University of Brighton

School of Architecture, Technology and Engineering

REAL IMPACT OF DWELL TIMES BETWEEN CASH, TOTO, MOBILE APP AND KEYCARD – B&H BUSES

Ashraf Uddin Tafadar

May 2024

I declare that no part of the work in this report has been submitted in support of an application for another degree or qualification at this or any other institute of learning.

Ashraf Uddin Tafadar

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my advisor, Sonia Timoteo Inacio, for the invaluable guidance, unwavering support, and profound expertise throughout this research journey. Her insightful feedback and encouragement have been instrumental in shaping this dissertation. I am also thankful to Chloe Ware for her assistance with data collection, analysis, and constructive discussions. Her contributions have enriched the depth of this work.

My heartfelt appreciation extends to my family for their unconditional love, patience, and belief in my abilities. Their encouragement sustained me during challenging times, and I am profoundly grateful.

This dissertation would not have been possible without the support and encouragement of these remarkable individuals, and for that, I am profoundly grateful.

ABSTRACT

This study investigates into the real impact of dwell times between Cash, TOTO, Keycard and Mobile App within the Route 7 bus service operating between George Street, Hove, and Brighton Marina. Leveraging transactional data collected from the E.P. Morris database and the Ticketer reporting system, the research aims to identify factors influencing dwell times, particularly focusing on payment methods and ticket types. Through descriptive statistics and exploratory data analysis, the study uncovers trends in passenger transactions, ticket popularity, and dwell times across different stops along the route. Additionally, deterministic models like polynomial regression shed light on the impact of specific variables on dwell times. Furthermore, machine learning models, including Linear Regression, K-Nearest Neighbors, XGBoost, and Random Forest Regression, are evaluated to predict dwell times more accurately. Following this analysis, cross-validation will be applied to enhance model performance and ensure stability. Findings from this study contribute to the optimization of bus operations by identifying areas for improvement and potential strategies to enhance passenger experience and efficient bus service. The dissertation concludes with recommendations for operational enhancements, future research directions, and the importance of data-driven decision-making in public transportation management.

Supervisor: **Sonia Timoteo Inacio**

CONTENTS

INTRODUCTION

BACKGROUND AND CONTEXT

Efficient public transportation systems are fundamental to the mobility and connectivity of urban areas. Among the essential factors contributing to the success of such systems is the optimization of dwell times—the time buses spend at stops for passenger boarding and alighting. Dwell times not only impact the operational efficiency of bus services but also significantly influence passenger satisfaction, overall service quality, and even environmental sustainability. Understanding the factors affecting dwell times is crucial for transit agencies and policymakers aiming to enhance the effectiveness and reliability of public transportation networks.

RESEARCH OBJECTIVES

This study aims to investigate the real impact of dwell times associated with cash, TOTO, mobile app, and Keycard payments on the Route 7 bus service, which operates between George Street, Hove, and Brighton Marina. The primary objectives include analysing the influence of various payment methods and ticket types on dwell times, identifying factors contributing to passenger boarding efficiency, and exploring patterns for optimizing bus operations to enhance service quality and passenger experience. Through this investigation, hidden patterns within the data will be uncovered, and a predictive model will be developed to fit the data, allowing for deeper insights and informed decision-making.

OVERVIEW OF DATASET

The dataset utilized in this study comprises transactional data collected from the E.P. Morris database ,Ticketer reporting system from ticketer itself and GPS data. Spanning the period from the 4th to the 10th of June 2023, the dataset encompasses a comprehensive record of bus operations, ticket sales, passenger counts, and other relevant metrics for the Route 7 bus service. Key variables include ticket types, payment methods, passenger demographics, dwell times, punctuality metrics, and operational parameters. By leveraging this dataset, the study aims to conduct a detailed analysis of factors influencing dwell times and passenger boarding efficiency.

SIGNIFICANCE OF THE STUDY

This research holds significant implications for transit agencies, urban planners, and policymakers seeking to enhance the efficiency, reliability, and sustainability of public transportation systems. By shedding light on the factors affecting dwell times and passenger boarding efficiency, the study contributes to the development of evidence-based strategies for optimizing bus operations and improving overall service quality. Ultimately, the findings of this research can inform decision-making processes aimed at enhancing urban mobility, reducing congestion, and promoting sustainable transportation solutions.

STRUCTURE OF THE DISSERTATION

The dissertation is structured into several sections, each focusing on different aspects of the research topic. Following this introduction, the subsequent sections will include a comprehensive literature review, detailing existing research and theoretical frameworks relevant to the study. This will be followed by a detailed methodology section outlining the data collection and preprocessing, descriptive statistics, exploratory analysis, and machine learning modelling techniques employed in the study. The main findings and analysis of the study will then be presented, followed by a discussion of the implications of the findings for theory, practice, and future research in conclusion. The dissertation will conclude with a summary of key findings, limitations of the study, and recommendations for further research and practice.

LITERATURE REVIEW

Efficient public transportation systems are essential components of urban infrastructure, facilitating mobility, reducing traffic congestion, and mitigating environmental impacts (Cats et al., 2016). Within the realm of public transportation, optimizing dwell times at bus stops is crucial for improving service reliability, enhancing passenger experience, and maximizing operational efficiency. This section provides a critical review of existing literature on factors influencing dwell times in bus services, payment methods, ticketing systems, and their implications for service quality and passenger satisfaction.

Factors Affecting Dwell Times: Research on dwell times in public transportation has identified several factors that influence the duration buses spend at stops (Schmöcker & Quddus, 2010). One of the primary determinants is passenger boarding and alighting behaviour, which can be influenced by factors such as passenger demographics, route characteristics, and bus stop design. For example, studies have found that dwell times tend to be longer at stops with higher passenger volumes or complex boarding processes, such as fare collection or ticket validation.

Payment Methods and Ticketing Systems: The adoption of digital payment methods and innovative ticketing systems has emerged as a promising strategy for reducing dwell times and improving bus service efficiency (Peters & Strathman, 2018). Mobile ticketing apps, contactless payment systems, and smart card technologies offer passengers convenient and efficient ways to pay for their fares, potentially reducing transaction times and boarding delays. Several studies have demonstrated the benefits of these technologies in reducing dwell times and enhancing passenger satisfaction, particularly in high-demand urban environments (Ding et al., 2020).

Impact on Service Quality and Passenger Satisfaction: Efficient dwell times not only contribute to operational efficiency but also play a critical role in shaping passenger perceptions of service quality and satisfaction (Wei & Young, 2019). Long dwell times can lead to overcrowding, delays, and increased perceived wait times, negatively impacting passenger experience and overall service reliability. Conversely, shorter dwell times resulting from streamlined payment processes and boarding procedures can enhance passenger satisfaction, encourage modal shift from private vehicles to public transit, and ultimately contribute to sustainable urban mobility.

Challenges and Opportunities: Despite the potential benefits, the widespread adoption of digital payment methods and innovative ticketing systems in public transportation faces several challenges (Litman, 2019). These include technological barriers, interoperability issues, cost considerations, and equity concerns related to access and affordability. Addressing these challenges requires collaboration among transit agencies, technology providers, policymakers, and stakeholders to develop integrated and user-friendly solutions that prioritize accessibility, equity, and convenience for all passengers.

Conclusion: The literature reviewed underscores the importance of optimizing dwell times in bus services to improve service quality, enhance passenger satisfaction, and promote sustainable urban mobility. While digital payment methods and innovative ticketing systems offer promising solutions for reducing dwell times and enhancing operational efficiency, addressing technological, financial, and equity challenges remains critical for their widespread adoption. Future research should focus on evaluating the effectiveness of different payment methods and ticketing systems in diverse urban contexts, exploring their impacts on service performance, passenger behaviour, and overall transit ridership.

METHODOLOGY

APPROACH

The approach for this project involves a comprehensive analysis of the dataset to understand the dynamics of dwell times and factors influencing passenger boarding efficiency in the Route 7 bus service. The analysis will be structured into several key sections, each serving a distinct purpose in uncovering insights and patterns within the data.

Descriptive Statistics: The initial step in the analysis will involve conducting descriptive statistics to summarize the main characteristics of the dataset. This will include measures such as mean, median, standard deviation, and range for relevant variables such as dwell times, passenger counts, and ticket types. Descriptive statistics will provide an overview of the data distribution and highlight any notable trends or outliers.

Exploratory Data Analysis (EDA): Following descriptive statistics, exploratory data analysis (EDA) will be conducted to delve deeper into the relationships and patterns within the dataset. EDA techniques such as data visualization, correlation analysis, and distribution plots will be employed to identify potential associations between variables and uncover insights into the underlying factors affecting dwell times.

Modelling: The modelling phase will involve developing statistical and machine learning models to predict dwell times and passenger boarding efficiency based on various factors such as payment methods, ticket types, and operational parameters. Three main modelling approaches will be explored:

Deterministic Modelling: Polynomial regression will be employed to analyse the likelihood of certain events occurring, such dwell time, based on the observed data. Polynomial regression will be utilized to model the relationship between predictor variables and the response variable in a deterministic manner, capturing nonlinearities and complex patterns in the data.

Artificial Intelligence Modelling: Advanced machine learning algorithms, including decision trees, random forests, and neural networks, will be employed to develop predictive models capable of capturing complex relationships and nonlinearities in the data. These models will be trained and evaluated using cross-validation techniques to ensure robust performance.

Cross-Validation: Cross-validation techniques will be utilized to assess the performance and universality of the developed models. This involves splitting the dataset into training and testing subsets multiple times and evaluating the model's performance on each iteration. By assessing the model's performance across multiple validation folds, we can obtain more reliable estimates of its predictive capabilities.

DESCRIPTIVE STATISTICS

As depicted in Appendix Code A1, the initial stage involves the utilization of ticket type data from E.P. Morris, which closely mirrors the Ticketer data, along with GPS data for descriptive and exploratory purposes. Subsequently, Ticketer data and ticket type data sourced from E.P. Morris will be merged using an inner join to combine them with GPS data for further analysis during the modelling phase. The rationale behind merging them together lies in the fact that Ticketer data and GPS data contain a 'Bus number' column, which is absent from the E.P. Morris data.

Ticket data: Most of the variables in ticket data are self-explanatory.

- ETM Route – Route Number (7)
- Bus Stop Name – Stop Name
- Op Date – Date of operation
- Transaction Time – Time the ticket was scanned/purchased on ticket machine (HH:MM:SS).
- Ticket Class – This is the ticket type, look at it in conjunction with the Ticket description and you should be able to figure out what method is used. If it just says Adult or Child, then it is likely a paper ticket which you can check by looking at the 'Net' column. This shows whether physical payment was taken (Cash).
- Trip Number – this relates to which trip the bus is on. A bus will generally do the same route all day (occasionally this will change for example to do a school) and it runs on a board. The board starts from when the bus leaves the garage to when it comes back in at the end of the day. These boards are often called running numbers or blocks and indicates what trips they do. When a timetable is created, each end-to-end is given a trip number which generally starts at the beginning of the day until the end. The bus will then go out on its own board and does certain trips within the timetable along with other buses.
- Ticket description – Indicates exactly what ticket it is – generally an M-Ticket will have an M preceding it and will be a QR code due to it needing to be scanned. Smart is a Keycard, whether it be a loaded product or a concessionary – OAP/disabled card.
- Net – This is the fare paid, this will only show if cash has been used, and shows how much they paid.
- Tickets – Number of tickets issued.
- Pass – number of passes, this is either done manually or automatically by the ticket machine when a paper or mobile QR code is scanned, or a paid product is found.
- Conc – This is the Concessions for the elderly or disabled.
- Passengers – number of passengers per transaction.

The descriptive statistics for the ticket data reveal insights into various aspects of passenger transactions. The dataset consists of 76,009 entries across 12 columns. Notably, the most frequent bus stop recorded is "Brighton Marina," accounting for 7,970 instances. Transaction times span 43,071 unique values, with the most common time being "14:35:14," occurring 9 times. Regarding ticket class descriptions, "Smart" tickets are predominant, comprising 25,992 entries, followed by other ticket types. However, observations such as negative values are observed in some variables, suggesting potential data issues requiring attention before the exploratory analysis phase. The statistics provide a comprehensive overview of ticketing patterns, including trip numbers, net transactions, ticket counts, pass counts, concurrence counts, and passenger counts, offering valuable insights into passenger behaviour and ticket usage.

Numerical Variables:

variables	count	mean	std	min	25%	50%	75%	max
ETM Route Code	76009.0	7.000000	0.000000	7.0	7.0	7.0	7.0	7.0
Trip Number	76009.0	140.946348	95.647448	1.0	76.0	138.0	194.0	1456.0
Net	76009.0	0.199891	0.888686	-36.0	0.0	0.0	0.0	36.0
Tickets	76009.0	0.402426	0.550958	-9.0	0.0	0.0	1.0	15.0
Pass	76009.0	0.394795	0.551750	0.0	0.0	0.0	1.0	33.0
Conc	76009.0	0.229473	0.420559	0.0	0.0	0.0	0.0	2.0
Passengers	76009.0	1.026694	0.318137	-9.0	1.0	1.0	1.0	33.0

Figure 1: Numerical ticket variables.

Categorical variables:

	count	unique	top	freq
Bus Stop	76009	37	Brighton Marina	7970
Transaction Time	76009	43071	14:35:14	9
Ticket Class Description	76009	11	Smart	25992
Ticket Description	76009	65	Adult TOTO	22468

Figure 2: Categorical ticket variables.

GPS data:

The GPS data shows:

- Block No. (Running/board no).
- Route No.
- Route description.
- Pattern Number – This indicates which route path the bus takes. Some early morning journeys go via Brighton Station and this is differentiated on the GPS with a pattern number.
- Direction – This will either be 1 or 2 depending on the direction of travel.
- Column F and I indicates the bus stop and whether it is a timing point or not. Timing points are stops where the driver should not be going past early, therefore there may be a longer dwell time at these stops. (The stops on this sheet are in stop order).
- Rec Vehicle No. – Fleet number of the bus.
- Location Number – Unique number for a particular stop (Bus Stop).
- Date – operational date.
- Planned Arrival – Time bus scheduled to be at stop.
- Planned Departure – Time bus scheduled to leave stop.
- Actual arrival time – Time bus actually arrived at stop.
- Actual Departure time – Time bus actually departed stop.

The descriptive analysis of the GPS data provides insights into the operational aspects of the bus service. The dataset comprises 40,726 entries across 14 columns, detailing information such as block numbers, route numbers, patterns, directions, and bus numbers. Notably, the mean block number is approximately 705.87, with a standard deviation of 14.63. The route description and pattern number indicate uniformity, with all entries having the same values. The direction column suggests that most entries have a direction value close to 1. The timing point column reveals that the majority of entries are non-timing points, indicating potential variability in bus stop classifications. Regarding bus stops, "North Road" appears to be the most frequent, occurring 1,516 times. However, it's notable that anomalies such as "NonTiming Point" entries and non-uniform timing point frequencies are observed. Additionally, planned and actual arrival/departure times exhibit variability, with some instances occurring more frequently than others. These findings highlight the need for further exploration and data preprocessing to address anomalies and ensure data consistency before further analysis.

Numerical variables:

Statistic	Block_No	Route_No	Route_Desc	Pattern_No	Direction	Bus number
Count	40726.0	40726.0	40726.0	40726.0	40726.0	40726.0
Mean	705.874527	7.0	7.0	50.439228	1.499386	739.878554

Std	14.627603	0.0	0.0	49.500570	0.500006	183.257257
Min	501.0	7.0	7.0	1.0	1.0	307.0
25%	703.0	7.0	7.0	1.0	1.0	825.0
50%	707.0	7.0	7.0	1.0	1.0	830.0
75%	710.0	7.0	7.0	100.0	2.0	834.0
Max	714.0	7.0	7.0	100.0	2.0	840.0

Figure 3: Numerical GPS variables.

Categorical variables:

Variable	Count	Unique	Top	Frequency
Timing Point	40726	2	NonTiming Point	28716
Location No	40726	52	679(0)	1513
Bus Stop	40726	38	North Road	1516
Planned Arrival Time	40726	11111	11:42:00	32
Planned Departure Time	40726	11111	11:42:00	32
Actual Arrival Time	40726	30532	11:04:54	6
Actual Departure Time	40726	30515	13:17:16	6

Figure 4: Categorical GPS variables.

EXPLORATORY DATA ANALYSIS

The characteristics of the data will be explored, and fundamental as well as intriguing questions pertaining to the dataset will be addressed by using ticket type data from the E.P. Morris database along with GPS data.

Number and Ratio of passengers from 4-10 June?

The number of passenger transactions remained consistent across most days, except for June 4, 2023, which stood out due to being a holiday (Sunday). As given in appendix Code A2 and shown in the visualization below, Figure 5 and 6 illustrates the contribution of passengers on each day.

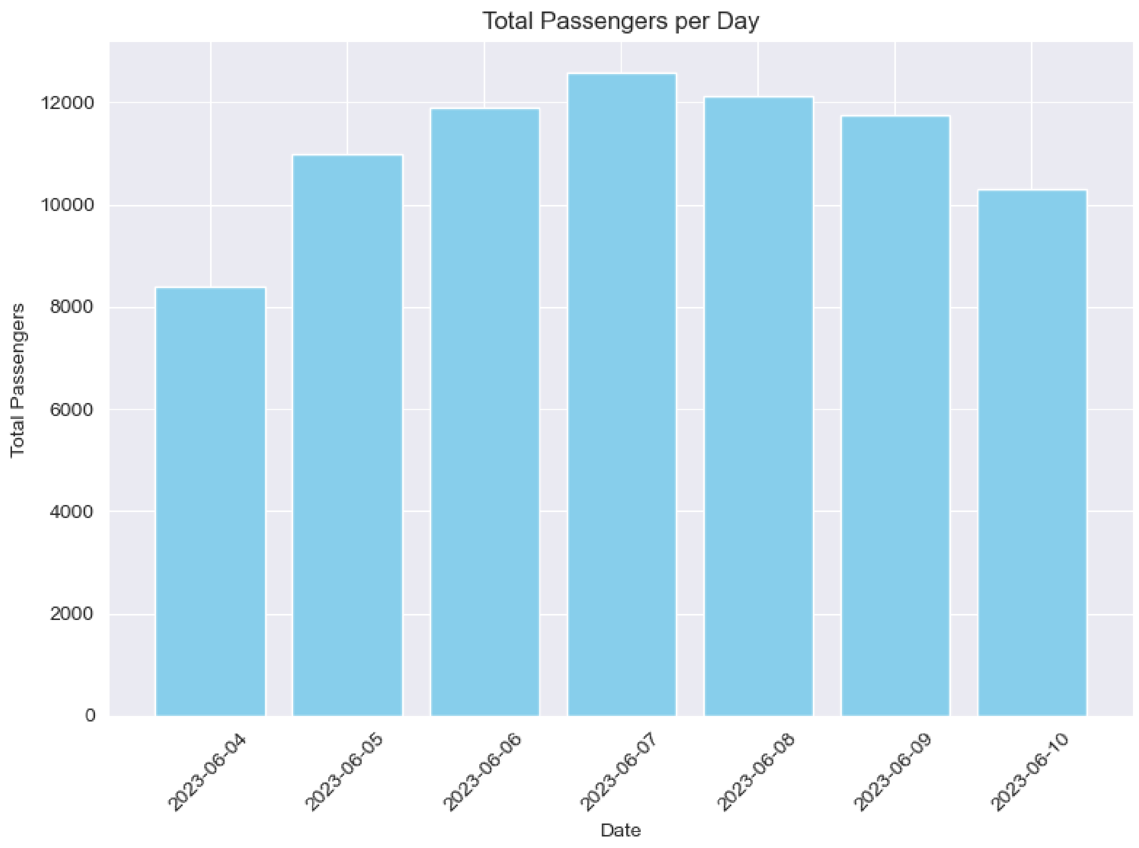


Figure 5: Passengers each date.

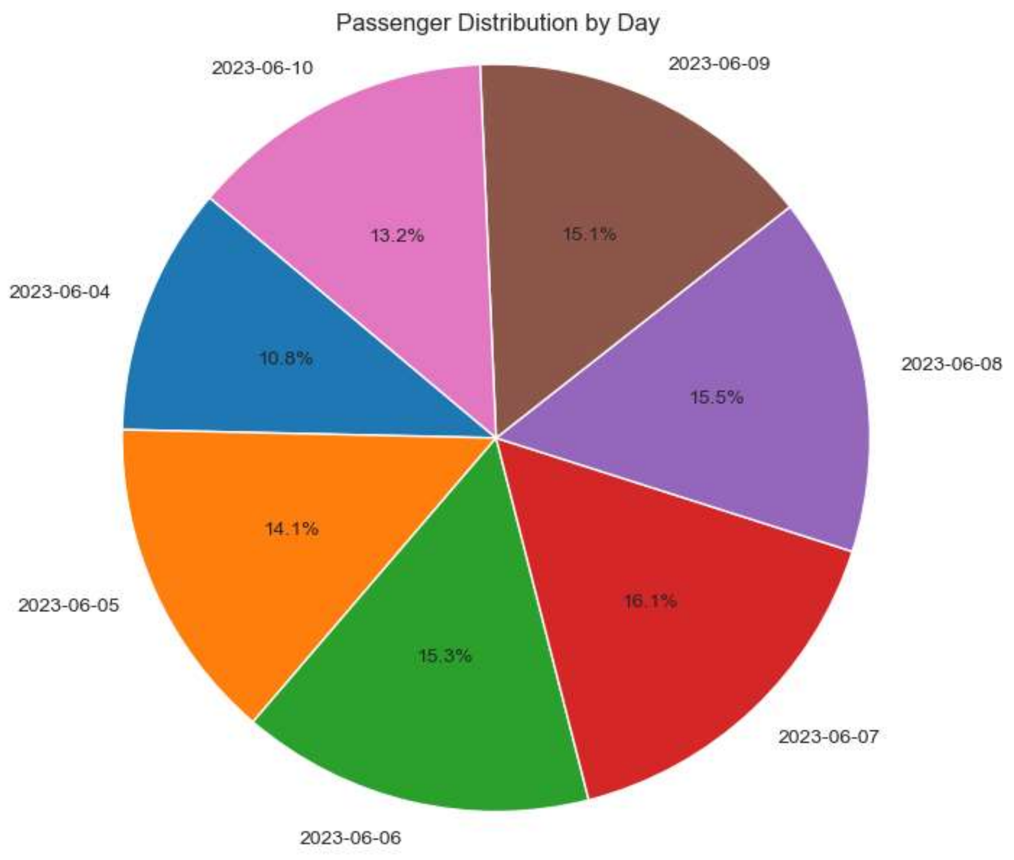


Figure 6: Passengers ratio each date.

Which ticket type is most popular?

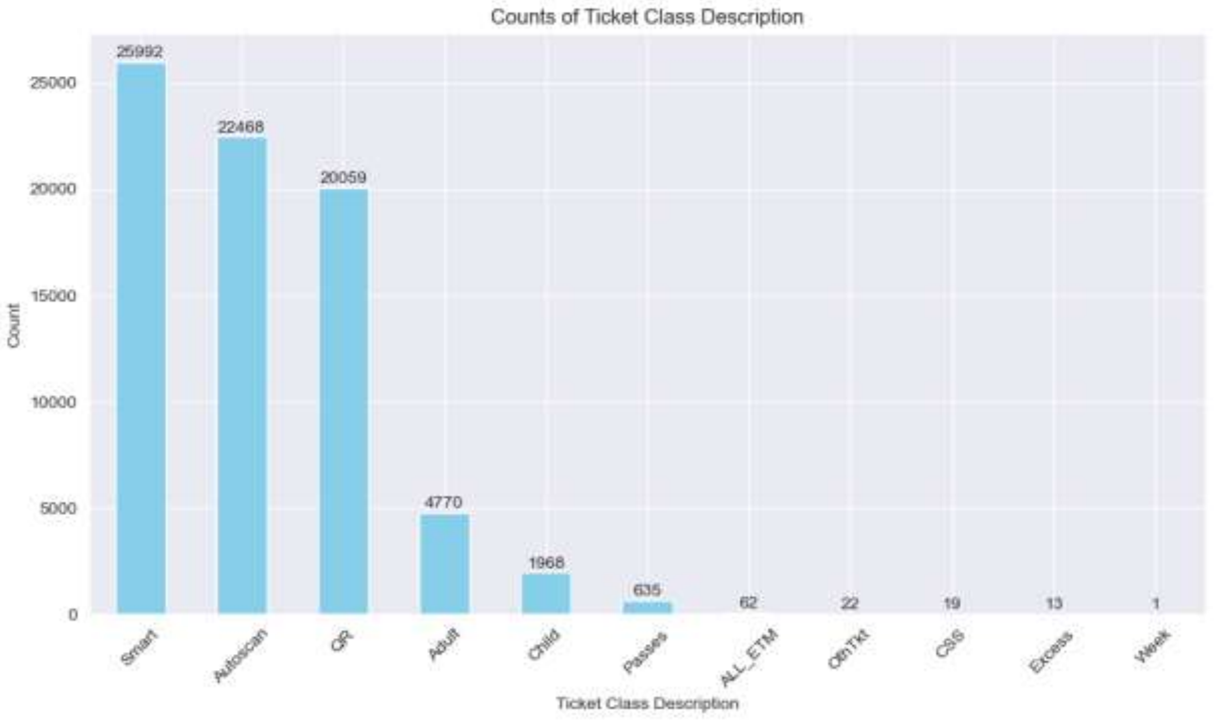
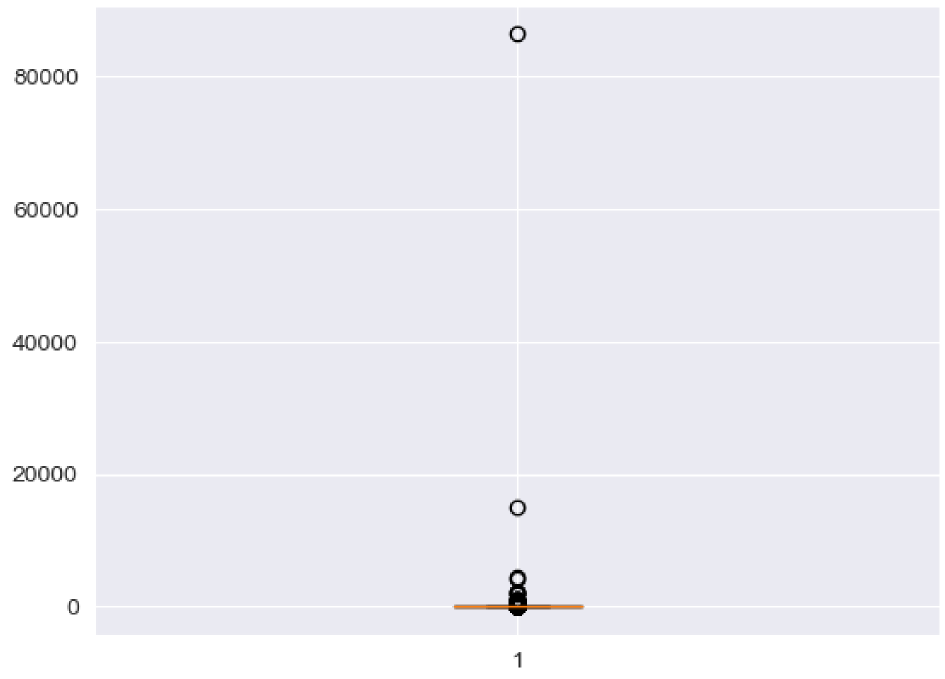


Figure 7: Ticket type popularity.

Figure 7 illustrates the popularity of various ticket types, highlighting that 'Smart,' 'Autoscan,' and 'QR' ticket types are predominant among the 11 types analysed. Notably, the fourth most popular ticket type, 'Adult,' is approximately four times less prevalent than the 'QR' ticket type. See appendix Code A3.

Which payment method is quicker or slower? What is the average transaction time of different ticket types?

To derive this, the first transaction time for each ticket type in seconds is calculated. A new column called 'T_time' (in seconds) is created by grouping the ticket data by 'Bus Stop', 'Trip Number', and 'Date', and then subtracting their subsequent 'Transaction Time' values from each other. Note that 'Transaction Time' refers to the actual recorded time, while 'T_time' represents the overall time taken by a ticket type. ‘T_time' boxplot:

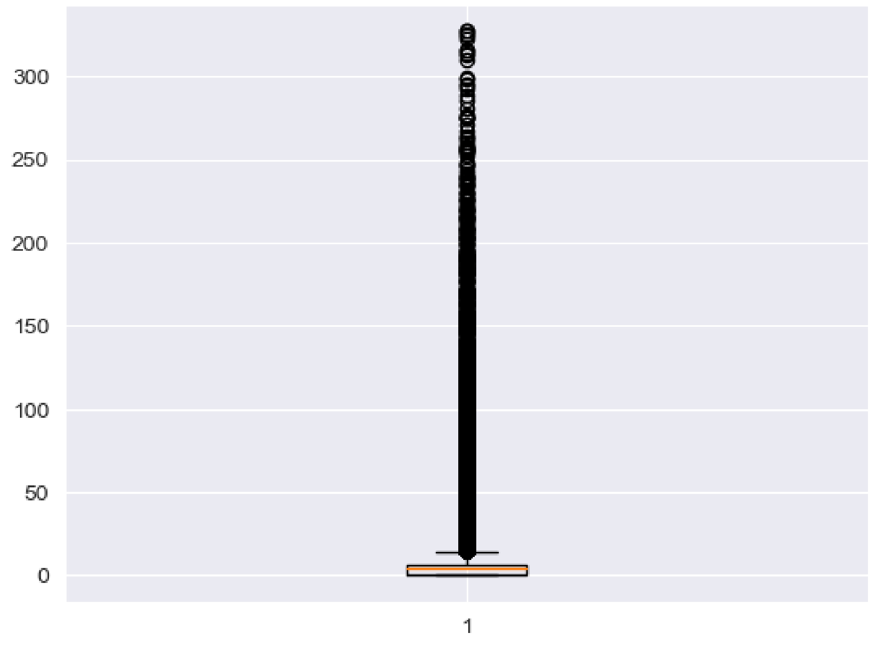


Clearly, there are some extreme observations, as the maximum value is 86,304, which is very high. To id entify these large data points, the column was filtered using the mean plus the standard deviation of the column. A total of 55 such observations were found and subsequently removed from the data.

```
Number of anomalies: 55
Time of anomalies:
                             Bus Stop Transaction Time   T_time
5639        George Street (stop J)         17:07:00    332.0
5758        George Street (stop J)         17:17:06    337.0
```

```
7131       George Street (stop J)         20:06:14    494.0
8192       George Street (stop J)         06:32:49    423.0
10317               Eaton Gardens         10:02:10    341.0
11898   Brighton Station (stop E)         13:17:01   2107.0
14135      George Street (stop J)         15:08:47    376.0
17328                Hove Station         19:04:50    778.0
18781      George Street (stop J)         05:19:05    412.0
21098                Arundel Road         09:53:24    338.0
24421             Brighton Marina         18:25:30  15061.0
24913                Hove Station         15:38:52    792.0
24915               Eaton Gardens         15:42:30   1090.0
24923   Brighton Station (stop E)         15:50:41   2175.0
27394              Chesham Street         17:31:45    393.0
28265                Hove Station         18:58:37    797.0
28269   Brighton Station (stop E)         19:07:41   2078.0
28863      George Street (stop J)         18:51:19    420.0
30092   Brighton Station (stop E)         22:44:21   1810.0
30514             Brighton Marina         05:25:00    545.0
31172              Compton Avenue         07:43:55    351.0
36007                Hove Station         13:32:04    481.0
40085                Hove Station         17:46:58    625.0
40549             Brighton Marina         18:32:44    368.0
41505        Upper Bedford Street         20:18:02    367.0
41531             Brighton Marina         20:13:29    416.0
42123             Brighton Marina         19:09:24   1037.0
42185      George Street (stop J)         21:34:36    467.0
42487             LiDL Superstore         22:01:40    388.0
42499             Brighton Marina         22:06:12   1071.0
42517       North Street (stop C)         23:16:48    513.0
42532   Brighton Station (stop A)         23:22:55   4192.0
46001             Brighton Marina         10:56:10    345.0
47473                Hove Station         13:41:05    806.0
47476               Eaton Gardens         13:44:28   1091.0
53247      George Street (stop J)         19:19:06    372.0
56831               Eaton Gardens         10:27:58   1073.0
57696       North Street (stop X)         11:04:32   1016.0
62049              Compton Avenue         16:47:25    426.0
63452               Eaton Gardens         18:34:30    597.0
63457               Eaton Gardens         18:54:14    391.0
64963               Sussex Square         20:50:45    504.0
65071      George Street (stop J)         20:17:24    344.0
66119             Brighton Marina         23:59:47  86304.0
67130  St James's Street (stop J)         09:34:41    360.0
70681             Montefiore Road         13:53:21    499.0
70682                 Osmond Road         13:54:18    509.0
71435                  Lyon Close         15:00:35    866.0
72733             LiDL Superstore         18:01:36    368.0
72745             Brighton Marina         18:06:05    982.0
74184      George Street (stop J)         19:14:59    516.0
74222   Brighton Station (stop A)         19:30:23    588.0
74225        Clock Tower (stop N)         19:32:02    437.0
74989   Brighton Station (stop E)         21:42:16   2070.0
74995                Hove Station         21:52:13   4395.0
```

‘T_time’ boxplot after removing large observations:



After this, the total transaction time for each type of ticket was calculated. Then, the total transaction time was divided by the total count of each ticket type to find the average transaction time.

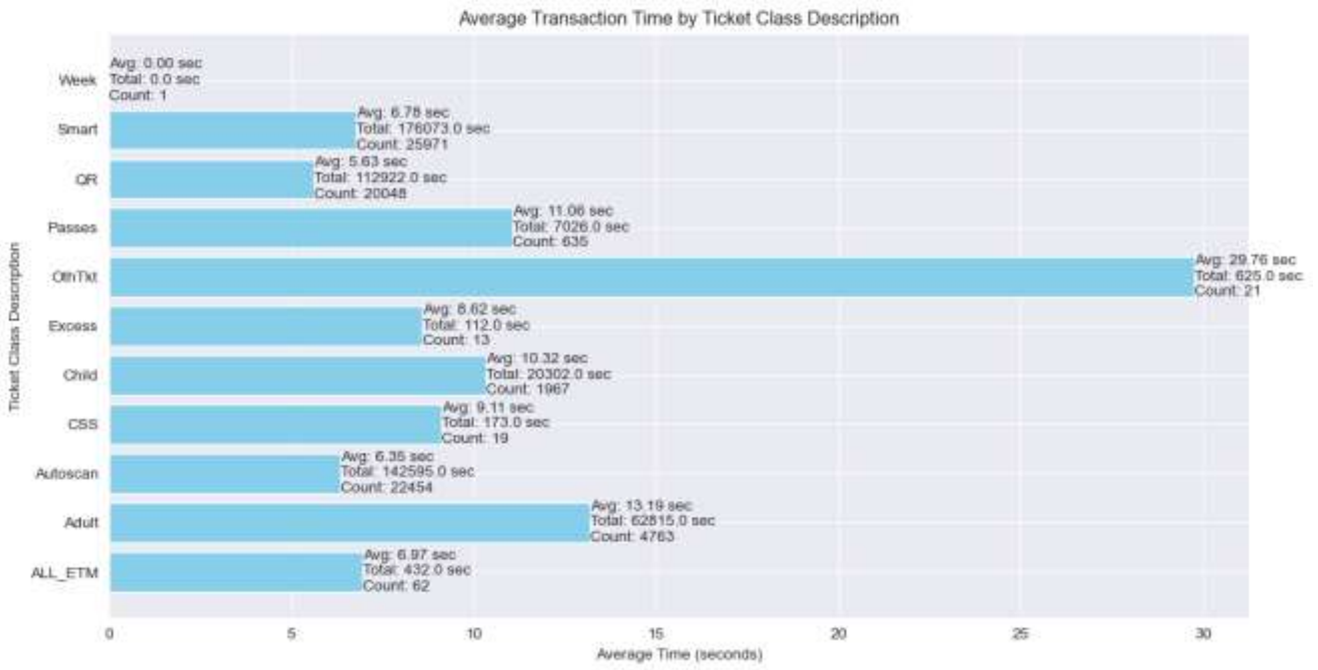


Figure 8: Average transaction time of tickets.

Examining the average transaction time for each ticket type provided insights into transaction efficiency and processing durations. For instance, 'QR' tickets exhibited the shortest average transaction time, while 'OthTkt' and 'Adult' tickets had comparatively longer transaction durations. Although, ‘Smart’ ticket has third lowest ticket time, interesting to note that, it has the highest ticket counts among all. Understanding transaction time variations among ticket types can inform strategies to streamline ticketing processes and reduce passenger wait times.

All steps were documented in Appendix Code A4.

Curious case of Bus Stops

Dwell time calculation

When attempting to extract dwell time from the GPS dataset, issues arose with the 'Actual Arrival Time' and 'Actual Departure Time' columns. Some values were in datetime format, containing both date and time (e.g., YYYY-MM-DD HH:MM:SS). To address this, anomalies were identified and converted to time format. Subsequently, dwell time was calculated for each bus at each stop, as shown in Appendix Code A5.

Dwell time calculation steps:

- Convert the 'Actual Arrival Time' and 'Actual Departure Time' columns from string format to **datetime** objects.

  Working with **datetime** objects allows for easier time calculations.
- **Lambda Function**: Applies a function to each row of the dataframe.
- **Time Conversion**:Converts the hours, minutes, and seconds of both 'Actual Arrival Time' and 'Actual Departure Time' into total seconds since the start of the day.
- **Conditional Check**:
  - If 'Actual Departure Time' is later than or equal to 'Actual Arrival Time':

    Simple subtraction to get the dwell time.
  - If 'Actual Departure Time' is earlier than 'Actual Arrival Time' (this could happen if the times cross midnight):

    Adds 24 hours (in seconds) to the departure time before subtracting the arrival time.

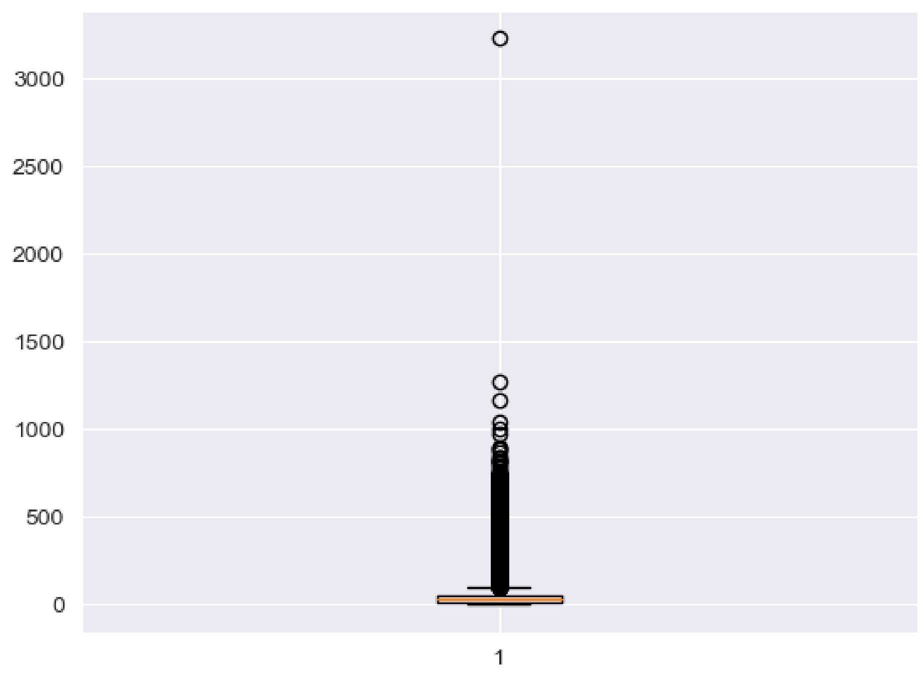
    This ensures the calculation correctly handles cases where the bus departs after midnight.

Example Scenario:

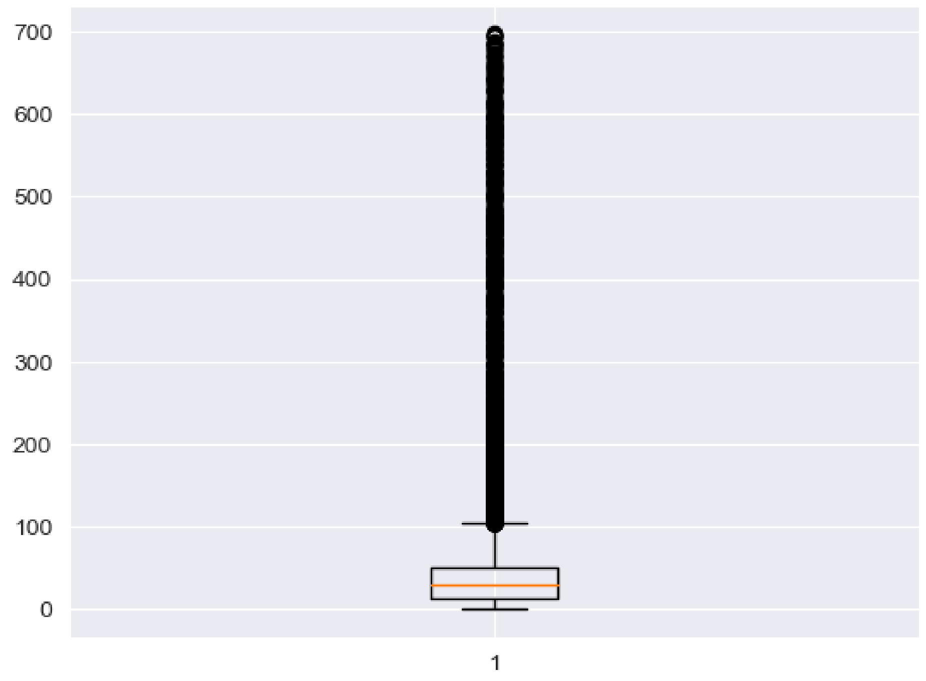
If 'Actual Arrival Time' is **23:55:00** and 'Actual Departure Time' is **00:05:00**:

The code will correctly handle this by adding 24 hours to the departure time, resulting in a dwell time of 600 seconds (10 minutes).

Let’s look at the observations of dwell time:



There are large data points which might result in bias in our analysis. By defining the threshold at 700 and removing the data points lets look at the dwell time now:



How long the buses are at each stop for?

Following this calculation, dwell time values were aggregated according to bus stops.
It appears that at the beginning of the journey, such as at George Street, buses may experience longer transaction and dwell times. This could be due to the assumption that buses arrive early at the first stop to pick up passengers, leading to longer transaction times as passengers board and tickets are processed. Consequently, there may be longer intervals between consecutive ticket transactions during this initial phase of the journey.

```
Bus Stop          Dwell time(seconds)

George Street HJ             247966
Brighton Station A            87297
Hove Station                  72749
North Street C                71830
Clock Tower (stop N)          71115
County Hospital               70602
Brighton Station E            64575
North Road                    60072
Clock Tower                   58586
North Street X                55925
Eaton Gardens                 53452
St James's St.                51825
Lyon Close                    50340
Bottom of Montefiore Road     50035
Wilbury Villas                47909
Upper Bedford Street          46644
Old Steine U                  46457
The Waterfront                43797
Arundel Road Btn              42817
Park Street                   42412
Lidl Superstore               41724
Roedean Road                  41551
College Place                 38002
Seven Dials/ Buckingham Pl    34981
St Marys Hall                 34371
Osmond Road                   34136
Compton Avenue                30697
Seven Dials/Goldsmid Rd       27360
Law Courts Brighton           24669
Livingstone Road              23899
St James Street Top           21234
Sussex Square                 19404
Devonshire Place              18874
Holland Road                  16073
Upper Rock Gardens            14828
Chesham Street                12753
Marina Cinema (Set Down)      12411
Brighton Marina                3924
```

Figure 9: Dwell time at Bus Stops

Which stop sees greater frequencies of customer?

It is intriguing to note that among all the stops, four specific stops stand out due to their notably higher passenger boarding numbers:

```
Bus Stop
Brighton Marina                8371
St James's Street (stop J)     6113
Brighton Station (stop A)      5427
George Street (stop J)         5314
County Hospital                4306
Brighton Station (stop E)      3685
North Street (stop C)          3676
North Street (stop X)          3312
Seven Dials                    3177
Clock Tower (stop L)           3002
Hove Station                   2968
Lyon Close                     2258
Wilbury Villas                 2249
Old Steine (stop U)            2240
Arundel Road                   2164
Upper Bedford Street           2131
Eaton Gardens                  1959
Clock Tower (stop N)           1872
Montefiore Road                1805
College Place                  1525
Park Street                    1417
Sussex Square                  1375
Devonshire Place               1354
LiDL Superstore                1250
St Marys Hall                   926
Osmond Road                     799
North Road                      754
Livingstone Road                639
Roedean Road                    478
Law Courts                      425
Compton Avenue                  388
Holland Road                    210
Rock Gardens                    165
Chesham Street                  147
Boundary Road (stop B)          142
The Waterfront                   11
Marina Cinema                     4
```

Figure 10: Passengers boarding.

Above data indicates that George Street ranks fourth in terms of boarding passengers (5314) . This observation provides some support for our assumption regarding longer transaction and dwell times at the beginning of the journey, particularly at George Street. The fact that George Street is not the top-ranking stop for boarding passengers suggests that this stop is potentially contributing to the anomalies observed in transaction and dwell times at the start of the journey.

These stops appear to attract a significantly larger volume of passengers compared to others, suggesting potential factors such as location, connectivity, or local attractions that may contribute to their popularity among commuters. Further analysis of these stops could provide valuable insights into passenger behaviour and transportation patterns within the area.

Which ticket type is popular at each stop?

A noteworthy discovery is that the ticket type popularity across all stops predominantly comprises either Smart or Autoscan ticket types, with the exception of St James's Street (stop J) which had QR tickets. This anomaly suggests potential underlying factors in the demographic makeup surrounding stop J. Further exploration into the demographics of the area surrounding St James's Street may shed light on why this stop deviates from the observed trend. Understanding these factors could provide valuable insights into passenger behaviour and preferences at this location.

All the steps involved in the process were documented in the provided Appendix Code A5.

Which time of the is the bus-iest between 4-10 June?

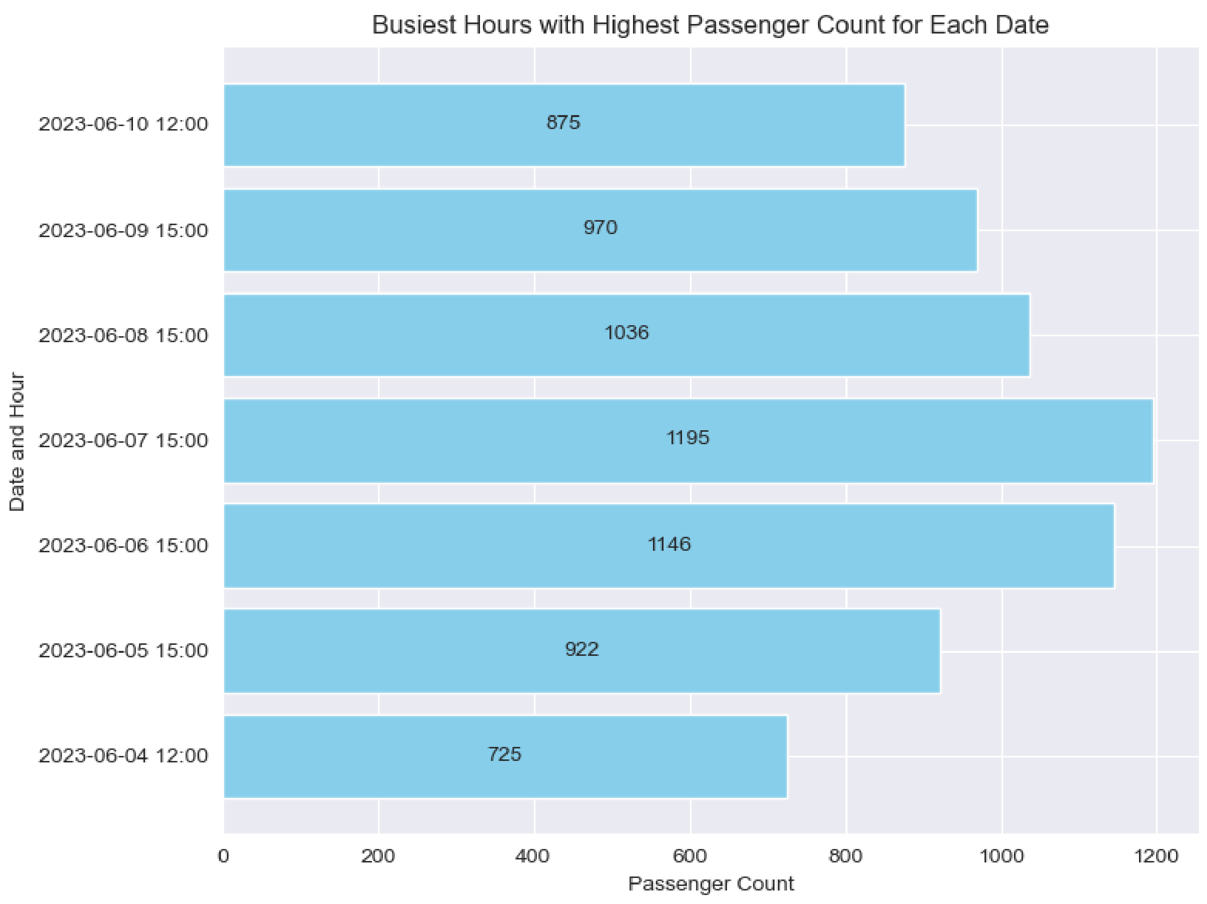


Figure 11: Busiest hours.

MODELLING

JOINING THE TABLES AND PREPROCESSING

The aim of Appendix Code A7 was to consolidate ticket-type data from E.P Morries and ticketer data, with a primary focus on ensuring that no bus stop information was lost during the integration process. This involved merging the datasets by standardizing and renaming columns to establish common identifiers. The key columns used for merging were 'Ticket Description', 'Transaction Time', 'Date', and 'Trip Number'.

Data Integration: Initial Data Merging: E.P Morries and ticketer data were merged, prioritizing the extraction of bus stop information. Duplicate entries were removed to ensure data integrity.

Joining Ticket Data with GPS Data: The ticket data was then joined with GPS data based on common fields such as 'Bus Stop', 'Bus Number', and 'Date'. This step aimed to enrich the ticket data with geographical information.

Redundant or unnecessary columns such as 'Route Number', 'Operator', 'Number', 'Value of Ride', 'Custom Formula', 'ETM Route Code', 'Route Number', and 'Route Description' were dropped. These columns either contained null values or redundant information.

Ticket Class Description Segmentation: The 'Ticket Class Description' column was segmented into five distinct categories to facilitate analysis:

'Cash' for ticket types 'Adult' and 'Child',

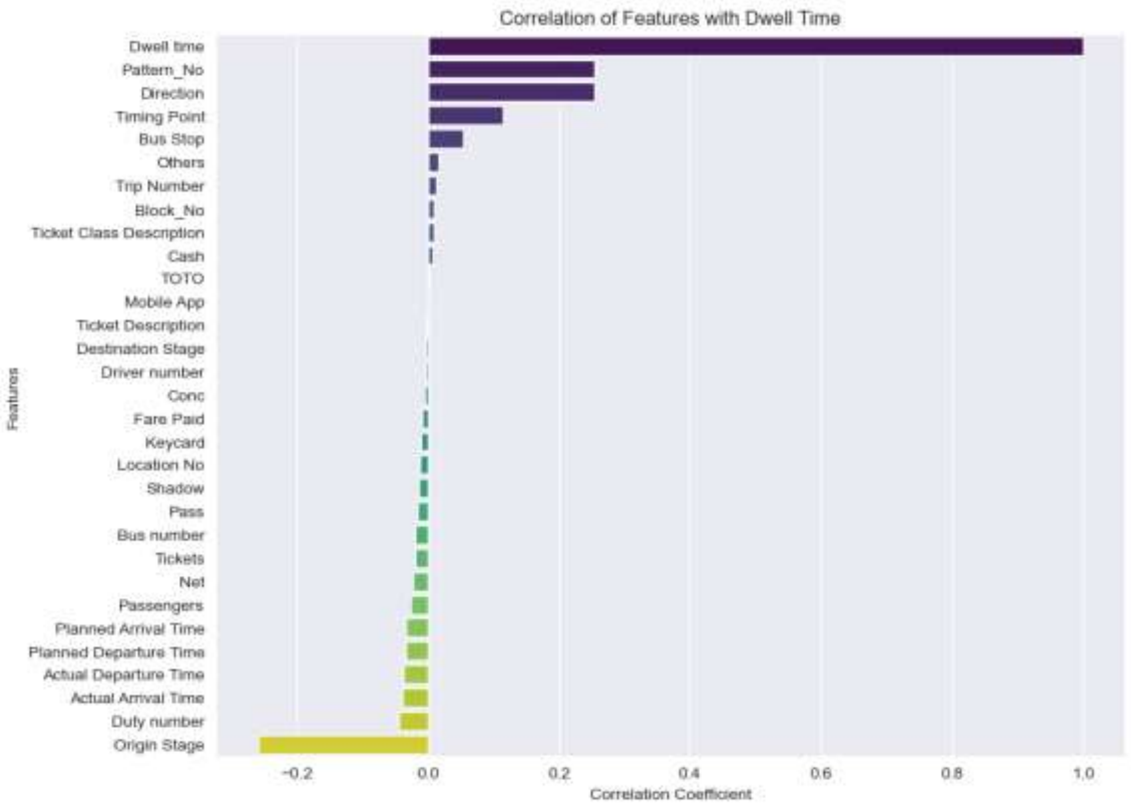
'TOTO' for 'Autoscan' tickets,

'Keycard' for 'Smart' tickets,

'Mobile App' for 'QR' tickets, and

'Others' for various ticket types including 'Passes', 'ALL_ETM', 'OthTkt', 'CSS', 'Excess', and 'Week'.

Categorical Variable Encoding: Categorical variables were encoded to prepare the data for modelling. This encoding step transformed categorical data into a numerical format that machine learning algorithms could process effectively.



The correlation among variables with 'Dwell time' suggests that Deterministic models like Ordinary Least Squares (OLS) regression can be appropriate for modelling continuous target variables, adding polynomial terms might be a better fit for the data due to the presence of both strong positive and negative correlations. OLS regression offers flexibility to capture complex, non-linear relationships between variables, which could be beneficial in this scenario. However, it is essential to consider potential challenges such as overfitting and the interpretation of higher-order terms. Proper feature engineering, model selection, and evaluation, including regularization techniques if needed, are crucial steps in implementing OLS regression effectively.

DETERMINISTIC MODELLING

Introduction to Polynomial Regression with OLS: Polynomial regression, when coupled with Ordinary Least Squares (OLS) estimation, offers a powerful framework for analysing non-linear relationships between variables. Unlike linear regression, which assumes a linear relationship between predictors and the response variable, polynomial regression with OLS can capture curvature and non-linearity in the data. This makes it particularly useful when the relationship between variables cannot be adequately represented by a straight line.

Polynomial regression with OLS is widely used in various fields, including economics, engineering, biology, and social sciences, where complex relationships between variables need to be accurately modelled. It offers greater flexibility and versatility compared to linear regression, allowing researchers to explore and analyse data with greater precision.

Purpose of Choosing Polynomial Regression with OLS: In this dissertation, the choice of polynomial regression with OLS serves multifaceted purposes. One primary objective is to investigate the individual impact of predictor variables on the response variable while accounting for non-linear relationships. Through the utilization of polynomial regression with OLS, we can model the non-linear relationship between each predictor variable and the response variable, thereby facilitating insights into the specific effects of each predictor.

Additionally, the significance of predictor variables in predicting the response variable can be assessed through polynomial regression with OLS. By examining the coefficients and their associated p-values obtained from OLS estimation, we can identify predictors with a statistically significant impact on the response variable. This aids in the identification of key variables that drive the variation in the response variable and understanding their relative importance.

Another purpose of selecting polynomial regression with OLS is to capture complex patterns and interactions between variables that may not be adequately captured by linear models. By leveraging OLS estimation within the polynomial regression framework, we can flexibly model curvature and non-linearity in the data, thereby providing a more accurate representation of the underlying relationships.

Description of the Model: Polynomial regression extends the linear regression model by introducing polynomial terms, which are powers of the predictor variables (Spanton, 2023). The polynomial regression equation can be expressed as follows:

$$Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3 + \ldots + \beta_n + \epsilon$$

- Here, Y represents the response variable, X represents the predictor variable, β_0, β_1, β_2,…, β_n represent the coefficients of the polynomial terms, n represents the degree of the polynomial (indicating the highest power of X included in the model), and ϵ represents the error term.

- By including polynomial terms of different degrees (e.g., X^2, X^3,X^4etc.), polynomial regression can capture non-linear relationships between the predictor and response variables. The choice of polynomial degree n^n determines the complexity of the model and the degree of flexibility in fitting the data.

- The coefficients $\beta_0, \beta_1, \beta_2, \ldots, \beta_n$ represent the effect of each polynomial term on the response variable. These coefficients can be interpreted similarly to those in linear regression, indicating the change in the response variable for a one-unit increase in the predictor variable.

Based on the provided model results outlined in appendix Code A8, let's interpret the impact of dwell time on Cash, Mobile App, Keycard, and TOTO and other variables:

Variable	Coefficient	P-value
Mobile App	4.6461	0.003
TOTO	4.6940	0.003
Cash	4.4715	0.003
Keycard	11.2892	0.004
Others	29.3117	0.001
Passengers	2.4691	0.154
Timing Point	31.4847	0.000
Origin Stage	0.3191	0.000
Direction	6.5998	0.003
Bus Stop	2.5714	0.000
Actual Arrival Time	-0.1837	0.018
Actual Departure Time	0.1779	0.022

Interpretation: The polynomial regression analysis reveals several significant predictors influencing dwell time in the bus transportation system. Coefficients represent the magnitude of change in the response variable (dwell time) for a one-unit change in the predictor variable, while p-values indicate the statistical significance of these relationships. For instance, let's focus on the impact of the Direction on dwell time. The coefficient associated with Direction is 6.5998, indicating that for every one-unit change in Direction transactions, there is an estimated increase of approximately 6.5998 units in dwell time, holding all other variables constant. Notably, transactions made through the Mobile App, TOTO, Cash, Keycard payment methods exhibit varying degrees of impact on dwell time. Specifically, Mobile App and TOTO transactions show comparable impacts, with coefficients of 4.6461 and 4.6940, respectively, both statistically significant ($p < 0.05$). Cash transactions also contribute significantly to dwell time, with a coefficient of 4.4715 ($p = 0.003$). However, the Keycard payment method emerges as the most influential, with a coefficient of 11.2892 and a low p-value (p = 0.004), signifying its substantial impact.

The variable representing the number of passengers aboard the bus does not significantly influence dwell time, as indicated by its coefficient of 2.4691 and a p-value of 0.154 (<0.05).

While both the Origin Stage and Actual Departure Time variables exhibit statistical significance (p < 0.05), their overall impact on dwell time is relatively modest, with coefficient values of 0.3191 and 0.1779, respectively.

Out of all the variables, only Actual Arrival Time has a negative coefficient, indicating that a one-unit change in this predictor corresponds to a decrease of 0.1837 in dwell time.

Major two significant variables include Timing Point arrivals and the 'Others' payment category, both exhibiting notable effects on dwell time. Timing Point arrivals display a coefficient of 31.4847 (p < 0.001), indicating a substantial increase in dwell time associated with these events. Similarly, the 'Others' category which is consists of 'Passes' , 'ALL_ETM', 'OthTkt', 'CSS', 'Excess', 'Week' ticket types also play a significant role, with a coefficient of 29.3117 (p < 0.001). The "Others" payment method showed a larger coefficient than TOTO, cash, keycard, and mobile app, suggesting that further investigation is needed into the five payment types included under "Others".

Model Fit: The R-squared value of 0.462 indicates that approximately 46.2% of the variance in dwell time is explained by the model. While this indicates a moderate level of explanatory power, there might still be other factors not included in the model that contribute to dwell time.

Potential Issues*:* The model notes that there might be issues with multicollinearity or other numerical problems due to the large condition number (Cond. No. 1.30e+16). This suggests that some predictor variables might be highly correlated with each other or other numerical problems, which can affect the stability and reliability of the coefficient estimates.

MACHINE LEARNING MODELLING

K-Nearest Neighbors (KNN): KNN is considered a simple and intuitive algorithm that is employed for both classification and regression tasks.

The principle behind KNN involves predicting the target variable of a new data point by considering the majority class (for classification) or the average value (for regression) of its k nearest neighbors in the feature space. Predictions in KNN are made by calculating the distance between the new data point and all existing data points in the training set, then selecting the k nearest neighbors based on this distance metric (typically Euclidean distance). KNN is characterized as a non-parametric method, meaning it makes no assumptions about the underlying distribution of the data. More information about KNN can be found in (James et al., 2013).

XGBoost (Extreme Gradient Boosting): XGBoost is recognized as a powerful and efficient implementation of the gradient boosting algorithm, which is widely utilized in supervised learning tasks. Gradient boosting is employed to build an ensemble of weak learners (usually decision trees) sequentially, with each new tree attempting to correct the errors made by the previous ones. XGBoost enhances traditional gradient boosting by incorporating regularization techniques to prevent overfitting and improve generalization performance. For detailed insights into XGBoost, refer to (Chen & Guestrin, 2016).

Additionally, it includes advanced features such as parallelization, tree pruning, and hardware optimization to achieve high computational efficiency and scalability.

Random Forest: Random Forest is acknowledged as an ensemble learning method that constructs multiple decision trees during training and outputs the average prediction (for regression) or the mode (for classification) of the individual trees. Each decision tree in the Random Forest is built independently using a subset of the training data and a subset of the features, which introduces randomness and helps reduce overfitting. Random Forest combines the predictions of multiple trees to produce a more robust and accurate model compared to any individual tree. It is renowned for its flexibility, robustness to noise, and resistance to overfitting, making it a popular choice for various regression and classification tasks. For a comprehensive explanation of the Random Forest algorithm, see (Breiman, 2001).

Using linear regression as a baseline model allows to compare the performance of these algorithms against a simpler, linear model and assess whether the added complexity of the ensemble methods (XGBoost and Random Forest) or the non-parametric approach (KNN) leads to improved predictive accuracy.

NESTED MACHINE LEARNING MODEL – KNN, XGBOOST, RANDOM FOREST

In the appendix, labelled as Code A9, 80% of the data was allocated for training, while the remaining 20% was designated for testing. This partitioning strategy was implemented to assess and contrast the performance of Linear Regression, KNN, XGBoost, and Random Forest regression models.

Model	MSE (Mean Squared Error)	RMSE (Root Mean Squared Error)	MAE (Mean Absolute Error)	R-squared
Linear Regression	922.69	30.38	20.70	0.15
Random Forest Regression	5.89	2.43	0.41	0.99
XGBoost Regression	373.69	19.33	12.93	0.66
K-Nearest Neighbors	20.51	4.53	1.01	0.98

Figure 12: Performance Comparison

Based on the provided performance metrics, the Random Forest Regression model outperforms the other models in terms of MSE, RMSE, and MAE, with significantly lower values indicating better accuracy. It also exhibits the highest R-squared value, suggesting a better fit to the data compared to the other models. The XGBoost Regression model performs reasonably well with relatively low MSE and RMSE values, but it has a higher MAE and lower R-squared compared to the Random Forest Regression. The Linear Regression model shows the poorest performance, with the highest MSE, RMSE, and MAE, and the lowest R-squared value, indicating a weaker fit to the data. The K-Nearest Neighbors Regression model performs moderately well but has slightly higher MSE, RMSE, and MAE compared to the Random Forest and XGBoost models, and a slightly lower R-squared value. Overall, the Random Forest Regression model appears to be the most suitable for the given dataset based on the provided performance metrics.

In addition to the comparison, it's evident that Linear Regression is not well-suited for this dataset as it exhibits the poorest performance across all metrics. With substantially higher values of MSE, RMSE, and MAE, and the lowest R-squared value among all models, Linear Regression fails to adequately capture the underlying relationships in the data. This indicates that the linear assumption may not hold true for this dataset, leading to a poor fit and inaccurate predictions. Therefore, more complex models like Random Forest Regression and XGBoost Regression, which can capture nonlinear relationships, are preferred for this dataset.

CROSS VALIDATION - KNN, XGBOOST, RANDOM FOREST

As demonstrated in appendix Code A10, the three machine learning models underwent a 5-fold cross-validation process. This approach offers a comprehensive assessment of model performance and mitigates the risk of overfitting (Ng, 1998) , maximising data utility, reducing bias, distinguishing it from the traditional method of building models individually (Moreno-Torres et al., 2012).

Here's a detailed interpretation of the cross-validation results:

Model	MSE (CV)	RMSE (CV)	MAE (CV)	R-squared (CV)
Random Forest Regression	9.87	3.14	0.42	0.99
XGBoost Regression	374.96	19.36	12.93	0.66
K-Nearest Neighbors	32.15	5.67	1.01	0.98

Figure 13: Cross-Validation Comparison

Including cross-validation results provides a more robust assessment of model performance by evaluating their generalizability to new data. Based on the cross-validated performance metrics, the Random Forest Regression model maintains its superiority with consistently lower MSE, RMSE, and MAE compared to the other models. Additionally, it demonstrates the highest R-squared value, indicating a better fit to the data even after cross-validation. The XGBoost Regression model shows higher error metrics and lower R-squared values compared to the Random Forest Regression, suggesting less robust performance across different data splits. The K-Nearest Neighbors Regression model also exhibits higher error metrics and a slightly lower R-squared value compared to the Random Forest Regression, indicating less favourable performance in generalizing to new data. Overall, the Random Forest Regression model remains the preferred choice for its superior performance and robustness demonstrated through cross-validation.

FEATURE IMPORTANCE - KNN, XGBOOST, RANDOM FOREST

The table below, extracted from Appendix Code A11, presents the feature importance values for each respective model. It's important to highlight that the feature importance values for Random Forest and

XGBoost were obtained directly, whereas for KNN, permutation importance was utilized since it's not a tree-based model like the former two.

Feature	Random Forest Importance	XGBoost Importance	KNN Permutation Importance
Actual Departure Time	0.335534	0.158655	45.73206
Actual Arrival Time	0.317356	0.109854	46.23626
Bus Stop	0.247154	0.326210	0.086993
Timing Point	0.065953	0.242614	0.000002
Origin Stage	0.019289	0.076621	0.395313
Direction	0.012303	0.058170	0.000007
Passengers	0.000839	0.007685	0.000145
Cash	0.000377	0.009206	0.000001
Mobile App	0.000339	0.002301	0.000005
TOTO	0.000315	0.002369	0.0000003
Keycard	0.000290	0.002239	-0.00000006
Others	0.000251	0.004076	0.000003

Based on the analysis of feature importances and model performance, several key insights can be derived from the Random Forest, XGBoost, and KNN Permutation methods. In the Random Forest model, the most important features were identified as actual departure time (0.335534), actual arrival time (0.317356), and bus stop (0.247154), with timing point (0.065953) also being moderately significant. In contrast, XGBoost placed the highest importance on bus stop (0.326210) and timing point (0.242614), followed by actual departure time (0.158655) and actual arrival time (0.109854). For the KNN Permutation method, actual arrival time (46.23626) and actual departure time (45.73206) were overwhelmingly important, while origin stage (0.395313) and bus stop (0.086993) had some significance.

Across all three models, features such as cash, TOTO, keycard, and mobile app consistently showed very low importance scores. For example, in the Random Forest model, their importance values were 0.000383, 0.000318, 0.000297, and 0.000329, respectively. Similar low importance was observed in the XGBoost model, with values of 0.009206 (cash), 0.002369 (TOTO), 0.002239 (keycard), and 0.002301 (mobile app). The KNN Permutation method also assigned minimal importance to these features, indicating that they contributed little to the predictive power of the models.

The negligible importance attributed to features such as Cash, Mobile App, TOTO, Keycard in predicting dwell time could be attributed to several assumptions. Firstly, it's possible that the efficiency of transactions associated with these payment methods is uniformly quick, resulting in minimal variability in dwell time across different payment types. Additionally, the dataset itself may exhibit low variability in the usage of these payment methods, limiting their impact on the model's predictions. Furthermore, these payment method features may have indirect effects on dwell time, with other factors such as delays, traffic conditions, and boarding procedures exerting a more significant influence on overall dwell time dynamics. Thus, while intuitively relevant, these payment method features may not directly contribute to the variability in dwell time captured by the model. Note that, all these assumptions needed to check with data provider or domain expert.

CONCLUSION

In this study, we delved into understanding dwell times along the Route 7 bus service, running between George Street, Hove, and Brighton Marina. Our goal was to uncover the factors affecting dwell times, focusing on payment methods and ticket types.

The analysis of transaction volume highlighted consistent patterns across most days, with June 4, 2023, standing out as a holiday, suggesting potential variations in travel demand during holidays compared to regular weekdays. Ticket type analysis revealed the dominance of 'Smart,' 'Autoscan,' and 'QR' ticket types, emphasizing passenger preferences and usage patterns. At initial stops like George Street, longer transaction and dwell times were observed, hinting at operational challenges related to boarding procedures. Anomalies in ticket type distribution, particularly the prevalence of 'QR' tickets at St James's Street, underscored potential demographic variations or passenger preferences around specific stops. Furthermore, peak hour analysis identified Sunday and Saturday as peak days at 12 PM, contrasting with other days where 3 PM emerged as the busiest hour. Wednesday notably emerged as the busiest day, highlighting variations in travel demand across weekdays. Examining transaction times provided insights into transaction efficiency, with 'QR' tickets exhibiting the shortest transaction times and 'OthTkt' and 'Adult' tickets experiencing longer processing durations. Although, ‘Smart’ tickets have biggest count and low average time. These findings collectively offer valuable insights for optimizing bus operations and improving service quality within the Route 7 bus service.

Utilizing deterministic modelling through polynomial regression, significant coefficients were identified for various payment methods. Cash transactions exhibited a coefficient of 4.4715 (SE = 1.526), indicating a 4.4715 increase in dwell time for each unit increase in Cash transactions, while Mobile App transactions showed a comparable impact with a coefficient of 4.6461 (SE = N/A). Keycard transactions displayed the highest coefficient of 11.2892 (SE = N/A), indicating a more substantial influence on dwell time compared to other payment methods. TOTO transactions exhibited a moderate impact with a coefficient of 4.6940 (SE = N/A). Furthermore, all four predictor variables (Cash, Mobile App, Keycard, TOTO) exhibited p-values less than 0.05, indicating their statistical significance at a 95% confidence level. This implies that these variables have a significant impact on dwell time. Regarding model fit, the R-squared value of 0.462 suggests that approximately 46.2% of the variance in dwell time is explained by the model. While this indicates a moderate level of explanatory power, there may still be other unaccounted factors contributing to dwell time.

Expanding our exploration into machine learning methodologies, we evaluated the performance of various regression models in predicting dwell times. Among individual models, Random Forest Regression emerged as the top performer, boasting significantly lower error metrics compared to Linear

Regression, XGBoost Regression, and K-Nearest Neighbors Regression. With a Mean Squared Error (MSE) of 5.89, Root Mean Squared Error (RMSE) of 2.43, Mean Absolute Error (MAE) of 0.41, and R-squared value of 0.99, the Random Forest model exhibited superior predictive accuracy and explained most of the variance in dwell time. Conversely, XGBoost Regression and K-Nearest Neighbors Regression also demonstrated promising results, outperforming Linear Regression. However, Random Forest Regression remained the frontrunner in forecasting dwell times within the Route 7 bus service, owing to its lower error metrics and higher R-squared value. Additionally, the cross-validation results further validated the efficacy of the Random Forest model, with consistently low error metrics and a high R-squared value across folds. Overall, Random Forest Regression emerges as the most suitable model for predicting dwell time, providing valuable insights for optimizing bus operations and improving service quality.

Based on the feature importance analysis from the Random Forest, XGBoost, and KNN Permutation methods, several key insights emerge. In the Random Forest model, the most important features were actual departure time (0.335534), actual arrival time (0.317356), and bus stop (0.247154), with timing point (0.065953) also being moderately significant. XGBoost highlighted bus stop (0.326210) and timing point (0.242614), followed by actual departure time (0.158655) and actual arrival time (0.109854). The KNN Permutation method found actual arrival time (46.23626) and actual departure time (45.73206) to be overwhelmingly important, with origin stage (0.395313) and bus stop (0.086993) also contributing. Consistently, features such as cash, TOTO, keycard, and mobile app showed very low importance across all models, indicating their minimal contribution to predictive power. For example, in the Random Forest model, their importance values were around 0.0003, and similarly low in XGBoost and KNN.

One limitation of the study is that dwell time is significantly influenced by external factors such as bus stops and timing points, which impact boarding frequency. A potential future direction could involve standardizing data collection methods to ensure consistency, such as adopting uniform terminology across all data collection methods, thereby preserving data integrity. Moreover, considering additional factors such as weather conditions, special events, or traffic patterns that may influence dwell times and passenger behaviour could enhance the comprehensiveness of future analyses.

An interesting recommendation from researching the data is to consider removing information related to initial Bus Stops like George Street and then analyse the data further. This idea comes from our initial exploration, which suggests that first stops often have longer dwell times compared to others.

It would also be interesting to consider converting ticket types along with their transaction times into seconds and transforming them into integer variables instead of categorical ones. This approach would allow for modelling to explore whether there are additional hidden patterns within this data.

Further exploration of the dataset should focus on understanding the variability and distribution of low-importance features. Additionally, consideration should be given to feature engineering, including the creation of new features or interactions to better capture the influence of less important variables. For instance, combining passenger numbers with specific bus stops or times could reveal hidden patterns. Model evaluation should involve continuous assessment and comparison of performance with and without certain features to determine their true impact on predictions.

In conclusion, this analysis sheds light on enhancing the efficiency, reliability, and sustainability of public transportation systems. Equipped with data-driven insights and predictive models, transit agencies and policymakers have the tools to navigate towards optimized bus operations, improved service quality, and enhanced passenger experiences.

REFERENCES

Cats, O., Jenelius, E., & Börjesson, M. (2016). The impact of bus stops attributes on accessibility and waiting time. Transportation Research Part A: Policy and Practice, 85, 165-180.

Ding, C., Jiang, X., & Yu, J. (2020). Exploring the impact of transit smart card on bus dwell time: Evidence from Nanjing, China. Journal of Transport Geography, 83, 102669.

Litman, T. (2019). Innovative Transport Pricing: The Next Frontier. Victoria Transport Policy Institute.

Peters, A., & Strathman, J. G. (2018). Evaluating the impacts of smart card and smartphone mobile payment applications on bus transit dwell time. Journal of Public Transportation, 21(3), 1-19.

Schmöcker, J. D., & Quddus, M. A. (2010). Analysis of boarding and alighting activities at bus stops: A case study in London. Transportation Research Part A: Policy and Practice, 44(10), 804-815.

Wei, Y., & Young, R. (2019). Impact of bus stop consolidation on passenger walking distance and wait time. Transportation Research Record, 2673(3), 216-225.

Breiman, L. (2001). Random Forests. Machine Learning, 45(1), 5–32.

Chen, T., & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (pp. 785–794).

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). An Introduction to Statistical Learning with Applications in R. Springer.

Spanton, R. (2023). Polynomial Regression: An Introduction | Built In. [online] builtin.com. Available at: https://builtin.com/machine-learning/polynomial-regression.

Ng, A. (1998). Preventing "Overfitting" of Cross-Validation Data. Proceedings of the Fourteenth International Conference on Machine Learning.

Moreno-Torres, J., Sáez, J. A., & Herrera, F. (2012). Study on the Impact of Partition-Induced Dataset Shift on k-Fold Cross-Validation. IEEE Transactions on Neural Networks and Learning Systems, 23, 1304-1312. DOI: 10.1109/TNNLS.2012.2199516.

APPENDIX

# Basic imports

```
import numpy as np

import pandas as pd

import statsmodels.api as sm

from sklearn.preprocessing import LabelEncoder

import missingno as msno # for missing values

import matplotlib.pyplot as plt

import seaborn as sns

sns.set_style('darkgrid')
```

Code A1: Script for reading, renaming, processing ticketer and ticket type (similar data),GPS data and Descriptive statistics alone with GPS data. Figure 1-4.

#tickter data and renaming columns

```
Concess = pd.read_excel('Route_7_Concessionary_data_Ticketer.xlsx',
'Concessionary_Data')

Concess = Concess.rename(columns={'Ticket Time': 'Transaction Time',
'Ticket Class': 'Ticket Description','Journey number': 'Trip Number'})
```

#ticket type data and renaming columns

```
ticketer = pd.read_csv('ticketData.csv',encoding='Latin1')

ticketer = ticketer.rename(columns={'Op Date': 'Date', 'Bus Stop
Name': 'Bus Stop'}) # renaming the columns

# processing the date column for common formatting
```

```
ticketer['Date'] = pd.to_datetime(ticketer['Date'], format='%d/%m/%Y')
# Convert the column to datetime objects

ticketer['Date'] = ticketer['Date'].dt.strftime('%Y-%m-%d')# Format
datetime objects with day and month swapped
```

#GPS data read and formatting

```
GPS = pd.read_excel('Route_7_GPS_Data.xlsx')

GPS = GPS.rename(columns={'Rec Vehicle No': 'Bus number', 'Unnamed:
5': 'Timing Point', 'Unnamed: 8': 'Bus Stop'})
```

#Descriptive Statistics

```
ticketer.info()

ticketer.describe(include='object').T # for categorical variables

ticketer.describe().T # for numerical variables.

GPS.info()

GPS.describe().T # for numerical variables.

GPS.describe(include='object').T # for numerical variables
```

Code A2: Python Script for Generating Figures 5 and 6.

```
df=ticket_data.copy()
```

# bar chart figure 5

```
# Group by 'Date' and sum the 'Passengers' for each day

daily_passengers =
df.groupby('Date')['Passengers'].sum().reset_index()

# Plot a bar chart of the total passengers for each day

plt.figure(figsize=(8, 6))
plt.bar(daily_passengers['Date'], daily_passengers['Passengers'], color='skyblue')

plt.xlabel('Date')

plt.ylabel('Total Passengers')

plt.title('Total Passengers per Day')

plt.xticks(rotation=45)  # Rotate date labels for better readability

plt.tight_layout()  # Adjust layout to ensure everything fits without overlap

plt.show()

# Pie chart for passengers' ratio

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

plt.pie(daily_passengers['Passengers'], labels=daily_passengers['Date'], autopct='%1.1f%%', startangle=140)

plt.title('Passenger Distribution by Day')

plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.

# Display the pie chart

plt.tight_layout()

plt.show()
```

Code A3: Python script for generating Figure 7.

```
ticket_counts = df['Ticket Class Description'].value_counts()

# Create a bar plot

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

bars = ticket_counts.plot(kind='bar', color='skyblue')
```

```
# Add counts on top of bars

for bar in bars.patches:

    plt.annotate(format(bar.get_height(), '.0f'),

                 (bar.get_x() + bar.get_width() / 2,

                  bar.get_height()),

                 ha='center', va='center',

                 xytext=(0, 5),

                 textcoords='offset points')

plt.xlabel('Ticket Class Description')

plt.ylabel('Count')

plt.title('Counts of Ticket Class Description')

plt.xticks(rotation=45)  # Rotate x-axis labels for better readability

plt.tight_layout()  # Adjust layout to prevent labels from getting cut
off.

plt.show()
```

Code A4: Script for generating Figure 8

```
# Calculating transaction time for each ticket (T_time column).

# Make a copy of the original DataFrame

df3 = df.copy()

# Convert 'Transaction Time' column to datetime

df3['Transaction Time'] = pd.to_datetime(df3['Transaction Time'])

# Calculate the time difference between consecutive rows

df3['T_time'] = df3.groupby(['Bus Stop', 'Date', 'Trip
Number'])['Transaction Time'].diff().dt.total_seconds().fillna(0)
```

```
# Convert the 'Transaction Time' column back to time format

df3['Transaction Time'] = df3['Transaction Time'].dt.time

# Print the DataFrame head to check the results

df3.head()
```

# Checking distribution in transaction time column.

```
Df3['T_time'].describe().T
```

# Checking anomalies in transaction time column.

```
# Calculate mean and standard deviation

mean_T_time = df3['T_time'].mean()

std_T_time = df3['T_time'].std()

# Define threshold for anomalies

threshold =  std_T_time

# Find anomalies

anomalies = df3[df3['T_time'] > mean_T_time + threshold]

# Print the number of anomalies

print("Number of anomalies:", len(anomalies))

print("Time of anomalies:")

print(anomalies[['Bus Stop','Transaction Time','T_time']])
```

# Removing anomalies from transaction time column.

```
# Calculate mean and standard deviation

mean_T_time = df3['T_time'].mean()

std_T_time = df3['T_time'].std()

# Define threshold for anomalies
```

```
threshold = std_T_time

# Find indices of anomalies

anomalies = df3[df3['T_time'] > mean_T_time + threshold].index

# Drop rows with anomalies

df3.drop(index=anomalies, inplace=True)

# Print the updated DataFrame information

print("Number of rows after dropping anomalies:", len(df3))
```

# Average ticket time

```
# Group by 'Ticket Class Description' and calculate average time

total_time_and_count_by_class = df3.groupby('Ticket Class
Description')['T_time'].agg(['sum', 'count']).abs().reset_index()

total_time_and_count_by_class.rename(columns={'sum': 'Total Time',
'count': 'Ticket Count'}, inplace=True)

# Calculate average time

total_time_and_count_by_class['Average Time'] =
total_time_and_count_by_class['Total Time'] /
total_time_and_count_by_class['Ticket Count']

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

bars = plt.barh(total_time_and_count_by_class['Ticket Class
Description'], total_time_and_count_by_class['Average Time'],
color='skyblue')

plt.ylabel('Ticket Class Description')

plt.xlabel('Average Time (seconds)')

plt.title('Average Transaction Time by Ticket Class Description')

# Add the average time, total time, and ticket count labels on the
bars
```

```
for index, bar in enumerate(bars):

    plt.text(

        bar.get_width(),  # x-coordinate position of the text

        bar.get_y() + bar.get_height() / 2,  # y-coordinate position
of the text

        f'Avg: {bar.get_width():.2f} sec\nTotal:
{total_time_and_count_by_class["Total Time"].iloc[index]} sec\nCount:
{total_time_and_count_by_class["Ticket Count"].iloc[index]}',  # text
to display (formatted to 2 decimal places)

        va='center',  # vertical alignment

        ha='left',  # horizontal alignment

        fontsize=10  # font size

    )

plt.tight_layout()  # Adjust layout to ensure everything fits without
overlap.

plt.show()
```

Code A5:

# Script for checking anomalies in Actual Arrival and Departure time.

```
GPS_df=GPS.copy()

# Function to check anomalies

def check_anomalies(column):

    anomalies = []

    for value in column:

        value_str = str(value)
        if len(value_str) != 8:  # Check if the length of the
string representation is not equal to 'HH:MM:SS'

            anomalies.append(value_str)

    return anomalies



# Check anomalies in 'Actual Arrival Time'

arrival_anomalies = check_anomalies(GPS_df['Actual Arrival
Time'])

# Check anomalies in 'Actual Departure Time'

departure_anomalies = check_anomalies(GPS_df['Actual Departure
Time'])

print("Anomalies in Actual Arrival Time:", arrival_anomalies)

print("Anomalies in Actual Departure Time:",
departure_anomalies)
```

# Script for dealing with anomalies in Actual Arrival and Departure time.

```
# Function to check anomalies and convert datetime to time
format

def convert_anomalies_to_time_format(df):

    for col in ['Actual Arrival Time', 'Actual Departure Time']:

        for idx, value in enumerate(df[col]):

            value_str = str(value)

            if len(value_str) != 8:  # Check if the length of
the string representation is not equal to 'HH:MM:SS'
```

```
                df.at[idx, col] = value_str.split()[1]  #
Extract only the time part

# Create a copy of the original DataFrame to avoid modifying it

GPS_df = GPS.copy()



# Convert anomalies in 'Actual Arrival Time' and 'Actual
Departure Time' columns to time format

convert_anomalies_to_time_format(GPS_df)

# Print the modified DataFrame

print(GPS_df)
```

# Script for calculating 'Dwell time'

```
GPS_df['Actual Arrival Time'] = pd.to_datetime(GPS_df['Actual
Arrival Time'], format='%H:%M:%S')

GPS_df['Actual Departure Time'] = pd.to_datetime(GPS_df['Actual
Departure Time'], format='%H:%M:%S')

# Calculate the time spent at each stop in seconds

GPS_df['Dwell time'] = GPS_df.apply(lambda row: (row['Actual
Departure Time'].hour * 3600 + row['Actual Departure
Time'].minute * 60 + row['Actual Departure Time'].second) -
(row['Actual Arrival Time'].hour * 3600 + row['Actual Arrival
Time'].minute * 60 + row['Actual Arrival Time'].second) if
row['Actual Departure Time'] >= row['Actual Arrival Time'] else
(row['Actual Departure Time'].hour * 3600 + row['Actual
Departure Time'].minute * 60 + row['Actual Departure
Time'].second + 24*3600) -
(row['Actual Arrival Time'].hour * 3600 + row['Actual Arrival
Time'].minute * 60 + row['Actual Arrival Time'].second), axis=1)
```

```
# Display the dataframe

print(GPS_df)
```

# Script for dwell time boxplot

```
plt.boxplot(GPS_df['Dwell time'])

plt.show()
```

# Script for generating Figure 9

```
# Group by 'Bus Stop' and sum the 'Dwell
time'total_time_spent_by_location = GPS_df.groupby('Bus
Stop')['Dwell time'].sum().sort_values(ascending=False)

# Display the result

print(total_time_spent_by_location)
```

# Code for popular tickets at Bus Stops

```
# Group by 'Bus Stop Name' and count the occurrences of each
'Ticket Class Description'

popular_tickets = ticketer.groupby('Bus Stop')['Ticket Class
Description'].agg(lambda x:
x.value_counts().index[0]).reset_index()

print(popular_tickets)
```

# Code for generating Figure 10

```
#Group by ('Bus Stop Name')

passengers_at_stops = df.groupby('Bus
Stop')['Passengers'].sum().sort_values(ascending=False)

print(passengers_at_stops)
```

Code A6: Script for generating Figure 11

```
# Convert 'Transaction Time' to datetime format

tkt['Transaction Time'] = pd.to_datetime(tkt['Transaction
Time'])

# Extract hour from 'Transaction Time'

tkt['Hour'] = tkt['Transaction Time'].dt.hour

# Group by date and hour, then sum the passenger count

busiest_hours = tkt.groupby(['Date',
'Hour'])['Passengers'].sum().reset_index()

# Find the hour with the highest passenger count for each date

busiest_hours =
busiest_hours.loc[busiest_hours.groupby('Date')['Passengers'].id
xmax()]

# Visualize the busiest hours

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

bars = plt.barh(busiest_hours['Date'].astype(str) + ' ' +
busiest_hours['Hour'].astype(str) + ':00',
busiest_hours['Passengers'], color='skyblue')

plt.ylabel('Date and Hour')

plt.xlabel('Passenger Count')

plt.title('Busiest Hours with Highest Passenger Count for Each
Date')

# Add the passenger count labels inside the bars

for index, bar in enumerate(bars):

    plt.text(
        bar.get_width() / 2,  # x-coordinate position of the
text (inside the bar)

        bar.get_y() + bar.get_height() / 2,  # y-coordinate
position of the text (inside the bar)

        f'{int(bar.get_width())}',  # text to display (passenger
count)

        ha='center',  # horizontal alignment

        va='center',  # vertical alignment

        fontsize=10  # font size

    )

plt.tight_layout()

plt.show()
```

Code A7:Script for joining and preprocessing the tables.

#Joining the tickter and ticket-types table.

```
# Bring the Date columns in common format

ticketer['Date'] = pd.to_datetime(ticketer['Date'])

Concess['Date'] = pd.to_datetime(Concess['Date'])

# Perform left join using merge function

ticket_data = pd.merge(ticketer,Concess,on=['Ticket
Description','Transaction Time','Date','Trip Number'],
how='inner')
```

# Dropping if there any duplicate rows

```
ticket_data.drop_duplicates()
```

# Merging GPS and ticket_data on common columns

```
GPS_df['Date'] = pd.to_datetime(GPS_df['Date'])

model_data = pd.merge(ticket_data, GPS_df, on=['Bus Stop','Bus
number','Date'], how='inner')


# Dropping column that is not necessary, redundant or no meaning

model_data.drop(columns=['Route number', 'Operator', 'Number',
'Value of Ride','Custom Formula', 'ETM Route Code', 'Route_No',
'Route_Desc'], inplace=True)

# Divide the ‘Ticket Class Description’ in 5 columns: Cash,
TOTO, Mobile App, Keycard and Others

# Function to map the ticket classes to the desired categories

def map_category(ticket_class):

    if ticket_class in ['Adult', 'Child']:

        return 'Cash'

    elif ticket_class == 'Autoscan':

        return 'TOTO'

    elif ticket_class == 'Smart':

        return 'Keycard'

    elif ticket_class == 'QR':

        return 'Mobile App'

    elif ticket_class in
['Passes','ALL_ETM','OthTkt','CSS','Excess','Week']:

        return 'Others'

    else:
        return None

# Create new columns based on the mapping

model_data['Cash'] = model_data['Ticket Class
Description'].apply(lambda x: x if map_category(x) == 'Cash'
else None)

model_data['TOTO'] = model_data['Ticket Class
Description'].apply(lambda x: x if map_category(x) == 'TOTO'
else None)

model_data['Keycard'] = model_data['Ticket Class
Description'].apply(lambda x: x if map_category(x) == 'Keycard'
else None)

model_data['Mobile App'] = model_data['Ticket Class
Description'].apply(lambda x: x if map_category(x) == 'Mobile
App' else None)

model_data['Others'] = model_data['Ticket Class
Description'].apply(lambda x: x if map_category(x) == 'Others'
else None)

print(model_data)
```

# Encoding columns for modelling

```
from sklearn.preprocessing import LabelEncoder

# Define columns to encode

columns_to_encode = ['Ticket Class Description','Ticket
Description', 'TOTO', 'Cash', 'Keycard', 'Mobile App', 'Others',

                     'Timing Point', 'Bus Stop','Location
No','Planned Arrival Time','Planned Departure Time',
                    'Actual Arrival Time','Actual Departure
Time']

# Create LabelEncoder instance

label_encoder = LabelEncoder()

# Apply LabelEncoder to each specified column

for column in columns_to_encode:

    model_data[column] =
label_encoder.fit_transform(model_data[column])
```

# See correlation of the data according to ‘Dwell time’

```
# Calculate the correlation matrix

correlation_matrix = model_data.corr()

# Extract and sort the correlation of all columns with 'Dwell
time'

dwell_time_correlation = correlation_matrix['Dwell
time'].sort_values(ascending=False)

# Plot the sorted correlation values

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

sns.barplot(x=dwell_time_correlation.values,
y=dwell_time_correlation.index, palette="viridis")

# Add titles and labels

plt.title('Correlation of Features with Dwell Time')

plt.xlabel('Correlation Coefficient')

plt.ylabel('Features')

# Show the plot
```

```
plt.show()
```

Code A8: Script for Polynomial Regression

```
from sklearn.preprocessing import PolynomialFeatures

import statsmodels.api as sm

# Define the predictor variables (X) and the target variable (y)

X = model_data[['Mobile App', 'TOTO', 'Cash', 'Keycard',
'Others', 'Passengers','Timing Point','Origin Stage',
'Direction', 'Bus Stop','Actual Arrival Time', 'Actual Departure
Time']]

y = model_data['Dwell time']

# Generate polynomial features

poly = PolynomialFeatures(degree=2, include_bias=False)

X_poly = poly.fit_transform(X)

# Fit the polynomial regression model

model_poly = sm.OLS(y, X_poly).fit()

# Print the summary of the polynomial regression model

print(model_poly.summary())
```

Code A9:Script for nested machine learning modelling.

```
from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression

from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor

from sklearn.neighbors import KNeighborsRegressor

from sklearn.metrics import mean_squared_error,
mean_absolute_error, r2_score

# Split data into train and test sets

X = model_data[['Mobile App' ,'TOTO', 'Cash', 'Keycard',
'Others', 'Passengers','Timing Point', 'Origin Stage',
'Direction', 'Bus Stop', 'Actual Arrival Time', 'Actual
Departure Time']]

y = model_data['Dwell time']

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

# Initialize and train the models

models = {

    "Linear Regression": LinearRegression(),

    "Random Forest Regression": RandomForestRegressor(),

    "XGBoost Regression": XGBRegressor(),

    "KNeighbor Regression": KNeighborsRegressor(n_neighbors=5)

}

 for name, model in models.items():

    model.fit(X_train, y_train)

    y_pred = model.predict(X_test)

    mse = mean_squared_error(y_test, y_pred)

    rmse = np.sqrt(mse)
    mae = mean_absolute_error(y_test, y_pred)

    r2 = r2_score(y_test, y_pred)

    print(f"{name} Metrics:")

    print(f"  Mean Squared Error: {mse}")

    print(f"  Root Mean Squared Error: {rmse}")

    print(f"  Mean Absolute Error: {mae}")

    print(f"  R-squared: {r2}")

    print()
```

Code A10: Code for cross validation.

```
import pandas as pd

from sklearn.model_selection import cross_val_score

from sklearn.model_selection import train_test_split

from sklearn.metrics import mean_squared_error,
mean_absolute_error, r2_score

from sklearn.ensemble import RandomForestRegressor

from xgboost import XGBRegressor

from sklearn.neighbors import KNeighborsRegressor

import numpy as np

# Split data into train and test sets

X = model_data[['Mobile
App','TOTO','Cash','Keycard','Others','Passengers','Timing
Point','Origin Stage','Direction','Bus Stop','Actual Arrival
Time','Actual Departure Time']]
```

```
y = model_data['Dwell time']

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)

# Define regression models

models = {

    "Random Forest Regression": RandomForestRegressor(),

    "XGBoost Regression": XGBRegressor(),

    "KNeighbor Regression": KNeighborsRegressor(n_neighbors=5)

}

# Perform k-fold cross-validation for each model

for name, model in models.items():

    # Fit the model to the training data

    model.fit(X_train, y_train)

    # Perform 5-fold cross-validation

    scores = cross_val_score(model, X_train, y_train, cv=5,
scoring='neg_mean_squared_error')

    # Convert negative MSE scores to positive

    scores = -scores

    # Calculate mean MSE and standard deviation

    mean_mse = scores.mean()

    std_mse = scores.std()

    # Calculate RMSE

    rmse = np.sqrt(mean_mse)
    # Calculate MAE

    mae = mean_absolute_error(y_test, model.predict(X_test))

    # Calculate R-squared

    r2 = model.score(X_test, y_test)

    # Print results

    print(f"{name}:\n")

    print(f"\tMean Squared Error: {mean_mse:.2f}")

    print(f"\tRoot Mean Squared Error: {rmse:.2f}")

    print(f"\tMean Absolute Error: {mae:.2f}")

    print(f"\tR-squared: {r2:.2f}\n")
```

Code A11: Feature Importance – Random Forest, XGBOOST, KNN

# Random Forest

```
from sklearn.ensemble import RandomForestRegressor

from sklearn.model_selection import train_test_split

# Define feature names

feature_names = ['Mobile App', 'TOTO', 'Cash', 'Keycard',
'Others', 'Passengers','Timing Point', 'Origin Stage',
'Direction', 'Bus Stop','Actual Arrival Time', 'Actual Departure
Time']

# Split data into train and test sets

X = model_data[feature_names]

y = model_data['Dwell time']
```

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

# Initialize and fit the Random Forest model

rf = RandomForestRegressor()

rf.fit(X_train, y_train)


# Get feature importances from the model

rf_feature_importances = rf.feature_importances_


# Creating a DataFrame for better visualization

rf_importances_df = pd.DataFrame({

    'Feature': feature_names,

    'Importance': rf_feature_importances

}).sort_values(by='Importance', ascending=False)


print("Random Forest Feature Importances:")

print(rf_importances_df)
```

# XGBOOST

```
from xgboost import XGBRegressor

from sklearn.model_selection import train_test_split

# Define feature names

feature_names = ['Mobile App', 'TOTO', 'Cash', 'Keycard', 
'Others', 'Passengers','Timing Point', 'Origin Stage',
'Direction', 'Bus Stop','Actual Arrival Time', 'Actual Departure
Time']

# Split data into train and test sets

X = model_data[feature_names]

y = model_data['Dwell time']

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

# Initialize and fit the XGBoost model

xgb = XGBRegressor()

xgb.fit(X_train, y_train)

# Get feature importances from the model

xgb_feature_importances = xgb.feature_importances_

# Creating a DataFrame for better visualization

xgb_importances_df = pd.DataFrame({

    'Feature': feature_names,

    'Importance': xgb_feature_importances

}).sort_values(by='Importance', ascending=False)

print("XGBoost Feature Importances:")

print(xgb_importances_df)
```

# KNN

```
from sklearn.neighbors import KNeighborsRegressor

from sklearn.model_selection import train_test_split

from sklearn.inspection import permutation_importance
```

```
# Define feature names

feature_names = ['Mobile App', 'TOTO', 'Cash', 'Keycard',
'Others', 'Passengers','Timing Point', 'Origin Stage',
'Direction', 'Bus Stop','Actual Arrival Time', 'Actual Departure
Time']

# Split data into train and test sets

X = model_data[feature_names]

y = model_data['Dwell time']

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

# Initialize and fit the KNN model

knn = KNeighborsRegressor()

knn.fit(X_train, y_train)

# Calculate permutation importance

knn_perm_importance = permutation_importance(knn, X_train,
y_train, n_repeats=10, random_state=42)

# Creating a DataFrame for better visualization

knn_perm_importances_df = pd.DataFrame({

    'Feature': feature_names,

    'Importance': knn_perm_importance.importances_mean

}).sort_values(by='Importance', ascending=False)

print("KNN Permutation Importances:")

print(knn_perm_importances_df)
```