Programme

Master of Software Engineering
180 Credits

Course

MSE803: Data Analytics
(15 credits)

Assessment 2

Case Study with Presentation

Student: Fabricio Mello Mulato
ID: 270516212
Lecturer: Mukesh Mishra
Date: July 2025

Total words (Latex): 5,086

Table of Contents

List of Figures

List of Tables

Introduction

This report presents a comprehensive data analytics case study aimed at improving a fitness tracking mobile application. The study combines advanced statistical and Machine Learning (ML) techniques to estimate user patterns, enhance app performance, and provide actionable insights for software engineering decisions. It also explores the ethical and cultural considerations necessary for responsibly handling user data, ensuring that all recommendations align with privacy regulations and inclusivity principles.

The report is structured around two main tasks. Task 1 uses statistical methods, exploratory analysis and machine learning to predict user behaviour and how people use a fitness app. It also aims to identify the key factors that influence this behaviour and how they affect decisions in software engineering. Task 2 focuses on ethical issues and the cultural context of data collection and analysis. It looks at privacy concerns, suggests ways to reduce risks, and recommends guidelines to meet industry standards and support the responsible design of software and data collection.

The analysis was carried out using Python libraries within a Jupyter Notebook, executed in Visual Studio Code, under a virtual environment defined by a requirements.txt file. To maintain a cleaner layout and improve readability, screenshots of code cells were intentionally omitted; instead, all relevant code is provided in the Appendix, found at the end of this document.

Below is a brief overview of the key libraries used, each selected for its particular strengths in handling different stages of the analytics process [1]–[5].

- pandas – for data manipulation and cleaning in tabular form.
- numpy – to perform numerical computations and array-based operations.
- matplotlib – for generating basic yet effective visual representations.
- seaborn – to enhance statistical plots with clear and attractive formatting.

- scikit-learn (sklearn) – applied to build and evaluate machine learning models, including regression and clustering.
- xgboost – a powerful tool for optimised predictive modelling in regression tass.
- warnings – used to suppress non-critical system alerts for a smoother workflow.
- scipy – provided tools for hierarchical clustering and dendrogram visualisation.
- StandardScaler (from sklearn) – standardised feature scales for fair model comparison.
- GridSearchCV (sklearn) – performed hyperparameter tuning to refine model performance.
- PCA (Principal Component Analysis) – reduced feature dimensions for clearer cluster visualisation.
- AgglomerativeClustering – implemented for hierarchical group analysis.
- KMeans – partitioned data into distinct user behaviour clusters.
- train_test_split – partition the data into train and test sets.
- r2_score & mean_squared_error – measured the accuracy and error of regression models.

These tools collectively allowed for a robust and insightful exploration of the data, balancing statistical rigour with clarity in communication.

1 Task 1: Advanced Statistical Analysis for Software Engineering

1.1 Understanding and Data Preparation

In order to understand the data, the provided CSV file was loaded into a Pandas dataframe, as shown in Table 1. In this initial analysis, we examined the number of columns, their respective names and data types, as well as the total number of records. In this step, it is important to check for any missing, zero, or null values. No null, zero, or missing (NaN) values were identified.

Table 1: Original data after loading into a pandas DataFrame

	User ID	Gender	Age	Activity Level	Location	App Sessions	Distance Travelled (km)	Calories Burned
0	1	Female	22	Active	Suburban	151	140	728
1	2	Male	50	Active	Rural	157	115	523
2	3	Male	36	Active	Urban	178	163	795
3	4	Female	36	Active	Suburban	155	85	715
4	5	Male	34	Sedentary	Rural	95	71	439
5	6	Male	41	Sedentary	Suburban	80	60	197

Generated by the author based on Python notebook.

The next step involves identifying the meaning of each variable, verifying whether the values are reasonable, and understanding the type of information each one provides, including their respective units of measurement. In Table 2 is a summary of our understanding.

Table 2: Description of dataset features.

Feature	Description
User ID	Unique identifier (removed for modelling)
Gender	Male/Female
Age	Numeric
Activity Level	Categorical (Sedentary, Moderate, Active)
Location	Categorical (Urban, Suburban, Rural)
App Sessions	Number of sessions (proxy for engagement)
Distance Travelled (km)	Physical activity distance
Calories Burned	Calories spent (measure of activity)

Summarised by the author based on Python notebook.

The User ID column is not useful for the analyses and, although it does not directly identify individuals, we chose to delete it from the dataframe for security, confidentiality, and anonymity purposes, for the next steps.

The dataset includes five numerical variables with 5,000 records each. The Age variable ranges from 18 to 59 years, with a mean of 38.42, indicating a relatively young adult population. App Sessions vary between 50 and 199, averaging 124.37 sessions per user. Distance Travelled spans from 26 km to 195 km, with a mean of 92.55 km, reflecting diverse physical activity levels. Finally, Calories Burned ranges widely from 102 to 987 calories, with an average of 432.32, consistent with the variation in distance and activity. See Table 3 for more details.

Table 3: Basic Statistics - Numerical

	Age	App Sessions	Distance Travelled (km)	Calories Burned
count	5000.00	5000.00	5000.00	5000.00
mean	38.42	124.37	92.55	432.32
std	12.16	42.69	36.75	187.43
min	18.00	50.00	26.00	102.00
25%	28.00	89.00	63.00	285.00
50%	39.00	124.00	89.00	406.00
75%	49.00	161.00	118.00	560.00
max	59.00	199.00	195.00	987.00

Generated by the author based on Python notebook.

Regarding categorical features, Table 4 shows a relatively balanced gender distribution, with males slightly dominant. Activity levels are mostly moderate, suggesting an intermediate user profile. Location is evenly spread, with a slight predominance of rural users, indicating no strong bias across the three geographic categories.

Table 4: Basic Statistics - Categorical

	Gender	Activity Level	Location
count	5000	5000	5000
unique	2	3	3
top	Male	Moderate	Rural
freq	2538	1734	1684

Generated by the author based on Python notebook.

1.2 Exploratory Data Analysis (EDA)

As noted in Seltman [6], Exploratory Data Analysis (EDA) is one of the initial steps in examining and visualising data to detect errors, identify patterns, test assumptions, and explore relationships, all without relying on formal statistical methods.

The first step involved plotting each numerical variable using box plots to assess the variability of the data and detect potential outliers, as shown in Figure 1.

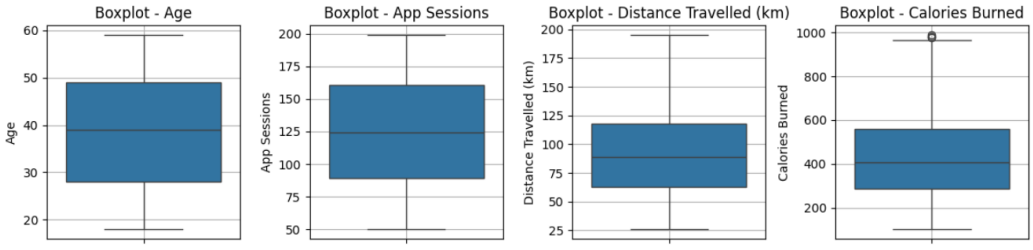


Figure 1: Boxplot analyses for numerical features

As discussed by VanderPlas [4] and Borman [7], outliers should be retained when they reflect true data variability, and removed only if they result from errors or distort analysis. In this study, we chose to keep them, as they appear valid and likely reflect natural variability due to the proximity of other values. In some cases, such values may also represent rare but meaningful events. Although a possible outlier was observed in Calories Burned, further inspection of the value suggests it is not a result of a data

entry error nor a typical outlier, but rather reflects the natural variability within the dataset.

For the categorical variables, we plotted simple bar charts to examine the distribution of each category. As shown in Figure 2, the data appear well balanced, with no dominant group in any of the categories.

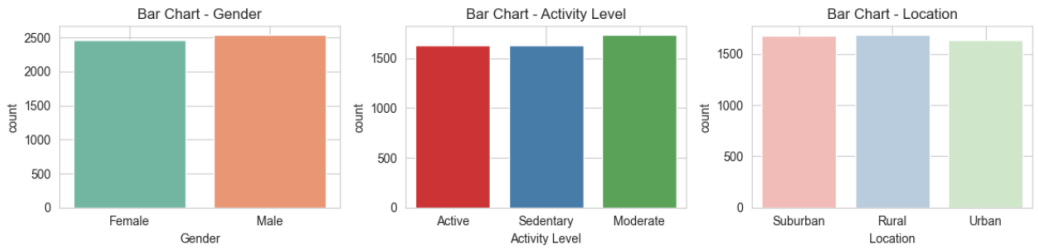


Figure 2: Bar Chart analyses for categorical features

The next step was to generate histograms. Histograms display the variability and distribution of the data, helping to identify intervals, minimum and maximum values, averages, and any abnormal patterns. Figure 3 was used for the following analyses.

The age of users ranges from around 18 to 60 years, with a relatively uniform distribution across this span. There is no significant concentration in any specific age group, suggesting that the app appeals to a wide and diverse audience. The absence of users under 18 may indicate age restrictions or a focus on adult users, while the upper limit near 60 could reflect a natural decline in digital engagement among older adults. Overall, the app appears inclusive across adult age groups.

Most users recorded between 80 and 150 app sessions, indicating moderate engagement. Distance travelled and calories burned both show right-skewed distributions, with most users covering 40–100 km and burning 200–500 calories. This implies that the app is mainly used for light to moderate physical activity. Overall, the patterns reflect balanced user demographics and typical usage behaviour.

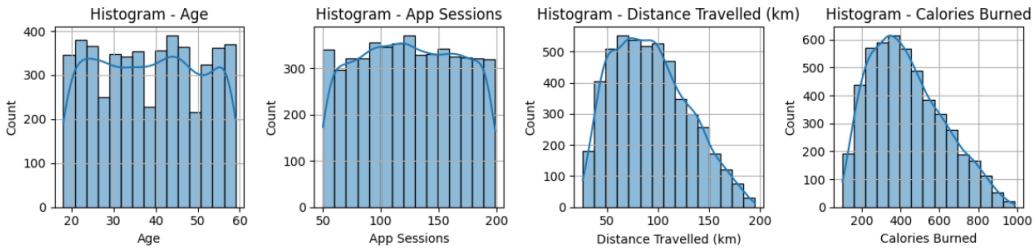


Figure 3: Histograms for numerical features

As shown in Figures 4, 5, and 6, the box plot analysis of each feature against engagement and app performance indicators suggests that only Activity Level significantly influences app sessions, distance travelled, and calories burned. The results show no meaningful differences by gender, while location presents slight variation, particularly in urban areas. Activity level remains the most distinctive factor, with active users showing higher values across all metrics, confirming common expectations.

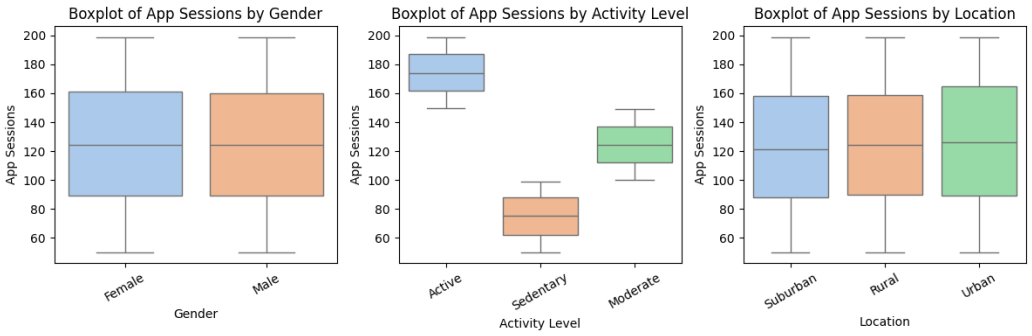


Figure 4: Box-plots for App Session

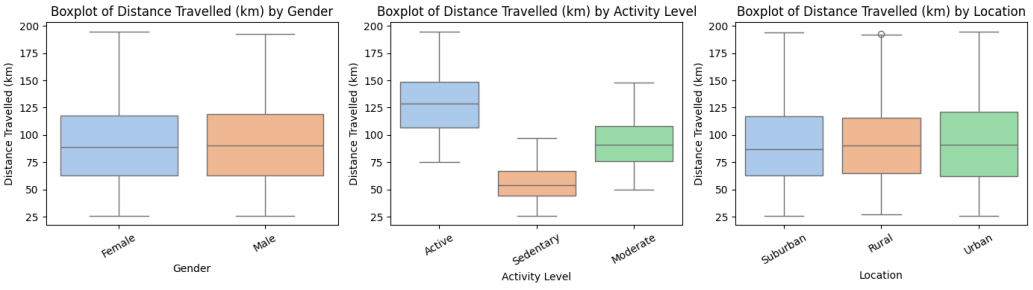


Figure 5: Box-plots for Distance Travelled

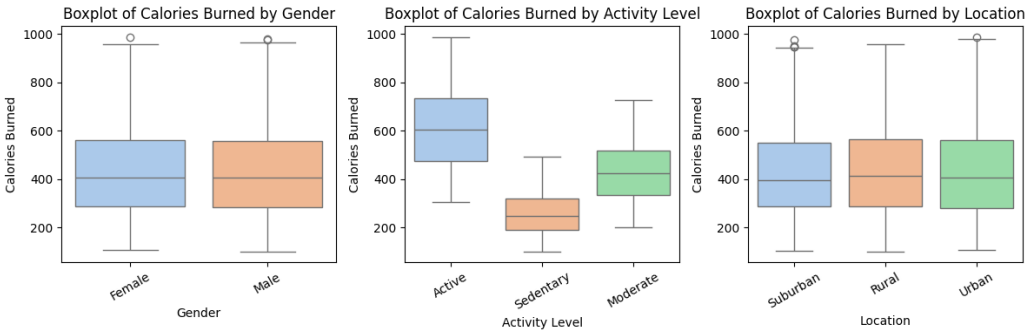


Figure 6: Box-plots for Calories Burned

Scatter plots were also generated for each variable; however, they did not reveal any new insights beyond those already observed in the previous graphs. For this reason, they are not included here. As an illustrative example, Figure 7 shows a scatter plot for Activity Level, which reinforces the earlier findings. The complete set of plots is available in the accompanying Python notebook and in the appendix of this report for further reference.

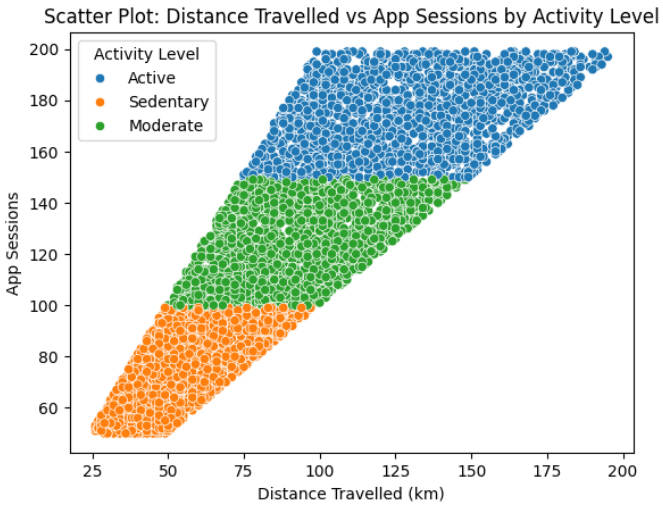


Figure 7: Scatter-plotter for Activity Level

The correlation matrix shows in Figure 8, that age has no meaningful impact on app usage or physical activity. In contrast, app sessions are strongly correlated with both distance travelled (0.86) and calories burned

(0.8). These three variables are closely related, indicating that greater engagement in the app aligns with higher physical activity levels. Pearson's method was chosen as the variables are continuous, approximately normally distributed, and exhibit linear relationships, making it suitable for parametric analysis, as explained by Borman [7].

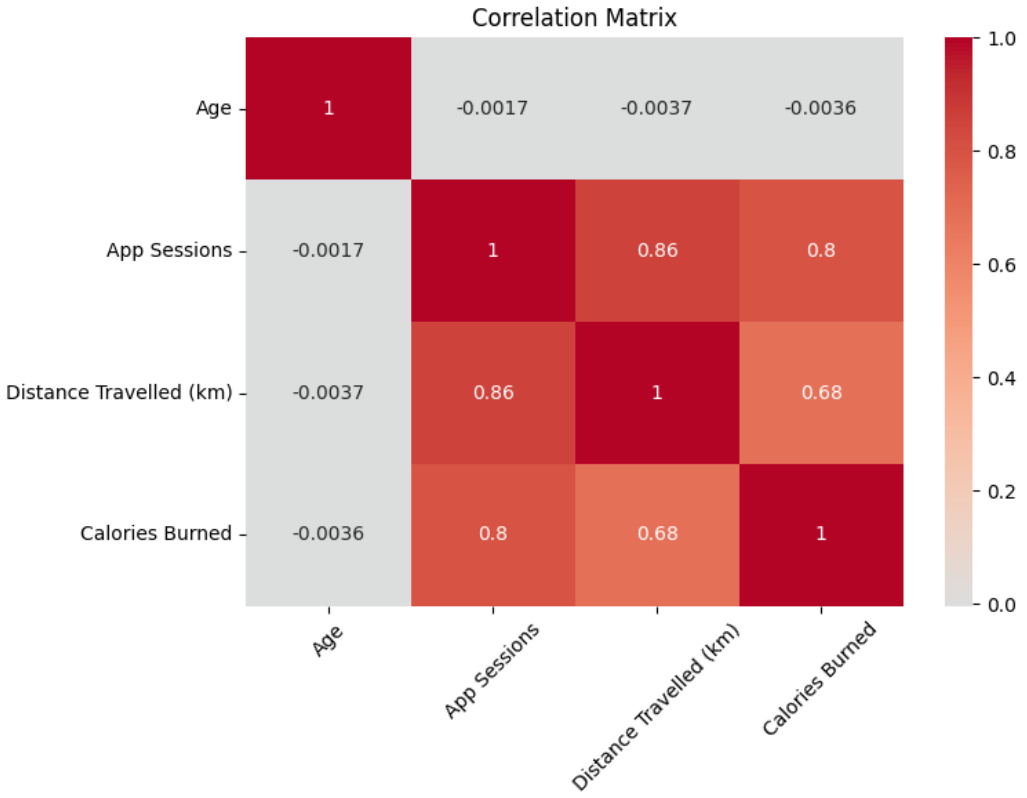


Figure 8: Correlation Matrix - original dataset

1.3 Pre-processing and Transformations

Before applying ML models, the dataset must be suitably prepared through appropriate pre-processing steps. This process includes selecting relevant features, performing necessary transformations to ensure model reliability and fairness, and segmenting the data to train and test.

Qualitative variables were transformed into values using ordinal mapping, as they represent ordered categories, eliminating the need for

dummies or one-hot encoding. Gender was encoded as 0 (Male) and 1 (Female); Location as 0 (Rural), 1 (Suburban), and 2 (Urban); and Activity Level as 1 (Sedentary), 2 (Moderate), and 3 (Active). Although Location has relatively low importance, it was retained and treated as ordinal based on a reasonable rural-to-urban progression.

Feature importance was then assessed using a Random Forest Regressor to identify which variables contribute most to predicting app engagement, as shown in Figure 9. The results of this analysis corroborate the findings presented in the correlation matrix, indicating that the most significant features, in descending order of importance, are: Distance Travelled (km), Calories Burned, and Age.

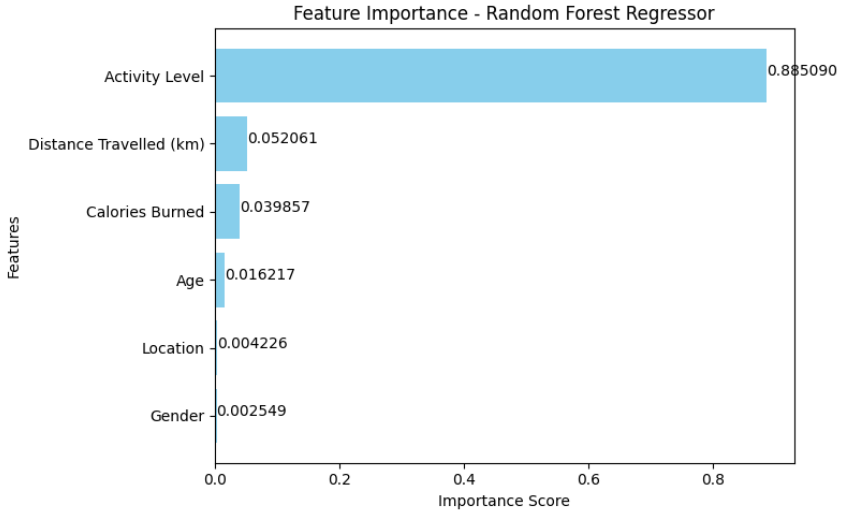


Figure 9: Features Importance

A new correlation matrix was generated, now including the categorical features transformed into numerical form for further analysis, which confirms the previous results indicating that Age, Location, and Gender have little or no influence on App Sessions.

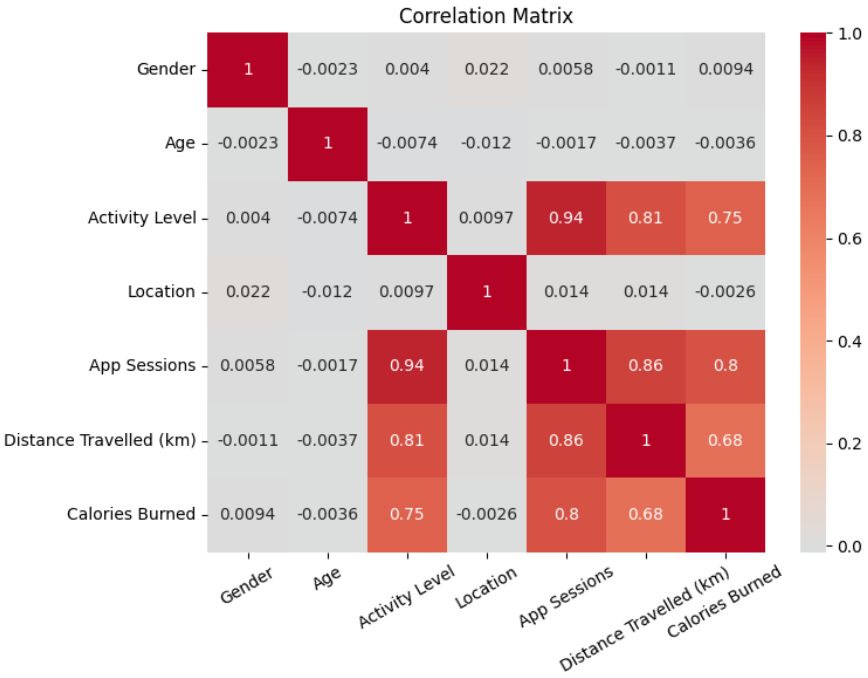


Figure 10: Correlation Matrix - after transformations in the dataset

In this analysis, we selected Activity Level, Distance Travelled (km), and Calories Burned as predictor variables (inputs, X) due to their clear representation of user physical effort and performance. Distance reflects the intensity of physical activity, while Calories represent the outcome of these efforts, both providing essential context for user behaviour. App Sessions, chosen as the target variable (output, y), serves as the primary indicator of user engagement with the app. This selection was justified by prior exploratory analyses, including correlation matrices and feature importance assessments, which demonstrated strong associations between these predictors and app usage, ensuring a robust and meaningful modelling approach.

The data sample was divided into training and testing subsets using a standard 80/20 split to guarantee that the model is assessed on unseen data. Since all selected features are continuous and numerical now, there was no need to apply dummy variables or one-hot encoding, which are typically used for categorical variables. Additionally, the dataset contains no missing or zero values, simplifying the transformation process and avoiding data imputation procedures.

To ensure the model treats each feature equally, avoiding bias, especially those with different scales, we applied standardisation to the predictor variables (X). Standardisation brings all features to a common scale, improving the performance and interpretability of algorithms that rely on distance or magnitude. It is important to note that standardisation was applied only to X, as the target variable (y) does not require scaling in regression tasks.

According Géron [2], feature engineering is a technique of modifying raw data into meaningful inputs for ML algorithms, whether by selecting the most relevant existing variables, creatively combining them to reveal hidden patterns, or crafting entirely new features through additional data collection.

As part of feature engineering, a new variable, Intensity per Session, was created to capture the average physical effort per app session. This feature combines information from three existing variables, Calories Burned, Distance Travelled (km), and App Sessions, by computing the calories burned per kilometre and then dividing by the number of sessions. This derived metric offers insight into the typical intensity of each session for a user, as Equation 1. However, to avoid multicollinearity, it will be used exclusively in the clustering stage, rather than in predictive modelling, where the original features remain more appropriate.

$$\text{Intensity per Session} = \frac{\text{[Calories Burned] / [Distance Travelled (km)]}}{\text{App Sessions}} \quad (1)$$

1.4 Modelling: Behaviour Prediction

Machine Learning (ML) is a multidisciplinary field that combines elements of artificial intelligence, statistics, and computer science to develop algorithms capable of learning from data and improving performance over time [5]. Géron [2] also added that this can be achieved without being explicitly programmed. These algorithms are generally divided into four main categories: supervised learning, unsupervised learning, semi-supervised learning, and reinforcement learning, each serving distinct purposes such as prediction, pattern discovery, or decision-making in uncertain contexts [8], [9].

This study concentrates on two core machine learning approaches: supervised and unsupervised learning. The first one, was applied through regression models to predict continuous values, specifically the number of app sessions based on user activity, allowing us to assess engagement and app performance. In parallel, unsupervised learning was employed for clustering, enabling the identification of patterns and similarities among users to group them based on shared characteristics. Reinforcement learning is not considered, as it does not align with the objectives of this study.

1.4.1 Supervised ML Models — Regression

According to Yuan, Rocha Neto, Rothenberg, *et al.* [9], supervised learning is based on labeled data to understand how input variables relate to output, enabling classification (to predict categories) or regression (to estimate continuous values).

A baseline was established using three models: Linear Regression (LR), Random Forest (RF), and K-Nearest Neighbours (KNN), applied directly to the original data without any transformations (models 1) and using default hyperparameter. This approach made it possible to assess the impact of subsequent pre-processing steps and transformations on model performance. After these transformations, a fourth algorithm, XGBoost, was applied. The number 2 identifies models after transformations. In an effort to further improve the results, the three best-performing models underwent hyperparameter optimisation using the GridSearch library. In total, ten models were tested.

- LinearRegression 1
- RandomForestRegressor 1
- K-NeighborsRegressor 1
- XGBoost 2
- LinearRegression 2
- RandomForestRegressor 2

- K-NeighborsRegressor 2
- RandomForestRegressor_optimized
- KNeighborsRegressor_optimized
- XGBoost_optimized

For this analysis, as the variables of interest (engagement and app performance) are numerical, regression algorithms are the most appropriate choice. Next, a brief overview of each selected algorithm is presented.

Linear Regression (LR) is a regression method designed to fit a hyperplane to the training set in order to predict real or continuous values. This model presumes a linear connection between the input and the output, minimising the sum of squared errors to determine the best-fitting model [2], [9].

Random Forest (RF) is an ensemble algorithm that merge the forecasts of several decision trees, using majority voting for classification and mean for regression, in order to produce a more reliable final prediction [10]. A Decision Tree (DT) is a directed tree framework in which non-leaf nodes represent features used to split data based on their distinct values, and leaf nodes provide the final labels. Unlike other models, it makes decisions by progressing from the root to the terminal nodes, handles categorical variables effectively, and produces results that are easy to interpret, although it has certain limitations [8], [10].

K-Nearest Neighbour (KNN) is a supervised ML method that assigns new data points to a class or value based on the most common class or average of the k closest training samples, using distance measures such as Euclidean distance, and is also useful for handling missing data through similarity between neighbours [2], [10].

XGBoost is a category of boosting technique that blend various weak classifiers to form a stronger model, typically decision trees, offering enhanced performance, particularly for imbalanced data. It is gradient-based and includes improvements that support parallel and distributed processing during tree construction [11].

A common strategy for boosting ML performance involves the tuning or refinement of hyperparameters, prioritising a balance between performance and tuning efficiency [12]. The scikit-learn library provides the grid search tool, which do an full search for the best combination of hyperparameter values [4]. In this study, **GridSearchCV** was applied with cross-validation for regression, classification, and other estimators, and was used to increase the results of the three best models by varying their hyperparameters [2].

All code written in Python is available in the Appendix: Python Code – Jupyter Notebook, containing fully annotated scripts and corresponding outputs.

1.4.2 Unsupervised ML Models — Clustering

As Yuan, Rocha Neto, Rothenberg, *et al.* [9] explain, unsupervised learning operates without labelled responses, with the aim of uncovering hidden behaviour or structures through strategies including clustering or reduction of dimensionality. These algorithms work with unlabelled datasets to identify meaningful patterns and relationships. Cluster-based methods group data points according to similarities, while association-based algorithms reveal dependencies between items, an approach that is particularly effective for generating business value, especially in marketing and commercial contexts [9], [10].

K-means is an iterative clustering model that divide N observations into K clusters by grouping data points based on proximity to cluster means. It begins with a random selection of initial means and, at each step, assigns points to the nearest mean, updating the means according to the assigned points. This process continues until the means stabilise [8].

Hierarchical clustering organises data through successive partitions, ranging from a single cluster containing all data points to multiple clusters, each with an individual point. The structure is typically visualised using a tree diagram known as a dendrogram. Unlike methods that define a fixed number of clusters, it forms groups step by step based on similarity. Agglomerative approaches merge data points into larger clusters, while divisive methods separate them into smaller ones. This technique is widely applied in text mining, bioinformatics, and market segmentation to reveal

hidden patterns [13].

Both models were run in a Jupyter Notebook using Python, available in the Appendix. Based on the analysis of the two models, it was possible to identify the suitable amount of clusters and assign each record to its respective cluster, allowing the examination of the characteristics of each group.

1.5 Model Evaluation and Results

Performance metrics are essential for evaluating and comparing how effectively machine learning models address specific problems. They provide quantitative measures of model accuracy, enabling researchers to assess prediction quality, identify areas for improvement, and support informed decision-making when selecting between algorithms.

1.5.1 Regression

To assess the regression models, we use Mean Squared Error (MSE) and R-squared (R^2), which are widely recognised as appropriate for such tasks [14].

MSE calculate the mean squared difference between forecast and true values, where reduced values denote greater accuracy. R^2 quantifies the percentage of variance in the dependent variable explained by the independent variables, ranging from 0 (no explanatory power) to 1 (perfect fit); thus, elevated R^2 scores reflect a more accurate model.

The Table 11 depicts that the XGBoost_optimized model achieved the best performance, with the minimum MSE (115.51) and the maximum R^2 value (0.938). It signifies enhanced predictive performance and clearer insight into data variability compared to the other models. Closely following, the RandomForestRegressor_optimized model also performed well, with an MSE of 116.37 and R^2 of 0.937, followed by the KNeighborsRegressor_optimized. Overall, the optimized models outperformed their non-optimized counterparts. Furthermore, ensemble

and boosting algorithms yielded superior results to linear regression models, which showed the highest errors and lowest R^2 values.

	Model	Mean Squared Error	R-squared
0	XGBoost_optimized	115.507339	0.937941
1	RandomForestRegressor_optimized	116.370811	0.937477
2	KNeighborsRegressor_optimized	133.104143	0.928487
3	XGBoost 2	134.066483	0.927969
4	K-NeighborsRegressor 2	136.374000	0.926730
5	RandomForestRegressor 2	139.267554	0.925175
6	LinearRegression 2	140.743202	0.924382
7	RandomForestRegressor 1	309.735909	0.833587
8	K-NeighborsRegressor 1	323.313800	0.826292
9	LinearRegression 1	331.928251	0.821663

Figure 11: Best models for regression problem

1.5.2 Clustering

For clustering, metrics like Elbow Method and Silhouette Score and are commonly used to guide model selection without requiring labelled data.

The Elbow method, or also known as Within-Cluster Sum of Squares (WCSS), identifies the ideal cluster count by analysing the reduction in inertia as clusters are added; The ‘elbow’ point, indicating a diminishing rate of decline, suggests the ideal cluster count [15]. According Muharram, Nalawati, Warsuta, *et al.* [15] the Silhouette Score evaluates how well data points fit within their clusters, with higher values indicating more cohesive and well-separated groups.

In the analyses, the Elbow Method indicated 2 or 3 clusters, while the Silhouette Score suggested 2 as optimal.

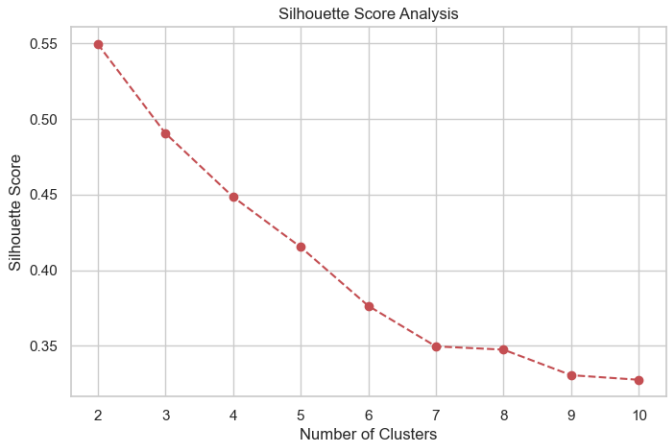


Figure 12: Silhouette Method for Clustering

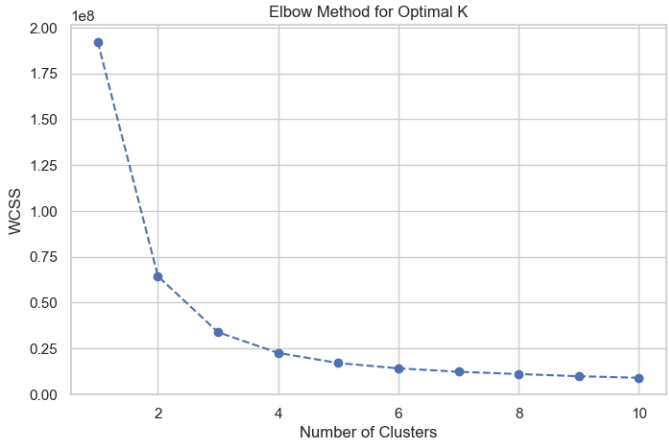


Figure 13: Elbow Method for Clustering

Principal Component Analysis (PCA) is a useful unsupervised technique for simplifying complex datasets by reducing their dimensionality, often from many variables to just two. These two new variables are combinations of the original ones and may not have direct meaning, but they allow for effective visualisation of data in a simple two-dimensional plot. This is particularly helpful for identifying and exploring clusters and understanding how many distinct groups exist in the data. PCA is widely used because it provides a clear first look at high-dimensional data [4], [8]

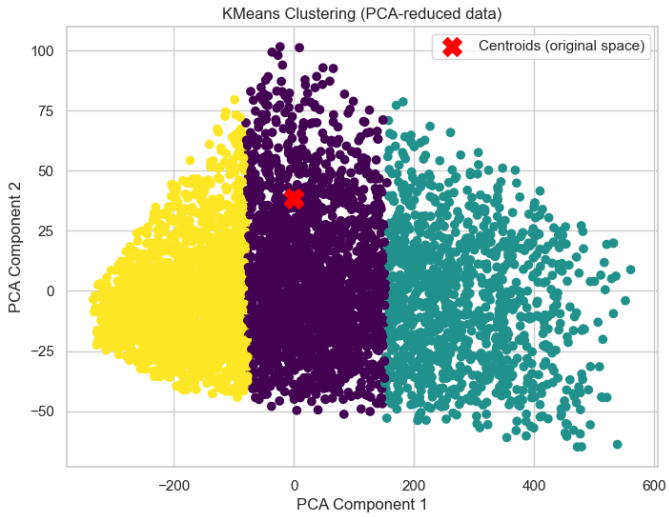


Figure 14: K-means groups, using PCA (Principal Components Analysis)

Regarding the choice of the best number of clusters, the combined use of WCSS, Silhouette Score, and Hierarchical Dendogram, provides useful insights. The Elbow method suggests an inflection point at K=2 or K=3, where adding more clusters yields diminishing returns. The Silhouette Score shows the best clustering quality at K=2, with a calculated value of 0.51. The dendrogram tree also suggests 3 clusters, as the cut line that best separates the data divides it into three well-defined groups.

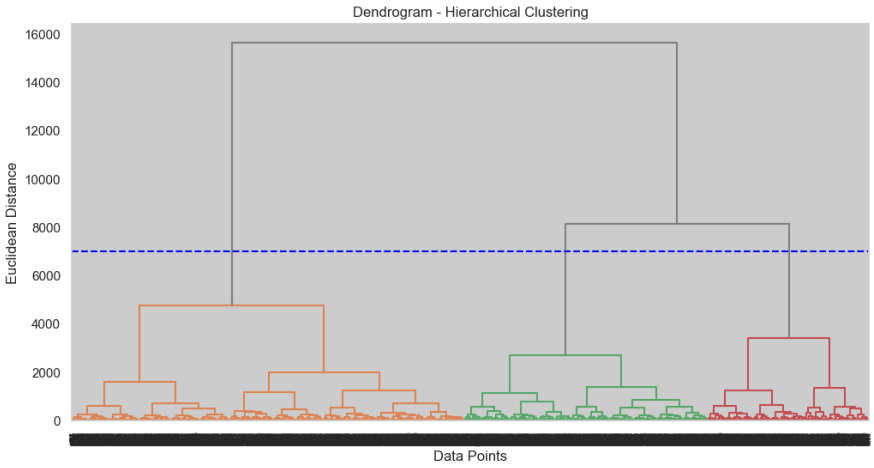


Figure 15: Hierarchical Method for Clustering

Based on these results, 3 clusters were chosen to balance statistical optimisation and practical interpretability. This allows segmentation that, while not reaching the highest Silhouette Score, still ensures good separation and clearer understanding of each group's characteristics.

In the Section 1.6, we examine the predominant characteristics of each group. It is also possible to classify a new user into one of these clusters based on limited usage data, enabling tailored marketing strategies to be directed at individual clients according to the cluster they most closely resemble.

Table 5: Visualising Clusters Characteristics

Cluster	Age	App Sessions	Distance Travelled (km)	Calories Burned	Intensity per Session	Gender	Activity_Level	Location
0	38.604451	137.494689	102.899848	454.188164	0.038278	[Male, 51.3]	[Moderate, 52.3]	[Rural, 34.5]
1	38.060000	169.677273	125.709091	710.659091	0.035651	[Male, 50.5]	[Active, 81.1]	[Urban, 34.6]
2	38.445658	84.950598	62.942798	250.611024	0.056736	[Male, 50.3]	[Sedentary, 72.9]	[Suburban, 34.8]

Generated by the author based on Python notebook.

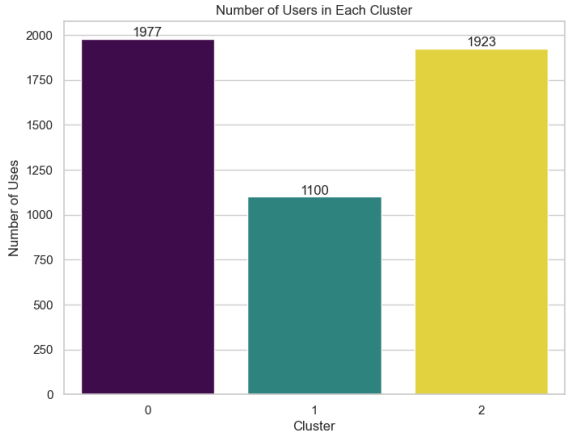


Figure 16: Number of Users in Each Cluster

1.6 Discussion

Concerning the supervisioned model for regression, the careful pre-processing, feature engineering, and dataset splitting established a solid basis for modelling. The applied transformations, together with the

chosen optimiser and hyperparameter tuning via GridSearchCV, effectively enhanced model performance. Supervised regression with ensemble and boosting methods, especially the optimized XGBoost, demonstrating superior accuracy and robustness. These findings highlight the critical role of thorough data preparation and optimisation in producing reliable, interpretable, and high-performing ML models. Hence, the most effective model may be utilised to predict user behaviour, app usage patterns, and engagement for new users, based on profile and activity data, particularly Activity Level, Distance Travelled (km), and Calories Burned.

Regarding clustering, all clusters show similar age distributions, indicating that age is not a key driver of app engagement. Some insights about the dataset and clusters:

Cluster 0: Moderately Active These users, mainly from rural areas, display steady app usage and moderate activity levels. This is the largest group, with 1,977 users. They record an average of 137 app sessions, 102.9 km travelled, and 454 kcal burned, with moderate intensity per session. They would benefit from personalised progress plans, milestone tracking, and gentle performance tips to support gradual improvement.

Cluster 1: Highly Active The most active group, predominantly urban, is driven by measurable outcomes. This is the smallest group, with 1,100 users. They record an average of 170 app sessions, 125.7 km travelled, and 711 kcal burned, but with lower intensity per session. They are likely to engage with advanced performance metrics, wearable integration, leaderboards, and social challenges.

Cluster 2: Sedentary This group shows the lowest activity and app usage, suggesting the need for greater encouragement. It is the second-largest group, with 1,923 users. They record an average of 85 app sessions, 62.9 km travelled, and 251 kcal burned. Notably, this group shows the highest intensity per session, meaning that although they are less active overall, their calorie burn per session is higher. This indicates untapped potential, as these users achieve strong results when they do engage. Therefore, they represent a key target for strategies aimed at increasing frequency and building sustainable habits. Gamified challenges, starter fitness paths, and motivational reminders could help increase engagement and support habit formation.

The clustering analysis, underpinned by rigorous pre-processing and thoughtful feature selection, identified clear user segments with distinct app engagement and activity patterns. These insights form a solid foundation for targeted marketing strategies and personalised features that better respond to user preferences, ultimately delivering more meaningful experiences and improving user retention and satisfaction.

The predictive models developed in this study (regression and clustering) provide valuable guidance for the software engineering team, enabling them to prioritise features that align with distinct user engagement patterns.

There are several implications for software engineering decision-making. For instance, higher predicted app use among active users supports investing in advanced workout tracking or performance dashboards, while recognising lower engagement in sedentary groups suggests introducing gamified challenges or motivational notifications to encourage positive habits. UX Customisation complements this by tailoring interfaces and features to users' activity levels, ensuring a personalised experience whether someone is more sedentary or active. Smart Notifications then deliver timely, relevant prompts, such as workout suggestions or achievable goals, based on individual behaviour, fostering engagement without overwhelming the user. Meanwhile, Goal Personalisation dynamically adjusts fitness targets using predicted behaviour to set realistic and motivating objectives that support each user's unique pace and potential.

2 Task 2: Ethical and Culturally Relevant Data Analysis

As part of the development team working on a personalised workout recommendation feature for a fitness tracking app, it is essential to approach data analysis through an ethically responsible and culturally relevant lens. While data-driven insights can enhance user engagement and improve health outcomes, the way in which data is collected, interpreted, and applied must be carefully considered to avoid ethical pitfalls and ensure inclusivity.

2.1 Ethical Data Collection in Software Engineering

The processing and use of personal data, for instance, age, gender, physical activity levels, and location, trigger substantial ethical debates, especially about privacy, consent, and informsation protection. Users may not fully understand how their information is processed or for what purposes it is retained. This creates a risk of misuse, such as unauthorised sharing with third parties, or algorithmic profiling that may lead to discriminatory outcomes.

Biable, Garcia, and Midekso [16] recommend a applicable ethical model for software requirements engineering that adapts 110 specific clauses from broad ACM/IEEE codes for daily use. Their work addresses industry concerns such as incomplete requirements, stakeholder reluctance, and data privacy. The framework offers clear guidance on ethical data collection, consent, and protection of personal information, helping engineers build safer and more ethical software products.

The main topics approached in this framework are **requirements identification** focuses on ensuring that all requirements are complete, relevant, and free from omissions or ambiguities. To bridge the **knowledge gap**, engineers must actively address missing information and promote ethical engagement from stakeholders. The **quality of requirements** must prioritise security, data privacy, and usability. The framework also warns against **forbidden actions** such as dishonesty, negligence, or introducing malicious requirements. Finally, it offers guidance for managing stakeholder

unwillingness to provide clear and precise requirements [16].

To mitigate such risks, clear data governance policies must be implemented like this framework or others. These include transparency regarding data usage, secure storage protocols, and the application of informed consent practices. Following the General Data Protection Regulation (GDPR) in the European Union, and New Zealand's Privacy Act 2020, users must be given the possibility to view, remove, and modify their personal information at any time. The principle of data minimisation, collecting only what is strictly necessary for the feature to function, is also central to compliance.

The GDPR is the European Union's legal framework that governs how personal data is collected, processed, and stored, ensuring strong protection for individuals' privacy rights [17]. Similarly, New Zealand's Privacy Act 2020 controls the management of personal data by organisations, promoting responsible data practices through transparency and accountability [17]. While both frameworks emphasise data minimisation and user rights, the GDPR is more stringent and includes obligations like appointing Data Protection Officers and managing international data transfers. In contrast, the Privacy Act is more flexible and context-driven, aligning with local cultural and legal expectations.

Furthermore, data should be encrypted during transmission and storage, and access restricted to authorised personnel only. Privacy impact assessments (PIAs) can be used at the planning stage to identify and address potential vulnerabilities, helping ensure that legal obligations and ethical standards are met.

2.2 Cultural Impact and Inclusive Design

The dataset shows variation across activity levels, location types (urban, rural, suburban), and gender. However, cultural factors such as socioeconomic background, access to green spaces, and cultural attitudes towards fitness are not directly captured. These elements can strongly influence how users engage with fitness apps.

For instance, individuals in rural or poorer communities may have

limited access to fitness facilities or may prefer physical activity forms not recognised by mainstream fitness metrics. Gender dynamics may also influence activity patterns, with some cultural groups showing differences in how men and women prioritise fitness. If these nuances are not considered, the algorithm could produce recommendations that are either irrelevant or unintentionally exclusive.

Therefore, it is crucial to ensure that data interpretation is contextualised. Demographic segmentation should be approached carefully, avoiding stereotypes or assumptions. Including optional self-reported cultural identifiers or preferred activity types could support more inclusive, personalised experiences.

I believe cultural factors play a key role in how people adopt and use fitness apps, as preferences differ by region in activity types, motivation, and privacy concerns. For instance, collectivist cultures often respond better to social challenges and group goals, while individualist cultures focus more on personal progress. Given this, the app should include culturally adaptive features, such as region-specific activities, inclusive language options, and flexible privacy settings, to increase relevance and user satisfaction. Such adaptations would make the app more appealing to a wider and more diverse audience.

2.3 Ethical Guidelines for Software Data

To endorse ethical integrity and comply with industry standards such as GDPR and ISO/IEC 27001, the following guidelines are recommended [16], [18], [19]:

- **Transparency and Consent:** Clearly inform users and obtain explicit consent for data collection, providing easy access to privacy controls.
- **Data Minimisation:** Collect only the data essential for each feature's functionality.
- **Anonymisation:** Anonymise personal data wherever possible to prevent user identification during analysis.

- **Fairness and Inclusivity:** Regularly audit algorithms for bias and ensure training datasets reflect diverse user populations.
- **Cultural Sensitivity:** Apply culturally responsive design principles by offering diverse fitness options and respecting different views on health and wellness.
- **User Empowerment:** Enable users to control their data fully, including options to delete, download, or restrict its use.

ISO/IEC 27001 serves as a global framework for Information Security Management Systems (ISMS). It enables organisations detect, assess, and handle information security risks, to maintain data integrity, accessibility, and confidentiality. It also enhances reputation and ethical and legal compliance, such as with the GDPR [18], [19].

Incorporating ethical and culturally relevant data practices is not just a matter of compliance; it is essential to building trustworthy, inclusive, and effective technology. By aligning the above principles with legal frameworks such as the GDPR, New Zealand's Privacy Act 2020, and Te Tiriti o Waitangi, and by centring user diversity and fairness in design, software teams can ensure that personalised fitness recommendations are both respectful and impactful.

Conclusion

In conclusion, this study successfully applied advanced data analytics to predict app engagement and segment users effectively. The findings provide valuable guidance for software engineering improvements and targeted user engagement strategies.

Through this project, I gained significant insights by connecting theory with practice. Applying statistical and machine learning concepts using real-world tools enhanced my understanding of data preprocessing, model evaluation, and ethical risk assessment. This practical experience helped reinforce theoretical knowledge and highlighted the challenges and opportunities of implementing data analytics and machine learning solutions in software engineering contexts.

Furthermore, ethical analysis highlighted the importance of data privacy, transparency, and cultural sensitivity, which are essential for building trustworthy and inclusive technology.

Despite the strengths of the models, some limitations remain. The dataset lacked detailed socioeconomic and cultural information, which may affect participation patterns, and the models may not generalise well to groups such as older adults or users with disabilities. Future work should address these gaps by using broader, more diverse datasets and incorporating user feedback to validate results. It should also include real-world testing, user consent for use of data, and regular reviews of model performance and fairness to ensure ongoing accuracy and ethical alignment.

References

[1] *Scikit-learn: Machine learning in python — scikit-learn 1.7.0 documentation*, https://scikit-learn.org/stable/, Accessed: Jun. 28, 2025, 2025.

[2] A. Géron, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*, 3rd ed. Sebastopol, CA: O'Reilly Media, 2022, ISBN: 9781098125974.

[3] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.*, "Scikit-learn: Machine learning in python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011. [Online]. Available: http://jmlr.org/papers/v12/pedregosa11a.html.

[4] J. VanderPlas, *Python Data Science Handbook: Essential Tools for Working with Data*, First. Beijing, Boston, Farnham, Sebastopol, Tokyo: O'Reilly Media, 2016.

[5] O. Simeone, *A brief introduction to machine learning for engineers*, arXiv preprint arXiv:1709.02840, Version 3, 2018. DOI: 10.48550/arXiv.1709.02840. [Online]. Available: https://doi.org/10.48550/arXiv.1709.02840.

[6] H. J. Seltman, *Experimental Design and Analysis*. Carnegie Mellon University, 2018, Available at http://www.stat.cmu.edu/~hseltman/309/Book/Book.pdf.

[7] D. Borman, *Statistics 101: From data analysis and predictive modeling to measuring distribution and determining probability, your essential guide to statistics* (Adams 101), First Adams Media hardcover edition. New York: Adams Media, 2018.

[8] R. N. Thomas and R. Gupta, "A survey on machine learning approaches and its techniques," in *2020 IEEE International Students' Conference on Electrical, Electronics and Computer Science (SCEECS)*, Bhopal, India: IEEE, 2020, pp. 1–6. DOI: 10.1109/SCEECS48394.2020.190.

[9] T. Yuan, W. B. da Rocha Neto, C. Rothenberg, K. Obraczka, C. Barakat, and T. Turletti, "Machine Learning for Next-Generation

Intelligent Transportation Systems: A Survey," working paper or preprint, Sep. 2019. [Online]. Available: https://inria.hal.science/hal-02284820.

[10] A. K. Azad, T. Atkison, and A. F. M. S. Shah, "A review on machine learning in intelligent transportation systems applications," *The Open Transportation Journal (TOTJ)*, vol. 18, no. 1, e26671212330496, Sep. 2024. DOI: 10.2174/0126671212330496240821114216.

[11] J. Tanha, Y. Abdi, N. Samadi, N. Razzaghi, and M. Asadpour, "Boosting methods for multi-class imbalanced data classification: An experimental review," *Journal of Big Data*, vol. 7, no. 1, p. 70, Dec. 2020. DOI: 10.1186/s40537-020-00349-y.

[12] W. Alawad, M. Zohdy, and D. Debnath, "Tuning hyperparameters of decision tree classifiers using computationally efficient schemes," in *2018 IEEE First International Conference on Artificial Intelligence and Knowledge Engineering (AIKE)*, Laguna Hills, CA: IEEE, Sep. 2018, pp. 168–169. DOI: 10.1109/AIKE.2018.00038.

[13] Y. Rong and Y. Liu, "Staged text clustering algorithm based on k-means and hierarchical agglomeration clustering," in *2020 IEEE International Conference on Artificial Intelligence and Computer Applications (ICAICA)*, 2020, pp. 124–127. DOI: 10.1109/ICAICA50127.2020.9182394.

[14] S. Guido and A. C. Müller, *Introduction to Machine Learning with Python*, First Edition. Sebastopol, CA: O'Reilly Media, Inc., 2016, ISBN: 978-1-449-36941-5.

[15] A. T. Muharram, R. E. Nalawati, B. Warsuta, I. M. Malik Matin, A. Pradiptyas, and M. Natanael, "Comparison of elbow, silhouette and dbi methods for clustering nutritional status of toddlers using k-means clustering," in *2024 12th International Conference on Cyber and IT Service Management (CITSM)*, 2024, pp. 1–6. DOI: 10.1109/CITSM64103.2024.10775935.

[16] S. E. Biable, N. M. Garcia, and D. Midekso, "Proposed ethical framework for software requirements engineering," *IET Software*, vol. 17, no. 4, pp. 526–537, Jul. 2023. DOI: 10.1049/sfw2.12136. [Online]. Available: https://doi.org/10.1049/sfw2.12136.

[17] Captain Compliance, *New zealand privacy act 2020 vs gdpr*, https://captaincompliance.com/education/new-zealand-privacy-act-2020-vs-gdpr/, Accessed: 18 June 2025, 2025.

[18] High-Level Expert Group on Artificial Intelligence (AI HLEG), "Ethics guidelines for trustworthy ai," European Commission, Brussels, Belgium, Tech. Rep., Apr. 2019, Document made public on 8 April 2019. [Online]. Available: https://ec.europa.eu/futurium/en/ai-alliance-consultation/guidelines.

[19] F. Kitsios, E. Chatzidimitriou, and M. Kamariotou, "The iso/iec 27001 information security management standard: How to extract value from data in the it sector," *Sustainability*, vol. 15, no. 7, p. 5828, 2023. DOI: 10.3390/su15075828. [Online]. Available: https://doi.org/10.3390/su15075828.

Appendix: Python Code - Jupyter Notebook

Content converted to PDF from the Jupyter Notebook (ipynb).

Exploratory Analyses

In [4]:
```
# Loading main libraries

import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import warnings
warnings.filterwarnings("ignore", message="X does not have valid feature names")
warnings.simplefilter(action='ignore', category=FutureWarning)

# Loading data from csv file
df = pd.read_csv('dataset_for_assignment_2.csv')
```

In [5]:
```
df.head(6)
```

Out[5]:

	User ID	Gender	Age	Activity Level	Location	App Sessions	Distance Travelled (km)	Calories Burned
0	1	Female	22	Active	Suburban	151	140	728
1	2	Male	50	Active	Rural	157	115	523
2	3	Male	36	Active	Urban	178	163	795
3	4	Female	36	Active	Suburban	155	85	715
4	5	Male	34	Sedentary	Rural	95	71	439
5	6	Male	41	Sedentary	Suburban	80	60	197

In [6]:
```
# Checking null values
df.isnull().sum()
```

Out[6]:
```
User ID                   0
Gender                    0
Age                       0
Activity Level            0
Location                  0
App Sessions              0
Distance Travelled (km)   0
Calories Burned           0
dtype: int64
```

In [7]:
```
# Checking Nan values
df.isna().sum()
```

Out[7]:
```
User ID                   0
Gender                    0
Age                       0
Activity Level            0
Location                  0
App Sessions              0
Distance Travelled (km)   0
Calories Burned           0
dtype: int64
```

In [8]:
```
# Deleting column for Id
df = df.drop(columns=['User ID'])
```

In [9]:
```
# some basic statistics - values
df.describe().round(2)
```

Out[9]:

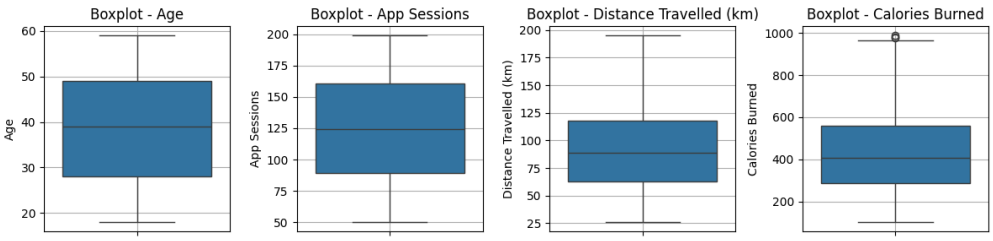
	Age	App Sessions	Distance Travelled (km)	Calories Burned
count	5000.00	5000.00	5000.00	5000.00
mean	38.42	124.37	92.55	432.32
std	12.16	42.69	36.75	187.43
min	18.00	50.00	26.00	102.00
25%	28.00	89.00	63.00	285.00
50%	39.00	124.00	89.00	406.00
75%	49.00	161.00	118.00	560.00
max	59.00	199.00	195.00	987.00

In [10]:
```
# some basic statistics - categories
df.describe(include=['object', 'category'])
```

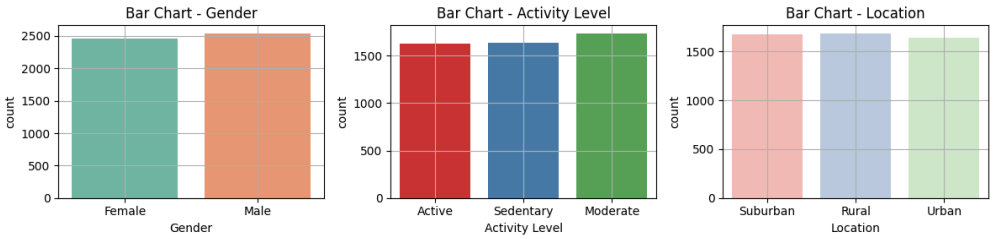
Out[10]:

	Gender	Activity Level	Location
count	5000	5000	5000
unique	2	3	3
top	Male	Moderate	Rural
freq	2538	1734	1684

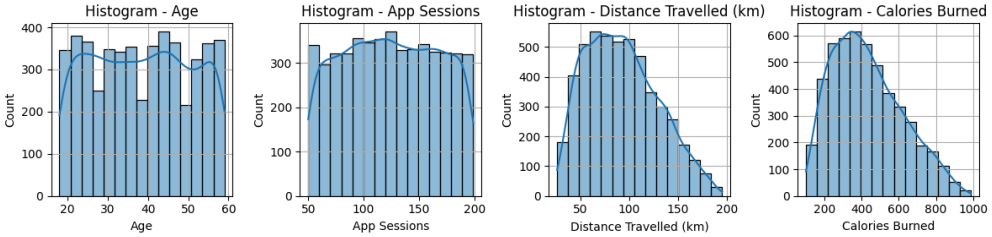
In [11]:
```
# --- SEARCHING FOR OUTLIERS ---

# USING BOXPLOTS
cols_num = ['Age', 'App Sessions', 'Distance Travelled (km)', 'Calories Burned']
fig, axes = plt.subplots(1, 4, figsize=(12, 3))

for i, col in enumerate(cols_num):
    sns.boxplot(y=df[col], ax=axes[i])
    axes[i].set_title(f'Boxplot - {col}')
    axes[i].grid(True)

plt.tight_layout()
plt.show()
```



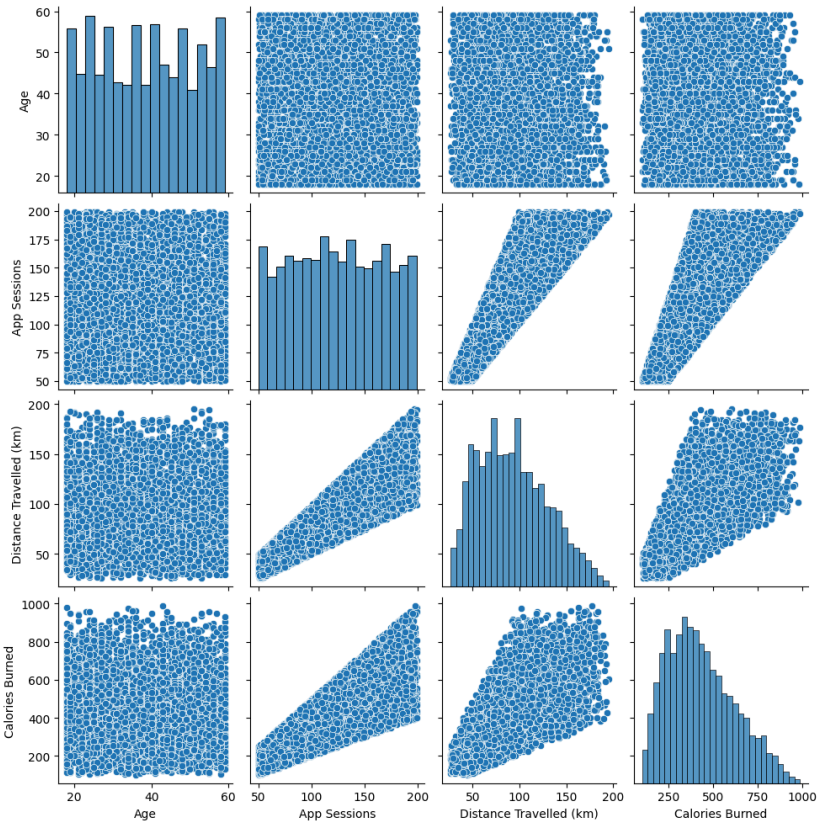
In [12]:
```
# USING BAR CHART FOR CATEGORICAL VARIABLES
import seaborn as sns
import matplotlib.pyplot as plt

cols_categ = ['Gender', 'Activity Level', 'Location']
fig, axes = plt.subplots(1, 3, figsize=(12, 3))

# Paletas conhecidas do Seaborn
palette_list = ['Set2', 'Set1', 'Pastel1']

for i, col in enumerate(cols_categ):
    sns.countplot(x=col, data=df, ax=axes[i], palette=palette_list[i])
    axes[i].set_title(f'Bar Chart - {col}')
    axes[i].grid(True)

plt.tight_layout()
plt.show()
```



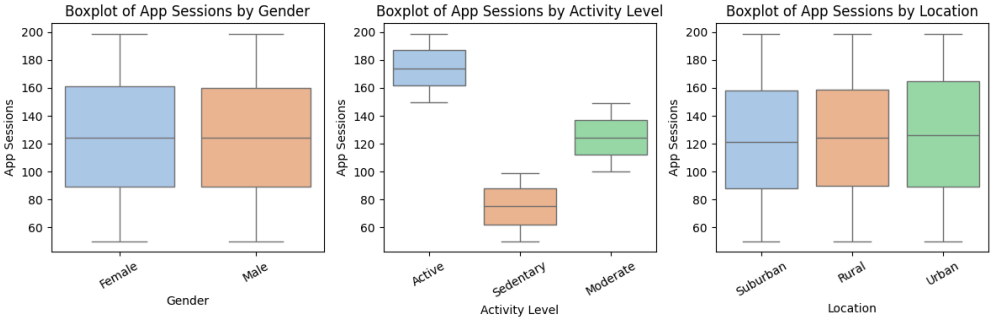
In [13]:
```
# USING HISTOGRAMS

fig, axes = plt.subplots(1, 4, figsize=(12, 3))

cols_num = ['Age', 'App Sessions', 'Distance Travelled (km)', 'Calories Burned']

for i, cols_num in enumerate(cols_num):
    sns.histplot(df[cols_num], kde=True, bins=15, ax=axes[i])
    axes[i].set_title(f'Histogram - {cols_num}')
    axes[i].grid(True)

plt.tight_layout()
plt.show()
```



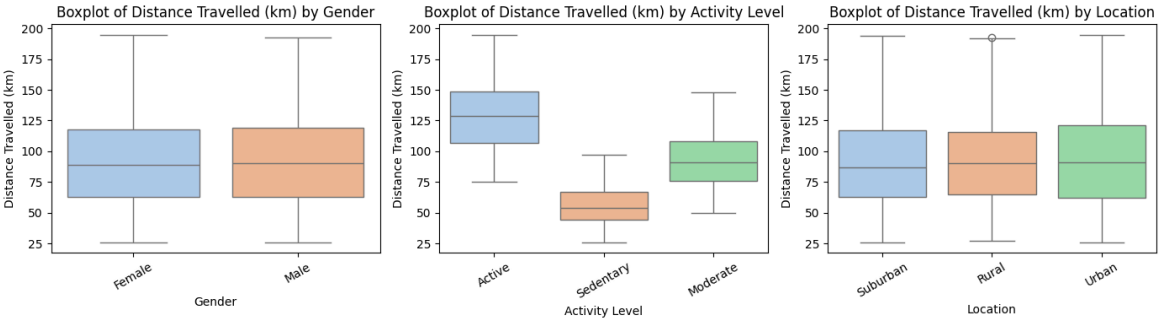
In [14]:
```
sns.pairplot(df)
plt.show()
```



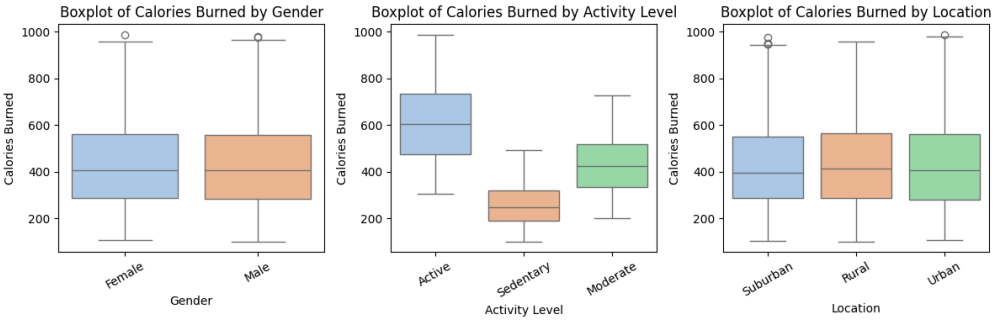
In [15]:
```
# ANALYSING APP SESSIONS BY INDEPENDENT VARIABLES

cols_x = ['Gender', 'Activity Level', 'Location'] # independent variables or inputs
y = 'App Sessions' # dependent variable or output

fig, axes = plt.subplots(1, 3, figsize=(12, 4))  # 1 row x 3 columns

for i, col in enumerate(cols_x):
    sns.boxplot(x=col, y=y, hue=col, data=df, ax=axes[i], palette='pastel', legend=False)
    axes[i].set_title(f'Boxplot of {y} by {col}')
    axes[i].tick_params(axis='x', rotation=30)  # rotate labels

plt.tight_layout()
plt.show()
```



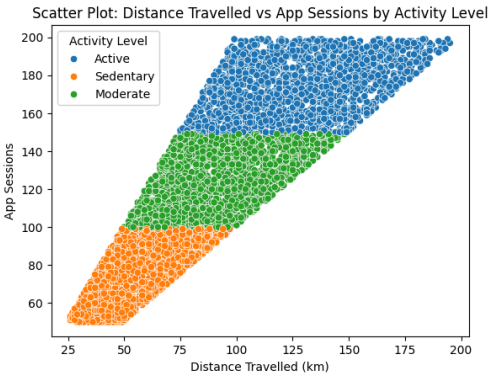
In [16]:
```
cols_x = ['Gender', 'Activity Level', 'Location']
y = 'Distance Travelled (km)'

fig, axes = plt.subplots(1, 3, figsize=(14, 4))  # 1 linha x 3 colunas

for i, col in enumerate(cols_x):
    sns.boxplot(x=col, y=y, hue=col, data=df, ax=axes[i], palette='pastel', legend=False)
    axes[i].set_title(f'Boxplot of {y} by {col}')
    axes[i].tick_params(axis='x', rotation=30)  # girar labels se necessário

plt.tight_layout()
plt.show()
```



In [17]:
```
cols_x = ['Gender', 'Activity Level', 'Location']
y = 'Calories Burned'

fig, axes = plt.subplots(1, 3, figsize=(12, 4))  # 1 linha x 3 colunas

for i, col in enumerate(cols_x):
    sns.boxplot(x=col, y=y, hue=col, data=df, ax=axes[i], palette='pastel', legend=False)
    axes[i].set_title(f'Boxplot of {y} by {col}')
    axes[i].tick_params(axis='x', rotation=30)  # girar labels se necessário

plt.tight_layout()
plt.show()
```



In [18]:
```
sns.scatterplot(data=df, x='Distance Travelled (km)', y='App Sessions', hue='Activity Level')
plt.title('Scatter Plot: Distance Travelled vs App Sessions by Activity Level')
plt.xlabel('Distance Travelled (km)')
plt.ylabel('App Sessions')
plt.show()
```

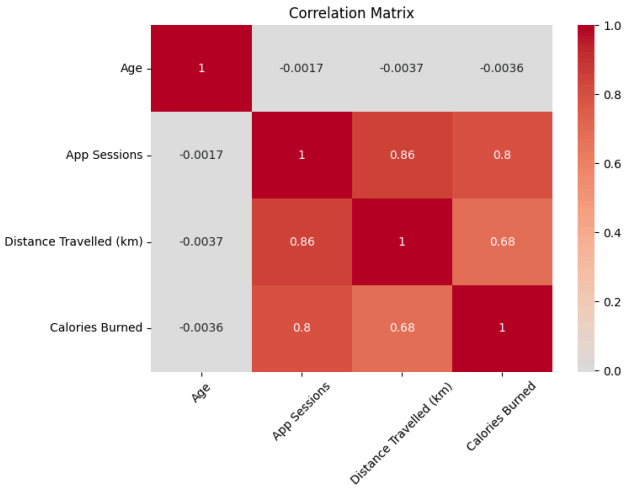


In [19]:
```
sns.scatterplot(data=df, x='Calories Burned', y='App Sessions', hue='Activity Level')
plt.title('Scatter Plot: Calories Burned vs App Sessions by Activity Level')
plt.xlabel('Calories Burned')
plt.ylabel('App Sessions')
plt.show()
```

In [20]:
```
sns.scatterplot(data=df, x='Calories Burned', y='App Sessions', hue='Gender')
plt.title('Scatter Plot: Calories Burned vs App Sessions by Activity Level')
plt.xlabel('Calories Burned')
plt.ylabel('App Sessions')
plt.show()
```

In [21]:
```
sns.scatterplot(data=df, x='Calories Burned', y='App Sessions', hue='Location')
plt.title('Scatter Plot: Calories Burned vs App Sessions by Activity Level')
plt.xlabel('Calories Burned')
plt.ylabel('App Sessions')
plt.show()
```

In [22]:
```
# ANALYSING CORRELATION BEFORE DATA TRANSFORMATION - ONLY NUMERICAL FEATURES

import seaborn as sns
from matplotlib import pyplot as plt

corr = df[['Age', 'App Sessions', 'Distance Travelled (km)', 'Calories Burned']].corr(method='pearson')
# Pearson: Used when variables are continuous, normally distributed, and have a linear relationship. Ideal for parametric data. --> used in this case
# Spearman: Used for monotonic (non-linear) relationships, ordinal variables, or when outliers are present. More robust to non-normality.

plt.figure(figsize=(8,6))
sns.heatmap(corr, annot=True, cmap='coolwarm', center=0)
plt.title('Correlation Matrix')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



Supervisioned Models - Regression

In [23]:
```
# creating a model for knn analyses to predict user behaviour and app usage patterns, like App Sessions, Distance Traveled (km), Calories Burned

from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor

# defining X and y:
X = df[['Age', 'Distance Travelled (km)', 'Calories Burned']]
y = df['App Sessions']

# spliting the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Baseline models without transformations

In [54]:
```
# KNN model
knn_model = KNeighborsRegressor(n_neighbors=5)

# Training the model
knn_model.fit(X_train, y_train)

# prediction
y_pred = knn_model.predict(X_test)

# Evaluating the model
from sklearn.metrics import mean_squared_error, r2_score
mse_knn1 = mean_squared_error(y_test, y_pred)
r2_knn1 = r2_score(y_test, y_pred)
print(f"Mean Squared Error: {mse_knn1}")
print(f"R-squared: {r2_knn1}")
```

```
Mean Squared Error: 136.374
R-squared: 0.9267297173028447
```

In [25]:
```
# do the same but using Random Forest Regressor

from sklearn.ensemble import RandomForestRegressor
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)
y_pred = rf_model.predict(X_test)
mse_rf1 = mean_squared_error(y_test, y_pred)
r2_rf1 = r2_score(y_test, y_pred)
print(f"Mean Squared Error: {mse_rf1}")
print(f"R-squared: {r2_rf1}")
```

```
Mean Squared Error: 309.73590900228686
R-squared: 0.833586771568936
```

In [26]:
```
# do the same but using Linear regression
from sklearn.linear_model import LinearRegression
lr_model = LinearRegression()
lr_model.fit(X_train, y_train)
y_pred = lr_model.predict(X_test)
mse_lr1 = mean_squared_error(y_test, y_pred)
r2_lr1 = r2_score(y_test, y_pred)
print(f"Mean Squared Error: {mse_lr1}")
print(f"R-squared: {r2_lr1}")
```

```
Mean Squared Error: 331.92825117843216
R-squared: 0.8216633903895458
```

In [27]:
```
# make an analysis of Features importance for App Sessions using random forest regressor
# Do Standarization for X2 to fit t the same scale to avoid bias - not needed for y2
df2 = df.copy()

df2['Gender'] = df2['Gender'].map({'Male': 0, 'Female': 1})
df2['Location'] = df2['Location'].map({'Rural': 0, 'Suburban': 1, 'Urban': 2})
df2['Activity Level'] = df2['Activity Level'].map({'Sedentary': 1, 'Moderate': 2, 'Active': 3})

df2.head(2)
```

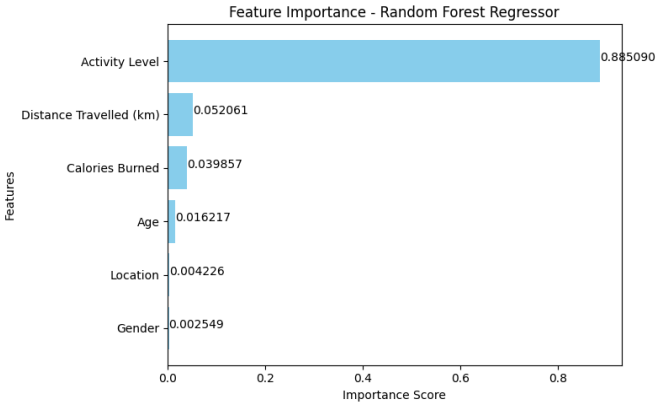
Out[27]:

	Gender	Age	Activity Level	Location	App Sessions	Distance Travelled (km)	Calories Burned
0	1	22	3	1	151	140	728
1	0	50	3	0	157	115	523

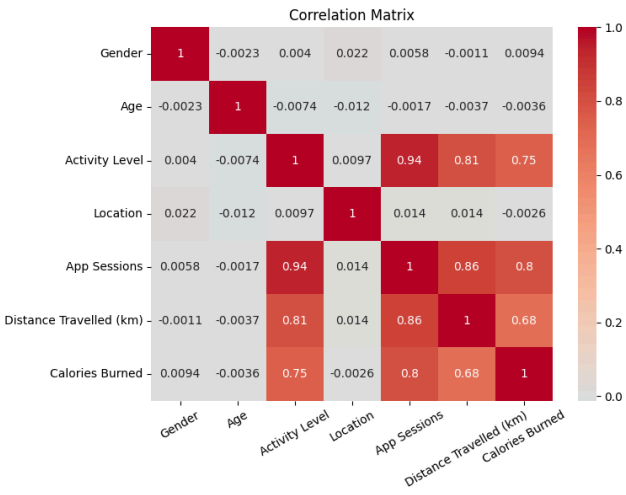
In [ ]:
```
# defining X and y
X = df2.drop(columns=['App Sessions'])
y = df2['App Sessions']
```

In [ ]:
```
# Split the dataset into training and testing sets, with 20% of the data reserved for testing.
# The random_state parameter ensures reproducibility of the split.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create and train the Random Forest Regressor
model = RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Get feature importances
importances = model.feature_importances_
feature_names = X.columns

# Create a DataFrame to display them nicely
feature_importance_df = pd.DataFrame({
    'Feature': feature_names,
    'Importance': importances
}).sort_values(by='Importance', ascending=False)

# Display the table
print(feature_importance_df)

# Optional: plot feature importances
plt.figure(figsize=(8,5))
plt.barh(feature_importance_df['Feature'], feature_importance_df['Importance'], color='skyblue')
plt.gca().invert_yaxis()  # Inverte o eixo Y para mostrar o mais importante no topo

plt.title('Feature Importance - Random Forest Regressor')
plt.xlabel('Importance Score')
plt.ylabel('Features')

for index, value in enumerate(feature_importance_df['Importance']):
    plt.text(value, index, f'{value:.6f}')

plt.tight_layout()
plt.show()
```

```
                   Feature  Importance
2          Activity Level    0.885090
4  Distance Travelled (km)    0.052061
5          Calories Burned    0.039857
1                      Age    0.016217
3                 Location    0.004226
0                   Gender    0.002549
```



In [30]:
```
# ANALYSING CORRELATION AFTER DATA TRANSFORMATION

import seaborn as sns
from matplotlib import pyplot as plt

corr2 = df2[['Gender', 'Age', 'Activity Level', 'Location', 'App Sessions', 'Distance Travelled (km)', 'Calories Burned']].corr(method='pearson')

plt.figure(figsize=(8,6))
sns.heatmap(corr2, annot=True, cmap='coolwarm', center=0)
plt.title('Correlation Matrix')
plt.xticks(rotation=30)
plt.tight_layout()
plt.show()
```



Feature Engineering: Intensity per Session

A new variable was created by combining existing features to capture the intensity of physical activity normalised by app engagement.
The formula is:

$$\text{Intensity per Session} = \frac{\text{Calories Burned}/\text{Distance Travelled (km)}}{\text{App Sessions}}$$

First, the calories per kilometre ratio is calculated, representing the caloric intensity per distance travelled. This value is then normalised by the number of app sessions to reflect usage frequency.

A higher value of this metric suggests a user who undertakes less frequent but more intense or longer sessions.

In [31]:
```
# Calculates the average intensity of calories burned per kilometre per app session.
# This metric normalises the calorie expenditure by distance travelled and number of sessions,
# enabling a fairer comparison of user effort regardless of usage frequency or total distance.

df['Intensity per Session'] = (df['Calories Burned'] / df['Distance Travelled (km)']) / df['App Sessions']
```

In [32]:
```
df2.head(3)
```

Out[32]:

	Gender	Age	Activity Level	Location	App Sessions	Distance Travelled (km)	Calories Burned
0	1	22	3	1	151	140	728
1	0	50	3	0	157	115	523
2	0	36	3	2	178	163	795

In [33]:
```
df.head(3)
```

Out[33]:

	Gender	Age	Activity Level	Location	App Sessions	Distance Travelled (km)	Calories Burned	Intensity per Session
0	Female	22	Active	Suburban	151	140	728	0.034437
1	Male	50	Active	Rural	157	115	523	0.028967
2	Male	36	Active	Urban	178	163	795	0.027401

In [34]:
```
#X2 = df2.drop(columns=['Calories Burned'])
X2 = df2.drop(columns=['App Sessions', 'Gender', 'Age', 'Location']) # drop features less important to simplify the model
y2 = df2['App Sessions']

X2.head(3)
```

Out[34]:

	Activity Level	Distance Travelled (km)	Calories Burned
0	3	140	728
1	3	115	523
2	3	163	795

In [35]:
```
y2.head(3)
```

Out[35]:
```
0    151
1    157
2    178
Name: App Sessions, dtype: int64
```

In [36]:
```
# Do Standarization for X2 to fit t the same scale to avoid bias - not needed for y2
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X2 = scaler.fit_transform(X2)

# checking transformations
X2[0:3]
```

Out[36]:
```
array([[1.23780048, 1.29141193, 1.57774974],
       [1.23780048, 0.61099949, 0.48388367],
       [1.23780048, 1.91739137, 1.93525719]])
```

In [37]:
```
# Do a prediction of Calories Burned using the Random Forest Regressor
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor

X_train, X_test, y_train, y_test = train_test_split(X2, y2, test_size=0.2, random_state=42)
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)
y_pred = rf_model.predict(X_test)
mse_rf2 = mean_squared_error(y_test, y_pred)
r2_rf2 = r2_score(y_test, y_pred)
print(f"Mean Squared Error: {mse_rf2}")
print(f"R-squared: {r2_rf2}")
```
```
Mean Squared Error: 139.26755381773594
R-squared: 0.9251750844092951
```

In [38]:
```
# do the same for KNeighborsRegressor
from sklearn.neighbors import KNeighborsRegressor

X_train, X_test, y_train, y_test = train_test_split(X2, y2, test_size=0.2, random_state=42)
knn_model = KNeighborsRegressor(n_neighbors=5)
knn_model.fit(X_train, y_train)
y_pred = knn_model.predict(X_test)
mse_knn2 = mean_squared_error(y_test, y_pred)
r2_knn2 = r2_score(y_test, y_pred)
print(f"Mean Squared Error: {mse_knn2}")
print(f"R-squared: {r2_knn2}")
```
```
Mean Squared Error: 136.374
R-squared: 0.926729717302844 7
```

In [39]:
```
# now for LinearRegression
from sklearn.linear_model import LinearRegression

X_train, X_test, y_train, y_test = train_test_split(X2, y2, test_size=0.2, random_state=42)
lr_model = LinearRegression()
lr_model.fit(X_train, y_train)
y_pred = lr_model.predict(X_test)
mse_lr2 = mean_squared_error(y_test, y_pred)
r2_lr2 = r2_score(y_test, y_pred)
```

```
print(f"Mean Squared Error: {mse_lr2}")
print(f"R-squared: {r2_lr2}")
```

```
Mean Squared Error: 140.74320170284278
R-squared: 0.9243822563210726
```

In [40]:
```
# now for XGBoost
from xgboost import XGBRegressor

X_train, X_test, y_train, y_test = train_test_split(X2, y2, test_size=0.2, random_state=42)
xgb_model = XGBRegressor()
xgb_model.fit(X_train, y_train)
y_pred = xgb_model.predict(X_test)
mse_xgb2 = mean_squared_error(y_test, y_pred)
r2_xgb2 = r2_score(y_test, y_pred)
print(f"Mean Squared Error: {mse_xgb2}")
print(f"R-squared: {r2_xgb2}")
```

```
Mean Squared Error: 134.0664825439453
R-squared: 0.9279694557189941
```

In [41]:
```
# Dataframe to compare metrics for models

models = ['LinearRegression 1', 'LinearRegression 2',
          'K-NeighborsRegressor 1', 'K-NeighborsRegressor 2',
          'RandomForestRegressor 1', 'RandomForestRegressor 2',
          'XGBoost']
mse_values = [mse_lr1, mse_lr2,
              mse_knn1, mse_knn2,
              mse_rf1, mse_rf2,
              mse_xgb2]
r2_values = [r2_lr1, r2_lr2,
             r2_knn1, r2_knn2,
             r2_rf1, r2_rf2,
             r2_xgb2]

metrics_df = pd.DataFrame({
    'Model': models,
    'Mean Squared Error': mse_values,
    'R-squared': r2_values
})

metrics_df[['Mean Squared Error', 'R-squared']] = metrics_df[['Mean Squared Error', 'R-squared']].round(3)
```

In [42]:
```
metrics_df.sort_values(by="R-squared", ascending=False).reset_index(drop=True)
```

Out[42]:

	Model	Mean Squared Error	R-squared
0	XGBoost	134.066	0.928
1	K-NeighborsRegressor 2	136.374	0.927
2	RandomForestRegressor 2	139.268	0.925
3	LinearRegression 2	140.743	0.924
4	RandomForestRegressor 1	309.736	0.834
5	K-NeighborsRegressor 1	323.314	0.826
6	LinearRegression 1	331.928	0.822

In [43]:
```
# try to improve the results even more

X_train, X_test, y_train, y_test = train_test_split(X2, y2, test_size=0.2, random_state=42)

# using Gridsearch for XGBoost, KNeighborsRegressor and Random Forest Regressor

# XGBoost
from xgboost import XGBRegressor
from sklearn.model_selection import GridSearchCV

param_grid = {
    'n_estimators': [50, 100, 150],
    'learning_rate': [0.1, 0.01, 0.001],
    'max_depth': [3, 5, 7],
    'min_child_weight': [1, 2, 3],
    'gamma': [0, 0.1, 0.2],
    'subsample': [0.8, 1.0],
    'colsample_bytree': [0.7, 1.0]
}

xgb_model_opt = XGBRegressor()
grid_search_xgb = GridSearchCV(xgb_model_opt, param_grid, cv=5, scoring='neg_mean_squared_error')
grid_search_xgb.fit(X_train, y_train)

best_params = grid_search_xgb.best_params_
best_model = XGBRegressor(**best_params)
best_model.fit(X_train, y_train)
y_pred = best_model.predict(X_test)
mse_xgb3 = mean_squared_error(y_test, y_pred)
r2_xgb3 = r2_score(y_test, y_pred)
print(f"XGBRegressor_optimized")
print(f"Mean Squared Error: {mse_xgb3}")
print(f"R-squared: {r2_xgb3}")
print(best_params)
```

```
XGBRegressor_optimized
Mean Squared Error: 115.50733947753906
R-squared: 0.9379408359527588
{'colsample_bytree': 1.0, 'gamma': 0.1, 'learning_rate': 0.1, 'max_depth': 5, 'min_child_weight': 1, 'n_estimators': 50, 'subsample': 1.0}
```

In [44]:
```
# KNeighborsRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import GridSearchCV

param_grid = {
    'n_neighbors': [3, 5, 7],
    'weights': ['uniform', 'distance'],
    'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
    'leaf_size': [10, 20, 30],
    'p': [1, 2, 3]
}

knn_model_opt = KNeighborsRegressor()
grid_search_knn = GridSearchCV(knn_model_opt, param_grid, cv=5, scoring='neg_mean_squared_error')
grid_search_knn.fit(X_train, y_train)

best_params = grid_search_knn.best_params_
best_model = KNeighborsRegressor(**best_params)
best_model.fit(X_train, y_train)
y_pred = best_model.predict(X_test)
mse_knn3 = mean_squared_error(y_test, y_pred)
r2_knn3 = r2_score(y_test, y_pred)
print(f"KNeighborsRegressor_optimized")
print(f"Mean Squared Error: {mse_knn3}")
print(f"R-squared: {r2_knn3}")
print(best_params)
```

```
KNeighborsRegressor_optimized
Mean Squared Error: 133.10414285714288
R-squared: 0.928486528404935
{'algorithm': 'ball_tree', 'leaf_size': 30, 'n_neighbors': 7, 'p': 1, 'weights': 'uniform'}
```

In [56]:
```
# Random Forest Regressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV
import warnings
from sklearn.exceptions import FitFailedWarning

import warnings
warnings.filterwarnings("ignore", message="X does not have valid feature names")
warnings.simplefilter(action='ignore', category=FutureWarning)
warnings.filterwarnings("ignore", category=FitFailedWarning)
warnings.filterwarnings("ignore", category=UserWarning)


param_grid = {
    'n_estimators': [50, 100, 150],
    'max_depth': [None, 5, 10],
    'min_samples_split': [2, 5, 10],
```

```
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['auto', 'sqrt', 'log2']
}

rf_model_opt = RandomForestRegressor()
grid_search_rf = GridSearchCV(rf_model_opt, param_grid, cv=5, scoring='neg_mean_squared_error')
grid_search_rf.fit(X_train, y_train)

best_params = grid_search_rf.best_params_
best_model = RandomForestRegressor(**best_params)
best_model.fit(X_train, y_train)
y_pred = best_model.predict(X_test)
mse_rf3 = mean_squared_error(y_test, y_pred)
r2_rf3 = r2_score(y_test, y_pred)
print(f"RandomForestRegressor_optimized")
print(f"Mean Squared Error: {mse_rf3}")
print(f"R-squared: {r2_rf3}")
print(best_params)
```

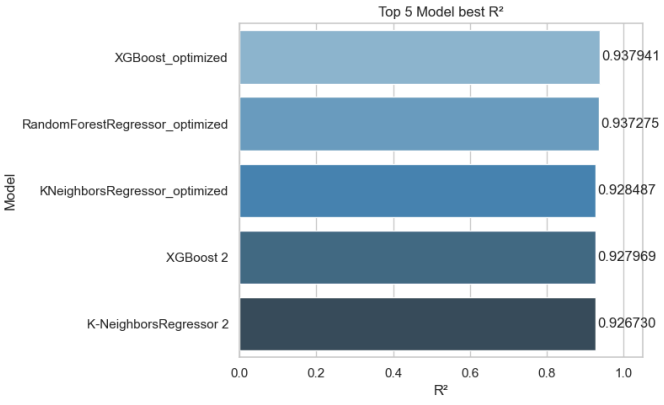
```
RandomForestRegressor_optimized
Mean Squared Error: 117.13110739788549
R-squared: 0.9370684342200571
{'max_depth': 10, 'max_features': 'log2', 'min_samples_leaf': 4, 'min_samples_split': 5, 'n_estimators': 100}
```

In [46]:
```
# Dataframe to compare metrics for models

models = ['LinearRegression 1', 'LinearRegression 2',
          'K-NeighborsRegressor 1', 'K-NeighborsRegressor 2',
          'RandomForestRegressor 1', 'RandomForestRegressor 2',
          'XGBoost 2', 'RandomForestRegressor_optimized',
          'KNeighborsRegressor_optimized', 'XGBoost_optimized']
mse_values = [mse_lr1, mse_lr2,
              mse_knn1, mse_knn2,
              mse_rf1, mse_rf2,
              mse_xgb2, mse_rf3,
              mse_knn3, mse_xgb3]
r2_values = [r2_lr1, r2_lr2,
             r2_knn1, r2_knn2,
             r2_rf1, r2_rf2,
             r2_xgb2, r2_rf3,
             r2_knn3, r2_xgb3]

metrics_df = pd.DataFrame({
    'Model': models,
    'Mean Squared Error': mse_values,
    'R-squared': r2_values
})

metrics_df[['Mean Squared Error', 'R-squared']] = metrics_df[['Mean Squared Error', 'R-squared']].round(6)
all_metrics = metrics_df.sort_values(by="R-squared", ascending=False).reset_index(drop=True)
all_metrics
```

Out[46]:

	Model	Mean Squared Error	R-squared
0	XGBoost_optimized	115.507339	0.937941
1	RandomForestRegressor_optimized	116.747238	0.937275
2	KNeighborsRegressor_optimized	133.104143	0.928487
3	XGBoost 2	134.066483	0.927969
4	K-NeighborsRegressor 2	136.374000	0.926730
5	RandomForestRegressor 2	139.267554	0.925175
6	LinearRegression 2	140.743202	0.924382
7	RandomForestRegressor 1	309.735909	0.833587
8	K-NeighborsRegressor 1	323.313800	0.826292
9	LinearRegression 1	331.928251	0.821663

In [47]:
```
# Graphs with best 5 moldels

top5 = all_metrics.head(5)

sns.set(style="whitegrid")

# Gráfico de barras horizontais
plt.figure(figsize=(8, 5))
sns.barplot(x='R-squared', y='Model', data=top5, palette='Blues_d')

# Adicionar rótulos
for index, value in enumerate(top5['R-squared']):
    plt.text(value + 0.005, index, f"{value:.6f}", va='center')

plt.title("Top 5 Model best R²")
plt.xlabel("R²")
plt.ylabel("Model")
plt.xlim(0, 1.05)
plt.tight_layout()
plt.show()
```



Unsupervisioned Models - Clustering

In [48]:
```
df.head(3)

df3 = df.copy()
df4 = df.copy()

df4['Gender'] = df4['Gender'].map({'Male': 0, 'Female': 1})
df4['Location'] = df4['Location'].map({'Rural': 0, 'Suburban': 1, 'Urban': 2})
df4['Activity Level'] = df4['Activity Level'].map({'Sedentary': 1, 'Moderate': 2, 'Active': 3})

df4.head(2)
```

Out[48]:

	Gender	Age	Activity Level	Location	App Sessions	Distance Travelled (km)	Calories Burned	Intensity per Session
0	1	22	3	1	151	140	728	0.034437
1	0	50	3	0	157	115	523	0.028967

In [49]:
```
df3.head(3)
```

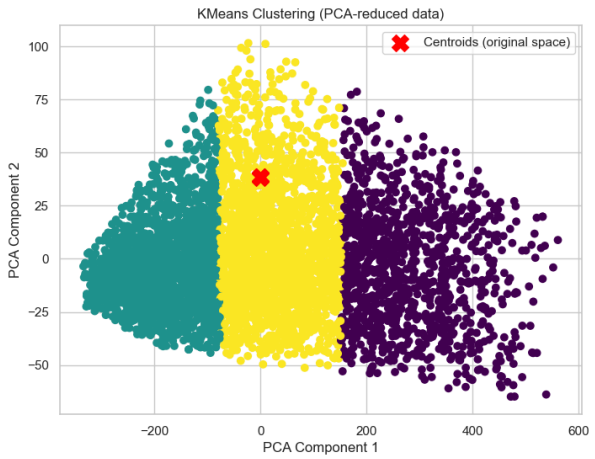
Out[49]:

	Gender	Age	Activity Level	Location	App Sessions	Distance Travelled (km)	Calories Burned	Intensity per Session
0	Female	22	Active	Suburban	151	140	728	0.034437
1	Male	50	Active	Rural	157	115	523	0.028967
2	Male	36	Active	Urban	178	163	795	0.027401

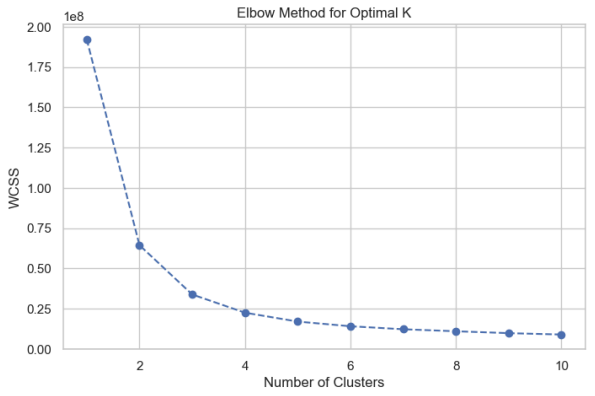
In [58]:
```
X_c = df4[['Gender', 'Age', 'Activity Level', 'Location', 'Distance Travelled (km)', 'Calories Burned', 'Intensity per Session']]
y_c = df4['App Sessions']
```

In [60]:
```
# use models for clustering

from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=3)
kmeans.fit(X_c)
y_pred_c = kmeans.predict(X_c)
df['Cluster'] = y_pred_c
```

In [61]:
```
# Visualize the clusters
from sklearn.decomposition import PCA

pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_c)

plt.figure(figsize=(8, 6))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y_pred_c, cmap='viridis')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
            s=200, c='red', marker='X', label='Centroids (original space)')
plt.title("KMeans Clustering (PCA-reduced data)")
plt.xlabel("PCA Component 1")
plt.ylabel("PCA Component 2")
plt.legend()
plt.show()
```



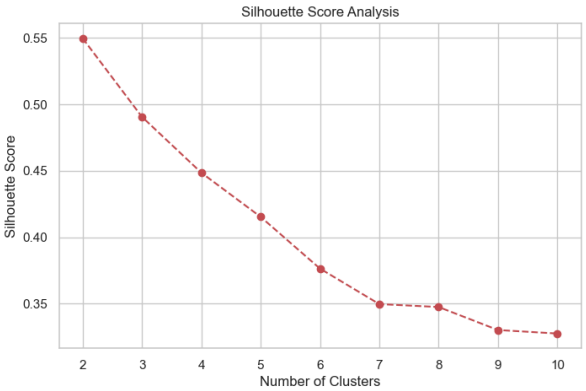
In [62]:
```
wcss = []
K = range(1, 11)

for k in K:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(df4)
    wcss.append(kmeans.inertia_)

# Plot the Elbow Curve
plt.figure(figsize=(8, 5))
plt.plot(K, wcss, marker="o", linestyle="--")
plt.xlabel("Number of Clusters")
plt.ylabel("WCSS")
plt.title("Elbow Method for Optimal K")
plt.show()
```



In [ ]:
```
from sklearn.metrics import silhouette_score

score = silhouette_score(X_c, y_pred_c)
print(f"Silhouette Score: {score:.2f}")  # Closer to 1 better
```

```
Silhouette Score: 0.51
```

In [67]:
```
silhouette_scores = []

for k in range(2, 11):
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    labels = kmeans.fit_predict(df4)
    silhouette_scores.append(silhouette_score(df4, labels))

# Plot Silhouette Scores
plt.figure(figsize=(8, 5))
plt.plot(range(2, 11), silhouette_scores, marker="o", linestyle="--", color='r')
plt.xlabel("Number of Clusters")
plt.ylabel("Silhouette Score")
plt.title("Silhouette Score Analysis")
plt.show()
```



From the graphs:

Elbow Method (WCSS):

The "elbow" point appears at K = 3 ou k = 4, where the rate of decrease in WCSS slows down significantly. This indicates diminishing returns in improving clustering by adding more clusters after 5.

Silhouette Score:

The highest silhouette score is observed at K = 2, which suggests that the clustering quality is optimal when there are 2 clusters, then decrease.

Since K = 3 or 4 is the elbow point and the silhouette score at K = 2 is the maximum, It is recommend starting with K = 2 or 3 for simplicity and interpretability.

In [70]:
```
# Applying K-Means Clustering

optimal_k = 3
kmeans = KMeans(n_clusters=optimal_k, random_state=42, n_init=10)
df4['Cluster'] = kmeans.fit_predict(df4)
```

In [72]:
```
# Visualizing Clusters Characteristics

# Gender        Age     Activity Level  Location        App Sessions    Distance Travelled (km) Calories Burned
numeric_columns = df4.select_dtypes(include=['number'])

# Group and calculate mean
cluster_summary = numeric_columns.groupby('Cluster').mean()

# Show results in a dataframe
cluster_summary
```

Out[72]:

Cluster	Gender	Age	Activity Level	Location	App Sessions	Distance Travelled (km)	Calories Burned	Intensity per Session
0	0.486596	38.604451	2.242286	0.976732	137.494689	102.899848	454.188164	0.038278
1	0.495455	38.060000	2.810909	1.011818	169.677273	125.709091	710.659091	0.035651
2	0.496620	38.445658	1.286011	0.993240	84.950598	62.942798	250.611024	0.056736

In [79]:
```
# 1. Numerical variables: mean per cluster
numeric_columns = df.select_dtypes(include=['number'])
cluster_summary_numeric = numeric_columns.groupby('Cluster').mean()

# 2. Categorical variables: mode per cluster
categorical_columns = df.select_dtypes(include=['object', 'category'])

# Create dictionary to store mode of each category per cluster
cluster_modes = {}

for col in categorical_columns.columns:
    mode_per_cluster = df.groupby('Cluster')[col].agg(lambda x: x.mode().iloc[0] if not x.mode().empty else None)
    cluster_modes[col] = mode_per_cluster

# 3. Convert dictionary to DataFrame
cluster_summary_categorical = pd.DataFrame(cluster_modes)

# 4. Combine both summaries (numerical + categorical)
cluster_summary = pd.concat([cluster_summary_numeric, cluster_summary_categorical], axis=1)

# 5. Display result
cluster_summary
```

Out[79]:

Cluster	Age	App Sessions	Distance Travelled (km)	Calories Burned	Intensity per Session	Gender	Activity Level	Location
0	38.604451	137.494689	102.899848	454.188164	0.038278	Male	Moderate	Rural
1	38.060000	169.677273	125.709091	710.659091	0.035651	Male	Active	Urban
2	38.445658	84.950598	62.942798	250.611024	0.056736	Male	Sedentary	Suburban

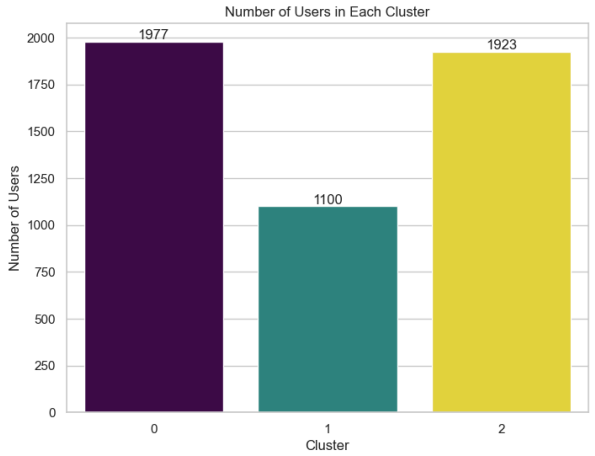
In [81]:
```
# Create dictionary to store mode and frequency percentage
categorical_modes_freq = {}

for col in categorical_columns.columns:
    mode_info = df.groupby('Cluster')[col].agg(
        lambda x: pd.Series({
            'Mode': x.mode().iloc[0] if not x.mode().empty else None,
            'Frequency (%)': round(100 * (x.value_counts(normalize=True).iloc[0]), 1)
        })
    )
    categorical_modes_freq[col] = mode_info

# Concatenate results from all categorical columns
categorical_summary = pd.concat(categorical_modes_freq, axis=1)

# Adjust multi-index (column → row)
# categorical_summary.columns = [''.join(col).strip() for col in categorical_summary.columns.values]
categorical_summary.columns = [''.join(col).strip().replace(' ', '_') for col in categorical_summary.columns.values]

categorical_summary.reset_index(inplace=True)

# Combine numerical and categorical summaries
cluster_summary = pd.concat([cluster_summary_numeric, categorical_summary], axis=1)

# Display
cols = ['Cluster'] + [col for col in cluster_summary.columns if col != 'Cluster']
cluster_summary = cluster_summary[cols]
cluster_summary
```

Out[81]:

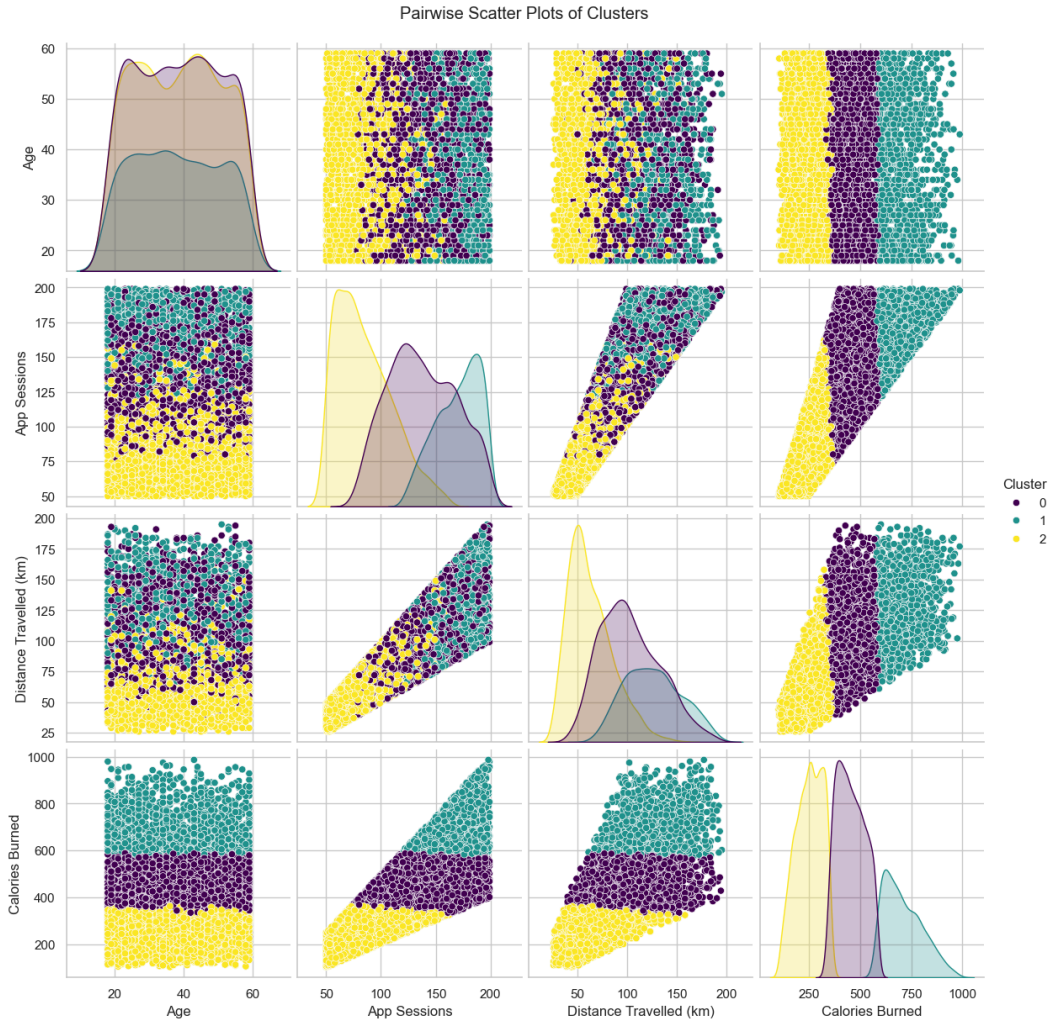
	Cluster	Age	App Sessions	Distance Travelled (km)	Calories Burned	Intensity per Session	Gender	Activity_Level	Location
0	0	38.604451	137.494689	102.899848	454.188164	0.038278	[Male, 51.3]	[Moderate, 52.3]	[Rural, 34.5]
1	1	38.060000	169.677273	125.709091	710.659091	0.035651	[Male, 50.5]	[Active, 81.1]	[Urban, 34.6]
2	2	38.445658	84.950598	62.942798	250.611024	0.056736	[Male, 50.3]	[Sedentary, 72.9]	[Suburban, 34.8]

In [83]:
```
import seaborn as sns

# Calculate cluster counts
cluster_counts = df['Cluster'].value_counts().sort_index()

# Create bar plot with Seaborn
plt.figure(figsize=(8, 6))  # Set figure size before plotting
ax = sns.barplot(x=cluster_counts.index, y=cluster_counts.values, palette='viridis', hue=cluster_counts.index, legend=False)
```

```
plt.title("Number of Users in Each Cluster")
plt.xlabel("Cluster")
plt.ylabel("Number of Users")
plt.xticks(rotation=0)

# Add value labels on each bar
for container in ax.containers:
    ax.bar_label(container, fmt='%d')

plt.show()
```



In [85]:
```
sns.pairplot(
    df[['Gender', 'Age', 'Activity Level', 'App Sessions', 'Distance Travelled (km)', 'Calories Burned', 'Location', 'Cluster']],
    hue='Cluster',
    palette='viridis',
    diag_kind='kde',
    height=3
)

plt.suptitle("Pairwise Scatter Plots of Clusters", y=1.02)
plt.show()
```



Cluster Analysis

All clusters have relatively similar age distributions, so age is not a primary driver of app engagement.

Cluster 0: Moderately Active

Average Age: 38.6 years
App Sessions: 137
Average Distance Travelled: 102.9 km
Calories Burned: 454 kcal
Predominant Gender: Male (51.3%)
Activity Level: Moderate (52.3%)
Location: Rural (34.5%)

Interpretation: This group consists of users who are moderately active and live mostly in rural areas. Their level of app usage suggests a steady, consistent engagement with physical activity.

Suggested Features: Personalised progress plans and milestone tracking. These users show steady engagement and a moderate level of physical activity. To support them, the app should offer custom weekly plans that gradually increase in intensity. Additionally, features like milestone tracking (e.g., "100 km total walked"), and gentle performance tips can help maintain motivation and foster progression without overwhelming the user. Provide personalised plans that gradually increase in difficulty.

Add in-app reminders and progress tracking to nudge them towards higher engagement. Use milestone rewards (e.g., badges for 100km walked or 30 active days).

Cluster 1: Highly Active

Average Age: 38.1 years
App Sessions: 170
Average Distance Travelled: 125.7 km
Calories Burned: 711 kcal
Predominant Gender: Male (50.5%)
Activity Level: Active (81.1%)
Location: Urban (34.6%)

Interpretation: These are the most physically active users. With the highest number of app sessions, greater distance covered, and significantly more calories burned, they appear to be performance-driven and motivated by measurable outcomes.

Suggested Features: Advanced analytics and wearable device integration. This segment is likely to appreciate detailed performance metrics, weekly reports, leaderboards, and training suggestions based on their activity data. Introduce advanced performance metrics (VO2 max, pace, progression tracking). Offer integration with wearables (e.g., smartwatches, heart rate monitors). Enable social/community features like leaderboards, challenges, or shared goals.

Cluster 2: Sedentary

Average Age: 38.4 years
App Sessions: 85
Average Distance Travelled: 62.9 km
Calories Burned: 251 kcal
Predominant Gender: Male (50.3%)
Activity Level: Sedentary (72.9%)
Location: Suburban (34.8%)

Interpretation: This group has the lowest activity levels, both in terms of physical performance and app usage. The sedentary lifestyle suggests a need for greater encouragement and onboarding support.

Suggested Features: Gamified challenges and motivational reminders. These users would benefit from light, accessible incentives, such as daily step goals, achievement badges, and friendly nudges to increase engagement and form healthy habits. Gamify the onboarding process (e.g., streaks, daily mini-goals). Offer "starter" fitness paths — very low barrier exercises with visual encouragement. Include push notifications and weekly summaries to maintain awareness.

Implications for Software Engineering Decisions Feature Prioritisation: Cluster insights can guide which features to build or improve. For example:

Advanced metrics for active users (Cluster 1) Motivational elements for low users (Cluster 2)

Personalisation & Targeting: Enable user-tailored experiences based on cluster membership → more efficient use of dev resources.

Retention Strategy: Predictive models can trigger automated retention flows, reducing user churn.

Hierarchical Clustering (Agglomerative)

Este método constrói clusters de baixo para cima, unindo os mais semelhantes em etapas sucessivas.

In [86]:
```
from sklearn.cluster import AgglomerativeClustering

# Aplicar Clustering Hierdrquico
agglo = AgglomerativeClustering(n_clusters=3)  # ou outro número desejado
df4['Cluster_Hierarchical'] = agglo.fit_predict(df4)
df4
```

Out[86]:

	Gender	Age	Activity Level	Location	App Sessions	Distance Travelled (km)	Calories Burned	Intensity per Session	Cluster	Cluster_Hierarchical
0	1	22	3	1	151	140	728	0.034437	1	1
1	0	50	3	0	157	115	523	0.028967	0	2
2	0	36	3	2	178	163	795	0.027401	1	1
3	1	36	3	1	155	85	715	0.054269	1	1
4	0	34	1	0	95	71	439	0.065085	0	2
...	...	...	...	...	...	...	...	...	...	...
4995	1	50	3	2	155	151	502	0.021448	0	2
4996	1	37	2	0	127	84	586	0.054931	0	2
4997	1	47	2	1	140	116	649	0.039963	1	1
4998	1	59	1	1	85	63	301	0.056209	2	0
4999	1	43	2	2	143	111	540	0.034020	0	2

5000 rows × 10 columns

In [87]:
```
from scipy.cluster.hierarchy import dendrogram, linkage
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import fcluster

# Generate the linkage matrix
Z = linkage(df4, method='ward')  # 'ward' is the default used by AgglomerativeClustering

cut_height = 7000

# Plot the dendrogram
plt.figure(figsize=(12, 6))
dendrogram(Z, color_threshold=cut_height, above_threshold_color='gray')
plt.axhline(y=cut_height, color='blue', linestyle='--', label=f'Cut at {cut_height}')  # ← cut line
plt.title("Dendrogram - Hierarchical Clustering")
plt.xlabel("Data Points")
plt.ylabel("Euclidean Distance")
plt.show()
```