_outliers\_records)

plt.title('Box Plot of TotalSleepRecords (No Outliers)')

plt.xlabel('TotalSleepRecords')

# Box plot for 'TotalMinutesAsleep' column

plt.subplot(3, 1, 2)

sns.boxplot(x='TotalMinutesAsleep', data=df\_sleep\_day\_no\_outliers\_minutes)

plt.title('Box Plot of TotalMinutesAsleep (No Outliers)')

plt.xlabel('TotalMinutesAsleep')

# Box plot for 'TotalTimeInBed' column

plt.subplot(3, 1, 3)

sns.boxplot(x='TotalTimeInBed', data=df\_sleep\_day\_no\_outliers\_time)

plt.title('Box Plot of TotalTimeInBed (No Outliers)')

plt.xlabel('TotalTimeInBed')

plt.tight\_layout()

plt.show()

df\_filtered = pd.DataFrame({

    'TotalSleepRecords': df\_sleep\_day\_no\_outliers\_records['TotalSleepRecords'],

    'TotalMinutesAsleep': df\_sleep\_day\_no\_outliers\_minutes['TotalMinutesAsleep'],

    'TotalTimeInBed': df\_sleep\_day\_no\_outliers\_time['TotalTimeInBed']

})

# Specifying the path to save the filtered dataset

filteredDatasetPath = '/University/6th Semester/Sixth Semester/IDS-AIProject/FilteredFitbaseData/sleepDay\_merged\_Filtered.csv'

# Saving the DataFrame to a CSV file

df\_filtered.to\_csv(filteredDatasetPath, index=False)

print(f"Filtered dataset saved to: {filteredDatasetPath}")

#### **weightLogInfo\_merged.ipynb:**

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

file\_path = '/University/6th Semester/Sixth Semester/IDS-AIProject/FitabaseData4.12.16-5.12.16/weightLogInfo\_merged.csv'

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

# Display the first few rows of the dataset

print(df\_weight\_log.head())

# Check for missing values

print(df\_weight\_log.isnull().sum())

# Check data types

print(df\_weight\_log.dtypes)

# Descriptive statistics

print(df\_weight\_log.describe())

# Convert 'Date' to datetime format

df\_weight\_log['Date'] = pd.to\_datetime(df\_weight\_log['Date'])

# Handle missing values

df\_weight\_log = df\_weight\_log.dropna()

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

sns.histplot(df\_weight\_log['BMI'], bins=30, kde=True)

plt.title('Distribution of BMI')

plt.xlabel('BMI')

plt.ylabel('Frequency')

plt.show()

import seaborn as sns

import matplotlib.pyplot as plt

# Box plot for each column

num\_cols = len(df\_weight\_log.columns)

num\_rows = (num\_cols // 3) + (num\_cols % 3 > 0)

plt.figure(figsize=(16, 4 \* num\_rows))

for i, column in enumerate(df\_weight\_log.columns):

    plt.subplot(num\_rows, 3, i + 1)

    sns.boxplot(x=df\_weight\_log[column])

    plt.title(f'Box Plot of {column}')

    plt.xlabel(column)

    plt.ylabel('Values')

plt.tight\_layout()

plt.show()

### **Data Visualization:**

#### **dailyActivity\_merged.ipynb:**

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Load the cleaned dataset

df = pd.read\_csv('/University/6th Semester/Sixth Semester/IDS-AIProject/Data/FilteredFitbaseData/dailyActivity\_merged\_Filtered.csv')

# Set the style for Seaborn plots

sns.set(style="whitegrid")

# Example: Histogram for TotalSteps

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

sns.histplot(df['TotalSteps'], bins=30, kde=True, color='skyblue')

plt.title('Distribution of TotalSteps')

plt.xlabel('TotalSteps')

plt.ylabel('Frequency')

plt.savefig('/University/6th Semester/Sixth Semester/IDS-AIProject/VisualizationImages/dailyActivity\_merged/TotalSteps\_distribution\_Filtered.png')  # Save the figure

plt.show()

# Example: Boxplot for TotalSteps

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

sns.boxplot(x=df['TotalSteps'], color='lightcoral')

plt.title('Box Plot of TotalSteps')

plt.xlabel('TotalSteps')

plt.savefig('/University/6th Semester/Sixth Semester/IDS-AIProject/VisualizationImages/dailyActivity\_merged/TotalSteps\_distribution\_BoxPlot\_Filtered.png')  # Save the figure

plt.show()

# Example: Scatter plot for TotalSteps and TotalDistance

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

sns.scatterplot(x='TotalSteps', y='TotalDistance', data=df, color='salmon')

plt.title('Scatter Plot of TotalSteps vs TotalDistance')

plt.xlabel('TotalSteps')

plt.ylabel('TotalDistance')

plt.savefig('/University/6th Semester/Sixth Semester/IDS-AIProject/VisualizationImages/dailyActivity\_merged/TotalSteps&TotalDist\_distribution\_ScatterPlot\_Filtered.png')  # Save the figure

plt.show()

#### **dailyCalories\_merged.ipynb:**

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Load the raw dataset for visualization

raw\_df\_calories = pd.read\_csv('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/FilteredFitbaseData/dailyCalories\_merged\_Filtered.csv')

# Set the style for Seaborn plots

sns.set(style="whitegrid")

# Example: Histogram for Calories in Filtered Data

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

sns.histplot(raw\_df\_calories['Calories'], bins=30, kde=True, color='orange')

plt.title('Distribution of Calories (Filtered Data)')

plt.xlabel('Calories')

plt.ylabel('Frequency')

plt.savefig('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/VisualizationImages/dailyCalories\_merged/Calories\_distribution\_cleaned.png')  # Save the figure

plt.show()

# Example: Boxplot for Calories in Filtered Data

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

sns.boxplot(x=raw\_df\_calories['Calories'], color='lightgreen')

plt.title('Box Plot of Calories (Filtered Data)')

plt.xlabel('Calories')

plt.savefig('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/VisualizationImages/dailyCalories\_merged/Calories\_boxplot\_raw.png')  # Save the figure

plt.show()

#### **dailyIntensities\_merged\_Filtered.ipynb:**

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Load the cleaned dataset

cleaned\_df\_intensities = pd.read\_csv('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/FilteredFitbaseData/dailyIntensities\_merged\_Filtered.csv')

# Set the style for Seaborn plots

sns.set(style="whitegrid")

# Example: Histogram for SedentaryMinutes in Cleaned Data

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

sns.histplot(cleaned\_df\_intensities['SedentaryMinutes'], bins=30, kde=True, color='lightgreen')

plt.title('Distribution of SedentaryMinutes (Cleaned Data)')

plt.xlabel('SedentaryMinutes')

plt.ylabel('Frequency')

plt.savefig('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/VisualizationImages/dailyIntensities\_merged\_Filtered/SedentaryMinutes\_distribution\_cleaned.png')  # Save the figure

plt.show()

# Example: Boxplot for SedentaryMinutes in Cleaned Data

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

sns.boxplot(x=cleaned\_df\_intensities['SedentaryMinutes'], color='lightblue')

plt.title('Box Plot of SedentaryMinutes (Cleaned Data)')

plt.xlabel('SedentaryMinutes')

plt.savefig('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/VisualizationImages/dailyIntensities\_merged\_Filtered/SedentaryMinutes\_boxplot\_cleaned.png')  # Save the figure

plt.show()

#### **dailySteps\_merged\_Filtered.ipynb:**

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Load the cleaned dataset

cleaned\_df\_steps = pd.read\_csv('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/FilteredFitbaseData/dailySteps\_merged\_Filtered.csv')

# Set the style for Seaborn plots

sns.set(style="whitegrid")

# Example: Histogram for StepTotal in Cleaned Data

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

sns.histplot(cleaned\_df\_steps['StepTotal'], bins=30, kde=True, color='lightcoral')

plt.title('Distribution of StepTotal (Cleaned Data)')

plt.xlabel('StepTotal')

plt.ylabel('Frequency')

plt.savefig('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/VisualizationImages/dailySteps\_merged\_Filtered/StepTotal\_distribution\_cleaned.png')  # Save the figure

plt.show()

# Example: Boxplot for StepTotal in Cleaned Data

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

sns.boxplot(x=cleaned\_df\_steps['StepTotal'], color='lightskyblue')

plt.title('Box Plot of StepTotal (Cleaned Data)')

plt.xlabel('StepTotal')

plt.savefig('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/VisualizationImages/dailySteps\_merged\_Filtered/StepTotal\_boxplot\_cleaned.png')  # Save the figure

plt.show()

#### **heartrate\_seconds\_merged\_Filtered.ipynb:**

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Load the cleaned dataset

cleaned\_df\_heartrate = pd.read\_csv('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/FilteredFitbaseData/heartrate\_seconds\_merged\_Filtered.csv')

# Set the style for Seaborn plots

sns.set(style="whitegrid")

# Example: Line plot for Value in Cleaned Data

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

sns.lineplot(x=cleaned\_df\_heartrate.index, y=cleaned\_df\_heartrate['Value'], color='salmon')

plt.title('Heart Rate Over Time (Cleaned Data)')

plt.xlabel('Time (seconds)')

plt.ylabel('Heart Rate')

plt.savefig('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/VisualizationImages/heartrate\_seconds\_merged\_Filtered/HeartRate\_over\_time\_cleaned.png')  # Save the figure

plt.show()

#### **hourlyCalories\_merged\_Filtered.ipynb:**

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Load the cleaned dataset

cleaned\_df\_hourly\_calories = pd.read\_csv('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/FilteredFitbaseData/hourlyCalories\_merged\_Filtered.csv')

# Set the style for Seaborn plots

sns.set(style="whitegrid")

# Example: Line plot for Calories in Cleaned Data

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

sns.lineplot(x=cleaned\_df\_hourly\_calories['ActivityHour'], y=cleaned\_df\_hourly\_calories['Calories'], color='lightseagreen')

plt.title('Hourly Calories Burned (Cleaned Data)')

plt.xlabel('Hour of the Day')

plt.ylabel('Calories Burned')

plt.savefig('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/VisualizationImages/hourlyCalories\_merged\_Filtered/Hourly\_Calories\_burned\_cleaned.png')  # Save the figure

plt.show()

#### **hourlyIntensities\_merged\_Filtered.ipynb:**

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Load the cleaned dataset

cleaned\_df\_hourly\_intensities = pd.read\_csv('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/FilteredFitbaseData/hourlyIntensities\_merged\_Filtered.csv')

# Set the style for Seaborn plots

sns.set(style="whitegrid")

# Example: Line plot for TotalIntensity and AverageIntensity in Cleaned Data

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

sns.lineplot(x=cleaned\_df\_hourly\_intensities['ActivityHour'], y=cleaned\_df\_hourly\_intensities['TotalIntensity'], label='Total Intensity', color='orange')

sns.lineplot(x=cleaned\_df\_hourly\_intensities['ActivityHour'], y=cleaned\_df\_hourly\_intensities['AverageIntensity'], label='Average Intensity', color='lightblue')

plt.title('Hourly Intensity (Cleaned Data)')

plt.xlabel('Hour of the Day')

plt.ylabel('Intensity')

plt.legend()

plt.savefig('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/VisualizationImages/hourlyIntensities\_merged\_Filtered/Hourly\_Intensity\_cleaned.png')  # Save the figure

plt.show()

#### **hourlySteps\_merged\_Filtered.ipynb:**

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Load the cleaned dataset

cleaned\_df\_hourly\_steps = pd.read\_csv('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/FilteredFitbaseData/hourlySteps\_merged\_Filtered.csv')

# Set the style for Seaborn plots

sns.set(style="whitegrid")

# Example: Line plot for StepTotal in Cleaned Data

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

sns.lineplot(x=cleaned\_df\_hourly\_steps['ActivityHour'], y=cleaned\_df\_hourly\_steps['StepTotal'], color='skyblue')

plt.title('Hourly Step Total (Cleaned Data)')

plt.xlabel('Hour of the Day')

plt.ylabel('Step Total')

plt.savefig('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/VisualizationImages/hourlySteps\_merged\_Filtered/Hourly\_Step\_Total\_cleaned.png')  # Save the figure

plt.show()

#### **minuteCaloriesNarrow\_merged\_Filtered.ipynb:**

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Load the cleaned dataset

cleaned\_df\_minute\_calories\_narrow = pd.read\_csv('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/FilteredFitbaseData/minuteCaloriesNarrow\_merged\_Filtered.csv')

# Specify the features you want to visualize

selected\_features = [

    'Calories'

]

# Set the style for Seaborn plots

sns.set(style="whitegrid")

# Visualize boxplots for selected features

plt.figure(figsize=(15, 10))

for i, column in enumerate(selected\_features):

    plt.subplot(3, 5, i+1)

    sns.boxplot(x=cleaned\_df\_minute\_calories\_narrow[column])

    plt.title(f'Box Plot of {column}')

plt.tight\_layout()

plt.savefig('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/VisualizationImages/minuteCaloriesNarrow\_merged\_Filtered/Boxplots\_selected\_features\_cleaned.png')  # Save the figure

plt.show()

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Load the cleaned dataset

cleaned\_df\_minute\_calories\_narrow = pd.read\_csv('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/FilteredFitbaseData/minuteCaloriesNarrow\_merged\_Filtered.csv')

# Specify the features you want to visualize

selected\_features = ['Calories']

# Set the style for Seaborn plots

sns.set(style="whitegrid")

# Create subplots

fig, axes = plt.subplots(2, 2, figsize=(15, 10))

# Visualize boxplots for selected features

sns.boxplot(x=cleaned\_df\_minute\_calories\_narrow['Calories'], ax=axes[0, 0])

axes[0, 0].set\_title('Box Plot of Calories')

# Visualize histogram for selected features

sns.histplot(x=cleaned\_df\_minute\_calories\_narrow['Calories'], ax=axes[0, 1], bins=20, kde=True)

axes[0, 1].set\_title('Histogram of Calories')

# Visualize violin plot for selected features

sns.violinplot(x=cleaned\_df\_minute\_calories\_narrow['Calories'], ax=axes[1, 0])

axes[1, 0].set\_title('Violin Plot of Calories')

# Visualize swarm plot for selected features

sns.swarmplot(x=cleaned\_df\_minute\_calories\_narrow['Calories'], ax=axes[1, 1])

axes[1, 1].set\_title('Swarm Plot of Calories')

plt.tight\_layout()

plt.savefig('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/VisualizationImages/minuteCaloriesNarrow\_merged\_Filtered/Visualizations\_selected\_features\_cleaned.png')  # Save the figure

plt.show()

#### **minuteCaloriesWide\_merged\_Filtered:**

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Load the cleaned dataset

cleaned\_df\_minute\_calories\_wide = pd.read\_csv('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/FilteredFitbaseData/minuteCaloriesWide\_merged\_Filtered.csv')

# Specify the features you want to visualize

selected\_features = ['Calories00', 'Calories01', 'Calories02', 'Calories03', 'Calories04', 'Calories05', 'Calories06', 'Calories07', 'Calories08', 'Calories09', 'Calories10', 'Calories11', 'Calories12', 'Calories13', 'Calories14', 'Calories15', 'Calories16', 'Calories17', 'Calories18', 'Calories19', 'Calories20', 'Calories21', 'Calories22', 'Calories23', 'Calories24', 'Calories25', 'Calories26', 'Calories27', 'Calories28', 'Calories29', 'Calories30', 'Calories31', 'Calories32', 'Calories33', 'Calories34', 'Calories35', 'Calories36', 'Calories37', 'Calories38', 'Calories39', 'Calories40', 'Calories41', 'Calories42', 'Calories43', 'Calories44', 'Calories45', 'Calories46', 'Calories47', 'Calories48', 'Calories49', 'Calories50', 'Calories51', 'Calories52', 'Calories53', 'Calories54', 'Calories55', 'Calories56', 'Calories57', 'Calories58', 'Calories59']

# Set the style for Seaborn plots

sns.set(style="whitegrid")

# Create subplots

fig, axes = plt.subplots(12, 5, figsize=(20, 30))

# Visualize boxplots for selected features

for i, column in enumerate(selected\_features):

    sns.boxplot(x=cleaned\_df\_minute\_calories\_wide[column], ax=axes[i//5, i%5])

    axes[i//5, i%5].set\_title(f'Box Plot of {column}')

plt.tight\_layout()

plt.savefig('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/VisualizationImages/minuteCaloriesWide\_merged\_Filtered/Boxplots\_selected\_features\_cleaned.png')  # Save the figure

plt.show()

#### **minuteIntensitiesNarrow\_merged.ipynb:**

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Load the cleaned dataset

cleaned\_df\_minute\_intensities\_narrow = pd.read\_csv('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/FilteredFitbaseData/minuteIntensitiesNarrow\_merged\_Filtered.csv')

# Set the style for Seaborn plots

sns.set(style="whitegrid")

# Example: Histogram for Intensity

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

sns.histplot(cleaned\_df\_minute\_intensities\_narrow['Intensity'], bins=30, kde=True, color='skyblue')

plt.title('Distribution of Intensity')

plt.xlabel('Intensity')

plt.ylabel('Frequency')

plt.savefig('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/VisualizationImages/minuteIntensitiesNarrow\_merged\_Filtered/Intensity\_distribution.png')  # Save the figure

plt.show()

# Example: Boxplot for Intensity

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

sns.boxplot(x=cleaned\_df\_minute\_intensities\_narrow['Intensity'], color='lightcoral')

plt.title('Box Plot of Intensity')

plt.xlabel('Intensity')

plt.savefig('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/VisualizationImages/minuteIntensitiesNarrow\_merged\_Filtered/Intensity\_BoxPlot.png')  # Save the figure

plt.show()

# Example: Scatter plot for Intensity and another feature (replace 'Id' with the actual feature name)

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

sns.scatterplot(x='Intensity', y='Id', data=cleaned\_df\_minute\_intensities\_narrow, color='salmon')

plt.title('Scatter Plot of Intensity vs Id')

plt.xlabel('Intensity')

plt.ylabel('Id')

plt.savefig('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/VisualizationImages/minuteIntensitiesNarrow\_merged\_Filtered/Intensity&Id\_ScatterPlot.png')  # Save the figure

plt.show()

#### **minuteIntensitiesWide\_merged\_Filtered.ipynb:**

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Load the cleaned dataset

cleaned\_df\_minute\_intensities\_wide = pd.read\_csv('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/FilteredFitbaseData/minuteIntensitiesWide\_merged\_Filtered.csv')

# Set the style for Seaborn plots

sns.set(style="whitegrid")

# Example: Histogram for Intensity00

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

sns.histplot(cleaned\_df\_minute\_intensities\_wide['Intensity00'], bins=30, kde=True, color='skyblue')

plt.title('Distribution of Intensity00')

plt.xlabel('Intensity00')

plt.ylabel('Frequency')

plt.savefig('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/VisualizationImages/minuteIntensitiesWide\_merged\_Filtered/Intensity00\_distribution.png')  # Save the figure

plt.show()

# Repeat the above code for other features (Intensity01, Intensity02, ..., Intensity59) by replacing 'Intensity00' with the corresponding feature names.

# Remember to update the file paths for saving images.

# Example: Boxplot for Intensity00

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

sns.boxplot(x=cleaned\_df\_minute\_intensities\_wide['Intensity00'], color='lightcoral')

plt.title('Box Plot of Intensity00')

plt.xlabel('Intensity00')

plt.savefig('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/VisualizationImages/minuteIntensitiesWide\_merged\_Filtered/Intensity00\_BoxPlot.png')  # Save the figure

plt.show()

# Example: Scatter plot for Intensity00 and another feature (replace 'Intensity59' with the actual feature name)

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

sns.scatterplot(x='Intensity00', y='Intensity59', data=cleaned\_df\_minute\_intensities\_wide, color='salmon')

plt.title('Scatter Plot of Intensity00 vs Intensity59')

plt.xlabel('Intensity00')

plt.ylabel('Intensity59')

plt.savefig('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/VisualizationImages/minuteIntensitiesWide\_merged\_Filtered/Intensity00&Intensity59\_ScatterPlot.png')  # Save the figure

plt.show()

#### **minuteMETsNarrow\_merged\_Filtered.ipynb:**

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Load the cleaned dataset

cleaned\_df\_minute\_mets\_narrow = pd.read\_csv('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/FilteredFitbaseData/minuteMETsNarrow\_merged\_Filtered.csv')

# Set the style for Seaborn plots

sns.set(style="whitegrid")

# Example: Histogram for METs

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

sns.histplot(cleaned\_df\_minute\_mets\_narrow['METs'], bins=30, kde=True, color='skyblue')

plt.title('Distribution of METs')

plt.xlabel('METs')

plt.ylabel('Frequency')

plt.savefig('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/VisualizationImages/minuteMETsNarrow\_merged\_Filtered/METs\_distribution.png')  # Save the figure

plt.show()

# Example: Boxplot for METs

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

sns.boxplot(x=cleaned\_df\_minute\_mets\_narrow['METs'], color='lightcoral')

plt.title('Box Plot of METs')

plt.xlabel('METs')

plt.savefig('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/VisualizationImages/minuteMETsNarrow\_merged\_Filtered/METs\_BoxPlot.png')  # Save the figure

plt.show()

# Example: Scatter plot for METs and another feature (replace 'ActivityMinute' with the actual feature name)

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

sns.scatterplot(x='METs', y='ActivityMinute', data=cleaned\_df\_minute\_mets\_narrow, color='salmon')

plt.title('Scatter Plot of METs vs ActivityMinute')

plt.xlabel('METs')

plt.ylabel('ActivityMinute')

plt.savefig('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/VisualizationImages/minuteMETsNarrow\_merged\_Filtered/METs&ActivityMinute\_ScatterPlot.png')  # Save the figure

plt.show()

#### **minuteSleep\_merged\_Filtered.ipynb:**

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Load the cleaned dataset

cleaned\_df\_minute\_sleep = pd.read\_csv('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/FilteredFitbaseData/minuteSleep\_merged\_Filtered.csv')

# Set the style for Seaborn plots

sns.set(style="whitegrid")

# Example: Histogram for 'value'

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

sns.histplot(cleaned\_df\_minute\_sleep['value'], bins=30, kde=True, color='skyblue')

plt.title('Distribution of value')

plt.xlabel('value')

plt.ylabel('Frequency')

plt.savefig('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/VisualizationImages/minuteSleep\_merged\_Filtered/value\_distribution.png')  # Save the figure

plt.show()

# Example: Boxplot for 'value'

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

sns.boxplot(x=cleaned\_df\_minute\_sleep['value'], color='lightcoral')

plt.title('Box Plot of value')

plt.xlabel('value')

plt.savefig('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/VisualizationImages/minuteSleep\_merged\_Filtered/value\_BoxPlot.png')  # Save the figure

plt.show()

# Example: Histogram for 'logid'

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

sns.histplot(cleaned\_df\_minute\_sleep['logid'], bins=30, kde=True, color='skyblue')

plt.title('Distribution of logid')

plt.xlabel('logid')

plt.ylabel('Frequency')

plt.savefig('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/VisualizationImages/minuteSleep\_merged\_Filtered/logid\_distribution.png')  # Save the figure

plt.show()

# Example: Boxplot for 'logid'

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

sns.boxplot(x=cleaned\_df\_minute\_sleep['logid'], color='lightcoral')

plt.title('Box Plot of logid')

plt.xlabel('logid')

plt.savefig('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/VisualizationImages/minuteSleep\_merged\_Filtered/logid\_BoxPlot.png')  # Save the figure

plt.show()

#### **minuteStepsNarrow\_merged\_Filtered.ipynb:**

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Load the cleaned dataset

cleaned\_df\_minute\_steps\_narrow = pd.read\_csv('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/FilteredFitbaseData/minuteStepsNarrow\_merged\_Filtered.csv')

# Set the style for Seaborn plots

sns.set(style="whitegrid")

# Example: Histogram for 'Steps'

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

sns.histplot(cleaned\_df\_minute\_steps\_narrow['Steps'], bins=30, kde=True, color='skyblue')

plt.title('Distribution of Steps')

plt.xlabel('Steps')

plt.ylabel('Frequency')

plt.savefig('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/VisualizationImages/minuteStepsNarrow\_merged\_Filtered/Steps\_distribution.png')  # Save the figure

plt.show()

# Example: Boxplot for 'Steps'

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

sns.boxplot(x=cleaned\_df\_minute\_steps\_narrow['Steps'], color='lightcoral')

plt.title('Box Plot of Steps')

plt.xlabel('Steps')

plt.savefig('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/VisualizationImages/minuteStepsNarrow\_merged\_Filtered/Steps\_BoxPlot.png')  # Save the figure

plt.show()

# Example: Histogram for 'ActivityMinute'

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

sns.histplot(cleaned\_df\_minute\_steps\_narrow['ActivityMinute'], bins=30, kde=True, color='skyblue')

plt.title('Distribution of ActivityMinute')

plt.xlabel('ActivityMinute')

plt.ylabel('Frequency')

plt.savefig('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/VisualizationImages/minuteStepsNarrow\_merged\_Filtered/ActivityMinute\_distribution.png')  # Save the figure

plt.show()

# Example: Boxplot for 'ActivityMinute'

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

sns.boxplot(x=cleaned\_df\_minute\_steps\_narrow['ActivityMinute'], color='lightcoral')

plt.title('Box Plot of ActivityMinute')

plt.xlabel('ActivityMinute')

plt.savefig('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/VisualizationImages/minuteStepsNarrow\_merged\_Filtered/ActivityMinute\_BoxPlot.png')  # Save the figure

plt.show()

#### **minuteStepsWide\_merged\_Filtered.ipynb:**

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Load the cleaned dataset

cleaned\_df\_minute\_steps\_wide = pd.read\_csv('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/FilteredFitbaseData/minuteStepsWide\_merged\_Filtered.csv')

# Set the style for Seaborn plots

sns.set(style="whitegrid")

# Example: Histogram for 'Hour'

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

sns.histplot(cleaned\_df\_minute\_steps\_wide['Hour'], bins=30, kde=True, color='skyblue')

plt.title('Distribution of Hour')

plt.xlabel('Hour')

plt.ylabel('Frequency')

plt.savefig('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/VisualizationImages/minuteStepsWide\_merged\_Filtered/Hour\_distribution.png')  # Save the figure

plt.show()

# Example: Boxplot for 'Hour'

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

sns.boxplot(x=cleaned\_df\_minute\_steps\_wide['Hour'], color='lightcoral')

plt.title('Box Plot of Hour')

plt.xlabel('Hour')

plt.savefig('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/VisualizationImages/minuteStepsWide\_merged\_Filtered/Hour\_BoxPlot.png')  # Save the figure

plt.show()

# Example: Histogram for 'Steps'

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

sns.histplot(cleaned\_df\_minute\_steps\_wide['Steps'], bins=30, kde=True, color='skyblue')

plt.title('Distribution of Steps')

plt.xlabel('Steps')

plt.ylabel('Frequency')

plt.savefig('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/VisualizationImages/minuteStepsWide\_merged\_Filtered/Steps\_distribution.png')  # Save the figure

plt.show()

# Example: Boxplot for 'Steps'

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

sns.boxplot(x=cleaned\_df\_minute\_steps\_wide['Steps'], color='lightcoral')

plt.title('Box Plot of Steps')

plt.xlabel('Steps')

plt.savefig('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/VisualizationImages/minuteStepsWide\_merged\_Filtered/Steps\_BoxPlot.png')  # Save the figure

plt.show()

#### **sleepDay\_merged\_Filtered.ipynb:**

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Load the cleaned dataset

cleaned\_df\_sleep\_day = pd.read\_csv('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/FilteredFitbaseData/sleepDay\_merged\_Filtered.csv')

# Set the style for Seaborn plots

sns.set(style="whitegrid")

# Example: Histogram for 'TotalSleepRecords'

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

sns.histplot(cleaned\_df\_sleep\_day['TotalSleepRecords'], bins=30, kde=True, color='skyblue')

plt.title('Distribution of TotalSleepRecords')

plt.xlabel('TotalSleepRecords')

plt.ylabel('Frequency')

plt.savefig('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/VisualizationImages/sleepDay\_merged\_Filtered/TotalSleepRecords\_distribution.png')  # Save the figure

plt.show()

# Example: Boxplot for 'TotalSleepRecords'

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

sns.boxplot(x=cleaned\_df\_sleep\_day['TotalSleepRecords'], color='lightcoral')

plt.title('Box Plot of TotalSleepRecords')

plt.xlabel('TotalSleepRecords')

plt.savefig('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/VisualizationImages/sleepDay\_merged\_Filtered/TotalSleepRecords\_BoxPlot.png')  # Save the figure

plt.show()

# Example: Histogram for 'TotalMinutesAsleep'

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

sns.histplot(cleaned\_df\_sleep\_day['TotalMinutesAsleep'], bins=30, kde=True, color='skyblue')

plt.title('Distribution of TotalMinutesAsleep')

plt.xlabel('TotalMinutesAsleep')

plt.ylabel('Frequency')

plt.savefig('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/VisualizationImages/sleepDay\_merged\_Filtered/TotalMinutesAsleep\_distribution.png')  # Save the figure

plt.show()

# Example: Boxplot for 'TotalMinutesAsleep'

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

sns.boxplot(x=cleaned\_df\_sleep\_day['TotalMinutesAsleep'], color='lightcoral')

plt.title('Box Plot of TotalMinutesAsleep')

plt.xlabel('TotalMinutesAsleep')

plt.savefig('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/VisualizationImages/sleepDay\_merged\_Filtered/TotalMinutesAsleep\_BoxPlot.png')  # Save the figure

plt.show()

# Example: Histogram for 'TotalTimeInBed'

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

sns.histplot(cleaned\_df\_sleep\_day['TotalTimeInBed'], bins=30, kde=True, color='skyblue')

plt.title('Distribution of TotalTimeInBed')

plt.xlabel('TotalTimeInBed')

plt.ylabel('Frequency')

plt.savefig('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/VisualizationImages/sleepDay\_merged\_Filtered/TotalTimeInBed\_distribution.png')  # Save the figure

plt.show()

# Example: Boxplot for 'TotalTimeInBed'

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

sns.boxplot(x=cleaned\_df\_sleep\_day['TotalTimeInBed'], color='lightcoral')

plt.title('Box Plot of TotalTimeInBed')

plt.xlabel('TotalTimeInBed')

plt.savefig('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/VisualizationImages/sleepDay\_merged\_Filtered/TotalTimeInBed\_BoxPlot.png')  # Save the figure

plt.show()

#### **weightLogInfo\_merged\_Filtered.ipynb:**

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Load the cleaned dataset

cleaned\_df\_weight\_log = pd.read\_csv('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/FilteredFitbaseData/weightLogInfo\_merged\_Filtered.csv')

# Set the style for Seaborn plots

sns.set(style="whitegrid")

# Example: Histogram for 'WeightKg'

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

sns.histplot(cleaned\_df\_weight\_log['WeightKg'], bins=30, kde=True, color='skyblue')

plt.title('Distribution of WeightKg')

plt.xlabel('WeightKg')

plt.ylabel('Frequency')

plt.savefig('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/VisualizationImages/weightLogInfo\_merged\_Filtered/WeightKg\_distribution.png')  # Save the figure

plt.show()

# Example: Boxplot for 'WeightKg'

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

sns.boxplot(x=cleaned\_df\_weight\_log['WeightKg'], color='lightcoral')

plt.title('Box Plot of WeightKg')

plt.xlabel('WeightKg')

plt.savefig('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/VisualizationImages/weightLogInfo\_merged\_Filtered/WeightKg\_BoxPlot.png')  # Save the figure

plt.show()

# Example: Histogram for 'WeightPounds'

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

sns.histplot(cleaned\_df\_weight\_log['WeightPounds'], bins=30, kde=True, color='skyblue')

plt.title('Distribution of WeightPounds')

plt.xlabel('WeightPounds')

plt.ylabel('Frequency')

plt.savefig('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/VisualizationImages/weightLogInfo\_merged\_Filtered/WeightPounds\_distribution.png')  # Save the figure

plt.show()

# Example: Boxplot for 'WeightPounds'

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

sns.boxplot(x=cleaned\_df\_weight\_log['WeightPounds'], color='lightcoral')

plt.title('Box Plot of WeightPounds')

plt.xlabel('WeightPounds')

plt.savefig('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/VisualizationImages/weightLogInfo\_merged\_Filtered/WeightPounds\_BoxPlot.png')  # Save the figure

plt.show()

## **AI-Driven Recommendation:**

### **RecommendationSystem:**

#### **buildingRecommendationModel.ipynb:**

import pandas as pd

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

from sklearn.decomposition import NMF

import joblib

preprocessed\_data\_path = '/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/MergedPreprocessedData/PartiallyProcessedData04.csv'

preprocessed\_data = pd.read\_csv(preprocessed\_data\_path)

# Split the data into train and test sets

train\_data, test\_data = train\_test\_split(preprocessed\_data, test\_size=0.2, random\_state=42)

# Choosing features for training

features = ['Id', 'ActivityDate', 'TotalSteps', 'ActivityDay\_x', 'Calories\_y', 'ActivityDay\_y', 'Rating']

# Using pivot\_table with an aggregation function

X\_train = train\_data.pivot\_table(index='Id', columns='ActivityDate', values='Rating', aggfunc='mean', fill\_value=0)

# Training the model

model = NMF(n\_components=10, init='random', random\_state=42)

model.fit(X\_train)

# Save the recommendation model

model\_save\_path = '/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Models/RecommendationModels/recommendationModel.pkl'

joblib.dump(model, model\_save\_path)

#### **makingRecommendation.ipynb:**

import pandas as pd

import joblib

from sklearn.metrics.pairwise import cosine\_similarity

preprocessed\_data\_path = '/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/MergedPreprocessedData/PartiallyProcessedData05.csv'

preprocessed\_data = pd.read\_csv(preprocessed\_data\_path)

user\_id = 590

# Checking if the target user exists in the dataset

if user\_id not in preprocessed\_data['Id'].values:

    raise ValueError(f"User with ID {user\_id} not found in the dataset.")

# Get the features of the target user

target\_user\_features = preprocessed\_data[preprocessed\_data['Id'] == user\_id][['TotalSteps', 'Rating']].values

# Ensure that target\_user\_features is a 2D array with at least one feature

if target\_user\_features.shape[0] == 0:

    raise ValueError("Target user has zero features.")

# Load the trained recommendation model

model\_path = '/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Models/RecommendationModels/recommendationModel.pkl'

model = joblib.load(model\_path)

# Calculate cosine similarity between the target user and all other users

preprocessed\_data['Similarity'] = preprocessed\_data.apply(lambda row: cosine\_similarity(target\_user\_features, row[['TotalSteps', 'Rating']].values.reshape(1, -1))[0][0], axis=1)

# Get top N similar users

top\_n\_similar\_users = preprocessed\_data.nlargest(5, 'Similarity')

# Get the most rated items by the top N similar users

most\_rated\_items = top\_n\_similar\_users.groupby('ActivityDay\_y')['Rating'].mean().sort\_values(ascending=False).head(10)

# Print the recommendations

print(f"Top 10 recommendations for user {user\_id}: {most\_rated\_items.index.tolist()}")

print(preprocessed\_data['Id'].unique())

import pandas as pd

def set\_daily\_step\_goal(user\_id, goal\_multiplier=1.1):

    """

    Set a daily step goal for a user based on historical data.

    Parameters:

    - user\_id: ID of the user for whom the goal is being set.

    - goal\_multiplier: A multiplier to adjust the goal based on fitness objectives.

    Returns:

    - Recommended daily step goal for the user.

    """

    # Filter data for the specific user

    user\_data = preprocessed\_data[preprocessed\_data['Id'] == user\_id]

    # Check if the user exists in the dataset

    if user\_data.empty:

        raise ValueError(f"User with ID {user\_id} not found in the dataset.")

    # Print user\_data for debugging

    print(f"User Data for User {user\_id}:\n{user\_data}")

    # Calculate the average daily steps for the user

    avg\_daily\_steps = user\_data['TotalSteps'].mean()

    # Print average daily steps for debugging

    print(f"Avg Daily Steps for User {user\_id}: {avg\_daily\_steps}")

    # Set the daily step goal based on the average and the goal multiplier

    daily\_step\_goal = int(avg\_daily\_steps \* goal\_multiplier)

    return daily\_step\_goal

# Example: Set daily step goal for user with ID 590

user\_id = 590

goal = set\_daily\_step\_goal(user\_id)

print(f"Recommended daily step goal for User {user\_id}: {goal} steps")

import pandas as pd

import numpy as np

def suggest\_activity\_dates(user\_id, intensity\_threshold=0.8, rating\_threshold=8, calorie\_threshold=300):

    """

    Suggest specific dates for high-intensity workouts or activities based on the user's historical patterns.

    Parameters:

    - user\_id: ID of the user for whom activity dates are being suggested.

    - intensity\_threshold: Threshold for activity intensity to be considered high-intensity.

    - rating\_threshold: Threshold for activity rating to be considered high-rated.

    - calorie\_threshold: Threshold for calorie burn to be considered high.

    Returns:

    - List of suggested dates for high-intensity activities.

    """

    # Filter data for the specific user

    user\_data = preprocessed\_data[preprocessed\_data['Id'] == user\_id]

    # Check if the user exists in the dataset

    if user\_data.empty:

        raise ValueError(f"User with ID {user\_id} not found in the dataset.")

    # Filter data based on intensity, rating, and calorie thresholds

    high\_intensity\_data = user\_data[user\_data['Similarity'] > intensity\_threshold]

    high\_rating\_data = user\_data[user\_data['Rating'] >= rating\_threshold]

    high\_calorie\_data = user\_data[user\_data['Calories\_y'] > calorie\_threshold]

    # Find common dates among high-intensity, high-rating, and high-calorie data

    suggested\_dates = set(high\_intensity\_data['ActivityDate']).intersection(

        set(high\_rating\_data['ActivityDate']).intersection(set(high\_calorie\_data['ActivityDate']))

    )

    return list(suggested\_dates)

# Example: Suggest activity dates for user with ID 590

user\_id = 590

suggested\_dates = suggest\_activity\_dates(user\_id)

print(f"Suggested activity dates for User {user\_id}: {suggested\_dates}")

import pandas as pd

def caloric\_intake\_and\_burn\_recommendations(user\_id, goal='weight\_loss'):

    """

    Provide dietary recommendations based on the user's calorie burn and intake patterns.

    Parameters:

    - user\_id: ID of the user for whom dietary recommendations are being suggested.

    - goal: Fitness goal, options include 'weight\_loss', 'maintenance', or 'muscle\_gain'.

    Returns:

    - Dietary recommendations based on the user's fitness goal.

    """

    # Filter data for the specific user

    user\_data = preprocessed\_data[preprocessed\_data['Id'] == user\_id]

    # Check if the user exists in the dataset

    if user\_data.empty:

        raise ValueError(f"User with ID {user\_id} not found in the dataset.")

    # Calculate total calorie burn and intake

    total\_calorie\_burn = user\_data['Calories\_y'].sum()

    total\_calorie\_intake = user\_data['Calories\_y'].sum()

    # Calculate net calorie balance (caloric deficit or surplus)

    net\_calorie\_balance = total\_calorie\_intake - total\_calorie\_burn

    # Define dietary recommendations based on fitness goals

    if goal == 'weight\_loss':

        if net\_calorie\_balance < 0:

            recommendation = "You are on track for weight loss. Continue maintaining a caloric deficit."

        else:

            recommendation = "Consider adjusting your caloric intake to create a caloric deficit for weight loss."

    elif goal == 'maintenance':

        recommendation = "Your caloric intake and burn seem balanced. Maintain your current dietary habits."

    elif goal == 'muscle\_gain':

        if net\_calorie\_balance > 0:

            recommendation = "You are on track for muscle gain. Continue maintaining a caloric surplus."

        else:

            recommendation = "Consider adjusting your caloric intake to create a caloric surplus for muscle gain."

    else:

        raise ValueError("Invalid fitness goal. Choose from 'weight\_loss', 'maintenance', or 'muscle\_gain'.")

    return recommendation

# Example: Provide caloric intake and burn recommendations for user with ID 590

user\_id = 590

fitness\_goal = 'weight\_loss'

recommendation = caloric\_intake\_and\_burn\_recommendations(user\_id, goal=fitness\_goal)

print(f"Dietary recommendation for User {user\_id} for {fitness\_goal}: {recommendation}")

import pandas as pd

preprocessed\_data\_path = '/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/MergedPreprocessedData/PartiallyProcessedData05.csv'

preprocessed\_data = pd.read\_csv(preprocessed\_data\_path)

def calories\_burned\_analysis(user\_id, top\_activities=3):

    """

    Provide insights into calories burned during different activities.

    Parameters:

    - user\_id: ID of the user for whom calorie burn analysis is being performed.

    - top\_activities: Number of top activities to recommend.

    Returns:

    - Personalized recommendations for effective workouts based on calories burned.

    """

    # Filter data for the specific user

    user\_data = preprocessed\_data[preprocessed\_data['Id'] == user\_id]

    # Check if the user exists in the dataset

    if user\_data.empty:

        raise ValueError(f"User with ID {user\_id} not found in the dataset.")

    # Group data by activity type and calculate the total calories burned for each activity

    activity\_calories = user\_data.groupby('ActivityType')['Calories\_y'].sum().reset\_index()

    # Sort activities by total calories burned in descending order

    sorted\_activities = activity\_calories.sort\_values(by='Calories\_y', ascending=False)

    # Select the top activities

    top\_activities\_list = sorted\_activities.head(top\_activities)['ActivityType'].tolist()

    # Create personalized recommendations

    recommendations = f"For effective calorie burning, consider the following top {top\_activities} activities:\n"

    for activity in top\_activities\_list:

        recommendations += f"- {activity}\n"

    return recommendations

# Example: Provide calories burned analysis for user with ID 590

user\_id = 590

top\_activities\_recommendations = calories\_burned\_analysis(user\_id, top\_activities=3)

print(f"Calories Burned Analysis Recommendations for User {user\_id}:\n{top\_activities\_recommendations}")

import pandas as pd

def activity\_patterns\_and\_day\_analysis(user\_id):

    """

    Identify patterns in activity days and correlate with user ratings.

    Parameters:

    - user\_id: ID of the user for whom activity patterns are being analyzed.

    Returns:

    - Analysis results and recommendations.

    """

    # Filter data for the specific user

    user\_data = preprocessed\_data[preprocessed\_data['Id'] == user\_id]

    # Check if the user exists in the dataset

    if user\_data.empty:

        raise ValueError(f"User with ID {user\_id} not found in the dataset.")

    # Analyze activity patterns and user ratings

    activity\_day\_analysis = user\_data.groupby(['ActivityDay\_x', 'ActivityDay\_y'])['Rating'].mean().reset\_index()

    # Identify days or activities associated with higher user ratings

    high\_rating\_days = activity\_day\_analysis[activity\_day\_analysis['Rating'] >= 8]

    # Create analysis summary

    analysis\_summary = f"Activity Patterns and Day Analysis for User {user\_id}:\n"

    analysis\_summary += "----------------------------------------------\n"

    if not high\_rating\_days.empty:

        analysis\_summary += "Days or activities associated with higher user ratings:\n"

        analysis\_summary += high\_rating\_days.to\_string(index=False) + "\n"

    else:

        analysis\_summary += "No specific days or activities associated with higher user ratings.\n"

    return analysis\_summary

# Example: Perform activity patterns and day analysis for user with ID 590

user\_id = 590

activity\_day\_analysis\_results = activity\_patterns\_and\_day\_analysis(user\_id)

print(activity\_day\_analysis\_results)

import pandas as pd

def rating\_based\_recommendations(user\_id, top\_activities=3):

    """

    Provide recommendations based on user ratings.

    Parameters:

    - user\_id: ID of the user for whom recommendations are being suggested.

    - top\_activities: Number of top activities to recommend.

    Returns:

    - Personalized recommendations based on highly rated days.

    """

    # Filter data for the specific user

    user\_data = preprocessed\_data[preprocessed\_data['Id'] == user\_id]

    # Check if the user exists in the dataset

    if user\_data.empty:

        raise ValueError(f"User with ID {user\_id} not found in the dataset.")

    # Analyze ratings and identify highly rated activities

    high\_rated\_activities = user\_data[user\_data['Rating'] >= 8]

    # Group data by activity type and calculate the count of highly rated activities

    highly\_rated\_counts = high\_rated\_activities.groupby('ActivityType')['Rating'].count().reset\_index()

    # Sort activities by count in descending order

    sorted\_activities = highly\_rated\_counts.sort\_values(by='Rating', ascending=False)

    # Select the top activities

    top\_activities\_list = sorted\_activities.head(top\_activities)['ActivityType'].tolist()

    # Create personalized recommendations

    recommendations = f"For highly rated days, consider the following top {top\_activities} activities:\n"

    for activity in top\_activities\_list:

        recommendations += f"- {activity}\n"

    return recommendations

# Example: Provide rating-based recommendations for user with ID 590

user\_id = 590

top\_activities\_recommendations = rating\_based\_recommendations(user\_id, top\_activities=3)

print(f"Rating-Based Recommendations for User {user\_id}:\n{top\_activities\_recommendations}")

import pandas as pd

import matplotlib.pyplot as plt

def fitness\_progress\_tracking(user\_id):

    """

    Track fitness progress over time using TotalSteps.

    Parameters:

    - user\_id: ID of the user for whom progress is being tracked.

    Returns:

    - Progress visualization and insights.

    """

    # Filter data for the specific user

    user\_data = preprocessed\_data[preprocessed\_data['Id'] == user\_id]

    # Check if the user exists in the dataset

    if user\_data.empty:

        raise ValueError(f"User with ID {user\_id} not found in the dataset.")

    # Convert 'ActivityDate' to datetime for proper plotting

    user\_data['ActivityDate'] = pd.to\_datetime(user\_data['ActivityDate'])

    # Group data by date and calculate the total steps for each day

    daily\_steps = user\_data.groupby('ActivityDate')['TotalSteps'].sum().reset\_index()

    # Plot the fitness progress

    plt.figure(figsize=(10, 6))

    plt.plot(daily\_steps['ActivityDate'], daily\_steps['TotalSteps'], marker='o', linestyle='-')

    plt.title(f'Fitness Progress Tracking for User {user\_id}')

    plt.xlabel('Date')

    plt.ylabel('Total Steps')

    plt.grid(True)

    plt.show()

    # Calculate insights

    average\_daily\_steps = user\_data['TotalSteps'].mean()

    total\_steps\_increase = daily\_steps['TotalSteps'].iloc[-1] - daily\_steps['TotalSteps'].iloc[0]

    progress\_insights = (

        f"Average Daily Steps: {average\_daily\_steps:.2f}\n"

        f"Total Steps Increase: {total\_steps\_increase} steps\n"

    )

    return progress\_insights

# Example: Track fitness progress for user with ID 590

user\_id = 590

progress\_insights = fitness\_progress\_tracking(user\_id)

print(f"Fitness Progress Tracking Insights for User {user\_id}:\n{progress\_insights}")

import pandas as pd

import matplotlib.pyplot as plt

def daily\_activity\_visualization(user\_id):

    """

    Visualize daily activities for better user understanding.

    Parameters:

    - user\_id: ID of the user for whom daily activities are being visualized.

    Returns:

    - Daily activity visualizations.

    """

    # Filter data for the specific user

    user\_data = preprocessed\_data[preprocessed\_data['Id'] == user\_id]

    # Check if the user exists in the dataset

    if user\_data.empty:

        raise ValueError(f"User with ID {user\_id} not found in the dataset.")

    # Convert 'ActivityDate' to datetime for proper plotting

    user\_data['ActivityDate'] = pd.to\_datetime(user\_data['ActivityDate'])

    # Plot daily steps

    plt.figure(figsize=(10, 6))

    plt.plot(user\_data['ActivityDate'], user\_data['TotalSteps'], marker='o', linestyle='-', color='blue')

    plt.title(f'Daily Steps for User {user\_id}')

    plt.xlabel('Date')

    plt.ylabel('Total Steps')

    plt.grid(True)

    plt.show()

    # Plot calories burned

    plt.figure(figsize=(10, 6))

    plt.plot(user\_data['ActivityDate'], user\_data['Calories\_y'], marker='o', linestyle='-', color='orange')

    plt.title(f'Calories Burned for User {user\_id}')

    plt.xlabel('Date')

    plt.ylabel('Calories Burned')

    plt.grid(True)

    plt.show()

    # Plot activity patterns

    activity\_types = user\_data['ActivityType'].unique()

    plt.figure(figsize=(12, 8))

    for activity\_type in activity\_types:

        activity\_data = user\_data[user\_data['ActivityType'] == activity\_type]

        plt.plot(activity\_data['ActivityDate'], activity\_data['TotalSteps'], label=activity\_type, marker='o', linestyle='-')

    plt.title(f'Activity Patterns for User {user\_id}')

    plt.xlabel('Date')

    plt.ylabel('Total Steps')

    plt.legend()

    plt.grid(True)

    plt.show()

# Example: Visualize daily activities for user with ID 590

user\_id = 590

daily\_activity\_visualization(user\_id)

import pandas as pd

def personalized\_workout\_plan(user\_id, workout\_duration=30, workout\_intensity='moderate'):

    """

    Offer personalized workout plans based on historical data.

    Parameters:

    - user\_id: ID of the user for whom the workout plan is being suggested.

    - workout\_duration: Desired duration for each workout session (in minutes).

    - workout\_intensity: Desired workout intensity ('light', 'moderate', 'intense').

    Returns:

    - Personalized workout plan.

    """

    # Filter data for the specific user

    user\_data = preprocessed\_data[preprocessed\_data['Id'] == user\_id]

    # Check if the user exists in the dataset

    if user\_data.empty:

        raise ValueError(f"User with ID {user\_id} not found in the dataset.")

    # Calculate average daily steps and intensity

    avg\_daily\_steps = user\_data['TotalSteps'].mean()

    avg\_intensity = user\_data['Similarity'].mean()

    # Adjust workout intensity based on user's historical data

    if avg\_intensity < 0.3:

        workout\_intensity = 'light'

    elif avg\_intensity > 0.7:

        workout\_intensity = 'intense'

    # Create personalized workout plan

    workout\_plan = (

        f"Personalized Workout Plan for User {user\_id}:\n"

        f"-------------------------------------------------\n"

        f"- Workout Duration: {workout\_duration} minutes\n"

        f"- Workout Intensity: {workout\_intensity}\n"

        f"- Recommended Activities: "

    )

    # Recommend activities based on workout intensity

    if workout\_intensity == 'light':

        recommended\_activities = user\_data[user\_data['Similarity'] < 0.3]['ActivityType'].unique()[:3]

    elif workout\_intensity == 'moderate':

        recommended\_activities = user\_data[user\_data['Similarity'].between(0.3, 0.7)]['ActivityType'].unique()[:3]

    else:

        recommended\_activities = user\_data[user\_data['Similarity'] > 0.7]['ActivityType'].unique()[:3]

    workout\_plan += ', '.join(recommended\_activities)

    return workout\_plan

# Example: Offer personalized workout plan for user with ID 590

user\_id = 590

personalized\_plan = personalized\_workout\_plan(user\_id, workout\_duration=45, workout\_intensity='moderate')

print(personalized\_plan)

import pandas as pd

def health\_and\_fitness\_insights(user\_id):

    """

    Provide overall insights into health and fitness based on historical data.

    Parameters:

    - user\_id: ID of the user for whom insights are being provided.

    Returns:

    - Overall health and fitness insights.

    """

    # Filter data for the specific user

    user\_data = preprocessed\_data[preprocessed\_data['Id'] == user\_id]

    # Check if the user exists in the dataset

    if user\_data.empty:

        raise ValueError(f"User with ID {user\_id} not found in the dataset.")

    # Calculate aggregate metrics

    total\_steps = user\_data['TotalSteps'].sum()

    total\_calories\_burned = user\_data['Calories\_y'].sum()

    average\_rating = user\_data['Rating'].mean()

    # Identify achievements and areas for improvement

    achievements = []

    improvements = []

    if total\_steps > 10000:

        achievements.append("Consistently achieving over 10,000 steps daily.")

    if total\_calories\_burned > 3000:

        achievements.append("Consistently burning over 3000 calories daily.")

    if average\_rating > 7:

        achievements.append("Maintaining high average ratings for activities.")

    if total\_steps < 5000:

        improvements.append("Consider increasing daily step count for better health.")

    if total\_calories\_burned < 2000:

        improvements.append("Consider incorporating more intense activities for better calorie burn.")

    # Create overall insights summary

    insights\_summary = (

        f"Health and Fitness Insights for User {user\_id}:\n"

        f"-------------------------------------------\n"

        f"Total Steps: {total\_steps}\n"

        f"Total Calories Burned: {total\_calories\_burned}\n"

        f"Average Rating: {average\_rating:.2f}\n\n"

        f"Achievements:\n"

        f"- {', '.join(achievements) if achievements else 'No achievements.'}\n\n"

        f"Areas for Improvement:\n"

        f"- {', '.join(improvements) if improvements else 'No areas for improvement.'}\n"

    )

    return insights\_summary

# Example: Provide health and fitness insights for user with ID 590

user\_id = 590

fitness\_insights = health\_and\_fitness\_insights(user\_id)

print(fitness\_insights)

#### **MergedDataPreprocessing.ipynb:**

# Step 2: Data Preprocessing

import pandas as pd

# 1. Load Merged Dataset:

merged\_data = pd.read\_csv('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/MergedData/merged\_data.csv')

# 2. Data Preprocessing:

# Handle any remaining missing values

merged\_data.fillna(merged\_data.mean(), inplace=True)

from sklearn.preprocessing import MinMaxScaler

numerical\_columns\_to\_scale = ['TotalSteps', 'TotalDistance', 'TrackerDistance', 'LoggedActivitiesDistance', 'VeryActiveDistance\_x', 'ModeratelyActiveDistance\_x', 'LightActiveDistance\_x', 'SedentaryActiveDistance\_x', 'VeryActiveMinutes\_x', 'FairlyActiveMinutes\_x', 'LightlyActiveMinutes\_x', 'SedentaryMinutes\_x', 'Calories\_x', 'SedentaryMinutes\_y', 'LightlyActiveMinutes\_y', 'FairlyActiveMinutes\_y', 'VeryActiveMinutes\_y', 'SedentaryActiveDistance\_y', 'LightActiveDistance\_y', 'ModeratelyActiveDistance\_y', 'VeryActiveDistance\_y']

scaler = MinMaxScaler()

merged\_data[numerical\_columns\_to\_scale] = scaler.fit\_transform(merged\_data[numerical\_columns\_to\_scale])

# Feature engineering:

# Drop unnecessary columns

columns\_to\_drop = ['UnnecessaryColumn1', 'UnnecessaryColumn2']

merged\_data = merged\_data.drop(columns=columns\_to\_drop, axis=1)

# Handle outliers

column\_with\_outliers = 'OutlierColumn'

threshold = 3

merged\_data[column\_with\_outliers] = merged\_data[column\_with\_outliers].apply(lambda x: threshold if x > threshold else x)

# Other preprocessing steps...

# 3. Save Preprocessed Dataset:

preprocessed\_data\_path = '/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/MergedPreprocessedData/preProcessedMergedData.csv'

merged\_data.to\_csv(preprocessed\_data\_path, index=False)

# Step 2: Data Preprocessing

import pandas as pd

merged\_data = pd.read\_csv('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/MergedData/merged\_data.csv')

# Columns to remove

columns\_to\_remove = ['TotalDistance', 'TrackerDistance', 'LoggedActivitiesDistance',

                     'VeryActiveDistance\_x', 'ModeratelyActiveDistance\_x', 'LightActiveDistance\_x',

                     'SedentaryActiveDistance\_x', 'VeryActiveMinutes\_x', 'FairlyActiveMinutes\_x',

                     'LightlyActiveMinutes\_x', 'SedentaryMinutes\_x', 'Calories\_x',

                     'SedentaryMinutes\_y', 'LightlyActiveMinutes\_y', 'FairlyActiveMinutes\_y',

                     'VeryActiveMinutes\_y', 'SedentaryActiveDistance\_y', 'LightActiveDistance\_y',

                     'ModeratelyActiveDistance\_y', 'VeryActiveDistance\_y']

# Drop unnecessary columns

merged\_data = merged\_data.drop(columns=columns\_to\_remove, axis=1)

# Save Partially Processed Dataset after dropping Columns:

partial\_processed\_data\_path = '/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/MergedPreprocessedData/PartiallyProcessedData01.csv'

merged\_data.to\_csv(partial\_processed\_data\_path, index=False)

# Remove redundant rows

merged\_data = merged\_data.drop\_duplicates()

# Reset index after dropping duplicates

merged\_data.reset\_index(drop=True, inplace=True)

# Save Partially Processed Dataset after Removing Redundant Values:

partial\_processed\_data\_path = '/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/MergedPreprocessedData/PartiallyProcessedData02.csv'

merged\_data.to\_csv(partial\_processed\_data\_path, index=False)

# 4. Scale numerical features

from sklearn.preprocessing import MinMaxScaler

numerical\_columns\_to\_scale = ['Id', 'TotalSteps','Calories\_y']

scaler = MinMaxScaler()

merged\_data[numerical\_columns\_to\_scale] = scaler.fit\_transform(merged\_data[numerical\_columns\_to\_scale])

# Save Partially Processed Dataset after applying MinMaxScaler:

partial\_processed\_data\_path = '/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/MergedPreprocessedData/PartiallyProcessedData03.csv'

merged\_data.to\_csv(partial\_processed\_data\_path, index=False)

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

preprocessed\_data\_path = '/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/MergedPreprocessedData/PartiallyProcessedData03.csv'

partially\_processed\_data = pd.read\_csv(preprocessed\_data\_path)

# Select columns for the box plot

columns\_for\_box\_plot = ['Id', 'ActivityDate', 'TotalSteps', 'ActivityDay\_x', 'Calories\_y', 'ActivityDay\_y']

# Create a box plot

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

sns.boxplot(data=partially\_processed\_data[columns\_for\_box\_plot])

plt.title('Box Plot of Preprocessed Data')

plt.show()

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

import numpy as np

# Load the partially preprocessed data

preprocessed\_data\_path = '/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/MergedPreprocessedData/PartiallyProcessedData03.csv'

partially\_processed\_data = pd.read\_csv(preprocessed\_data\_path)

# Select columns for the box plot

columns\_for\_box\_plot = ['Id', 'ActivityDate', 'TotalSteps', 'ActivityDay\_x', 'Calories\_y', 'ActivityDay\_y']

np.random.seed(42)  # Setting seed for reproducibility

partially\_processed\_data['Rating'] = np.random.randint(1, 11, size=len(partially\_processed\_data))

# Create a box plot

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

sns.boxplot(data=partially\_processed\_data[columns\_for\_box\_plot + ['Rating']])

plt.title('Box Plot of Preprocessed Data with Rating')

plt.show()

# Save the dataset with the new 'Rating' column

output\_path = '/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/MergedPreprocessedData/PartiallyProcessedData04.csv'

partially\_processed\_data.to\_csv(output\_path, index=False)

import pandas as pd

import numpy as np

# Load your preprocessed dataset

path = '/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/MergedPreprocessedData/PartiallyProcessedData04.csv'

preprocessed\_data = pd.read\_csv(path)

# Replace all values in the 'Id' column with random values

preprocessed\_data['Id'] = np.random.randint(1, 1000, size=len(preprocessed\_data))

# Save the modified dataset

modified\_path = '/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/MergedPreprocessedData/PartiallyProcessedData05.csv'

preprocessed\_data.to\_csv(modified\_path, index=False)

import pandas as pd

import numpy as np

# Load the dataset

file\_path = '/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/MergedPreprocessedData/PartiallyProcessedData05.csv'

preprocessed\_data = pd.read\_csv(file\_path)

# Check the current values in the 'TotalSteps' column

print("Original 'TotalSteps' column:")

print(preprocessed\_data['TotalSteps'].head())

# Generate random values to replace 'TotalSteps'

random\_values = np.random.randint(1000, 10000, size=len(preprocessed\_data))  # Adjust the range as needed

# Replace 'TotalSteps' column with random values

preprocessed\_data['TotalSteps'] = random\_values

# Check the updated values in the 'TotalSteps' column

print("\nUpdated 'TotalSteps' column:")

print(preprocessed\_data['TotalSteps'].head())

# Save the updated DataFrame to the same file

preprocessed\_data.to\_csv(file\_path, index=False)

print("\nDataset with random 'TotalSteps' values saved.")

import pandas as pd

import numpy as np

file\_path = '/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/MergedPreprocessedData/PartiallyProcessedData05.csv'

preprocessed\_data = pd.read\_csv(file\_path)

# Check the current values in the 'Calories\_y' column

print("Original 'Calories\_y' column:")

print(preprocessed\_data['Calories\_y'].head())

# Generate random values to replace 'Calories\_y'

random\_values\_calories = np.random.uniform(100, 500, size=len(preprocessed\_data))  # Adjust the range as needed

# Replace 'Calories\_y' column with random values

preprocessed\_data['Calories\_y'] = random\_values\_calories

# Check the updated values in the 'Calories\_y' column

print("\nUpdated 'Calories\_y' column:")

print(preprocessed\_data['Calories\_y'].head())

# Save the updated DataFrame to the same file

preprocessed\_data.to\_csv(file\_path, index=False)

print("\nDataset with random 'Calories\_y' values saved.")

import pandas as pd

import numpy as np

file\_path = '/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/MergedPreprocessedData/PartiallyProcessedData05.csv'

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

# Add an "ActivityType" column with random values

np.random.seed(42)  # Setting seed for reproducibility

df['ActivityType'] = np.random.choice(['Running', 'Swimming', 'Cycling', 'Yoga', 'Weightlifting'], size=len(df))

# Save the updated dataset to a new CSV file

output\_path = '/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/MergedPreprocessedData/PartiallyProcessedData05.csv'

df.to\_csv(output\_path, index=False)

print(f"Dataset with ActivityType column saved to: {output\_path}")

#### **MergingData.ipynb:**

# Merge and Preprocess Data for Fitness Recommendations

import pandas as pd

# Loading all the cleaned datasets

df\_dailyActivity\_merged\_Filtered = pd.read\_csv('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/FilteredFitbaseData/dailyActivity\_merged\_Filtered.csv')

df\_dailyCalories\_merged\_Filtered = pd.read\_csv('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/FilteredFitbaseData/dailyCalories\_merged\_Filtered.csv')

df\_dailyIntensities\_merged\_Filtered = pd.read\_csv('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/FilteredFitbaseData/dailyIntensities\_merged\_Filtered.csv')

df\_dailySteps\_merged\_Filtered = pd.read\_csv('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/FilteredFitbaseData/dailySteps\_merged\_Filtered.csv')

df\_heartrate\_seconds\_merged\_Filtered = pd.read\_csv('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/FilteredFitbaseData/heartrate\_seconds\_merged\_Filtered.csv')

df\_hourlyCalories\_merged\_Filtered = pd.read\_csv('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/FilteredFitbaseData/hourlyCalories\_merged\_Filtered.csv')

df\_hourlyIntensities\_merged\_Filtered = pd.read\_csv('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/FilteredFitbaseData/hourlyIntensities\_merged\_Filtered.csv')

df\_hourlySteps\_merged\_Filtered = pd.read\_csv('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/FilteredFitbaseData/hourlySteps\_merged\_Filtered.csv')

df\_minuteCaloriesNarrow\_merged\_Filtered = pd.read\_csv('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/FilteredFitbaseData/minuteCaloriesNarrow\_merged\_Filtered.csv')

df\_minuteCaloriesWide\_merged\_Filtered = pd.read\_csv('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/FilteredFitbaseData/minuteCaloriesWide\_merged\_Filtered.csv')

df\_minuteIntensitiesNarrow\_merged\_Filtered = pd.read\_csv('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/FilteredFitbaseData/minuteIntensitiesNarrow\_merged\_Filtered.csv')

df\_minuteIntensitiesWide\_merged\_Filtered = pd.read\_csv('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/FilteredFitbaseData/minuteIntensitiesWide\_merged\_Filtered.csv')

df\_minuteMETsNarrow\_merged\_Filtered = pd.read\_csv('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/FilteredFitbaseData/minuteMETsNarrow\_merged\_Filtered.csv')

df\_minuteSleep\_merged\_Filtered = pd.read\_csv('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/FilteredFitbaseData/minuteSleep\_merged\_Filtered.csv')

df\_minuteStepsNarrow\_merged\_Filtered = pd.read\_csv('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/FilteredFitbaseData/minuteStepsNarrow\_merged\_Filtered.csv')

df\_minuteStepsWide\_merged\_Filtered = pd.read\_csv('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/FilteredFitbaseData/minuteStepsWide\_merged\_Filtered.csv')

df\_sleepday\_merged\_Filtered = pd.read\_csv('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/FilteredFitbaseData/sleepday\_merged\_Filtered.csv')

df\_weightLogInfo\_merged\_Filtered = pd.read\_csv('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/FilteredFitbaseData/weightLogInfo\_merged\_Filtered.csv')

# Merge datasets based on 'Id'

merged\_data = pd.merge(df\_dailyActivity\_merged\_Filtered, df\_dailyCalories\_merged\_Filtered, on='Id', how='inner')

merged\_data = pd.merge(merged\_data, df\_dailyIntensities\_merged\_Filtered, on='Id', how='inner')

# merged\_data = pd.merge(merged\_data, df\_dailySteps\_merged\_Filtered, on='Id', how='inner')

# merged\_data = pd.merge(merged\_data, df\_heartrate\_seconds\_merged\_Filtered, on='Id', how='inner')

# merged\_data = pd.merge(merged\_data, df\_hourlyCalories\_merged\_Filtered, on='Id', how='inner')

# merged\_data = pd.merge(merged\_data, df\_hourlyIntensities\_merged\_Filtered, on='Id', how='inner')

# merged\_data = pd.merge(merged\_data, df\_hourlySteps\_merged\_Filtered, on='Id', how='inner')

# merged\_data = pd.merge(merged\_data, df\_minuteCaloriesNarrow\_merged\_Filtered, on='Id', how='inner')

# merged\_data = pd.merge(merged\_data, df\_minuteCaloriesWide\_merged\_Filtered, on='Id', how='inner')

# merged\_data = pd.merge(merged\_data, df\_minuteIntensitiesWide\_merged\_Filtered, on='Id', how='inner')

# merged\_data = pd.merge(merged\_data, df\_minuteIntensitiesNarrow\_merged\_Filtered, on='Id', how='inner')

# merged\_data = pd.merge(merged\_data, df\_minuteMETsNarrow\_merged\_Filtered, on='Id', how='inner')

# merged\_data = pd.merge(merged\_data, df\_minuteSleep\_merged\_Filtered, on='Id', how='inner')

# merged\_data = pd.merge(merged\_data, df\_minuteStepsNarrow\_merged\_Filtered, on='Id', how='inner')

# merged\_data = pd.merge(merged\_data, df\_minuteStepsWide\_merged\_Filtered, on='Id', how='inner')

# merged\_data = pd.merge(merged\_data, df\_sleepday\_merged\_Filtered, on='Id', how='inner')

# merged\_data = pd.merge(merged\_data, df\_weightLogInfo\_merged\_Filtered, on='Id', how='inner')

# Save the merged dataset

merged\_data.to\_csv('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/MergedData/merged\_dataNew.csv', index=False)

import pandas as pd

# Defining the file paths for each dataset

dataset\_paths = {

    'dailyActivity': '/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/FilteredFitbaseData/dailyActivity\_merged\_Filtered.csv',

    'dailyCalories': '/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/FilteredFitbaseData/dailyCalories\_merged\_Filtered.csv',

    'dailyIntensities': '/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/FilteredFitbaseData/dailyIntensities\_merged\_Filtered.csv',

    'dailySteps': '/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/FilteredFitbaseData/dailySteps\_merged\_Filtered.csv',

    'hourlyCalories': '/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/FilteredFitbaseData/hourlyCalories\_merged\_Filtered.csv',

    'hourlyIntensities': '/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/FilteredFitbaseData/hourlyIntensities\_merged\_Filtered.csv',

    'hourlySteps': '/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/FilteredFitbaseData/hourlySteps\_merged\_Filtered.csv',

    'minuteCaloriesNarrow': '/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/FilteredFitbaseData/minuteCaloriesNarrow\_merged\_Filtered.csv',

    'minuteCaloriesWide': '/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/FilteredFitbaseData/minuteCaloriesWide\_merged\_Filtered.csv',

    'minuteIntensitiesWide': '/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/FilteredFitbaseData/minuteIntensitiesWide\_merged\_Filtered.csv',

    'minuteIntensitiesNarrow': '/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/FilteredFitbaseData/minuteIntensitiesNarrow\_merged\_Filtered.csv',

    'minuteMETsNarrow': '/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/FilteredFitbaseData/minuteMETsNarrow\_merged\_Filtered.csv',

    'minuteSleep': '/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/FilteredFitbaseData/minuteSleep\_merged\_Filtered.csv',

    'minuteStepsNarrow': '/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/FilteredFitbaseData/minuteStepsNarrow\_merged\_Filtered.csv',

    'minuteStepsWide': '/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/FilteredFitbaseData/minuteStepsWide\_merged\_Filtered.csv',

    'sleepDay': '/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/FilteredFitbaseData/sleepDay\_merged\_Filtered.csv',

    'weightLogInfo': '/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/FilteredFitbaseData/weightLogInfo\_merged\_Filtered.csv',

}

# Load datasets into dataframes

datasets = {}

for name, path in dataset\_paths.items():

    datasets[name] = pd.read\_csv(path)

# Updated Key User Features

key\_user\_features = ['Id', 'ActivityDate', 'Calories', 'SedentaryMinutes', 'LightlyActiveMinutes', 'TotalSteps','AverageHeartRate', 'MaxHeartRate', 'TotalMinutesAsleep', 'TotalTimeInBed', 'WeightKg', 'BMI']

# Updated Key Item Features

key\_item\_features = ['ActivityType', 'Intensity00', 'Intensity01','Intensity59', 'HourlyCalories', 'HourlySteps','HourlyIntensity', 'RunningDistance', 'CyclingDistance', 'WalkingDistance', 'SleepEfficiency','SleepStartTimestamp', 'SleepEndTimestamp', 'WeightPounds', 'Fat', 'IsManualReport', 'TotalSleepRecords']

dailyActivity\_columns = ['Id', 'ActivityDate', 'Calories', 'SedentaryMinutes', 'LightlyActiveMinutes', 'TotalSteps']

dailyCalories\_columns = ['Id', 'ActivityDay', 'Calories']

dailyIntensities\_columns = ['Id', 'ActivityDay', 'SedentaryMinutes', 'LightlyActiveMinutes', 'FairlyActiveMinutes', 'VeryActiveMinutes', 'SedentaryActiveDistance', 'LightActiveDistance', 'ModeratelyActiveDistance', 'VeryActiveDistance']

dailySteps\_columns = ['Id', 'ActivityDay', 'StepTotal']

hourlyCalories\_columns = ['Id', 'ActivityHour', 'Calories']

hourlyIntensities\_columns = ['Id', 'ActivityHour', 'TotalIntensity', 'AverageIntensity']

hourlySteps\_columns = ['Id', 'ActivityHour', 'StepTotal']

minuteCaloriesNarrow\_columns = ['Id', 'ActivityMinute', 'Calories']

minuteCaloriesWide\_columns = ['Id', 'ActivityHour', 'Calories00', 'Calories59']

minuteIntensitiesWide\_columns = ['Id', 'ActivityHour', 'Intensity00','Intensity59']

minuteIntensitiesNarrow\_columns = ['Id', 'ActivityMinute', 'Intensity']

minuteMETsNarrow\_columns = ['Id', 'ActivityMinute', 'METs']

minuteSleep\_columns = ['Id', 'date', 'value', 'logId']

minuteStepsNarrow\_columns = ['Id', 'ActivityMinute', 'Steps']

minuteStepsWide\_columns = ['Id', 'ActivityHour', 'Hour', 'Steps']

sleepDay\_columns = ['TotalSleepRecords', 'TotalMinutesAsleep', 'TotalTimeInBed']

weightLogInfo\_columns = ['Id', 'Date', 'WeightKg', 'WeightPounds', 'Fat', 'BMI', 'IsManualReport', 'LogId']

# Create a dictionary to map datasets to their key features

dataset\_key\_features = {

    'dailyActivity': dailyActivity\_columns,

    'dailyCalories': dailyCalories\_columns,

    'dailyIntensities': dailyIntensities\_columns,

    'dailySteps': dailySteps\_columns,

    'hourlyCalories': hourlyCalories\_columns,

    'hourlyIntensities': hourlyIntensities\_columns,

    'hourlySteps': hourlySteps\_columns,

    'minuteCaloriesNarrow': minuteCaloriesNarrow\_columns,

    'minuteCaloriesWide': minuteCaloriesWide\_columns,

    'minuteIntensitiesWide': minuteIntensitiesWide\_columns,

    'minuteIntensitiesNarrow': minuteIntensitiesNarrow\_columns,

    'minuteMETsNarrow': minuteMETsNarrow\_columns,

    'minuteSleep': minuteSleep\_columns,

    'minuteStepsNarrow': minuteStepsNarrow\_columns,

    'minuteStepsWide': minuteStepsWide\_columns,

    'sleepDay': sleepDay\_columns,

    'weightLogInfo': weightLogInfo\_columns,

}

user\_features = dataset\_key\_features['dailyActivity']  # User-based features

item\_features = dataset\_key\_features['dailyIntensities']  # Item-based features

# Print the identified features

print("User-based features:", user\_features)

print("Item-based features:", item\_features)

import pandas as pd

dailyActivity\_df = pd.read\_csv(dataset\_paths['dailyActivity'])

dailyCalories\_df = pd.read\_csv(dataset\_paths['dailyCalories'])

dailyIntensities\_df = pd.read\_csv(dataset\_paths['dailyIntensities'])

dailySteps\_df = pd.read\_csv(dataset\_paths['dailySteps'])

# heartrate\_seconds\_df = pd.read\_csv(dataset\_paths['heartrate\_seconds'])

hourlyCalories\_df = pd.read\_csv(dataset\_paths['hourlyCalories'])

hourlyIntensities\_df = pd.read\_csv(dataset\_paths['hourlyIntensities'])

hourlySteps\_df = pd.read\_csv(dataset\_paths['hourlySteps'])

minuteCaloriesNarrow\_df = pd.read\_csv(dataset\_paths['minuteCaloriesNarrow'])

minuteCaloriesWide\_df = pd.read\_csv(dataset\_paths['minuteCaloriesWide'])

minuteIntensitiesWide\_df = pd.read\_csv(dataset\_paths['minuteIntensitiesWide'])

minuteIntensitiesNarrow\_df = pd.read\_csv(dataset\_paths['minuteIntensitiesNarrow'])

minuteMETsNarrow\_df = pd.read\_csv(dataset\_paths['minuteMETsNarrow'])

minuteSleep\_df = pd.read\_csv(dataset\_paths['minuteSleep'])

minuteStepsNarrow\_df = pd.read\_csv(dataset\_paths['minuteStepsNarrow'])

minuteStepsWide\_df = pd.read\_csv(dataset\_paths['minuteStepsWide'])

sleepDay\_df = pd.read\_csv(dataset\_paths['sleepDay'])

weightLogInfo\_df = pd.read\_csv(dataset\_paths['weightLogInfo'])

# Identify common identifiers and merge the datasets

comprehensive\_df = pd.merge(dailyActivity\_df, dailyCalories\_df, on='Id', how='outer')

comprehensive\_df = pd.merge(comprehensive\_df, dailyIntensities\_df, on='Id', how='outer')

comprehensive\_df = pd.merge(comprehensive\_df, dailySteps\_df, on='Id', how='outer')

# comprehensive\_df = pd.merge(comprehensive\_df, heartrate\_seconds\_df, on='Id', how='outer')

comprehensive\_df = pd.merge(comprehensive\_df, hourlyCalories\_df, on='Id', how='outer')

comprehensive\_df = pd.merge(comprehensive\_df, hourlyIntensities\_df, on='Id', how='outer')

comprehensive\_df = pd.merge(comprehensive\_df, hourlySteps\_df, on='Id', how='outer')

comprehensive\_df = pd.merge(comprehensive\_df, minuteCaloriesNarrow\_df, on='Id', how='outer')

comprehensive\_df = pd.merge(comprehensive\_df, minuteCaloriesWide\_df, on='Id', how='outer')

comprehensive\_df = pd.merge(comprehensive\_df, minuteIntensitiesWide\_df, on='Id', how='outer')

comprehensive\_df = pd.merge(comprehensive\_df, minuteIntensitiesNarrow\_df, on='Id', how='outer')

comprehensive\_df = pd.merge(comprehensive\_df, minuteMETsNarrow\_df, on='Id', how='outer')

comprehensive\_df = pd.merge(comprehensive\_df, minuteSleep\_df, on='Id', how='outer')

comprehensive\_df = pd.merge(comprehensive\_df, minuteStepsNarrow\_df, on='Id', how='outer')

comprehensive\_df = pd.merge(comprehensive\_df, minuteStepsWide\_df, on='Id', how='outer')

comprehensive\_df = pd.merge(comprehensive\_df, sleepDay\_df, on='Id', how='outer')

comprehensive\_df = pd.merge(comprehensive\_df, weightLogInfo\_df, on='Id', how='outer')

print(comprehensive\_df.head())

## **Health Trend Analysis:**

### **HealthTrendAnalysis**

#### **healthTrendAnalysis.ipynb:**

import pandas as pd

file\_path = '/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/MergedPreprocessedData/PartiallyProcessedData05.csv'

preprocessed\_data = pd.read\_csv(file\_path)

# Convert 'ActivityDate' to datetime for time series analysis

preprocessed\_data['ActivityDate'] = pd.to\_datetime(preprocessed\_data['ActivityDate'])

# Setting 'ActivityDate' as the index

preprocessed\_data.set\_index('ActivityDate', inplace=True)

import matplotlib.pyplot as plt

# Plot daily steps over time

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

plt.plot(preprocessed\_data['TotalSteps'], marker='o', linestyle='-')

plt.title('Daily Steps Over Time')

plt.xlabel('Date')

plt.ylabel('Total Steps')

plt.grid(True)

plt.show()

from statsmodels.tsa.seasonal import STL

# Decompose the time series

preprocessed\_data['TotalSteps'].plot(figsize=(12, 6))

plt.title('Total Steps Over Time')

plt.xlabel('Date')

plt.ylabel('Total Steps')

plt.show()

stl\_result = STL(preprocessed\_data['TotalSteps'], seasonal=7).fit()

weekly\_data = preprocessed\_data['TotalSteps'].resample('W').mean()

stl\_result = STL(weekly\_data).fit()

from statsmodels.graphics.tsaplots import plot\_acf

plot\_acf(preprocessed\_data['TotalSteps'], lags=50)

plt.show()

from statsmodels.tsa.statespace.sarimax import SARIMAX

sarima\_model = SARIMAX(preprocessed\_data['TotalSteps'], order=(1, 1, 1), seasonal\_order=(1, 1, 1, 7))

sarima\_result = sarima\_model.fit(disp=False)

forecast = sarima\_result.get\_forecast(steps=30)

forecast\_ci = forecast.conf\_int()

# Plot the forecast

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

plt.plot(preprocessed\_data['TotalSteps'], label='Actual Steps')

plt.plot(forecast.predicted\_mean, color='red', label='Forecasted Steps')

plt.fill\_between(forecast\_ci.index, forecast\_ci.iloc[:, 0], forecast\_ci.iloc[:, 1], color='red', alpha=0.2)

plt.title('Daily Steps Forecast with SARIMA Model')

plt.xlabel('Date')

plt.ylabel('Total Steps')

plt.legend()

plt.grid(True)

plt.show()

## **Goal Tracking and Achievement**

### **GoalTracking:**

#### **goalTracking.ipynb:**

import pandas as pd

file\_path = '/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/MergedPreprocessedData/PartiallyProcessedData05.csv'

preprocessed\_data = pd.read\_csv(file\_path)

def track\_fitness\_goal(user\_id, goal\_type, goal\_value, actual\_performance):

    """

    Track fitness goal achievement based on user-defined goals.

    Parameters:

    - user\_id: ID of the user for whom the goal is being tracked.

    - goal\_type: Type of fitness goal ('steps', 'calories', etc.).

    - goal\_value: User-defined goal value.

    - actual\_performance: User's actual performance in the specified metric.

    Returns:

    - Goal tracking result.

    """

    if goal\_value <= 0:

        raise ValueError("Goal value must be greater than zero.")

    goal\_met = actual\_performance >= goal\_value

    goal\_progress = (actual\_performance / goal\_value) \* 100 if goal\_met else 0

    result = {

        "user\_id": user\_id,

        "goal\_type": goal\_type,

        "goal\_value": goal\_value,

        "actual\_performance": actual\_performance,

        "goal\_met": goal\_met,

        "goal\_progress": goal\_progress,

    }

    return result

# Example: Track fitness goal for user with ID 590

user\_id = 590

goal\_type = 'steps'

goal\_value = 8000

actual\_performance = preprocessed\_data[preprocessed\_data['Id'] == user\_id]['TotalSteps'].sum()

goal\_tracking\_result = track\_fitness\_goal(user\_id, goal\_type, goal\_value, actual\_performance)

# Display goal tracking result

print("Goal Tracking Result:")

print(goal\_tracking\_result)

## **User Profiling**

### **Machine Learning Model:**

#### **dailyActivity\_merged\_Filtered\_UserProfiling.ipynb:**

import pandas as pd

from sklearn.preprocessing import StandardScaler

from sklearn.cluster import KMeans

import joblib

# Load the cleaned dataset

df = pd.read\_csv('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/FilteredFitbaseData/dailyActivity\_merged\_Filtered.csv')

# Select the features for user profiling

selected\_features = [

    'TotalSteps','TotalDistance', 'VeryActiveMinutes','FairlyActiveMinutes', 'LightlyActiveMinutes', 'SedentaryMinutes','Calories'

]

# Prepare the data for clustering

X = df[selected\_features]

# Standardize the data

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# Train a clustering model (KMeans in this case)

num\_clusters = 3

kmeans = KMeans(n\_clusters=num\_clusters, random\_state=42)

df['Cluster'] = kmeans.fit\_predict(X\_scaled)

# Save the clustering model

model\_filename = '/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/MLModels/dailyActivity\_merged\_Model.pkl'

joblib.dump(kmeans, model\_filename)

# Save the dataset with cluster labels

output\_filename = '/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/ClusteredData/dailyActivity\_merged\_Clustered.csv'

df.to\_csv(output\_filename, index=False)

#### **dailyCalories\_merged\_Filtered\_UserProfiling.ipynb:**

import pandas as pd

from sklearn.preprocessing import StandardScaler

from sklearn.cluster import KMeans

import joblib

# Load the cleaned dataset

df = pd.read\_csv('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/FilteredFitbaseData/dailyCalories\_merged\_Filtered.csv')

# Select the features for user profiling

selected\_features = ['Id', 'Calories']

# Prepare the data for clustering

X = df[selected\_features]

# Standardize the data

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# Train a clustering model (KMeans in this case)

num\_clusters = 3

kmeans = KMeans(n\_clusters=num\_clusters, random\_state=42)

df['Cluster'] = kmeans.fit\_predict(X\_scaled)

# Save the clustering model

model\_filename = '/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/MLModels/dailyCalories\_merged\_Model.pkl'

joblib.dump(kmeans, model\_filename)

# Save the dataset with cluster labels

output\_filename = '/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/ClusteredData/dailyCalories\_merged\_Clustered.csv'

df.to\_csv(output\_filename, index=False)

#### **dailyIntensities\_merged\_Filtered\_UserProfiling.ipynb:**

import pandas as pd

from sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler

import joblib

# Load the cleaned dataset

df = pd.read\_csv('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/FilteredFitbaseData/dailyIntensities\_merged\_Filtered.csv')

# Select relevant features for user profiling

selected\_features = ['SedentaryMinutes', 'LightlyActiveMinutes', 'FairlyActiveMinutes', 'VeryActiveMinutes','LightActiveDistance', 'ModeratelyActiveDistance', 'VeryActiveDistance']

# Extract the selected features

X = df[selected\_features]

# Standardize the data

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# Train a clustering model (KMeans)

kmeans = KMeans(n\_clusters=3, random\_state=42)

df['Cluster'] = kmeans.fit\_predict(X\_scaled)

# Save the clustering model

model\_save\_path = '/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/MLModels/dailyIntensities\_merged\_Model.pkl'

joblib.dump(kmeans, model\_save\_path)

# Save the dataset with cluster labels

clustered\_data\_save\_path = '/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/ClusteredData/dailyIntensities\_merged\_Clustered.csv'

df.to\_csv(clustered\_data\_save\_path, index=False)

#### **dailySteps\_merged\_Filtered\_UserProfiling.ipynb:**

import pandas as pd

from sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler

import joblib

# Load the cleaned dataset

df = pd.read\_csv('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/FilteredFitbaseData/dailySteps\_merged\_Filtered.csv')

# Select relevant features for user profiling

selected\_features = ['StepTotal']

# Extract the selected features

X = df[selected\_features]

# Standardize the data

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# Train a clustering model (KMeans)

kmeans = KMeans(n\_clusters=3, random\_state=42)

df['Cluster'] = kmeans.fit\_predict(X\_scaled)

# Save the clustering model

model\_save\_path = '/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/MLModels/dailySteps\_merged\_Model.pkl'

joblib.dump(kmeans, model\_save\_path)

# Save the dataset with cluster labels

clustered\_data\_save\_path = '/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/ClusteredData/dailySteps\_merged\_Clustered.csv'

df.to\_csv(clustered\_data\_save\_path, index=False)

#### **heartrate\_seconds\_merged\_Filtered\_UserProfiling.ipynb:**

import pandas as pd

from sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler

import joblib

# Load the cleaned dataset

df = pd.read\_csv('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/FilteredFitbaseData/heartrate\_seconds\_merged\_Filtered.csv')

# Select relevant features for user profiling

selected\_features = ['Value']

# Extract the selected features

X = df[selected\_features]

# Standardize the data

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# Train a clustering model (KMeans)

kmeans = KMeans(n\_clusters=3, random\_state=42)

df['Cluster'] = kmeans.fit\_predict(X\_scaled)

# Save the clustering model

model\_save\_path = '/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/MLModels/heartrate\_seconds\_merged\_Model.pkl'

joblib.dump(kmeans, model\_save\_path)

# Save the dataset with cluster labels

clustered\_data\_save\_path = '/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/ClusteredData/heartrate\_seconds\_merged\_Clustered.csv'

df.to\_csv(clustered\_data\_save\_path, index=False)

#### **hourlyCalories\_merged\_Filtered\_UserProfiling.ipynb:**

import pandas as pd

from sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler

import joblib

# Load the cleaned dataset

df = pd.read\_csv('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/FilteredFitbaseData/hourlyCalories\_merged\_Filtered.csv')

# Select relevant features for user profiling

selected\_features = ['Calories']

# Extract the selected features

X = df[selected\_features]

# Standardize the data

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# Train a clustering model (KMeans)

kmeans = KMeans(n\_clusters=3, random\_state=42)

df['Cluster'] = kmeans.fit\_predict(X\_scaled)

# Save the clustering model

model\_save\_path = '/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/MLModels/hourlyCalories\_merged\_Model.pkl'

joblib.dump(kmeans, model\_save\_path)

# Save the dataset with cluster labels

clustered\_data\_save\_path = '/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/ClusteredData/hourlyCalories\_merged\_Clustered.csv'

df.to\_csv(clustered\_data\_save\_path, index=False)

#### **hourlyIntensities\_merged\_Filtered\_UserProfiling.ipynb:**

import pandas as pd

from sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler

import joblib

# Load the cleaned dataset

df = pd.read\_csv('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/FilteredFitbaseData/hourlyIntensities\_merged\_Filtered.csv')

# Select relevant features for user profiling

selected\_features = ['TotalIntensity', 'AverageIntensity']

# Extract the selected features

X = df[selected\_features]

# Standardize the data

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# Train a clustering model (KMeans)

kmeans = KMeans(n\_clusters=3, random\_state=42)

df['Cluster'] = kmeans.fit\_predict(X\_scaled)

# Save the clustering model

model\_save\_path = '/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/MLModels/hourlyIntensities\_merged\_Model.pkl'

joblib.dump(kmeans, model\_save\_path)

# Save the dataset with cluster labels

clustered\_data\_save\_path = '/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/ClusteredData/hourlyIntensities\_merged\_Clustered.csv'

df.to\_csv(clustered\_data\_save\_path, index=False)

#### **hourlySteps\_mergd\_Filtered\_UserProfiling.ipynb:**

import pandas as pd

from sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler

import joblib

# Load the cleaned dataset

df = pd.read\_csv('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/FilteredFitbaseData/hourlySteps\_merged\_Filtered.csv')

# Select relevant features for user profiling

selected\_features = ['StepTotal']

# Extract the selected features

X = df[selected\_features]

# Standardize the data

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# Train a clustering model (KMeans)

kmeans = KMeans(n\_clusters=3, random\_state=42)

df['Cluster'] = kmeans.fit\_predict(X\_scaled)

# Save the clustering model

model\_save\_path = '/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/MLModels/hourlySteps\_merged\_Model.pkl'

joblib.dump(kmeans, model\_save\_path)

# Save the dataset with cluster labels

clustered\_data\_save\_path = '/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/ClusteredData/hourlySteps\_merged\_Clustered.csv'

df.to\_csv(clustered\_data\_save\_path, index=False)

#### **minuteCaloriesNarrow\_merged\_Filtered\_UserProfiling.ipynb:**

import pandas as pd

from sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler

import joblib

# Load the cleaned dataset

df = pd.read\_csv('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/FilteredFitbaseData/minuteCaloriesNarrow\_merged\_Filtered.csv')

# Select relevant features for user profiling

selected\_features = ['Calories']

# Extract the selected features

X = df[selected\_features]

# Standardize the data

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# Train a clustering model (KMeans)

kmeans = KMeans(n\_clusters=3, random\_state=42)

df['Cluster'] = kmeans.fit\_predict(X\_scaled)

# Save the clustering model

model\_save\_path = '/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/MLModels/minuteCaloriesNarrow\_merged\_Model.pkl'

joblib.dump(kmeans, model\_save\_path)

# Save the dataset with cluster labels

clustered\_data\_save\_path = '/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/ClusteredData/minuteCaloriesNarrow\_merged\_Clustered.csv'

df.to\_csv(clustered\_data\_save\_path, index=False)

#### **minuteCaloriesWide\_merged\_Filtered\_UserProfiling.ipynb:**

import pandas as pd

from sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler

import joblib

# Load the cleaned dataset

df = pd.read\_csv('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/FilteredFitbaseData/minuteCaloriesWide\_merged\_Filtered.csv')

# Select relevant features for user profiling

selected\_features = [f'Calories{i:02d}' for i in range(60)]

# Extract the selected features

X = df[selected\_features]

# Standardize the data

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# Train a clustering model (KMeans)

kmeans = KMeans(n\_clusters=3, random\_state=42)

df['Cluster'] = kmeans.fit\_predict(X\_scaled)

# Save the clustering model

model\_save\_path = '/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/MLModels/minuteCaloriesWide\_merged\_Model.pkl'

joblib.dump(kmeans, model\_save\_path)

# Save the dataset with cluster labels

clustered\_data\_save\_path = '/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/ClusteredData/minuteCaloriesWide\_merged\_Clustered.csv'

df.to\_csv(clustered\_data\_save\_path, index=False)

#### **minuteIntensitiesNarrow\_merged\_Filtered\_UserProfiling.ipynb:**

import pandas as pd

from sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler

import joblib

# Load the cleaned dataset

df = pd.read\_csv('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/FilteredFitbaseData/minuteIntensitiesNarrow\_merged\_Filtered.csv')

# Select relevant features for user profiling

selected\_features = ['Intensity']

# Extract the selected features

X = df[selected\_features]

# Standardize the data

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# Train a clustering model (KMeans)

kmeans = KMeans(n\_clusters=3, random\_state=42)

df['Cluster'] = kmeans.fit\_predict(X\_scaled)

# Save the clustering model

model\_save\_path = '/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/MLModels/minuteIntensitiesNarrow\_merged\_Model.pkl'

joblib.dump(kmeans, model\_save\_path)

# Save the dataset with cluster labels

clustered\_data\_save\_path = '/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/ClusteredData/minuteIntensitiesNarrow\_merged\_Clustered.csv'

df.to\_csv(clustered\_data\_save\_path, index=False)

#### **minuteIntensitiesWide\_merged\_Filtered\_UserProfiling.ipynb:**

import pandas as pd

from sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler

import joblib

# Load the cleaned dataset

df = pd.read\_csv('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/FilteredFitbaseData/minuteIntensitiesWide\_merged\_Filtered.csv')

# Select relevant features for user profiling

selected\_features = [f'Intensity{i:02d}' for i in range(60)]

# Extract the selected features

X = df[selected\_features]

# Standardize the data

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# Train a clustering model (KMeans)

kmeans = KMeans(n\_clusters=3, random\_state=42)

df['Cluster'] = kmeans.fit\_predict(X\_scaled)

# Save the clustering model

model\_save\_path = '/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/MLModels/minuteIntensitiesWide\_merged\_Model.pkl'

joblib.dump(kmeans, model\_save\_path)

# Save the dataset with cluster labels

clustered\_data\_save\_path = '/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/ClusteredData/minuteIntensitiesWide\_merged\_Clustered.csv'

df.to\_csv(clustered\_data\_save\_path, index=False)

#### **minuteMETsNarrow\_merged\_Filtered\_UserProfiling.ipynb:**

import pandas as pd

from sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler

import joblib

# Load the cleaned dataset

df = pd.read\_csv('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/FilteredFitbaseData/minuteMETsNarrow\_merged\_Filtered.csv')

# Select relevant features for user profiling

selected\_features = ['METs']

# Extract the selected features

X = df[selected\_features]

# Standardize the data

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# Train a clustering model (KMeans)

kmeans = KMeans(n\_clusters=3, random\_state=42)

df['Cluster'] = kmeans.fit\_predict(X\_scaled)

# Save the clustering model

model\_save\_path = '/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/MLModels/minuteMETsNarrow\_merged\_Model.pkl'

joblib.dump(kmeans, model\_save\_path)

# Save the dataset with cluster labels

clustered\_data\_save\_path = '/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/ClusteredData/minuteMETsNarrow\_merged\_Clustered.csv'

df.to\_csv(clustered\_data\_save\_path, index=False)

#### **minuteSleep\_merged\_Filtered\_UserProfiling.ipynb:**

import pandas as pd

from sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler

import joblib

# Load the cleaned dataset

df = pd.read\_csv('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/FilteredFitbaseData/minuteSleep\_merged\_Filtered.csv')

# Select relevant features for user profiling

selected\_features = ['value']

# Extract the selected features

X = df[selected\_features]

# Standardize the data

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# Train a clustering model (KMeans)

kmeans = KMeans(n\_clusters=3, random\_state=42)

df['Cluster'] = kmeans.fit\_predict(X\_scaled)

# Save the clustering model

model\_save\_path = '/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/MLModels/minuteSleep\_merged\_Model.pkl'

joblib.dump(kmeans, model\_save\_path)

# Save the dataset with cluster labels

clustered\_data\_save\_path = '/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/ClusteredData/minuteSleep\_merged\_Clustered.csv'

df.to\_csv(clustered\_data\_save\_path, index=False)

#### **minuteStepsNarrow\_merged\_Filtered\_UserProfiling.ipynb:**

import pandas as pd

from sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler

import joblib

# Load the cleaned dataset

df = pd.read\_csv('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/FilteredFitbaseData/minuteStepsNarrow\_merged\_Filtered.csv')

# Select relevant features for user profiling

selected\_features = ['Steps']

# Extract the selected features

X = df[selected\_features]

# Standardize the data

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# Train a clustering model (KMeans)

kmeans = KMeans(n\_clusters=3, random\_state=42)

df['Cluster'] = kmeans.fit\_predict(X\_scaled)

# Save the clustering model

model\_save\_path = '/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/MLModels/minuteStepsNarrow\_merged\_Model.pkl'

joblib.dump(kmeans, model\_save\_path)

# Save the dataset with cluster labels

clustered\_data\_save\_path = '/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/ClusteredData/minuteStepsNarrow\_merged\_Clustered.csv'

df.to\_csv(clustered\_data\_save\_path, index=False)

#### **minuteStepsWide\_merged\_Filtered\_UserProfiling.ipynb:**

import pandas as pd

from sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler

from sklearn.impute import SimpleImputer

import joblib

# Load the cleaned dataset

df = pd.read\_csv('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/FilteredFitbaseData/minuteStepsWide\_merged\_Filtered.csv')

# Select relevant features for user profiling

selected\_features = ['Steps']

# Extract the selected features

X = df[selected\_features]

# Handle missing values (impute with mean)

imputer = SimpleImputer(strategy='mean')

X\_imputed = imputer.fit\_transform(X)

# Standardize the data

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X\_imputed)

# Train a clustering model (KMeans)

kmeans = KMeans(n\_clusters=3, random\_state=42)

df['Cluster'] = kmeans.fit\_predict(X\_scaled)

# Save the clustering model

model\_save\_path = '/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/MLModels/minuteStepsWide\_merged\_Model.pkl'

joblib.dump(kmeans, model\_save\_path)

# Save the dataset with cluster labels

clustered\_data\_save\_path = '/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/ClusteredData/minuteStepsWide\_merged\_Clustered.csv'

df.to\_csv(clustered\_data\_save\_path, index=False)

#### **sleepDay\_merged\_Filtered\_UserProfiling.ipynb:**

import pandas as pd

from sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler

from sklearn.impute import SimpleImputer

import joblib

# Load the cleaned dataset

df = pd.read\_csv('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/FilteredFitbaseData/sleepDay\_merged\_Filtered.csv')

# Select relevant features for user profiling

selected\_features = ['TotalSleepRecords', 'TotalMinutesAsleep', 'TotalTimeInBed']

# Extract the selected features

X = df[selected\_features]

# Handle missing values (impute with mean)

imputer = SimpleImputer(strategy='mean')

X\_imputed = imputer.fit\_transform(X)

# Standardize the data

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X\_imputed)

# Train a clustering model (KMeans)

kmeans = KMeans(n\_clusters=3, random\_state=42)

df['Cluster'] = kmeans.fit\_predict(X\_scaled)

# Save the clustering model

model\_save\_path = '/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/MLModels/sleepDay\_merged\_Model.pkl'

joblib.dump(kmeans, model\_save\_path)

# Save the dataset with cluster labels

clustered\_data\_save\_path = '/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/ClusteredData/sleepDay\_merged\_Clustered.csv'

df.to\_csv(clustered\_data\_save\_path, index=False)

#### **weightLogInfo\_merged\_Filtered\_UserProfiling.ipynb:**

import pandas as pd

from sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler

from sklearn.impute import SimpleImputer

import joblib

# Load the cleaned dataset

df = pd.read\_csv('/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/FilteredFitbaseData/weightLogInfo\_merged\_Filtered.csv')

# Select relevant features for user profiling

selected\_features = ['WeightKg', 'WeightPounds', 'Fat', 'BMI']

# Extract the selected features

X = df[selected\_features]

# Handle missing values (impute with mean)

imputer = SimpleImputer(strategy='mean')

X\_imputed = imputer.fit\_transform(X)

# Standardize the data

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X\_imputed)

# Train a clustering model (KMeans)

kmeans = KMeans(n\_clusters=3, random\_state=42)

df['Cluster'] = kmeans.fit\_predict(X\_scaled)

# Save the clustering model

model\_save\_path = '/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/MLModels/weightLogInfo\_merged\_Model.pkl'

joblib.dump(kmeans, model\_save\_path)

# Save the dataset with cluster labels

clustered\_data\_save\_path = '/University/6th Semester/Sixth Semester/AI-Enhanced-Fitness-Wellness-Analyzer-Project/Data/ClusteredData/weightLogInfo\_merged\_Clustered.csv'

df.to\_csv(clustered\_data\_save\_path, index=False)

## **User Interface Development**

### **Web\_app**

#### **Html:**

<!DOCTYPE html>

<html lang="en">

  <head>

    <meta charset="UTF-8" />

    <meta http-equiv="X-UA-Compatible" content="IE=edge" />

    <meta name="viewport" content="width=device-width, initial-scale=1.0" />

    <title>Fitlife - Work Hard To Get Better Life</title>

    <!--

    - favicon

  -->

    <link rel="shortcut icon" href="./favicon.svg" type="image/svg+xml" />

    <!--

    - custom css link

  -->

    <link rel="stylesheet" href="./assets/css/style.css" />

    <!--

    - google font link

  -->

    <link rel="preconnect" href="https://fonts.googleapis.com" />

    <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin />

    <link

      href="https://fonts.googleapis.com/css2?family=Catamaran:wght@600;700;800;900&family=Rubik:wght@400;500;800&display=swap"

      rel="stylesheet"

    />

    <link

      rel="stylesheet"

      href="{{ url\_for('static', filename='styles.css') }}"

    />

    <!--

    - preload images

  -->

    <link rel="preload" as="image" href="./assets/images/hero-banner.png" />

    <link rel="preload" as="image" href="./assets/images/hero-circle-one.png" />

    <link rel="preload" as="image" href="./assets/images/hero-circle-two.png" />

    <link rel="preload" as="image" href="./assets/images/heart-rate.svg" />

    <link rel="preload" as="image" href="./assets/images/calories.svg" />

  </head>

  <body id="top">

    <!--

    - #HEADER

  -->

    <header class="header" data-header>

      <div class="container">

        <a href="#" class="logo">

          <ion-icon name="barbell-sharp" aria-hidden="true"></ion-icon>

          <span class="span">Fitlife</span>

        </a>

        <nav class="navbar" data-navbar>

          <button

            class="nav-close-btn"

            aria-label="close menu"

            data-nav-toggler

          >

            <ion-icon name="close-sharp" aria-hidden="true"></ion-icon>

          </button>

          <ul class="navbar-list">

            <li>

              <a href="#home" class="navbar-link active" data-nav-link>Home</a>

            </li>

            <li>

              <a href="#about" class="navbar-link" data-nav-link>About Us</a>

            </li>

            <li>

              <a href="#class" class="navbar-link" data-nav-link>Classes</a>

            </li>

            <li>

              <a href="#blog" class="navbar-link" data-nav-link>Blog</a>

            </li>

            <li>

              <a href="#" class="navbar-link" data-nav-link>Contact Us</a>

            </li>

          </ul>

        </nav>

        <a href="#" class="btn btn-secondary">Join Now</a>

        <button class="nav-open-btn" aria-label="open menu" data-nav-toggler>

          <span class="line"></span>

          <span class="line"></span>

          <span class="line"></span>

        </button>

      </div>

    </header>

    <main>

      <article>

        <!--

        - #HERO

      -->

        <section

          class="section hero bg-dark has-after has-bg-image"

          id="home"

          aria-label="hero"

          data-section

          style="background-image: url('./assets/images/hero-bg.png')"

        >

          <div class="container">

            <div class="hero-content">

              <p class="hero-subtitle">

                <strong class="strong">The Best</strong>Fitness Club

              </p>

              <h1 class="h1 hero-title">Work Hard To Get Better Life</h1>

              <p class="section-text">

                Duis mollis felis quis libero dictum vehicula. Duis dictum lorem

                mi, a faucibus nisi eleifend eu.

              </p>

              <a href="#" class="btn btn-primary">Get Started</a>

            </div>

            <div class="hero-banner">

              <img

                src="./assets/images/hero-banner.png"

                width="660"

                height="753"

                alt="hero banner"

                class="w-100"

              />

              <img

                src="./assets/images/hero-circle-one.png"

                width="666"

                height="666"

                aria-hidden="true"

                alt=""

                class="circle circle-1"

              />

              <img

                src="./assets/images/hero-circle-two.png"

                width="666"

                height="666"

                aria-hidden="true"

                alt=""

                class="circle circle-2"

              />

              <img

                src="./assets/images/heart-rate.svg"

                width="255"

                height="270"

                alt="heart rate"

                class="abs-img abs-img-1"

              />

              <img

                src="./assets/images/calories.svg"

                width="348"

                height="224"

                alt="calories"

                class="abs-img abs-img-2"

              />

            </div>

          </div>

        </section>

        <!--

        - #ABOUT

      -->

        <section class="section about" id="about" aria-label="about">

          <div class="container">

            <div class="about-banner has-after">

              <img

                src="./assets/images/about-banner.png"

                width="660"

                height="648"

                loading="lazy"

                alt="about banner"

                class="w-100"

              />

              <img

                src="./assets/images/about-circle-one.png"

                width="660"

                height="534"

                loading="lazy"

                aria-hidden="true"

                alt=""

                class="circle circle-1"

              />

              <img

                src="./assets/images/about-circle-two.png"

                width="660"

                height="534"

                loading="lazy"

                aria-hidden="true"

                alt=""

                class="circle circle-2"

              />

              <img

                src="./assets/images/fitness.png"

                width="650"

                height="154"

                loading="lazy"

                alt="fitness"

                class="abs-img w-100"

              />

            </div>

            <div class="about-content">

              <p class="section-subtitle">About Us</p>

              <h2 class="h2 section-title">Welcome To Our Fitness Gym</h2>

              <p class="section-text">

                Nam ut hendrerit leo. Aenean vel ipsum nunc. Curabitur in tellus

                vitae nisi aliquet dapibus non et erat. Pellentesque porta

                sapien non accumsan dignissim curabitur sagittis nulla sit amet

                dolor feugiat.

              </p>

              <p class="section-text">

                Integer placerat vitae metus posuere tincidunt. Nullam suscipit

                ante ac aliquet viverra vestibulum ante ipsum primis.

              </p>

              <div class="wrapper">

                <div class="about-coach">

                  <figure class="coach-avatar">

                    <img

                      src="./assets/images/about-coach.jpg"

                      width="65"

                      height="65"

                      loading="lazy"

                      alt="Trainer"

                    />

                  </figure>

                  <div>

                    <h3 class="h3 coach-name">Denis Robinson</h3>

                    <p class="coach-title">Our Coach</p>

                  </div>

                </div>

                <a href="#" class="btn btn-primary">Explore More</a>

              </div>

            </div>

          </div>

        </section>

        <!--

        - #VIDEO

      -->

        <section class="section video" aria-label="video">

          <div class="container">

            <div

              class="video-card has-before has-bg-image"

              style="background-image: url('./assets/images/video-banner.jpg')"

            >

              <h2 class="h2 card-title">Explore Fitness Life</h2>

              <button class="play-btn" aria-label="play video">

                <ion-icon name="play-sharp" aria-hidden="true"></ion-icon>

              </button>

              <a href="#" class="btn-link has-before">Watch More</a>

            </div>

          </div>

        </section>

        <!--

        - #CLASS

      -->

        <section

          class="section class bg-dark has-bg-image"

          id="class"

          aria-label="class"

          style="background-image: url('./assets/images/classes-bg.png')"

        >

          <div class="container">

            <p class="section-subtitle">Our Classes</p>

            <h2 class="h2 section-title text-center">

              Fitness Classes For Every Goal

            </h2>

            <ul class="class-list has-scrollbar">

              <li class="scrollbar-item">

                <div class="class-card">

                  <figure

                    class="card-banner img-holder"

                    style="--width: 416; --height: 240"

                  >

                    <img

                      src="./assets/images/class-1.jpg"

                      width="416"

                      height="240"

                      loading="lazy"

                      alt="Weight Lifting"

                      class="img-cover"

                    />

                  </figure>

                  <div class="card-content">

                    <div class="title-wrapper">

                      <img

                        src="./assets/images/class-icon-1.png"

                        width="52"

                        height="52"

                        aria-hidden="true"

                        alt=""

                        class="title-icon"

                      />

                      <h3 class="h3">

                        <a href="#" class="card-title">Weight Lifting</a>

                      </h3>

                    </div>

                    <p class="card-text">

                      Suspendisse nisi libero, cursus ac magna sit amet,

                      fermentum imperdiet nisi.

                    </p>

                    <div class="card-progress">

                      <div class="progress-wrapper">

                        <p class="progress-label">Class Full</p>

                        <span class="progress-value">85%</span>

                      </div>

                      <div class="progress-bg">

                        <div class="progress-bar" style="width: 85%"></div>

                      </div>

                    </div>

                  </div>

                </div>

              </li>

              <li class="scrollbar-item">

                <div class="class-card">

                  <figure

                    class="card-banner img-holder"

                    style="--width: 416; --height: 240"

                  >

                    <img

                      src="./assets/images/class-2.jpg"

                      width="416"

                      height="240"

                      loading="lazy"

                      alt="Cardio & Strenght"

                      class="img-cover"

                    />

                  </figure>

                  <div class="card-content">

                    <div class="title-wrapper">

                      <img

                        src="./assets/images/class-icon-2.png"

                        width="52"

                        height="52"

                        aria-hidden="true"

                        alt=""

                        class="title-icon"

                      />

                      <h3 class="h3">

                        <a href="#" class="card-title">Cardio & Strenght</a>

                      </h3>

                    </div>

                    <p class="card-text">

                      Suspendisse nisi libero, cursus ac magna sit amet,

                      fermentum imperdiet nisi.

                    </p>

                    <div class="card-progress">

                      <div class="progress-wrapper">

                        <p class="progress-label">Class Full</p>

                        <span class="progress-value">70%</span>

                      </div>

                      <div class="progress-bg">

                        <div class="progress-bar" style="width: 70%"></div>

                      </div>

                    </div>

                  </div>

                </div>

              </li>

              <li class="scrollbar-item">

                <div class="class-card">

                  <figure

                    class="card-banner img-holder"

                    style="--width: 416; --height: 240"

                  >

                    <img

                      src="./assets/images/class-3.jpg"

                      width="416"

                      height="240"

                      loading="lazy"

                      alt="Power Yoga"

                      class="img-cover"

                    />

                  </figure>

                  <div class="card-content">

                    <div class="title-wrapper">

                      <img

                        src="./assets/images/class-icon-3.png"

                        width="52"

                        height="52"

                        aria-hidden="true"

                        alt=""

                        class="title-icon"

                      />

                      <h3 class="h3">

                        <a href="#" class="card-title">Power Yoga</a>

                      </h3>

                    </div>

                    <p class="card-text">

                      Suspendisse nisi libero, cursus ac magna sit amet,

                      fermentum imperdiet nisi.

                    </p>

                    <div class="card-progress">

                      <div class="progress-wrapper">

                        <p class="progress-label">Class Full</p>

                        <span class="progress-value">90%</span>

                      </div>

                      <div class="progress-bg">

                        <div class="progress-bar" style="width: 90%"></div>

                      </div>

                    </div>

                  </div>

                </div>

              </li>

              <li class="scrollbar-item">

                <div class="class-card">

                  <figure

                    class="card-banner img-holder"

                    style="--width: 416; --height: 240"

                  >

                    <img

                      src="./assets/images/class-4.jpg"

                      width="416"

                      height="240"

                      loading="lazy"

                      alt="The Fitness Pack"

                      class="img-cover"

                    />

                  </figure>

                  <div class="card-content">

                    <div class="title-wrapper">

                      <img

                        src="./assets/images/class-icon-4.png"

                        width="52"

                        height="52"

                        aria-hidden="true"

                        alt=""

                        class="title-icon"

                      />

                      <h3 class="h3">

                        <a href="#" class="card-title">The Fitness Pack</a>

                      </h3>

                    </div>

                    <p class="card-text">

                      Suspendisse nisi libero, cursus ac magna sit amet,

                      fermentum imperdiet nisi.

                    </p>

                    <div class="card-progress">

                      <div class="progress-wrapper">

                        <p class="progress-label">Class Full</p>

                        <span class="progress-value">60%</span>

                      </div>

                      <div class="progress-bg">

                        <div class="progress-bar" style="width: 60%"></div>

                      </div>

                    </div>

                  </div>

                </div>

              </li>

            </ul>

          </div>

        </section>

        <!--

        - #BLOG

      -->

        <section class="section blog" id="blog" aria-label="blog">

          <div class="container">

            <p class="section-subtitle">Our News</p>

            <h2 class="h2 section-title text-center">Latest Blog Feed</h2>

            <ul class="blog-list has-scrollbar">

              <li class="scrollbar-item">

                <div class="blog-card">

                  <div

                    class="card-banner img-holder"

                    style="--width: 440; --height: 270"

                  >

                    <img

                      src="./assets/images/blog-1.jpg"

                      width="440"

                      height="270"

                      loading="lazy"

                      alt="Going to the gym for the first time"

                      class="img-cover"

                    />

                    <time class="card-meta" datetime="2022-07-07"

                      >7 July 2022</time

                    >

                  </div>

                  <div class="card-content">

                    <h3 class="h3">

                      <a href="#" class="card-title"

                        >Going to the gym for the first time</a

                      >

                    </h3>

                    <p class="card-text">

                      Praesent id ipsum pellentesque lectus dapibus condimentum

                      curabitur eget risus quam. In hac habitasse platea

                      dictumst.

                    </p>

                    <a href="#" class="btn-link has-before">Read More</a>

                  </div>

                </div>

              </li>

              <li class="scrollbar-item">

                <div class="blog-card">

                  <div

                    class="card-banner img-holder"

                    style="--width: 440; --height: 270"

                  >

                    <img

                      src="./assets/images/blog-2.jpg"

                      width="440"

                      height="270"

                      loading="lazy"

                      alt="Parturient accumsan cacus pulvinar magna"

                      class="img-cover"

                    />

                    <time class="card-meta" datetime="2022-07-07"

                      >7 July 2022</time

                    >

                  </div>

                  <div class="card-content">

                    <h3 class="h3">

                      <a href="#" class="card-title"

                        >Parturient accumsan cacus pulvinar magna</a

                      >

                    </h3>

                    <p class="card-text">

                      Praesent id ipsum pellentesque lectus dapibus condimentum

                      curabitur eget risus quam. In hac habitasse platea

                      dictumst.

                    </p>

                    <a href="#" class="btn-link has-before">Read More</a>

                  </div>

                </div>

              </li>

              <li class="scrollbar-item">

                <div class="blog-card">

                  <div

                    class="card-banner img-holder"

                    style="--width: 440; --height: 270"

                  >

                    <img

                      src="./assets/images/blog-3.jpg"

                      width="440"

                      height="270"

                      loading="lazy"

                      alt="Risus purus namien parturient accumsan cacus"

                      class="img-cover"

                    />

                    <time class="card-meta" datetime="2022-07-07"

                      >7 July 2022</time

                    >

                  </div>

                  <div class="card-content">

                    <h3 class="h3">

                      <a href="#" class="card-title"

                        >Risus purus namien parturient accumsan cacus</a

                      >

                    </h3>

                    <p class="card-text">

                      Praesent id ipsum pellentesque lectus dapibus condimentum

                      curabitur eget risus quam. In hac habitasse platea

                      dictumst.

                    </p>

                    <a href="#" class="btn-link has-before">Read More</a>

                  </div>

                </div>

              </li>

            </ul>

          </div>

        </section>

      </article>

    </main>

    <!--

    - #FOOTER

  -->

    <footer class="footer">

      <div

        class="section footer-top bg-dark has-bg-image"

        style="background-image: url('./assets/images/footer-bg.png')"

      >

        <div class="container">

          <div class="footer-brand">

            <a href="#" class="logo">

              <ion-icon name="barbell-sharp" aria-hidden="true"></ion-icon>

              <span class="span">Fitlife</span>

            </a>

            <p class="footer-brand-text">

              Etiam suscipit fringilla ullamcorper sed malesuada urna nec odio.

            </p>

            <div class="wrapper">

              <img

                src="./assets/images/footer-clock.png"

                width="34"

                height="34"

                loading="lazy"

                alt="Clock"

              />

              <ul class="footer-brand-list">

                <li>

                  <p class="footer-brand-title">Monday - Friday</p>

                  <p>7:00Am - 10:00Pm</p>

                </li>

                <li>

                  <p class="footer-brand-title">Saturday - Sunday</p>

                  <p>7:00Am - 2:00Pm</p>

                </li>

              </ul>

            </div>

          </div>

          <ul class="footer-list">

            <li>

              <p class="footer-list-title has-before">Our Links</p>

            </li>

            <li>

              <a href="#" class="footer-link">Home</a>

            </li>

            <li>

              <a href="#" class="footer-link">About Us</a>

            </li>

            <li>

              <a href="#" class="footer-link">Classes</a>

            </li>

            <li>

              <a href="#" class="footer-link">Blog</a>

            </li>

            <li>

              <a href="#" class="footer-link">Contact Us</a>

            </li>

          </ul>

          <ul class="footer-list">

            <li>

              <p class="footer-list-title has-before">Contact Us</p>

            </li>

            <li class="footer-list-item">

              <div class="icon">

                <ion-icon name="location" aria-hidden="true"></ion-icon>

              </div>

              <address class="address footer-link">

                1247/Plot No. 39, 15th Phase, Colony, Kukatpally, Hyderabad

              </address>

            </li>

            <li class="footer-list-item">

              <div class="icon">

                <ion-icon name="call" aria-hidden="true"></ion-icon>

              </div>

              <div>

                <a href="tel:18001213637" class="footer-link">1800-121-3637</a>

                <a href="tel:+915552348765" class="footer-link"

                  >+91 555 234-8765</a

                >

              </div>

            </li>

            <li class="footer-list-item">

              <div class="icon">

                <ion-icon name="mail" aria-hidden="true"></ion-icon>

              </div>

              <div>

                <a href="mailto:info@fitlife.com" class="footer-link"

                  >info@fitlife.com</a

                >

                <a href="mailto:services@fitlife.com" class="footer-link"

                  >services@fitlife.com</a

                >

              </div>

            </li>

          </ul>

          <ul class="footer-list">

            <li>

              <p class="footer-list-title has-before">Our Newsletter</p>

            </li>

            <li>

              <form action="" class="footer-form">

                <input

                  type="email"

                  name="email\_address"

                  aria-label="email"

                  placeholder="Email Address"

                  required

                  class="input-field"

                />

                <button

                  type="submit"

                  class="btn btn-primary"

                  aria-label="Submit"

                >

                  <ion-icon

                    name="chevron-forward-sharp"

                    aria-hidden="true"

                  ></ion-icon>

                </button>

              </form>

            </li>

            <li>

              <ul class="social-list">

                <li>

                  <a href="#" class="social-link">

                    <ion-icon name="logo-facebook"></ion-icon>

                  </a>

                </li>

                <li>

                  <a href="#" class="social-link">

                    <ion-icon name="logo-instagram"></ion-icon>

                  </a>

                </li>

                <li>

                  <a href="#" class="social-link">

                    <ion-icon name="logo-twitter"></ion-icon>

                  </a>

                </li>

              </ul>

            </li>

          </ul>

        </div>

      </div>

      <div class="footer-bottom">

        <div class="container">

          <p class="copyright">

            &copy; 2022 Fitlife. All Rights Reserved By

            <a

              href="https://github.com/ShaiikhAbdullah"

              class="copyright-link"

              target="\_blank"

              >Abdullah & Awais.</a

            >

          </p>

          <ul class="footer-bottom-list">

            <li>

              <a href="#" class="footer-bottom-link has-before"

                >Privacy Policy</a

              >

            </li>

            <li>

              <a href="#" class="footer-bottom-link has-before"

                >Terms & Condition</a

              >

            </li>

          </ul>

        </div>

      </div>

    </footer>

    <!--

    - #BACK TO TOP

  -->

    <a

      href="#top"

      class="back-top-btn"

      aria-label="back to top"

      data-back-top-btn

    >

      <ion-icon name="caret-up-sharp" aria-hidden="true"></ion-icon>

    </a>

    <!--

    - custom js link

  -->

    <script src="./assets/js/script.js" defer></script>

    <!--

    - ionicon link

  -->

    <script

      type="module"

      src="https://unpkg.com/ionicons@5.5.2/dist/ionicons/ionicons.esm.js"

    ></script>

    <script

      nomodule

      src="https://unpkg.com/ionicons@5.5.2/dist/ionicons/ionicons.js"

    ></script>

  </body>

</html>

#### **Css:**

/\*-----------------------------------\*\

  #style.css

\\*-----------------------------------\*/

/\*\*

 \* copyright 2023

 \*/

/\*-----------------------------------\*\

  #CUSTOM PROPERTY

\\*-----------------------------------\*/

:root {

  /\*\*

   \* colors

   \*/

  --rich-black-fogra-29\_50: hsl(210, 26%, 11%, 0.5);

  --rich-black-fogra-29-1: hsl(210, 26%, 11%);

  --rich-black-fogra-29-2: hsl(210, 50%, 4%);

  --silver-metallic: hsl(212, 9%, 67%);

  --coquelicot\_20: hsla(12, 98%, 52%, 0.2);

  --coquelicot\_10: hsla(12, 98%, 52%, 0.1);

  --sonic-silver: hsl(0, 0%, 47%);

  --cadet-gray: hsl(214, 15%, 62%);

  --light-gray: hsl(0, 0%, 80%);

  --coquelicot: hsl(12, 98%, 52%);

  --gainsboro: hsl(0, 0%, 88%);

  --white\_20: hsl(0, 0%, 100%, 0.2);

  --white\_10: hsl(0, 0%, 100%, 0.1);

  --black\_10: hsl(0, 0%, 0%, 0.1);

  --white: hsl(0, 0%, 100%);

  /\*\*

   \* typography

   \*/

  --ff-catamaran: 'Catamaran', sans-serif;

  --ff-rubik: 'Rubik', sans-serif;

  --fs-1: 3.8rem;

  --fs-2: 3rem;

  --fs-3: 2.5rem;

  --fs-4: 2rem;

  --fs-5: 1.8rem;

  --fs-6: 1.5rem;

  --fw-900: 900;

  --fw-800: 800;

  --fw-700: 700;

  --fw-500: 500;

  /\*\*

   \* spacing

   \*/

  --section-padding: 80px;

  /\*\*

   \* shadow

   \*/

  --shadow-1: 0 0 20px var(--black\_10);

  --shadow-2: 0px 10px 24px var(--coquelicot\_20);

  /\*\*

   \* border radius

   \*/

  --radius-10: 10px;

  --radius-8: 8px;

  --radius-5: 5px;

  /\*\*

   \* transition

   \*/

  --transition-1: 0.25s ease;

  --transition-2: 0.5s ease;

  --cubic-in: cubic-bezier(0.51, 0.03, 0.64, 0.28);

  --cubic-out: cubic-bezier(0.33, 0.85, 0.4, 0.96);

}

/\*-----------------------------------\*\

  #RESET

\\*-----------------------------------\*/

\*,

\*::before,

\*::after {

  margin: 0;

  padding: 0;

  box-sizing: border-box;

}

li { list-style: none; }

a {

  text-decoration: none;

  color: inherit;

}

a,

img,

span,

input,

button,

strong,

ion-icon { display: block; }

img { height: auto; }

input,

button {

  background: none;

  border: none;

  font: inherit;

}

input { width: 100%; }

button { cursor: pointer; }

ion-icon { pointer-events: none; }

address { font-style: normal; }

html {

  font-family: var(--ff-rubik);

  font-size: 10px;

  scroll-behavior: smooth;

}

body {

  background-color: var(--white);

  color: var(--sonic-silver);

  font-size: 1.6rem;

  line-height: 1.6;

}

:focus-visible { outline-offset: 4px; }

::-webkit-scrollbar { width: 5px; }

::-webkit-scrollbar-track { background-color: var(--light-gray); }

::-webkit-scrollbar-thumb { background-color: var(--coquelicot); }

::-webkit-scrollbar-thumb:hover { background-color: var(--rich-black-fogra-29-1); }

/\*-----------------------------------\*\

  #REUSED STYLE

\\*-----------------------------------\*/

.container { padding-inline: 15px; }

.section { padding-block: var(--section-padding); }

.bg-dark {

  background-color: var(--rich-black-fogra-29-1);

  color: var(--silver-metallic);

}

.has-bg-image {

  background-repeat: no-repeat;

  background-position: top left;

}

.has-before,

.has-after {

  position: relative;

  z-index: 1;

}

.has-before::before,

.has-after::after {

  content: "";

  position: absolute;

}

.h1,

.h2,

.h3 {

  font-family: var(--ff-catamaran);

  line-height: 1.25;

}

.h1 {

  color: var(--white);

  font-size: var(--fs-1);

  font-weight: var(--fw-900);

}

.h2,

.h3 {

  color: var(--rich-black-fogra-29-1);

  font-weight: var(--fw-800);

}

.h2 { font-size: var(--fs-2); }

.h3 { font-size: var(--fs-4); }

.section-text { font-size: var(--fs-6); }

.btn {

  max-width: max-content;

  font-size: var(--fs-6);

  text-transform: uppercase;

  font-weight: var(--fw-500);

  padding: 15px 35px;

  border-radius: var(--radius-8);

  transition: var(--transition-1);

}

.btn-primary {

  background-color: var(--coquelicot);

  color: var(--white);

}

.btn-primary:is(:hover, :focus) {

  background-color: var(--white);

  color: var(--coquelicot);

  box-shadow: var(--shadow-2);

}

.btn-secondary {

  background-color: var(--white);

  color: var(--coquelicot);

}

.btn-secondary:is(:hover, :focus) { background-color: var(--rich-black-fogra-29-1); }

.w-100 { width: 100%; }

.circle,

.abs-img { position: absolute; }

.circle {

  top: 50%;

  left: 50%;

  transform: translate(-50%, -56%);

  width: 100%;

  z-index: -1;

  animation: rotate360 15s linear infinite;

}

@keyframes rotate360 {

  0% { transform: translate(-50%, -56%) rotate(0); }

  100% { transform: translate(-50%, -56%) rotate(1turn); }

}

.circle-2 { animation-direction: reverse; }

.hero-subtitle,

.section-subtitle {

  font-family: var(--ff-catamaran);

  font-weight: var(--fw-700);

  text-transform: uppercase;

  max-width: max-content;

}

.section-subtitle {

  background-color: var(--coquelicot\_10);

  color: var(--coquelicot);

  padding: 8px 20px;

  border-radius: var(--radius-8);

}

.section-title { margin-block: 18px 35px; }

.btn-link {

  --color: var(--white);

  color: var(--color);

  font-size: var(--fs-6);

  font-weight: var(--fw-500);

  text-transform: uppercase;

  max-width: max-content;

  transition: var(--transition-1);

}

.btn-link::before {

  bottom: 0;

  left: 0;

  width: 100%;

  height: 2px;

  background-color: var(--color);

  transition: var(--transition-1);

}

.btn-link:is(:hover, :focus) { --color: var(--coquelicot); }

.text-center { text-align: center; }

.img-holder {

  aspect-ratio: var(--width) / var(--height);

  background-color: var(--light-gray);

  overflow: hidden;

}

.img-cover {

  width: 100%;

  height: 100%;

  object-fit: cover;

}

.has-scrollbar {

  display: flex;

  gap: 25px;

  overflow-x: auto;

  padding-block-end: 30px;

  scroll-snap-type: inline mandatory;

}

.scrollbar-item {

  min-width: 100%;

  scroll-snap-align: start;

}

.has-scrollbar::-webkit-scrollbar { height: 10px; }

.has-scrollbar::-webkit-scrollbar-track,

.has-scrollbar::-webkit-scrollbar-thumb { border-radius: 50px; }

.has-scrollbar::-webkit-scrollbar-thumb:hover { background-color: var(--coquelicot); }

.has-scrollbar::-webkit-scrollbar-button { width: calc(25% - 25px); }

/\*-----------------------------------\*\

  #HEADER

\\*-----------------------------------\*/

.header .btn { display: none; }

.header {

  background-color: var(--white);

  position: fixed;

  top: 0;

  left: 0;

  width: 100%;

  padding-block: 10px;

  box-shadow: var(--shadow-1);

  z-index: 4;

}

.header > .container {

  display: flex;

  justify-content: space-between;

  align-items: center;

}

.logo {

  color: var(--rich-black-fogra-29-1);

  font-family: var(--ff-catamaran);

  font-size: 3.5rem;

  font-weight: var(--fw-900);

  display: flex;

  align-items: center;

  margin-inline-start: -8px;

}

.logo ion-icon {

  color: var(--coquelicot);

  font-size: 40px;

  transform: rotate(90deg) translate(-2px, -5px);

}

.nav-open-btn {

  background-color: var(--coquelicot);

  padding: 20px 15px;

  border-radius: var(--radius-8);

}

.nav-open-btn .line {

  background-color: var(--white);

  width: 30px;

  height: 3px;

}

.nav-open-btn .line:not(:last-child) { margin-block-end: 6px; }

.nav-open-btn .line:nth-child(2) {

  width: 25px;

  margin-inline-start: auto;

}

.navbar {

  background-color: var(--coquelicot);

  color: var(--white);

  position: fixed;

  top: 100%;

  left: 0;

  width: 100%;

  height: 100%;

  display: grid;

  place-content: center;

  visibility: hidden;

  transition: 0.25s var(--cubic-in);

}

.navbar.active {

  visibility: visible;

  transform: translateY(-100%);

  transition: 0.5s var(--cubic-out);

}

.nav-close-btn {

  position: absolute;

  top: 10px;

  right: 15px;

  background-color: var(--rich-black-fogra-29-1);

  color: var(--white);

  font-size: 40px;

  padding: 10px;

  border-radius: var(--radius-8);

}

.navbar-link {

  font-family: var(--ff-catamaran);

  font-size: var(--fs-4);

  text-align: center;

  padding-block: 10px;

  margin-block-end: 20px;

  transition: var(--transition-1);

}

.navbar-link:is(:hover, :focus, .active) { color: var(--rich-black-fogra-29-1); }

/\*-----------------------------------\*\

  #HERO

\\*-----------------------------------\*/

.hero {

  color: var(--cadet-gray);

  text-align: center;

  padding-block-start: calc(var(--section-padding) + 80px);

  padding-block-end: 0;

  overflow: hidden;

}

.hero::after {

  bottom: 0;

  left: 0;

  width: 100%;

  height: 240px;

  background-color: var(--coquelicot);

  z-index: -1;

}

.hero-content { margin-block-end: 90px; }

.hero-subtitle {

  background-color: var(--white\_10);

  color: var(--white);

  margin-inline: auto;

  padding: 5px;

  padding-inline-end: 15px;

  border-radius: var(--radius-8);

}

.hero-subtitle .strong {

  display: inline-block;

  background-color: var(--coquelicot);

  padding: 2px 15px;

  margin-inline-end: 15px;

  border-radius: var(--radius-5);

}

.hero-title { margin-block: 30px 8px; }

.hero .section-text { margin-block-end: 40px; }

.hero .btn { margin-inline: auto; }

.hero-banner { position: relative; }

.abs-img-1 {

  top: 20px;

  right: -50px;

  width: 190px;

}

.abs-img-2 {

  bottom: -50px;

  left: -40px;

  width: 280px;

}

.hero .abs-img { animation: move 3s linear infinite alternate; }

@keyframes move {

  0% { transform: translate(0, 0); }

  50% { transform: translate(-5px, 10px); }

  100% { transform: translate(5px, 20px); }

}

.hero .abs-img-2 { animation-direction: alternate-reverse; }

/\*-----------------------------------\*\

  #ABOUT

\\*-----------------------------------\*/

.about { overflow: hidden; }

.about-banner { margin-block-end: 50px; }

.about-banner::after {

  bottom: 0;

  left: 0;

  width: 100%;

  height: 50%;

  background-color: var(--coquelicot);

  border-radius: var(--radius-10);

  z-index: -2;

}

.about-banner .abs-img {

  bottom: 0;

  left: 0;

  z-index: -1;

  animation: moveUp 2.5s ease infinite;

}

@keyframes moveUp {

  0%,

  30%,

  60%,

  100% { transform: translateY(0); }

  20% { transform: translateY(-30px); }

  40% { transform: translateY(-15px); }

}

.about .section-text:not(:last-of-type) { margin-block-end: 15px; }

.about .wrapper { margin-block-start: 30px; }

.about-coach {

  display: flex;

  align-items: center;

  gap: 20px;

  margin-block-end: 30px;

}

.about .coach-avatar {

  overflow: hidden;

  border-radius: 50%;

}

.about .coach-name {

  font-weight: var(--fw-700);

  margin-block-end: 5px;

}

.about .coach-title { font-size: var(--fs-6); }

.about .btn-primary:is(:hover, :focus) {

  background-color: var(--rich-black-fogra-29-1);

  color: var(--white);

  box-shadow: none;

}

/\*-----------------------------------\*\

  #VIDEO

\\*-----------------------------------\*/

.video {

  padding-block: 0;

  margin-block-end: -250px;

}

.video-card {

  background-color: var(--light-gray);

  background-size: cover;

  background-position: center;

  height: 500px;

  border-radius: var(--radius-10);

  display: flex;

  flex-direction: column;

  justify-content: center;

  align-items: center;

  overflow: hidden;

}

.video-card::before {

  top: 0;

  left: 0;

  bottom: 0;

  right: 0;

  background-color: var(--rich-black-fogra-29\_50);

  z-index: -1;

}

.video-card .card-title {

  color: var(--white);

  font-size: var(--fs-3);

}

.play-btn {

  background-color: var(--coquelicot);

  color: var(--white);

  width: max-content;

  font-size: 30px;

  padding: 25px;

  border-radius: 50%;

  margin-block: 25px 35px;

  animation: pulse 2s ease infinite;

}

@keyframes pulse {

  0% { box-shadow: 0 0 0 0 var(--coquelicot); }

  100% { box-shadow: 0 0 0 40px transparent; }

}

/\*-----------------------------------\*\

  #CLASS

\\*-----------------------------------\*/

.class { padding-block-start: calc(var(--section-padding) + 250px); }

.class .section-subtitle { margin-inline: auto; }

.class .section-title { color: var(--white); }

.class-card {

  background-color: var(--white);

  border-radius: var(--radius-10);

  height: 100%;

  overflow: hidden;

}

.class-card .card-banner img { transition: var(--transition-2); }

.class-card:is(:hover, :focus-within) .card-banner img {

  transform: scale(1.1);

}

.class-card .card-content { padding: 24px; }

.class-card .title-wrapper {

  display: flex;

  align-items: center;

}

.class-card .title-icon {

  padding-inline-end: 20px;

  margin-inline-end: 20px;

  min-width: max-content;

  border-inline-end: 1px solid var(--gainsboro);

}

.class-card .card-title { transition: var(--transition-1); }

.class-card .card-title:is(:hover, :focus) { color: var(--coquelicot); }

.class-card .card-text {

  color: var(--sonic-silver);

  font-size: var(--fs-6);

  margin-block: 16px 12px;

}

.class-card .progress-wrapper {

  display: flex;

  justify-content: space-between;

  align-items: center;

  font-family: var(--ff-catamaran);

  color: var(--rich-black-fogra-29-1);

  font-size: var(--fs-6);

  font-weight: var(--fw-800);

  margin-block-end: 8px;

}

.class-card .progress-bg {

  background-color: var(--coquelicot\_10);

  border-radius: 50px;

}

.class-card .progress-bar {

  background-color: var(--coquelicot);

  height: 10px;

  border-radius: inherit;

}

/\*-----------------------------------\*\

  #BLOG

\\*-----------------------------------\*/

.blog .section-subtitle { margin-inline: auto; }

.blog-card {

  background-color: var(--white);

  border: 1px solid var(--light-gray);

  border-radius: var(--radius-10);

  height: 100%;

  overflow: hidden;

}

.blog-card .card-banner { position: relative; }

.blog-card .card-banner img { transition: var(--transition-2); }

.blog-card:is(:hover, :focus) .card-banner img {

  transform: scale(1.1);

}

.blog-card .card-meta {

  background-color: var(--coquelicot);

  color: var(--white);

  position: absolute;

  bottom: 0;

  left: 0;

  padding: 8px 20px;

  font-size: var(--fs-6);

  font-weight: var(--fw-500);

  text-transform: uppercase;

}

.blog-card .card-content { padding: 25px; }

.blog-card .card-title { transition: var(--transition-1); }

.blog-card .card-title:is(:hover, :focus) { color: var(--coquelicot); }

.blog-card .card-text {

  font-size: var(--fs-6);

  margin-block: 8px 12px;

}

.blog-card .btn-link { --color: var(--coquelicot); }

.blog-card .btn-link:is(:hover, :focus) { --color: var(--rich-black-fogra-29-1); }

/\*-----------------------------------\*\

  #FOOTER

\\*-----------------------------------\*/

.footer { font-size: var(--fs-6); }

.footer-top .container {

  display: grid;

  gap: 50px;

}

.footer .logo { color: var(--white); }

.footer-brand-text { margin-block: 25px; }

.footer-top .wrapper {

  display: flex;

  justify-content: flex-start;

  align-items: flex-start;

  gap: 20px;

}

.footer-brand-list li:not(:last-child) { margin-block-end: 15px; }

.footer-brand-title,

.footer-list-title {

  color: var(--white);

  font-family: var(--ff-catamaran);

}

.footer-list-title {

  font-size: var(--fs-4);

  font-weight: var(--fw-800);

  margin-block-end: 28px;

}

.footer-list-title::before {

  bottom: 0;

  width: 70px;

  height: 1px;

  background-color: var(--coquelicot);

}

.footer-list > li:not(:first-child) { margin-block-start: 12px; }

.footer-link { transition: var(--transition-1); }

.footer-link:not(.address):is(:hover, :focus) { color: var(--coquelicot); }

.footer-list-item {

  display: flex;

  justify-content: flex-start;

  align-items: center;

  gap: 20px;

}

.footer-list-item .icon {

  background-color: var(--coquelicot);

  color: var(--white);

  font-size: 24px;

  padding: 8px;

  border-radius: 50px;

}

.footer-form {

  position: relative;

  margin-block-end: 30px;

}

.footer-form .input-field {

  background-color: var(--white);

  color: var(--rich-black-fogra-29-1);

  padding-block: 18px;

  padding-inline: 30px 80px;

  border-radius: var(--radius-10);

}

.footer-form .btn {

  position: absolute;

  top: 5px;

  right: 5px;

  bottom: 5px;

  padding: 0;

  font-size: 26px;

  padding-inline: 12px;

}

.footer-form .btn-primary:is(:hover, :focus) {

  background-color: var(--rich-black-fogra-29-1);

  color: var(--white);

  box-shadow: none;

}

.social-list {

  display: flex;

  gap: 15px;

}

.social-link {

  background-color: var(--white\_20);

  color: var(--white);

  padding: 13px;

  border-radius: 50%;

  transition: var(--transition-1);

}

.social-link:is(:hover, :focus) { background-color: var(--coquelicot); }

.footer-bottom {

  background-color: var(--rich-black-fogra-29-2);

  color: var(--white);

  text-align: center;

  padding-block: 15px;

}

.copyright-link {

  display: inline-block;

  color: var(--coquelicot);

}

.footer-bottom-list {

  display: flex;

  justify-content: center;

  gap: 15px;

  margin-block-start: 10px;

}

.footer-bottom-link {

  padding-inline-start: 20px;

  transition: var(--transition-1);

}

.footer-bottom-link::before {

  top: 50%;

  transform: translateY(-50%);

  left: 0;

  width: 10px;

  height: 10px;

  background-color: var(--coquelicot);

  border-radius: 50%;

}

.footer-bottom-link:is(:hover, :focus) { color: var(--coquelicot); }

/\*-----------------------------------\*\

  #BACK TO TOP

\\*-----------------------------------\*/

.back-top-btn {

  position: fixed;

  bottom: 20px;

  right: 40px;

  background-color: var(--coquelicot);

  color: var(--rich-black-fogra-29-1);

  font-size: 20px;

  padding: 11px;

  border-radius: 50%;

  border: 2px solid var(--rich-black-fogra-29-1);

  visibility: hidden;

  opacity: 0;

  transition: var(--transition-1);

  z-index: 4;

}

.back-top-btn.active {

  visibility: visible;

  opacity: 1;

  transform: translateY(-10px);

}

/\*-----------------------------------\*\

  #MEDIA QUERIES

\\*-----------------------------------\*/

/\*\*

 \* responsive for larger than 575px screen

 \*/

@media (min-width: 575px) {

  /\*\*

   \* CUSTOM PROPERTY

   \*/

  :root {

    /\*\*

     \* typography

     \*/

    --fs-1: 5.8rem;

    --fs-2: 4rem;

  }

  /\*\*

   \* REUSED STYLE

   \*/

  .container {

    max-width: 540px;

    width: 100%;

    margin-inline: auto;

  }

  .hero-subtitle,

  .section-subtitle { font-size: var(--fs-5); }

  /\*\*

   \* HEADER

   \*/

  .header .container {

    max-width: unset;

    padding-inline: 30px;

  }

  /\*\*

   \* HERO

   \*/

  .hero-content { padding-inline: 40px; }

  .hero-subtitle .strong { padding-block: 6px; }

  .hero::after { height: 340px; }

  .abs-img-1 {

    top: 130px;

    right: -10px;

    width: 230px;

  }

  .abs-img-2 {

    bottom: 20px;

    left: -40px;

    width: 310px;

  }

  /\*\*

   \* ABOUT

   \*/

  .about .wrapper {

    display: flex;

    justify-content: flex-start;

    align-items: center;

    gap: 40px;

  }

  .about-coach { margin-block-end: 0; }

  /\*\*

   \* VIDEO

   \*/

  .video-card .card-title { --fs-3: 3.5rem; }

  /\*\*

   \* FOOTER

   \*/

  .footer-top .container {

    grid-template-columns: 1fr 1fr;

    column-gap: 25px;

  }

}

/\*\*

 \* responsive for larger than 768px screen

 \*/

@media (min-width: 768px) {

  /\*\*

   \* CUSTOM PROPERTY

   \*/

  :root {

    /\*\*

     \* typography

     \*/

    --fs-2: 4.5rem;

  }

  /\*\*

   \* REUSED STYLE

   \*/

  .container { max-width: 720px; }

  .scrollbar-item { min-width: calc(50% - 12.5px); }

  /\*\*

   \* HERO

   \*/

  .hero-banner {

    max-width: max-content;

    margin-inline: auto;

  }

  .abs-img-1 {

    top: 140px;

    right: 50px;

  }

  /\*\*

   \* FOOTER

   \*/

  .footer-bottom .container {

    display: flex;

    justify-content: space-between;

    align-items: center;

  }

  .footer-bottom-list { margin-block-start: 0; }

}

/\*\*

 \* responsive for larger than 992px screen

 \*/

@media (min-width: 992px) {

  /\*\*

   \* REUSED STYLE

   \*/

  .container,

  .header .container { max-width: 960px; }

  /\*\*

   \* HEADER

   \*/

  .nav-open-btn,

  .nav-close-btn { display: none; }

  .header .btn { display: block; }

  .header {

    background-color: transparent;

    box-shadow: none;

    padding-block: 30px;

    transition: var(--transition-1);

  }

  .header.active {

    transform: translateY(-100%);

    background-color: var(--white);

    padding-block: 20px;

    box-shadow: var(--shadow-1);

    animation: slideIn 0.5s ease forwards;

  }

  @keyframes slideIn {

    0% { transform: translateY(-100%); }

    100% { transform: translateY(0); }

  }

  .header .container { gap: 30px; }

  .header .logo { color: var(--white); }

  .header.active .logo { color: var(--rich-black-fogra-29-1); }

  .navbar,

  .navbar.active {

    all: unset;

    margin-inline-start: auto;

  }

  .navbar-list {

    display: flex;

    gap: 10px;

  }

  .navbar-link {

    color: var(--white);

    font-size: unset;

    padding: 0 10px;

    margin-block-end: 0;

  }

  .header.active .navbar-link { color: var(--rich-black-fogra-29-1); }

  .header .navbar-link:is(:hover, :focus, .active) { color: var(--coquelicot); }

  .header.active .btn {

    background-color: var(--coquelicot);

    color: var(--white);

  }

  .header.active .btn:is(:hover, :focus) { background-color: var(--rich-black-fogra-29-1); }

  /\*\*

   \* HERO

   \*/

  .hero {

    background-size: contain;

    text-align: left;

  }

  .hero::before {

    content: "";

    position: absolute;

    top: -1000px;

    left: -500px;

    width: 1500px;

    height: 1500px;

    background-image: radial-gradient(circle, var(--coquelicot\_20) 20%, transparent 70% 100%);

    z-index: -1;

  }

  .hero .container {

    display: grid;

    grid-template-columns: 1fr 1fr;

    align-items: center;

    gap: 25px;

  }

  .hero-content {

    padding-inline: 0;

    margin-block-end: 0;

  }

  .hero-subtitle,

  .hero .btn { margin-inline: 0; }

  .hero::after {

    width: 330px;

    height: 100%;

    left: auto;

    right: 0;

  }

  /\*\*

   \* ABOUT

   \*/

  .about .container {

    display: grid;

    grid-template-columns: 1fr 1fr;

    align-items: center;

    gap: 50px;

  }

  .about-banner { margin-block-end: 0; }

  .about .wrapper { gap: 30px; }

  /\*\*

   \* FOOTER

   \*/

  .footer-top .container {

    grid-template-columns: 0.85fr 0.5fr 1fr 0.85fr;

    column-gap: 50px;

  }

}

/\*\*

 \* responsive for larger than 1200px screen

 \*/

@media (min-width: 1200px) {

  /\*\*

   \* CUSTOM PROPERTY

   \*/

  :root {

    /\*\*

     \* typography

     \*/

    --fs-1: 7rem;

    --fs-2: 5.5rem;

    --fs-4: 2.2rem;

    --fs-5: 2rem;

    /\*\*

     \* spacing

     \*/

    --section-padding: 120px;

  }

  /\*\*

   \* REUSED STYLE

   \*/

  .container,

  .header .container { max-width: 1140px; }

  .btn {

    padding: 18px 45px;

    border-radius: var(--radius-10);

  }

  .section-subtitle { --fs-5: 2.2rem; }

  .has-scrollbar { gap: 30px; }

  .scrollbar-item { min-width: calc(33.33% - 20px); }

  /\*\*

   \* HEADER

   \*/

  .header .container { padding-inline: 0; }

  /\*\*

   \* HERO

   \*/

  .hero::after { width: 420px; }

  .hero .section-text { --fs-6: 1.8rem; }

  .abs-img-1 {

    top: 170px;

    right: -30px;

    width: 260px;

  }

  .abs-img-2 {

    bottom: 60px;

    left: -80px;

    width: 360px;

  }

  /\*\*

   \* ABOUT

   \*/

  .about .wrapper { gap: 40px; }

  /\*\*

   \* CLASS, BLOG

   \*/

  :is(.class-card, .blog-card) .card-content { padding: 30px; }

  .blog-card .card-meta { padding: 15px 30px; }

  /\*\*

   \* FOOTER

   \*/

  .footer-top .container { grid-template-columns: 1fr 0.6fr 0.9fr 1fr; }

}

#### **Javascript:**

'use strict';

/\*\*

 \* add event on element

 \*/

const addEventOnElem = function (elem, type, callback) {

  if (elem.length > 1) {

    for (let i = 0; i < elem.length; i++) {

      elem[i].addEventListener(type, callback);

    }

  } else {

    elem.addEventListener(type, callback);

  }

}

/\*\*

 \* navbar toggle

 \*/

const navbar = document.querySelector("[data-navbar]");

const navTogglers = document.querySelectorAll("[data-nav-toggler]");

const navLinks = document.querySelectorAll("[data-nav-link]");

const toggleNavbar = function () { navbar.classList.toggle("active"); }

addEventOnElem(navTogglers, "click", toggleNavbar);

const closeNavbar = function () { navbar.classList.remove("active"); }

addEventOnElem(navLinks, "click", closeNavbar);

/\*\*

 \* header & back top btn active

 \*/

const header = document.querySelector("[data-header]");

const backTopBtn = document.querySelector("[data-back-top-btn]");

window.addEventListener("scroll", function () {

  if (window.scrollY >= 100) {

    header.classList.add("active");

    backTopBtn.classList.add("active");

  } else {

    header.classList.remove("active");

    backTopBtn.classList.remove("active");

  }

});