Muhammad Abdullah ***01-134211-049***
Awais Ali ***01-134211-014***

AI-Enhanced-Fitness-Wellness-Analyzer-Project

Contents

Muhammad Abdullah ***01-134211-049***
Awais Ali ***01-134211-014***

Data Analysis:

1. Exploring and Preprocessing:

1. 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())
```

2. *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
```

Muhammad Abdullah *01-134211-049*
Awais Ali *01-134211-014*

```
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}")
```

3. *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())
```

Muhammad Abdullah ***01-134211-049***
Awais Ali ***01-134211-014***

```
# 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=LightlyActiveMinutesUpperl
imit)
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=moderatelyActiveDistan
ceUpperLimit)

# 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=lightActiveDistanceUpperLim
it)

# 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=sedentaryActiveDistance
UpperLimit)

# 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}")
```

4. *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}")
```

5. *heartrate_seconds_merged.ipynb:*

```
import pandas as pd
```

Muhammad Abdullah ***01-134211-049***
Awais Ali ***01-134211-014***

```
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}")
```

6. *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}")
```

7. *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}")
```

8. *hourlySteps_merged.ipynb:*

```
import pandas as pd

filePath = '/University/6th Semester/Sixth Semester/IDS-
AIProject/FitabaseData4.12.16-5.12.16/hourlySteps_merged.csv'
```

Muhammad Abdullah ***01-134211-049***
Awais Ali ***01-134211-014***

```
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}")
```

9. *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}")
```

10. 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}")
```

11. 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}")
```

12. 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())
```

Muhammad Abdullah ***01-134211-049***
Awais Ali ***01-134211-014***

```
# 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}")
```

13. 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}")
```

14. 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}")
```

15. 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}")
```

16. 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}")
```

17. 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}")
```

18. 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()
```

2. Data Visualization:

1. 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_Fil
tered.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_Box
Plot_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_distri
bution_ScatterPlot_Filtered.png')  # Save the figure
plt.show()
```

2. *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()
```

3. *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()
```

4. *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()
```

5. *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()
```

6. *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_bur
ned_cleaned.png')  # Save the figure
plt.show()
```

7. *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()
```

8. *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_clea
ned.png')  # Save the figure
plt.show()
```

9. *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_sele
cted_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()
```

10. 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_select
ed_features_cleaned.png')  # Save the figure
plt.show()
```

11. 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()
```

12. 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()
```

13. 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&ActivityMinu
te_ScatterPlot.png')  # Save the figure
plt.show()
```

14. 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()
```

15. 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_distributi
on.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.pn
g')  # 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_d
istribution.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_B
oxPlot.png')  # Save the figure
plt.show()
```

16. 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()
```

17. 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_distributi
on.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.pn
g')  # Save the figure
plt.show()
```

18. 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_distributio
n.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:

1. RecommendationSystem:

1. 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)
```

2. *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(asce
nding=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_da
ta['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)
```

Muhammad Abdullah ***01-134211-049***
Awais Ali ***01-134211-014***

```
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)
```

3. *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}")
```

4. *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']
```

Muhammad Abdullah ***01-134211-049***
Awais Ali ***01-134211-014***

```
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,
}
```

Muhammad Abdullah ***01-134211-049***
Awais Ali ***01-134211-014***

```
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:

1. HealthTrendAnalysis

1. *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)
```

Muhammad Abdullah ***01-134211-049***
Awais Ali ***01-134211-014***

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)
```

Muhammad Abdullah ***01-134211-049***
Awais Ali ***01-134211-014***

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;80
0;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>
        </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>
          </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>
```

Muhammad Abdullah ***01-134211-049***
Awais Ali ***01-134211-014***

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
\*-----------------------------------*/
```

Muhammad Abdullah *01-134211-049*
Awais Ali *01-134211-014*

```css
*,
*::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; }
```

Muhammad Abdullah *01-134211-049*
Awais Ali *01-134211-014*

```
.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
\*-----------------------------------*/
```

Muhammad Abdullah *01-134211-049*
Awais Ali *01-134211-014*

```css
.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; }
```

Muhammad Abdullah *01-134211-049*
Awais Ali *01-134211-014*

```
.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;
}




/*-----------------------------------*\
```

Muhammad Abdullah *01-134211-049*
Awais Ali *01-134211-014*

```
  #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); }
```

Muhammad Abdullah *01-134211-049*
Awais Ali *01-134211-014*

```css
/*-----------------------------------*\
  #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; }
```

Muhammad Abdullah *01-134211-049*
Awais Ali *01-134211-014*

```
.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;
}
```

```css
.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");
  }
});
```