Annual Hackathon 2025

Reflex, Cain Road, Bracknell, Berkshire RG12 1HL
T: 01344 203999 F 01344 203998
info@theknowledgeacademy.com – www.theknowledgeacademy.com

Table of Contents

Introduction

Welcome to the **Annual Hackathon 2025.** This hackathon is designed to bring professionals together to solve real-world challenges using Data Analysis (DA) and Artificial Intelligence (AI).

Our goal is to equip participants with the skills and tools needed to complete a full DA/AI pipeline while fostering creativity and collaboration. Whether you're a seasoned professional or someone exploring new skills, this handbook will guide you every step of the way.

The event is structured into four major tasks:

1. **Data Preprocessing** – Clean and prepare your data for analysis.
2. **Insights (EDA)** – Explore the dataset to uncover meaningful patterns.
3. **AI Tasks** – Build models to address key questions or challenges.
4. **Report Generation** – Summarize your findings and share actionable insights.

Use this handbook to follow along with demos, step-by-step instructions, and best practices to excel in every task. Good luck and have fun creating innovative solutions!

Hackathon Guidelines

Timeline and Milestones

The hackathon spans two days. On the first day, participants will focus on completing the first three tasks. Those who solve these tasks correctly and effectively will be shortlisted for the second day, where they will work on Task 4. Their submissions for Task 4 will be evaluated by our judging panel.

Rules and Regulations

- Teams can consist of **3 to 5 members**.
- Only **original work** is allowed. Plagiarised or pre-built projects will lead to disqualification.
- All deliverables must be submitted through the official platform.
- Follow the instructions mentioned in the notebook.
- Follow ethical guidelines while coding and presenting your solutions.

Resources Provided

Participants will have access to:

- Sample datasets for tasks.
- Hackathon Questionnaire in a python notebook.
- Pre-configured virtual environments for AI/DA projects.
- Documentation and code repositories.

Support Channels

Dedicated support will be available via:

1. **Deloitte Representatives**: Expert guidance on technical queries.
2. **TKA Representatives**: Help with event logistics and tools.

Use these channels proactively to resolve issues or gain insights.

Task Overview

Breakdown of the Four Major Tasks

1. **Data Preprocessing**: Clean and prepare data for analysis.
2. **Exploratory Data Analysis (EDA)**: Identify patterns and insights in the dataset.
3. **AI Modelling**: Build predictive models to address challenges.
    - **Prompt Engineering**: Write clear and concise prompts to extract the specific information you need from the provided chatbot.
4. **Report Generation**: Create a concise and actionable summary of findings.

Each task builds upon the previous one, creating a seamless workflow from data preparation to project delivery.

Dependencies and Flow Between Tasks

- **Data Preprocessing** is essential for accurate EDA and model building.
- **EDA Insights** inform model design and evaluation.
- **AI Modelling** results contribute to meaningful reporting.
- **Report Generation** highlights the overall impact and conclusions.

Tools and Technologies to Use

Participants are encouraged to utilise:

- **Programming Languages**: Python
- **Frameworks**: TensorFlow, PyTorch, Pandas, Scikit-learn
- **Visualisation Tools**: Matplotlib, Seaborn
- **Collaborative Platforms**: Teams

Experimenting with these tools is recommended before the event for maximum efficiency and impact.

Pre-Setup Tasks

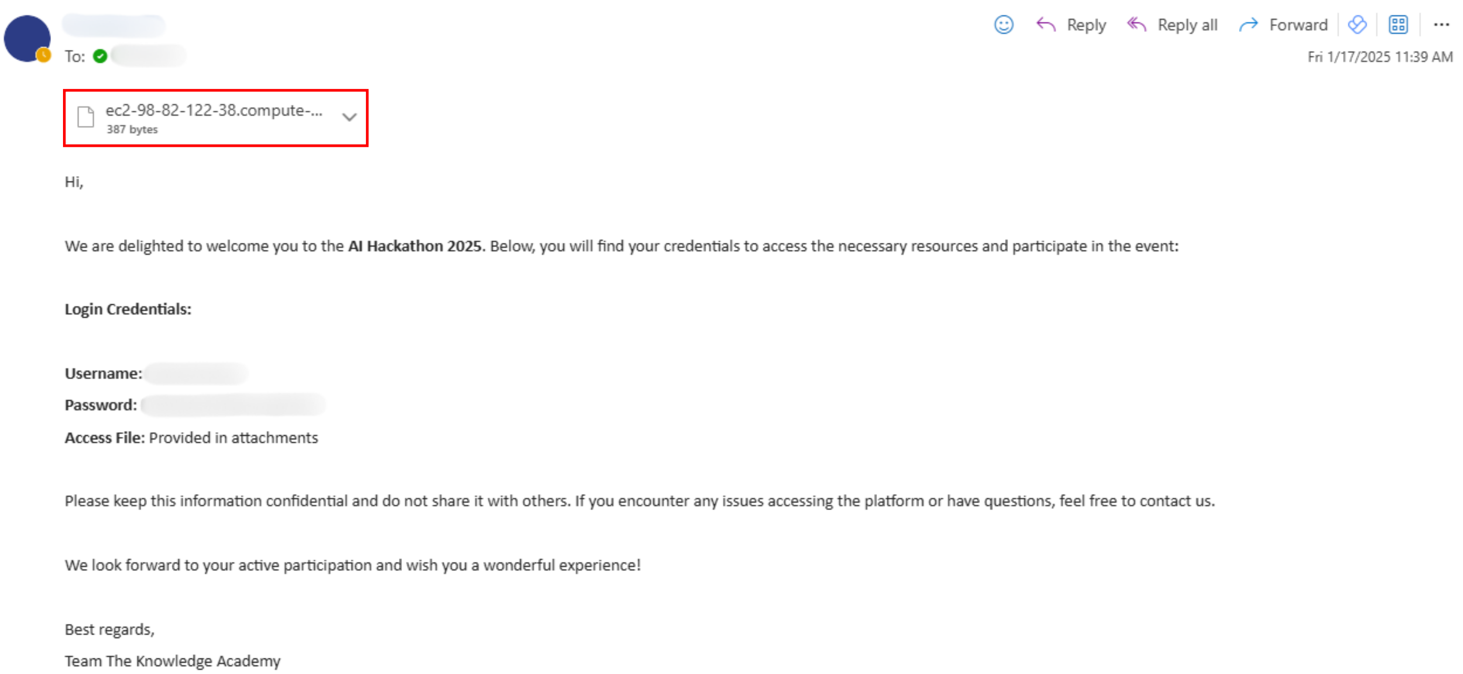
Before diving into the hackathon tasks, let’s take a moment to complete some essential setup steps. These include:

1. Familiarizing Yourself with the Platform:
    - Understand the tools, environment, and resources provided for the hackathon.
2. Loading the Data:
    - Import the dataset into your working environment to prepare for analysis and modeling.

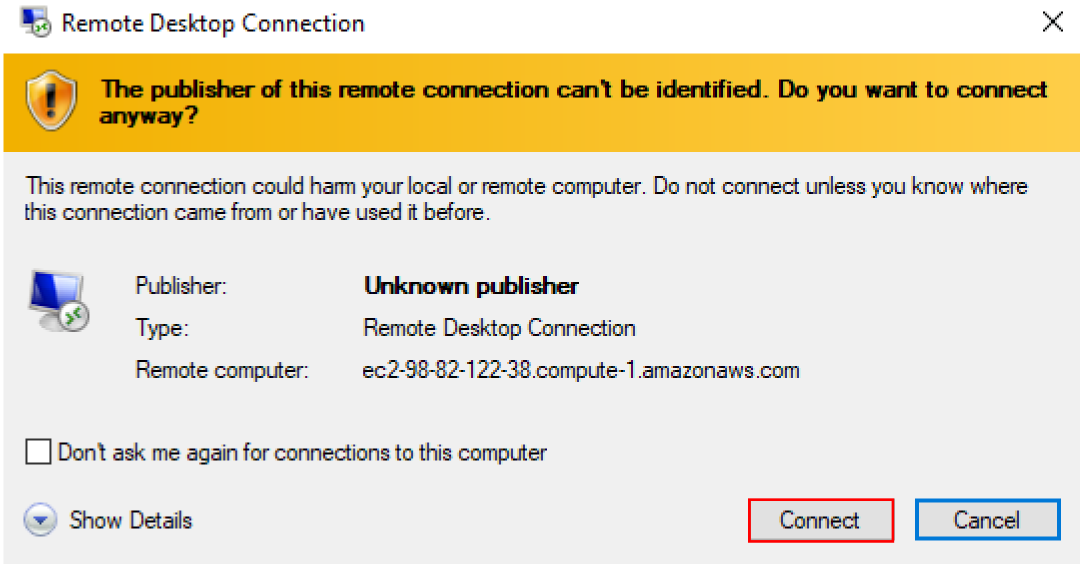
Familiarizing with the Platform

Let’s get started with the process. All participants should receive an email like this from a TKA representative.

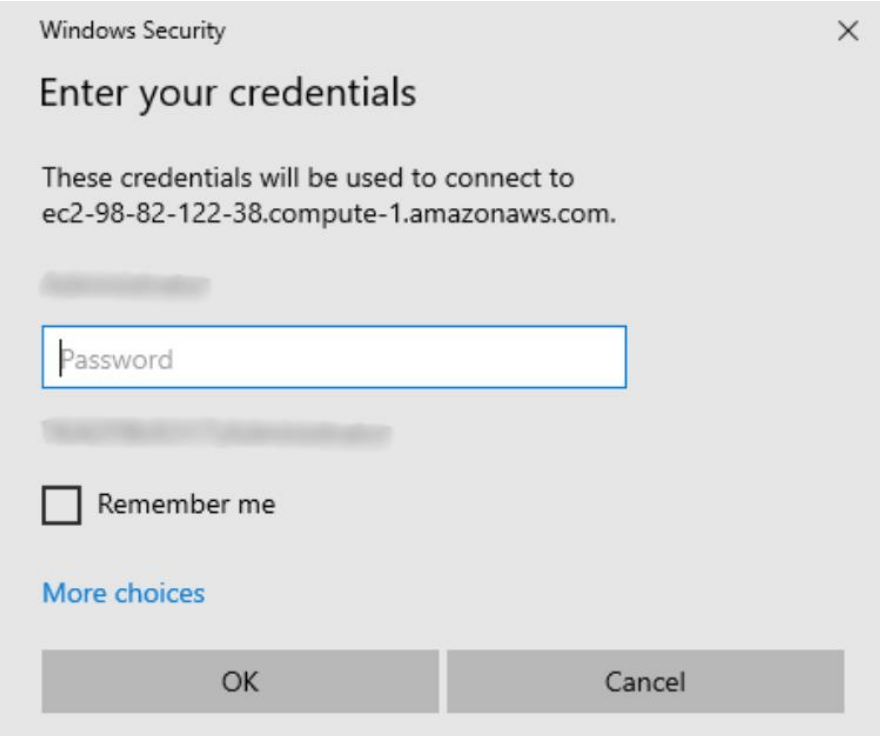
The email will contain the VM (Highlighted in red), as well as the credentials required to login to the VM.



Run the VM from your downloads. Click on connect when this prompt shows up.

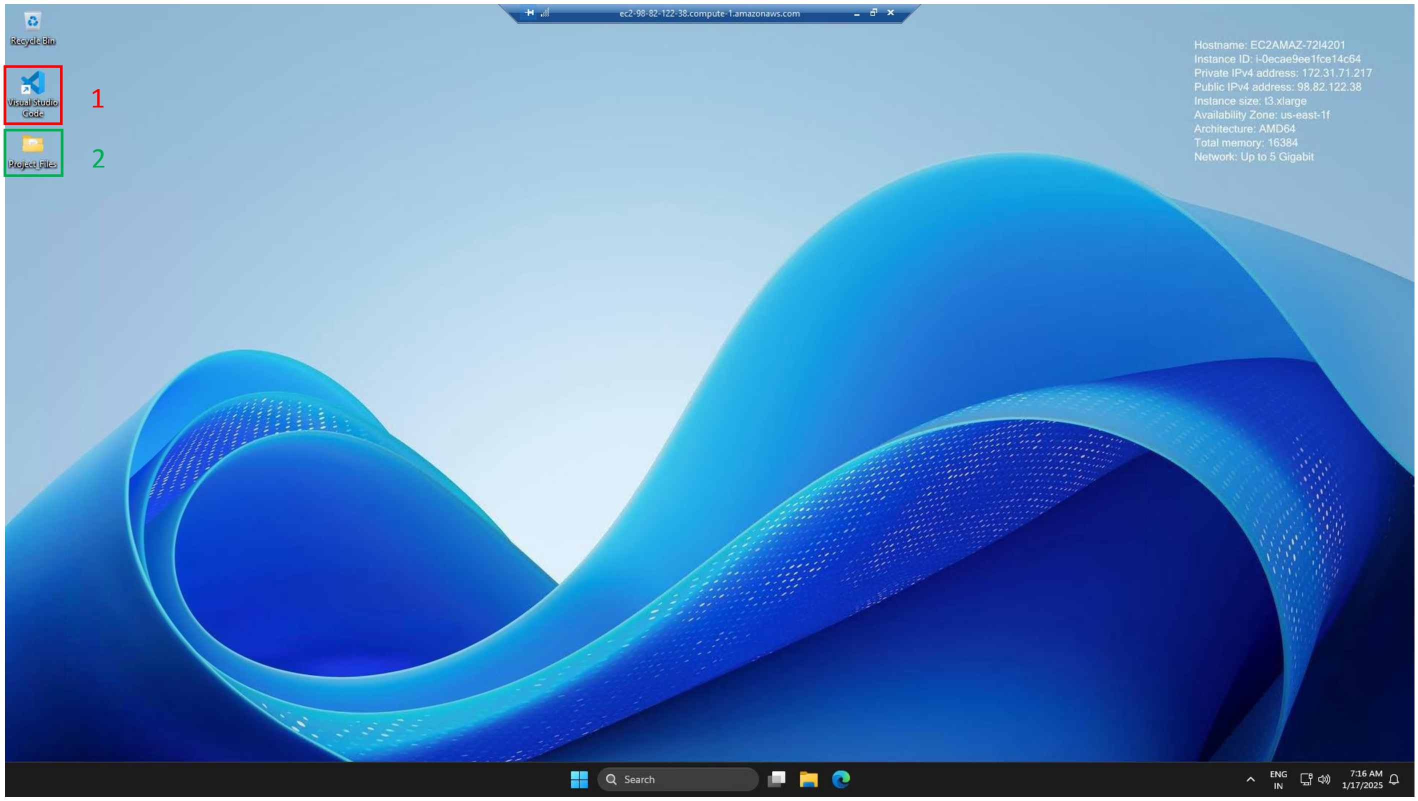


Enter the password shared with you on your email to connect to the VM.

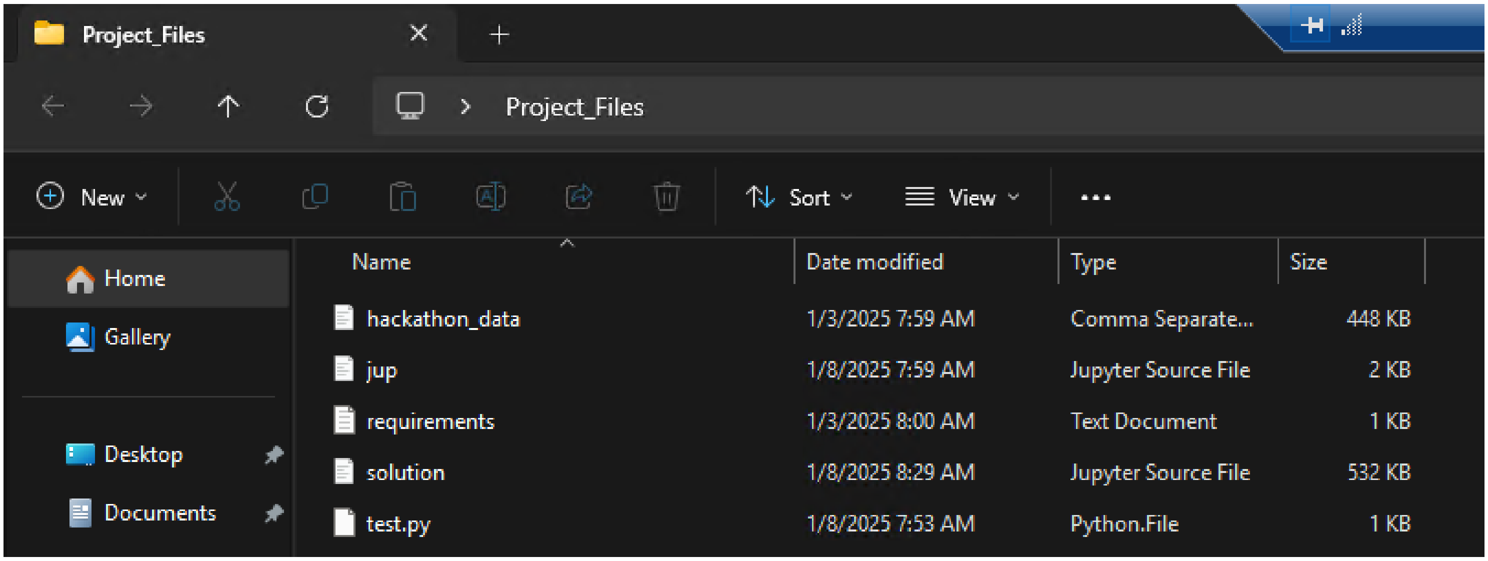


With this Notification, simply click on yes to start the VM

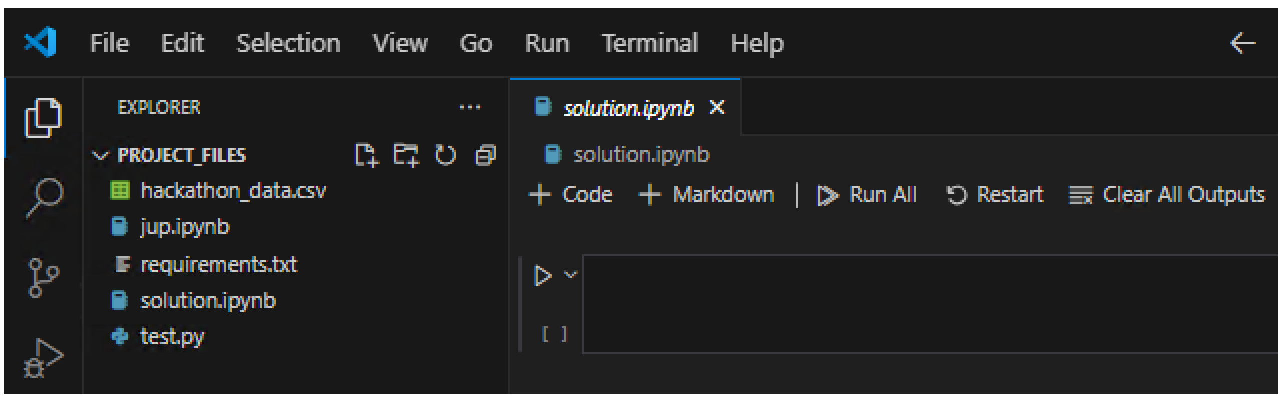
You will be greeted with this screen. This is the VM. From here you can access the Visual Studio Code (1) and the hackathon files (2)



The following images are the files that will be used for the hackathon.



The following is what is shown to you when you open the VSCode. It will open directly into the solutions Notebook.



Loading the Data

We will provide you with the dataset for this hackathon, which is located in the same directory as the Notebook. You can find it when you open VSCode. To get started, you'll need to load the data using pandas and import a few essential libraries for Task 1 and Task 2. The imports specific to Task 3 will be introduced later.

Here's how to load the data and the necessary imports for Task 1 and Task 2:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

data = pd.read_csv("DemoData.csv")
```

Make sure the file name matches the dataset provided. These imports will help with data preprocessing, visualization, and analysis.

Task 1: Data Preprocessing

What is Data Preprocessing?

Data preprocessing is the foundational stage in any data analysis or AI project. This process involves cleaning, transforming, and organizing raw data to make it usable for analysis or model building. Poorly preprocessed data can lead to misleading insights, inaccurate models, or even total project failure.

Why is Data Preprocessing Important?

- **Ensures Data Quality**: Raw data often contains errors, missing values, or inconsistencies. Preprocessing resolves these issues, ensuring reliable results.
- **Reduces Noise**: Unnecessary or irrelevant data can distract from meaningful patterns. Preprocessing removes these distractions.
- **Improves Model Performance**: Well-structured and clean data enhances the accuracy and efficiency of AI models.
- **Standardizes Formats**: Diverse datasets often come in different formats. Preprocessing ensures compatibility and uniformity.

Raw Data (Without Preprocessing):

User ID	Age	Gender	Workout Type	Calories Burned	BMI	Workout Duration (mins)
101	25	Male	Running		22.5	45
102	34	Female	Cycling	310		60
103		Male	Yoga	150	19.8	-30
104	29		Running	450	23.0	50

Issues in Raw Data:

1. Missing values for **Calories Burned**, **BMI**, and **Age**.
2. Incorrect values like negative Workout Duration.
3. Inconsistent data, such as missing **Gender**.
4. No standardization of units (e.g., some data might use seconds instead of minutes for duration).

Preprocessed Data (After Preprocessing):

User ID	Age	Gender	Workout Type	Calories Burned	BMI	Workout Duration (mins)
101	25	Male	Running	350	22.5	45
102	34	Female	Cycling	310	21.7	60
103	30	Male	Yoga	150	19.8	30
104	29	Female	Running	450	23.0	50

Steps Taken in Preprocessing:

1. **Missing Values Handled:** Estimated calories for User ID 101 using averages or predictive methods.
2. **Incorrect Data Corrected:** Replaced negative workout duration (-30) with a valid value based on data trends.
3. **Inconsistent Data Fixed:** Filled in missing Gender using context or a default value.
4. **Standardization:** Ensured all durations were in minutes.

Key Components of Data Preprocessing

1. **Data Cleaning**
   - Handling missing values (e.g., imputation or removal).
   - Detecting and removing duplicates.
   - Identifying and correcting outliers.
2. **Data Transformation**
   - Normalization or standardization of numerical features.
   - Encoding categorical data for compatibility with algorithms.
   - Creating new features from existing ones (feature engineering).
3. **Data Integration**
   - Combining data from multiple sources.
   - Resolving conflicts in data formats or schemas.
4. **Data Reduction**
   - Dimensionality reduction techniques to simplify data without losing critical information.
   - Sampling methods for handling large datasets efficiently.

Common Challenges in Data Preprocessing

- **Messy or Incomplete Data**: Real-world datasets are rarely perfect.
- **Overfitting Risks**: Over-preprocessing can lead to loss of important details.
- **Choosing the Right Techniques**: The methods depend on the dataset and the problem.

Best Practices

- Understand the dataset thoroughly before starting preprocessing.
- Document each step to maintain transparency and reproducibility.
- Visualize the data at various stages to check progress.
- Validate the processed data by running small experiments or analyses.

Steps to carry out Data Processing.

Step 1: Identify Issues

Identify the issues with the dataset, but running a check on the Nulls in each column and generate a boxplot of all numerical columns to verify the existence (or lack of) Outliers.

Outliers are data instances that significantly differ from the rest of the data. These values appear far outside the normal distribution or range of the majority of the data points, often referred to as "freak values."

An outlier can be a value that is much larger or much smaller than the others in a dataset. For example, if one employee has an unusually high `Weight` (e.g., 500 kg) or a very long `Session Duration (hrs)` (e.g., 100 hours), these would be considered outliers as they deviate significantly from the rest of the data.

Outliers can occur due to various reasons:

- Data entry errors or mistakes (e.g., a typo)
- Natural variation in the data (e.g., exceptionally talented individuals)
- Special or rare cases that are important for the analysis but might skew results

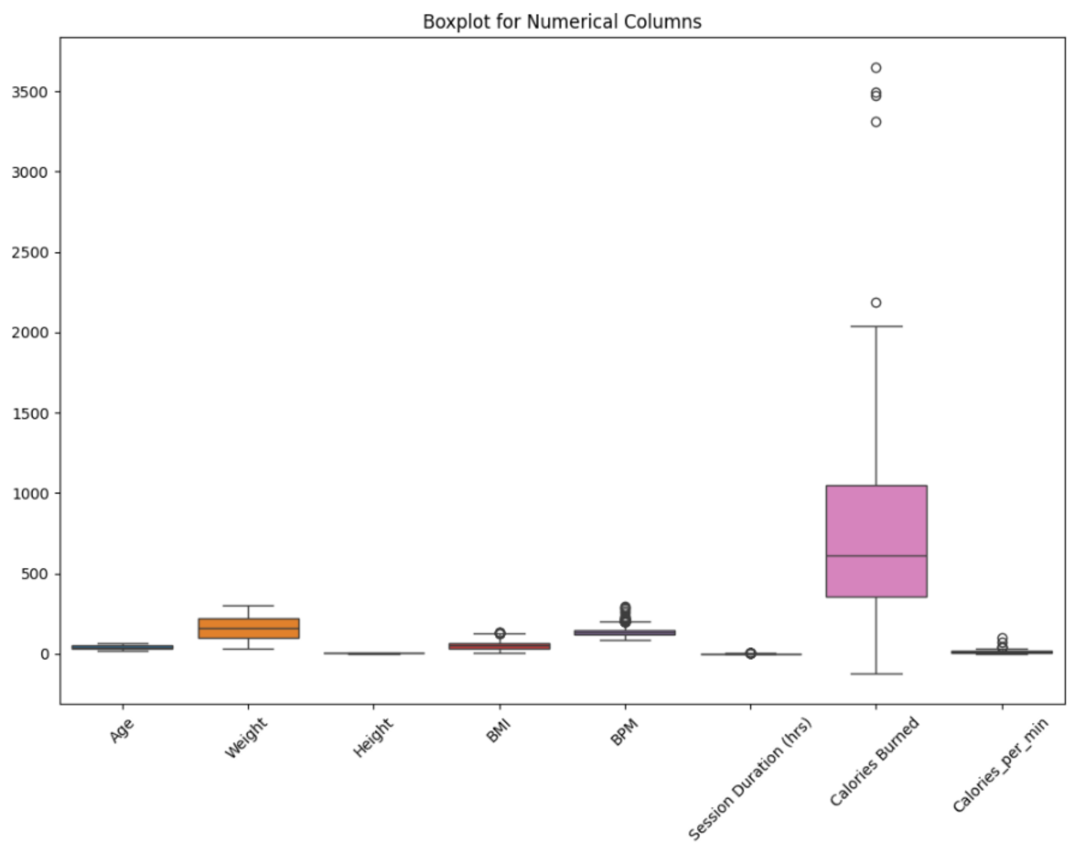
Outliers can distort statistical analyses and lead to incorrect conclusions, which is why it's important to identify and handle them appropriately.

Identifying Outliers

One of the easiest and most common methods for identifying outliers is using a boxplot. A boxplot visually represents the distribution of the data, showing the range and the spread of the data in quartiles. The key components of a boxplot are:

- 0-25% (Lower Quartile): The first quartile, or the 25th percentile.
- 26-50% (Median): The middle value of the dataset.
- 51-75% (Upper Quartile): The third quartile, or the 75th percentile.
- 75-100%: The maximum value.

```
plt.figure(figsize=(12, 8))
sns.boxplot(data=numeric_data)
plt.title('Boxplot for Numerical Columns')
plt.xticks(rotation=45)
plt.show()
```



The box in a boxplot shows the interquartile range (IQR), which represents the middle 50% of the data. Any data points that fall outside of this range are considered potential outliers. These outliers will appear as tiny circles or points outside the upper and lower "whiskers" of the boxplot, marking their position in the distribution.

Identifying Nulls

Null values represent missing information. If not addressed, they can lead to incorrect results during analysis, such as skewed averages or misleading trends. For example, calculating the average calorie burn from incomplete data will produce an inaccurate figure. Most analytical tools and machine learning models cannot handle null values directly. Leaving them unaddressed can cause errors or crashes during processing, rendering the data unusable. Identifying nulls provides insights into where and why data is missing. This can highlight issues in data collection processes, such as faulty sensors or incomplete surveys.

You can check for Nulls using the following the code:

```
data.isna().sum()
```

	0
Gender	0
Age	0
Weight	0
Height	0
BMI	40
BPM	40
Workout	0
Session Duration (hrs)	40
Calories Burned	40

dtype: int64

Step 2: Filling Nulls

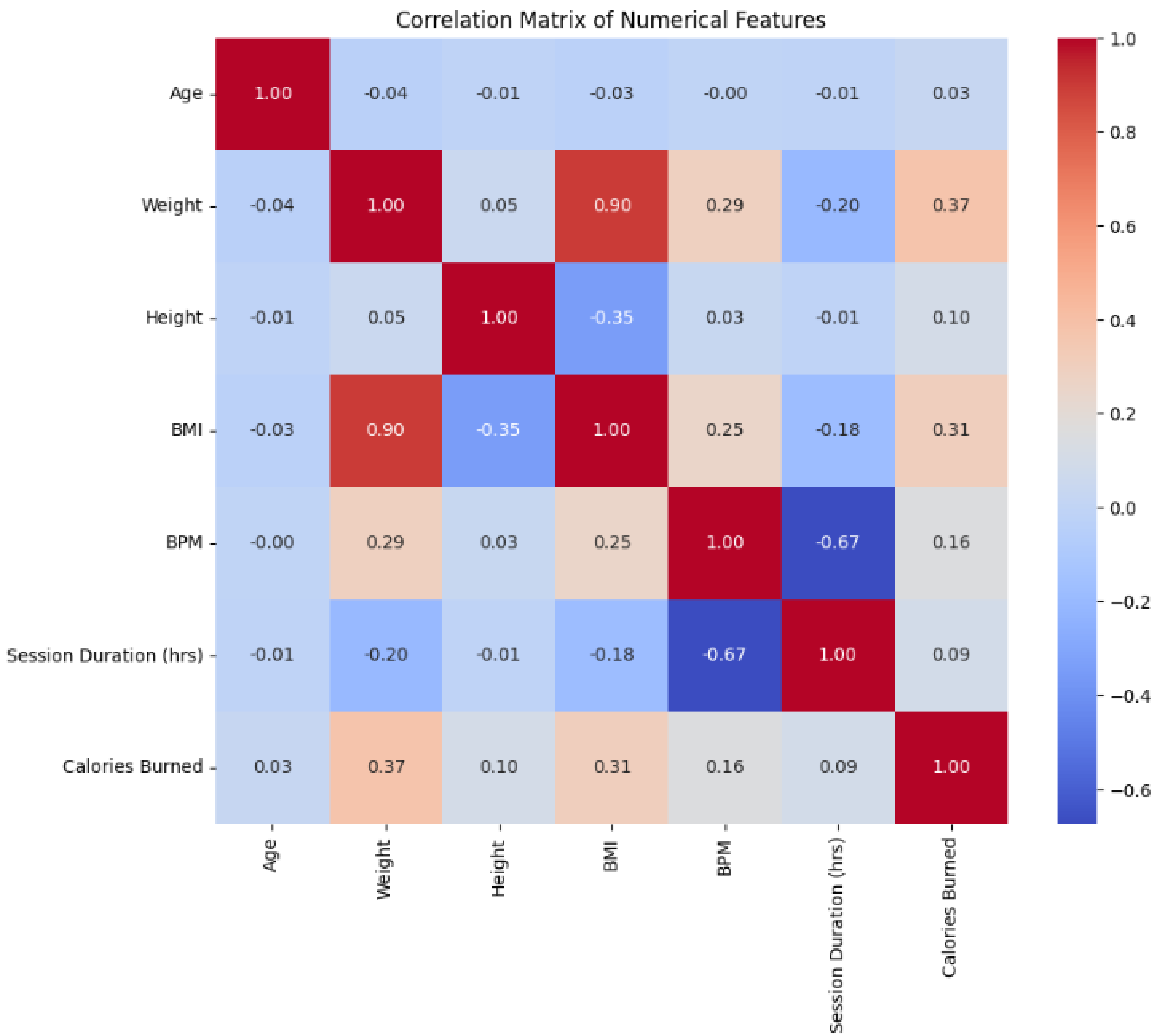
Find Correlation

When faced with missing data, it's essential to go beyond traditional methods and think creatively. Every dataset tells a unique story, and sometimes the best way to fill in the blanks is to uncover the hidden relationships between the data points. Rather than simply applying blanket techniques like filling with the mean or dropping rows, challenge yourself to explore deeper patterns and correlations.

For example, if you notice a strong correlation between BMI and Weight or between Calories Burned and Session Duration, you can leverage these relationships to estimate missing values. By doing so, you're not just filling gaps; you're enriching your data, ensuring that your analysis remains as accurate and insightful as possible. Remember, the goal isn't just to clean the data, but to do so in a way that preserves the integrity and meaning of the dataset.

For this example, we only have numerical values. Let us see if there is any correlation by plotting a heatmap:

```
numeric_data = data.select_dtypes(include=['number'])

corr_matrix = numeric_data.corr()

plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix of Numerical Features')
plt.show()
```



From the heatmap, we can infer the following:

- **BMI** is strongly correlated with **Weight** (0.9)

With that information, we can fill the Nulls and NaNs in a more meaningful manner.

Some of the more common and simple methods to fill values are explained below:

- **Mean:** Replace numerical null values with the average of that column.
- **Median:** Use the middle value for numerical columns to avoid influence from outliers.
- **Mode:** For categorical data, fill with the most frequently occurring value.

Step 3.1: Filling Nulls in BMI

Use the average **BMI-to-Weight ratio** from rows with non-missing values to estimate missing **BMI**.

1. Calculate the **BMI-to-Weight ratio** for non-missing values

```
ratio = data['BMI'] / data['Weight']
median_ratio = ratio.median()
```

2. Estimate missing **BMI**

```
data.loc[data['BMI'].isnull(), 'BMI'] = data['Weight'] * median_ratio
```

Note: Since BPM and Session Duration are interdependent and both have missing values, we will fill the missing entries using the median (if outliers have not been addressed) or the mean (if outliers have been handled), rather than employing a more complex imputation method.

Step 3.2: Filling Nulls with Median

The primary reason for using the Median is its robustness against outliers. The Mean is sensitive to extreme values (outliers), which can skew the average and potentially lead to inaccurate imputations. The Median, on the other hand, represents the middle value when the data is sorted, making it less affected by outliers. This is especially useful when dealing with data that may contain unusual or extreme values.

```
data['BMI'].fillna(data['BMI'].median(), inplace=True)
data['BPM'].fillna(data['BPM'].median(), inplace=True)
data['Session Duration (hrs)'].fillna(data['Session Duration (hrs)'].median(), inplace=True)
data['Calories Burned'].fillna(data['Calories Burned'].median(), inplace=True)
```

Note: If you choose to fill **BMI** and **Calories Burned** using the correlation-based method outlined in Steps 3.1 and 3.4, do not use the above approach of filling missing values with the median.

Step 3.3: Filling Nulls with Mean

If you've already handled outliers (step 4) in your dataset you can fill the null values using the Mean of the column.

While the Mean can be sensitive to extreme values (outliers), if you've already addressed these outliers in the previous steps, the Mean can be a suitable and straightforward choice. It reflects the average value of the data and works well when the distribution is relatively normal and free of extreme values.

```
data['BMI'].fillna(data['BMI'].mean(), inplace=True)
data['BPM'].fillna(data['BPM'].mean(), inplace=True)
data['Session Duration (hrs)'].fillna(data['Session Duration (hrs)'].mean(), inplace=True)
data['Calories Burned'].fillna(data['Calories Burned'].mean(), inplace=True)
```

Note: If you choose to fill **BMI** and **Calories Burned** using the correlation-based method outlined in Steps 3.1 and 3.4, do not use the above approach of filling missing values with the median.

Step 3.4: Filling Nulls in Calories Burned

Calories Burned depends on **Session Duration**. Using a simple proportional method, we can calculate a **Calories Burned per Session Minute** ratio and use it to estimate missing values.

1. Calculate the ratio:

```
data['Calories_per_min'] = data['Calories Burned'] / (data['Session Duration (hrs)'] * 60)
```

2. Estimate missing **Calories Burned**:

```
median_calories_per_min = data['Calories_per_min'].median()
data.loc[data['Calories Burned'].isnull(), 'Calories Burned'] = median_calories_per_min * data['Session Duration (hrs)'] * 60
```

Step 3.5: Filling Nulls with Mode

In the case of Categorical data, we can fill Nulls using Mode. An example:

```
data['Gender'].fillna(data['Gender'].mode()[0], inplace=True)
```

Step 3.6: Verify Columns

In case any null values still remain after the initial imputation steps, implement a redundancy check to fill these remaining nulls using the same method as before: either the median, mean, or mode, depending on the method previously applied.

```
# Redundancy code to fill in any remaining Nulls using median:

data['BMI'].fillna(data['BMI'].median(), inplace=True)
data['BPM'].fillna(data['BPM'].median(), inplace=True)
data['Session Duration (hrs)'].fillna(data['Session Duration (hrs)'].median(), inplace=True)
data['Calories Burned'].fillna(data['Calories Burned'].median(), inplace=True)
```

Run the following code to display all columns and verify whether all null values have been successfully cleared:

```
data.isna().sum()
```

	0
Gender	0
Age	0
Weight	0
Height	0
BMI	0
BPM	0
Workout	0
Session Duration (hrs)	0
Calories Burned	0
Calories_per_min	40

Any temporary columns (such as **Calories_per_min**) created to fill null values in other columns can be safely dropped after the imputation process is complete.

```
data.drop(columns=['Calories_per_min'], inplace=True)
```

Reflex, Cain Road, Bracknell, Berkshire RG12 1HL
T: 01344 203999 F 01344 203998
info@theknowledgeacademy.com – www.theknowledgeacademy.com

Step 4: Fixing Outliers

To fix outliers, we will be using the Inter Quartile Range method.

The IQR represents the range between the first quartile (Q1) and third quartile (Q3) of a dataset, capturing the middle 50% of the data. Any values outside the range defined by 1.5 times the IQR above Q3 or below Q1 are considered outliers.

Steps to Detect Outliers Using IQR:

1. **Calculate Q1 and Q3:**

   The first and third quartiles represent the 25th and 75th percentiles, respectively.

2. **Compute the IQR:**

   IQR = Q3 − Q1

3. **Determine Outlier Bounds:**

   - **Lower Bound:** Q1 − 1.5 × IQR
   - **Upper Bound:** Q3 + 1.5 × IQR

4. **Identify Outliers:**

   Any data points outside of the lower and upper bounds are considered outliers.

Example:

In the case of columns like BMI, BPM, Calories Burned, and Session Duration (hrs), outliers can be detected using IQR and then removed. For instance:

A BMI of 50 could be an outlier if most of the data falls between 20 and 30.

Similarly, a Session Duration (hrs) of 100 hours might not be realistic in the context of regular workout sessions.

Handling Outliers:

After detecting outliers, you have a few options for dealing with them:

- **Remove Outliers:** If the outliers are due to data errors or are too extreme, you may choose to remove the rows containing these values.
- **Cap or Transform Values:** For some cases, capping outliers to a certain threshold or using transformations (e.g., log transformation) may be more appropriate.

The IQR method is effective for detecting outliers in numerical data and can help ensure the accuracy of subsequent analyses and modelling.

Removing Outliers

To remove outliers, we can use the following code:

```
# Function to remove outliers using IQR
def remove_outliers(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
    return df[(df[column] >= lower_bound) & (df[column] <= upper_bound)]

# Remove outliers for each column
data = remove_outliers(data, 'BMI')
data = remove_outliers(data, 'BPM')
data = remove_outliers(data, 'Calories Burned')
data = remove_outliers(data, 'Session Duration (hrs)')

# Optionally, reset index after removing rows with outliers
data.reset_index(drop=True, inplace=True)
```

Capping Outliers

Capping is the process of limiting the extreme values (outliers) in a dataset by setting them to a specific threshold. This is done by replacing any values outside a defined range (e.g., lower than the 5th percentile or higher than the 95th percentile) with the nearest acceptable value within that range.

How It Works:

- **Lower Bound:** Any values below the lower threshold are set to the lower bound.
- **Upper Bound:** Any values above the upper threshold are set to the upper bound.

Purpose:

Capping prevents outliers from disproportionately affecting statistical measures (like mean or standard deviation) and model performance, while keeping the rest of the data intact.

Example:

If your dataset's BMI has values between 18 and 30, but you have a value of 50, capping would replace 50 with the upper bound (say, 30), making it less extreme but still retaining the value within a reasonable range.

To cap outliers, we can use the following code:

```
# Function to cap outliers using IQR
def cap_outliers(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

    # Cap outliers
    df[column] = df[column].clip(lower=lower_bound, upper=upper_bound)
    return df

# Cap outliers for each column
data = cap_outliers(data, 'BMI')
data = cap_outliers(data, 'BPM')
data = cap_outliers(data, 'Calories Burned')
data = cap_outliers(data, 'Session Duration (hrs)')
```

Transforming Outliers:

Transforming outliers involves applying a mathematical function (such as a logarithmic or square root transformation) to reduce the impact of extreme values. The goal is to make the distribution of data more normal (bell-shaped) and decrease the influence of outliers, often without completely removing or changing the outliers.

How It Works:

- Transformation modifies extreme values so they have less influence on the overall dataset.
- Common transformations include:
    - **Log Transformation:** Useful when data spans several orders of magnitude.
    - **Square Root Transformation:** Reduces the impact of large values.

Purpose:

Transformation is often used when the dataset is heavily skewed, or the presence of outliers violates assumptions of normality, which could impact models like linear regression or algorithms that assume normality in data.

Example:

For a column like Session Duration (hrs), if you have values like 1, 10, 100, and 1000 hours, a log transformation would reduce the weight of the extreme 1000-hour value, making it closer to the rest of the data.

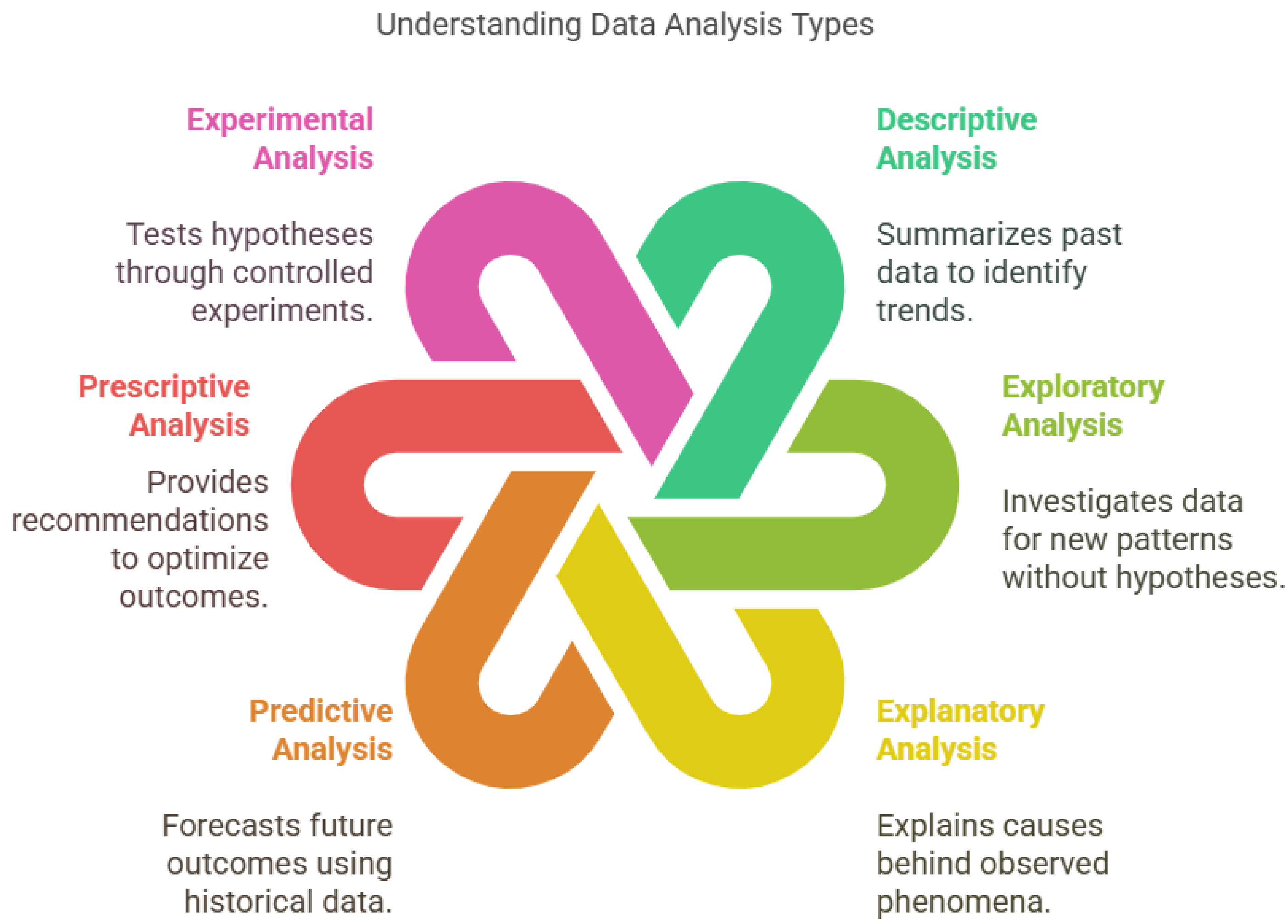
To transform outliers, use the following code:

```
# Function to log transform outliers
def transform_outliers(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

    # Apply log transformation to values greater than the upper bound
    df[column] = np.where(df[column] > upper_bound, np.log(df[column] + 1), df[column])
    # Apply log transformation to values less than the lower bound
    df[column] = np.where(df[column] < lower_bound, np.log(df[column] + 1), df[column])

    return df

# Transform outliers for each column (optional, based on the context)
data = transform_outliers(data, 'BMI')
data = transform_outliers(data, 'BPM')
data = transform_outliers(data, 'Calories Burned')
data = transform_outliers(data, 'Session Duration (hrs)')
```

Task 2: Insights (EDA)

Analysis in data refers to the process of examining, organizing, and interpreting data to extract meaningful insights and support decision-making. It involves applying various statistical, computational, and visualization techniques to identify patterns, relationships, trends, or anomalies in the data. The goal is to transform raw data into actionable information that can inform strategies, solve problems, and predict future outcomes.

Types of Analysis



- To understand **what happened** (descriptive analysis).
- To investigate **what is going on** (exploratory analysis)
- To determine **why it happened** (diagnostic or explanatory analysis).
- To predict **what will happen** (predictive analysis).
- To recommend **what should be done** (prescriptive analysis).
- To test **how effective solutions are** (experimental analysis).

Let us have a brief understanding of each type of analysis with an example:

1. Descriptive Analysis

- **Purpose**: Understand **what happened**.
- **Example**: A retail company analyses last year's sales data and finds that total revenue was ₹10 crore, with the highest sales during December.
- **Explanation**: Descriptive analysis focuses on summarizing historical data to identify trends and patterns. It's like looking at a report card to understand your performance.

2. Exploratory Analysis (EDA)

- **Purpose**: Investigate **what is going on**.
- **Example**: A data analyst visualizes customer purchase behaviour using bar charts, scatter plots, and correlations to find that customers in Tier-1 cities prefer online shopping, while rural customers prefer in-store purchases.
- **Explanation**: EDA is about exploring the data to uncover patterns, relationships, and anomalies that might not be immediately obvious. It helps guide further analysis.

3. Diagnostic Analysis (Explanatory Analysis)

- **Purpose**: Determine **why it happened**.
- **Example**: After observing a drop-in sale in February, the company finds that a key supplier had delivery delays, which affected product availability.
- **Explanation**: Diagnostic analysis digs deeper into data relationships and root causes. It's like diagnosing why a plant isn't growing—lack of water, sunlight, or nutrients?

4. Predictive Analysis

- **Purpose**: Forecast **what will happen**.
- **Example**: Using historical sales data and seasonality trends, the company predicts a 15% increase in sales for the upcoming December.
- **Explanation**: Predictive analysis uses statistical and machine learning models to make future predictions based on patterns in historical data.

5. Prescriptive Analysis

- **Purpose**: Recommend **what should be done**.
- **Example**: The company uses optimization models to decide they should offer a 20% discount on specific products and increase marketing spend in November to maximize December sales.
- **Explanation**: Prescriptive analysis provides actionable strategies based on insights, helping decision-makers take the best possible course of action.

6. Experimental Analysis

- **Purpose**: Test **how effective solutions are**.
- **Example**: The company runs an A/B test where one group of customers receives personalized email campaigns, and another doesn't. They find that the group receiving emails had a 25% higher conversion rate.
- **Explanation**: Experimental analysis involves testing different strategies to evaluate their effectiveness, often using controlled experiments.

Importance of Exploratory Data Analysis

Exploratory Data Analysis (EDA) is important because it helps you **understand your data before diving into complex analyses**. Think of it as getting to know your data like you would get to know a new friend—by exploring what's inside!

Here's why EDA is important, explained simply:

1. **It Helps You Spot Problems**
   - EDA helps you find errors, missing data, or strange values that don't make sense. For example, if your dataset has a person's age as 200, you know something is wrong!

2. **It Tells You What's in Your Data**
   - You explore the data to see patterns, like which products are selling most or what time of year sales are high. It's like looking at a map before starting a journey.
3. **It Simplifies Decision-Making**
   - By visualizing data (with charts, graphs, etc.), you can quickly understand what's going on. For instance, a graph showing monthly sales can tell you if business is growing or slowing down.
4. **It Guides Your Next Steps**
   - Once you understand the data, you'll know what to focus on, like which variables are most important or which trends need deeper analysis.
5. **It Builds a Strong Foundation**
   - If you don't explore the data first, you might miss critical insights or make wrong predictions. EDA ensures you start with a clear picture.
6. **It Helps Uncover Hidden Patterns**
   - EDA often reveals interesting or unexpected insights that might not be obvious at first glance. For example, you might discover that customers in one region buy more during weekends, or that certain product categories are frequently bought together.

Best Practices for EDA

1. **Know Your Goal**: Be clear about what you're trying to find or solve in your data.
2. **Look at the Big Picture**: Check the size of your dataset, the number of rows and columns, and understand what each column means.
3. **Clean Your Data**: Remove duplicates, fix missing values, and check for incorrect entries (like negative ages or strange dates).
4. **Summarize Your Data**: Use simple stats like mean, median, max, and min to understand the numbers.
5. **Visualize It**: Create charts and graphs (like histograms, scatter plots, or bar charts) to see trends and patterns easily.
6. **Check Relationships**: Look at how one variable affects another (e.g., does age impact spending?).
7. **Spot Outliers**: Identify weird or extreme values that might mess up your analysis.
8. **Ask Questions**: Always ask, "**Does this make sense?**" or "**Why does this happen?**" to dig deeper.
9. **Document Your Findings**: Write down what you observe so you don't forget and can explain it later.
10. **Don't Overcomplicate**: Stick to simple tools and methods unless your data needs something advanced.

Key Insight Questions

Question 1: How does the average calorie burn vary by Workout Type and Gender?

Objective

Identify patterns in calorie expenditure across workout types and genders to understand potential differences in workout efficacy.

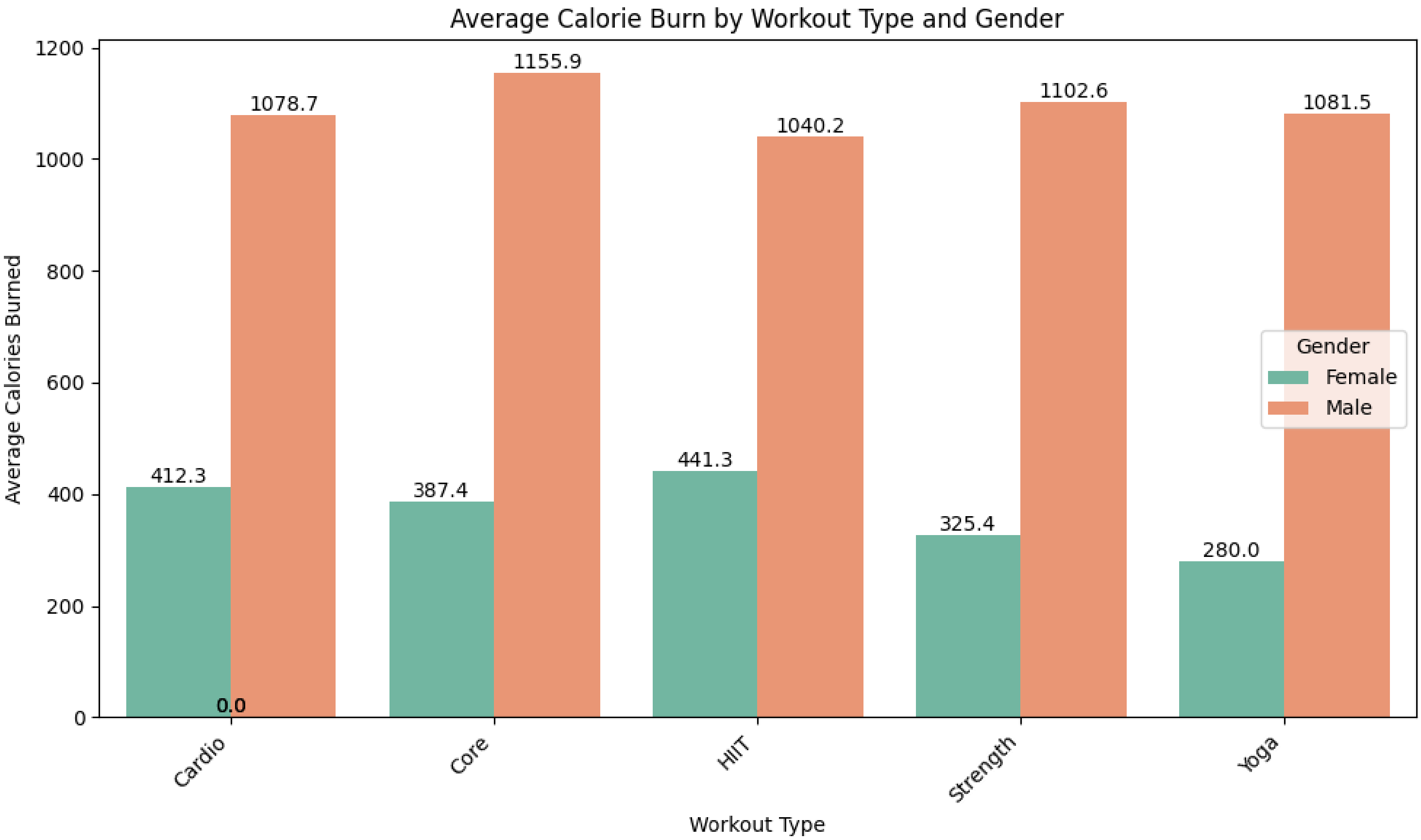
How to Solve

1. Group the data by Workout and Gender. You'll use the **groupby**() function from pandas to group the DataFrame by both **'Workout Type'** and **'Gender'** columns. This creates subgroups for each unique combination of these two categorical variables.
2. Calculates the average calorie burn for each group created in the previous step. After grouping the data, you'll apply the **mean**() function to the **'Calories Burned'** column within each group. This will give you the average calorie burn for each combination of **'Workout Type'** and **'Gender'**.
3. Visualize the results using a bar plot with gender as a hue. You'll use a bar plot to visualize the data.
    1. The x-axis will represent the **'Workout Type'**.
    2. The y-axis will represent the **'Average Calories Burned'**.
    3. The hue parameter in the **sns.barplot**() function will be set to **'Gender'**, which will create separate bars for each gender within each **'Workout Type'**.

Code

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Calculate average calorie burn by workout type and gender
average_calories = df.groupby(['Workout', 'Gender'])['Calories Burned'].mean().reset_index()

# Create a bar plot
plt.figure(figsize=(10, 6))
sns.barplot(x='Workout', y='Calories Burned', hue='Gender', data=average_calories, palette="Set2")
plt.title('Average Calorie Burn by Workout Type and Gender')
plt.xlabel('Workout Type')
plt.ylabel('Average Calories Burned')
plt.xticks(rotation=45, ha='right')  # Rotate x-axis labels for better readability

# Add data labels on top of the bars
for p in plt.gca().patches:
    plt.gca().annotate(f'{p.get_height():.1f}', (p.get_x() + p.get_width() / 2., p.get_height()),
                       ha='center', va='center', xytext=(0, 5), textcoords='offset points')

plt.tight_layout()  # Adjust layout to prevent labels from overlapping
plt.show()
```

Chart/Graph



Insights

This analysis helps in understanding which workout types are more effective for calorie burn and whether there are significant gender differences in their impact.

Question 2: What is the relationship between Age and average Calories Burned across different genders?

Objective

Understand how age impacts calorie burn and whether this varies significantly between male and female participants.

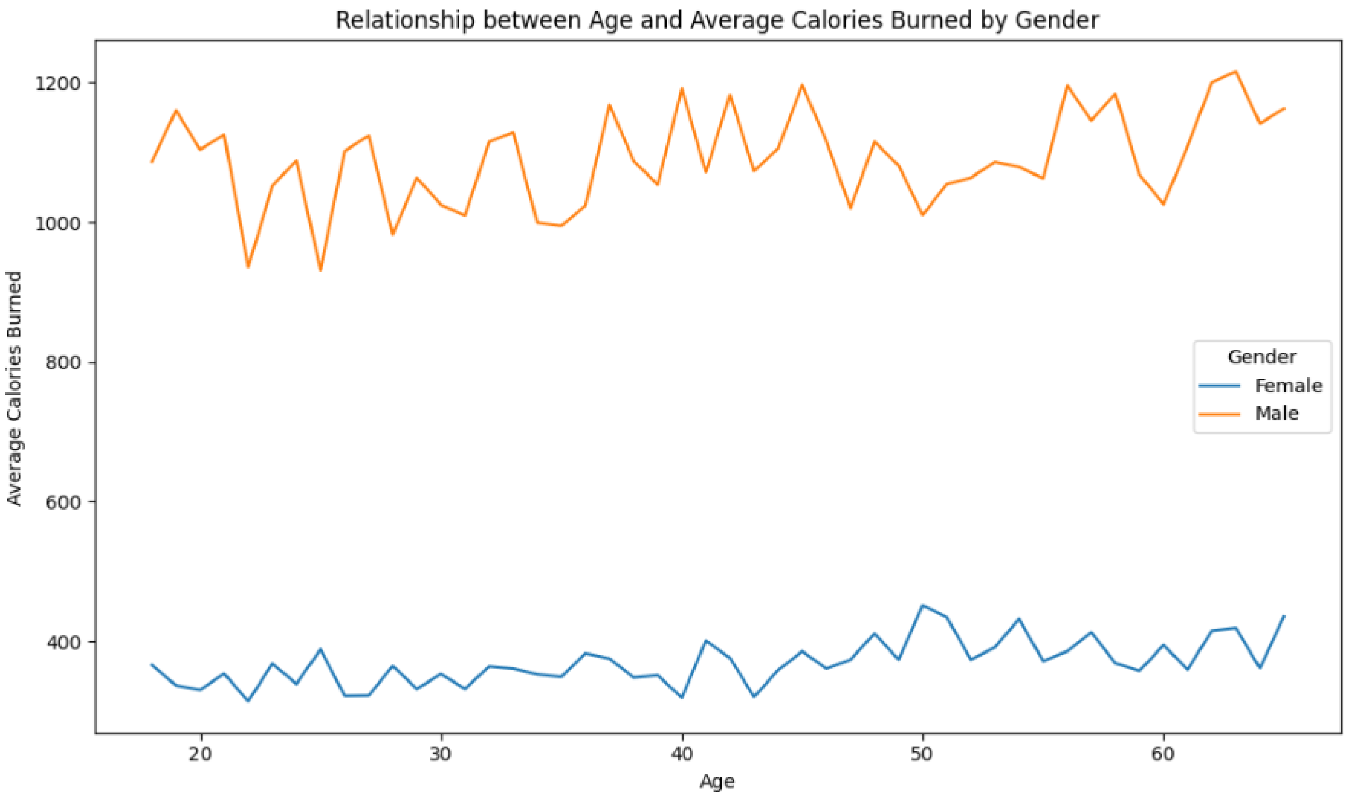
How to Solve

1. Group the data by Age and Gender. You'll use the **groupby()** function from pandas to group the DataFrame by both **'Age'** and **'Gender'** columns. This creates subgroups for each unique combination of these two categorical variables.
2. Calculate the mean of Calories Burned for each group. After grouping the data, you'll apply the **mean()** function to the **'Calories Burned'** column within each group. This will give you the average calorie burn for each combination of **'Age'** and **'Gender'**.
3. Use a line plot to visualize the relationship, with **Age** on the x-axis and **Calories Burned** on the y-axis.
    1. The x-axis will represent **'Age'**.
    2. The y-axis will represent **'Average Calories Burned'**.
    3. The hue parameter in the **sns.lineplot**() function will be set to **'Gender'**, which will create separate lines for each gender on the plot.

Code

```
# Calculate average calories burned by Age and Gender
average_calories_by_age_gender = df.groupby(['Age', 'Gender'])['Calories Burned'].mean().reset_index()

# Create a line plot
plt.figure(figsize=(10, 6))
sns.lineplot(x='Age', y='Calories Burned', hue='Gender', data=average_calories_by_age_gender)
plt.title('Relationship between Age and Average Calories Burned by Gender')
plt.xlabel('Age')
plt.ylabel('Average Calories Burned')
plt.tight_layout()
plt.show()
```

Chart/Graph



Insights

This analysis reveals age-related trends in calorie expenditure and highlights any gender-based differences.

Question 3: How does the Session Duration impact Calories Burned for participants with above-average BMI compared to those with below-average BMI?

Objective

Explore how session duration influences calorie burn, specifically for participants categorized into above-average and below-average BMI groups.

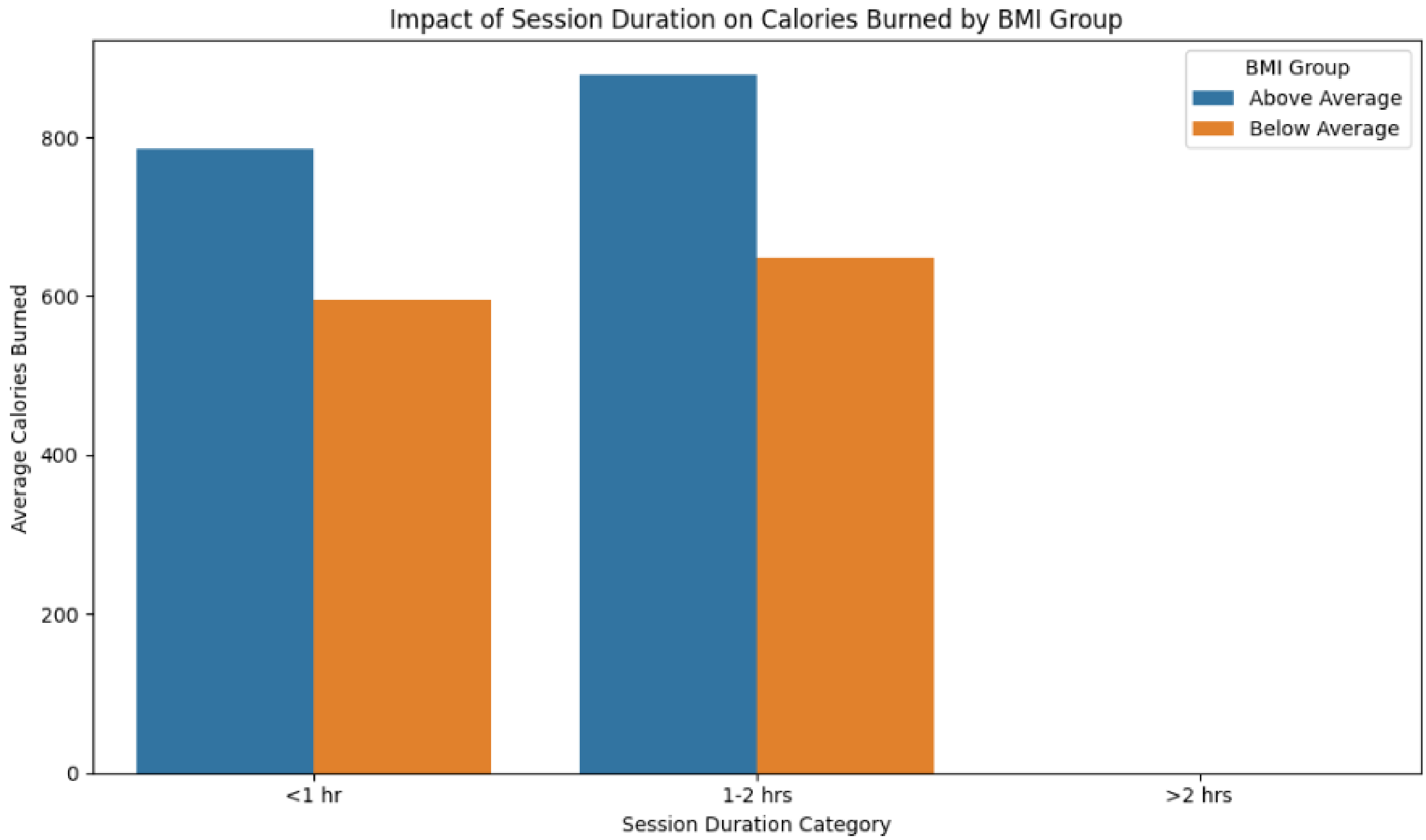
How to Solve

1. Calculate the average BMI. You'll calculate the mean of the **'BMI'** column in your DataFrame.
2. This average BMI value will be used as the threshold for categorization. You'll create a new column (e.g., **'BMI Group'**) in your DataFrame. Using a conditional statement, you'll assign **'Above Average'** to participants with BMI greater than the average BMI, and **'Below Average'** to those with BMI less than or equal to the average BMI.
3. Categorize participants into Above Average and Below Average BMI groups. You'll use the **pd.cut()** function from pandas to create bins for the **'Session Duration (hrs)'** column. Define the bin edges (e.g., [0, 1, 2, 3]) to create categories like **'<1 hr'**, **'1-2 hrs'**, and **'>2 hrs'**. Assign the resulting bin labels to a new column (e.g., **'Session Duration Category'**) in your DataFrame.
4. Bin session durations into categories (e.g., <1 hr, 1-2 hrs, >2 hrs).
5. Group data by BMI Group and Session Duration Category.
6. Visualize using a grouped bar chart.

Code

```
# Calculate the average BMI
average_bmi = df['BMI'].mean()

# Create BMI groups
df['BMI Group'] = ['Above Average' if bmi > average_bmi else 'Below Average' for bmi in df['BMI']]

# Bin session durations into categories
df['Session Duration Category'] = pd.cut(df['Session Duration (hrs)'], bins=[0, 1, 2, 3], labels=['<1 hr', '1-2 hrs', '>2 hrs'])

# Calculate average calories burned by BMI Group and Session Duration Category
calories_by_bmi_duration = df.groupby(['BMI Group', 'Session Duration Category'])['Calories Burned'].mean().reset_index()

# Create a grouped bar chart
plt.figure(figsize=(10, 6))
sns.barplot(x='Session Duration Category', y='Calories Burned', hue='BMI Group', data=calories_by_bmi_duration)
plt.title('Impact of Session Duration on Calories Burned by BMI Group')
plt.xlabel('Session Duration Category')
plt.ylabel('Average Calories Burned')
plt.tight_layout()
plt.show()
```

Chart/Graph



Insights

This analysis shows how session duration affects calorie expenditure differently for individuals with varying BMI levels.

Question 4: How does BMI correlate with Calories Burned across different Workout Types?

Objective

Examine the relationship between BMI and calorie burn for each workout type to uncover potential patterns or trends.

How to Solve

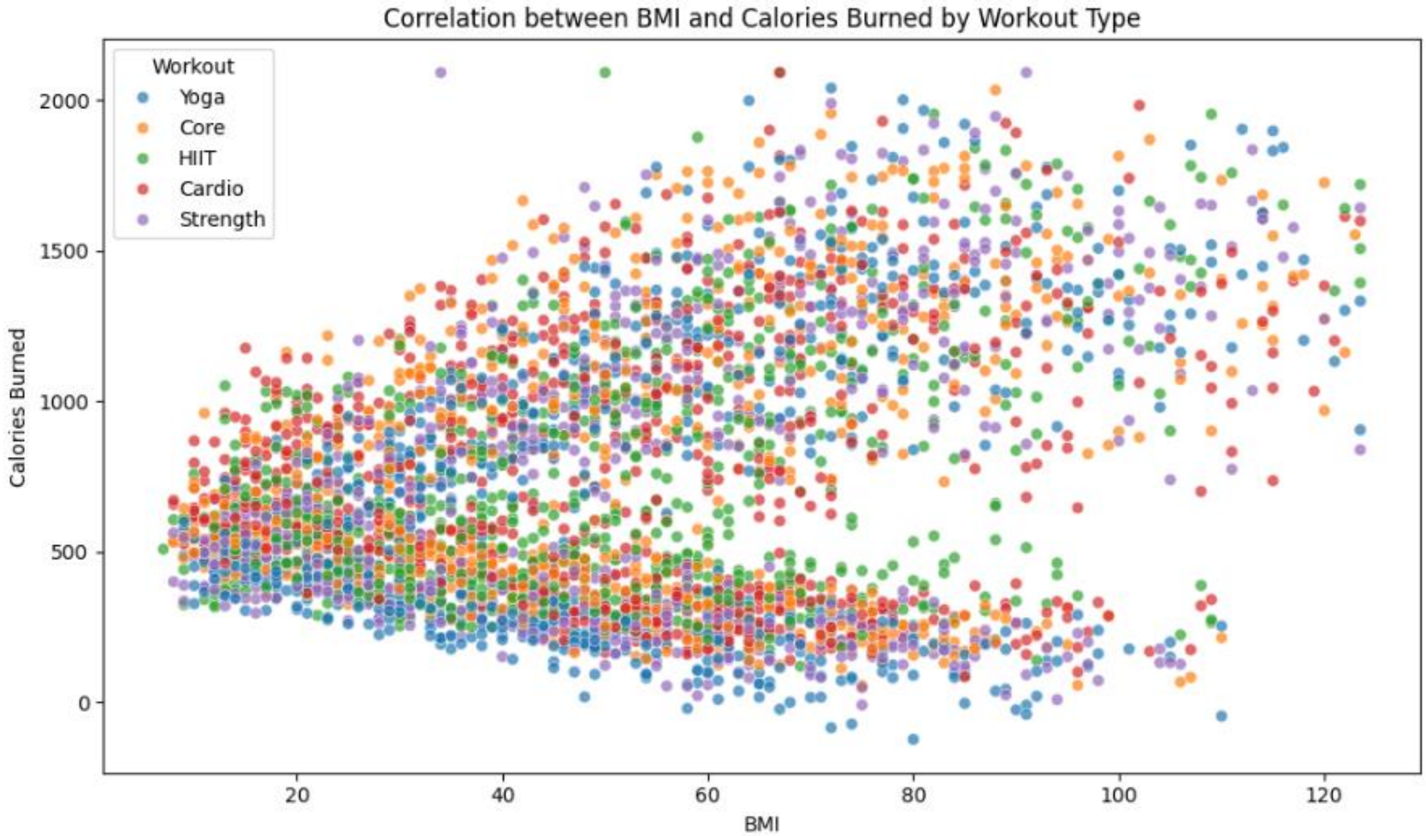
1. Create a scatter plot with BMI on the x-axis and Calories Burned on the y-axis, differentiating by Workout Type.
    1. Use a scatter plot to represent each data point as a dot.
    2. The x-axis should represent the BMI values.
    3. The y-axis should represent the Calories Burned.
    4. Use a different color or marker to represent each distinct Workout Type. This will allow us to compare the relationship between BMI and Calories Burned for different workout types.
2. Add regression lines to show trends for each workout type.
    1. Fit a regression line (e.g., linear regression) to the data points for each Workout Type.
    2. Plot these regression lines on the scatter plot.
    3. The slope of the regression line will indicate the general trend of the relationship (positive, negative, or no correlation).
3. Calculate and print the correlation coefficient for BMI and Calories Burned within each workout type.
    1. Calculate the correlation coefficient (e.g., Pearson correlation coefficient) between BMI and Calories Burned for each Workout Type.
    2. Print the correlation coefficient for each Workout Type. A value closer to 1 indicates a strong positive correlation, a value closer to -1 indicates a strong negative correlation, and a value close to 0 indicates little to no correlation.

Code

```
# Scatter plot with BMI and Calories Burned colored by Workout Type
plt.figure(figsize=(10, 6))
sns.scatterplot(x='BMI', y='Calories Burned', hue='Workout', data=df, alpha=0.7)
plt.title('Correlation between BMI and Calories Burned by Workout Type')
plt.xlabel('BMI')
plt.ylabel('Calories Burned')
plt.tight_layout()
plt.show()

# Calculate correlation for each Workout Type
for workout_type in df['Workout'].unique():
    subset = df[df['Workout'] == workout_type]
    correlation = subset['BMI'].corr(subset['Calories Burned'])
    print(f"Correlation between BMI and Calories Burned for {workout_type}: {correlation:.2f}")
```

Reflex, Cain Road, Bracknell, Berkshire RG12 1HL
T: 01344 203999 F 01344 203998
info@theknowledgeacademy.com – www.theknowledgeacademy.com

Chart/Graph




```
Correlation between BMI and Calories Burned for Yoga: 0.37
Correlation between BMI and Calories Burned for Core: 0.26
Correlation between BMI and Calories Burned for HIIT: 0.41
Correlation between BMI and Calories Burned for Cardio: 0.18
Correlation between BMI and Calories Burned for Strength: 0.33
```

Insights

This analysis identifies whether BMI is a significant factor in calorie burn and how its influence varies across workout types.

Reflex, Cain Road, Bracknell, Berkshire RG12 1HL
T: 01344 203999 F 01344 203998
info@theknowledgeacademy.com – www.theknowledgeacademy.com

Task 3: Artificial Intelligence Integration

What is AI Integration?

AI integration is the process of applying Artificial Intelligence techniques to solve real-world problems using data. This task involves building, training, and evaluating AI models to extract insights, make predictions, or automate processes.

In the context of this hackathon, AI integration builds upon the insights gained during the EDA stage to create models that generate actionable outputs.

Why is AI Integration Important?

- **Enhances Decision-Making**: AI models can process complex datasets and uncover patterns that are beyond human analysis.
- **Automation and Efficiency**: Tasks that are repetitive or computationally intensive can be automated using AI.
- **Scalability**: Once integrated, AI systems can process large-scale data efficiently and consistently.
- **Predictive Capabilities**: AI models empower organizations to anticipate trends and outcomes.

Common Challenges in AI Integration

- **Data Quality Issues**: Garbage in, garbage out – AI models are only as good as the data they're trained on.
- **Overfitting**: Models might perform well on training data but fail to generalize to new data.
- **Resource Constraints**: Training complex AI models requires computational power and time.
- **Interpretability**: Explaining how AI models make decisions can be critical in some domains.

Best Practices

- Start simple with baseline models to set a benchmark.
- Use feature importance techniques to understand which factors drive predictions.
- Regularly test the model with fresh, unseen data to ensure reliability.
- Document assumptions and findings for reproducibility.

AI Questions

By working through the following tasks, you'll gain valuable experience in both **classification** and **regression**, which are foundational in data analysis and machine learning.

Classification Task:

- Question: Can we classify individuals as 'above average' or 'below average' calorie burners based on their personal and session details?
- Objective: Learn to categorize individuals into two groups—'above average' or 'below average'—based on their calorie burn. This task involves deriving a binary target variable from the Calories Burned column and building predictive models to perform the classification.
- Steps 3a to 7b will cover this.

Regression Task:

- Question: What factors most significantly influence the number of calories burned during a workout?
- Objective: Understand how variables such as Age, Weight, Height, BMI, BPM, Workout Type, and Session Duration impact the number of calories burned. You'll develop regression models to quantify these relationships and evaluate their significance.
- Steps 8a to 12b will cover this.

Task 3.5 Chatbot Prompts

A chatbot will be available to provide insights and recommendations for solving the problems you encounter. You can interact with this chatbot to gain a better understanding of how to solve the problem. Please ensure your questions focus solely on the dataset and remain on-topic. The following is an example of how the chatbot works. Instructions on how to use the Chatbot will also be provided

Urban Mobility Chatbot

Ask questions about the urban mobility dataset and hackathon tasks.

Enter your question:

Provide me with the code to fill the Nulls into the Columns

Submit

Chat History:

YOU: Provide me with the code to fill the Nulls into the Columns

Chatbot: I can provide guidance on how to fill missing data using suitable methods. To fill null values in the dataset, you can consider using methods such as mean, median, mode imputation for numerical columns, and mode imputation for categorical columns. Another approach is to use more advanced techniques like K-Nearest Neighbors (KNN) imputation or predictive modeling to fill missing values based on other features in the dataset. Would you like more details on any specific method for filling missing values in the dataset?

YOU: Tell me about the Dataset

Chatbot: The dataset provided contains information related to urban mobility, including features such as traffic volume, passenger count, noise levels, air quality (PM2.5 Level, AQI), average speed, location, day of the week, and time of the day. It offers a comprehensive view of various factors that can impact urban transportation and environment. The categorical columns in the dataset are 'Location', 'Day of the Week', and 'Time of the Day'. The dataset allows for in-depth analysis and modeling to derive insights for improving urban mobility systems.

How to Use the Chatbot

Welcome to the Urban Mobility Chatbot! This chatbot is designed to help you analyze and understand the urban mobility dataset. Here's how you can use it:

- **Ask Questions**: Type your question in the input box below and click "Submit".
- **Dataset Insights**: The chatbot can provide insights about traffic volume, air quality, noise levels, and more.
- **Hackathon Tasks**: Use the chatbot to get guidance on tasks like data analysis, advanced analysis, and AI modeling.

Example Questions:

- What is the average traffic volume?
- Which location has the highest AQI?
- What are the high correlations in the dataset?

Steps in AI Integration

Step 1: Decide what you are going to do

The first step in AI integration is to determine the specific problem you aim to solve and select an appropriate AI model based on the nature of your data. This decision heavily influences the rest of the workflow.

Common AI Models to Consider:

1. Linear Regression
   - Use Case: Predicting a continuous variable (e.g., housing prices).
   - Strengths: Simple and interpretable.
   - Limitations: Assumes linear relationships between features and the target.
2. Ridge and Lasso Regression
   - Use Case: Predicting a continuous variable while handling multicollinearity or feature selection.
   - Strengths: Regularization helps prevent overfitting.
   - Limitations: May still struggle with non-linear patterns in data.
3. Logistic Regression
   - Use Case: Predicting binary outcomes (e.g., churn prediction).
   - Strengths: Works well for classification problems with clear decision boundaries.
   - Limitations: Assumes a linear relationship between features and the log-odds of the outcome.
4. K-Nearest Neighbours (KNN)
   - Use Case: Classification or regression tasks where simplicity and adaptability are needed.
   - Strengths: Intuitive and non-parametric (no assumptions about data distribution).
   - Limitations: Computationally expensive for large datasets and sensitive to irrelevant features.

Reflex, Cain Road, Bracknell, Berkshire RG12 1HL
T: 01344 203999 F 01344 203998
info@theknowledgeacademy.com – www.theknowledgeacademy.com

Step 2: Import the required Libraries

```
# Common Imports
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split

# Preprocessing Tools
from sklearn.preprocessing import LabelEncoder, StandardScaler

# Regression Models
from sklearn.linear_model import LinearRegression, Ridge, Lasso

# Classification Models
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsRegressor

# Regression Metrics
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# Classification Metrics
from sklearn.metrics import accuracy_score, classification_report, precision_score, recall_score
```

To implement AI models, specific libraries must be imported depending on the task. The imports can be categorized into the following groups:

1. Common Imports

These are essential libraries that are required regardless of the AI model or task you're implementing.

- **pandas**: For data manipulation and handling datasets.
- **numpy**: For numerical computations and operations.
- **seaborn**: For advanced data visualization (works well alongside Matplotlib).
- **train_test_split**: For splitting the data into training and testing sets.

2. Preprocessing Tools

If additional preprocessing is required (e.g., encoding categorical variables or scaling numerical features), you will need to import the appropriate tools:

- **Label Encoder**: To convert categorical string values into numerical values.
- **Standard Scaler**: For normalizing or standardizing numerical features to improve model performance.

3. Model-Specific Imports

The choice of model depends on the nature of the problem you're addressing, specifically whether it's a classification or regression task:

- **For Regression Tasks** (predicting continuous numerical values):
  - **Linear Regression**: For basic linear regression.
  - **Ridge** and **Lasso**: For regularized linear regression models to prevent overfitting.
- **For Classification Tasks** (predicting categorical outcomes):
  - **Logistic Regression**: For binary classification problems (e.g., predicting whether a customer will churn).
  - **K Nearest Neighbours Classifier**: For classifying data based on nearest neighbours.

4. Model Evaluation Metrics

Once a model is trained, it's important to assess its performance using relevant evaluation metrics:

- **For Regression Models**:
  - **mean_absolute_error**: Measures the average magnitude of errors in predictions.
  - **mean_squared_error**: Measures the average squared differences between predicted and actual values.
  - **r2_score**: Indicates how well the model's predictions match the actual outcomes.
- **For Classification Models**:
  - **accuracy_score**: Measures the proportion of correct predictions.
  - **classification_report**: Provides a comprehensive performance report including precision, recall, and F1-score.
  - **precision_score**: Measures the proportion of true positive results in all predicted positives.
  - **recall_score**: Measures the proportion of true positive results in all actual positives.

Step 3a: Setup a Scaler and Encoder (LogisticRegression)

Scalers and **encoders** are essential tools in data preprocessing for machine learning. Scalers standardize or normalize numerical data to ensure that features are on a similar scale, which is particularly important for algorithms like KNN, SVM, and linear models that are sensitive to feature magnitudes. Common scalers include **StandardScaler** (scales data to have a mean of 0 and a standard deviation of 1) and **MinMaxScaler** (scales data to a fixed range, typically 0 to 1).

On the other hand, **encoders** convert categorical data into numerical form, as machine learning models require numeric input. Techniques like **Label Encoding** assign unique numbers to each category, while **One-Hot Encoding** creates binary columns for each category, preserving more information about the data. Together, scalers and encoders prepare raw data for efficient and accurate model training. Some models require **Scalers** and **Encoders**, some require only **Encoders**, while others require only **Scaler**.

Note: For the AI task, we copied the original cleaned data into a new DataFrame called data2.

For this specific task, our target variable is the **Calorie_Burner** column, which represents whether an individual's calorie burn is above or below average. We derive this binary target from the Calories Burned column by comparing each individual's calorie burn against the average calorie burn in the dataset. We then create a new column, **Calorie_Burner**, where a value of 1 indicates above-average calorie burn and 0 represents below-average calorie burn

```
average_calories = data2['Calories Burned'].mean()
data2['Calorie_Burner'] = (data2['Calories Burned'] > average_calories).astype(int)
```

```
scaler = StandardScaler()

# Apply the scaler to relevant columns, e.g., BMI, session duration
data2[['BMI', 'Session Duration (hrs)']] = scaler.fit_transform(data[['BMI', 'Session Duration (hrs)']])
```

```
label_encoder = LabelEncoder()
data2['Gender'] = label_encoder.fit_transform(data['Gender'])
data2['Workout'] = label_encoder.fit_transform(data['Workout'])
```

```
print("Encoded Data:")
print(data2[['Gender', 'Workout']])
```

```
Encoded Data:
      Gender  Workout
0          1        4
1          0        4
2          0        1
3          0        1
4          1        2
...      ...      ...
3905       0        2
3906       1        4
3907       0        0
3908       0        1
3909       1        4
```

Step 4a: Selecting Target Variable

In this step, we define the features (independent variables) and the target (dependent variable) for our model. We drop the Calories Burned and Calorie_Burner columns for the following reasons:

- **Calories Burned:** This column contains the continuous values that represent the total calories burned by an individual. Since we are classifying individuals based on whether their calorie burn is above or below average, the raw calorie values are not needed for model training. The model will rely on other features to determine whether the individual belongs to the 'above average' or 'below average' group.
- **Calorie_Burner:** This is the target variable that we want the model to predict. In a supervised learning model, the target variable is separate from the features. Since we are training the model to classify individuals into either above or below average calorie burners, Calorie_Burner serves as the outcome we are predicting, not a feature that contributes to the prediction.

```
X = data2.drop(columns=['Calories Burned', 'Calorie_Burner'])  # Features
y = data2['Calorie_Burner']  # Target variable
```

Step 5a: Splitting the Data into Test and Training Sets

Before training a machine learning model, it's important to divide the data into two parts: a **training set** and a **testing set**. The **training set** is used to teach the model by allowing it to learn patterns in the data, while the **testing set** is reserved to evaluate how well the model generalizes to unseen data. A common practice is to allocate 80% of the data for training and 20% for testing. This ensures the model has enough data to learn while also having a separate dataset to validate its performance.

For **Logistic Regression**, which is used for binary classification tasks, we follow this split to ensure the model has a balanced dataset for both training and evaluation. Below is the code to achieve this using `train_test_split` from scikit-learn:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=42)
```

Note: In machine learning, the random_state parameter is used to ensure reproducibility when performing operations that involve randomness, such as splitting data into training and testing sets or initializing the weights in a model.

When we split data into training and testing sets using train_test_split, the function randomly selects samples for the test set, which could result in slightly different splits each time the code is run. To make sure that we get the same result each time we run the code (even after restarting the kernel or running it on a different machine), we use the random_state parameter. By setting it to a specific integer value (e.g., random_state=42), you ensure that the random process is controlled, and the same split occurs every time.

Step 6a: Set-up the Model and Train it (LogisticRegression)

We initialize the Logistic Regression model using the LogisticRegression() function from scikit-learn. Logistic regression is a popular machine learning algorithm used for binary classification tasks, where the goal is to classify data into one of two possible outcomes. In our case, the model will classify individuals as either 'above average' or 'below average' calorie burners. After initializing the model, we train it using the fit() method, passing in the training data (X_train for the features and y_train for the target variable). This step involves the model learning from the data, identifying patterns, and adjusting its internal parameters to best predict the target variable (in this case, the Calorie_Burner). The result is a trained model that can then be evaluated on unseen data to assess its performance.

```
logreg = LogisticRegression()
logreg.fit(X_train, y_train)
```

Step 7a: Predictions and Model Evaluation

Once the Logistic Regression model has been trained, the next step is to evaluate its performance on the test data. This involves making predictions and comparing them to the actual outcomes. Evaluation metrics help us understand how well the model is performing and highlight areas for improvement. Below are the key metrics used in this step:

- *Confusion Matrix:*
  - The confusion matrix provides a breakdown of the model's predictions, showing the number of true positives, true negatives, false positives, and false negatives. This helps us understand the model's classification performance in detail.
- *Accuracy:*
  - Accuracy measures the proportion of correct predictions (both true positives and true negatives) out of the total predictions. It gives a general sense of how often the model predicts correctly.
- *Precision:*
  - Precision focuses on the positive class and indicates the proportion of true positive predictions out of all predicted positives. This metric is crucial when minimizing false positives is important.
- *Recall:*
  - Recall (or Sensitivity) measures the proportion of actual positive cases that the model correctly identifies. It is important in scenarios where missing a positive case (false negative) is costly.
- *F1-Score:*
  - The F1-Score is the harmonic mean of precision and recall. It balances these two metrics, providing a single measure of a model's classification performance, especially when dealing with imbalanced datasets.
- *ROC-AUC Score:*
  - The ROC-AUC Score measures the model's ability to distinguish between the positive and negative classes across different thresholds. A higher score indicates better discriminatory power.

```
y_pred = logreg.predict(X_test)
```

```
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

# Accuracy
print("\nAccuracy:", accuracy_score(y_test, y_pred))

# Precision
print("\nPrecision:", precision_score(y_test, y_pred))

# Recall
print("\nRecall:", recall_score(y_test, y_pred))

# F1-Score
print("\nF1-Score:", f1_score(y_test, y_pred))

# ROC-AUC Score
print("\nROC-AUC Score:", roc_auc_score(y_test, y_pred))
```

```
Confusion Matrix:
[[854  50]
 [ 33 663]]

Accuracy: 0.948125

Precision: 0.929873772791023 8

Recall: 0.9525862068965517

F1-Score: 0.9410929737402413

ROC-AUC Score: 0.9486382361916386
```

Step 3b: Setup a Scaler and Encoder (KNeighborsClassifier)

For K-Nearest Neighbors (KNN), scaling the features is essential to ensure that all numerical columns contribute equally to the distance calculations. Unlike Logistic Regression, KNN relies on the concept of proximity, and features with larger ranges can disproportionately influence the model if left unscaled.

In this step, we normalize the numerical features using StandardScaler, which standardizes the data by removing the mean and scaling it to unit variance. Additionally, categorical variables are encoded using LabelEncoder to transform them into numeric format, as KNN requires numerical input for all features. Unlike Logistic Regression, the Scaler is employed afte Step 4 and before Step 5.

```
[42] label_encoder = LabelEncoder()
     data2['Gender'] = label_encoder.fit_transform(data['Gender'])
     data2['Workout'] = label_encoder.fit_transform(data['Workout'])
```

```
[43] print("Encoded Data:")
     print(data2[['Gender', 'Workout']])
```

```
Encoded Data:
      Gender  Workout
0          1        4
1          0        4
2          0        1
3          0        1
4          1        2
...      ...      ...
3995       1        4
3996       0        0
3997       0        1
3998       0        2
3999       1        4

[4000 rows x 2 columns]
```

```
[44] average_calories = data2['Calories Burned'].mean()
     data2['Calorie_Burner'] = (data2['Calories Burned'] > average_calories).astype(int)
```

```
[45] X = data2.drop(columns=['Calories Burned', 'Calorie_Burner'])  # Features
     y = data2['Calorie_Burner']  # Target variable
```

```
[46] scaler = StandardScaler()
     X_scaled = scaler.fit_transform(X)
```

```
[47] X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=42)
```

Step 4b: Selecting Target Variable

This step remains the same as step 4a. We define the features (independent variables) and the target (dependent variable) for our model. We drop the Calories Burned and Calorie_Burner columns for the following reasons:

- **Calories Burned:** This column contains the continuous values that represent the total calories burned by an individual. Since we are classifying individuals based on whether their calorie burn is above or below average, the raw calorie values are not needed for model training. The model will rely on other features to determine whether the individual belongs to the 'above average' or 'below average' group.
- **Calorie_Burner:** This is the target variable that we want the model to predict. In a supervised learning model, the target variable is separate from the features. Since we are training the model to classify individuals into either above or below average calorie burners, Calorie_Burner serves as the outcome we are predicting, not a feature that contributes to the prediction.

```
[45] X = data2.drop(columns=['Calories Burned', 'Calorie_Burner'])  # Features
     y = data2['Calorie_Burner']  # Target variable
```

Step 5b: Splitting the dataset

This step also remains the same as step 5a. Before training a machine learning model, it's important to divide the data into two parts: a **training set** and a **testing set**. The **training set** is used to teach the model by allowing it to learn patterns in the data, while the **testing set** is reserved to evaluate how well the model generalizes to unseen data. A common practice is to allocate 80% of the data for training and 20% for testing. This ensures the model has enough data to learn while also having a separate dataset to validate its performance.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=42)
```

Reflex, Cain Road, Bracknell, Berkshire RG12 1HL
T: 01344 203999 F 01344 203998
info@theknowledgeacademy.com – www.theknowledgeacademy.com

Step 6b: Set-up the Model and Train it (K Neighbors Classifier)

We initialize the K-Nearest Neighbors (KNN) model using the KNeighborsClassifier() function from scikit-learn. KNN is a versatile machine learning algorithm used for classification tasks, where predictions are made based on the proximity of data points. Unlike Logistic Regression, which uses a mathematical function to model the relationship between features and the target, KNN classifies data points by looking at the labels of their nearest neighbors.

In our case, the model will classify individuals as either 'above average' or 'below average' calorie burners. After initializing the model with a specified value for k (the number of nearest neighbors to consider), we train it using the fit() method, passing in the training data (X_train for the features and y_train for the target variable).

```
knn = KNeighborsClassifier(n_neighbors=7)
```

```
knn.fit(X_train, y_train)
```

```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=7)
```

```
y_pred = knn.predict(X_test)
```

Step 7b: Predictions and Model Evaluation

This step is the same as 7a. Once the K Neighbours Classification model has been trained, the next step is to evaluate its performance on the test data. This involves making predictions and comparing them to the actual outcomes. Evaluation metrics help us understand how well the model is performing and highlight areas for improvement. Below are the key metrics used in this step:

- *Confusion Matrix:*
  - The confusion matrix provides a breakdown of the model's predictions, showing the number of true positives, true negatives, false positives, and false negatives. This helps us understand the model's classification performance in detail.
- *Accuracy:*
  - Accuracy measures the proportion of correct predictions (both true positives and true negatives) out of the total predictions. It gives a general sense of how often the model predicts correctly.
- *Precision:*
  - Precision focuses on the positive class and indicates the proportion of true positive predictions out of all predicted positives. This metric is crucial when minimizing false positives is important.
- *Recall:*
  - Recall (or Sensitivity) measures the proportion of actual positive cases that the model correctly identifies. It is important in scenarios where missing a positive case (false negative) is costly.
- *F1-Score:*
  - The F1-Score is the harmonic mean of precision and recall. It balances these two metrics, providing a single measure of a model's classification performance, especially when dealing with imbalanced datasets.
- *ROC-AUC Score:*
  - The ROC-AUC Score measures the model's ability to distinguish between the positive and negative classes across different thresholds. A higher score indicates better discriminatory power.

```
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("\nAccuracy:", accuracy_score(y_test, y_pred))
print("\nPrecision:", precision_score(y_test, y_pred))
print("\nRecall:", recall_score(y_test, y_pred))
print("\nF1-Score:", f1_score(y_test, y_pred))
print("\nROC-AUC Score:", roc_auc_score(y_test, y_pred))
```

```
Confusion Matrix:
[[690 214]
 [339 357]]

Accuracy: 0.654375

Precision: 0.6252189141856392

Recall: 0.5129310344827587

F1-Score: 0.56353591160221

ROC-AUC Score: 0.6381026853829723
```

Step 8a: Setup a Scaler and Encoder (LinearRegression)

For linear regression, we first scale relevant numerical features to ensure they are on the same scale, especially because linear models are sensitive to the magnitude of the features. In this case, we apply the `StandardScaler` to the `BMI` and `Session Duration (hrs)` columns. This step ensures that the model performs optimally by avoiding issues where one feature may disproportionately influence the model due to its scale.

```
scaler = StandardScaler()
data2[['BMI', 'Session Duration (hrs)']] = scaler.fit_transform(data2[['BMI', 'Session Duration (hrs)']])
```

```
label_encoder = LabelEncoder()

# Encode 'Workout' and 'Gender' separately
data2['Workout'] = label_encoder.fit_transform(data2['Workout'])
data2['Gender'] = label_encoder.fit_transform(data2['Gender'])
```

Step 9a: Selecting Target Variable

This step involves separating the features (independent variables) from the target variable (dependent variable). The features are the attributes that the model will use to make predictions, while the target variable is the value that we want to predict.

In this case, the features are all the columns except for 'Calories Burned', which is the target variable. We assign the target variable, 'Calories Burned', to y, while the remaining columns are assigned to X as the features to predict the calorie burn.

```
X = data2.drop(columns=['Calories Burned'])  # Features
y = data2['Calories Burned']  # Target variable
```

Step 10a: Splitting the Data into Test and Training Sets

This step follows the same procedure as before, where we split the dataset into training and testing subsets. The training set is used to train the model, while the testing set is reserved to evaluate its performance on unseen data.

We use the train_test_split function from scikit-learn to randomly divide the data into an 80% training set and a 20% testing set. This ensures that the model has enough data to learn from while leaving sufficient data to validate its accuracy and reliability. The random_state parameter ensures reproducibility by controlling the random splitting process.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Step 11a: Set-up the Model and Train it (LinearRegression)

Before we can setup the model and train it, we must convert all column names to strings. This step ensures that all column names in the training data (`X_train`) are of string type. Some machine learning models and functions in scikit-learn require column names to be strings for proper feature handling and error-free operation. If the column names are not strings (e.g., integers or tuples), the code may raise errors during model training or evaluation.

```
# Ensure column names are strings
X_train.columns = X_train.columns.astype(str)
```

we initialize a Linear Regression model using the LinearRegression() function from scikit-learn. Linear regression is a fundamental machine learning algorithm used for predicting continuous numerical values. It establishes a linear relationship between the features (independent variables) and the target variable (dependent variable).

The model is trained using the fit() method, where X_train (features) and y_train (target values) are provided as inputs. During training, the model learns the relationship between the features and the target by adjusting its coefficients to minimize the error.

```
regressor = LinearRegression()  # Initialize Linear Regression model
regressor.fit(X_train, y_train)
```

```
▾ LinearRegression
LinearRegression()
```

Step 12a: Predictions and Model Evaluation

Once the model is trained, we use the predict() method to make predictions on the test data (X_test). These predictions (y_pred) represent the model's estimated values for the target variable, which will later be compared with the actual values (y_test) to evaluate the model's performance.

```
y_pred = regressor.predict(X_test)
```

```
print("Mean Absolute Error (MAE):", mean_absolute_error(y_test, y_pred))
print("Mean Squared Error (MSE):", mean_squared_error(y_test, y_pred))
print("R-squared Score:", r2_score(y_test, y_pred))
```

```
Mean Absolute Error (MAE): 166.8832737794195
Mean Squared Error (MSE): 41593.37729671587
R-squared Score: 0.7899822775648168
```

After making predictions, it's crucial to evaluate how well the model performs. Below are the metrics used and what they signify:

- *Mean Absolute Error (MAE):*
  - MAE measures the average magnitude of errors between the predicted and actual values. It provides a straightforward interpretation of how far, on average, the predictions are from the true values. A lower MAE indicates that the model's predictions are closer to the actual data.

- *Mean Squared Error (MSE):*
  - MSE calculates the average of the squared differences between predicted and actual values. By squaring the errors, MSE gives more weight to larger errors, making it particularly sensitive to outliers. A smaller MSE reflects better model performance and closer alignment between predictions and actual outcomes.

- *R-squared Score (R^2):*
  - The R-squared score represents the proportion of variance in the target variable that can be explained by the model. It ranges from 0 to 1, with higher values indicating better model performance. An R^2 value of 1

means the model perfectly explains the variance, while a value of 0 indicates that the model performs no better than predicting the mean of the target variable.

Each of these metrics provides valuable insights into the accuracy and reliability of the model. Together, they help assess whether the model is fit for its intended purpose.

Step 8b: Setup a Scaler and Encoder (Ridge Regression)

This step is the same as before. For Ridge regression, we first scale relevant numerical features to ensure they are on the same scale, especially because linear models are sensitive to the magnitude of the features. In this case, we apply the `StandardScaler` to the `BMI` and `Session Duration (hrs)` columns. This step ensures that the model performs optimally by avoiding issues where one feature may disproportionately influence the model due to its scale.

```
scaler = StandardScaler()
data2[['BMI', 'Session Duration (hrs)']] = scaler.fit_transform(data2[['BMI', 'Session Duration (hrs)']])
```

```
label_encoder = LabelEncoder()

# Encode 'Workout' and 'Gender' separately
data2['Workout'] = label_encoder.fit_transform(data2['Workout'])
data2['Gender'] = label_encoder.fit_transform(data2['Gender'])
```

Step 9b: Selecting Target Variable

This step involves separating the features (independent variables) from the target variable (dependent variable). The features are the attributes that the model will use to make predictions, while the target variable is the value that we want to predict.

In this case, the features are all the columns except for 'Calories Burned', which is the target variable. We assign the target variable, 'Calories Burned', to y, while the remaining columns are assigned to X as the features to predict the calorie burn.

```
X = data2.drop(columns=['Calories Burned'])  # Features
y = data2['Calories Burned']  # Target variable
```

Step 10b: Splitting the Data into Test and Training Sets

This step follows the same procedure as before, where we split the dataset into training and testing subsets. The training set is used to train the model, while the testing set is reserved to evaluate its performance on unseen data.

We use the train_test_split function from scikit-learn to randomly divide the data into an 80% training set and a 20% testing set. This ensures that the model has enough data to learn from while leaving sufficient data to validate its accuracy and reliability. The random_state parameter ensures reproducibility by controlling the random splitting process.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Step 11b: Set-up the Model and Train it (RidgeRegression)

Before we can setup the model and train it, we must convert all column names to strings. This step ensures that all column names in the training data (`X_train`) are of string type. Some machine learning models and functions in scikit-learn require column names to be strings for proper feature handling and error-free operation. If the column names are not strings (e.g., integers or tuples), the code may raise errors during model training or evaluation.

```
# Ensure column names are strings
X_train.columns = X_train.columns.astype(str)
```

we initialize a Ridge Regression model using the Ridge() function from scikit-learn. In Ridge regression, the model aims to minimize both the residual sum of squares (like in linear regression) and the sum of the squares of the coefficients, scaled by a regularization parameter (λ). This regularization parameter controls the strength of the penalty and can be tuned to achieve the best balance between bias and variance. It can range from 0.01 to 1000. Run different iterations to find the highest accuracy.

The primary advantage of Ridge regression is that it can handle multicollinearity (when predictor variables are highly correlated) and avoid overfitting by shrinking the coefficients of less important features. This makes Ridge regression a valuable tool when working with datasets that contain many features or when dealing with multicollinearity issues. The model is trained using the fit() method, where X_train (features) and y_train (target values) are provided as inputs. During training, the model learns the relationship between the features and the target by adjusting its coefficients to minimize the error.

```
regressor = Ridge(alpha=1)
regressor.fit(X_train, y_train)
```

```
▾ Ridge
Ridge(alpha=1)
```

Step 12b: Predictions and Model Evaluation

This step is the same as before. Once the model is trained, we use the predict() method to make predictions on the test data (X_test). These predictions (y_pred) represent the model's estimated values for the target variable, which will later be compared with the actual values (y_test) to evaluate the model's performance.

```
y_pred = regressor.predict(X_test)
```

```
print("Mean Absolute Error (MAE):", mean_absolute_error(y_test, y_pred))
print("Mean Squared Error (MSE):", mean_squared_error(y_test, y_pred))
print("R-squared Score:", r2_score(y_test, y_pred))
```

```
Mean Absolute Error (MAE): 166.92966121165782
Mean Squared Error (MSE): 41599.68344676643
R-squared Score: 0.7899504358785429
```

After making predictions, it's crucial to evaluate how well the model performs. Below are the metrics used and what they signify:

- *Mean Absolute Error (MAE):*
  - MAE measures the average magnitude of errors between the predicted and actual values. It provides a straightforward interpretation of how far, on average, the predictions are from the true values. A lower MAE indicates that the model's predictions are closer to the actual data.

- *Mean Squared Error (MSE):*
  - MSE calculates the average of the squared differences between predicted and actual values. By squaring the errors, MSE gives more weight to larger errors, making it particularly sensitive to outliers. A smaller MSE reflects better model performance and closer alignment between predictions and actual outcomes.

- *R-squared Score (R^2):*
  - The R-squared score represents the proportion of variance in the target variable that can be explained by the model. It ranges from 0 to 1, with higher values indicating better model performance. An R^2 value of 1 means the model perfectly explains the variance, while a value of 0 indicates that the model performs no better than predicting the mean of the target variable.

Each of these metrics provides valuable insights into the accuracy and reliability of the model. Together, they help assess whether the model is fit for its intended purpose.

Step 8c: Setup a Scaler and Encoder (Lasso Regression)

This step is the same as before. For Lasso Regression, we scale the relevant numerical features to ensure that they are on the same scale. This is important because Lasso, which uses L1 regularization (penalty based on the absolute values of the coefficients), is sensitive to the magnitude of the features. Features with larger values could dominate the regularization term, leading to suboptimal results.

Reflex, Cain Road, Bracknell, Berkshire RG12 1HL
T: 01344 203999 F 01344 203998
info@theknowledgeacademy.com – www.theknowledgeacademy.com

In this case, we apply the StandardScaler to the BMI and Session Duration (hrs) columns. This ensures that both features have a mean of 0 and a standard deviation of 1. By scaling the features, we allow the Lasso model to apply the regularization evenly across all features, which helps with feature selection and prevents any one feature from disproportionately influencing the model. It ensures that the penalty term shrinks less important coefficients to zero, effectively removing irrelevant features while retaining the most important ones.

```
scaler = StandardScaler()
data2[['BMI', 'Session Duration (hrs)']] = scaler.fit_transform(data2[['BMI', 'Session Duration (hrs)']])
```

```
label_encoder = LabelEncoder()

# Encode 'Workout' and 'Gender' separately
data2['Workout'] = label_encoder.fit_transform(data2['Workout'])
data2['Gender'] = label_encoder.fit_transform(data2['Gender'])
```

Step 9c: Selecting Target Variable

This is the same step as before. This step involves separating the features (independent variables) from the target variable (dependent variable). The features are the attributes that the model will use to make predictions, while the target variable is the value that we want to predict.

In this case, the features are all the columns except for 'Calories Burned', which is the target variable. We assign the target variable, 'Calories Burned', to y, while the remaining columns are assigned to X as the features to predict the calorie burn.

```
X = data2.drop(columns=['Calories Burned'])  # Features
y = data2['Calories Burned']  # Target variable
```

Step 10c: Splitting the Data into Test and Training Sets

This step follows the same procedure as before, where we split the dataset into training and testing subsets. The training set is used to train the model, while the testing set is reserved to evaluate its performance on unseen data.

We use the train_test_split function from scikit-learn to randomly divide the data into an 80% training set and a 20% testing set. This ensures that the model has enough data to learn from while leaving sufficient data to validate its accuracy and reliability. The random_state parameter ensures reproducibility by controlling the random splitting process.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Step 11c: Set-up the Model and Train it (LassoRegression)

Before we can setup the model and train it, we must convert all column names to strings. This step ensures that all column names in the training data (`X_train`) are of string type. Some machine learning models and functions in scikit-learn require column names to be strings for proper feature handling and error-free operation. If the column names are not strings (e.g., integers or tuples), the code may raise errors during model training or evaluation.

```
# Ensure column names are strings
X_train.columns = X_train.columns.astype(str)
```

We initialize a Lasso Regression model using the Lasso() function from scikit-learn. In Lasso regression, the model aims to minimize both the residual sum of squares (like in linear regression) and the sum of the absolute values of the coefficients, scaled by a regularization parameter (λ). This regularization parameter controls the strength of the penalty, and it can be tuned to find the optimal balance between bias and variance. Lasso regression can be particularly useful when dealing with datasets that have many features, as it encourages sparsity by shrinking the coefficients of less important features to zero, effectively removing them from the model. This feature selection process helps with both reducing the complexity of the model and improving generalization.

The regularization parameter (λ) can range from 0.01 to 1000, and different values can be tested to find the best one for the dataset, typically by evaluating the model's performance through cross-validation. A higher λ increases the regularization strength, resulting in more coefficients being shrunk to zero, while a lower λ reduces regularization and allows the model to fit more closely to the training data.

The model is trained using the fit() method, where the training data (X_train for the features and y_train for the target variable) are passed in as inputs. During training, the model learns the relationship between the features and the target variable by adjusting its coefficients to minimize the error, while simultaneously shrinking the coefficients of less important features to zero. This process helps the model focus on the most relevant variables, improving both its accuracy and interpretability.

```
regressor = Lasso(alpha=0.1)
regressor.fit(X_train, y_train)
```

```
▾ Lasso
Lasso(alpha=0.1)
```

Step 12c: Predictions and Model Evaluation

This step is the same as before. Once the model is trained, we use the predict() method to make predictions on the test data (X_test). These predictions (y_pred) represent the model's estimated values for the target variable, which will later be compared with the actual values (y_test) to evaluate the model's performance.

```
y_pred = regressor.predict(X_test)
```

```
print("Mean Absolute Error (MAE):", mean_absolute_error(y_test, y_pred))
print("Mean Squared Error (MSE):", mean_squared_error(y_test, y_pred))
print("R-squared Score:", r2_score(y_test, y_pred))
```

```
Mean Absolute Error (MAE): 168.14721159857137
Mean Squared Error (MSE): 42026.10571008291
R-squared Score: 0.7877972990486453
```

After making predictions, it’s crucial to evaluate how well the model performs. Below are the metrics used and what they signify:

- *Mean Absolute Error (MAE):*
    - MAE measures the average magnitude of errors between the predicted and actual values. It provides a straightforward interpretation of how far, on average, the predictions are from the true values. A lower MAE indicates that the model's predictions are closer to the actual data.

- *Mean Squared Error (MSE):*
    - MSE calculates the average of the squared differences between predicted and actual values. By squaring the errors, MSE gives more weight to larger errors, making it particularly sensitive to outliers. A smaller MSE reflects better model performance and closer alignment between predictions and actual outcomes.

- *R-squared Score (R^2):*
    - The R-squared score represents the proportion of variance in the target variable that can be explained by the model. It ranges from 0 to 1, with higher values indicating better model performance. An R^2 value of 1 means the model perfectly explains the variance, while a value of 0 indicates that the model performs no better than predicting the mean of the target variable.

Each of these metrics provides valuable insights into the accuracy and reliability of the model. Together, they help assess whether the model is fit for its intended purpose.

Task 4: Report Creation

Report creation is a critical component of any hackathon because it serves as a comprehensive summary of the team's journey, insights, and outcomes. While the technical solutions or models built during the event are essential, a well-crafted report translates complex findings into actionable and understandable information for various stakeholders. Here's why report creation is indispensable:

- Documenting the Journey

  The report captures the entire process—from understanding the problem statement to presenting the solution. It highlights the team's methodologies, challenges encountered, and decisions made, ensuring that their efforts are fully documented.

- Providing Clarity and Transparency

  A structured report explains how data was handled, models were chosen, and insights were derived. This transparency builds trust in the team's work and ensures evaluators can easily follow their approach.

- Showcasing Key Insights

  Hackathons often generate valuable insights that could be overlooked without proper documentation. A report consolidates these insights into a format that is easily accessible to all stakeholders, including judges, mentors, and sponsors.

- Supporting Evaluation

  Judges and evaluators rely on reports to assess the depth, accuracy, and creativity of a team's work. A detailed report provides the context and justification for decisions, enabling fair and thorough evaluation.

- Driving Decision-Making

  For hackathons tied to real-world problems, reports serve as a bridge between technical results and actionable recommendations. Stakeholders can use the insights to implement changes or explore further developments.

- Enhancing Collaboration and Learning

  A well-written report is a valuable resource for team members, mentors, and the broader community. It fosters collaboration by allowing others to understand and build upon the work done during the hackathon.

What a Report Does for All in a Hackathon

- For Participants
  - Helps articulate their work, showcasing both technical and problem-solving skills.
  - Serves as a portfolio piece for future opportunities.
- For Judges
  - Provides a comprehensive view of the solution, methodology, and innovation.
  - Simplifies comparison and evaluation across multiple teams.
- For Mentors and Organizers
  - Highlights successful strategies and common challenges.
  - Offers feedback for improving future hackathons.

Structure of a Good Report

Reports should effectively summarize the findings, methodologies, and insights gained during the hackathon tasks. The report will be a PowerPoint presentation covering the following topics. Here's a suggested structure for the report:

Title Page

- Hackathon Name: **Annual Hackathon 2025**
- Team Name: [Insert Team Name]
- Team Members: [List Names]
- Submission Date: [Insert Date]

Table of Contents

1. Executive Summary
2. Introduction
3. Methodology and Approach
4. Key Findings
    - Data Preprocessing
    - Exploratory Data Analysis (EDA)
    - AI Task Results
5. Insights and Recommendations
6. Conclusion and Future Work
7. Appendices

1. Executive Summary

- Brief overview of the report.
- Summary of the problem statement and the team's approach to solving it.
- Highlights of key insights and actionable recommendations.

2. Introduction

- Objective of the hackathon.
- Overview of the dataset and tasks.
- Challenges addressed during the event.

3. Methodology and Approach

- Description of the tools and technologies used (e.g., Python, TensorFlow, Pandas).
- Detailed workflow for each task:
  - **Task 1: Data Preprocessing**
    - Techniques used for cleaning and transforming data.
    - Handling missing values, outliers, and data standardization.
  - **Task 2: Insights (EDA)**
    - Key questions explored.
    - Graphical insights using visualizations like bar charts, scatter plots, and heatmaps.
  - **Task 3: AI Tasks**
    - AI models implemented (e.g., Logistic Regression).
    - Training and evaluation methods.

4. Key Findings

- **Data Preprocessing**
  - Issues identified in the raw data.
  - Steps taken to resolve them.
- **Exploratory Data Analysis (EDA)**
  - Trends and patterns uncovered.
  - Examples:
    - How gender influences calorie burn across workout types.
    - Correlations between BMI and session duration.

- **AI Task Results**
  - Model performance metrics (e.g., accuracy, precision, recall).
  - Key predictions and insights derived.

5. Insights and Recommendations

- Business or operational implications of the findings.
- Suggestions for improving the dataset or processes.
- Potential use cases for the AI models developed.

6. Conclusion and Future Work

- Summary of the project's impact.
- Limitations encountered.
- Future improvements or additional areas for exploration.

7. Appendices

- Code snippets for key steps.
- Visualizations and charts.
- References and external links.

Judgement Criteria

The **Judgment Criteria** section outlines how the participants' work will be assessed during the hackathon. It ensures that the evaluation process is fair, transparent, and aligned with the hackathon's goals. Here’s what this section will typically include:

Judging Criteria	**Description**
Understanding Problem Statement	Clarity in understanding the problem, objectives, and constraints.
Identifying Important Parameters	Ability to identify key features and parameters influencing the solution.
Valid Statistical and Numerical Inferences	Accuracy and relevance of statistical or numerical insights drawn from the data.
Exploratory Data Analysis (EDA)	Quality and depth of data visualization and analysis for uncovering patterns.
Model Selection	Appropriateness of the model(s) chosen for the problem statement.
Model Evaluation	Robustness, accuracy, and interpretability of the model(s).
Prompt Scoring	Creativity and accuracy in crafting effective prompts for solutions.

Scoring Allotment

1. **Preprocessing Tasks**: 10 points
    - This includes data cleaning, transformation, handling missing values, and ensuring the dataset is ready for analysis.
2. **Exploratory Data Analysis (EDA) Tasks**: 30 points
    - Focus on uncovering patterns, trends, and insights using data visualization and summary statistics.
3. **AI Tasks**: 30 points
    - This includes selecting and building appropriate AI or machine learning models, optimizing their performance, and ensuring they align with the problem statement.
4. **Reporting Task**: 30 points
    - Emphasis on presenting findings, models, and recommendations clearly and effectively through reports, visualizations, or presentations.

Frequently Asked Questions

1. What resources are provided during the hackathon?

Participants will have access to:

- Sample datasets for tasks.
- A Python notebook with the hackathon questionnaire.
- Pre-configured virtual environments for AI/DA projects.
- Documentation and code repositories.

2. What are the key deliverables for the hackathon?

Each team must submit:

- A final report summarizing findings and insights.
- Visualizations and models created during the hackathon.
- Any code or scripts used in the tasks.

3. How will the submissions be judged?

Submissions will be evaluated based on criteria like creativity, technical implementation, data handling, actionability, and presentation. Detailed rubrics are provided in the Judging Criteria section.

4. Are there any penalties for late submissions?

No, submissions after deadlines are not accepted.

5. Can we use external tools or datasets?

No, participants cannot use external tools or datasets.

6. Is there a Q&A session with mentors?

Yes, dedicated support channels will be available throughout the hackathon, including sessions with Deloitte and TKA representatives for technical and logistical queries.

Conclusion

The **Annual Hackathon 2025** has been an incredible platform for collaboration, creativity, and innovation. Participants have demonstrated exceptional problem-solving skills, utilizing advanced tools and methodologies to address complex real-world challenges.

Through this journey, teams have:

- Gained hands-on experience with data preprocessing, exploratory data analysis, and AI model development.
- Explored innovative solutions and actionable insights.
- Strengthened their understanding of DA/AI pipelines.

We hope this hackathon has inspired participants to continue exploring the fields of data analysis and artificial intelligence. Remember, the skills and knowledge gained here can pave the way for impactful contributions in the future.

Thank you for participating, and we look forward to seeing you in future hackathons!

Appendices

A. Glossary of Terms

- **EDA (Exploratory Data Analysis):** A process of examining datasets to summarize their main characteristics.
- **IQR (Interquartile Range):** A measure of statistical dispersion used to detect outliers.
- **Overfitting:** A modeling error where the model performs well on training data but poorly on new data.

B. Code Snippets

Include code snippets for:

- Data preprocessing (e.g., handling missing values, fixing outliers).
- EDA visualizations (e.g., heatmaps, bar charts, scatter plots).
- AI model implementation (e.g., logistic regression, evaluation metrics).

C. Troubleshooting Guide

- **Common Issue:** Dataset not loading.
  - **Solution:** Ensure the file path matches the provided dataset name.
- **Common Issue:** Errors in AI model training.
  - **Solution:** Verify data preprocessing steps (e.g., scaling and encoding).
- **Common Issue:** Visualization issues in notebooks.
  - **Solution:** Ensure all necessary libraries (e.g., Matplotlib, Seaborn) are installed and imported.

D. References and Links

- Python Documentation: https://docs.python.org/
- Scikit-learn Documentation: https://scikit-learn.org/

Reflex, Cain Road, Bracknell, Berkshire RG12 1HL
T: 01344 203999 F 01344 203998
info@theknowledgeacademy.com – www.theknowledgeacademy.com