University of Padova

Department of Mathematics "Tullio Levi-Civita"

Master Thesis in Data Science

From Data Cleaning to Predictive Models: A Strategic Approach to Analyzing Bus and Ship Trajectories

Supervisor
Prof. Lamberto Ballan
University of Padova

Co-supervisor
Prof. Esteban Zimanyi
Université libre de Bruxelles

Master Candidate
Satria Bagus Wicaksono

Student ID
2105082

Academic Year
2023-2024

I dedicate this thesis to my mom, my partner, my dad, and my brother. Your unwavering encouragement and love have meant the world to me and helped me through every step of this journey.

Abstract

Trajectory data holds significant potential for advanced analytics, driven by the development of sophisticated models and tools. This thesis aims to unlock this potential by addressing the challenges inherent in trajectory data, meticulously preprocessing the data, and performing predictive analysis. We present two use cases: the primary use case focuses on predicting battery consumption in electric buses, while the secondary use case involves predicting ship trajectories. These contributions have important implications for enabling intelligent transportation systems (ITS) and promoting greener urban transportation, as well as enhancing maritime navigation and safety.

In the primary use case, our research thoroughly examines and preprocesses the trajectory data, employing methods such as map matching, trajectory segmentation, and linear interpolation. Additionally, we enrich the trajectories with semantic information to create a clean and usable dataset for prediction tasks. In the predictive analysis, we compare a baseline model with a deep learning model, with the latter outperforming the former, achieving an RMSE of 1.31 kWh and an R^2 score exceeding 80%.

In the secondary use case, we apply similar preprocessing techniques, including the detection of temporal and spatial gaps in the trajectory data. The predictive analysis reveals that the best prediction accuracy is achieved with a 30-minute observation window, predicting 5 minutes into the future, resulting in a MAPE of 1.2% and a mean distance error of 0.35 kilometers from the actual trajectories.

Contents

Listing of figures

Listing of tables

Listing of acronyms

POI ..... Points of Interest

RNN ..... Recurrent Neural Networks

GPS ..... Global Positioning System

AIS ..... Automatic Identification System

AVL ..... Automatic Vehicle Location

VTS ..... Vessel Traffic Services

MMSI ..... Maritime Mobile Service Identity

HMM ..... Hidden Markov Model

ARIMA ..... Autoregressive Integrated Moving Average

SAX ..... Symbolic Aggregation Approximation

LSTM ..... Long Short-Term Memory

MLP ..... Multilayer Perceptron

MSE ..... Mean Squared Error

RMSE ..... Root Mean Squared Error

MAE ..... Mean Absolute Error

MAPE ..... Mean Absolute Percentage Error

ID ..... Identification Number

kWh ..... kilowatt-hours

GTFS ..... General Transit Feed Specification

RFE ..... Recursive Feature Elimination

SOG ..... Speed Over Ground

COG ..... Course Over Ground

ROT ..... Rate of Turn

1 Introduction

1.1 Movement and Trajectories

Movement data is ubiquitous, encompassing human movement, vehicle movement, and animal movement. This data provides valuable insights into patterns or phenomena within movement. Furthermore, movement data can be extensive, consisting of numerous smaller segments. For instance, examining a person's movement over an entire day reveals several smaller movements, such as trips to the supermarket or going for a run. These smaller sets of movements, or segments, can be defined as trajectories [6].

To understand these smaller segments better, we need a clear definition. Trajectory data is defined as a series of occurrences that match the time-referenced, recorded locations of moving objects [7]. This means a trajectory contains spatial information about the location of an object and temporal information that is referenced by the spatial data, providing a comprehensive view of the movement. Integrating both spatial and temporal information in trajectory data offers an overview of the evolution of object movement, which is highly valuable in various real-world applications.

The real-world applications of trajectory data are numerous. One notable example is in public transportation. By recording the movement of a bus from start to finish, we can obtain information regarding the bus's trajectories, including stops and variations in routes depending on the day. Another example is tracking ship movement; we can monitor a ship throughout

its journey, knowing its location at any given time. However, modeling trajectory data solely based on spatial and temporal information has its limitations.

One of the primary limitations of raw trajectory data, which includes only spatial and temporal information, is its lack of context. According to Ferrero et al. [8], adding information about the locations visited by an object, referred to as Points of Interest (POIs), can enhance the contextual understanding of the trajectory. Additionally, there are challenges in balancing the accuracy of data with privacy concerns. The abundance of spatial and temporal information can be maliciously exploited by attackers to extract sensitive information, such as home addresses, personal preferences, and even medical data [9]. Conversely, the accuracy of this data is essential for providing precise analyses. Nevertheless, new techniques continue to be developed to address these issues effectively.

1.2 Analysis and Prediction on Trajectory Data

With the limitations and difficulties inherent in representing trajectory data, specialized tools have been developed to facilitate trajectory data analysis. One such example is MobilityDB [10], a PostgreSQL extension that allows for the analysis and processing of spatio-temporal data. Additionally, the development of MobilityDB has led to the creation of other trajectory analysis tools, such as PyMEOS [11], which enables analysis and preprocessing in a Python environment. Similarly, tools like MovingPandas [12] provide comprehensive functionalities for performing trajectory data analysis.

These new technologies have significantly simplified the processes of analyzing and preprocessing trajectory data, thereby enhancing the insights and analytics derived from such data. Advanced analysis and preprocessing capabilities are now more accessible, allowing for efficient transformation and representation of trajectory data. For instance, interpolation of trajectory data based on location and time can be easily performed using these tools. Visualization of trajectory data has also become more straightforward, facilitating better understanding and interpretation. Moreover, the advanced preprocessing capabilities of these tools enable the integration of trajectory data with other advanced techniques, such as deep learning.

Deep learning has gained considerable popularity in recent years, driven by advancements in techniques and increased computational power. This has made it possible to uncover hidden patterns within trajectory data. Given the sequential nature of trajectory data, deep learning techniques are well-suited for analyzing such data. Techniques like Recurrent Neural Networks (RNNs) [13], which can learn patterns from sequences, are particularly effective. By

combining these tools and techniques, we can perform more in-depth analysis and make predictions based on trajectory data.

When performing predictions using deep learning techniques, trajectory data typically needs to be preprocessed first [14]. This preprocessing is crucial as it prepares the data for analysis by performing preliminary steps and transforming the data into a format suitable for deep learning models. Preprocessing steps may involve representing the trajectory data in different forms, such as aggregating trajectories or downsampling them. This comprehensive preparation ensures that the deep learning models can accurately interpret and analyze the data, ultimately leading to more reliable and insightful predictions. This capability opens up numerous real-world applications, enabling advancements in fields such as transportation, healthcare, and urban planning.

One real-world application involves analyzing aviation data. For example, MobilityDB has been used to analyze busy hours for flights or congested areas in the sky [15], demonstrating the importance of these tools in data analysis. In this thesis, a specific application focuses on the intelligent transportation system, predicting battery consumption from bus trajectory data. This is crucial for developing green transportation systems and optimizing battery usage. Despite these advancements in tools and techniques, there are still challenges that need to be addressed in the future.

1.3 Challenges in Analyzing and Predicting Trajectory Data

Analyzing and predicting trajectory data present significant challenges, even with the advancements in techniques and tools. One of the primary challenges is the varying characteristics of trajectory data. These variations can include spatial resolution (ranging from fine to coarse), spatial dimensions (2D or 3D), temporal resolution (from sparse to frequent sampling), sampling intervals (regular or irregular), and the overall size of the data [14].

These differences in characteristics add complexity to the preprocessing of trajectory data. Applying the right techniques and tools to handle these variations is crucial. For instance, managing spatial dimensions requires careful representation of each spatial information type, such as Point or LineString. Additionally, the characteristics of the data influence how it should be cleaned. Cleaning trajectory data for vehicle movement differs significantly from cleaning trajectory data for human movement. Proper preprocessing is essential as it forms the foundation

for building accurate predictive models.

In predicting trajectory data, correctly modeling the problem is critical to achieving reliable predictions. For example, trajectory prediction tasks often involve forecasting the next sequence based on historical data. Accurately defining the problem helps in selecting appropriate techniques for model development. In real-world scenarios, the challenges vary depending on the specific task and context of the trajectory data.

In this thesis, the *primary use case* involves predicting battery consumption from electric bus trajectory data. This task presents several challenges in the data prerprocessing, including Global Positioning System (GPS) inaccuracies, the sparsity of trajectory data (recorded every 30 seconds), handling missing values and adding context from the raw trajectories. These issues must be carefully addressed to develop an accurate predictive model. Additionally, further challenges arise in ensuring the quality and reliability of the model predictions.

The *secondary use case* involves trajectory prediction for ship movement data, particularly using the Automatic Identification System (AIS) dataset. This use case faces similar challenges, such as GPS inaccuracies and data cleaning requirements. These issues are common in the analysis and prediction of trajectory data, necessitating a strategic approach to address them effectively.

1.4 Addressing the Challenges and Contributions

In this section, we will discuss how to address the challenges mentioned in the previous section, focusing on those presented in the primary and secondary use cases. We will explore specific strategies for data preprocessing and predictive modeling, highlighting our contributions to the field. By systematically addressing these challenges, we aim to demonstrate the effectiveness and versatility of our approach in overcoming them.

One of the challenges in our primary use case is preprocessing the raw bus trajectories in the form of Automatic Vehicle Location (AVL) data. To address this, we will define a data pipeline strategy to extract the necessary information from the bus trajectories before feeding it into the prediction model. This data pipeline will include data cleaning and preprocessing techniques such as map matching, linear interpolation, and stop detection. Additionally, we will augment the data with contextual information such as weather and elevation data. It is also necessary to aggregate the trajectory data to ensure it is comprehensive and ready for model building.

After preprocessing the data, our next challenge is to build an accurate predictive model from our trajectory data. To address this, we will employ a strategic approach that begins with

developing a baseline model, followed by iterative improvements to enhance prediction performance. We will evaluate the predictions using metrics such as MSE and standard deviation to assess both the performance and consistency of our models. This systematic approach ensures that we can refine our models effectively, leading to more reliable predictive outcomes and laying the groundwork for our key contributions.

Building on this foundation, our contribution through this approach to predicting electric consumption from electric bus trajectory data includes the creation of a comprehensive data pipeline for processing bus trajectory data within public transportation networks. Additionally, we will develop a predictive model to forecast battery consumption based on this data. This work will aid in creating greener cities and increasing the efficiency of public transportation networks by optimizing planning and routing based on forecasted battery consumption. Ultimately, this will enhance our contributions to intelligent transportation systems.

Additionally, we are including a secondary use case to demonstrate the versatility of our approach across different types of trajectory data. This ensures that our methods are robust and adaptable to various real-world scenarios. The challenges in this use case involve building a comprehensive data pipeline to process our AIS dataset, which includes data cleaning and preprocessing.

Building on the insights gained from addressing these challenges, additional contributions from both use cases will include the development of predictive models for these datasets and the integration of preprocessing and modeling techniques into a cohesive method for trajectory prediction on dense trajectory data. Overall, these two use cases will contribute to advancing intelligent transportation systems.

1.5 Structure of the Thesis

In this section, we will outline the structure of the thesis, which is organized to address the challenges identified in previous sections and to present our systematic approach to solving them. By detailing the organization of each chapter, we aim to provide a clear road-map of the research process, methodologies, and findings. This structured overview will help readers understand the logical progression of our work and the contributions made through our research.

Chapter 2 will review the necessary literature relevant to addressing the challenges and building our solution. This chapter will begin with an overview of trajectory data, followed by the techniques required for cleaning and processing it, and conclude with methods for developing predictive models.

Building on this foundation, Chapter 3 will provide an in-depth discussion of our primary use case. This includes a detailed examination of the dataset, the construction of our data pipeline, and the validation of our preprocessed data from the data pipeline.

Continuing from the data pipeline construction, Chapter 4 will describe the process of building the predictive models. We will start from the baseline and progressively improve it, detailing the metrics used for evaluating our predictions and presenting our analysis of the prediction results.

Extending our approach, Chapter 5 will deepen the exploration of our challenges by integrating the secondary use case. This chapter will explain the entire prediction process, from building the data pipeline to assessing the prediction results for this additional scenario.

Finally, Chapter 6 will present the conclusion, summarizing our key findings and reflecting on how we addressed the challenges in both use cases. It will also discuss the broader impact of our findings on intelligent transportation systems and outline potential future work.

2

Literature Review

In this chapter, we will introduce the necessary literature that forms the foundation of our thesis, with a focus on intelligent transportation and maritime navigation. This review explores the methodologies and techniques employed in trajectory data representation, cleaning, prediction, and evaluation, laying the groundwork for the experimental studies presented in subsequent chapters.

The literature review is organized into several sections, each systematically addressing key areas relevant to our research. We begin in Section 2.1, which introduces trajectory data, its various representations, and the types of trajectory data considered in this study. Building on this understanding, Section 2.2 focuses on the methods used for cleaning trajectory data, particularly emphasizing map matching, trajectory segmentation, and the enrichment of trajectories with semantic context.

Transitioning from data cleaning to predictive analysis, Section 2.3 explores the tasks of prediction within trajectory data, with a specific emphasis on intelligent transportation and maritime navigation domains. This section also discusses the methods used for performing these prediction analysis.

Building on this, Section 2.4 examines the application of deep learning techniques, highlighting relevant architectures and their effectiveness in processing trajectory data. To ensure the robustness and accuracy of our models, Section 2.5 outlines the evaluation metrics applied throughout this thesis. Finally, Section 2.6 concludes the chapter by summarizing the key insights from the literature review.

2.1 Trajectory Data

Trajectory data is a fundamental concept in understanding the movement patterns of objects over time and space. According to Spaccapietra et al. [16], a trajectory can be defined as a segment of an object's movement that is delimited by a specific time interval $[t_{\text{begin}}, t_{\text{end}}]$, where it is described as a continuous function from the time interval to space. Additionally, Zheng [2] defines a trajectory as a trace generated by a moving object, represented as chronologically ordered points, with each point comprising both spatial and temporal information.

Spaccapietra et al. [16] further explain that a trajectory has two key facets: the geometric facet and the semantic facet. The geometric facet includes the spatio-temporal recordings during the object's movement, while the semantic facet encompasses the trajectory's characteristics and its application-specific meanings. This combination of spatial, temporal, and semantic information is crucial as it enables more complex analyses of trajectory data, such as understanding movement patterns, optimizing routes, or predicting future movements.

2.1.1 Trajectory Data Representation

After understanding the definition of trajectory data, it is important to explore how trajectory data is represented. According to Spaccapietra et al. [17], there are three primary representations of trajectory data: continuous, discrete, and segmented. Continuous trajectories typically consist of a finite sequence of spatio-temporal information, supplemented by an interpolation function that allows for the retrieval of these details at any given instant within the time interval $[t_{\text{begin}}, t_{\text{end}}]$. In contrast, discrete trajectories are composed of a finite list of spatio-temporal positions without providing continuity in the movement of the object. These continuous and discrete trajectories are common forms of representing trajectory data and are often referred to as raw trajectories.

Segmented trajectories, on the other hand, are defined by a step function that maps the interval $[t_{\text{begin}}, t_{\text{end}}]$ to a finite set of values, with each value defining an annotation or "episode" in the trajectory. Understanding these representations is crucial, as they form the foundation for how trajectory data is structured and analyzed. For instance, the continuous and discrete trajectories, typically contain basic location and time information and serve as the base representation for many types of trajectory data. Additionally, the segmented representation is closely linked to trajectory segmentation techniques and semantic trajectories, providing a foundation for more advanced topics in trajectory analysis. All these representations and definitions are discussed

in detail by Spaccapietra et al. [17]. Building on this foundational understanding of trajectory data representation, we can now introduce and define the types of data utilized in this thesis, which will be explained in the following sections.

2.1.2 AVL Data

An AVL system is used to track vehicle locations and has become a critical component of intelligent transportation systems. In public transportation, particularly for buses, AVL systems are widely employed to monitor and manage bus movements and positions. As explained by Hickman [18], AVL systems in buses operate by communicating their positions at periodic frequencies, such as every 30 or 60 seconds, through polling. In addition to location and timestamp data, AVL systems can capture other relevant information, providing not only real-time tracking but also a comprehensive historical record of movements, enriched with contextual information.

Additionally, as explained by Furth et al. [19] and cited in Gerstle [20], there are two types of AVL data: location-at-time data and time-at-location data. Location-at-time data aligns with Hickman's definition [18], where buses report their location via the Global Positioning System (GPS) at specific times, capturing the bus positions with corresponding timestamps in a periodic manner. Time-at-location data, on the other hand, is collected when a bus passes a particular location, meaning that the timestamps may not be evenly spaced depending on when the bus reaches those locations.

Given these definitions, data from AVL systems, especially location-at-time data, can be treated as trajectory data. Each position point from the AVL data within a given time interval can be considered a discrete trajectory. With the application of an interpolation function, this data can be represented continuously. Furthermore, by employing step functions and mapping trajectory data to a set of values, it can be represented as segmented trajectories. This enables more advanced analytics, such as in the work by Ankit et al. [21], where AVL data was used to predict bus arrival and journey times.

Our primary dataset, which focuses on the battery consumption of a bus, is essentially location-at-time AVL data collected at 30-second intervals. In addition to positional information and timestamps, the dataset includes contextual data differentiated by signal attributes such as battery percentage, battery autonomy, total distance, and cumulative battery consumption. Each signal attribute value provides specific data relevant to its context, which is crucial for analyzing battery usage patterns. For our analysis, we will specifically focus on cumulative battery

consumption. A detailed overview of the data structure will be presented in Chapter 3.

2.1.3 AIS Data

AIS is a vital tracking mechanism that enables vessels to broadcast essential information, such as identification, speed, and course, to other ships and Vessel Traffic Services (VTS) stations [22]. AIS plays a crucial role in enhancing maritime safety by facilitating search and rescue operations and supporting the advancement of sophisticated maritime systems. As outlined by Aremu [5], AIS data is categorized into three primary types: dynamic, static, and voyage-related information.

Dynamic information includes variables that fluctuate throughout a voyage, such as time, position, speed, course, and heading. In contrast, static information comprises attributes that remain constant, such as the Maritime Mobile Service Identity (MMSI), ship type, and other fixed characteristics. Additionally, voyage-related information, including the vessel's destination and estimated time of arrival, is also transmitted.

As described in [5], dynamic information is typically transmitted every 2 to 10 seconds when the vessel is in motion and every 3 minutes when it is anchored. Voyage-related information is broadcast every 6 minutes. This comprehensive dataset offers detailed insights into a vessel's spatial and temporal position, along with supplementary contextual information. Due to its composition, AIS data is often categorized as trajectory data, incorporating both positional and temporal elements. Devogele et al. [23] provide a detailed methodology for processing raw AIS position data into structured trajectories, highlighting the importance of converting raw data into usable trajectory information. This processed trajectory data is essential for numerous applications, including trajectory prediction analysis, which has been explored in a variety of studies [24, 25, 26].

In this study, we employ a secondary dataset comprising AIS data from Danish Maritime Authority [27], the structure of which is detailed in Table A.1 in the Appendix. The dataset conforms to the standard AIS attributes discussed earlier, providing a solid foundation for our analysis. However, not all available information from the AIS dataset will be utilized in our analytics. In Chapter 5, we will detail how specific elements of this dataset are incorporated into our analysis and prediction tasks.

2.2 Trajectory Data Cleaning and Preprocessing

After determining how we will represent our data and understanding the specific type of our dataset, it is essential to establish the foundation and techniques for preprocessing and cleaning our trajectory data. Proper preprocessing is crucial because the quality of the trajectory data directly impacts the accuracy of the subsequent analysis, such as movement pattern recognition and prediction tasks.

From a broader perspective, Parent et al. [28] describe the process of transforming raw trajectories into more manageable and reliable representations. This process includes cleaning raw trajectory data, performing map matching, and compressing trajectory data, offering a general overview of the key steps involved in preprocessing and cleaning. Various studies have proposed different approaches to trajectory data cleaning, each tailored to the specific characteristics of the trajectories and the intended applications.

For instance, addressing inaccuracies in positioning systems like GPS, Marketos et al. [1] emphasize the need for both basic and advanced cleaning techniques. They discuss methods such as applying thresholds to filter out erroneous data points, alongside more sophisticated solutions like the Kalman Filter [29], which corrects these points by estimating their true positions. In cases where trajectories are influenced by external factors such as road networks, map matching becomes an indispensable technique. This method corrects errors by aligning the trajectories with the appropriate paths, a topic that will be explored in detail in subsequent sections.

Another approach to trajectory data cleaning is introduced by Li et al. [30], who propose a data-driven method known as the historical trajectory point cloud, offering a novel solution for trajectory cleaning. Complementing this, Zheng [2] presents a comprehensive overview of trajectory data preprocessing techniques, including trajectory segmentation, noise filtering, and stay point detection. Together, these methods establish a robust framework for effectively handling trajectory data, ensuring that the cleaned data is accurate and ready for subsequent analysis.

In alignment with these methodologies, our thesis will focus on map matching as the primary technique for cleaning the trajectory data in our primary dataset. Additionally, we will focus on trajectory segmentation and the incorporation of semantic information, further refining our dataset to support advanced analytical tasks.

2.2.1 Map Matching

Map matching is an established technique used to correct GPS inaccuracies and errors, particularly in trajectories constrained by road networks. For trajectories confined to a road network, a map-matching algorithm works by aligning each trajectory point with the corresponding road segment. This process produces new coordinates that are accurately matched to the road network, effectively removing inaccurate points that fall outside the network.

The concept of map matching can be visualized in Figure 2.1, where trajectory points are aligned with the appropriate road segments. In the figure, the trajectory is represented by points P_i and road segments S_j, with points P_1, P_3, and P_4 successfully map-matched to their corresponding road segments. However, point P_2 has multiple potential road segment candidates, S_1 and S_2, illustrating a common challenge in map matching. Different map-matching approaches address this challenge in various ways, determining how to accurately align points with the correct segment.

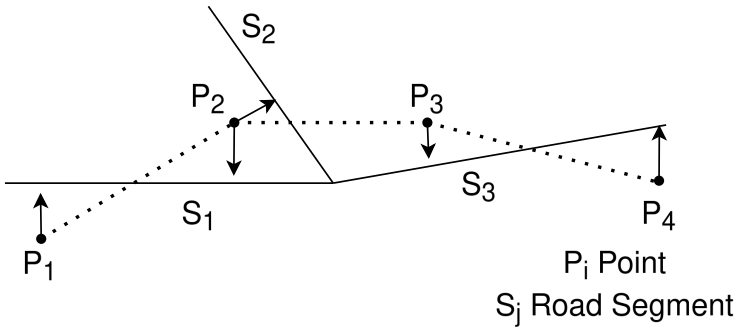


Figure 2.1: Map matching algorithm [1]

Chao et al. [31] discuss several approaches to map matching, including the similarity model, state-transition model, and candidate evolving model. A basic approach involves directly mapping trajectory points to the nearest road segment, but this method is prone to errors as it works solely on closeness, similar to the similarity model. To improve accuracy, more sophisticated approaches, such as the state-transition model, take the object's movement into account. In this thesis, we will focus on the state-transition model, specifically the Hidden Markov Model (HMM), due to its widespread adoption and the availability of several production-ready open-source solutions.

One of the major advancements in map matching using HMM is the work by Newson et al. [32], who developed a novel algorithm for implementing map matching. This algorithm is particularly robust, having been tested on lower frequency data with higher levels of noise. In

their approach, road segments are modeled as HMM states, and the noisy vehicle positions are treated as state measurements. The objective is to match each position measurement with the corresponding road segment state, a logical choice given that state transitions are constrained by the actual road network. This approach is further supported by calculating measurement probabilities to determine which measurements are associated with particular road segments, and transition probabilities to estimate the likelihood of movement between road segments. The transition probability is calculated based on the distance traveled.

These measurement and transition probabilities are then utilized by the Viterbi algorithm to calculate the optimal path. The Viterbi algorithm works with the HMM to identify the most likely sequence of states and observations. The results of this algorithm are impressive; even with a sampling frequency of 30 seconds, the algorithm performs well with minimal error, and at higher sampling intervals, it demonstrates robustness to noise, with standard deviations as large as 50 meters. This algorithm forms the basis for the implementation of open-source tools such as Valhalla*, which use it in their map matching modules.

Interestingly, while the algorithm developed by Newson et al. [32] is not the first HMM-based map matching algorithm, it has become one of the most frequently cited in the literature. Previous researches [33, 34], and more recent studies [35, 36] have also developed HMM-based approaches, underscoring the popularity and widespread adoption of this method in map matching models. In our thesis, we will specifically employ the map matching approach using the HMM model with the Viterbi algorithm, as implemented by the open-source tool Valhalla in its map matching modules. This will enable us to preprocess our AVL data, which contains erroneous GPS points, and allow us to perform advanced preprocessing as described in Chapter 3.

2.2.2 Trajectory Segmentation

In addition to the map matching algorithm, another crucial strategy for preprocessing and cleaning our data is trajectory segmentation. This method is particularly effective for managing complex datasets by breaking down trajectories into smaller, more manageable segments. As discussed in the previous section, segmented trajectories are one of the key representations of trajectory data.

Spaccapietra et al. [17] describe segmented trajectories as subsequences of tuples ordered by temporal information, where all the time intervals are disjointed. This approach, as illus-

*Valhalla open source routing engine

trated in Figure 2.2, demonstrates how trajectories can be divided into segments S_i, resulting in distinct, non-overlapping segments. Segmentation not only simplifies the analysis but also provides an opportunity to uncover patterns that may be hidden within the larger trajectory.

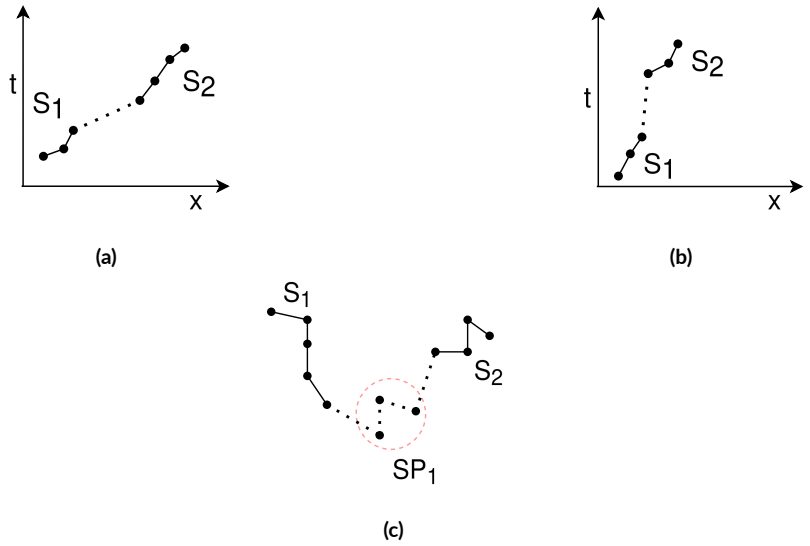


Figure 2.2: (a) Segmented trajectories by spatial gaps [1] (b) Segmented trajectories by temporal gaps [1] (c) Segmented trajectories by stay points [2]

Zheng [2] emphasizes that trajectory segmentation is valuable not only for managing complexity but also for revealing unseen patterns within subsets of the data. By focusing on smaller segments, we can better understand specific behaviors or movements that might be obscured within the full trajectory. This granular approach to trajectory analysis highlights the importance of segmentation in enhancing the clarity and usability of trajectory data.

There are several approaches to segmenting trajectories. One such approach is discussed by Marketos et al. [1], who explore segmentation in the context of reconstructing raw trajectories using key parameters. These parameters—including temporal gaps, spatial gaps, maximum speed, noise duration, and tolerance distance—are crucial in determining how raw trajectories are segmented. By applying these parameters, we can systematically segment trajectories, thereby making the dataset easier to manage and analyze.

One practical application of these parameters is illustrated in Figure 2.2, where segmentation is performed using temporal and spatial gaps. As shown, when the gap between two spatio-temporal points exceeds the threshold, a new segment is created. This method not only helps in organizing the data but also aligns with the previously discussed benefits of segmentation, reinforcing its importance in trajectory preprocessing.

Another parameter in reconstructing trajectories involves the use of tolerance distance, which defines the maximum distance between two positions of the same object for it to be considered stationary [1]. Closely related to this concept, Zheng [2] introduces the idea of stay points, where objects remain stationary for a certain period. Detecting these Stay Points allows us to transform trajectory data into more meaningful representations, as illustrated in the following equation adapted from [2]:

$$T = \{T_1, T_2, T_3, \ldots, T_n\} \Rightarrow S = \{S_1 \rightarrow S_2 \rightarrow S_3, \ldots, S_n\}$$

Here, each S_j segments includes trajectory points T_i that are separated by stay points. This refined segmentation process not only helps in managing the complexity of trajectory data but also facilitates more advanced analyses by distinguishing between stationary and non-stationary points. Figure 2.2, illustrates how stay points SP_i, depicted in red circles, can be used to define segments in the data.

In this thesis, we apply these concepts to segment our trajectories, thereby producing smaller trips and managing the complexity of the trajectory data. For our primary dataset, we use the Stay Point approach to segment the trajectories, a process that will be detailed in Chapter 3. In our secondary dataset, we combine temporal gaps and spatial gaps to construct the trajectories, as will be explained in Chapter 5.

2.2.3 Semantic Trajectories

Another important aspect of building trajectories is the addition of semantic information to raw trajectory data. Spaccapietra et al. [17] define semantics in this context as any representation of a trajectory that has been enriched with contextual data or transformed to add meaning to the raw data. This enrichment process enables us to create semantic trajectories, where the added contextual information provides deeper insights into the trajectory data.

The next crucial step in processing trajectory data is enriching it with semantic information. Parent et al. [28] describe semantic enrichment as the process of enhancing raw trajectories with contextual repositories to produce a set of semantically annotated trajectories. Additionally, Marketos et al. [1] provide a practical framework for constructing these semantic trajectories. Their approach involves first segmenting the trajectory data into episodes, followed by annotating these segments with relevant contextual information, such as regions, lines, or points.

The first step in this framework is constructing episodes or segments. Once these segments are established, they can be enriched with additional context. One approach is to annotate the

trajectories with regions by performing spatial joins with data sources containing geo-objects, thereby adding geographical context. Another method is to annotate the segments with lines, such as road networks, which adds a further layer of meaning. Techniques like map matching, mentioned earlier, are particularly useful for this purpose. Finally, adding points of interest helps identify significant stop episodes within the trajectories, further enriching the dataset with valuable insights.

By applying these methods, we can significantly enhance the semantic value of our trajectory data. In Chapter 3, we demonstrate how these approaches are used to annotate trajectories with lines through map matching and to enrich them with geographical information. This added semantic information proves invaluable, enabling us to extract more detailed behaviors and insights from our trajectories, ultimately leading to more informed analysis and decision-making.

2.3 Trajectory Data Prediction

Navigating the spatio-temporal context of trajectory data presents significant challenges. However, with a solid understanding of how to define, represent, preprocess, and clean this data, we are now well-positioned to conduct more advanced analyses. One such analysis involves making predictions based on our datasets. It is crucial to approach predictive modeling with care, given that trajectories encompass not only spatio-temporal aspects but also their associated semantic contexts.

Research in this field has extensively explored various predictive modeling tasks using trajectory data. In their review paper, Graser et al. [14] outlined several common prediction tasks that utilize trajectory data. These include trajectory prediction, arrival time estimation, and subtrajectory classification, which are often applied to dense trajectory datasets.

In addition to these tasks, predictive modeling can also extend to more specific applications. For example, tasks like next location prediction, anomaly detection, synthetic data generation, and location classification are typically performed on aggregated trajectory data. Other studies have explored predictions related to environmental and vehicle performance metrics, such as emission prediction [37], electric vehicle range estimation [38], and fuel consumption forecasting [39].

Applying these predictive tasks to our trajectory data provides concrete use cases and demonstrates the practical application of our preprocessed and cleaned datasets. Building on this foundation, we introduce two key use cases: predicting electric bus battery consumption as

our primary focus and predicting ship trajectories using AIS data as a secondary focus.

2.3.1 Predicting Vehicle Battery Consumption

Our primary use case with the AVL bus dataset involves analyzing battery consumption by vehicles over specific trips. This analysis is critical for developing smarter transportation planning strategies, which ultimately contribute to greener urban transport systems. To establish a robust foundation for our approach, it is essential to review related research on fuel usage prediction, as these methodologies provide valuable insights and draw parallels to studies on battery consumption.

Prior research in fuel consumption prediction has provided valuable insights that are directly applicable to battery consumption modeling. Statistical models, such as regression and Autoregressive Integrated Moving Average (ARIMA), have been widely utilized to predict fuel usage, as demonstrated in studies by Kabir et al. [40] and Cappiello et al. [37]. Additionally, machine learning methods, particularly the random forest approach, have been successfully applied in this context, as evidenced by the work of Yao et al. [41] and Perrotta et al. [39].

Furthermore, advancements in prediction techniques, including artificial neural networks, have significantly enhanced the accuracy of fuel consumption forecasts. Studies in [40, 41, 42] have shown how these more advanced models can capture complex, nonlinear relationships in the data. These developments in fuel consumption prediction have naturally led to their adaptation and application in the context of battery consumption prediction for electric vehicles.

Research specifically targeting battery consumption has also employed various statistical modeling techniques. For instance, studies in [43, 44] have explored different approaches to predicting battery usage. Caewer et al. [44] utilized multiple linear regression models, with one model incorporating kinematic factors and another integrating acceleration variables, both aggregated over entire trips. Additionally, they developed a micro-model to predict consumption on a more granular level, such as micro-trips. These studies demonstrated that more detailed models tend to offer greater accuracy. However, the micro-model's regression coefficients did not fully align with the physical principles of battery consumption, even though it achieved accuracy comparable to other models, suggesting potential for further refinement and exploration.

More recent studies have explored the use of deep learning techniques for battery consumption prediction. For example, Bundgaard et al. [45] applied deep learning features to predict battery consumption at the trip segment level, achieving notable advancements. Similarly, in

Zarei et al. [46], deep learning combined with Symbolic Aggregation Approximation (SAX) filtering was employed to predict battery consumption in electric buses. This approach demonstrated significant improvements in accuracy compared to traditional regression models, highlighting the effectiveness of deep learning methods.

In this thesis, we aim to leverage both statistical and deep learning models. Regression will serve as our baseline model due to its interpretability, which provides a clearer understanding of the predictor variables. In parallel, we will employ more advanced deep learning models to further enhance prediction accuracy. The detailed methodology and implementation of these models are discussed in Chapter 4.

2.3.2 Predicting Ship Trajectories

A secondary use case that we can explore with our AIS dataset, which contains ship trajectories, is trajectory prediction. While not the primary focus of our study, predicting future ship trajectories is nonetheless crucial for enhancing maritime navigation safety, as these predictions can help prevent collisions and assist in planning new courses for ships. A classical approach to trajectory prediction has its origins in the field of computer vision.

In computer vision, prior research on predicting future trajectories, such as that conducted in [47, 48, 49], primarily addresses short-term prediction. These studies utilize deep learning models to predict short-term human movement trajectories and include additional context related to human interactions or environmental segmentation in the prediction process. Research dealing with long-term trajectory prediction, such as [50], also employs deep learning, incorporating techniques like transfer learning to extend the prediction horizon. Additionally, [51] adopts a probabilistic method for long-term prediction.

A similar application has been implemented in the context of ship trajectory prediction. For example, [25, 52, 53] have developed trajectory prediction models using deep learning techniques that incorporate environmental context, while [54] employs probabilistic methods for prediction. A more general approach, involving the creation of a framework for trajectory prediction, has been proposed by [55, 56]. Specifically, [55] enhances trajectory prediction for online usage by focusing on streaming data, whereas [56] concentrates on building a deep learning framework for trajectory prediction.

However, these approaches often introduce a degree of complexity. A simpler method, as proposed by [24], involves using a straightforward deep learning model combined with linear and spline interpolation to predict future ship trajectories. This approach relies only on ship

speed and geographical coordinates (latitude and longitude) for the prediction.

In our research, as a secondary use case, we will utilize the same approach by employing a straightforward deep learning model and using the same features, which will be detailed in Chapter 5. This secondary use case aims to demonstrate the flexibility of our methodology, drawing from consistent knowledge in representing, preprocessing, and cleaning trajectory data.

2.4 Trajectory Data Prediction Model

In the previous section, we observed that several applications of trajectory data prediction utilize deep learning methods for their analysis. These deep learning algorithms have been extensively studied and, due to their robustness, have been successfully applied to the trajectory data domain. One of the most common and well-researched deep learning methods for prediction analysis on trajectory data is the use of RNN.

In [57], the authors employed the RNN architecture to exploit the sequential nature of trajectory data. Additionally, recent studies by Graser et al. [14, 58] indicated that RNN-based architectures account for 39% of recent publications related to the application of deep learning to trajectory data. This popularity is largely due to the fact that trajectories can be naturally viewed as sequences ordered by their timestamps, making RNN-based architectures well-suited to capture and exploit the sequential attributes inherent in trajectories.

In the following sections, we will provide an overview of Artificial Neural Networks as a foundation for our deep learning model, followed by a discussion on RNN architecture. We will then focus on one of its most prominent variants, the Long Short-Term Memory (LSTM) network. These sections will explain how these architectures are constructed and emphasize the specific properties that make them particularly effective for predicting sequential data.

2.4.1 Aritficial Neural Networks

One of the foundations of deep learning is the feedforward neural network, also known as the multilayer perceptron (MLP). Goodfellow et al. [4] describe that the goal of this network is to approximate some function f_*. Additionally, in this specific network, there is no feedback from the output back through the model, which is why it is called a feedforward neural network. Next, we can examine this network by looking at its components.

As explained by Nielsen [3], these networks are composed of small components called neurons. One of the earliest approaches to modeling a neuron is the perceptron, developed by Rosenblatt [59]. The perceptron takes each input and multiplies it by a weight. The output of the perceptron is influenced by a threshold, known as the bias, which determines the neuron's activation. The operation of the perceptron can be mathematically expressed as a dot product between the weights and the inputs, represented as $w \cdot x$, with the bias added, resulting in the following equation [3]:

$$f = w \cdot x + b$$

$$\text{Output} = \begin{cases} 0 & \text{if} f \leq 0 \\ 1 & \text{if} f > 0 \end{cases}$$

Instead of using the perceptron's step function, other activation functions can be used within neurons. With the perceptron, a small change in the weight or bias can cause the output to flip, which is often too extreme. To address this issue, the sigmoid function can be introduced, allowing for continuous inputs and producing a smoother transition in the output. The sigmoid function is applied as follows: $\sigma(w \cdot x + b)$. The sigmoid function is described by the following equation [3]:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

The choice of activation function is not limited to the sigmoid; other functions can be used depending on the desired behavior of the network. This selection of functions, known as activation functions, is crucial in determining the output of each layer.

As illustrated in Figure 2.3, the complete structure of a feedforward neural network is depicted, where each layer comprises a set of neurons. This network architecture includes hidden layers, which are responsible for additional computations before the final output is produced. The formal structure of this network can be described as outlined by Goodfellow et al. [4], where the network function is represented as $f(x) = f_3(f_2(f_1(x)))$. In this formulation, f_1 corresponds to the first layer, f_2 and f_3 represent the subsequent layers, and during the training process, each layer is iteratively adjusted to approximate the optimal function $f_*(x)$.

While feedforward neural networks are effective for approximating complex functions in model building, they have certain limitations. Specifically, these networks do not incorporate feedback from the output back into the network, which restricts their ability to maintain state

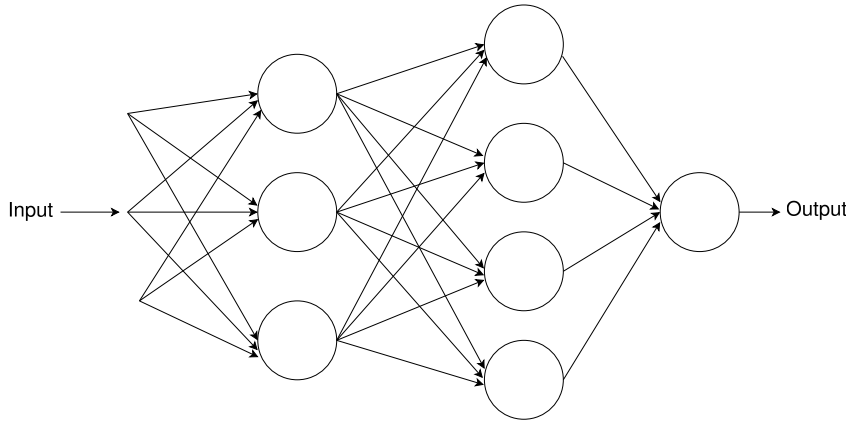


Figure 2.3: Neural networks architecture [3]

or context across different layers. To address this limitation, other types of neural networks have been developed that integrate feedback mechanisms, allowing the network to retain information over time. One such network is RNN, which will be discussed in the following section.

2.4.2 Recurrent Neural Networks

Recurrent neural networks (RNN), as described by [4], are a class of neural networks specifically designed to work with sequential data. When presented with a sequence of inputs, $x_1, \ldots, x_t$, an RNN can learn the dependent patterns within the sequence by maintaining a memory of previous computations.

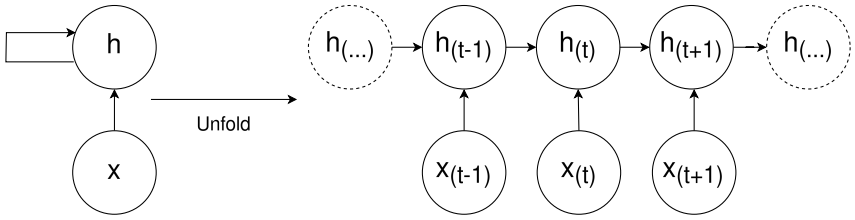


Figure 2.4: Illustration of an RNN architecture [4]

As depicted in Figure 2.4, an RNN can be conceptualized as a loop that recursively processes the sequential data, with each iteration referencing the previous timestep. When this loop is unfolded, it becomes apparent that each layer within the RNN corresponds to a specific timestep of the input sequence. The operations of the RNN can be mathematically formalized as fol-

lows, based on the equations provided in [4]:

$$f(t) = b + W \cdot h(t-1) + U \cdot x(t) \tag{2.1}$$

$$h(t) = \tanh(f(t)) \tag{2.2}$$

$$o(t) = c + V \cdot h(t) \tag{2.3}$$

In these equations, the matrices W, U, and V represent the weights that connect different parts of the network. Specifically, W is the weight matrix for the connections between hidden states, U is the weight matrix for the connections from the input to the hidden state, and V is the weight matrix for the connections from the hidden state to the output.

Equation 2.1 describes the computation at each timestep t, where the function $f(t)$ is computed by combining the previous hidden state $h(t-1)$, the current input $x(t)$, and a bias term b. The current hidden state $h(t)$ is then obtained by applying the tanh activation function to $f(t)$, as shown in Equation 2.2. Finally, the output activation $o(t)$ is calculated by adding a bias vector c to the weighted hidden state $h(t)$, as expressed in Equation 2.3.

From the equation we can see why RNN is powerful in taking sequence input, since it incorporated the previous hidden state in the current time step as the part of the equation. However this not without a problem, Hochreiter in [60] and Bengio et al. [61], described that major drawbacks for this RNN network is the vanishing gradient problem and the difficulties learning much more longer sequences. Hence, in [60] new variant of RNN network is inroduced, which called Long-term short memory (LSTM).

LSTM networks introduce the capability to control the flow of information, selectively retaining what is essential and discarding what is irrelevant. As illustrated in Figure 2.5, this flow management is achieved through structures known as gates. The LSTM cell comprises three primary gates: the input gate, the forget gate, and the output gate. Each gate serves a specific function, as represented by the following equations [4]:

$$f_i^{(t)} = \sigma\left(b_i^f + \sum_j U_{i,j}^f x_j^{(t)} + \sum_j W_{i,j}^f h_j^{(t-1)}\right) \tag{2.4}$$

As shown in Equation 2.4, the forget gate's primary role is to determine whether information from the previous timestep should be retained or discarded. This decision is made by applying

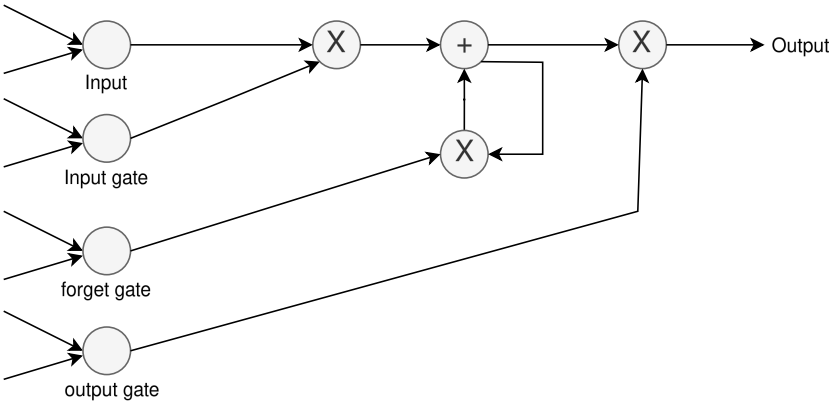


Figure 2.5: Illustration of an LSTM architecture [4]

a sigmoid activation function to the hidden state of the previous timestep.

$$s_i^{(t)} = f_i^{(t)} s_i^{(t-1)} + g_i^{(t)} \sigma \left(b_i + \sum_j U_{i,j} x_j^{(t)} + \sum_j W_{i,j} h_j^{(t-1)} \right) \tag{2.5}$$

$$g_i^{(t)} = \sigma \left(b_i^g + \sum_j U_{i,j}^g x_j^{(t)} + \sum_j W_{i,j}^g h_j^{(t-1)} \right) \tag{2.6}$$

The input gate, as seen in Equation 2.5 and Equation 2.6, governs the extent to which new input information should be integrated into the cell state. This update process involves a combination of information from the forget gate, which decides the retention of previous cell states, and the input gate, which determines the amount of new information to be incorporated.

$$h_i^{(t)} = \tanh \left(s_i^{(t)} \right) \cdot q_i^{(t)} \tag{2.7}$$

$$q_i^{(t)} = \sigma \left(b_i^o + \sum_j U_{i,j}^o x_j^{(t)} + \sum_j W_{i,j}^o h_j^{(t-1)} \right) \tag{2.8}$$

Finally, the output gate, as described in Equation 2.8 and Equation 2.7, determines whether the information from the current hidden state will be transmitted as output. In these equations, b^*, U^*, and W^* represent the bias, input weight, and recurrent weight at each gate, respectively.

The introduction of these gates significantly enhances the LSTM's ability to manage information flow, making it more effective than a standard RNN, particularly when working with

longer sequences. By controlling the retention and update of information through the input, forget, and output gates, LSTM networks address the limitations of traditional RNNs in handling long-term dependencies. In this thesis, we employ LSTM as the primary model for both prediction analyses, which will be discussed in detail in Chapter 4 and Chapter 5.

2.5 Evaluation Metrics

To evaluate our predictive analysis, we will define the metrics used and provide the rationale behind their selection. The evaluation metrics chosen are straightforward and have been selected to facilitate easy interpretation of our results. As discussed in [62], common quantitative measures for assessing the performance of regression models include Mean Squared Error (MSE) and Root Mean Squared Error (RMSE). Additionally, the R^2 Score can be used to assess the model's ability to explain variance in the data. We will use these metrics to evaluate our models and will also explore the Mean Absolute Error (MAE) and Mean Absoulute Percentage Error (MAPE).

R^2 Score: The R^2 score, or coefficient of determination, as explained by [62], indicates the proportion of the variance in the dependent variable that is predictable from the independent variables. This metric helps compare the predictive performance of models by indicating how well they explain the variance in the data. The formula for the R^2 score is as follows:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

MSE: MSE is calculated by taking the average of the squared differences between the actual and predicted values. MSE is sensitive to outliers, as it disproportionately penalizes larger errors. The formula for MSE is given by:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

RMSE: RMSE is similar to MSE but is more interpretable, as it is on the same scale as the data. It is calculated by taking the square root of the MSE and is also sensitive to outliers. The formula for RMSE is:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

MAE: As explained by [63], MAE is a more natural measure of average error in prediction results. It is less sensitive to outliers compared to MSE, as it simply averages the absolute differences between the actual and predicted values. The formula for MAE is:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}\left|y_i - \hat{y}_i\right|$$

MAPE: Similar to MAE, MAPE calculates the mean of the percentage error between the actual data and predicted data. It is particularly useful when we want to express the error as a percentage. The formula for MAPE is:

$$\text{MAPE} = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right| \times 100$$

Defining these evaluation metrics is crucial. In our primary use case, the R^2 score and RMSE will be used for comparative analysis between the baseline and deep learning models, providing a clear evaluation of predictive performance. In the secondary use case, RMSE and MAPE will be employed to measure the deviation of the predicted values from the actual data.

2.6 Summary

In this literature review, we have thoroughly and extensively explored the foundational literature that underpins our thesis. We began by examining the literature on trajectory data, focusing on how to preprocess, clean, and use this data for predictive analysis. This review is an integral part of our research as it provides the guidelines for our approach and identifies the gaps that our study aims to fill.

Our research is focused on applying foundational theories regarding trajectories to tangible, real-world applications. We started by understanding the characteristics of trajectories, drawing upon studies such as [2, 16, 17]. We then adopted an extensive and exhaustive approach to preprocess and clean our trajectory data, informed by the work of [1, 28, 32]. Finally, we examined methods for our predictive analysis by reviewing prior predictive analysis tasks in [14, 38] and studies regarding deep learning models in [3, 4].

Based on our literature review, [6] provided a comprehensive overview of all aspects of trajectories, including their applications. Our research extends this theoretical foundation by incorporating more modern and practical approaches, utilizing tools such as MobilityDB [10],

PyMEOS [11], MovingPandas [12] , and Valhalla to represent and process trajectory data. In the application phase of our research, we adopt advanced tools for predictive analysis, integrating deep learning models across both of our use cases.

In our primary use case, which involves predicting electric bus battery consumption, prior research, such as [42, 45, 46], has applied deep learning models to predict vehicle energy consumption. However, these studies do not extensively address the trajectory aspect. In our research, we thoroughly investigate our dataset and carefully transform it into a more meaningful representation using the foundational theories we have reviewed. Techniques such as map matching and trajectory segmentation are employed. In the prediction analysis, research in [46] is the most closely related to our work, as it also predicts electric bus energy consumption using public transportation data. However, we extend this approach by incorporating road segment information as suggested in [45].

To complement our analysis, we provide a secondary use case predicting ship trajectory data, following the same approaches in representing and preprocessing trajectories. This secondary use case offers a different perspective, demonstrating the versatility of our approach by employing concepts such as temporal and spatial gaps detection. In the prediction task, abundant research has been conducted in the field of trajectory prediction, including [47, 48], with a particular focus on ship trajectory prediction in [24, 25, 52]. Our approach is most similar to [24, 25], as we utilize the same prediction model. However, their approach is more focused on the characteristics of AIS datasets, whereas our research emphasizes general trajectory data.

In conclusion, we will conduct predictive analysis on trajectory data, basing our methods on foundational literature related to trajectories. We will represent, preprocess, and clean our data, and employ advanced models such as deep learning to perform the predictive analysis.

3

Data Preprocessing for the Electric Bus Usecase

Given the vast amount of trajectory data available, particularly in urban environments, one significant application of trajectory prediction is to enhance intelligent and eco-friendly transportation systems, especially public transit. By leveraging trajectory data, we can implement smart planning strategies in public transportation to improve trip efficiency and minimize energy consumption. This is increasingly important with the growing adoption of electric vehicles, particularly in Europe. As a preliminary step toward smart planning, we can begin by predicting the battery consumption of vehicles during their trips.

In predicting battery consumption, prior research such as [45, 46] has focused on electric vehicle data. Specifically, [45] focused on predicting battery consumption for electric cars, while [46] concentrated on electric buses. In this use case, we will focus on predicting electric bus consumption using bus trajectories in the form of AVL data.

Section 3.1 provides a brief overview of our dataset. Section 3.2 explores the challenges present in our data. Section 3.3 explains how we address these challenges and clean the data. Section 3.4 offers a comprehensive overview of our data cleaning pipeline to build the data for prediction. Finally, Section 3.5 summarizes the steps taken and outlines potential improvements.

3.1 Dataset Overview

In order to build a comprehensive analysis and develop a predictive model for forecasting battery consumption from an electric bus trajectory, it is crucial to thoroughly explain the dataset and acknowledge the associated constraints. By doing so, we can establish a clear path and strategy for implementing an effective data preprocessing method, which serves as the foundation for constructing our prediction model.

Understanding the specifics of our dataset is the first step in this process. The primary dataset comprises bus trajectories collected over a period of 20 days. This dataset includes key attributes such as the vehicle identification number (ID), the recorded signal values, and the geographical position of the bus, represented by latitude and longitude coordinates. The recorded signals consist of battery autonomy, battery percentage, cumulative vehicle distance, and cumulative battery consumption.

For the purposes of our thesis, we focus on the cumulative battery consumption signal. This filtered dataset retains essential information including the vehicle ID, cumulative battery consumption measured in kilowatt-hours (kWh), and geographical coordinates. Each trajectory point in the dataset is also timestamped, which is crucial for accurately sequencing the trajectory points. The timestamp data plays a critical role in our analysis, enabling precise reconstruction of the bus trajectories. Table 3.1 provides detailed information regarding the data type of each attribute.

Attribute	**Datatype**
T	Timestamp
Vehicle ID	String
Cumulative Battery Consumption	Integer
Latitude	Float
Longitude	Float

Table 3.1: Filtered primary dataset: Attributes and data types

By understanding the structure and constraints of our dataset, we can establish a robust foundation for our subsequent analysis. This knowledge allows us to identify and address potential issues that could impact our predictive model's performance. In the next section, we will conduct a preliminary analysis of the dataset, identifying specific challenges and outlining the steps necessary for effective data preprocessing.

3.2 Data Exploration

A thorough investigation of the constraints and limitations within our dataset allows us to strategically address these challenges and build a comprehensive data preprocessing pipeline. This step is vital before proceeding with the development of our prediction model. By recognizing and tackling these issues, we can implement the most suitable techniques to improve the quality and accuracy of our dataset.

3.2.1 GPS Inaccuracies

After understanding the structure of our dataset, it is essential to examine the characteristics of the data to identify any challenges that need to be addressed through data preprocessing. One of the first aspects to consider is the nature of our trajectories. By plotting the trajectories on a map, we can inspect our raw data visually. Figure 3.1 illustrates the spread of one vehicle's trajectory over a single day, depicted in blue, along with the bus road networks depicted in orange.

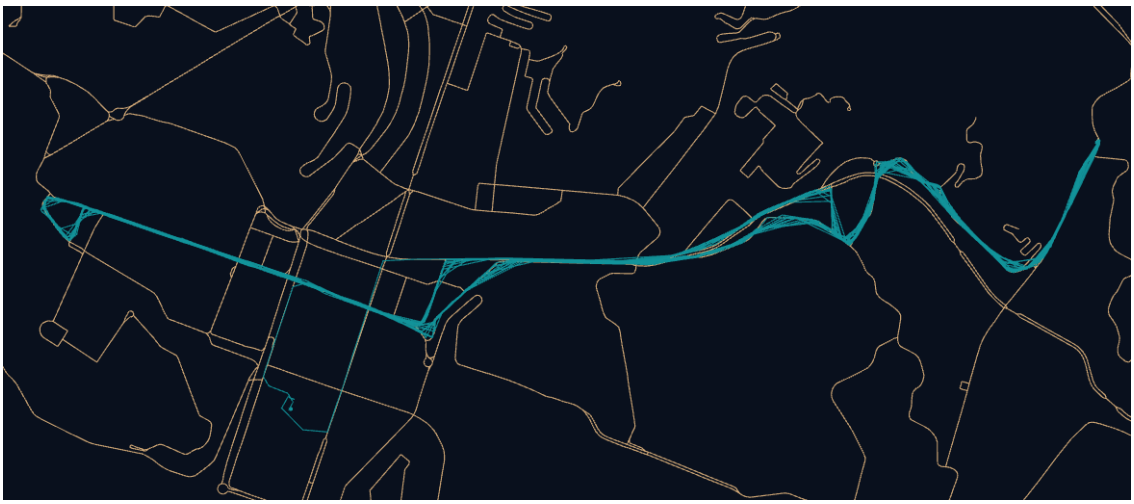


Figure 3.1: A day of vehicle trajectory

As shown in Figure 3.1 the trajectories are primarily concentrated along bus road networks, with slight deviations in certain areas. This visualization allows us to assess the general path followed by the vehicle. However, to gain a more comprehensive understanding of the data, it is necessary to closely examine the individual points within the trajectory. This detailed inspection can reveal specific data points and their alignment with the actual bus road networks.

Figure 3.2 provides a closer look at these individual points within the trajectory, highlighting how they align with the bus routes and where deviations occur.

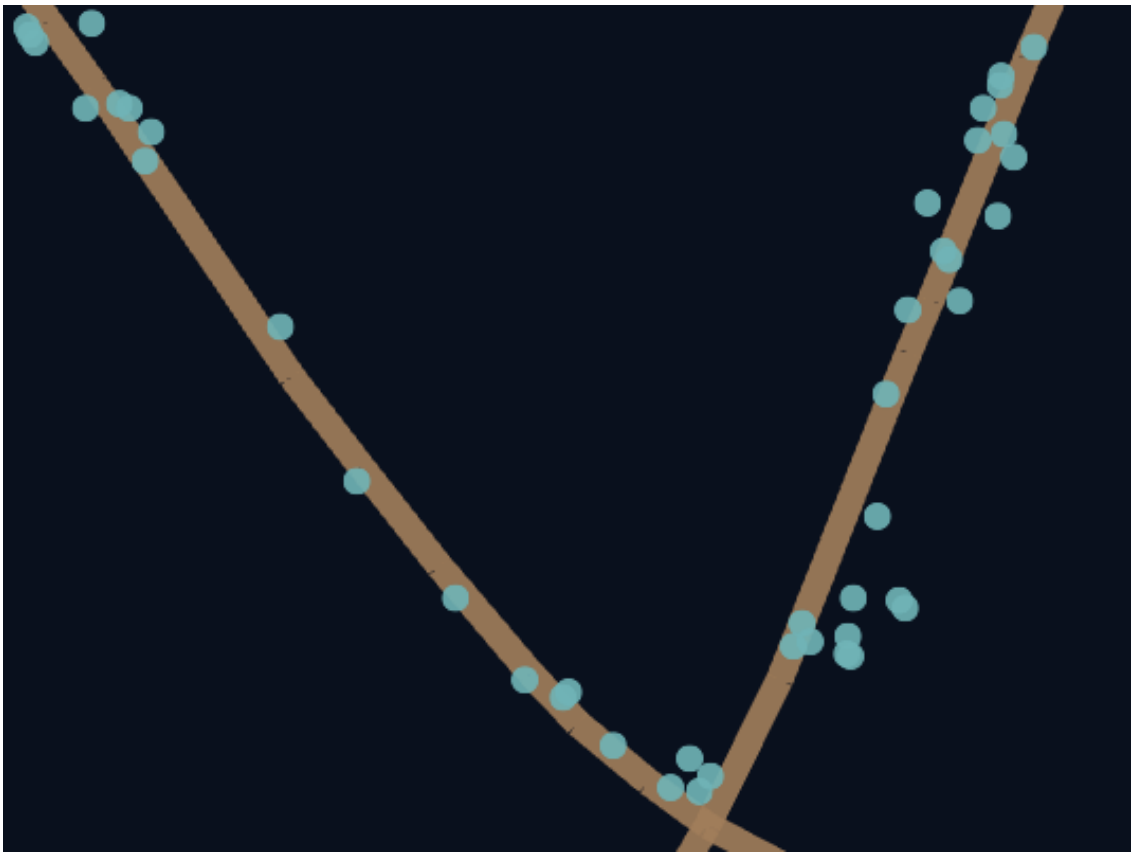


Figure 3.2: Unprocessed points along the bus road network

From Figure 3.2, it is evident that points do not align perfectly with the bus road networks depicted on the map. This discrepancy is expected, as GPS data is not always precise. Understanding these inaccuracies is crucial for refining our data preprocessing methods to improve the overall accuracy of our predictive model.

3.2.2 Discontinuities and Frequencies in Bus Trajectories

Another important aspect to consider from our trajectory data is that buses do not travel continuously throughout the day but make several stops along their routes, either to charge their batteries or to wait for the next scheduled trip. This is illustrated in Figure 3.3, where several points close to each other are indicated in pink. Detecting these stop points is important as it allows us to refine our algorithms and segment the bus trajectory more precisely.

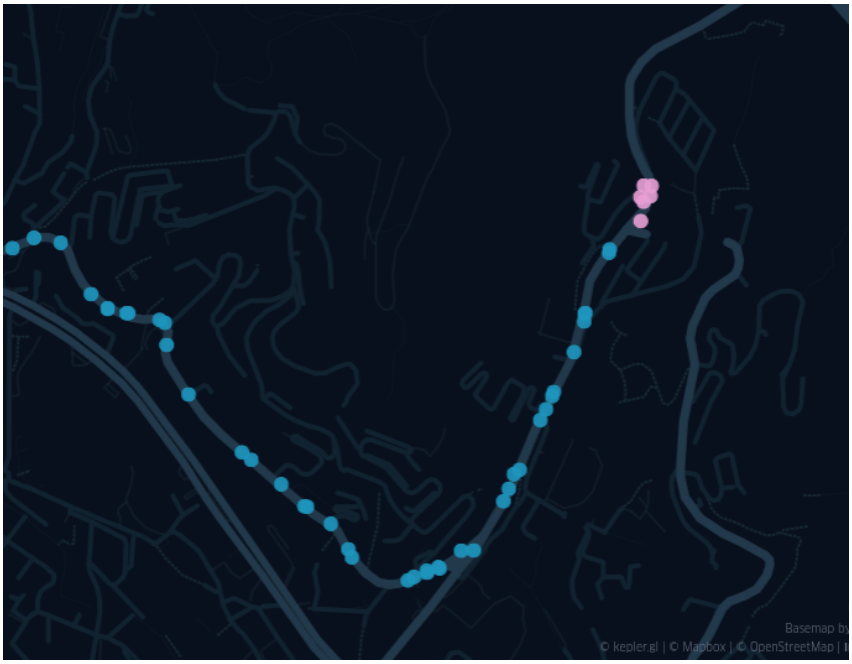


Figure 3.3: Possible stopping points in trajectories

These issues, coupled with the dataset's frequency, further complicate our analysis. The frequency of our dataset introduces additional challenges. Our data points are recorded every 30 seconds, or 0.03 Hz, typically resulting in a 30-second gap between each trajectory point. However, not all data points adhere strictly to this frequency; some gaps can be as large as 7 hours 40 minutes, resulting in frequencies as low as 0.00003 Hz, creating a much sparser trajectory. Importantly, no trajectory points are recorded with a gap shorter than 30 seconds. This sparsity presents a significant challenge in accurately estimating battery consumption. In comparison, previous studies [45, 46] have utilized datasets with frequencies of 1 Hz and 0.1-1 Hz, respectively. Thus, our dataset is notably more sparse at 0.03 Hz, complicating the data analysis and preprocessing tasks further.

3.2.3 Cumulative Battery Energy Consumption

In addition to the sparsity of the dataset, the battery consumption value is calculated cumulatively, adding complexity to analyzing energy consumption at a finer granularity, such as at the trajectory point level. Figure 4.1 shows the cumulative battery consumption over a day. As we can see, the consumption gradually increased from zero to the maximum consumed for that day. Additionally, we observe slight fluctuations in the consumption data, and in some cases,

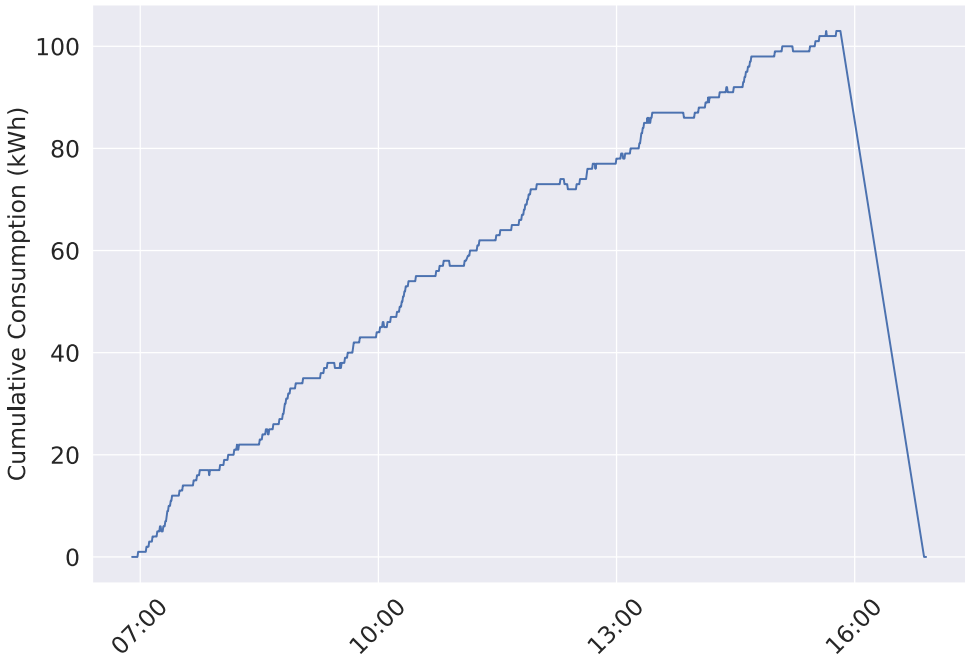


Figure 3.4: Cumulative battery consumption (kWh) throughout the day

the readings can drop to zero at the end due to issues with the reading device. This cumulative nature of the data, necessitates additional preprocessing steps to address these challenges effectively.

3.2.4 Dataset Context Deficiency

Moreover, we must consider the context within our dataset, which includes spatial and temporal information. The spatial context is represented by the position of the vehicle through Longitude and Latitude, while the temporal information is provided by the timestamp value. These two pieces of information are sufficient to reconstruct the trajectory and show the changes in cumulative battery consumption over time. However, they do not fully capture the reality of the situation, leading to challenges in accurately reflecting the factors influencing battery consumption. Addressing these contextual limitations is crucial for improving the accuracy of our predictive model.

This lack of context can lead to problems in capturing more variables that affect battery consumption. Having more context from the dataset will help capture more variables and en-

able more accurate predictions of battery consumption. Knowing this, we will need to add context to our dataset that might affect the prediction of battery consumption, but we must do this carefully to ensure data integrity and accuracy. By thoroughly investigating the constraints and limitations of our dataset, we can strategically address these challenges to build a comprehensive data preprocessing pipeline. This is an important step before developing our prediction model.

3.3 Data Cleaning

In this section, we will systematically address the various issues in our dataset and provide a thorough explanation of the algorithms, techniques, and tools used to resolve these problems. We will begin by addressing the GPS inaccuracies. Next, we will tackle the issue of discontinuity in the bus trajectories. Following that, we will address the challenges associated with the cumulative battery consumption values in our dataset. Finally, we will explain how we added additional context to enhance the dataset.

3.3.1 Map Matching

We recognize the issues related to GPS inaccuracies in our dataset, and addressing these inaccuracies is crucial for our analysis. Handling bus trajectory data presents specific challenges, such as the need to correct the inaccurate GPS points and align them with the road network. This correction process, known as map matching, enhances the precision of the GPS locations by aligning them with the actual road network.

A straightforward approach to tackle GPS inaccuracies is to snap the inaccurate GPS points to the nearest points on the road network. However, this method is not optimal because it does not consider the sequential order of the points. More sophisticated algorithms, such as those utilizing fuzzy logic or HMM [64], provide better solutions by maintaining the order and context of the points. Employing these advanced techniques can significantly improve the accuracy of the matched GPS points in our dataset.

In this study, we utilize a HMM-based map matching algorithm, specifically the Viterbi search algorithm. This method is widely used and supported by tools such as Leuven Map Matching [35] and Valhalla Meili. We selected Valhalla Meili for its comprehensive documentation and efficient C++ implementation [*], accessed via its trace route API.

[*] Valhalla Meili implementation details

For implementing map matching with Valhalla, we group the data by vehicle per day. This grouping ensures that the map matching process handles an optimal amount of data while retaining sufficient information about the trajectory. Table 3.2 illustrates the trajectory parameters passed to the map matching algorithm. Latitude and longitude are used to derive the map-matched trajectory, and time data maintains the sequence of the trajectory points. These trajectory points, combined with other parameters for map matching, are essential for the map matching process.

Attribute	Type
Latitude	Float
Longitude	Float
Time	Integer

Table 3.2: Trajectory attributes for map matching

Table 3.3 outlines the parameters used for map matching. The search radius parameter, set to 10 meters, defines the radius within which the algorithm searches for corresponding road segments, enhancing accuracy. The shape match parameter specifies the map matching algorithm, and the costing algorithm is set to 'bus' to ensure matching on bus road networks. With these parameters configured, we can execute the map matching process.

Parameters	Value
Search Radius	10m
Shape Match	map snap
Costing	bus

Table 3.3: Configuration parameters for map matching

After defining the parameters, we apply the map matching algorithm as shown in Algorithm 3.1. This algorithm processes each trajectory by combining it with the map matching parameters to accurately align the trajectory with the corresponding road segments. In addition to generating the map-matched points, we also extract further information about the road segments to facilitate additional preprocessing of our dataset.

The output of the map matching process includes a map-matched trajectory with new coordinates for each point, along with the relevant road segment details. These attributes are crucial for estimating energy consumption at the road segment level. Furthermore, we calculate the

distance traveled along each segment, which aids in precise distance estimation. This processed data will form the foundation for further analysis in subsequent stages.

Algorithm 3.1 Map Matching Function

Require: trajectory $\mathcal{T}$

```
parameters = get_parameters()
map_matched_trajectory = callValhallaMapMatch(𝒯,
                         parameters)
for point in map_matched_trajectory
    road_segment_ids = get_road_segment_id(point)
    road_segment_length = get_road_segment_length(point)
    distance_along_road_segment = get_distance_along_road_segment(point)
end for
return (map_matched_trajectories, road_segment_ids,
            road_segment_length,
            distance_along_road_segment)
```

Following the application of the map matching algorithm, we have obtained cleaned trajectories that are accurately aligned with the bus road network. Figure 3.5 illustrates the difference between the trajectories before and after applying the map matching process. The red points indicate the positions before map matching, while the blue points indicate the positions after map matching. The next step involves segmenting the trajectory data, which will be detailed in the following section.

Figure 3.5: Map matching result

3.3.2 Segmenting the Trajectories

Previously, we noted that buses do not travel continuously throughout the entire day; instead, they have stops between their trips. To distinguish between actual bus trips and stops, we need to segment the trajectory data. By segmenting the trajectory and dividing the bus data into individual trips, we obtain a clearer representation of battery consumption during these trips.

There are several approaches to segmenting the data. One possible approach is using General Transit Feed Specification (GTFS) information from the bus schedule. However, this approach is challenging due to the nature of our data, which lacks necessary information such as the specific routes served by the vehicles or the bus stops visited during the journey. Additionally, deviations from the route, or if the bus is running too early or too late from the schedule, are not captured in the GTFS data. Therefore, we need an alternative method for segmenting the trajectory.

A viable approach is to detect when the vehicle stops for a certain duration. This approach is based on the logic that a vehicle typically stops before starting another trip according to its schedule. By identifying these stopping points, we can segment the trajectory accordingly, separating the periods when the bus is stationary from when it is moving. This allows us to define different trips, with stopping points serving as the boundaries between trips.

To achieve this, we utilize the stop detection tools provided by MovingPandas [12] using

Figure 3.6: Segmented trajectories

version v0.18.1 [65]. We define parameters for detecting stops, as depicted in Table 3.4. The tool works by collecting points within a defined radius and, if the duration within that radius meets the minimum threshold, assigning those points as stopping points. The result of this stop detection process is a set of points identified as stops, which we can then use to split the trajectory.

Parameters	Value
Maximum Radius	100m
Minimum Duration	240 seconds

Table 3.4: Parameters for stop detection

After splitting the trajectory based on the stopping points, we obtain segmented trajectories. As shown in Figure 3.6, the segmentation results in several segments, including stopping segments. These segmented trajectories allow us to distinguish between actual bus trips and periods when the bus is stationary. This distinction is valuable for building the data pipeline for our predictive model. In the following section, we will discuss how we handle the cumulative battery consumption values in the dataset.

3.3.3 Estimating Battery Consumption

Another significant issue we encounter is that our battery consumption values are cumulative. This poses a problem, as we cannot determine the consumption value at each trajectory point. Knowing the value at each trajectory point is crucial as it serves as a basis for estimating consumption at each road segment. To estimate this value, we will employ linear interpolation.

Linear interpolation is a useful technique for estimating battery consumption at specific points in time. It is a straightforward method, making it a viable option given the limitations and constraints of our data. Since our dataset includes temporal information, we can leverage this to perform linear interpolation. Figure 3.7 illustrates how we apply linear interpolation using temporal data.

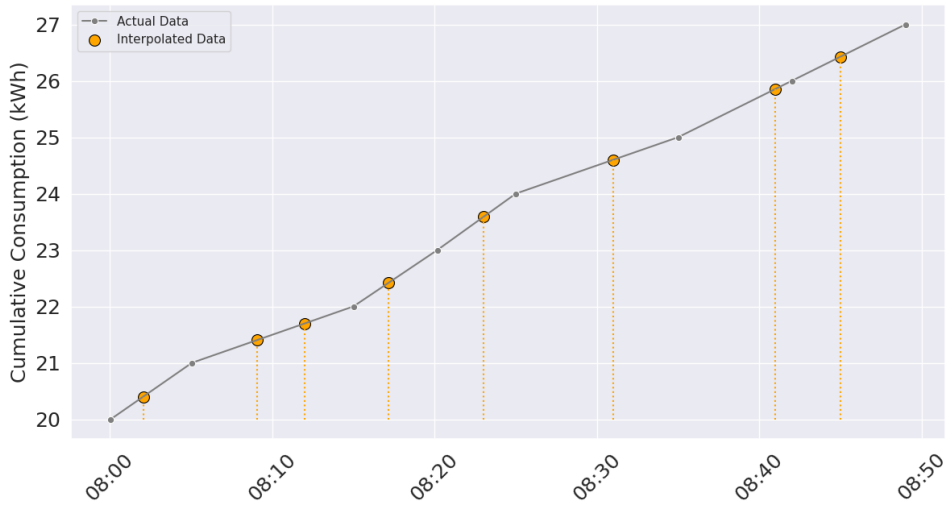


Figure 3.7: Linear interpolation of battery consumption (kWh) over time

To explain this further, suppose we have a battery consumption value X bounded by timestamps $T1$ and $T2$. We can interpolate the battery consumption values at all points between $T1$ and $T2$. For each value in the trajectory, we create time bounds and interpolate for all points that lie within these bounds.

After applying linear interpolation, we obtain an estimated consumption value for each trajectory point. This estimation is crucial as it allows us to determine the battery consumption for each road segment. These values will be utilized in the subsequent section, where we will add more context to our dataset.

3.3.4 Enhancing Dataset Context

As mentioned in the previous section, one of the challenges in our dataset is the lack of contextual information. To represent conditions more accurately, we will enrich the data with weather information, elevation data, and road grade details for each road segment. To achieve this, we will utilize open-source APIs to obtain additional data and combine it with our dataset. We begin by adding weather context.

Capturing weather-related information in our dataset is crucial, as factors such as temperature, rainfall, and wind can significantly influence bus battery consumption. Environmental conditions impact the efficiency and energy usage of the bus, making it essential to include them in our analysis. We obtain this weather information using the OpenMeteo API [66], which provides comprehensive historical weather data. By integrating this data, we can better understand and model the external factors affecting battery performance.

The weather dataset from OpenMeteo has a spatial resolution of nine square kilometers and a temporal resolution of one hour. To integrate this data, we call the OpenMeteo API with specific parameters, as explained in Table 3.5. Given the nine square kilometers spatial resolution, a single query is sufficient to cover the areas included in our dataset. We focus on requesting the most relevant weather information that may impact battery consumption, such as temperature variations, precipitation levels, and wind speed and direction. This enriched dataset will enable us to build a more accurate predictive model by accounting for the environmental conditions in which the buses operate.

Parameter	Value
timezone	Europe/Berlin
hourly	temperature_2m, relative_humidity_2m, precipitation, wind_speed_10m, wind_direction_10m, sunshine_duration
location	Coordinate of the dataset

Table 3.5: OpenMeteo Weather History API parameters

After obtaining the weather information, we will further enhance our dataset by incorporating elevation and road grade (the gradient level of the road) for each road segment. Since our dataset does not contain embedded elevation information, we will utilize external data from the Tinitaly 1.1 dataset [67]. Tinitaly provides raster elevation data with a 10-meter resolution for Italy, which is sufficiently precise.

To integrate the elevation data into our dataset, we first need to gather the spatial information

of our road segments. This can be achieved using a tool called Overpass Turbo [†], which allows us to extract the coordinates that define our road segments. Once we have these coordinates, we can overlay them on the Tinitaly 1.1 raster dataset to obtain the corresponding elevation data.

With the spatial information in hand, we then use PostgreSQL with the PostGIS extension to retrieve the elevation data from the Tinitaly 1.1 dataset. This process involves obtaining elevation values for each coordinate within the road segments. Using this elevation data, we can calculate the average elevation for each segment. Additionally, we compute the average road segment grade and angle, as described by Equation 3.1 and Equation 3.2

$$\text{grade} = \left(\frac{\text{elevation\_change}}{\text{horizontal\_distance}} \right) \times 100 \tag{3.1}$$

$$\text{angle} = \text{deg}(\arctan \left(\frac{\text{grade}}{100} \right)) \tag{3.2}$$

These calculations involve determining the grade for each consecutive pair of coordinates within a road segment and then averaging these values to get the overall grade and angle for the segment. This detailed analysis allows us to understand the characteristics of each road segment, such as whether it is primarily inclining or declining and the overall steepness. This enriched data provides a more comprehensive understanding of the road conditions affecting battery consumption.

3.4 Building the Data Preprocessing Pipeline

After addressing the issues in our dataset, we can move on to building the data preprocessing pipeline. Since our goal is to estimate battery consumption, we will break down the problem by estimating consumption at the road segment level. First, we need to calculate battery consumption for each road segment. After that, we will validate our calculations against the dataset to ensure they remain within the cumulative values of the actual dataset. Finally, we will explain the final dataset used to build our prediction model.

[†]Overpass Turbo tools for getting road segment informations

3.4.1 Road Segment Battery Consumption

Since we have estimated the battery consumption value for each trajectory point, we can now utilize this data to calculate the battery consumption for each road segment. From the map matching results, we obtain information about the road segments traversed between two trajectory points, including the length of each road segment and the proportion of the segment that has been traversed. Using this information, we can estimate the battery consumption for each road segment.

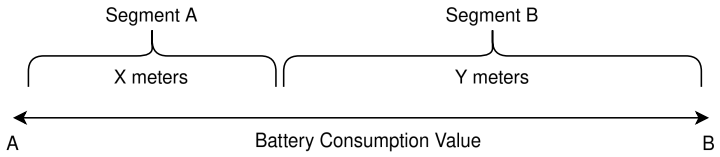


Figure 3.8: Battery consumption estimation at the road segment level

To accomplish this, we employ a straightforward approach for estimating segment consumption. As illustrated in Figure 3.8, we calculate the segment consumption using the proportion of the traversed segment length relative to the total distance traveled between points A and B. By applying this proportion to a simple equation, shown in Equation 3.3, we can obtain the segment consumption value.

$$\text{segment consumption} = \left(\frac{\text{segment\_length}}{\text{total\_length}}\right) \times \text{Battery consumption value} \tag{3.3}$$

Furthermore, after calculating the battery consumption for each road segment, we account for cases where two trajectory points fall within the same segment. In such instances, we aggregate the segment consumption values of the consecutive points within the same road segment. This approach ensures that we have comprehensive consumption data for each road segment. The next step involves validating our calculations to ensure accuracy and consistency with the cumulative values in the dataset. By doing so, we can confirm the reliability of our data and its suitability for further analysis.

3.4.2 Validation of Road Segment Consumption Calculation

Now that we have calculated the segment consumption, the next step is to validate our dataset. To accomplish this, we need a method to ensure the accuracy of our consumption estimates.

Our approach for validation involves comparing the cumulative battery consumption values with the sum of all road segment consumption values. This method is straightforward and effective.

However, we face challenges due to the precision of the cumulative values being limited to integers, which results in the loss of detailed information, especially when battery consumption data remains constant at the end of the trajectory. Additionally, the map matching algorithm may remove erroneous trajectory points, leading to missing values in the cumulative battery consumption data.

Attribute	Value
Mean	0.815476
Standard Deviation	0.578912

Table 3.6: Difference between daily ground truth and aggregated segment consumption

To assess the validity of our consumption estimates, we will use the standard deviation and the mean difference between the maximum cumulative value from the dataset and the summation of all road segment battery values. Table 3.6 depicts these differences, showing that the spread and the mean difference are sufficiently small. This indicates that our dataset is reliable for estimating battery consumption and building a predictive model.

3.4.3 Building the Data for Prediction

After calculating the road segment consumption and addressing the issues in our dataset, we have developed a comprehensive data preprocessing pipeline. Figure 3.9 illustrates the complete data preprocessing pipeline. This pipeline includes all the steps from raw data collection to the final preparation of road segment battery consumption data. Each stage in the pipeline is designed to ensure the data is accurate, comprehensive, and ready for input into our predictive model.

With this robust preprocessing pipeline, we have a reliable base dataset that can be effectively used for building and training our prediction model. This comprehensive approach ensures that our data is well-prepared to yield accurate and meaningful predictions about battery consumption.

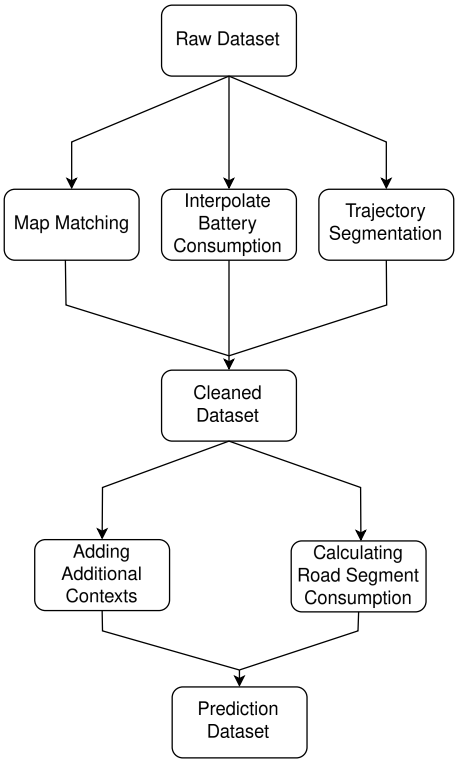


Figure 3.9: Data cleaning and preprocessing pipeline

3.5 Summary

In this chapter, we have thoroughly explained how we explored and addressed the challenges in the dataset. Our exploration involved identifying problems within the dataset and uncovering the challenges present. Furthermore, we developed a careful approach to address these challenges and applied appropriate techniques to resolve them. These techniques are crucial for effectively addressing the issues in the dataset.

Techniques such as map matching are essential, as they clean inaccurate GPS positions and help identify the road segments associated with each GPS point. Additionally, segmenting the trajectory aids in identifying stopping points in our dataset, dividing the trajectory into several smaller trips for more detailed analysis. Linear interpolation is used to estimate battery consumption at the trajectory point level, providing a more granular understanding of energy usage.

Building on these techniques, we ensure the quality of the extracted data compared to our raw data by employing metrics such as mean and standard deviation to measure daily battery

consumption differences. This evaluation method resulted in a mean of only 0.8 kWh per day and a standard deviation of 0.5 kWh, indicating a high level of consistency in our estimation process.

However, while our current methods have proven effective, there is room for further enhancement, particularly in the estimation of battery consumption. By utilizing more advanced interpolation methods that account for unseen factors in the trajectory data, we can improve accuracy beyond linear interpolation. Additionally, incorporating detailed traffic information, roundabouts, and vehicle speed could further enrich our dataset. The authors in [68, 69] highlight the significant effects of roundabouts and speed on electric vehicle battery consumption, suggesting that these factors should be integrated into our prediction models for improved accuracy.

In conclusion, we have employed an exhaustive approach to address the challenges present in our vehicle trajectory dataset. We have successfully estimated battery consumption at the road segment level and validated it against the raw trajectory dataset using daily consumption levels. However, further improvements are worth exploring to enrich the dataset and to refine the estimation of battery consumption from cumulative values.

4

Prediction for the Electric Bus Usecase

In this chapter, we will explain the process of building prediction models from preprocessed trajectory data to predict battery consumption, step by step, starting with the baseline model and progressing to more advanced models. To facilitate this progression, Section 4.1 will detail the construction of the prediction model, outline the goals of the prediction, and describe the validation process for the model. Section 4.3 will provide a step-by-step guide on building our baseline model and analyzing the important features from the dataset.

Following the baseline model, Section 4.4 discusses how we incorporate recurrent neural networks to enhance the prediction capabilities of our model. Section 4.5 then compares the performance of the baseline model with our recurrent neural networks model. Finally, we will summarize the findings of this chapter in Section 4.6.

4.1 Building the Prediction Models

After preprocessing our dataset through the data preprocessing pipeline, we obtained cleaned data with the additional context we need. Table 4.1 shows the cleaned data after preprocessing, highlighting additional information regarding road segments and temperature. To build better predictions, we can further enrich the dataset by analyzing it and extracting more features.

Using this enriched dataset, our objective is to predict battery consumption with our model. Specifically, we aim to generate predictions at the road segment level and then aggregate these predictions to obtain a daily forecast. We plan to conduct the predictions iteratively, starting

with simple models and progressively moving to more complex ones. Initially, we will develop and validate a baseline model, followed by more complex models which will be validated against the baseline.

The baseline model will employ a simple regression technique, enabling us to understand the relationship between predictor variables and the predicted values. Subsequently, we will integrate more complex models, specifically Deep Neural Networks, to uncover hidden patterns within our dataset. It is essential to validate these models to assess their predictive performance.

We will validate the models using various performance metrics. RMSE will be used to evaluate prediction accuracy, while the standard deviation of predictions will assess consistency, and the R^2 score will be used to assess model quality. By comparing these metrics, we can determine which model performs better in terms of accuracy and reliability. This comprehensive approach will enable us to select the most effective model for predicting battery consumption.

Attribute	Data Type
Vehicle ID	String
T	Timestamp
Road Segment ID	String
Battery Consumption	Float
Trip ID	String
Temperature	Float
Humidity	Float
Precipitation	Float
Wind Speed	Float
Wind Direction	Float
Average Elevation	Float
Average Grade	Float
Average Angle	Float

Table 4.1: Prediction dataset attributes

4.2 Preparing and Analysing the Dataset

Before building the prediction model, we will explore the features of our dataset to identify patterns and extract additional information. We will start by analyzing the temporal information our dataset, and exploiting it to get broader view of its effect to the battery consumption. Additionally we will also look into the weather information to gain additional insights from the data.

4.2.1 Adding Temporal Information

We achieve this by adding details about the day of the week, indicating which day each trajectory belongs to. Additionally, we divide the time into hourly bins to provide a broader view regarding the battery consumption. This helps us identify patterns such as rush hours or the impact of holidays on battery consumption. Figure 4.1 illustrate the distribution of road segment consumption based on the day of the week and the time range.

In our analysis, we observed an upward trend in battery consumption during weekends and on Monday. Additionally, the mean battery consumption is relatively higher in the morning compared to the rest of the day, with a sudden spike towards the end of the day. This temporal context, as shown in the data, enables us to discern clearer patterns that affect battery consumption. Incorporating this enriched dataset provides valuable insights for our analysis and modeling, enhancing our understanding of the factors influencing battery usage.

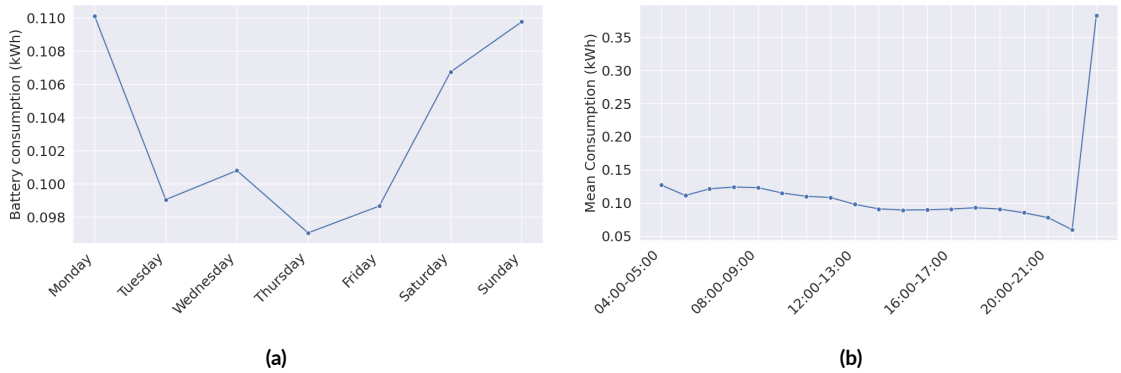


Figure 4.1: Mean battery consumption (kWh) by (a) day of the week (b) time range

4.2.2 Inspecting the Weather

After analyzing the temporal information, we will also examine weather factors to determine how precipitation and temperature affect battery consumption. This focus is due to the impact weather has on the utility usage of a bus, consequently affecting its performance. We will begin by inspecting the effect of precipitation.

To analyze precipitation, we will categorize it for easier interpretation. According to [70], precipitation can be classified into four distinct levels: no rain, light rain, moderate rain, and heavy rain. After preprocessing the precipitation data, we will examine its impact on battery consumption. As shown in Figure 4.2, there is no significant difference in mean battery con-

sumption between no rain and moderate rain. However, during light rain, battery consumption is noticeably lower. Furthermore, our observations indicate a correlation between temperature trends and battery consumption, with increases in temperature leading to lower rates of battery consumption.

Incorporating these weather data analyses with the temporal information, we can integrate additional features—day of the week, time range, and rain status—into our model. This will aid in feature importance analysis, as we will use a decision tree-based model to determine the significance of each feature.

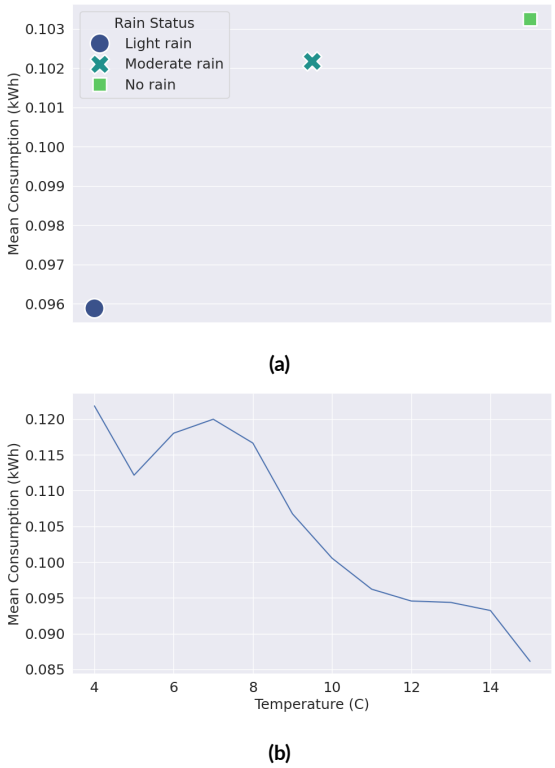
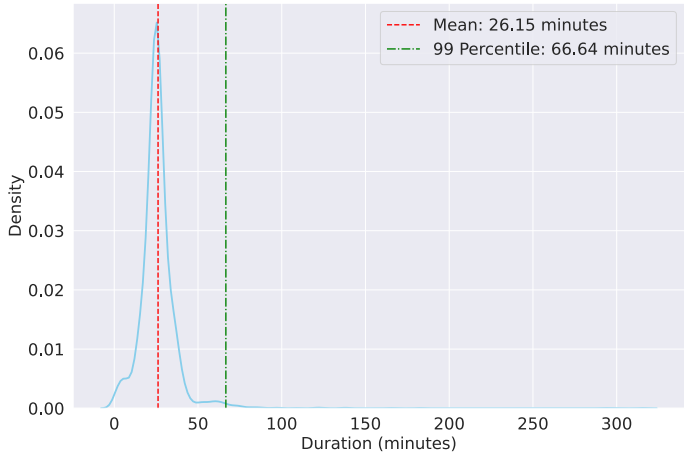


Figure 4.2: Mean battery consumption (kWh) by (a) rain condition (b) temperature

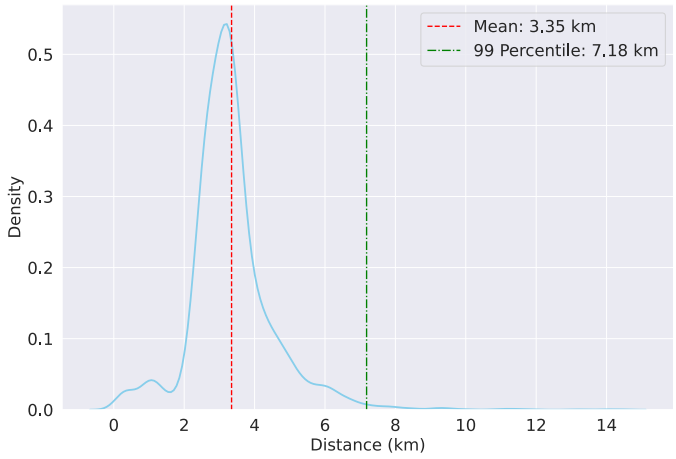
4.2.3 Analyzing Trips

Subsequently, by segmenting our data into multiple trips during the cleaning process, we can closely examine the characteristics of these trips within our trajectory data. This analysis allows us to better understand the nature of the trips and how they might influence battery consumption prediction. We begin by examining the duration of the trips. As shown in Figure 4.3, the

average trip length is 26 minutes and 10 seconds, with 99% of the trips lasting less than 66 minutes and 38 seconds. This indicates that the trips in the dataset are of moderate duration, with the longest trips being relatively manageable in length.



(a)



(b)

Figure 4.3: Trip distribution density for all trajectories by (a) duration (hours) (b) distance (km)

In addition to trip duration, we also analyze the distance of the trips. As illustrated in Figure 4.3, the average trip distance is 3.35 kilometers, with 99% of the trips covering less than 7.18 kilometers. These findings suggest that the trips are sufficiently long to provide meaningful data for analysis, yet not excessively long, ensuring that the dataset remains practical for

modeling and prediction tasks. This analysis confirms that the trips contain enough data to support accurate and reliable predictions.

4.3 Establishing the Baseline Model

In this section, we will explain the process of building our baseline model. Establishing a baseline model is crucial as it provides a foundation for comparison with more complex models. A simple and interpretable model serves as a starting point, allowing us to understand the basic patterns in the data and set a benchmark for future improvements.

Following feature selection, we introduce techniques to choose the best model, ensuring robustness and accuracy. The final step involves validating the model to confirm its predictive performance. Additionally, we will experiment by creating separate models for each vehicle type and comparing their performance with our baseline model. This comparison will help us understand the nuances in the data and refine our approach.

4.3.1 Feature Selection and Engineering

One of the methods that will assist us with feature selection is utilizing tree-based algorithms. In [71, 72], the authors explored the use of random forests and Gini importance for feature selection. In this study, we will employ a Random Forest Regressor to determine the most significant features using Gini importance. We will fit our training dataset into the random forest regression model and obtain the Gini importance for each feature. Figure 4.4 illustrates the Gini importance of each feature. As demonstrated, features such as mean elevation, segment length, and average angle are among the most crucial for making accurate predictions.

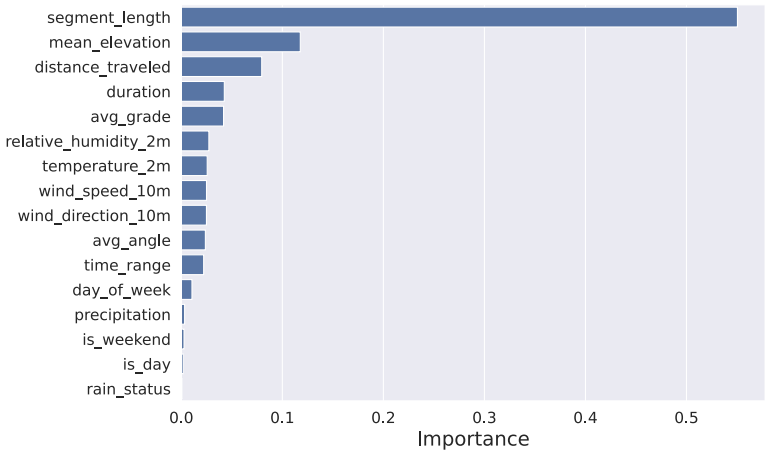


Figure 4.4: Gini importance of each features

After obtaining the candidate features through Gini importance, we will further preprocess our data. Specifically, we will translate the categorical features using one-hot encoding. This approach is optimal because converting categorical data into numerical features works best with linear regression, which we will use for our baseline model.

Additionally, we will scale the values of our data using a scaler to normalize the feature magnitudes. Normalizing the features is crucial because it ensures that all features are on a similar scale, preventing features with larger magnitudes from dominating the learning process and mitigating the effects of differing units of measurement across features. After implementing these preprocessing steps, we will have a complete dataset ready to feed into our model.

4.3.2 Training the Models

To build our baseline model, we will begin by splitting our dataset into training and testing sets. Additionally, we will utilize Grid Search and Cross Validation to obtain the optimal parameters for our models. We will also explore different combinations of models: one trained on the entire training set of all vehicles and others trained on training data divided by each vehicle.

The dataset will be split into training and validation sets using a 70-30 ratio. This split ratio is chosen to ensure that we have sufficient data for training while retaining enough validation data to assess the generalizability of our predictions. The data will be split based on individual trips, ensuring that the training and validation sets contain distinct, non-overlapping trips, allowing us to more effectively evaluate the model's ability to generalize to unseen data.

After splitting the data, the next step is to identify the best parameters for our models. We

aim to determine the optimal number of features for use in our regression algorithm. For this, we will employ Grid Search with Cross Validation and Recursive Feature Elimination (RFE) as estimators. This approach ensures that the number of features used in the regression algorithm is optimal.

Figure 4.5 illustrates the effect of the number of features on the performance of the Linear Regression model. Through this method, we determined that using 28 features is the most optimal. Subsequently, we will utilize these parameters in building our Linear Regression model and fit the training data to the model.

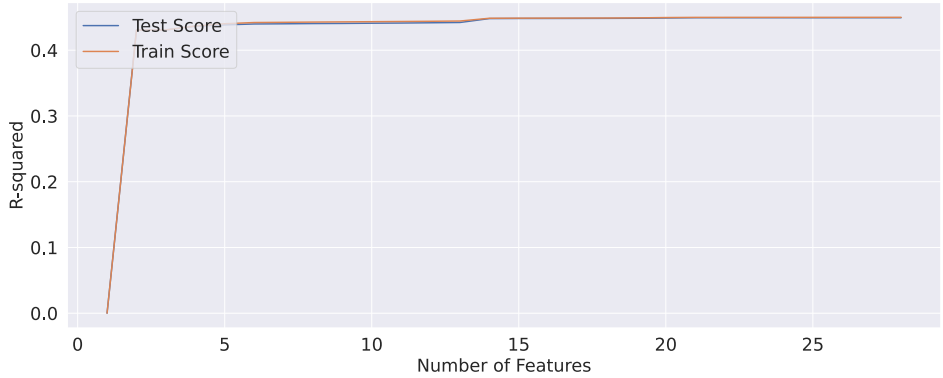


Figure 4.5: Number of optimal parameter searches using K-fold and grid search

In addition to the model trained on data from all vehicles, we will also train separate models for each vehicle, resulting in a total of 15 models. This approach allows us to evaluate performance and determine whether individualized models perform better in capturing the specific data of each vehicle. The rationale behind this is that each vehicle might behave differently. However, the drawback is that more models need to be trained if the number of vehicles increases.

4.3.3 Baseline Models Validation

After training our models, the next step is to validate their performance using the test dataset. This validation involves comparing the predicted battery consumption for each trip against the actual consumption values from the raw, unprocessed data, which serves as the ground truth. We will use RMSE to quantify the deviation of the predicted values from the actual values. Additionally, we will calculate the standard deviation to assess the consistency of the

predictions, and the R^2 score to evaluate the models' ability to approximate the actual values based on the predictor variables.

We will compare the performance between the model trained on data from all vehicles, referred to as the *Unified Vehicle Model*, and the models trained separately on data from different vehicles, referred to as *Individual Vehicle Models*. The first comparison, as depicted in Table 4.2, aggregates the predictions for all trips.

R^2		RMSE		STD	
Individual	Unified	Individual	Unified	Individual	Unified
0.457	**0.491**	2.285	**2.213**	2.284	**2.206**

Table 4.2: Baseline models performance comparison

The Unified Vehicle Model performs slightly better than the Individual Vehicle Models. However, both models exhibit relatively low R^2 scores, with values below 0.5, indicating that neither model can explain more than 50% of the variance in the data. The RMSE for each trip is approximately 2.2 kWh, which is a moderate value. However, the standard deviation is quite high, also around 2.2 kWh, suggesting considerable variability in the predictions. In the next section, we will analyze the performance at the individual vehicle level to determine whether training a model for each vehicle offers any advantages.

In Table 4.3, it is evident that the Unified Vehicle Model outperforms the Individual Vehicle Models in terms of overall performance. However, the Individual Vehicle Models demonstrate greater stability. The RMSE for the Unified Vehicle Model is generally lower compared to that of the Individual Vehicle Models. When examining the R^2 scores, we observe that both models exhibit relatively low scores for most vehicles, indicating a limited ability to explain the variability in the data.

	R^2		RMSE		Std	
Vehicle	Individual	Unified	Individual	Unified	Individual	Unified
Vehicle 1	**0.638**	0.575	**1.703**	1.845	**2.116**	2.214
Vehicle 2	0.375	**0.434**	2.127	**2.024**	2.157	**2.139**
Vehicle 3	**0.357**	0.304	**2.204**	2.293	**2.409**	2.555
Vehicle 4	-0.015	-0.014	2.479	**2.478**	2.276	**2.264**
Vehicle 5	**0.513**	0.506	2.135	**2.149**	**2.377**	2.433
Vehicle 6	0.368	**0.412**	2.727	**2.631**	**2.718**	2.782
Vehicle 7	0.387	**0.433**	2.474	**2.380**	**2.060**	2.131
Vehicle 8	0.183	**0.274**	2.214	**2.087**	1.968	**1.966**
Vehicle 9	**0.446**	0.443	**2.584**	2.593	**2.674**	2.679
Vehicle 10	0.568	**0.627**	1.905	**1.770**	**2.333**	2.576
Vehicle 11	-0.010	-0.025	**2.515**	2.535	**2.379**	2.567
Vehicle 12	0.268	**0.317**	1.899	**1.834**	**1.972**	2.023
Vehicle 13	0.778	**0.811**	2.079	**1.921**	**3.629**	3.802
Vehicle 14	0.102	**0.223**	2.269	**2.111**	**2.496**	2.500
Vehicle 15	0.617	**0.683**	2.499	**2.274**	**3.382**	3.595

Table 4.3: Baseline models performance comparison by vehicle

Overall, our predictions were made at the road segment level and validated against the actual consumption values at the trip level. This approach was chosen to provide a more comprehensive understanding of the predictions and to enable meaningful analysis, particularly in determining the battery consumption for each trip. The results indicate that both baseline models generally exhibited relatively moderate performance, with an RMSE of approximately 2.2 kWh and R^2 scores below 0.5, suggesting that the models were unable to explain more than half of the variance in the data. Additionally, no significant benefit was observed in training the model individually for each vehicle. Consequently, we have decided to use the Unified Vehicle Model as the baseline for our predictions.

4.4 Applying LSTM Model

After building our baseline model, we aim to improve our predictions to make them more accurate and consistent. One approach is to utilize deep learning methods. Given that our dataset contains over 200,000 trajectory points, deep learning models may be particularly useful in uncovering hidden patterns within the data. To select the appropriate model, it is crucial to

thoroughly understand the nature of our data.

The trajectory data can essentially be viewed as a sequence sorted by timestamp. Similarly, road segments within the trajectory can also be considered as sequential data. This sequential nature suggests that attributes from previous sequences might influence the battery consumption of subsequent ones. In this context, we can employ RNN, particularly LSTM networks.

LSTM networks are well-suited for handling sequence data as their architecture maintains and regulates the flow of information over extended periods. The authors in the previous research [45, 46] have successfully utilized LSTMs for prediction tasks. In this study, we will incorporate embedding layers to represent categorical features within our model. To determine the optimal parameters, we will conduct hyperparameters tuning. After building the model, we will validate it using the same metrics as the baseline model: RMSE and R^2 for measuring model performance, and standard deviation to assess the consistency of the predictions.

4.4.1 Defining the Sequences

To construct the sequences, we employ a rolling window approach on our trajectory data. Specifically, we use a window size of four. In this setup, the first three sequences within each window serve as the input, while the fourth sequence is used as the output for prediction. Table 4.4 is the trajectory data that we will process as sequences, and as shown in Table 4.5 here is how we define the sequences with window size of four from the trajectory data.

Trajectory	Feature 1	Feature 2	Target
T_1	T_1F_1	T_1F_2	T_1Y
....	....	....	....
T_n	T_nF_1	T_nF_2	T_nY

Table 4.4: Trajectory dataset definition

Input Sequence	Target
$[\{T_1F_1, T_1F_2\}, \{T_2F_1, T_2F_2\}, \{T_3F_1, T_3F_2\}]$	T_4Y
$[\{T_2F_1, T_2F_2\}, \{T_3F_1, T_3F_2\}, \{T_4F_1, T_4F_2\}]$	T_5Y
$[\{T_3F_1, T_3F_2\}, \{T_4F_1, T_4F_2\}, \{T_5F_1, T_5F_2\}]$	T_6Y
$[\{T_4F_1, T_4F_2\}, \{T_5F_1, T_5F_2\}, \{T_6F_1, T_6F_2\}]$	T_7Y

Table 4.5: Input sequences and their corresponding targets

Additionally, we incorporate padding into the windows to enhance our data and address cases where the sequence length is less than the window size. The windows with padding are shown in Table 4.6 We use zero as the padding value to indicate that the padding contains no information. This approach ensures that we do not lose information at the beginning or end of the sequences. The rolling windows with padding will be used to implement the sequence of our dataset effectively

Input Sequence	Target
$[\{0,0\},\{0,0\},\{0,0\}]$	T_1Y
$[\{0,0\},\{0,0\},\{T_1F_1,T_1F_2\}]$	T_2Y
$[\{0,0\},\{T_1F_1,T_1F_2\},\{T_2F_1,T_2F_2\}]$	T_3Y
$[\{T_1F_1,T_1F_2\},\{T_2F_1,T_2F_2\},\{T_3F_1,T_3F_2\}]$	T_4Y
$[\{T_2F_1,T_2F_2\},\{T_3F_1,T_3F_2\},\{T_4F_1,T_4F_2\}]$	T_5Y

Table 4.6: Input sequences with padding

4.4.2 Embedding the Categorical Features

Our dataset also contains categorical features, such as the day of the week and the time range. To handle these categorical features, we will use an embedding layer within our model. Instead of one-hot encoding the categorical data, using embeddings can capture more complex relationships between the categories.

The embedding layer is defined with a size of $N \times M$, where N is the number of unique categories and M is the dimension of the embedding. The embedding dimension is determined according to [73] as the fourth root of the number of unique categories. Using this approach, we can apply the formula in Equation 4.1. This rule of thumb allows us to quickly determine the embedding dimension.

$$M = \text{round}(\text{num\_of\_categories}^{1/4}) \tag{4.1}$$

Table 4.7 depicted the size of the categories and resulted embedding dimensions based on the Equation 4.1. As we can see, the resulted embedding dimension for day of week and time range is only two considering we only have seven categories and 19 categories respectively for these two attributes. This shows that our categories is not that complex as we do not need a lot of dimensions to capture the relationship.

Name	Number of Categories	Embedding Dimension
Day of Week	7	2
Time Range	19	2

Table 4.7: Embedding dimension of categorical attributes

4.4.3 LSTM Model Development

After creating the sequences from our data and explaining the embedding of our attributes, the next step is to build our LSTM model. We will provide an in-depth view of our LSTM network architecture and incorporate the embedding layer to process the categorical information from our attributes.

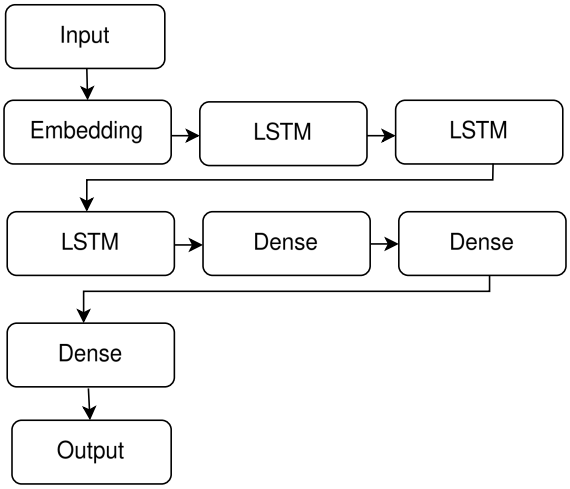


Figure 4.6: Architecture of the prediction model

Figure 4.6 shows the architecture of our model, which involves the embedding layer, LSTM layers, and dense layers. To explain the model, we first take the input, which includes categorical information. The categorical information is passed to the embedding layer. The weights in the embedding layer are initialized using Xavier initialization [74], a standard procedure that yields better performance compared to random weight initialization. We then concatenate the categorical information from the embedding layer with the input.

After concatenation, the input is passed through our LSTM layers, to capture the temporal information from our data. The first and second LSTM layers each have 128 units, followed by a final LSTM layer with 64 units. The output from the LSTM layers is then passed to the dense layers, which have 64 and 32 units, respectively. These dense layers, are used to capture

intrinsic patterns in the data beyond the temporal dependencies.

To further refine the model, we incorporate dropout layers between the dense layers to enhance generalization. Additionally, we introduce an activation function to add non-linearity to our network, ensuring it can learn complex relationships within the data. Specifically, the LSTM layers use the tanh activation function, while the dense layers use the ReLU activation function. Finally, the output layer, is designed with a single unit, as we are predicting the battery consumption of the electric bus.

4.4.4 Model Parameters

Hyperparameters	Value
Activation	ReLU, PReLU, **leaky ReLU**
Optimizer	**Adam**, Adagrad, sgd, RMSProp
Learning Rate	0.05, **0.001**, 0.0005
Batch Size	32, **64**, 128
Sequence Length	3, **4**, 5
Epoch	30
Patience	10
Delta	0.0005

Table 4.8: Hyperparameters for training

After developing the model architecture, we will outline the configuration of our hyperparameters for training. The training process will span 30 epochs, with early stopping implemented to enhance training efficiency. The early stopping parameters are configured with a patience of 10 and a delta of 0.0005, meaning that if the validation loss does not improve by more than 0.0005 for 10 consecutive epochs, the training will be halted.

The hyperparameter search is conducted using a grid search algorithm, facilitated by the WandB tool*. To limit the search space, hyperparameters are tuned individually, and the best resulting parameter is selected based on the tuning results. Table 4.8 shows the selected hyperparameters and the candidates.

We will utilize the Leaky ReLU activation function [75], a variant of ReLU that addresses the dying neurons problem. Additionally, we will employ the Adam optimizer [76], which effectively combines the benefits of Adagrad [77] and RMSProp [78]. The optimizer will be

*Weights & Biases: The AI Developer Platform

configured with a learning rate of 0.001 to ensure a balanced learning pace. The batch size will be set to 64, and the sequence length will be set to four.

4.4.5 Training the Model

After defining all the parameters, the next step is to train the model. The dataset is split into three splits: 50% for training, 20% for validation, and 30 % for testing. This split ensures sufficient data for training while maintaining the same size for the testing data as the baseline model. Once the data split is defined, we proceed with training the model.

During training, the model is trained on the training dataset and validated against the validation dataset in each epoch. The validation loss is monitored to determine whether the model has reached convergence, utilizing the early stopping mechanism we implemented. Furthermore, we save the best model weights based on the validation loss. These best weights are then loaded and used to evaluate the model against the testing dataset.

4.4.6 LSTM Model Validation

The next step is to validate the model's performance using the testing dataset. We will evaluate the model with the same metrics as the baseline model: RMSE and R^2 score to assess the quality of the predictions, and standard deviation to evaluate the consistency of the predictions.

RMSE	STD	R^2
1.315	1.31	0.820

Table 4.9: Model performance by metrics

First, we validate the model's overall performance. As shown in Table 4.9, the model demonstrates good and consistent predictions, as indicated by the R^2 scores, RMSE, and standard deviation values. An RMSE of 1.315 kWh suggests that the predictions do not deviate significantly from the actual values for each trip. Additionally, with an R^2 score of 0.820, the model is able to explain a substantial portion of the variance in the data. The model also shows consistency in its predictions, with a standard deviation around 1.31 kWh.

Overall, the LSTM model shows a marked improvement in prediction accuracy compared to previous models. The next step is to compare this LSTM model with the baseline model and provide more detailed insights into the model's performance and its behavior on our dataset. This analysis will allow us to examine the predictions in greater depth.

4.5 Comparative Analysis between Models

In this chapter, we present a comparative analysis between the baseline model and the LSTM model. The comparative analysis employs metrics such as RMSE and R^2 to assess model accuracy and prediction quality, while the standard deviation (Std) is used to evaluate model consistency. We will analyze the overall performance of the model first, and looks into the prediction based on the duration of the trip.

As illustrated in Table 4.10, the LSTM model outperforms the baseline model in terms of prediction quality, accuracy, and consistency. The LSTM model shows significant improvements across all metrics compared to the baseline. However, to gain further insights, we will also compare the predictions based on trip duration to uncover any underlying prediction patterns.

	Overall RMSE		Overall R^2		Overall Std	
Model	LSTM	Unified	LSTM	Unified	LSTM	Unified
Value	**1.315**	2.213	**0.820**	0.491	**1.31**	2.206

Table 4.10: Overall comparison between the baseline and LSTM models

As shown in Figure 4.7, the RMSE values in the predictions exhibit a pattern based on trip duration. Generally, the longer the trip duration, the slightly less accurate the model becomes. This trend is expected, as the prediction is built from the road segment level; longer trips involve more segments, leading to a compounding of errors at the aggregated battery consumption prediction at the trip level.

Additionally, there is a noticeable deviation in the predictions for trips with durations between 50 and 60 minutes. To explore this further, we examine the values within this specific duration bin. As shown in Figure 4.8, the usual battery consumption for trips lasting 50 to 60 minutes is typically under 20 kWh. However, for this particular trip, the actual battery consumption is 23 kWh. This discrepancy likely leads the model to underestimate the prediction. To extend the analysis, we will examine predictions with similar consumption values.

As shown in Figure 4.8, other trips with battery consumption values in the range of 20-25 kWh in the real data are predicted more accurately by the model. This suggests that, for this particular trip, there may be an unseen factor not captured by our predictors, leading to the spike in battery consumption and impacting the model's performance. However, aside from

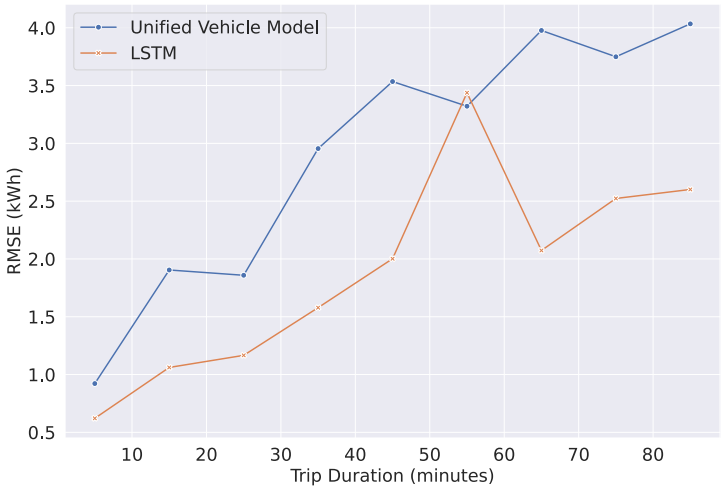


Figure 4.7: Prediction RMSE (kWh) by duration of the trips

this anomaly, the model generally performs well and can reliably predict battery consumption for most trips, particularly when using the LSTM model.

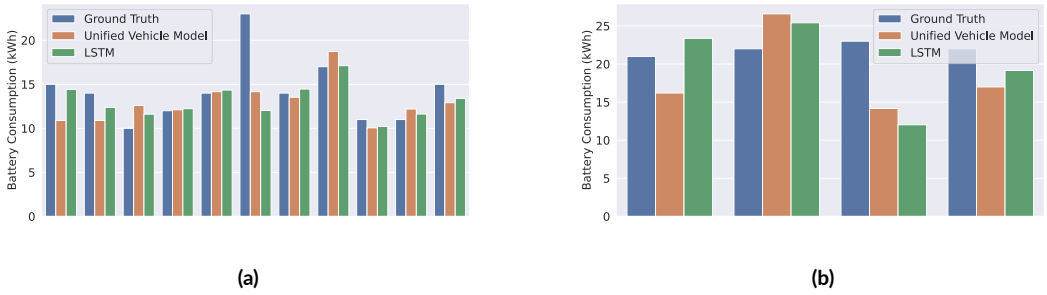


Figure 4.8: (a) Comparison of predicted vs. actual battery consumption (kWh) in the 50-60 minutes range (b) Comparison of predicted vs. actual battery consumption in the 20-25 kWh range

4.6 Summary

In this chapter, we constructed our prediction models step by step, starting with the baseline model implemented using Linear Regression due to its simplicity and ease in explaining the variables. We experimented with building a Unified Vehicle Model, trained on data from all vehicles, as well as Individual Vehicle Models, where each model was trained on data from a

single vehicle. Progressing from the baseline model, we employed recurrent neural networks, specifically LSTM, to develop a more advanced model by treating the data as a sequence, leveraging the fact that the trajectories were already sorted by timestamp. This sequential approach allowed us to capture dependencies over time, leading to a more nuanced understanding of battery consumption patterns. Subsequently, we conducted a comparative analysis between the baseline and LSTM models to better understand their performance differences.

Our analysis demonstrated that training individual models for each vehicle did not yield performance gains, leading us to adopt the Unified Vehicle Model as our baseline. In comparing the LSTM model with the baseline, the LSTM model consistently showed superior performance. By incorporating an embedding layer, the LSTM model effectively captured complex relationships within the categorical features. Additionally, by modeling the data as a sequence, the LSTM was able to account for the influence of prior sequences on the current sequence's battery consumption.

We also observed that the duration of the trips impacted prediction performance, with the models being relatively sensitive to certain outliers. Specifically, for trips with unusually high consumption compared to similar trips of the same duration, our model tended to underestimate the predictions. This suggests that incorporating additional contextual factors, such as vehicle load, speed, or traffic information, could enhance model performance in future iterations.

Overall, our models, particularly the LSTM model, demonstrated the ability to generate reliable and accurate predictions of battery consumption across trips.

5

Ship Trajectory Prediction Usecase

Trajectory prediction is a fundamental problem in the analysis of trajectory data. The prediction of an object's future movement has been extensively studied, particularly in the field of computer vision, with a significant focus on human trajectory prediction. Prior research in this area, such as [47, 48, 49], has primarily addressed short-term human trajectory prediction. Additionally, there has been substantial research on predicting longer-term trajectories, as demonstrated by studies like [50, 51]. Beyond computer vision, trajectory prediction has also been applied to raw trajectory data across various domains.

The raw trajectories contain latitude and longitude information, along with additional contextual details about the trajectory. According to [14], prior research has been conducted in diverse environments, including maritime, artificial, and urban settings. In the maritime environment, trajectory prediction can be utilized to implement intelligent transportation systems. This application allows for better control of sea traffic, thereby enhancing safety at sea. In this chapter, we will focus specifically on the maritime environment, with an emphasis on predicting future ship trajectories as a use case. By doing so, we aim to contribute to the development of more efficient and safer maritime navigation systems.

To provide a structured approach to this research, Section 5.1 provides a brief overview of our ship prediction use case. Section 5.2 discusses the challenges and approaches in cleaning ship trajectory data. Section 5.3 outlines our approach to predicting ship trajectories. Finally, Section 5.4 summarizes the work done and suggests future improvements for this chapter.

5.1 Overview

To illustrate the diverse approaches required for addressing various problems and challenges in predicting trajectory data, another use case is incorporated. This case aims to demonstrate that trajectory data necessitates distinct methodologies depending on the specific issue at hand. Furthermore, it highlights the management of different types of trajectory data, including larger datasets and denser trajectories. In this use case, we will utilize the AIS dataset provided by the Danish government [27].

A prevalent issue associated with the AIS dataset is predicting ship trajectories based on historical data. Numerous studies have explored this domain. For instance, the author in [24] utilized an LSTM network with a TimeDistributed dense layer to predict ship trajectories in the Strait of Singapore, whereas the author in [25] employed LSTM alongside additional feature engineering to forecast ship trajectories near Sweden.

These trajectory prediction tasks present several challenges that necessitate meticulous data processing and precise prediction methods. The inclusion of this use case will demonstrate the extensive range of applications for trajectory data prediction and underscore the unique challenges posed by varying trajectory data contexts.

5.2 Data Preprocessing

In this use case, we will utilize the AIS dataset provided by the Danish government, which encompasses comprehensive historical ship data from the waters surrounding Denmark, with records spanning from 2006 to the present. Given the extensive volume of data, our analysis will be confined to a three-day period, and a bounding box will be applied to limit the area of the trajectory. This approach ensures a more manageable dataset for preprocessing and analysis.

Our analysis will focus specifically on tanker, cargo, and passenger ships. We will exclude moored, anchored, and aground ships based on their status. Furthermore, we will filter the AIS data based on attributes such as Speed Over Ground (SOG), Course Over Ground (COG), Rate of Turn (ROT), and heading, ensuring these attributes fall within valid ranges as detailed in Table 5.1. This initial filtering stage will facilitate more sophisticated preprocessing of our dataset.

The original dataset comprises various pieces of information beyond spatial and temporal data. To maintain simplicity, as shown in Table 5.2, we will only incorporate key attributes from the dataset. The primary attributes used in our analysis will be timestamp, latitude, lon-

Attribute	Unit	Valid Value	N.A Default Value
Longitude	Degrees	[-180, 180]	181
Latitude	Degrees	[-90, 90]	91
Rate of Turn (ROT)	Degrees / m	[-127, 127]	128
Speed Over Ground (SOG)	Knots	[0, 102.2]	102.3
Course Over Ground (COG)	Degrees	[0, 359.9]	360.0
Heading	Degrees	[0, 359]	511

Table 5.1: Range and default values of AIS attributes [5]

gitude, and SOG. In the following sections, we will provide an in-depth explanation of our data preprocessing methods before making any predictions.

Attribute	Datatype
MMSI	String
T	Timestamp
Longitude	Float
Latitude	Float
Speed Over Ground (SOG)	Float

Table 5.2: AIS attributes used for data prediction

5.2.1 Analyzing the Trajectories

Before proceeding with the data cleaning process, it is essential to thoroughly explore our dataset to understand its characteristics and uncover any underlying patterns. This preliminary analysis will provide a foundation for the subsequent data cleaning steps. We will analyze the trajectories grouped by the MMSI number of each ship.

Figure 5.1 presents the average duration of the ship journeys, which is approximately 21 hours, with only 25% of journeys lasting less than 3 hours and 25 minutes. Additionally, the average distance of the ship trajectories, indicating that on average, each ship travels 62 kilometers. These extensive voyages do not imply continuous sailing; there may be stops or gaps in the trajectories due to lost signals or errors in signal transmission. Consequently, it will be necessary to segment these long trajectories into smaller, more manageable segments.

Building on the observation of lengthy trajectories, we will also examine the SOG of the ships. Analyzing the SOG provides insights into the velocity variations of vessels at sea. As depicted in Figure 5.2, the average SOG distribution reveals that the average speed over ground

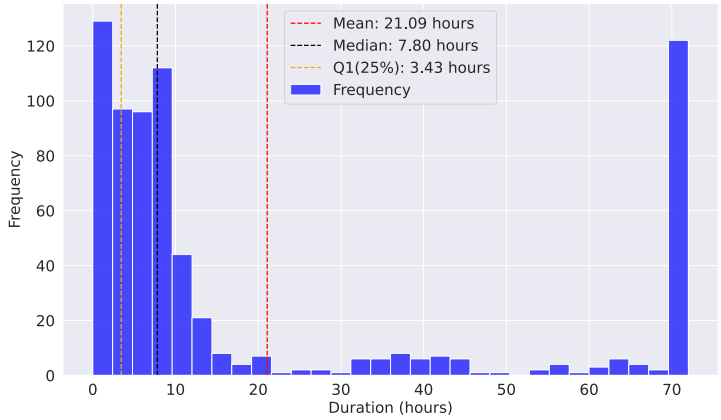


Figure 5.1: Distribution of vessels' trajectories durations (hours)

is approximately 2.59 knots, with 25% of the ships traveling at a speed of 0.23 knots, which is almost zero. This suggests that many ships travel slowly, with some nearly stopping during their journeys.

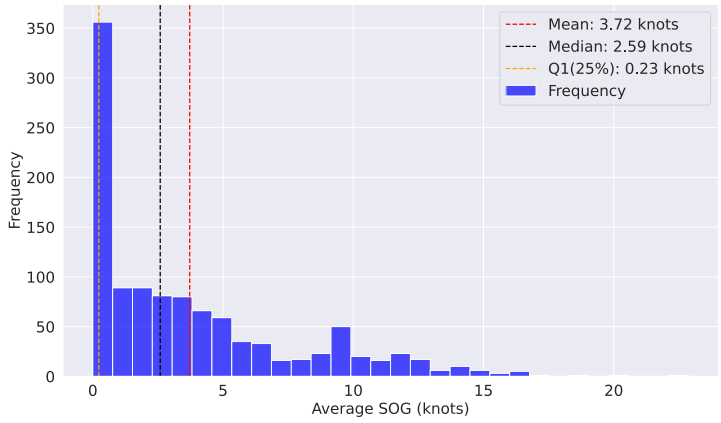


Figure 5.2: Distribution of vessels' average SOG (knots)

To gain a more detailed understanding, we will further analyze the SOG data. Figure 5.3 shows the evolution of the SOG for a particular ship, highlighting variations in speed and instances where the SOG is zero or very low and also highlight the ship trajectory over time. We observe that the ship accelerates at times and then decelerates, maintaining a SOG near zero. This analysis suggests that it may be beneficial to remove these stopping points to refine our

data further.

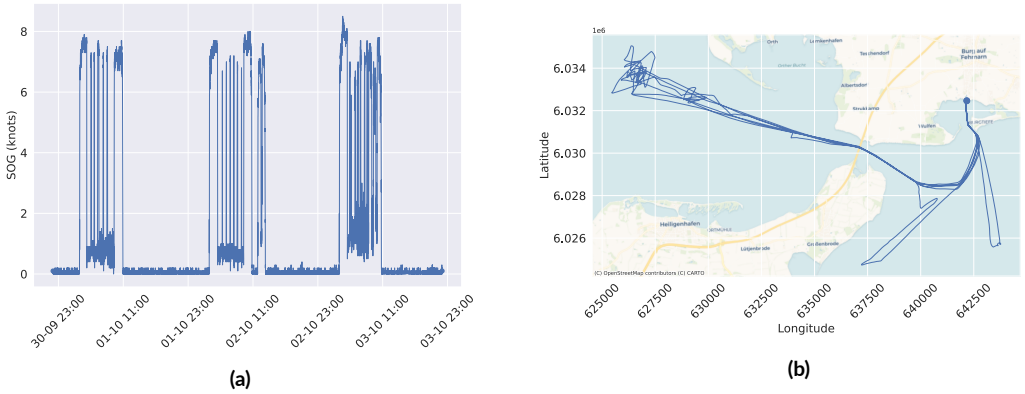


Figure 5.3: (a) Ship SOG (knots) attribute over the entire course (b) Ship trajectories over the entire course

With the insights gained from this preliminary analysis, we are now prepared to move on to the trajectory cleaning process. In the following section, we will explain the methods used to clean the trajectory data and prepare it for prediction.

5.2.2 Cleaning the Trajectories

After analyzing the trajectories, the next step is to clean the data before making predictions. This step is crucial as it ensures the creation of a clean dataset for our predictive models. In this cleaning process, we will utilize tools such as MobilityDB [10] and PyMEOS [11]. These tools simplify data preprocessing by providing native functions to handle trajectory data effectively.

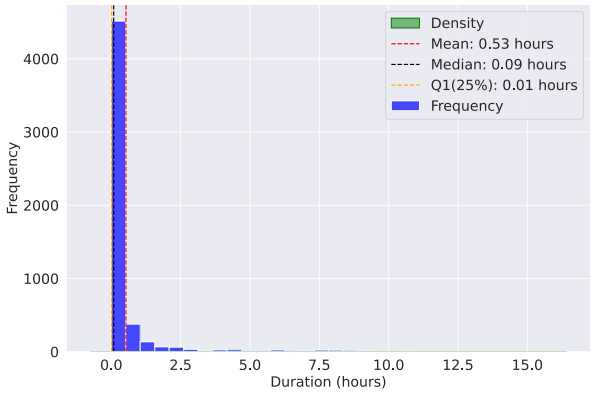


Figure 5.4: Distribution of ships' trip durations (hours) after gap splitting

To begin the cleaning process, we must address the long voyages of the ships, as there may be gaps in the journeys. We define a gap as a 10-minute interval between consecutive timestamps or a 10-kilometer distance between points. Using these criteria, we will split the trajectory data at these gaps. This process will be carried out using MobilityDB in PostgreSQL, as demonstrated in Listing 5.1.

Listing 5.1: Splitting trajectory at defined gaps

```
trip = unnest(
    sequences(
        tgeompointSeqSetGaps(
            array_agg(tgeompoint(
                st_transform(geom, 25832), T
            )
            ORDER BY T),
            interval '10mins', 10000
            )
        )
)
```

These commands will identify and split sequences that have gaps exceeding the defined maximum time interval or distance. By segmenting the trajectories at these gaps, we can effectively manage significant interruptions in the data, converting one extensive trajectory into multiple separate trips. This step is crucial for ensuring the accuracy and reliability of our subsequent analysis. In addition to this, we also filter out trips where the speed is less than 2 knots, as

traveling at such a slow speed can be considered stationary in the data.

After applying this function, multiple trips will be obtained from a single trajectory. It is important to assess the duration and length of the processed trajectories with gaps. As shown in Figure 5.4, 50% of the trips have a duration of only 0.09 hours (or 5 minutes), which is too short for our prediction purposes. Additionally, 25% of the trips have a distance of less than 44 meters, which is insufficient for meaningful trajectory analysis. Therefore, further filtering is necessary to ensure the data is suitable for prediction.

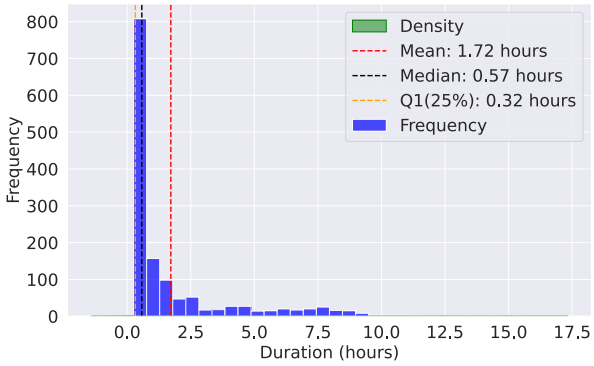


Figure 5.5: Distribution of ships' trip durations (hours) after cleaning by minimum duration and number of points

To address this, we will set a minimum trajectory duration of 15 minutes and a minimum of 40 points per trajectory. This ensures that each trip contains sufficient information for accurate trajectory prediction. As shown in Figure 5.5, the average duration is now approximately 1 hour and 43 minutes, as short trips have been removed from the data. Furthermore, the average distance is now approximately 27 kilometers, with 25% of the trips being less than 4.9 kilometers in distance. These adjustments result in longer, but still manageable, trips for the trajectory analysis.

Additionally, our dataset contains irregular time gaps between trajectory points, which could affect prediction accuracy. To mitigate this issue, we will resample our dataset at one-minute intervals. This standardizes the trajectory data, ensuring that each minute has a corresponding trajectory point. After completing this step, the trajectory cleaning process will be finalized.

Upon completing these steps, we will have a cleaned trajectory dataset ready for prediction. Figure 5.6 illustrates the cleaned trajectory data after the data preprocessing steps. In the subsequent section, we will elaborate on how trips are defined for our trajectory analysis, providing essential context and framework for the prediction process.

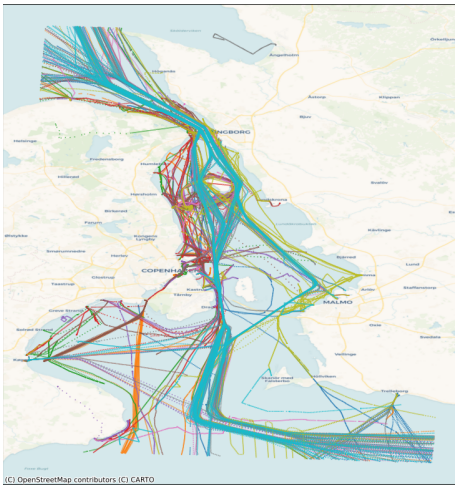


Figure 5.6: Cleaned ships' trajectories

5.3 Ship Trajectory Prediction

After cleaning our AIS dataset, the next step is to predict the ship trajectories. To build an effective trajectory prediction model, we can conceptualize the trajectory data as a sequence. Since our data is sampled at one-minute intervals, we can utilize the first X minutes of the sequence to predict the subsequent Y minutes.

Using this sequential information, we can train our model to predict future trajectories. One approach is to employ a Recurrent Neural Network (RNN) to identify patterns within the sequence. A suitable variant of RNN for this purpose is the Long Short-Term Memory (LSTM) network, which is designed for longer sequences and performs better in mitigating issues such as the exploding gradient problem compared to vanilla RNNs. This can be combined with convolutional layers.

Convolutional layers can exploit the spatial dependencies in our dataset. By leveraging these layers, we can extract spatial dependencies from the latitude and longitude information in the dataset and then combine them with the recurrent layers.

After constructing the model, we can perform predictions using the dataset. To demonstrate the flexibility of our model, we will conduct predictions with different prediction steps, utilizing a 30-minute window of historical data as the basis for our predictions.

We will evaluate the prediction performance using several metrics. The mean distance from the predicted points to the ground truth will provide an overall measure of accuracy. Additionally, we will use RMSE and MAPE to further analyze the deviation of predicted coordinates

from the actual trajectory coordinates. These evaluations will help us comprehensively assess the accuracy and reliability of our model.

5.3.1 Defining the Sequence for Prediction

To generate predictions with our model, it is essential to define the sequence from our dataset. This step is crucial as it structures the data for input into our network. Our dataset consists of multiple trajectories, and each trajectory is composed of several trajectory points. The definition of a trajectory point is provided in Equation 5.1.

$$T = \{Latitude, Longitude, Speed\ Over\ Ground\} \tag{5.1}$$

Given this data, we need to establish the window size and the prediction steps for our model. In this example, let us define the window size as five and the prediction step as one. Formally, we can define the window size for prediction as in Equation 5.2 and since we aim to predict the subsequent sequence based on the window size, the prediction steps can be defined as in Equation 5.3.

$$W = \{T_1, T_2, ..., T_{\text{window\_size}}\} \tag{5.2}$$

$$P = \{T_{\text{window\_size}+1}, ..., T_{\text{window\_size}+\text{prediction\_steps}}\} \tag{5.3}$$

Additionally, we will use a rolling window approach, meaning that we will move the window size's starting position and the prediction step's starting position one time step at a time. As shown in Table 5.3, the trajectory data has been split into windows for prediction and the corresponding prediction size. With a window size of five, we use the first five sequences to predict the subsequent sequence, in this case, one step ahead. This method allows for the flexibility to define different window sizes and prediction steps, which can be adapted for various prediction scenarios. Since the trajectory data is sampled at one-minute intervals, we can conveniently use minutes as the unit to define our window size and prediction steps.

Window	Prediction
$\{T_1, ..., T_5\}$	$\{T_6\}$
...	...
$\{T_n, ..., T_{n+4}\}$	$\{T_{n+5}\}$

Table 5.3: Sequences for the prediction

5.3.2 Explaining the Model

After defining our sequence, we can proceed to build the model for trajectory prediction. In this section, we will explain the architecture of the model, the rationale behind each layer, and the loss function used for the prediction. Figure 5.7 illustrates the general architecture of our model.

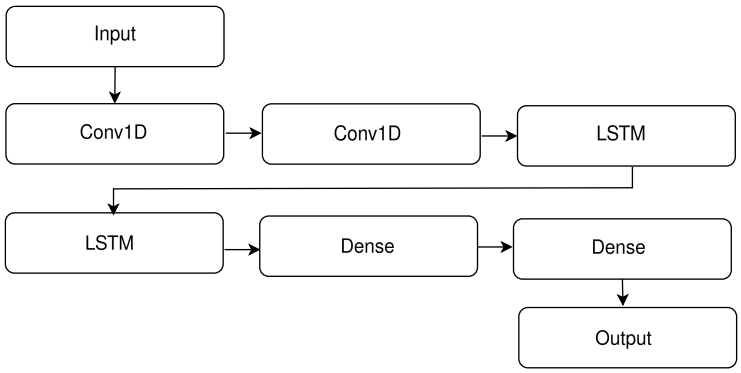


Figure 5.7: Model architecture for prediction

The convolutional layers are configured with 128 and 64 filters, respectively. These layers are applied to the latitude and longitude data to exploit the spatial dependencies present in the trajectory. Following the convolutional layers, the results are concatenated with the non-spatial features and then fed into the recurrent layers.

The recurrent component of our model comprises two Long Short-Term Memory (LSTM) layers, each containing 250 units. These LSTM layers are specifically designed to capture temporal dependencies within the sequences. In addition, we integrate a time-distributed dense layer, to model non-linear relationships within the data. The incorporation of the time-distributed dense layer enables the computation of outputs for each timestep of the LSTM, rather than solely the final timestep. Each time-distributed dense layer is configured with 150 units. After processing through the time-distributed dense layer, the model generates the predicted trajectory.

Simultaneously, to enhance the model's learning capability, we apply activation functions to each layer. In the convolutional and time-distributed dense layers, we use the Rectified Linear Unit (ReLU) activation function to mitigate issues such as the vanishing gradient problem during training. For the recurrent layers, we employ the tanh activation function, which is commonly used in recurrent networks due to its ability to handle the vanishing gradient problem and sustain long-range dependencies. By carefully selecting these activation functions, we ensure that each component of the model contributes effectively to the overall prediction performance.

$$\text{Distance} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{5.4}$$

Furthermore, we have designed a custom loss function for this model based on the Euclidean distance between coordinates. Given that we are using the planar representation of our coordinates, the distance between two points is calculated using the formula shown in Equation 5.4. The mean squared distance is then used as the loss function for training. This custom loss function aims to minimize the distance between the predicted trajectory and the actual trajectory, thereby ensuring that the predicted trajectory closely approximates the actual trajectory.

5.3.3 Hyperparameters Tuning

After defining the model, the next step is to tune the hyperparameters. To implement hyperparameter tuning, we are using Wandb as the tool to search for the optimal hyperparameters. We employ Bayesian search to tune our parameters, running it multiple times to prune improbable candidates and limit our search space. This approach allows us to efficiently identify the best hyperparameters without exhaustively iterating through every possible combination.

As shown in Table 5.4, we list all possible parameters considered for hyperparameter tuning and the selected hyperparameters for training. Additionally, we incorporate an early stopping mechanism in our training process. This means that if no improvement in validation loss is observed after a certain number of iterations, known as the patience parameter, the training process will be halted. This ensures efficient use of computational resources and prevents overfitting.

Hyperparameter	Value
LSTM layers	1, **2**, 3
LSTM hidden size	150, **250**, 300
Batch size	64, 128, **256**
Dense layer size	50, 100, **150**
Optimizer	**adam**, adagrad, rmsprop, sgd
Learning rate	**0.0005**, 0.001, 0.005
Epoch	50
Patience	20
Delta	0.5

Table 5.4: Hyperparameters for training

5.3.4 Training the Model

Additionally, we need to define the training, validation, and test splits to train our model effectively. For our case, we will allocate 60% of the data for the training set, 20% for the validation set, and the remaining 20% for the testing set.

We will train the model by iterating through a number of epochs and validating it against the validation set using our custom loss function to obtain the validation loss. Throughout the training process, we will monitor the best validation loss and use the corresponding weights to make predictions on the test set. Moreover, we will define metrics for evaluating our predictions.

As mentioned earlier, we will use several metrics to assess the prediction performance, primarily RMSE and MAPE. These metrics measure the deviation of the predicted latitude and longitude from the actual values in the test dataset. Additionally, we will calculate the mean Euclidean distance from our planar trajectory representation of the latitude and longitude to evaluate the real-world distance between the predicted and actual trajectories.

After defining these parameters, the next step is to evaluate our model. We will define several window sizes and prediction steps to demonstrate the model's flexibility. Specifically, we will use a fixed window size of 30 minutes to predict the next 1, 5, 10, 20, and 30 minutes of trajectory data. By using a fixed window size, we can analyze the effect of forecasting longer trajectories on our prediction model.

5.3.5 Prediction Results

In this section, we will discuss the prediction results using the metrics explained in the previous section. We have defined several prediction steps using a window size of 30 minutes for the trajectory data. As shown in Table 5.5, the error increases with the length of the prediction steps. This is because predicting longer sequences is inherently more challenging, and the model needs to capture dependencies over longer intervals.

Length	Steps	RMSE	MAPE	Mean Distance	Epoch
30 min	1 min	0.002461	2.347%	0.489 km	30
30 min	5 min	0.001943	1.269%	0.357 km	31
30 min	10 min	0.002834	2.416%	0.506 km	49
30 min	20 min	0.006603	5.827%	1.242 km	24
30 min	30 min	0.009515	6.847%	1.538 km	24

Table 5.5: Performance comparison for a 30-minute window with varying prediction steps

For the RMSE of the predictions, all models resulted in an RMSE of less than 0.01 degrees for the coordinates. Additionally, the MAPE of the predictions yielded less than 10%, indicating that the predicted coordinates are reasonably accurate. We also examined the mean distance in kilometers, and for all predictions, the difference from the real trajectory was less than two kilometers. On average, the model required approximately 30 epochs to train.

Figure 5.8: Model prediction on a straight trajectory

The best performance was observed with prediction steps of 5 minutes from the 30-minute window, even compared to predicting 1-minute trajectories. Although the model achieved low RMSE and a low mean distance between the predicted and actual trajectories, it struggles to accurately predict turns or stopping points. This limitation is illustrated in Figure 5.9, where deviations from the actual path are evident during maneuvers.

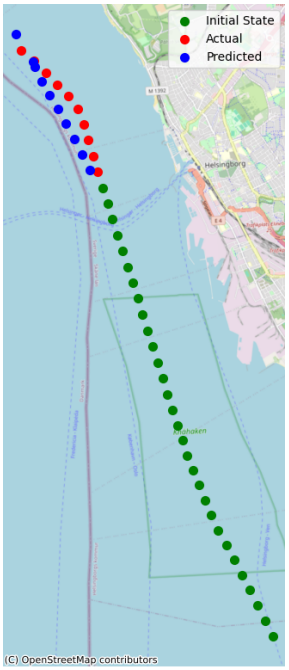


Figure 5.9: Model prediction on a trajectory with turns

Conversely, the model managed to predict the trajectory on straight trajectory segments, as shown in Figure 5.8. In these scenarios, the model demonstrates its ability to predict with minimal error, maintaining high accuracy and consistency. Overall, the prediction model exhibits strong performance in terms of accuracy and consistency, particularly for shorter prediction steps and straight trajectory segments.

5.4 Summary

To summarize this chapter, our trajectory prediction process begins with preprocessing the dataset. The data preprocessing starts with uncovering information such as average speed over ground, duration, and distance of the trajectories. Following this, we split the trajectories based on gaps to extract individual trips. Subsequently, we filter out short trips based on the number of points and the duration of the trips. Finally, we resample the trajectory data to one-minute intervals to handle irregularities, resulting in a cleaned dataset.

Despite our exhaustive approach to data cleaning, there are opportunities to extend these methods by detecting anomalies in the ship trajectories to further eliminate erroneous points. Methods such as those proposed in [79] could be integrated into our data cleaning process. Ad-

ditionally, instead of solely relying on speed over ground readings from the dataset, we could estimate the speed to achieve more accurate measurements. These improvements warrant further investigation to assess their impact on prediction accuracy.

For prediction, the optimal step for our 30-minute window is a 5-minute prediction window. This configuration achieved an RMSE of 0.001943 and a MAPE of 1.269% with the real coordinates, with a mean distance of 0.357 km from the actual trajectory. However, improvements are needed, especially in predicting maneuvers in the trajectory.

To enhance prediction accuracy, we can draw ideas from other trajectory prediction domains. For instance, [49] demonstrated the use of navigation pooling, which measures crossing probability from past observations to determine possible new position candidates. We could apply a similar approach by dividing our trajectory data into grid cells and measuring the probability of trajectory crossings within these cells. This approach poses a challenge for the ship dataset due to the vast areas covered and the sparsity of the trajectories.

In conclusion, we successfully cleaned the dataset and applied trajectory prediction for the ship trajectory prediction use case, with potential improvements for future work. Cross-domain knowledge from other trajectory prediction problems, particularly human trajectory prediction, may also offer valuable insights for enhancing our predictions.

6

Conclusion and Future Work

6.1 Conclusion

In this thesis, we applied the theoretical foundation regarding trajectories and extended its use to more advanced approaches for predictive analysis using deep learning models. We carefully preprocessed our trajectory data, identified the challenges, and applied appropriate methods to address them. In our predictive analysis, we developed a deep learning-based model, specifically an LSTM network, to perform predictive analysis on our trajectory data, recognizing that trajectories can be modeled as sequences ordered by their timestamps.

To demonstrate our approach, we presented two use cases, with the primary focus on predicting the battery consumption of electric buses. This primary use case enabled us to identify several key contributions. While previous research often involved data preprocessing, the emphasis was primarily on model improvement rather than on the trajectory data itself. In contrast, our approach involved a comprehensive cleaning and preprocessing of the trajectory data, guided by theoretical frameworks. This process began with data cleaning, followed by trajectory segmentation, and enriching the trajectories with semantic information to make them more meaningful for analysis.

In addition to these preprocessing steps, our work addressed specific challenges associated with moderate-frequency data, sampled at 1/30 Hz, as opposed to the high-frequency data typically used in prior studies. Handling this moderate-frequency data required careful consid-

eration; in this context, the map matching algorithm still performed reasonably well, allowing us to accurately retrieve the road segment data.

Finally, we tackled the challenge of working with cumulative battery consumption values, a problem not commonly addressed in previous research, which often relied on pointwise data. To address this, we utilized linear interpolation to estimate pointwise consumption values from the cumulative data. By grounding our methods in robust theoretical principles and refining our data through these processes, we successfully transformed our raw trajectories into clean, analyzable datasets that are well-suited for predictive analysis.

In the predictive analysis, we incorporated a baseline linear regression model and compared its performance with our LSTM model. The LSTM model outperformed the baseline in terms of RMSE and demonstrated better quality as indicated by the R^2 score. We evaluated our data at the trip level to provide more meaningful results. Our LSTM model achieved an RMSE of 1.31 kWh and an R^2 score of 0.820 for energy consumption at the trip level.

Additionally, we included a secondary use case involving the prediction of a ship's trajectory. In this use case, we followed similar techniques and preprocessing steps to clean the data, using the same theoretical foundation. This secondary use case demonstrates the versatility of our approach and its underlying theoretical principles. Techniques such as spatio-temporal gaps detection, and trajectory sampling were used to clean and segment the data.

For the predictive analysis in this secondary use case, we used an LSTM model to predict the ship's trajectory over different time steps within a 30-minute time window. Our experiments showed that the best prediction was achieved for a 5-minute prediction within the 30-minute window, with a MAPE of 1.2% and a mean distance error of only 0.357 kilometers from the actual trajectories.

In conclusion, our method, grounded in the literature on trajectory analysis and supported by careful identification of the challenges related to trajectory data and its preprocessing, has been demonstrated through its application in two use cases. Our research builds on the theoretical foundation by integrating more recent and practical approaches. In the primary use case, we addressed existing gaps in the prediction of electric bus battery consumption. Additionally, a secondary use case was provided to further illustrate the application of the techniques established as our theoretical basis.

6.2 Future Work

As for the future work, a more generic approach that can handle all types of trajectories, from data cleaning to prediction analysis, would be highly beneficial. Potential research in this area has been conceptualized in [80], though a complete implementation has yet to be realized. Such an approach would reduce redundancy when working with different sets of trajectories, providing a more unified method for handling various types of trajectory data.

In our primary use case, investigating more advanced estimation functions to improve the accuracy of pointwise consumption data could be highly valuable. Additionally, instead of extracting data at the road segment level, we could define segments between two bus stops along a route. This approach would provide more meaningful insights into our predictions, allowing us to analyze changes in consumption between bus stops. Furthermore, incorporating more comprehensive information regarding traffic conditions and utilizing larger datasets, such as multiple years' worth of data, would significantly enhance our analysis.

For the second use case, we plan to explore techniques from trajectory prediction in computer vision, as these methods have the potential to improve the accuracy of our trajectory predictions, particularly in better handling turns within the predicted trajectories. Additionally, incorporating contextual information, such as the presence of land within the trajectory or the interaction with other ships, is another avenue we will explore. These techniques, which have been effectively used in human trajectory prediction, could offer valuable insights in maritime contexts as well.

References

[1] G. Marketos, M. L. Damiani, N. Pelekis, Y. Theodoridis, and Z. Yan, "Trajectory collection and reconstruction," in *Mobility Data: Modeling, Management, and Understanding*. Cambridge University Press, 2013, ch. 2, pp. 23–41.

[2] Y. Zheng, "Trajectory data mining: An overview," *ACM Trans. Intell. Syst. Technol.*, vol. 6, no. 3, pp. 1–4, May 2015.

[3] M. A. Nielsen, *Neural Networks and Deep Learning*. Determination Press, 2015, creative Commons Licence.

[4] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[5] J. A. Aremu, "Quality assessment of maritime AIS data," Master's Thesis, Novia University of Applied Sciences, Jul 2023.

[6] C. Renso, S. Spaccapietra, and E. Zimányi, Eds., *Mobility Data: Modeling, Management, and Understanding*. Cambridge University Press, 2013.

[7] G. Andrienko, N. Andrienko, P. Bak, D. Keim, and S. Wrobel, *Visual Analytics of Movement*. Springer Science & Business Media, 2013.

[8] C. A. Ferrero, L. O. Alvares, and V. Bogorny, "Multiple Aspect Trajectory Data Analysis: Research challenges and opportunities," *GeoInfo*, pp. 56–67, 2016.

[9] J. Zhao, J. Mei, S. Matwin, Y. Su, and Y. Yang, "Risk-aware individual trajectory data publishing with differential privacy," *IEEE Access*, vol. 9, pp. 7421–7438, 2021.

[10] E. Zimányi, M. Sakr, A. Lesuisse, and M. Bakli, "MobilityDB: A mainstream moving object database system," in *Proc. 16th Int. Symp. on Spatial and Temporal Databases (SSTD)*. ACM, 2019.

[11] E. Zimányi, M. Duarte, and V. Diví, "MEOS: An open source library for mobility data management," in *Proc. 27th Int. Conf. on Extending Database Technology (EDBT)*, May 2024.

[12] A. Graser, "MovingPandas: Efficient structures for movement data in Python," *GI_Forum*, vol. 7, pp. 54–68, 2019.

[13] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[14] A. Graser *et al.*, "MobilityDL: A review of deep learning from trajectory data," *Geoinformatica*, 2024.

[15] A. Broniewski, M. I. Tirmizi, E. Zimányi, and M. Sakr, "Using MobilityDB and Grafana for aviation trajectory analysis," *Engineering Proc.*, vol. 28, no. 1, p. 17, 2023.

[16] S. Spaccapietra, C. Parent, M. L. Damiani, J. A. de Macedo, F. Porto, and C. Vangenot, "A conceptual view on trajectories," *Data Knowledge Engineering*, vol. 65, no. 1, pp. 126–146, 2008.

[17] S. Spaccapietra, C. Parent, and L. Spinsanti, "Trajectories and their representations," in *Mobility Data: Modeling, Management, and Understanding*. Cambridge University Press, 2013, ch. 1, pp. 3–23.

[18] M. Hickman, "Bus Automatic Vehicle Location (AVL) systems," in *Assessing the Benefits and Costs of ITS*. Springer, 2004.

[19] P. G. Furth, B. Hemily, T. H. J. Muller, and J. G. Strathman, *Using Archived AVL-APC Data to Improve Transit Performance and Management*, ser. TCRP Report. Transportation Research Board of the National Academies, 2006, no. 113.

[20] D. G. Gerstle, "Understanding bus travel time variation using AVL data," Master's Thesis, Massachusetts Institute of Technology, 2012.

[21] A. Taparia and M. Brady, "Bus journey and arrival time prediction based on archived AVL/GPS data using Machine Learning," in *Proc. 2021 7th Int. Conf. on Models and Technologies for Intelligent Transportation Systems (MT-ITS)*, 2021.

[22] H. M. Perez, R. Chang, R. Billings, and T. L. Kosub, "Automatic Identification Systems (AIS) data use in marine vessel emission estimation," in *Proc. 18th Annual Int. Emission Inventory Conf.*, vol. 14, 2009.

[23] T. Devogele, L. Etienne, and C. Ray, "Maritime monitoring," in *Mobility Data: Modeling, Management, and Understanding*. Cambridge University Press, 2013, ch. 11, pp. 221–239.

[24] A. W. Multazam, "Maritime data open access and analytic," Master's Thesis, National University of Singapore, Jul 2020.

[25] N. Mehta, "Ship trajectory prediction in confined waters," Master's Thesis, Norwegian University of Science and Technology, Jul 2023.

[26] D. Nguyen and R. Fablet, "A Transformer network with sparse augmented data representation and Cross Entropy Loss for AIS-Based vessel trajectory prediction," *IEEE Access*, vol. 12, pp. 21 596–21 609, 2024.

[27] Danish Maritime Authority, "AIS data," https://www.dma.dk/safety-at-sea/navigational-information/ais-data, 2024.

[28] C. Parent *et al.*, "Semantic trajectories modeling and analysis," *ACM Comput. Surv.*, vol. 45, no. 4, aug 2013.

[29] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Trans. ASME J. Basic Eng.*, vol. 82, no. Series D, pp. 35–45, 1960.

[30] L. Li, X. Chen, Q. Liu, and Z. Bao, "A data-driven approach for GPS trajectory data cleaning," in *Database Systems for Advanced Applications*. Springer, 2020.

[31] P. Chao, Y. Xu, W. Hua, and X. Zhou, "A survey on Map-Matching algorithms," in *Databases Theory and Applications*. Springer, 2020.

[32] P. Newson and J. Krumm, "Hidden Markov map matching through noise and sparseness," in *Proc. 17th ACM SIGSPATIAL Int. Conf. on Advances in Geographic Information Systems*, 2009.

[33] B. Hummel, "Map matching for vehicle guidance," in *Dynamic and Mobile GIS*. CRC Press, 2006.

[34] J. Krumm, "A markov model for driver turn prediction," in *Society of Automotive Engineers (SAE) World Congress,*, 2008.

[35] W. Meert and M. Verbeke, "HMM with non-emitting states for map matching," in *Proc. European Conf. on Data Analysis (ECDA)*, Paderborn, Germany, July 2018.

[36] J.-H. Haunert and B. Budig, "An algorithm for map matching given incomplete road data," in *Proc. 20th ACM SIGSPATIAL Int. Conf. on Advances in Geographic Information Systems*. ACM, 2012.

[37] A. Cappiello, I. Chabini, E. Nam, A. Lue, and M. A. Zeid, "A statistical model of vehicle emissions and fuel consumption," in *Proc. 5th Int. IEEE Conf. on Intelligent Transportation Systems (ITSC)*, 2002.

[38] B. O. Varga, A. Sagoian, and F. Mariasiu, "Prediction of electric vehicle range: A comprehensive review of current issues and challenges," *Energies*, vol. 12, no. 5, p. 946, 2019.

[39] F. Perrotta, T. Parry, and L. C. Neves, "Application of machine learning for fuel consumption modelling of trucks," in *Proc. 2017 IEEE Int. Conf. on Big Data*, 2017.

[40] R. Kabir, S. M. Remias, J. Waddell, and D. Zhu, "Time-series fuel consumption prediction assessing delay impacts on energy using vehicular trajectory," *Transportation Research Part D: Transport and Environment*, vol. 117, p. 103678, 2023.

[41] Y. Yao, X. Zhao, C. Liu, J. Rong, Y. Zhang, Z. Dong, and Y. Su, "Vehicle fuel consumption prediction method based on driving behavior data collected from smartphones," *J. Adv. Transp.*, vol. 2020, p. 9263605, 2020.

[42] S. Kanarachos, J. Mathew, and M. E. Fitzpatrick, "Instantaneous vehicle fuel consumption estimation using smartphones and recurrent neural networks," *Expert Systems with Applications*, vol. 120, pp. 436–447, 2019.

[43] F. C. López and R. Álvarez Fernández, "Predictive model for energy consumption of battery electric vehicle with consideration of self-uncertainty route factors," *J. Cleaner Prod.*, vol. 276, p. 124188, 2020.

[44] C. De Cauwer, J. Van Mierlo, and T. Coosemans, "Energy consumption prediction for electric vehicles based on real-world data," *Energies*, vol. 8, no. 8, pp. 8573–8593, 2015.

[45] J. H. Bundgaard, B. B. T. Madsen, and C. P. Nørskov, "Using spatial and temporal context for predicting energy consumption of electric vehicles," Aalborg Universitet, Tech. Rep., 2019.

[46] P. Zarei, "Data-driven power estimation of electric buses using deep LSTM neural networks and symbolic aggregate approximation filtering," Master's Thesis, University of British Columbia, Dec 2020.

[47] F. Bartoli, G. Lisanti, L. Ballan, and A. Del Bimbo, "Context-aware trajectory prediction," in *Int. Conf. on Pattern Recognition (ICPR)*. IEEE Computer Society, Aug 2018.

[48] L. F. Chiara, P. Coscia, S. Das, S. Calderara, R. Cucchiara, and L. Ballan, "Goal-driven self-attentive recurrent networks for trajectory prediction," in *IEEE/CVF Conf. on Computer Vision and Pattern Recognition Workshops (CVPR W)*, 2022.

[49] M. Lisotto, P. Coscia, and L. Ballan, "Social and scene-aware trajectory prediction in crowded spaces," in *IEEE/CVF Int. Conf. on Computer Vision Workshop (ICCVW)*, 2019.

[50] S. Das, G. Camporese, S. Cheng, and L. Ballan, "Distilling knowledge for short-to-long term trajectory prediction," *preprint arXiv:2305.08553*, 2024.

[51] P. Coscia, F. Castaldo, F. A. Palmieri, A. Alahi, S. Savarese, and L. Ballan, "Long-term path prediction in urban scenarios using circular distributions," *Image and Vision Computing*, vol. 69, pp. 81–91, 2018.

[52] S. Mehri, A. A. Alesheikh, and A. Basiri, "A contextual hybrid model for vessel movement prediction," *IEEE Access*, vol. 9, pp. 45 600–45 613, 2021.

[53] P. Han, M. Zhu, and H. Zhang, "Interaction-aware short-term marine vessel trajectory prediction with deep generative models," *IEEE Trans. Ind. Inf.*, vol. 20, no. 3, pp. 3188–3196, 2024.

[54] G. Spadon *et al.*, "Building a safer maritime environment through multi-path long-term vessel trajectory forecasting," *CoRR*, vol. abs/2310.18948, 2023.

[55] P. Petrou *et al.*, "Argo: A big data framework for online trajectory prediction," in *Proc. 16th Int. Symp. on Spatial and Temporal Databases (SSTD)*. ACM, 2019.

[56] Y. Suo, W. Chen, C. Claramunt, and S. Yang, "A ship trajectory prediction framework based on a recurrent neural network," *Sensors*, vol. 20, no. 18, p. 5133, 2020.

[57] A. Ip, L. Irio, and R. Oliveira, "Vehicle trajectory prediction based on LSTM recurrent neural networks," in *Proc. 2021 IEEE 93rd Vehicular Technology Conf. (VTC)*, 2021.

[58] A. Graser, A. Naghibzadeh-Jalali, J. Lampert, A. Weißenfeld, and K. Janowicz, "Deep Learning from Trajectory Data: A review of deep neural networks and the trajectory data representations to train them," in *Proc. Workshop on Big Mobility Data Analytics (BMDA)*, G. Fletcher and V. Kantere, Eds., vol. 3379, 2023.

[59] F. Rosenblatt, "Principles of neurodynamics. perceptrons and the theory of brain mechanisms," Cornell Aeronautical Lab Inc Buffalo NY, Tech. Rep., 1961.

[60] S. Hochreiter, "The vanishing gradient problem during learning recurrent neural nets and problem solutions," *Int. J. Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 6, no. 2, pp. 107–116, 1998.

[61] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Trans. Neural Netw.*, vol. 5, no. 2, pp. 157–166, 1994.

[62] M. Kuhn and K. Johnson, *Applied Predictive Modeling*. Springer New York, 2013.

[63] C. J. Willmott and K. Matsuura, "Advantages of the Mean Absolute Error (MAE) over the Root Mean Square Error (RMSE) in assessing average model performance," *Climate Research*, vol. 30, no. 1, pp. 79–82, 2005.

[64] M. Hashemi and H. A. Karimi, "A Critical Review of Real-Time Map-Matching Algorithms: Current issues and future directions," *Computers, Environment and Urban Systems*, vol. 48, pp. 153–165, 2014.

[65] A. Graser *et al.*, "MovingPandas: v0.18.1," May 2024.

[66] P. Zippenfenig, "Open-Meteo.com weather API," Jan 2024.

[67] S. Tarquini, I. Isola, M. Favalli, A. Battistini, and G. Dotta, "TINITALY, a digital elevation model of Italy with a 10 meters cell size (Version 1.1)," Istituto Nazionale di Geofisica e Vulcanologia (INGV), 2023.

[68] R. Sasse David, E. Zimányi, K. Torp, and M. Sakr, "Roundabouts and the energy consumption of electrical vehicles," in *Proc. 16th ACM SIGSPATIAL Int. Workshop on Computational Transportation Science (IWCTS)*. ACM, 2024.

[69] ——, "Speed and energy consumption for electrical vehicles," in *Proc. 15th ACM SIGSPATIAL Int. Workshop on Computational Transportation Science (IWCTS)*. ACM, 2022.

[70] Meteorological Service of Canada, *Manual of Surface Weather Observations (MANOBS)*, 7th ed., Meteorological Service of Canada, Canada, Jan 2013.

[71] B. H. Menze, B. M. Kelm, R. Masuch, U. Himmelreich, W. Petrich, and F. A. Hamprecht, "A comparison of Random Forest and its Gini Importance with Standard Chemometric Methods for the feature selection and classification of spectral data," *BMC Bioinformatics*, vol. 10, p. 213, 2009.

[72] K.-Q. Shen, C.-J. Ong, Z. Hui, X.-P. Li, and E. Wilder-Smith, "A feature selection method for multilevel mental fatigue EEG classification," *IEEE Trans. on Biomedical Engineering*, vol. 54, no. 7, pp. 1231–1237, 2007.

[73] V. Lakshmanan, S. Robinson, and M. Munn, *Machine Learning Design Patterns*. O'Reilly Media, 2020.

[74] X. Glorot and Y. Bengio, "Understanding the difficulty of training Deep Feedforward Neural Networks," in *Proc. 13th Int. Conf. on Artificial Intelligence and Statistics (AISTATS)*, 2010.

[75] A. L. Maas, A. Y. Hannun, A. Y. Ng *et al.*, "Rectifier nonlinearities improve neural network acoustic models," in *Proc. International Conference on Machine Learning (ICML)*, vol. 30, no. 1, 2013.

[76] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. 3rd Int. Conf. on Learning Representations (ICLR)*, 2015.

[77] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *J. of Machine Learning Research*, vol. 12, no. 7, 2011.

[78] S. Ruder, "An overview of gradient descent optimization algorithms," *preprint arXiv:1609.04747*, 2016.

[79] S. Guo, J. Mou, L. Chen, and P. Chen, "An anomaly detection method for AIS trajectory based on kinematic interpolation," *J. of Marine Science and Engineering*, vol. 9, no. 6, 2021.

[80] M. Musleh, "Towards a unified deep model for trajectory analysis," in *Proc. 30th ACM SIGSPATIAL Int. Conf. on Advances in Geographic Information Systems*. ACM, 2022.

Acknowledgments

These past two years studying in the BDMA program have been very hard and full of struggle, especially while working on this thesis. Without the help of others, I would not have survived this master's program.

I would like to extend my deepest gratitude to my supervisors, Prof. Lamberto Ballan and Prof. Esteban Zimanyi. Prof. Ballan, who graciously agreed to supervise my thesis, provided invaluable guidance throughout the entire process. I am also immensely grateful to Prof. Esteban Zimanyi, who involved me in his lab during the thesis work and encouraged me to discuss my ideas with other lab members.

I would also like to express my sincere thanks to Prof. Mahmoud Sakr, whose advice was instrumental in guiding my thesis experiments.

I want to thank the lab members with whom I collaborated this semester. Their input and guidance made a significant difference, and without their support, completing this thesis would have been much harder.

Finally, I want to thank my best friend Luis, who has helped me get through this thesis and motivated me during this difficult process. Thank you very much.

A

Appendices

A.1 Appendix A

Attribute	Datatype	Description
Timestamp	Timestamp	Timestamp from the AIS basestation
Type of mobile	String	Describes the type of target
MMSI	String	MMSI number of the vessel
Latitude	Float	Latitude of message report
Longitude	Float	Longitude of message report
Navigational status	String	Navigational status from AIS message
ROT	Float	Rate of Turn from AIS message
SOG	Float	Speed over ground from AIS message
COG	Float	Course over ground from AIS message
Heading	Float	Heading from AIS message
IMO	String	IMO number of the vessel
Callsign	String	Callsign of the vessel
Name	String	Name of the vessel
Ship type	String	Describes the AIS ship type of this vessel
Cargo type	String	Type of cargo from the AIS message
Width	Float	Width of the vessel
Length	Float	Length of the vessel
Type of position fixing device	String	Type of positional fixing device from the AIS message
Draught	Float	Draught field from AIS message
Destination	String	Destination from AIS message
ETA	Timestamp	Estimated Time of Arrival, if available
Data source type	String	Data source type
Size A	Float	Length from GPS to the bow
Size B	Float	Length from GPS to the stern
Size C	Float	Length from GPS to starboard side
Size D	Float	Length from GPS to port side

Table A.1: AIS data structure

A.2 Appendix B

All codes and implementation related to this project are available at the following GitHub repository:

`https://github.com/satriabw/master-thesis`