Czech Technical University in Prague

Faculty of Mechanical Engineering – Department of Instrumentation and Control Engineering

Diploma Thesis

City traffic time series forecasting using RNN LSTM

Bc. Martin Křeček

Supervisor: Ing. Matouš Cejnek Ph.D

Study programme: Automation and Instrumentation Engineering

Prague 2023

Statement

I declare that I have worked on this thesis independently assuming that the results of thesis can be also used at discretion of the supervisor of the thesis as its co-author. I also agree with the potential of publication of the results of the thesis or its substantial part, provided I will be listed as the co-author.

In Prague ....

Bc. Martin Křeček

Acknowledgements Thanks to the thesis supervisor Ing. Matouš Cejnek, Ph.D. for his patience, invaluable advice and professional help in the preparation of the thesis.

Bc. Martin Křeček

MASTER'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Křeček Martin** Personal ID number: **482608**

Faculty / Institute: **Faculty of Mechanical Engineering**

Department / Institute: **Department of Instrumentation and Control Engineering**

Study program: **Automation and Instrumentation Engineering**

Specialisation: **Automation and Industrial Informatics**

II. Master's thesis details

Master's thesis title in English:

City traffic time series forecasting using RNN LSTM

Master's thesis title in Czech:

Předpovídání časových řad v městské dopravě pomocí RNN LSTM

Guidelines:

1. Automatic integration of traffic and weather data
2. Creation of a predictive model
3. Prediction of traffic situation based on historical data and weather forecast

Bibliography / sources:

[1] P., B.G.E. (2016) Time Series Analysis: Forecasting and Control. Hoboken N.J.: Wiley.
[2] Peixeiro, M. (2022) Time series forecasting in Python. Manning Publications Co. LLC.
[3] Montgomery, D.C., Jennings, C.L. and Kulahci, M. (2008) Introduction to time series analysis and forecasting. Hoboken, NJ: Wiley-Interscience.
[4] Harenslak, B. and Ruiter, J.de (2021) Data Pipelines with Apache airflow. Shelter Island, NY: Manning Publications Co.

Name and workplace of master's thesis supervisor:

Ing. Matouš Cejnek, Ph.D. U12110.3

Name and workplace of second master's thesis supervisor or consultant:

Date of master's thesis assignment: **28.04.2023** Deadline for master's thesis submission: **08.06.2023**

Assignment valid until: _____

Ing. Matouš Cejnek, Ph.D.	Head of department's signature	doc. Ing. Miroslav Španiel, CSc.
Supervisor's signature		Dean's signature

III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

Date of assignment receipt _____ Student's signature _____

Annotation sheet

Thesis title:	City traffic time series forecasting using RNN LSTM
Author:	Bc. Martin Křeček
Year:	2023
Study Program:	Automation and Instrumentation Engineering
Department:	Department of Instrumentation and Control Engineering
Supervisor:	Ing. Matouš Cejnek Ph.D
Bibliographic data:	Number of pages: 87 Number of figures: 23 Number of appendices: 2

Abstract:

This diploma thesis focuses on the development of a comprehensive forecasting system for urban mobility, with a specific emphasis on parking occupancy and public transport delays. Robust data collection and preprocessing pipelines are established to incorporate data from diverse sources. State-of-the-art machine learning techniques, particularly Long Short-Term Memory (LSTM) models, are implemented to forecast urban mobility variables. External factors such as weather data are integrated to enhance forecasting accuracy. A user-friendly web application is developed for data exploration and analysis. The research highlights the potential of data-driven forecasting systems in optimizing transportation efficiency and contributes to the field of intelligent transportation systems.

Keywords:	City traffic, Time series, Open data, LSTM, Forecasting, Machine learning, Neural networks

Contents

1 Introduction

The urban landscape is constantly evolving and cities face increasing challenges in managing their transport systems. As urban populations grow, the demand for efficient, reliable and sustainable mobility solutions becomes paramount. In this context, data-driven forecasting systems have emerged as a powerful tool to address these challenges and optimize urban transport.

The aim of this thesis is to develop a comprehensive predictive system for urban mobility, focusing on key aspects such as urban traffic, parking occupancy and tram delays. By accurately predicting these variables, I can provide valuable insights and support decision-making processes for commuters, urban planners, transportation authorities, and others.

Advances in machine learning techniques have revolutionized the field of urban mobility forecasting. By leveraging diverse and large datasets, I can capture complex patterns and correlations within urban transport systems. This allows us to develop predictive models that can produce accurate forecasts and improve the overall efficiency of urban mobility.

The main goals of this thesis are to develop robust pipelines for data collection and preprocessing: I will collect data from a variety of sources, including traffic sensors, parking sensors, and tram monitoring systems. This data will undergo thorough preprocessing to ensure its quality and suitability for training predictive models.

Furthermore, the implementation of state-of-the-art machine learning models: LSTM-based models, known for their ability to capture time dependencies, will be used to predict urban traffic, parking occupancy and tram delays. These models will be trained and evaluated on the collected data to optimize their performance.

Integration of external factors: I will incorporate external factors such as weather data to increase the accuracy of the prediction and capture other influences on urban mobility variables. By incorporating these factors, our models can provide more comprehensive and reliable forecasts.

Finally, Develop a user-friendly web application: a web application will be developed to facilitate the exploration and analysis of forecast data. This application will provide an intuitive interface for users to interact with forecasts, visualize raw data, and make informed decisions based on the insights provided.

Throughout this thesis I will explore the complex relationships between different urban mobility variables and demonstrate the potential of data-driven forecasting systems in improving transport efficiency. The developed forecasting system will enable commuters to plan their trips more efficiently, assist urban planners in making informed decisions, and contribute to the overall goal of creating sustainable and optimized urban transportation networks.

At the end of this thesis, I want to provide the public with a comprehensive solution that addresses multiple aspects of urban mobility forecasting. I hope that this research will contribute to the ongoing efforts to develop intelligent transportation systems and pave the way for future innovations in urban mobility.

Theoretical part

2 Related work

This chapter provides a comprehensive overview of existing research and studies that are relevant to the topic of this diploma. In this chapter, we explore and analyze previous works conducted by researchers and experts in the field, shedding light on their methodologies, findings, and contributions. By examining these related works, we aim to establish a strong foundation for our research, identify gaps in the existing knowledge, and build upon the advancements made by previous researchers. Through this exploration of related work, we strive to gain valuable insights, understand the current state of the field, and position our research within the broader academic context. By critically reviewing and synthesizing prior studies, we can effectively contribute to the existing body of knowledge and advance the field further.

The paper "Traffic Flow Forecasting using Multivariate Time-Series Deep Learning and Distributed Computing" [1] by Ngoc-Phap Trinh, Anh-Khoa N. Tran, and Trong-Hop Do addresses the problem of traffic flow prediction. The paper proposes several univariate and multivariate time series models, including LSTM, TCN, Seq2Seq, NBeats, ARIMA, and Prophet, using distributed deep learning to deal with the traffic flow prediction problem. The models are implemented and evaluated on a dataset of traffic flows in Ireland. The proposed multivariate models take the combination of traffic flow data, weather in the local area, and graph data of connections between traffic positions to produce the prediction of the traffic flow.

This paper highlights that the proposed multivariate deep learning models achieved better prediction accuracy compared to the univariate models and machine learning models. The authors conducted several experiments to examine the performances of these models in different scenarios to help understand more about the performance of these models. The paper also discusses the contributions of the study, which include building a dataset that combines traffic data, weather, and graph data showing the relationship between traffic locations and routes. The authors proposed multivariate deep learning time series models to predict traffic flow and conducted several experiments with various deep learning and

machine learning univariate and multivariable time series models and compared their performances. The proposed multivariate deep learning model that utilizes weather and graph data achieved the best performance. The paper explains that the most representative data-driven approach is the neural network and deep learning, which can automatically extract the relevant high-level features of traffic flow data.

The next article "Short-Term Traffic Flow Prediction with Weather Conditions: Based on Deep Learning Algorithms and Data Fusion" [2] by Yue Hou, Zhiyuan Deng, and Hanke Cui discusses the challenges of short-term traffic flow prediction and proposes a novel combined framework of stacked autoencoder (SAE) and radial basis function (RBF) neural network to predict traffic flow. The authors argue that traffic flow data with temporal features and periodic characteristics are vulnerable to weather effects, making short-term traffic flow prediction a challenging issue. The existing models do not consider the influence of weather changes on traffic flow, leading to poor performance under some extreme conditions. Therefore, it is necessary to conduct research studies on traffic flow prediction driven by both traffic data and weather data.

The proposed model incorporates SAE and RBF to capture the features of traffic flow and weather conditions. SAE is used to learn the temporal correlation in traffic flow, RBF to learn the periodic evolution under weather disturbance, and another RBF to realize the decision-level data fusion of the former models. This combined framework can effectively learn the periodicity and temporal correlation of traffic flow and the disturbance of weather conditions to improve the accuracy and robustness of the prediction model. The authors also discuss the limitations of existing methods, such as single models having limitations in processing complex data. To integrate the advantages of single models to achieve more accurate traffic flow forecasting, a variety of combined models have emerged.

In summary, the article proposes a novel combined framework of SAE and RBF to predict traffic flow, driven by both traffic data and weather data. The authors argue that this approach can effectively learn the periodicity and temporal correlation of traffic flow and the disturbance of weather conditions to improve the accuracy and robustness of the prediction model. The article also discusses the limitations of existing methods and the different categories of short-term traffic flow prediction models, including statistical models, traditional machine learning models, and deep learning models.

The article titled "Neural Network-based Model for Traffic Prediction in the City of Valencia" [3] discusses the development of a model that can predict traffic flow in the city of Valencia, Spain, based on data collected by electromagnetic loops

distributed throughout the city. The aim of the study is to design a model that can predict traffic flow in different streets of Valencia at different hours of the day, which can be useful for administration authorities and researchers attempting to improve the living conditions of citizens. With a good traffic prediction, it will be possible to foresee possible traffic jams and trigger countermeasures to mitigate them.

The study uses two models based on two recurrent neural networks of Long Short-Term Memory (LSTM) type to predict traffic flow. The authors also study the influence of the specific characteristics used on the accuracy of the model. The results of the experiments show that, despite the high heterogeneity in terms of per-street traffic behavior, it is possible to reach useful prediction models with low errors.

The study is significant because it provides a model that can predict traffic flow in the city of Valencia, which can be useful for administration authorities and researchers attempting to improve the living conditions of citizens. The model uses electromagnetic loops distributed throughout the city to collect data, which is then used to predict traffic flow. The use of LSTM neural networks is also significant because they are capable of processing sequential data, which is essential for predicting traffic flow.

The study also highlights the importance of accurate traffic prediction in mitigating traffic jams and improving the living conditions of citizens. The authors suggest that the model can be used to trigger countermeasures to mitigate traffic jams, which can improve the flow of traffic and reduce the time spent by citizens in traffic.

In conclusion, the study provides a useful model for predicting traffic flow in the city of Valencia, which can be useful for administration authorities and researchers attempting to improve the living conditions of citizens. The use of LSTM neural networks and electromagnetic loops distributed throughout the city to collect data is significant because it allows for the processing of sequential data, which is essential for predicting traffic flow. The study also highlights the importance of accurate traffic prediction in mitigating traffic jams and improving the living conditions of citizens.

Another document called "City traffic prediction based on real-time traffic information for Intelligent Transport Systems" [4] discusses the use of Intelligent Transportation Systems (ITS) technology to mitigate traffic congestion on city roads. The paper highlights the importance of accurate traffic prediction in the efficient operation of ITS. The authors propose two distinct modeling approaches for predicting traffic volume on city roads.

The first model is based on the propagation of traffic flow along successive road links on a route, while the second model is based on time-varied spare flow capacity on the concerned links. The validity of the models is ensured by incorporating real-time traffic information, which reflects the up-to-date traffic situation in the network. Hence, the proposed models are adaptive to the dynamic change of traffic flow pattern.

The paper notes that the features of a city traffic network include short road links and a large number of links, making the prediction of traffic situations on all links potentially computationally costly. Additionally, traffic flow is frequently split up due to the prevalent existence of intersections, and traffic management at intersections, such as traffic signal control, has a strong impact on traffic flow patterns. The occurrence of traffic congestion is more accurately indicated by traffic volume on road links rather than travel speed.

The authors highlight that most of the existing highway traffic prediction models cannot be effectively applied to city roads due to the intrinsic differences between a city traffic network and a highway system. Therefore, the proposed models seek to predict traffic volume on city roads by adopting two distinct modeling approaches.

The proposed models are implemented to predict the traffic volume in Cologne, Germany, and the real data are collected through simulations in the traffic simulator SUMO. The results show that both of the proposed models reduce the prediction error up to 52% and 30% in the best cases compared to the existing Shift Model. In addition, the authors found that Model-1 is suitable for short prediction intervals that are in the same magnitude as the link travel time, while Model-2 demonstrates superiority when the prediction interval is larger than one minute.

3 Theoretical part

3.1 Data

Data is a fundamental building block for any predictive model, and it plays a critical role in the success of machine learning algorithms [5]. Without high-quality data, machine learning models cannot accurately forecast future outcomes. The quality and quantity of data that is fed into a machine learning model directly impact the accuracy of the model's predictions.

Data for forecasting problems should be comprehensive, reliable, and timely [6]. Historical data is often used to train machine learning models to make predictions about future events [7]. The data should include relevant variables that impact the outcome of interest, such as weather patterns, traffic volume, and road conditions in the case of city traffic forecasting.

It is also important to ensure that the data is properly cleaned and pre-processed to remove any inconsistencies, errors, or missing values [8]. The accuracy of the model's predictions will be compromised if the data contains inaccuracies or inconsistencies. Furthermore, the size of the data set is also an important factor to consider. The more data that is available, the more accurate the model's predictions are likely to be. However, having a large amount of data also requires powerful computing resources, which can pose a challenge for some applications.

In summary, data is the backbone of any predictive model, and its quality and quantity directly impact the accuracy of the model's predictions [5]. It is therefore essential to ensure that the data is, as already mentioned, comprehensive, reliable, timely and therefore also properly cleaned and pre-processed.

For the city traffic forecasting project, the data that will be used is obtained from the Prague Public API [9]. This API provides access to a wide range of data, including city traffic information, availability of parking spaces, and other relevant variables for city traffic forecasting. The API also provides historical data, which can be used to train the forecasting models accurately.

In addition to the traffic-related data, weather data will also be included in the analysis. Weather patterns are a significant variable that can impact traffic volumes and road conditions. Therefore, it is vital to consider this variable when forecasting city traffic patterns accurately. Weather data will be extracted from the publicly available API provided by Open Meteo [10] , which provides access to global weather data with hourly resolution. By including this data in the analysis, the model can account for the impact of weather on traffic patterns and provide more accurate predictions.

The data for the city traffic forecasting project will be obtained from multiple sources, including the Prague Public API and the Open Meteo API. The Prague Public API provides reliable and up-to-date traffic-related data, while the Open Meteo API provides weather data. By combining these two sources of data, it is possible to create accurate forecasting models that can help city officials make informed decisions to manage traffic more efficiently. The use of this comprehensive and reliable data will enable the creation of accurate forecasting models that can help optimize traffic management in the city.

3.1.1 Golemio API

The Golemio API is part of the larger Golemio platform [9], which is dedicated to providing open and accessible data for the city of Prague. Golemio is a civic tech project that seeks to make public data more easily accessible and understandable to citizens, businesses, and organizations. The Golemio platform includes a variety of datasets, such as transportation data, environmental data, and cultural data.

It is a key component of the Golemio platform, providing real-time data on various aspects of city traffic. This data can be used to analyze traffic patterns, identify areas of congestion, and make predictions about future traffic flow. By making this data available to the public, the Golemio platform promotes transparency and encourages citizen involvement in the management of urban infrastructure.

In this diploma, I will be using the Golemio API to download traffic data about Prague. Specifically, the parking data and information on public transport to develop a predictive model for city traffic. The Golemio platform's commitment to open and accessible data makes it an ideal resource for this type of research. By leveraging the power of the Golemio API, we can gain insights into the complex nature of city traffic and work towards developing solutions for more efficient and sustainable transportation systems.

3.1.1.1 Parking

I will be utilizing parking data to predict city traffic patterns using machine learning techniques. Parking data provides information on the availability of parking spaces in P+R facilities, which are large parking lots located on the outskirts of the city designed to encourage commuters to use public transport for the remainder of their journey. The availability of parking spaces in P+R facilities can provide valuable insights into the level of demand for public transport and suggest the potential for increased city traffic and congestion.

By analyzing and applying parking data, we can create a predictive model for city traffic forecasting that considers the availability of parking spaces in different areas of the city. This model can provide valuable insights into traffic flow patterns and help city planners and transportation agencies better manage traffic and reduce congestion. The analysis of parking data is a crucial step in developing an accurate predictive model for city traffic, and this data can be a valuable source of information for addressing traffic-related issues in urban areas.

3.1.1.2 Vehicle positions

Public transportation data is another key entity that I will be leveraging in our effort to predict city traffic patterns. This data includes real-time information on the positions and delays of buses, trams and trains, as well as information on routes and the sequence of vehicles. By analyzing this data, we can gain valuable insights into how public transportation affects traffic flow and congestion in the city.

By implementing machine learning techniques, we can analyze public transportation data to identify patterns and trends that can be used to make predictions about future traffic flow. For example, we can use historical data to identify which routes experience the most congestion at specific times of day and use this information to predict when and where traffic jams are likely to occur. We can also use this data to identify areas of the city where public transportation is underutilized and develop strategies to encourage more people to use public transportation, which can help to reduce traffic congestion.

Overall, by applying public transportation data in conjunction with other sources of data, such as parking data, we can develop more accurate predictive models for city traffic patterns. This research has the potential to make a significant impact on the management of traffic flow and congestion in the city, ultimately leading to more efficient and sustainable transportation systems.

3.1.1.3 General Transit Feed Specification

Another valuable source of data that I will be utilizing in our effort to predict city traffic patterns is GTFS data. GTFS, or General Transit Feed Specification, data provides information on the routes of vehicles, including where they will ride, the services offered, the stops of the vehicles, and which vehicles will be operating on specific trips.

By analyzing this data, we can gain further insights into how public transportation affects traffic flow and congestion in the city. For example, we can use GTFS data to identify the most popular routes and stops, which can help us predict where congestion is likely to occur. We can also use this data to identify areas of the city where public transportation is underutilized and develop strategies to encourage more people to use public transportation, which can help to reduce traffic congestion.

3.1.2 Weather API

In addition to parking data, public transportation data and GTFS data, I will also be utilizing weather data to enhance the accuracy of our predictive models for city traffic patterns. Weather conditions, such as rain, snow, and temperature, can have a significant impact on traffic flow and congestion [11]. By incorporating weather data into our models, we can account for these external factors and make more accurate predictions about future traffic density.

For example, weather can have a significant impact on public transportation, as heavy rain or snow can lead to delays or cancellations of buses and trams. This can result in overcrowding and longer wait times for passengers, which can ultimately lead to increased congestion and traffic on the roads. By analyzing historical weather data in conjunction with public transportation data, we can identify correlations and trends that can be used to make predictions about how future weather conditions are likely to impact public transportation and subsequently affect traffic flow in the city.

I will be using a public weather API to download weather data for the city of Prague, which will include information such as temperature, precipitation, and snowfall. By utilizing weather data into our predictive models, we can develop a more comprehensive understanding of the factors that contribute to traffic flow and congestion in the city, and develop more effective strategies for managing traffic and reducing congestion.

3.2 Python

Python is an essential component in this diploma, as it will be used for virtually every aspect of the project. Python is a popular programming language [12] that has become the standard in the data science community due to its simplicity, readability, and versatility. I will use Python for extracting data from public APIs, building data pipelines using Apache Airflow, and creating and training predictive models.

Python will be used to build downloaders for extracting data from API endpoints, which will then be used to develop the predictive model. Additionally, I will use Python to build an Airflow workflow for automating the data collection and model training process. Finally, I will use Python for the creation and training of the predictive model itself.

Python is an essential tool for project like this, as it provides a flexible and powerful platform for developing and implementing predictive models. By leveraging the power of Python, we can create an efficient and scalable workflow for data processing and machine learning, ultimately leading to more accurate and effective predictive models for city traffic patterns.

3.3 Google Cloud Platform

Google Cloud Platform (GCP) [13] is a comprehensive suite of cloud computing services designed to help businesses and organizations manage their data and workloads in a secure and scalable environment. In this project, I will be utilizing several components of GCP to help us collect, process, and analyze data, build predictive models, and monitor our progress.

One of the most critical components of GCP that I will be using is the virtual machine (VM). Specifically, I will be leveraging a Debian VM instance to host data processing workflows and run predictive models. The Python-based extractors that I have written will run on this VM instance, and there will be a MySQL database that will store the extracted data. Additionally, I will be implementing Apache Airflow to build and manage our data pipelines, which will be hosted on the VM instance.

Moreover, I will take advantage of GCP's monitoring features to keep track of resource utilization, such as CPU usage and disk usage, to ensure that our workflows are running smoothly and to optimize performance as needed. These monitoring features will enable me to proactively identify and address any issues that may arise, allowing me to maintain the efficiency and accuracy of my data processing and modeling workflows.

By utilizing GCP's powerful tools and features in conjunction with Python, I can leverage the full potential of cloud computing to build accurate and effective predictive models for traffic patterns, with the ultimate goal of improving traffic management and reducing congestion in the city. There were several reasons why I chose Google Cloud Platform (GCP) over other options. One of the main factors was the availability of a free trial, which allowed me to test and evaluate the platform's services and capabilities without any financial commitment. Additionally, GCP's reputation for reliability, scalability, and a wide range of advanced features played a significant role in my decision. The extensive documentation, strong community support, and seamless integration with other Google services were also compelling factors that contributed to my choice of GCP for my project.

3.4 Apache Airflow

Apache Airflow [14] is an open-source platform designed to help users schedule, and monitor workflows. In this project, I will be implementing Airflow to manage data extracting workflows, data processing workflows and machine learning models.

With Airflow, I can easily create and manage complex workflows, including tasks that depend on the output of other tasks. This makes it easy to manage the various data processing tasks involved in our project, such as the extraction of data from public APIs, the transformation of data using SQL scripts, and the training and prediction of machine learning models.

Furthermore, Airflow provides a user-friendly web interface that allows us to easily monitor the status of our workflows and quickly identify and address any issues that may arise. With its powerful features and flexibility, Airflow provides an ideal platform for managing the complex workflows involved in our city traffic prediction project.

I chose Apache Airflow for my project due to its powerful workflow management capabilities and extensive library of pre-built operators. Its scalable and flexible architecture, along with its rich ecosystem and active community, made it the ideal choice for orchestrating complex data workflows and automating data pipelines.

3.4.1 DAG

In graph theory, a Directed Acyclic Graph (DAG) can be mathematically defined as a finite directed graph G = (V, E), where V is a set of vertices or nodes, and E is a set of directed edges. The edges in the DAG represent the

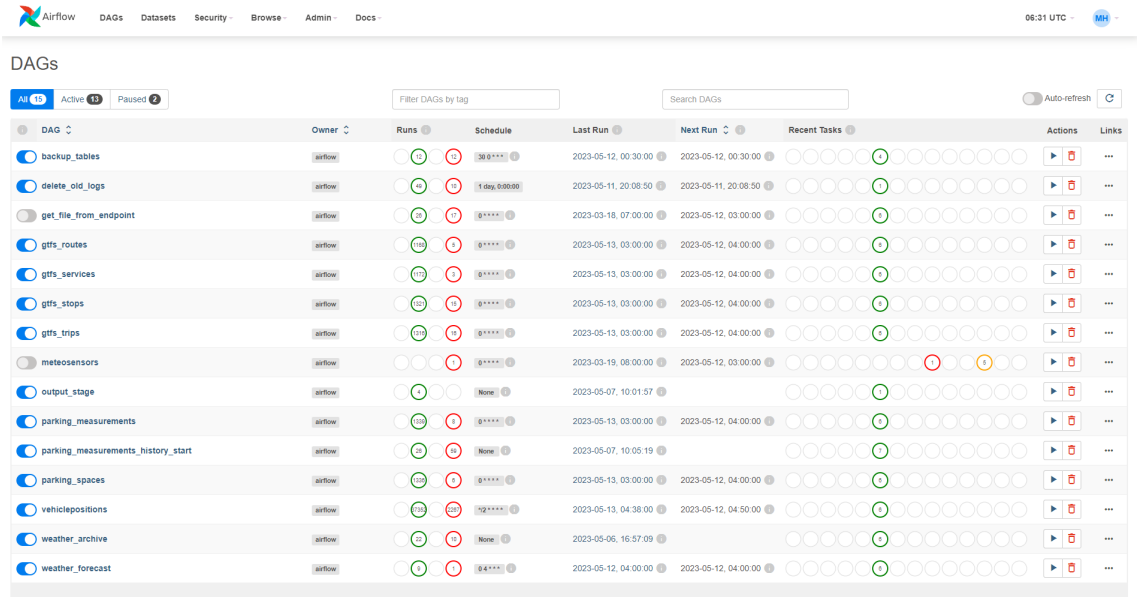

Figure 1: Airflow User Interface

directed connections between the nodes [15]. Formally, a DAG satisfies the following properties:

Acyclicity: There are no directed cycles in the graph. This means that it is not possible to traverse through a sequence of directed edges and return to the starting node. Directionality: The edges have a direction associated with them, indicating the flow or dependency between the nodes. The directed edges can only be traversed in the specified direction. For any pair of nodes u and v in the DAG, there exists a directed path from u to v if and only if there is a directed edge connecting u to v. This property ensures that the graph remains acyclic and allows for the modeling of dependencies and relationships between different elements.

DAG in Airflow represents a collection of tasks and their dependencies, allowing users to define and visualize the workflow logic in a structured and intuitive manner. Each task within a DAG represents a distinct unit of work, such as executing a script, running a SQL query, or transferring data between systems. By defining the dependencies between tasks, Airflow enables the automation and execution of workflows with a high level of control and reliability.

3.4.2 UI

The Airflow user interface (UI), which can be seen in Figure 1, provides a user-friendly web interface. interface for interacting with and monitoring workflows. The UI allows users to view and manage DAGs, track the status and progress of individual tasks, and access logs and task history. With its intuitive design and

real-time updates, the UI facilitates easy navigation and management of workflows, empowering users to monitor and troubleshoot their workflows effectively.

3.4.3 Scheduler

The Scheduler is a critical component of Apache Airflow that is responsible for triggering and executing tasks based on the defined schedule and dependencies. The Scheduler continuously scans the DAGs and their associated tasks, determining which tasks need to be executed based on their dependencies and the schedule configuration. It ensures that tasks are executed in the correct order and according to the specified schedule, orchestrating the flow of data and processing within the workflow. The Scheduler plays a pivotal role in automating the execution of tasks, enabling the timely and efficient completion of complex workflows.

3.5 SQL

Structured Query Language (SQL) is a standard language for managing and manipulating data in a relational database management system (RDBMS) [16]. SQL provides a set of commands for creating, modifying, and querying data in a relational database. SQL is used to define the structure of a database, manipulate data within the database, and control access to the data [17].

SQL is a critical tool for data manipulation and management in the context of machine learning and predictive modeling. In this project, I will use SQL to transform data obtained from the Prague Public API into a format that is suitable for use in predictive models. Each entity will have its own SQL script, which will convert the data from the JSON format returned by the API into a relational database schema with appropriate columns and rows.

SQL is also used to define and create the database schema using Data Definition Language (DDL) statements. DDL statements are used to define the tables, columns, data types, constraints, and other database objects in a structured and organized manner [18]. This ensures that the data is organized, consistent, and easily manageable. Additionally, the data dictionary is also maintained by SQL, which provides a unified view of the database schema. The data dictionary helps in maintaining the consistency and accuracy of the data, which is crucial for the success of the predictive models.

In summary, SQL is a powerful tool for managing and manipulating data in a relational database management system. In my diploma, SQL will be used to

transform data from the Prague Public API into a format that is suitable for use in predictive models and ensuring the accuracy and consistency of the data over time.

3.5.1 MySQL

MySQL is a popular open-source relational database management system used for managing large amounts of structured data. In this diploma, MySQL will be used to store and manage the data obtained from the Prague Public API, including traffic-related variables, parking space availability, and weather data. MySQL is an efficient and reliable database management system that can handle large amounts of data and ensure data consistency [19].

One of the primary advantages of using MySQL is its flexibility in handling data from different sources. As mentioned earlier, the data obtained from the Prague Public API is in JSON format. To make this data compatible with the relational database schema, we will use SQL scripts to transform the data into columns and rows. This allows us to create a structured and organized database that is easy to manage and query.

MySQL is also scalable, making it suitable for projects that involve large amounts of data. As we collect more data over time, MySQL can handle the increase in data size without affecting performance. Additionally, MySQL provides various security features to ensure that the data is protected from unauthorized access and manipulation [20].

In summary, MySQL is a powerful database management system that is well-suited for handling large amounts of structured data, such as the data used in the city traffic forecasting project. With MySQL, we can easily store, manage, and query the data obtained from the Prague Public API, and use SQL scripts to transform the data into a relational database schema. The flexibility, scalability, and security features of MySQL make it an excellent choice for this project.

I chose MySQL as the database management system for my project because of its reliability, robustness, and widespread adoption in the industry. MySQL offers excellent performance, scalability, and support for large datasets, making it suitable for handling diverse data requirements. Additionally, its ease of use, extensive documentation, and compatibility with various programming languages made it a convenient choice for my project's database needs.

3.6 Predictive models

Predictive modeling has become an essential tool for organizations seeking to make data-driven decisions. As Hastie et al. [5] describe, predictive models are mathematical representations of real-world phenomena that enable us to forecast future events or behavior based on past observations. These models can take various forms, including classical statistical models, modern machine learning models, and neural networks. Predictive modeling involves using historical data to create a model that can then be used to make predictions about future outcomes. This can help businesses and organizations to identify patterns and trends, make informed decisions, and optimize their operations.

Classical statistical models, such as autoregressive (AR), moving average (MA), autoregressive moving average (ARMA), and autoregressive integrated moving average (ARIMA), have been widely used for time series forecasting [21]. These models rely on a set of assumptions about the underlying process and use statistical methods to estimate the model parameters. While classical models are often simple and interpretable, they may not capture the full complexity of the data.

Modern machine learning models, such as random forest, support vector machines, and gradient boosting machines, are also commonly used for prediction [22]. These models are designed to automatically learn complex relationships between the input features and the target variable using a large amount of data. Machine learning models can capture more complex patterns in the data than classical models, but they may be more difficult to interpret.

Neural networks, such as recurrent neural networks (RNNs) and long short-term memory (LSTM) networks, are a type of machine learning model that are particularly well-suited for time series forecasting [23]. These models are designed to learn sequential patterns in the data and can capture long-term dependencies. Neural networks are highly flexible and can learn complex relationships between the input features and the target variable. However, they can be computationally expensive and may require a large amount of data to train effectively.

In summary, different types of models have their own strengths and weaknesses, and the choice of model depends on the problem at hand and the available data. In this city traffic forecasting project, the focus will be on using neural network models, particularly RNN-LSTM, to forecast traffic patterns over time.

3.6.1 Classical statistical

Classical statistical models are a popular category of predictive models that have been widely used for time series forecasting. These models are based on a set of assumptions about the underlying process and use statistical methods to estimate the model parameters. The autoregressive (AR), moving average (MA), autoregressive moving average (ARMA), and autoregressive integrated moving average (ARIMA) are some examples of classical statistical models [21]. Classical models are often simple and interpretable, making them popular for forecasting tasks in various fields. However, they may not capture the full complexity of the data and require a careful selection of the model parameters. Despite their limitations, classical statistical models continue to be used in practice, either on their own or as part of more complex models [24].

3.6.1.1 AR

Autoregressive (AR) models are another type of classical statistical model that has been widely used for time series forecasting. These models assume that the current value of a time series is a linear combination of its past values, with an error term that represents the deviation from the predicted value [21]. The order of the AR model specifies how many past values to include in the linear combination. Estimation of the AR model parameters can be done using methods such as the maximum likelihood estimation or the method of moments [24]. AR models are simple and interpretable, making them suitable for forecasting tasks in various domains. However, they are limited in their ability to capture more complex patterns in the data, such as seasonality or long-term trends.

3.6.1.2 MA

Moving average (MA) models are a type of classical statistical models that are commonly used for time series analysis and forecasting. These models assume that the value of a variable at a given time is a linear combination of the errors at that time and the preceding time periods [25]. The MA model is based on the moving average operator, which is a weighted average of the past error terms. The order of an MA model is determined by the number of past error terms that are included in the model. MA models are useful for modeling and forecasting time series data that exhibit a constant level but with random fluctuations around that level [24]. Although MA models have been largely replaced by more complex models

such as ARMA and ARIMA, they continue to be used as building blocks for more sophisticated models [21].

3.6.1.3 ARMA

Autoregressive moving average (ARMA) models are another class of classical statistical models that have been widely used for time series analysis and forecasting. ARMA models combine the autoregressive (AR) and moving average (MA) models, with the aim of capturing both the auto-correlation and the moving average patterns in the data [21]. These models rely on a set of assumptions about the underlying process, such as stationarity, and use statistical methods to estimate the model parameters. ARMA models have been successfully applied in various fields, including finance, economics, and engineering, to name a few [25]. However, like other classical models, ARMA models have their limitations, including the assumption of stationarity and the need for a careful selection of the model parameters. Despite their limitations, ARMA models continue to be widely used in practice, either on their own or as part of more complex models [24].

3.6.1.4 ARIMA

ARIMA (Autoregressive Integrated Moving Average) models are a type of classical statistical model widely used in time series analysis and forecasting. ARIMA models are particularly useful for non-stationary time series, where the statistical properties of the series change over time. The model consists of autoregressive (AR) and moving average (MA) terms, as well as a differencing operation that is used to remove the trend and make the series stationary. The order of the AR and MA terms, as well as the order of differencing, are determined based on the properties of the time series. ARIMA models have been used in various fields, including finance, economics, and meteorology, to name a few [21]. Despite being a classical statistical model, ARIMA has been found to outperform more advanced models in some cases, particularly in situations where the time series exhibits a high degree of autocorrelation, non-linearity, or seasonality [26]. However, the effectiveness of ARIMA compared to other models depends on the specific characteristics of the time series and the forecasting problem at hand.

3.6.2 Machine Learning

Machine learning models have gained popularity in recent years for their ability to handle complex time series data and make accurate predictions. These models are

trained on historical data to learn the patterns and relationships between the input features and the target variable. They can capture more complex patterns in the data than classical statistical models, but may be more difficult to interpret. Random forest, support vector machines, and gradient boosting machines are some examples of machine learning models that are commonly used for time series forecasting [5, 27]. However, the choice of model depends on the specific problem at hand and the available data. In practice, machine learning models are often combined with classical statistical models to improve forecasting accuracy and interpretability [24].

3.6.2.1 Decision trees

Decision trees are a type of machine learning model used for prediction and classification tasks. They are constructed by recursively partitioning the feature space into smaller regions based on a set of binary decisions about the input features. The resulting tree structure can be used to make predictions for new data points by following a path down the tree to a leaf node and returning the predicted value associated with that node. Decision trees are often used in fields such as finance, healthcare, and marketing due to their interpretability and ability to handle complex data structures. Various methods have been developed to improve the performance and interpretability of decision trees, including pruning techniques, ensemble methods, and tree-based models like random forests [28]. Despite their effectiveness, decision trees can suffer from overfitting and instability in the presence of noise and outliers, and care must be taken in their construction and validation [29, 28].

3.6.2.2 Random forests

Random forest is a popular ensemble learning technique used in machine learning for classification and regression tasks. It combines multiple decision trees by aggregating their predictions to improve the accuracy and reduce overfitting. The method was first introduced by Leo Breiman in 2001 [30]. In random forest, each decision tree is built on a randomly selected subset of the training data and a randomly selected subset of the features. This introduces randomness and diversity in the trees, making them less correlated and therefore more effective in reducing overfitting. The final prediction is obtained by averaging or taking a majority vote of the predictions from individual trees. Random forest has been shown to perform well in a wide range of applications, including healthcare, finance, and marketing [31].

3.6.2.3 Support vector machines

Support vector machines (SVMs) are a class of machine learning models that have been used in various fields, including time series forecasting. SVMs are particularly effective when dealing with high-dimensional data and can be used for both classification and regression tasks. In the context of time series forecasting, SVMs can be used to predict future values based on historical observations. SVMs work by finding a hyperplane that maximally separates the data into different classes or predicts the target variable in the case of regression. The hyperplane is chosen to maximize the margin between the two classes or the distance between the hyperplane and the closest data points. SVMs can be further improved by using kernel functions that map the original feature space to a higher-dimensional space, allowing the model to capture more complex relationships between the input features and the target variable. Despite their effectiveness, SVMs can be sensitive to the choice of kernel function and may require careful tuning of the model parameters [32].

In terms of forecasting performance, SVMs have been found to perform well in certain scenarios. For example, SVMs have been successfully used to forecast the price of oil and other commodities [33]. However, like all machine learning models, SVMs may not perform well in all scenarios. Therefore, it is important to carefully evaluate the performance of the model on the specific forecasting task at hand and to compare it with other models to determine the best approach.

3.6.2.4 Gradient boosting machines

Gradient boosting machines (GBMs) are a powerful class of machine learning models that have been widely used in time series forecasting. GBMs are a type of ensemble model that combines multiple weak learners, such as decision trees or linear models, to create a more accurate and robust predictor. GBMs work by iteratively adding new models to the ensemble, with each new model attempting to correct the errors made by the previous models. This process continues until a stopping criterion is met, such as a maximum number of models or a minimum improvement in performance. GBMs are particularly effective in capturing complex nonlinear relationships between the input features and the target variable, making them well-suited for time series forecasting problems with complex and noisy data.

GBMs have been found to perform well in various time series forecasting tasks. For example, GBMs have been used to accurately forecast electricity demand [34] or stock prices [35]. However, GBMs can be computationally expensive and require careful tuning of hyperparameters to achieve optimal performance.

In summary, GBMs are a powerful class of machine learning models that have been successfully applied in time series forecasting. GBMs are able to capture complex relationships between the input features and the target variable, making them well-suited for forecasting problems with noisy and complex data. However, they can be computationally expensive and require careful tuning of hyperparameters.

3.6.3 Neural Networks

Neural networks are a powerful class of machine learning models that have been increasingly used in time series forecasting due to their ability to capture complex patterns and nonlinear relationships in the data. Neural networks consist of layers of interconnected nodes, or neurons, that are designed to mimic the behavior of neurons in the human brain. Each neuron receives inputs from other neurons, performs a nonlinear transformation on the input, and produces an output. The output of one layer is then used as the input for the next layer, and so on, until the final layer produces the predicted output.

There are several types of neural networks that can be used for time series forecasting, including feedforward neural networks (FFNNs), recurrent neural networks (RNNs), and long short-term memory (LSTM) networks. FFNNs are a type of neural network that consist of one or more layers of neurons, with each neuron in a given layer connected to every neuron in the previous layer. FFNNs are particularly well-suited for time series forecasting problems with a fixed set of input features. RNNs, on the other hand, are designed to handle time-varying input and output sequences. RNNs use a feedback loop to feed the output of the previous time step back into the network as an input for the current time step. This allows the network to capture temporal dependencies in the data and is particularly useful for time series forecasting. LSTM networks are a type of RNN that are specifically designed to handle long-term dependencies in the data. LSTM networks use a special type of memory cell that can selectively remember or forget information from previous time steps, allowing the network to maintain a longer memory of past inputs and outputs.

Neural networks have been shown to be effective in a wide range of time series forecasting tasks. For example, neural networks have been used to forecast stock prices [36], and electricity demand [37]. Neural networks have also been found to outperform traditional time series forecasting methods, such as ARIMA and exponential smoothing, in certain scenarios [26]. However, neural networks can be computationally expensive and require a large amount of data to train effectively.

Furthermore, neural networks can be difficult to interpret, making it challenging to understand how the model arrived at its predictions.

In summary, neural networks are a powerful class of machine learning models that have been increasingly used in time series forecasting due to their ability to capture complex patterns and nonlinear relationships in the data. There are several types of neural networks that can be used for time series forecasting, including FFNNs, RNNs, and LSTM networks. Neural networks have been shown to be effective in a wide range of forecasting tasks, but they can be computationally expensive and difficult to interpret. As with any predictive modeling approach, it is important to carefully evaluate the performance of neural network models on the specific forecasting task at hand and to compare them with other models to determine the best approach.

3.6.3.1 FFNNs

Feedforward neural networks (FFNNs) are a type of neural network that consists of one or more layers of interconnected neurons, with each neuron in a given layer connected to every neuron in the previous layer. The input is fed into the first layer, and the output of each layer is then fed into the next layer until the final layer produces the predicted output. FFNNs are particularly well-suited for time series forecasting problems with a fixed set of input features. In the context of time series forecasting, the input features might include lagged values of the time series, as well as exogenous variables that are known to influence the time series.

FFNNs have been shown to be effective in a wide range of time series forecasting tasks. For example, FFNNs have been used to forecast electricity demand [38] or air quality [39]. However, FFNNs can be sensitive to the choice of hyperparameters, such as the number of layers and the number of neurons in each layer. Furthermore, FFNNs may not perform as well as other types of neural networks, such as RNNs and LSTM networks, when dealing with time-varying input and output sequences or long-term dependencies in the data.

Despite their limitations, FFNNs are still a useful tool for time series forecasting, particularly in situations where the input features are fixed and known in advance. As with any predictive modeling approach, it is important to carefully evaluate the performance of FFNNs on the specific forecasting task at hand and to compare them with other models to determine the best approach.

FFNNs are a type of neural network that can be used for time series forecasting problems with a fixed set of input features. FFNNs have been shown to be effective in a wide range of forecasting tasks, but they can be sensitive to the choice of

hyperparameters and may not perform as well as other types of neural networks in certain scenarios.

3.6.3.2 CNN

Convolutional neural networks (CNNs) are a type of neural network that have been widely used in image and signal processing tasks, including time series forecasting. CNNs are particularly well-suited for problems with high-dimensional input data, such as images or time series data with multiple channels. In CNNs, each neuron in a given layer is connected only to a small region of the previous layer, rather than every neuron in the previous layer. This local connectivity reduces the number of parameters in the model and allows the network to effectively learn spatial and temporal patterns in the data.

In the context of time series forecasting, CNNs can be used to extract features from the time series data by convolving the input data with a set of learnable filters. The output of each convolutional layer is then fed into a fully connected layer to produce the final predicted output. CNNs have been used to forecast various time series, such as solar power [40] and financial markets [41]. CNNs have also been found to outperform other traditional time series forecasting methods, such as ARIMA and LSTM, in certain scenarios.

However, CNNs can be computationally expensive and require a large amount of data to train effectively. Additionally, interpreting the learned filters in a CNN can be challenging, making it difficult to understand how the model is making predictions. Despite these limitations, CNNs are a useful tool for time series forecasting, particularly in situations where the input data has a high-dimensional structure.

In summary, CNNs are a powerful class of neural networks that have been increasingly used in time series forecasting. CNNs are particularly well-suited for problems with high-dimensional input data and can be used to effectively learn spatial and temporal patterns in the data. Although CNNs can be computationally expensive and difficult to interpret, they have been shown to outperform traditional time series forecasting methods in certain scenarios. As with any predictive modeling approach, it is important to carefully evaluate the performance of CNNs on the specific forecasting task at hand and to compare them with other models to determine the best approach.

3.6.3.3 RNN

Recurrent neural networks (RNNs) are a type of neural network that are designed to handle time-varying input and output sequences. Unlike FFNNs, which treat each input as independent of the others, RNNs are able to maintain a memory of past inputs and use that information to make predictions about the future. This is achieved through the use of a feedback loop in the network, which allows the output of one time step to be fed back into the network as an input for the next time step. In this way, RNNs can capture temporal dependencies in the data and are particularly useful for time series forecasting.

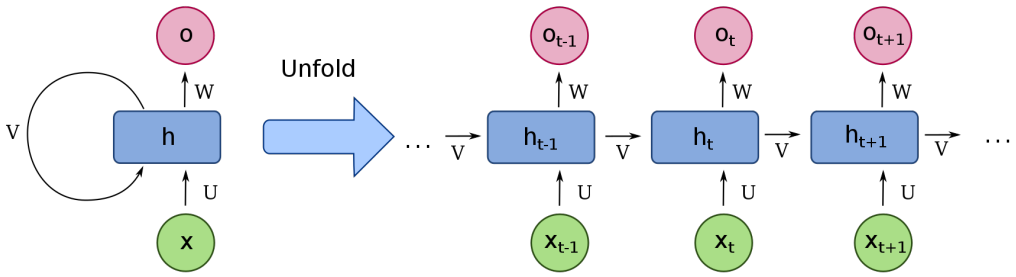

Figure 2: Recurrent neural network [42]

However, standard RNNs suffer from the problem of vanishing gradients, which can cause the network to have difficulty learning long-term dependencies in the data. This problem arises because the gradients used to update the weights of the network can become very small over time, leading to very slow learning or even no learning at all. To overcome this problem, long short-term memory (LSTM) networks were introduced as a variant of RNNs that are specifically designed to handle long-term dependencies.

3.6.3.4 LSTM

LSTM networks, a special type of RNN, use a special type of memory cell that can selectively remember or forget information from previous time steps, allowing the network to maintain a longer memory of past inputs and outputs. The memory cell is controlled by three gates: an input gate, which controls the amount of new information that is allowed into the memory cell; a forget gate, which controls the amount of old information that is allowed out of the memory cell; and an output gate, which controls the amount of information that is passed on to the next layer of the network. By selectively updating and forgetting information in the memory cell,

LSTM networks are able to maintain a longer memory of past inputs and outputs than standard RNNs, making them particularly useful for time series forecasting tasks that involve long-term dependencies.

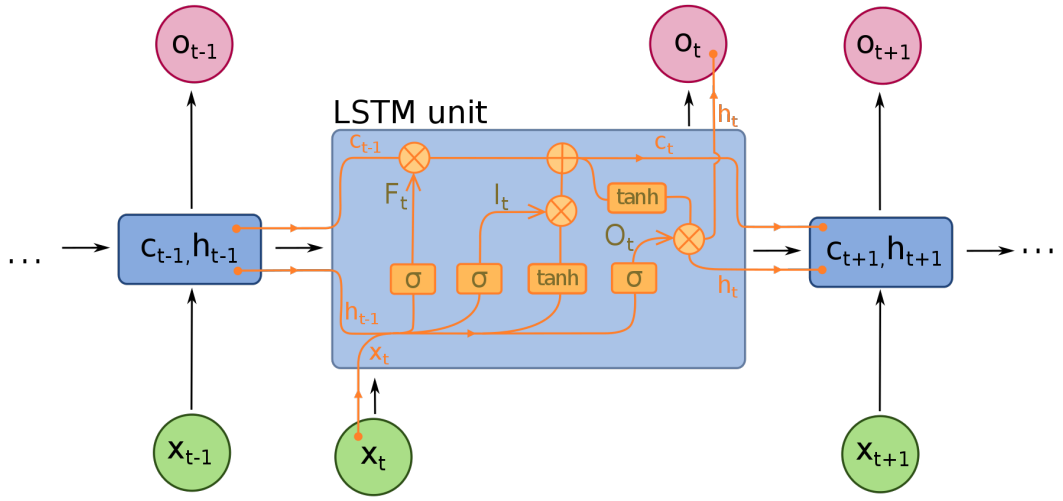

Figure 3: Long short-term memory unit [43]

LSTM networks have been shown to be effective in a wide range of time series forecasting tasks, such as traffic flow prediction [44], energy consumption forecasting [45], and cloud datacenters workload [46]. In certain scenarios, LSTM networks have been found to outperform other types of neural networks, such as FFNNs and standard RNNs, in terms of forecasting accuracy. However, like all neural networks, LSTMs can be computationally expensive and require a large amount of data to train effectively. Furthermore, interpreting the learned parameters in an LSTM network can be challenging, making it difficult to understand how the model is making predictions.

$$
\begin{aligned}
f_t &= \sigma_g\left(W_f x_t + U_f h_{t-1} + b_f\right) \\
i_t &= \sigma_g\left(W_i x_t + U_i h_{t-1} + b_i\right) \\
o_t &= \sigma_g\left(W_o x_t + U_o h_{t-1} + b_o\right) \\
\tilde{c}_t &= \sigma_c\left(W_c x_t + U_c h_{t-1} + b_c\right) \\
c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \\
h_t &= o_t \odot \sigma_h\left(c_t\right)
\end{aligned}
\tag{3.1}
$$

The LSTM equations 3.1 describe the inner workings of the memory cell. The input vector to the LSTM unit is denoted as $x_t \in \mathbb{R}^d$. The forget gate's activation vector is $f_t \in (0,1)^h$, the input/update gate's activation vector is $i_t \in (0,1)^h$, and the

output gate's activation vector is $o_t \in (0,1)^h$. The hidden state vector, also known as the output vector of the LSTM unit, is $h_t \in (-1,1)^h$, the cell input activation vector is $\tilde{c}_t \in (-1,1)^h$, and the cell state vector is $c_t \in \mathbb{R}^h$. The weight matrices are denoted as $W \in \mathbb{R}^{h \times d}$ and $U \in \mathbb{R}^{h \times h}$, while the bias vector parameters are $b \in \mathbb{R}^h$. The superscripts d and h refer to the number of input features and number of hidden units, respectively.

$$
\begin{aligned}
x_t \in \mathbb{R}^d &: \text{input vector to the LSTM unit} \\
f_t \in (0,1)^h &: \text{forget gate's activation vector} \\
i_t \in (0,1)^h &: \text{input/update gate's activation vector} \\
o_t \in (0,1)^h &: \text{output gate's activation vector} \\
h_t \in (-1,1)^h &: \text{hidden state vector, output vector of the LSTM unit} \\
\tilde{c}_t \in (-1,1)^h &: \text{cell input activation vector} \\
c_t \in \mathbb{R}^h &: \text{cell state vector} \\
W \in \mathbb{R}^{h \times d}, U \in \mathbb{R}^{h \times h} &: \text{weight matrices} \\
b \in \mathbb{R}^h &: \text{bias vector parameters, learned during training}
\end{aligned}
$$

The activation functions used in the LSTM equations are the sigmoid function σ_g for the forget, input/update, and output gates, and the hyperbolic tangent function σ_c for the cell input activation vector and σ_h for the hidden state vector. The peephole LSTM paper [47] suggests using $\sigma_h(x) = x$.

$-\sigma_g$: sigmoid function.

$-\sigma_c$: hyperbolic tangent function.

$-\sigma_h$: hyperbolic tangent function or$\sigma_h(x) = x$.

In summary, RNNs and LSTM networks are powerful classes of neural networks that are designed to handle time-varying input and output sequences. RNNs use a feedback loop to maintain a memory of past inputs, while LSTMs use a special type of memory cell to selectively remember or forget information from previous time steps, allowing them to handle long-term dependencies. Both RNNs and LSTMs have been shown to be effective in a wide range of time series forecasting tasks, but they can be computationally expensive and difficult to interpret. As with any predictive modeling approach, it is important to carefully evaluate the performance

of these models on the specific forecasting task at hand and to compare them with other models to determine the best approach.

Practical part

4 Introduction

The practical part of this thesis involves implementing a time series forecasting system for city traffic data. The system will be developed using Apache Airflow for data ingestion and ETL processing, with data being sourced from an external API. The processed data will be loaded into a MySQL relational database for use in model training and evaluation. The predictive models will be implemented using various machine learning techniques, including classical model ARIMA, as well as deep learning models such as CNNs and RNNs. The primary model used will be an RNN LSTM, a type of neural network that is well-suited for capturing long-term dependencies in sequential data. The performance of each model will be evaluated using a variety of metrics. Finally, a web application will be developed to display the model's predictions in real-time, providing valuable insights into city traffic patterns and helping to inform decision-making for urban planning and transportation management.

5 Environment

In this chapter, we will discuss the environment and tools used for the implementation of the time series forecasting system. The system will be deployed on Google Cloud Platform (GCP), which provides a scalable and cost-effective infrastructure for hosting web applications and data processing. The workflow management platform used in this project is Apache Airflow, which provides a flexible and scalable way to manage ETL workflows. The data storage component will be implemented using MySQL, an open-source relational database management system that is widely used for storing structured data. We will also describe the process of setting up and configuring each of these tools in detail. This will include the steps required to create a GCP instance, install and configure Airflow, and set up a MySQL database for data storage.

5.1 Google Cloud Platform

Google Cloud Platform (GCP) is a cloud computing platform that offers a wide range of services and tools for building and deploying applications, including virtual machines (VMs), databases, and machine learning services. GCP allows users to run applications in a scalable and cost-effective manner, with the ability to easily adjust resources as needed. This makes it an ideal platform for running data-intensive applications such as time series forecasting.

To begin using GCP, I first created an account on the platform and set up billing information. I then created a VM instance on GCP using the Debian operating system. The VM instance is used to run data extractions continuously, which cannot be done on my laptop due to its limited resources. The VM instance provides a scalable and flexible environment for running my time series forecasting system.

In addition to the VM instance, I also set up monitoring on GCP to track resource usage and ensure that the system is running smoothly. This includes monitoring CPU and disk usage to prevent resource bottlenecks and optimize performance. Additionally, I configured the network settings on GCP to ensure that

the data is securely transferred between the VM instance and other components of the system.

Using GCP as the platform for my time series forecasting system provides many benefits, including scalability, cost-effectiveness, and flexibility. With GCP, I am able to run my system in a reliable and secure environment, while also being able to easily adjust resources as needed. This allows me to focus on developing and improving my forecasting models, without the need to worry about the underlying infrastructure.

5.1.1 Virtual Machine Instance

The virtual machine instance or VM runs the Debian operating system and was created to enable the continuous data extraction necessary for this project. The VM instance can be viewed on the GCP console, where its details, including the instance name, status, creation time, machine configuration, and networking settings, can be accessed. The machine type selected for the VM instance is e2-custom-4-6912, with Intel Broadwell as the CPU platform. The architecture of the instance is x86/64, and it features 4 vCPUs with 6.91 GB of memory. The boot disk of the instance is a balanced persistent disk with a size of 100 GB, and its interface type is SCSI. The machine has a default network interface, and its primary internal IP address is 10.132.0.4. The VM instance can be scaled up or down depending on the specific requirements of the project. For example, the disk can be scaled to a larger size to accommodate more data or to improve the performance of the instance.

Initially, I started with a disk size of 30 GB for my VM instance on GCP, but soon realized that it was not sufficient for my needs. I had to scale up the disk size twice, first to 50 GB and then to 100 GB, to accommodate my data extractions and other processes. This flexibility to scale up the resources as needed is one of the great advantages of cloud computing instances, allowing users to easily adjust their resource allocation to match their changing needs.

5.1.1.1 SSH connection

To connect to the VM instance on GCP, SSH protocol is used. SSH provides a secure, encrypted connection to the VM instance, allowing remote access to its command-line interface. The SSH connection requires authentication using a public/private key pair, with the public key stored on the VM instance and the private key stored locally on the user's computer. Once authenticated, the user can access the command-line interface of the VM instance and perform various tasks such as installing software, managing files, and monitoring system performance. SSH also

provides the ability to securely transfer files between the local computer and the VM instance using the SCP protocol.

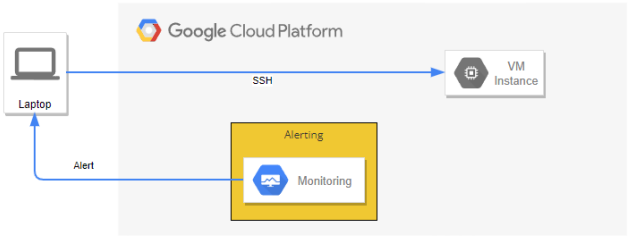

Figure 4: Google Cloud Platform schema

5.1.2 Monitoring

In addition to providing a reliable and scalable infrastructure, Google Cloud Platform (GCP) also offers powerful monitoring capabilities. In my project, I have set up monitoring to track the CPU usage and disk size of my VM instance. By monitoring these metrics, I can keep track of the resources being used by my system and ensure that I have enough capacity to support my workloads. To facilitate this monitoring, I have set up alerts in GCP that notify me if the CPU usage or disk size exceeds certain thresholds. This allows me to take proactive steps to address any issues before they become critical, such as deleting unnecessary files or increasing the boot disk size or CPU of my VM instance. Overall, monitoring is a crucial component of any cloud-based system, and GCP provides powerful and flexible tools for monitoring resource utilization and ensuring the reliability of my system.

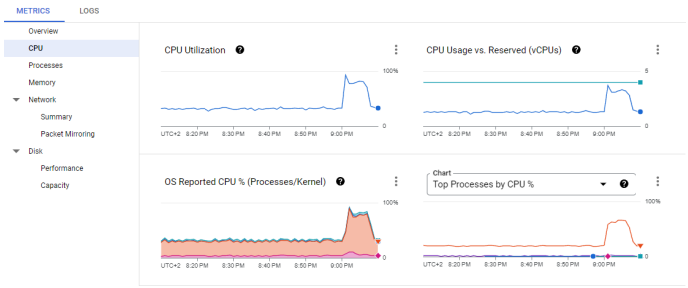

Figure 5: CPU monitoring console

5.2 Airflow

To set up Airflow on my GCP Debian VM, I followed the official Apache Airflow documentation. First, I ensured that my VM was set up with the necessary resources, including 4 CPUs and 16GB of RAM. Then, I accessed the VM using SSH and installed Python 3 and `pip3`, the package installer for Python 3.

Next, I installed `virtualenv`, which allows me to create an isolated environment for installing Python packages. I then created a new virtual environment called `venv` using the command `virtualenv -p python venv` and activated it using `source venv/bin/activate`. With my virtual environment set up, I then installed Airflow using the `pip3` command and the appropriate constraints file for my Python version.

After successfully installing Airflow, I created a new directory called `diploma` and set the `AIRFLOW_HOME` environment variable to point to that directory. I then initialized the Airflow metadata database using the `airflow db init` command, and started the Airflow scheduler using `airflow scheduler` in one terminal window.

In another terminal window, I activated my virtual environment and set the `AIRFLOW_HOME` variable once again. I then created a new Airflow user with administrator privileges using the `airflow users create` command, providing a username, first name, last name, role, and email. I set the password to `admin` for simplicity.

Finally, I listed the Airflow users to confirm that the new user had been created successfully, and started the Airflow web server using `airflow webserver -p 8080`. With Airflow up and running, I was able to begin developing my time series forecasting system for city traffic data.

5.2.1 Systemd

After installing and configuring Airflow on my GCP VM, I wanted to ensure that the webserver and scheduler would run continuously without the need for me to manually start them up. To achieve this, I used systemd to create a service that would start up the webserver and scheduler on boot and keep them running in the background.

I created a script file called `airflow_webserver.sh` in the `scripts` folder that contained the necessary commands to start up the webserver. I then created a systemd service file called `airflow-webserver.service` that specified the location of the script file and other relevant settings. Since I didn't have sufficient permissions to create a system-wide service file in the `/etc/systemd/system` directory, I created

the service file in my user directory and then copied it to the system directory using the command `sudo cp airflow-webserver.service /etc/systemd/system/`.

After copying the service file, I reloaded the systemd daemon, enabled the service to start up on boot, and started the service using the following commands:

Listing 5.1: Starting the Airflow webserver service using systemd

```
sudo systemctl daemon-reload
sudo systemctl enable airflow-webserver.service
sudo systemctl start airflow-webserver.service
sudo systemctl status airflow-webserver.service
```

With this setup, the webserver and scheduler will start up automatically on boot and will continue running in the background even if I log out of the VM.

5.2.2 Directed Acyclic Graphs (DAGs)

To create a DAG in Airflow, you need to define the various tasks that will make up the DAG, set their dependencies, and schedule the DAG to run at regular intervals. Each task in the DAG is defined as an operator, which can be a PythonOperator, BashOperator, or one of several other types.

In the DAG `parking_measurements`, a small sample of which is available below, we start by defining some variables that will be used throughout the DAG, including the entity name, API endpoint, query parameters, and database connection ID. We then define a DAG object with a unique ID, start date, and schedule interval. In this case, we set the DAG to run hourly.

Next, we define two Python functions that will be used as tasks in the DAG. The first function checks if the API endpoint is available and raises an exception if it is not. The second function downloads data from the API and saves it to a CSV file.

We then define four MySQLOperator tasks that create a new table in the database, insert the downloaded data into the table, and transform the data as necessary. Each task corresponds to a SQL file in a specific directory, which allows for easy organization and management of the SQL scripts.

Finally, we set the dependencies between the tasks by using the "»" operator to specify which task should run after which. As seen in Figure 6, the check endpoint task is set to run before the download file task in order to ensure the necessary data is available before it is downloaded and processed and so on.

It's important to note that this is just one example DAG and that each DAG will have its own unique set of tasks, dependencies, and parameters. However, by

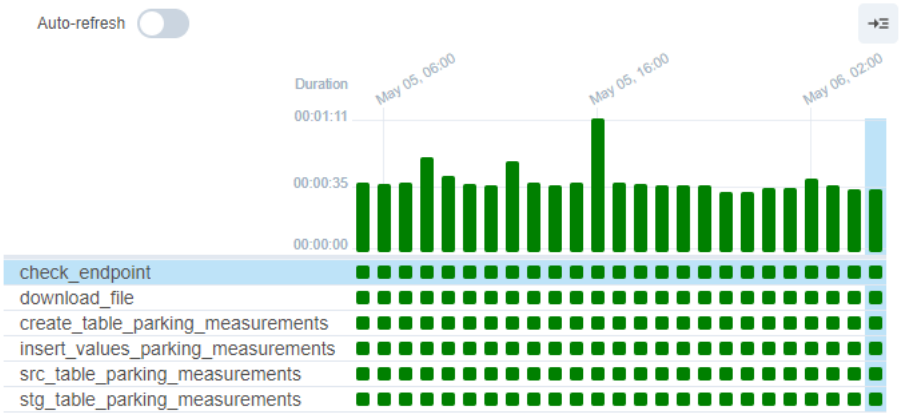

Figure 6: Airflow tasks detail view

following the general structure outlined above, you can create a wide variety of DAGs to meet your specific needs.

Listing 5.2: DAG for parking data

```
.
.
# Define the DAG
dag = DAG(
    dag_id=def_entity,
    start_date=datetime(2023, 3, 12),
    schedule_interval='0 * * * *',
    catchup=False,
    template_searchpath=["/home/diploma/sql"]
)
.
.
# Define a task to download the file
download_file_task = PythonOperator(
    task_id='download_file',
    python_callable=download_file,
    op_kwargs={
        'headers': headers,
        'endpoint': def_endpoint,
        'query': def_query,
        'filename': def_entity
    },
```

```
    dag=dag,
)
.
.
# Set task dependencies
check_endpoint_task >> download_file_task >> create_table >>
insert_values >> src_table >> stg_table
```

In addition to the configurations, all DAGs that I have developed for my time series forecasting system are also available on my Git repository or in the attachment of my diploma.

5.3 MySQL

To install MySQL on my VM, I followed the instructions provided by the official Google Cloud Platform documentation [48]. After installing MySQL, I made sure to improve the installation security by removing anonymous users, disabling remote root login, and removing test databases.

Next, I needed to ensure that MySQL was configured to allow remote access. By default, MySQL is configured to only listen to connections from localhost. To modify this setting, I added the following lines to the MySQL configuration file located at /etc/mysql/mysql.conf.d/mysqld.cnf:

Listing 5.3: MySQL configuration file

```
port = 3306
bind-address = 0.0.0.0
```

These lines specify that MySQL should listen on all available IP addresses and not just the localhost IP address.

After modifying the MySQL configuration file, I restarted the MySQL service to ensure that the changes took effect by running the following command:

Listing 5.4: Restart MySQL service

```
sudo service mysql restart
```

To create a new MySQL user with remote access privileges, I logged into the MySQL server as the root user:

Listing 5.5: Log in to MySQL as root user

```
mysql -u root -p
```

I then created a new user with remote access privileges and granted them access to the database I wanted to access:

Listing 5.6: Create new user with remote access privileges

```
CREATE USER 'new_user'@'%' IDENTIFIED BY 'password';
GRANT ALL PRIVILEGES ON mydatabase.* TO 'new_user'@'%';
```

The above SQL commands create a new user named "new_user" with the password "password" and grant them all privileges on the "mydatabase" database. The "%" wildcard character specifies that the user should be able to connect from any IP address.

Finally, I used the MySQL client in the MySQL Workbench to connect to the database, specifying the IP address of my Debian VM as the hostname and using the new username and password to log in. With this setup, I was able to efficiently collect and transform data from an API and store it in a MySQL database using Airflow.

5.3.1 Backup files

MySQL binary log files, also known as binlogs, are used for data recovery and replication. However, if not managed properly, they can accumulate and take up a significant amount of disk space. To prevent this from happening, I configured a cron job to delete binlog files older than one week every Sunday. This ensured that my VM had enough disk space for new data and prevented any potential issues that could arise from running out of disk space. Proper management of binlog files is crucial to maintaining a healthy and efficient MySQL database.

6 Data

This chapter provides an overview of the data used in the time series forecasting system for city traffic data implemented in this thesis. This chapter is divided into three sections: Data Sources, Data Extraction, and Data Preprocessing.

In the Data Sources section, we discuss the various data sources used in the system and how they were tested for reliability and accuracy. We focus on three main data sources: parking data, vehicle data, and weather data. We provide an overview of each data source and discuss how they were integrated into the forecasting system.

In the Data Extraction section, we describe the process of extracting data from the various sources using Python and the Apache Airflow platform. We discuss the importance of data orchestration and how we used Airflow to schedule and automate the extraction process. We also provide an overview of the data loading process into the MySQL database.

Finally, in the Data Preprocessing section, we discuss the various SQL transformation techniques used to preprocess the data for model training. We discuss the importance of data quality and how we cleaned, transformed, and prepared the data to be used in the forecasting models.

Overall, this chapter provides a comprehensive overview of the data used in the forecasting system and how it was extracted, processed, and prepared for model training.

6.1 Data Sources

The Data Sources section of this chapter focuses on the three main data sources used in the time series forecasting system for city traffic data implemented in this thesis. The data is sourced from external APIs and is collected and processed to provide valuable insights into city traffic patterns. The first data source is parking data, which provides information about parking availability in the P+R spots, large parking lots around the city. The second data source is vehicle data, which provides information about public transport vehicles, including their location, speed, and

delay. This data is the most important data source for the system, as it forms the basis for the time series forecasting models. The third data source is weather data, which provides information about temperature, humidity, snow, and other weather-related variables. All three data sources are integrated into the time series forecasting system to provide a comprehensive picture of city traffic patterns.

6.1.1 Golemio

For this thesis, I investigated several APIs to identify a data source for city traffic forecasting. Among them, the Golemio API appeared to be a valuable resource for acquiring public transport data in Prague. The API provides access to various datasets related to public transportation in Prague, including data on vehicle locations, routes, schedules, and occupancy rates. Golemio's REST-based interface is user-friendly and allows developers to quickly and easily obtain the data they require. The API is an essential data source for the time series forecasting system implemented in this thesis as it provides important information about the flow of public transport in the city at different times of the day.

To gain authorization for accessing the Golemio API, I created an account using my Gmail address and obtained an API key. The key is used to authenticate each request to the API and track usage. To ensure that my API requests were authenticated, I added the API key to the headers of each request. Using Python scripts, I performed extensive testing to ensure that the API provided the necessary data and excluded irrelevant or redundant information. Moreover, I performed various statistical analyses on the data to verify its accuracy and suitability for use in forecasting models.

6.1.1.1 Data exploration

The code snippet below is a Python script that extracts data from the Golemio API. It begins by importing the necessary libraries, requests, and json. The headers variable contains the authorization token, which is necessary to access the Golemio API.

Listing 6.1: Example of Python script for extracting data from Golemio API

```
import requests
import json

headers = {"Content-Type": "application/json; charset=utf-8",
          "x-access-token": 'X'}
```

```
def get_parking_data(headers, endpoint, query):
    url = f'https://api.golemio.cz/v2/{endpoint}{query}'
    print(url)
    response=requests.get(url, headers=headers)
    data = response.json()
    with open('test_parking.json', 'w') as f:
        f.write(json.dumps(data, sort_keys=True, indent=4))
    return data

data = get_parking_data(headers, 'parking/measurements',
      '?source=TSK&sourceId=534017')
```

The `get_parking_data()` function is defined with parameters for headers, endpoint, and query. This function constructs the API request URL using the endpoint and query parameters and sends a GET request to the URL using the headers parameter to authenticate. The response is then converted to JSON format and saved to a local file called `test_parking.json` for further analysis.

The final line of the code calls the `get_parking_data()` function with the headers, endpoint, and query parameters to retrieve data from the Golemio API related to parking measurements for a specific source and source ID. In this example, I analyzed the data of 1 parking house with ID = 534017. It is a P+R parking house Chodov. This data can then be used for further analysis or processing as necessary for use in the time series forecasting system.

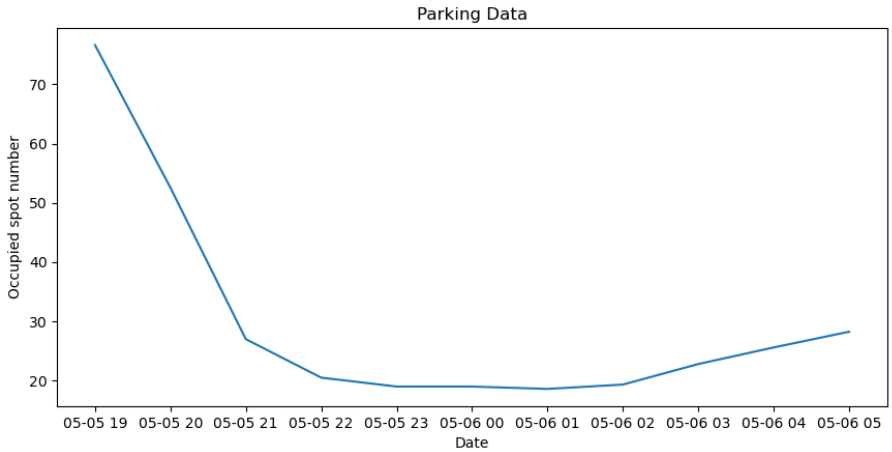

Figure 7: Visualization of parking occupancy

Figure 7 shows a visualization of parking data from evening 5th May to morning 6th May. The plot reveals a clear trend of fewer cars in the parking lot in the evening as people start leaving the P+R, resulting in a decrease in the number of occupied spots. On the other hand, there is an increasing trend in the morning as people start arriving, leading to a rise in the number of occupied spots. This information can be valuable for parking lot management as it helps in planning resources and improving the user experience.

6.1.1.2 Summary

The testing process ensured that the data extracted from the Golemio API was of high quality and suitable for use in the time series forecasting system. The Golemio API proved to be a robust and reliable source of public transportation data in Prague. Its accessibility as a public API made it easy to access and use, and the documentation and community resources provided by Golemio made it straightforward to work with.

6.1.2 Weather

In order to obtain weather data for the city traffic forecasting system, I explored various weather APIs that were available. After careful consideration, I chose to use the Open Meteo API (https://open-meteo.com/) for several reasons. First and foremost, the API provides all of the weather data that is needed for the forecasting system, including temperature, humidity, snow, and rain. Additionally, the API provides historical weather data, allowing for the retrieval of past weather information to use in the forecasting models. Finally, the API offers a forecast API that provides weather predictions for up to 7 days in advance, which is crucial for accurate traffic forecasting.

To ensure the data obtained from the API was of high quality and met the needs of the forecasting system, extensive testing was performed using Python scripts to extract, clean, and analyze the data. This testing confirmed that the Open Meteo API provided the necessary data and excluded any irrelevant or redundant information. Furthermore, various statistical analyses were performed on the data to verify its accuracy and suitability for use in the forecasting models.

6.1.2.1 Data exploration

The script below demonstrates the use of the Open-Meteo weather API to retrieve historical weather data for a specific location. The `get_weather_data`

function takes an endpoint and a query string as parameters, which are then used to construct the API URL. The query string specifies the latitude and longitude of the location, the start and end dates for the data, and the hourly and daily data parameters that are requested. In the example, we are testing historical data from 1/1/2021 to 1/5/2023, i.e., Prague data, which is determined by longitude and latitude values. I have chosen the time zone Europe/Berlin, which corresponds to the time zone of Prague.

Listing 6.2: Example of Python script for extracting data from Golemio API

```
import requests
import json

def get_weather_data(endpoint, query):
    url = f'https://archive-api.open-meteo.com/{endpoint}{query}'
    print(url)
    response=requests.get(url)
    data = response.json()
    with open('test_weather.json', 'w') as f:
        f.write(json.dumps(data, sort_keys=True, indent=4))
    return data

data = get_weather_data('v1/archive','?latitude=50.09&longitude=14.42
        &start_date=2021-01-01&end_date=2023-05-01&
        hourly=temperature_2m,rain,snowfall&daily=sunrise
        &timezone=Europe%2FBerlin')
```

The API returns the requested data in JSON format, which is then stored in a file called `"test_weather.json"` using the json module. The data contains hourly temperature, rain and snowfall data and daily sunrise data. This script allows easy access to historical weather data for any location by editing query parameters, which can be useful for analyzing correlations between weather and other variables.

The time series plots of temperature, snow, and rain from January 2021 to May 2023 provide valuable insights into the seasonal patterns and trends of weather conditions in the city. The temperature plot 8 shows a clear seasonal trend, with warmer temperatures occurring in the summer months and colder temperatures in the winter months. This is reflected in the periodic peaks and valleys in the plot, with the highest temperatures occurring in the summer and the lowest temperatures in the winter.

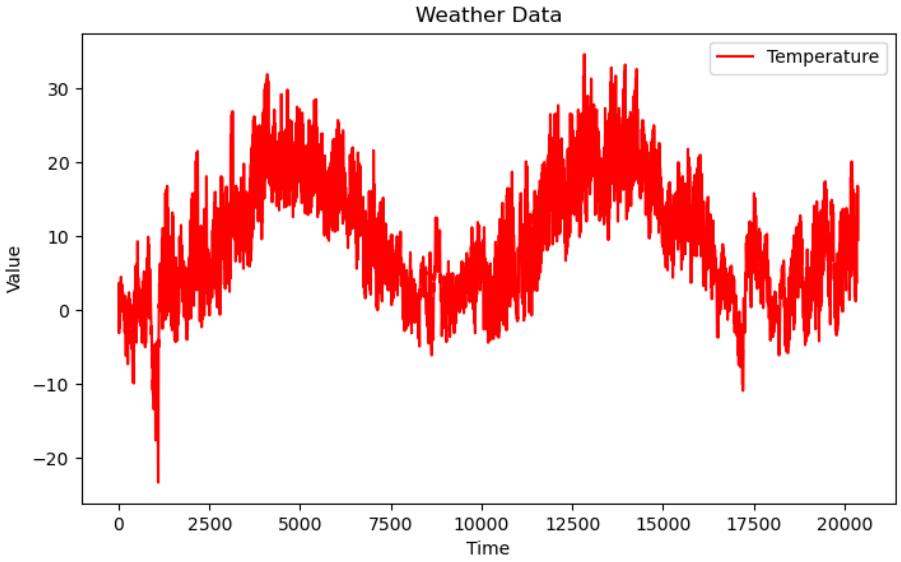

Figure 8: Visualization of temperature

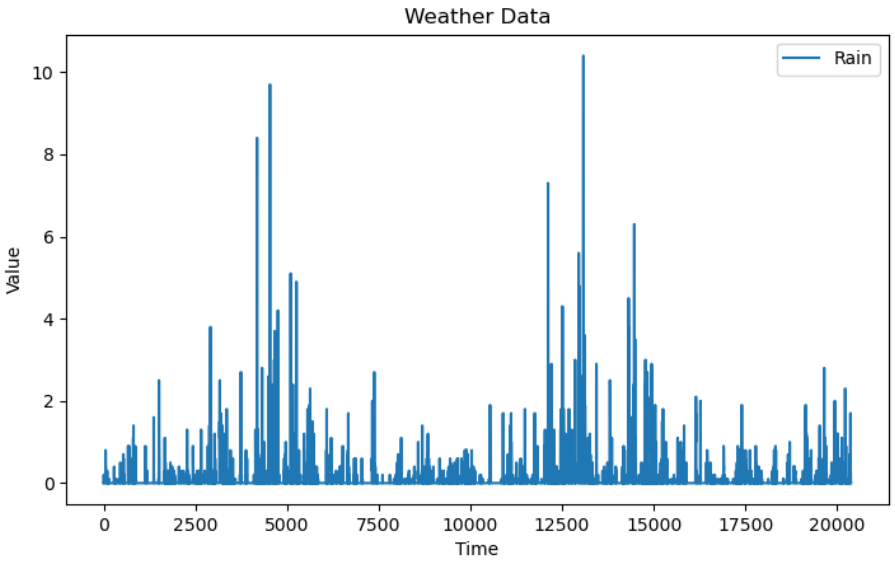

Figure 9: Visualization of rain

The snow plot 10 shows a much more erratic pattern, with snowfall occurring sporadically throughout the time period. However, it is possible to observe that snowfall mostly occurs during the winter months. Finally, the rain plot 9 also exhibits a seasonal pattern, with higher amounts of rainfall occurring in the spring and summer months, and lower amounts in the fall and winter. Overall, these plots provide valuable information about the seasonal patterns of weather conditions in the city, which can be used to inform decisions related to urban planning and transportation management.

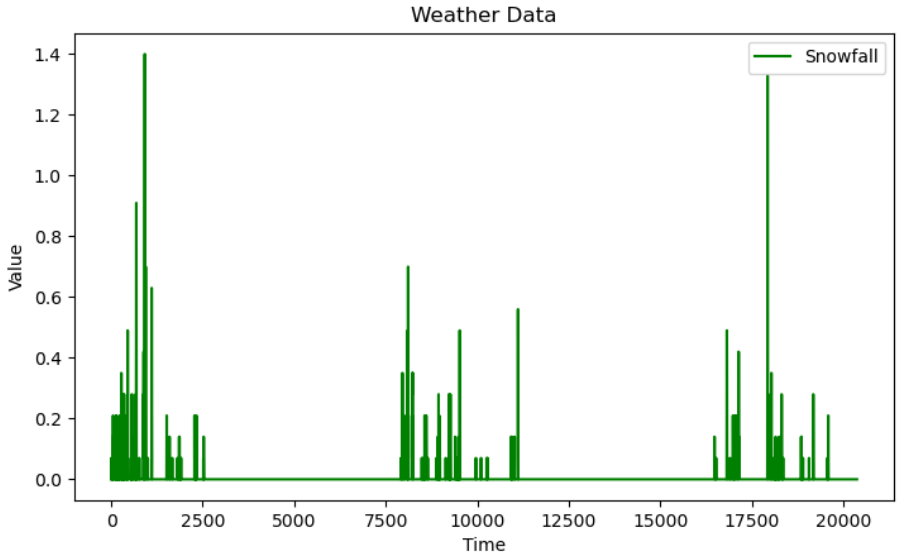

Figure 10: Visualization of snowfall

6.1.2.2 Summary

In summary, the Open Meteo API proved to be a reliable and comprehensive source of weather data for the city traffic forecasting system. Its simplicity and availability of historical data made it a clear choice for the project. The forecast API also proved to be a crucial component, providing important predictions for future weather conditions. The thorough testing and analysis of the data obtained from the API ensured that the data was of high quality and suitable for use in the time series forecasting system.

6.2 Data extraction

The data extraction process is a crucial step in the implementation of any data-driven system. In the context of the city traffic forecasting system, data

extraction involves retrieving traffic and weather data from external APIs and storing it in a format suitable for use in forecasting models.

One of the challenges in obtaining data for the time series forecasting system is that the public transport data that is needed for the system does not have historical data readily available. This means that simply downloading data from the API at a single point in time is not sufficient for the forecasting system, as it requires historical data to train the models. As a result, a process was needed to automatically extract the data from the API at regular intervals to create a historical database of public transport information. While it is possible to download historical weather data directly from APIs like the Open Meteo API, this is not possible for public transport data, which makes the use of Airflow necessary for creating a comprehensive historical database.

Two tasks in the Airflow DAGs are dedicated to data extraction: the Check Endpoint task and the Download File task. The Check Endpoint task is responsible for ensuring that the API endpoint is functioning correctly before attempting to download any data. The Download File task, on the other hand, is responsible for using Python scripts to extract the necessary data from the APIs and save it in a .json file format. Both tasks are discussed in more detail in the following chapters, where I will describe everything on the example of parking data, as the mechanisms used for all data sources are the same.

6.2.1 Endpoint check

The first task in the DAG is the endpoint check. This task is designed to ensure that the API endpoint is accessible before proceeding with the data download. If the endpoint is unavailable or returns an error message, the task will fail, and the data download task will not execute. The Python function that performs this task uses the requests library to send an HTTP GET request to the endpoint and checks the response code to determine if the endpoint is available. If the response code is not 200, indicating that the endpoint is not available, a ValueError is raised, which results in the task failure. The endpoint check is crucial because if the endpoint is down, or the data is not available, it is pointless to proceed with the data download. Therefore, this task helps ensure the integrity of the data and the efficiency of the entire DAG. In the following chapter, the data download task is described in more detail.

Listing 6.3: Definition of Airflow task Check Endpoint

```
# Define a Python function to check if the endpoint is available
```

```
def check_endpoint(headers, endpoint, query):
    url = f'https://api.golemio.cz/v2/{endpoint}{query}'
    response = requests.get(url,headers=headers)
    if response.status_code != 200:
        raise ValueError('Endpoint not available')

# Define a task to check if the endpoint is available
check_endpoint_task = PythonOperator(
    task_id='check_endpoint',
    python_callable=check_endpoint,
    op_kwargs={
        'headers': headers,
        'endpoint': def_endpoint,
        'query': def_query,
        'filename': def_entity
    },
    dag=dag,
)
```

6.2.2 Python extraction

The second task in the DAG is the download file task, which executes the `download_file` Python function. This function uses the requests library to send an HTTP GET request to the API endpoint and stream the response content to a local file on the server. The task is defined as a PythonOperator with the `download_file` function specified as the callable. The function takes the same arguments as the `check_endpoint` function (headers, endpoint, query) in addition to a filename argument to specify the local file to save the data to. The file is saved to the /tmp directory with the same name as the entity (in this case, `parking_measurements.csv`). The `download_file_task` is dependent on the successful execution of the `check_endpoint_task`, and must be completed before the subsequent tasks can run.

Listing 6.4: Definition of Airflow task Download File

```
# Define a Python function to download the file
def download_file(headers, endpoint, query, filename):
    url = f'https://api.golemio.cz/v2/{endpoint}{query}'
    response = requests.get(url, headers=headers, stream=True)
```

```
    with open(f'/tmp/{filename}.csv', 'wb') as f:
        for chunk in response.iter_content(chunk_size=1024):
            if chunk:
                f.write(chunk)

# Define a task to download the file
download_file_task = PythonOperator(
    task_id='download_file',
    python_callable=download_file,
    op_kwargs={
        'headers': headers,
        'endpoint': def_endpoint,
        'query': def_query,
        'filename': def_entity
    },
    dag=dag,
)
```

The output of the downloaded file you see below in listing 6.5 is a list of dictionaries.. Each dictionary in the list represents a parking measurement and contains the following keys: `"available_spot_number"`, `"closed_spot_number"`, `"date_modified"`, `"occupied_spot_number"`, `"parking_id"`, `"source"`, `"source_id"`, and `"total_spot_number"`. The values associated with these keys provide information about the parking spot availability, the time when the measurement was taken, and other details about the parking location. The data is in JSON format and will be transformed into tables in MySQL format in the Data Preprocessing section.

Listing 6.5: Example of extracted parking data

```
[
    {
        "available_spot_number": 604,
        "closed_spot_number": null,
        "date_modified": "2023-05-06T05:19:00.000Z",
        "occupied_spot_number": 29,
        "parking_id": "tsk-534016",
        "source": "tsk",
        "source_id": "534016",
```

```
        "total_spot_number": 633
    },
    {
        "available_spot_number": 604,
        "closed_spot_number": null,
        "date_modified": "2023-05-06T05:12:58.000Z",
        "occupied_spot_number": 29,
        "parking_id": "tsk-534016",
        "source": "tsk",
        "source_id": "534016",
        "total_spot_number": 633
    },
```


6.3 Data Preprocessing

In the previous chapter, I downloaded the required data in JSON format. In this section, I will load the data into the MySQL database and transform it from JSON to tables. The first step is to create the necessary tables according to the schema of my diploma project. I will have three tables in my data flow: `pre_`, `src_`, and `stg_`. The `pre_` table will store the data from the downloaded file, which will contain multiple records in a single list. I will need to parse these records into separate rows. The `src_` table will be the source table, which will contain the parsed data from the `pre_` table. Finally, the `stg_` table will be the stage table, where the parsed JSON data will be transformed into typical table format. Once the data is processed and transformed, I will create output views for further analysis.

6.3.1 Create tables

In the context of parking data, the first step in data preprocessing is to create tables in the MySQL database to hold the downloaded data. The `create_table` task in the DAG executes a SQL script that creates three tables: `pre_parking_measurements`, `src_parking_measurements`, and `stg_parking_measurements`. These tables are designed to store the parking data in different stages of transformation.

Listing 6.6: Create table task

```
create_table = MySqlOperator(
```

```
    sql=f'/{def_endpoint}/create_table.sql',
    task_id=f"create_table_{def_entity}",
    mysql_conn_id=def_conn_id,
)
```

The `pre_parking_measurements` table is the initial destination of the data downloaded from the Golemio API. It has a single column called `jdoc` that stores the data in JSON format. The table also has several metadata columns (`_sys_record_id`, `_sys_load_id`, `_sys_load_at`, and `_sys_is_deleted`) to keep track of the source and loading details.

The `src_parking_measurements` table is used to parse the data in the `jdoc` column of the `pre_parking_measurements` table into individual records. It has two additional columns compared to the `pre_parking_measurements` table: `i` and `jdoc`. The `i` column represents the index of a particular parking measurement in the original JSON array, and the `jdoc` column stores the parsed JSON data for that particular measurement. The table also has the same metadata columns as the `pre_parking_measurements` table.

Listing 6.7: Create table SQL scripts

```
CREATE TABLE IF NOT EXISTS pre_parking_measurements (
    _sys_record_id INT NOT NULL AUTO_INCREMENT,
    jdoc JSON,
    _sys_load_id INT,
    _sys_load_at TIMESTAMP,
    _sys_is_deleted BOOLEAN,
    CONSTRAINT id PRIMARY KEY (_sys_record_id, _sys_load_at)
);

CREATE TABLE IF NOT EXISTS src_parking_measurements (
    _sys_record_id INT,
    _sys_load_id INT,
    _sys_load_at TIMESTAMP,
    _sys_is_deleted BOOLEAN,
    i INT,
    jdoc JSON,
    CONSTRAINT id PRIMARY KEY (_sys_record_id,i,_sys_load_at)
);
```

```
CREATE TABLE IF NOT EXISTS stg_parking_measurements (
    parking_measurement_id VARCHAR(255) NOT NULL,
    source VARCHAR(255),
    source_id VARCHAR(255),
    parking_id VARCHAR(255),
    date_modified TIMESTAMP,
    total_spot_number VARCHAR(255),
    closed_spot_number VARCHAR(255),
    occupied_spot_number VARCHAR(255),
    available_spot_number VARCHAR(255),
    _sys_record_id INT,
    _sys_load_id INT,
    _sys_load_at TIMESTAMP,
    _sys_is_deleted VARCHAR(255),
    i INT,
    PRIMARY KEY (parking_measurement_id)
  );

CREATE TABLE IF NOT EXISTS stg_parking_measurements_backup (
    parking_measurement_id VARCHAR(255) NOT NULL,
    source VARCHAR(255),
    source_id VARCHAR(255),
    parking_id VARCHAR(255),
    date_modified TIMESTAMP,
    total_spot_number VARCHAR(255),
    closed_spot_number VARCHAR(255),
    occupied_spot_number VARCHAR(255),
    available_spot_number VARCHAR(255),
    _sys_record_id INT,
    _sys_load_id INT,
    _sys_load_at TIMESTAMP,
    _sys_is_deleted VARCHAR(255),
    i INT,
    PRIMARY KEY (parking_measurement_id)
  );
```

The stg_parking_measurements table is the final destination of the parsed parking data. It has columns for all the relevant fields in the parking data (parking_measurement_id, source, source_id, parking_id, date_modified, total_spot_number, closed_spot_number, occupied_spot_number, and available_spot_number), as well as the same metadata columns as the pre_parking_measurements and src_parking_measurements tables. The stg_parking_measurements_backup table is a backup copy of the stg_parking_measurements table.

These tables are designed to hold the parking data at different stages of the data transformation process. The pre_parking_measurements table stores the raw data downloaded from the Golemio API, while the src_parking_measurements table parses the data into individual records. The stg_parking_measurements table is the final destination of the parsed data, with all the relevant fields in separate columns. The stg_parking_measurements_backup table is a backup copy of the stg_parking_measurements table, ensuring that the data is not lost in case of any issues during the transformation process. The backup process will be described in more detail a few sections below.

6.3.2 Insert into MySQL

In the next step of data preprocessing, the downloaded data in JSON format is loaded into the pre_ table using the insert_values task in the DAG. The task executes a SQL script that inserts the JSON data into the pre_ table with the corresponding metadata columns. The jsondata variable is used to pass the downloaded data to the SQL script, and the current timestamp is used for the _sys_load_at column.

Listing 6.8: Insert values task

```
insert_values = MySqlOperator(
    sql=f"INSERT INTO pre_{def_entity} VALUES (\'0\',\'{jsondata}
         \',\' 0 \',\'"+str(datetime.now())+"\',\'0\');",
    task_id=f"insert_values_{def_entity}",
    mysql_conn_id=def_conn_id,
)
```

6.3.3 Source tables

In the context of parking data, the next step in data preprocessing is to insert data into source tables. The `src_table` task in the DAG executes a SQL script that parses the data from the `pre_parking_measurements` table into individual records and inserts them into the `src_parking_measurements` table. The SQL script uses the JSON_TABLE function to extract the individual records from the `jdoc` column of the `pre_parking_measurements` table and stores them in the `jdoc` column of the `src_parking_measurements` table. The `i` column represents the index of a particular parking measurement in the original JSON array.

Listing 6.9: Source table task

```
src_table = MySqlOperator(
    sql=f'/{def_endpoint}/010_src.sql',
    task_id=f"src_table_{def_entity}",
    mysql_conn_id=def_conn_id,
)
```

The table has the same metadata columns as the `pre_parking_measurements` table, in addition to the `i` column and the `jdoc` column. The `i` column is used to associate the individual records with their respective positions in the original JSON array, while the `jdoc` column stores the individual records in JSON format.

Listing 6.10: Source table SQL scripts

```
INSERT IGNORE INTO src_parking_measurements
(
SELECT
    _sys_record_id,
    _sys_load_id,
    _sys_load_at,
    _sys_is_deleted,
    r.*
FROM
    pre_parking_measurements,
    JSON_TABLE(
        jdoc,
        '$[*]'
        COLUMNS (
            i FOR ORDINALITY,
```

```
            jdoc JSON PATH '$[0]'
            )
    ) AS r
);

TRUNCATE pre_parking_measurements
```

The creation of the `src_parking_measurements` table is an important step in the data preprocessing process, as it allows for more efficient and effective querying and analysis of the parking data. Finally, the `pre_parking_measurements` table is truncated to remove the raw data that has already been parsed and inserted into the `src_parking_measurementss` table. This ensures that the data is not duplicated during the subsequent stages of transformation.

6.3.4 Stage tables

In the data preprocessing workflow, the final stage is to transform the data from the source table into the stage table, where it will be in a format suitable for further analysis and modeling. The `stg_table` task in the DAG executes a SQL script that takes the data from the `src_parking_measurements` table and transforms it into the `stg_parking_measurements` table.

Listing 6.11: Stage table task

```
stg_table = MySqlOperator(
    sql=f'/{def_endpoint}/020_stg.sql',
    task_id=f"stg_table_{def_entity}",
    mysql_conn_id=def_conn_id,
)
```

The SQL script parses the JSON data in the `jdoc` column of the `src_parking_measurements` table and extracts the relevant fields, such as `source`, `source_id`, `parking_id`, `date_modified`, `total_spot_number`, `closed_spot_number`, `occupied_spot_number`, and `available_spot_number`. It then concatenates these fields to create a unique `parking_measurement_id` and inserts the data into the `stg_parking_measurements` table.

The `stg_parking_measurements` table has columns for all the relevant fields extracted from the JSON data, as well as the same metadata columns as the `pre_parking_measurements` and `src_parking_measurements` tables. It also has

a primary key on the parking_measurement_id column, which ensures that each record is unique and can be easily joined with other tables in the database.

Listing 6.12: Stage table SQL scripts

```
INSERT IGNORE INTO stg_parking_measurements
(
SELECT
    CONCAT(‘source_id‘, ’_’, ‘parking_id‘, ’_’, ‘date_modified‘)
        AS ‘parking_measurement_id‘,
    ‘source‘,
    ‘source_id‘,
    ‘parking_id‘,
    ‘date_modified‘,
    ‘total_spot_number‘,
    ‘closed_spot_number‘,
    ‘occupied_spot_number‘,
    ‘available_spot_number‘,
    ‘_sys_record_id‘,
    ‘_sys_load_id‘,
    ‘_sys_load_at‘,
    ‘_sys_is_deleted‘,
    ‘i‘
FROM (
SELECT
TRIM(BOTH ’"’ FROM JSON_EXTRACT(‘jdoc‘, ’$.source’))
    AS ‘source‘,
TRIM(BOTH ’"’ FROM JSON_EXTRACT(‘jdoc‘, ’$.source_id’))
    AS ‘source_id‘,
TRIM(BOTH ’"’ FROM JSON_EXTRACT(‘jdoc‘, ’$.parking_id’))
    AS ‘parking_id‘,
TRIM(BOTH ’"’ FROM JSON_EXTRACT(‘jdoc‘, ’$.date_modified’))
    AS ‘date_modified‘,
TRIM(BOTH ’"’ FROM JSON_EXTRACT(‘jdoc‘, ’$.total_spot_number’))
    AS ‘total_spot_number‘,
TRIM(BOTH ’"’ FROM JSON_EXTRACT(‘jdoc‘, ’$.closed_spot_number’))
    AS ‘closed_spot_number‘,
TRIM(BOTH ’"’ FROM JSON_EXTRACT(‘jdoc‘, ’$.occupied_spot_number’))
    AS ‘occupied_spot_number‘,
```

```
TRIM(BOTH '"' FROM JSON_EXTRACT(`jdoc`, '$.available_spot_number'))
    AS `available_spot_number`,
`_sys_record_id`,
`_sys_load_id`,
`_sys_load_at`,
`_sys_is_deleted`,
`i`
FROM src_parking_measurements
) a
);

TRUNCATE src_parking_measurements;
```

The creation of the `stg_parking_measurements` table marks the end of the data preprocessing stage for the parking data. The table contains the data in a format suitable for further analysis and modeling, and can be used for various applications, such as predicting parking spot availability or analyzing parking demand.

6.3.5 Output views

The final step in the data processing workflow is to create views that present the transformed data in a format that is convenient for further analysis and modeling. These views are created based on the tables generated in the previous stages of the workflow.

The SQL script executed by the `create_output_stage` task in the DAG defines three views that contain the required columns for model training. The `out_weather` view, shown in listing 6.13 below, combines the `stg_weather_forecast` view and `stg_weather_archive` tables to include all weather data for the relevant time period. It also adds an `_sys_source_table` column to indicate the source table for each record.

Listing 6.13: Output stage view for weather data

```
CREATE OR REPLACE VIEW `out_weather` AS
select
  `stg_weather_forecast`.`weather_forecast_id` AS `weather_id`,
  `stg_weather_forecast`.`time_ts` AS `time_ts`,
  `stg_weather_forecast`.`time` AS `time`,
  `stg_weather_forecast`.`precipitation` AS `precipitation`,
```

```
  `stg_weather_forecast`.`temperature` AS `temperature`,
  `stg_weather_forecast`.`snowfall` AS `snowfall`,
  `stg_weather_forecast`.`rain` AS `rain`,
  'stg_forecast' AS `_sys_source_table`
from
  `stg_weather_forecast`
where
  (
    `stg_weather_forecast`.`time_ts` > '2023-04-30'
  )
union
select
  `stg_weather_archive`.`weather_archive_id` AS `weather_id`,
  `stg_weather_archive`.`time_ts` AS `time_ts`,
  `stg_weather_archive`.`time` AS `time`,
  `stg_weather_archive`.`precipitation` AS `precipitation`,
  `stg_weather_archive`.`temperature` AS `temperature`,
  `stg_weather_archive`.`snowfall` AS `snowfall`,
  `stg_weather_archive`.`rain` AS `rain`,
  'stg_archive' AS `_sys_source_table`
from
  `stg_weather_archive`
where
  (
    `stg_weather_archive`.`time_ts` <= '2023-04-30'
  )
  ORDER BY time_ts DESC;
```

The remaining views `out_vehiclepositions` and `out_parking_measurements` were created in a similar way. The creation of these views marks the end of the data processing workflow. The transformed and aggregated data can now be used for various applications, such as predicting parking spot availability or analyzing parking demand.

The table in figure 11 is an example of the `out_weather` view created in the data processing workflow. It contains weather data for a specific time period, including the weather forecast ID, the timestamp, the time in ISO 8601 format, the precipitation, temperature, snowfall, and rain measurements, and the source table for each record

weather_id	time_ts	time	precipitation	temperature	snowfall	rain	_sys_source_table
2023-05-06T06:00_0.9_8.2	2023-05-06 06:00:00	2023-05-06T06:00	1.0	8.2	0.0	0.9	stg_forecast
2023-05-06T05:00_0.6_8.7	2023-05-06 05:00:00	2023-05-06T05:00	0.6	8.7	0.0	0.6	stg_forecast
2023-05-06T04:00_1.2_9.6	2023-05-06 04:00:00	2023-05-06T04:00	1.2	9.6	0.0	1.2	stg_forecast
2023-05-06T03:00_5.1_9.9	2023-05-06 03:00:00	2023-05-06T03:00	5.1	9.9	0.0	5.1	stg_forecast
2023-05-06T02:00_1.5_10.7	2023-05-06 02:00:00	2023-05-06T02:00	1.5	10.7	0.0	1.5	stg_forecast
2023-05-06T01:00_1.0_11.7	2023-05-06 01:00:00	2023-05-06T01:00	1.0	11.7	0.0	1.0	stg_forecast
2023-05-06T00:00_0.1_12.7	2023-05-06 00:00:00	2023-05-06T00:00	0.1	12.7	0.0	0.1	stg_forecast
2023-05-05T23:00_0.2_13.6	2023-05-05 23:00:00	2023-05-05T23:00	0.2	13.6	0.0	0.2	stg_forecast
2023-05-05T22:00_0.0_14.3	2023-05-05 22:00:00	2023-05-05T22:00	0.0	14.3	0.0	0.0	stg_forecast
2023-05-05T21:00_0.0_15.3	2023-05-05 21:00:00	2023-05-05T21:00	0.0	15.3	0.0	0.0	stg_forecast

Figure 11: Weather output view

indicated by the `_sys_source_table` column. The table has a clear and organized format with borders separating each column and rows, making it easy to read and analyze the weather data. It can be used for further analysis and modeling, such as predicting parking demand based on weather conditions.

7 Model

In this chapter, I will develop a prediction model for city traffic time series forecasting, with a particular focus on the Recurrent Neural Network with Long Short-Term Memory (RNN LSTM). This model has shown exceptional performance in capturing temporal dependencies and retaining long-term memory in time series data. By leveraging the power of RNN LSTM, I aim to accurately forecast city traffic patterns and contribute to effective traffic management strategies. Additionally, I will include two other models, namely the ARIMA model and Convolutional Neural Networks (CNN), for testing and comparison purposes. However, the RNN LSTM model holds significant importance due to its ability to handle sequential data and capture complex patterns inherent in city traffic dynamics. Each model will be explored in separate chapters, providing a comprehensive understanding of their underlying principles and methodologies. Through this research, I strive to develop an advanced prediction model that can significantly enhance city traffic forecasting capabilities and contribute to the optimization of transportation systems.

7.1 ARIMA

In this section, we explore the implementation and analysis of the ARIMA (Autoregressive Integrated Moving Average) model for city traffic time series forecasting. The ARIMA model combines autoregressive (AR), differencing (I), and moving average (MA) components to capture temporal patterns and dependencies in the data.

We delve into the stages of the ARIMA modeling process, starting with "Model Creation." Here, we outline the steps of setting up the ARIMA model, including data preprocessing, determining the model order, and configuring parameters. Next, we move to "Model Training," where we fit the ARIMA model to the training data and estimate the model parameters. Finally, we address "Model Evaluation," assessing the performance and accuracy of the trained ARIMA model using appropriate evaluation metrics.

The ARIMA model offers the advantage of incorporating historical values and previous observations, capturing the influence of past values and accounting for trend, seasonality, and errors. By leveraging these capabilities, we aim to generate reliable predictions for city traffic time series, enabling informed decision-making and effective traffic management strategies. Through this exploration, we gain valuable insights into the characteristics of city traffic data, empowering us to make meaningful forecasts and improve urban transportation systems.

7.1.1 Model creation, training and evaluation

In this subsection, we focus on the ARIMA (Autoregressive Integrated Moving Average) model for city traffic time series forecasting. The ARIMA model combines autoregressive (AR), differencing (I), and moving average (MA) components to capture temporal patterns and dependencies in the data.

The model creation phase involves developing the specific predictive models using the ARIMA approach. We utilize the Python programming language and the statsmodels library for implementation. The necessary libraries and packages, including `statsmodels.tsa.arima_model`, are imported. The city traffic data is retrieved from a MySQL database and preprocessed by converting it to a suitable format, setting appropriate indexes, and handling missing values.

Next, the ARIMA model is created by specifying the model order, which includes the autoregressive (AR) order, differencing (I) order, and moving average (MA) order. These parameters are determined based on the characteristics of the time series data, typically through analysis and experimentation.

The model training phase involves obtaining the input data from the database, preprocessing it by calculating rolling mean and standard deviation values, and resampling it to an hourly frequency. The dataset is split into standard weeks, with approximately 75% used for training and the remaining 25% for testing. The ARIMA model is implemented using the ARIMA class from the statsmodels.tsa.arima.model module. It is fit to the training data using the fit() function, which estimates the model parameters. Once trained, the model is ready for making predictions on new data. One of the predictions can be seen on Figure 12.

The model evaluation phase reveals suboptimal performance in predicting city traffic time series data. The limitations arise from the model's reliance on a limited set of input features. ARIMA models are more suitable for univariate time series forecasting, while the dataset incorporates multiple input features such as parking occupancy and weather data. This mismatch between the model's assumptions and data characteristics contributes to the poor predictive performance.

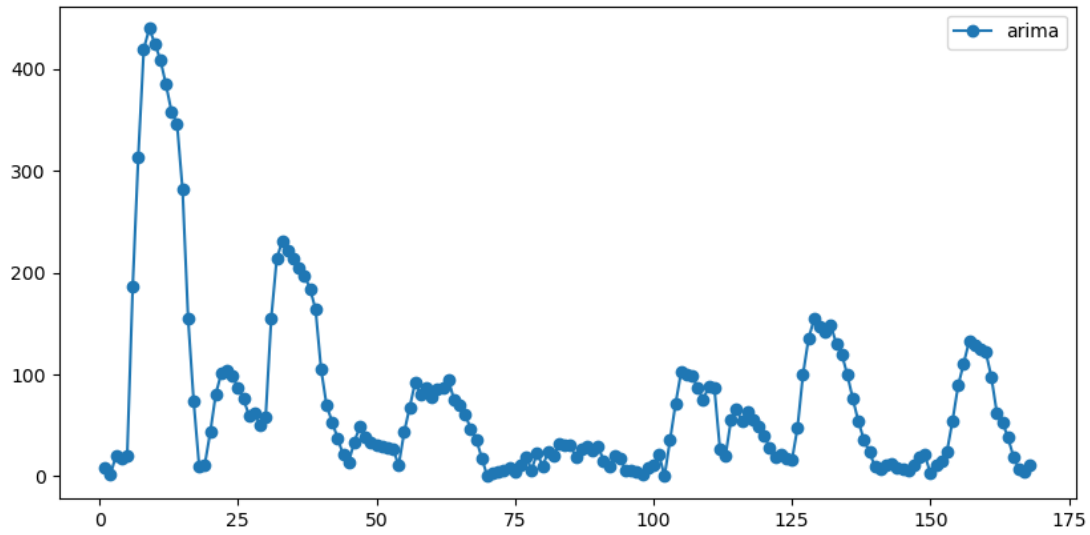

Figure 12: ARIMA Model prediction

Despite these challenges, the evaluation of the ARIMA model provides valuable insights into its limitations in city traffic time series forecasting. It highlights the need for more advanced models, such as neural networks or hybrid approaches, that can leverage the available input features and capture the inherent complexities of urban traffic dynamics. The limitations and findings emphasize the potential for further improvement and exploration in developing accurate and robust forecasting models for city traffic.

7.2 CNN

In the practical part of this study, we focus on the implementation of Convolutional Neural Networks (CNNs) for city traffic time series forecasting. The objective is to leverage the power of CNNs in capturing spatial and temporal patterns in the data and utilizing them for accurate predictions. The CNN model will be trained and evaluated using the city traffic dataset, with the goal of achieving improved forecasting performance compared to other models. The implementation process will involve data preprocessing, model configuration, training, and prediction. By applying CNNs to the city traffic forecasting problem, we aim to demonstrate the practical application and potential benefits of this deep learning approach.

7.2.1 Model creation, training and evaluation

The model training process for the Convolutional Neural Network (CNN) begins with the preparation of the dataset, including both parking and weather data. The parking data is preprocessed by applying rolling mean and standard deviation calculations to remove outliers. The data is then resampled to an hourly frequency and adjusted to set values from 10 PM to 5 AM to 0. The weather data is also resampled to an hourly frequency. The two datasets are combined based on their timestamps.

To train the CNN model, the dataset is split into training and testing sets. The training data is further divided into input-output pairs using a sliding window approach. The model architecture consists of multiple convolutional layers followed by max-pooling layers. The flattened outputs from each variable's channel are concatenated, and dense layers are added for interpretation. The model is compiled using the mean squared error (MSE) loss function and the Adam optimizer.

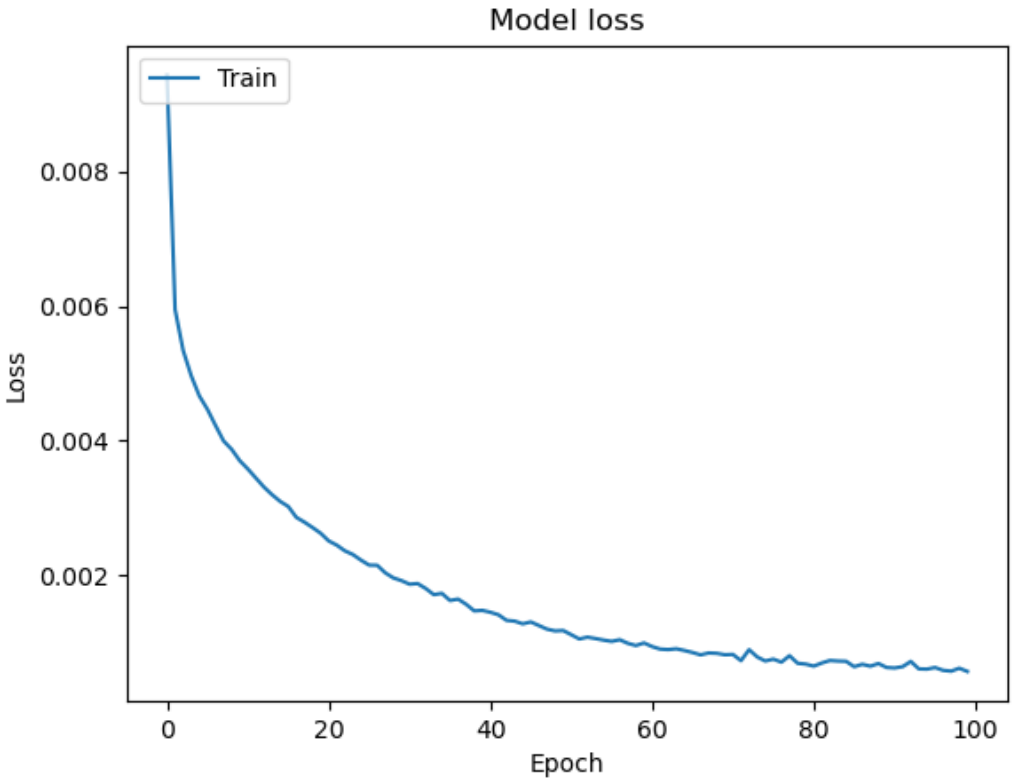

Figure 13: Loss function of CNN model

The model is trained using the training data, and the process is repeated for a specified number of epochs. The training loss and validation loss are monitored to

assess the model's performance during training. Once trained, the model is ready for predictions. The predictions made by the trained Convolutional Neural Network (CNN) model show improved performance compared to the ARIMA model. However, the predictions are not yet optimal. A Figure 14 depicting the predicted values for the parking occupancy over a specific time period can be observed. The figure highlights the model's ability to capture certain patterns and trends in the data but also illustrates the need for further refinement to achieve more accurate predictions. This indicates that there is room for improvement in the model's ability to capture the complexities of the city traffic time series data.

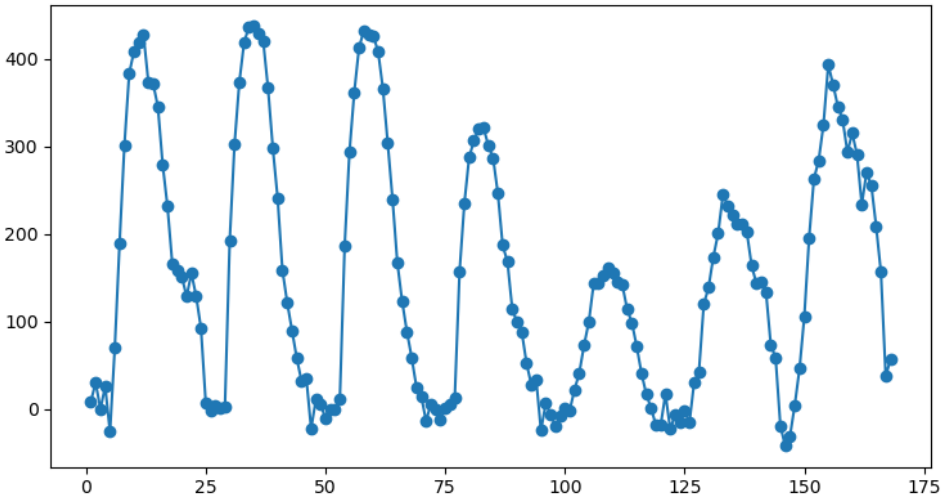

Figure 14: CNN Model prediction

For evaluation, the model is utilized to make predictions on the test data. The predictions are compared to the actual values, and the root mean squared error (RMSE) is calculated for each day. Additionally, an overall RMSE score is computed by considering all the days. The model's predictions are visualized by plotting the forecasted values against the corresponding hours. The RMSE scores provide a measure of the model's accuracy in predicting city traffic time series.

The training and evaluation process for the CNN model enables us to assess its performance in capturing the complex temporal patterns and dependencies in the city traffic data. The trained model can be further utilized for forecasting future city traffic patterns and aiding in traffic management decision-making.

7.3 RNN LSTM

In this section, I focus on the implementation and analysis of Recurrent Neural Networks (RNN) with Long Short-Term Memory (LSTM) units for city traffic time series forecasting. RNNs with LSTM units are a powerful class of deep learning models specifically designed for sequential data analysis and prediction. They excel at capturing complex temporal dependencies and have shown promising results in various time series forecasting tasks.

The use of LSTM units in RNNs addresses the vanishing gradient problem commonly encountered in traditional RNNs, allowing them to effectively capture long-term dependencies in time series data. By incorporating memory cells and gated mechanisms, LSTM units can selectively retain and update information, making them well-suited for modeling and predicting dynamic traffic patterns.

Within this section, we delve into the different stages of implementing and evaluating RNN LSTM models for city traffic forecasting. We begin with "Data preparation", continue with "Model Creation," where we outline the steps involved in setting up the RNN LSTM model architecture, including defining the number of LSTM layers, the number of units in each layer, and other architectural choices. Subsequently, we move on to "Model Training," which focuses on training the RNN LSTM model using the prepared dataset and optimizing its parameters through backpropagation and gradient descent. Lastly, we address "Model Evaluation," which involves assessing the performance and accuracy of the trained RNN LSTM model using appropriate evaluation metrics.

By leveraging the capabilities of RNN LSTM models, we aim to generate accurate and reliable predictions for city traffic time series, enabling better decision-making and more efficient traffic management strategies. Through this practical exploration, we gain valuable insights into the behavior and capabilities of RNN LSTM models in capturing the dynamics of city traffic, paving the way for improved urban transportation systems and traffic forecasting methodologies.

7.3.1 Data preprocessing

In the data preparation phase, we focus on preparing the city traffic and weather data for training the RNN LSTM model. The data is obtained from a MySQL database. We establish a connection to the database and retrieve the relevant data using SQL queries.

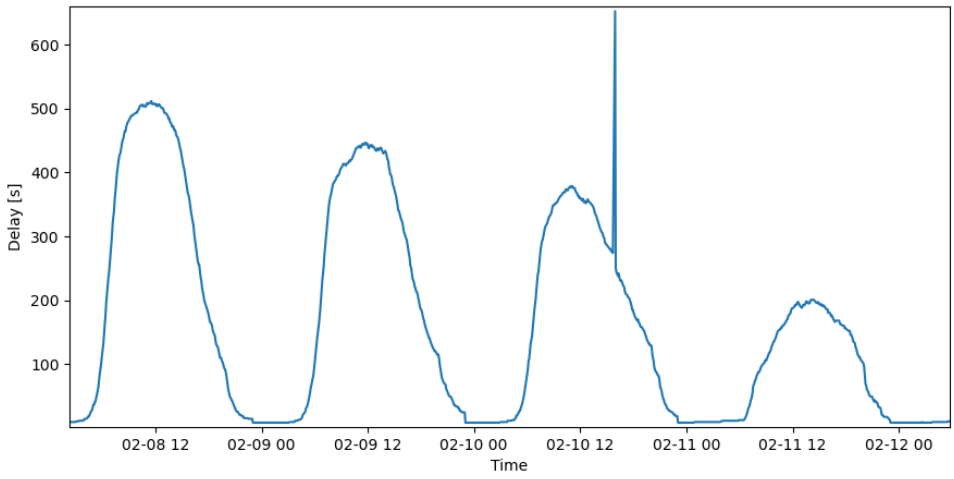

Figure 15: Visibly corrupted data on 2023-02-10

For the city traffic data, we perform several preprocessing steps to handle outliers and missing values. We calculate the rolling mean and standard deviation using a window size of 15, which helps identify and smooth out anomalies in the data. Any values that deviate significantly from the rolling mean are considered outliers and are removed from the dataset.

Next, we resample the city traffic data to an hourly frequency using linear interpolation. This ensures that the data is consistently represented at regular time intervals, facilitating analysis and modeling. Additionally, we set the occupied spot numbers to zero during the nighttime hours (from 10 PM to 5 AM) when the parking lots are closed or the traffic is minimal. To incorporate weather data into the analysis, we preprocess the weather data by resampling it also to an hourly frequency and performing linear interpolation to fill in any missing values.

Then I join the city traffic and weather data based on their timestamps, creating a combined dataset that includes both sets of information. This dataset will be used for training the RNN LSTM model to predict city traffic time series. I am performing these operations in this step using python because similar calculations directly in the database using SQL would be very difficult.

Finally I applied the MinMaxScaler function to normalize the selected columns of the parking and weather data before training the LSTM model. By normalizing the data, I ensured that the features were brought to a consistent range and placed on a similar scale. This preprocessing step was crucial for the LSTM model to learn effectively, as it prevented features with larger values from overshadowing those with smaller values during the training process. Normalization also improved the

convergence and overall performance of the model by creating a more balanced representation of the data. The MinMaxScaler function from the Scikit-learn library played a pivotal role in achieving this normalization, as it computed the minimum and maximum values of the selected columns and transformed the data to fall within the range of 0 to 1.

By performing these data preparation steps, we ensure that the input data is in a suitable format for training the RNN LSTM model. The prepared dataset captures the temporal dynamics of city traffic and incorporates relevant weather information, setting the foundation for accurate and informative predictions.

7.3.2 Model training

The `to_supervised` function converts the input data into a supervised learning format. It takes the training data, the number of input steps, and the number of output steps as parameters. The function flattens the training data and iterates over it, defining the input and output sequences based on the specified input and output lengths. It ensures that there is enough data for each instance by checking if the end index falls within the data range. The input and output sequences are appended to separate lists, and the function returns the input and output arrays. This conversion is necessary for training the LSTM model, as it requires input-output pairs to learn the temporal dependencies in the data.

Listing 7.1: to_supervised function

```
def to_supervised(train, n_input, n_out=24):
    # flatten data
    data = train.reshape((train.shape[0]*train.shape[1],
                          train.shape[2]))
    X, y = list(), list()
    in_start = 0
    # step over the entire history one time step at a time
    for _ in range(len(data)):
        # define the end of the input sequence
        in_end = in_start + n_input
        out_end = in_end + n_out
        # ensure we have enough data for this instance
        if out_end < len(data):
            x_input = data[in_start:in_end, 0]
            x_input = x_input.reshape((len(x_input), 1))
```

```
            X.append(data[in_start:in_end, :])
            y.append(data[in_end:out_end, 0])
        # move along one time step
        in_start += 1
    return array(X), array(y)
```

The `build_model`, see code 7.2, function is responsible for constructing and training the LSTM model. It takes the training data and the number of input steps as parameters. First, the function prepares the data by splitting it into training and validation sets. It then calls the `to_supervised` function to convert the training data into input-output pairs suitable for LSTM training. The function defines the model architecture, including the LSTM layer with 200 units and a `tanh` activation function. Additional layers such as a dense layer with 100 units are added to capture complex patterns in the data. The model is compiled with the mean squared error (MSE) loss function and the Adam optimizer. The model is trained using the training data, with validation performed using the validation data. The training history is stored, and the trained model is saved for future use.

Listing 7.2: build_model function

```
def build_model(train, n_input):
    # prepare data
    train_size = int(len(train) * 0.8)
    train_data = train[:train_size]
    val_data = train[train_size:]
    train_x, train_y = to_supervised(train_data, n_input)
    val_x, val_y = to_supervised(val_data, n_input)
    # define parameters
    verbose, epochs, batch_size = 1, 100, 24
    n_timesteps, n_features, n_outputs = train_x.shape[1],
           train_x.shape[2], train_y.shape[1]
    # reshape output into [samples, timesteps, features]
    train_y = train_y.reshape((train_y.shape[0], train_y.shape[1], 1))
    optimizer = Adam(learning_rate=0.001)
    # define model
    model = Sequential()
    model.add(LSTM(200, activation='tanh',
             input_shape=(n_timesteps, n_features)))
    model.add(Dense(100, activation='relu'))
```

```
    model.add(Dense(n_outputs))
    model.compile(loss='mse', optimizer=optimizer)
    # fit network
    history = model.fit(train_x, train_y, validation_data=
        (val_x, val_y), epochs=epochs,
        batch_size=batch_size, verbose=verbose)
    # Save the model
    model.save(f'model_{table}_{parking_id}.h5')
    return model
```

The plot of the loss function and validation loss function, see figure 16, provides valuable insights into the training process and model performance. When examining the plot, it is desirable to see both the loss and validation loss decreasing steadily over the epochs. This indicates that the model is effectively learning from the training data and generalizing well to unseen data. Ideally, the loss and validation loss should converge and reach a plateau, suggesting that the model has achieved a good balance between underfitting and overfitting. A smooth and gradual decrease in both curves indicates a successful training process, while a significant gap between the two curves may indicate overfitting. By monitoring the loss plot, it is possible to assess the model's learning progress and make adjustments if necessary to optimize its performance.

Once the `build_model` function completes training, the model is ready to be used for predictions. The trained LSTM model has learned the patterns and relationships within the training data, enabling it to make predictions based on new input sequences. With the model saved, it can be loaded and used to generate forecasts for future time steps.

The script is designed to run every Friday evening as part of a Directed Acyclic Graph (DAG) in Apache Airflow. This ensures regular and automated execution of the model training workflow. The script utilizes data from the previous week along with the entire historical dataset to train the model. By incorporating both recent and past data, the model can capture temporal patterns and make accurate predictions.

7.3.2.1 Grid Search

Grid search is a technique used to systematically explore different combinations of hyperparameters in order to identify the best set of hyperparameters for a

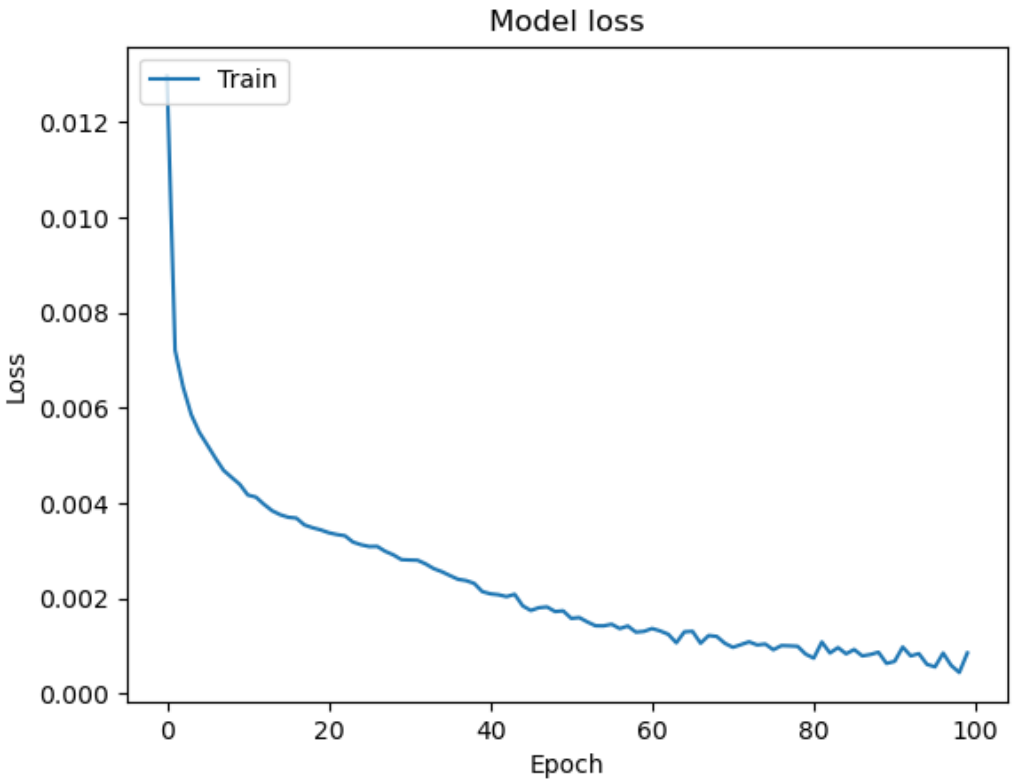

Figure 16: Loss and Validation Loss functions

machine learning model. It involves creating a grid of possible parameter values and evaluating the model's performance for each combination of values.

In the context of model training, grid search is beneficial because it allows us to tune the hyperparameters of the model to find the optimal configuration that yields the best performance. By systematically trying out different combinations of hyperparameters, we can identify the set of values that results in the highest accuracy or lowest error for our specific problem.

I implemented grid search to optimize the hyperparameters of the LSTM recurrent network model for time series forecasting. The hyperparameters that were explored include the number of LSTM units, the number of dense units, the number of epochs, and the batch size. I used the GridSearchCV class from scikit-learn to perform the grid search, which exhaustively searches through the specified parameter grid and evaluates the model's performance using cross-validation.

Listing 7.3: Model creation using Grid Search

```
# define the model
def create_model(n_timesteps, n_features, n_outputs, lstm_units=200,
```

```
            dense_units=100):
    model = Sequential()
    model.add(LSTM(lstm_units, activation='tanh',
            input_shape=(n_timesteps, n_features)))
    model.add(Dense(dense_units, activation='relu'))
    model.add(Dense(n_outputs))
    model.compile(loss='mse', optimizer='adam')
    return model

# train the model
def build_model(train, n_input):
    # prepare data
    train_x, train_y = to_supervised(train, n_input)
    n_timesteps, n_features, n_outputs = train_x.shape[1],
           train_x.shape[2], train_y.shape[1]
    # define parameters
    verbose = 1
    # create KerasRegressor
    model = KerasRegressor(build_fn=create_model,
           n_timesteps=n_timesteps, n_features=n_features,
           n_outputs=n_outputs, verbose=verbose)
    # define grid search parameters
    param_grid = {'lstm_units': [200,400], 'dense_units':
                [50,100,200], 'epochs': [100,150],
                'batch_size': [12, 24, 48,96]}
    # create grid search
    grid = GridSearchCV(estimator=model, param_grid=param_grid,
                    n_jobs=-1, cv=3)
    # fit grid search
    grid_result = grid.fit(train_x, train_y)
    # get best model
    best_model = grid_result.best_estimator_.model
    # fit network
    history = best_model.fit(train_x, train_y,
           epochs=grid_result.best_params_['epochs'],
           batch_size=grid_result.best_params_['batch_size'],
           verbose=1)
```

```
    return best_model
```

After running the grid search, I obtained the best combination of hyperparameters that resulted in the highest accuracy. The best model was then trained using these optimal hyperparameters. To visualize the training process, I plotted the training and validation loss values at each epoch.

By employing grid search, I was able to fine-tune the LSTM model and improve its performance for time series forecasting tasks. This approach allowed me to systematically explore different hyperparameter combinations and select the optimal configuration, leading to more accurate predictions.

7.3.3 Model prediction

To begin the prediction process, I utilize a pre-trained LSTM model that has been trained on historical parking data and weather conditions. The model has learned patterns and relationships from past data, enabling it to make reliable predictions.

For each prediction, I gather the necessary data, including the parking occupancy information from the previous seven days for a specific parking lot. This data is retrieved from a MySQL database using a query that fetches the relevant information based on the parking ID. Additionally, I collect weather data for the next seven days, including factors such as temperature and precipitation. Once the data is obtained, I preprocess it the same way as training dat is. After preprocessing, I merge the parking occupancy data with the weather data based on their timestamps. This creates a unified dataset that includes features such as occupancy, temperature, precipitation, and day-of-week indicators.

Listing 7.4: forecast function

```
# make a forecast
def forecast(model, history, n_input_day):
    # flatten data
    data = array(history)
    data = data.reshape((data.shape[0]*data.shape[1], data.shape[2]))
    input_x = data[:n_input_day, :]
    # reshape into [1, n_input, n]
    input_x = input_x.reshape((1, input_x.shape[0], input_x.shape[1]))
    # forecast the next week
    yhat = model.predict(input_x, verbose=0)
    # we only want the vector forecast
```

```
    yhat = yhat[0]
    return yhat
```

Using the loaded LSTM model, I make predictions by feeding the input data into the model. The model considers the historical parking occupancy and weather conditions to forecast the occupancy for the next time step. By iteratively predicting each time step, I generate a sequence of future occupancy values. To ensure the predictions are meaningful, I perform inverse normalization on the predicted values using a MinMaxScaler. This step restores the predictions to their original scale, allowing for easy interpretation.

To visualize the predicted occupancy patterns on figure 17, I generate plots showing the predicted occupancy values for each hour of the day. These plots provide a clear understanding of the forecasted occupancy trends.

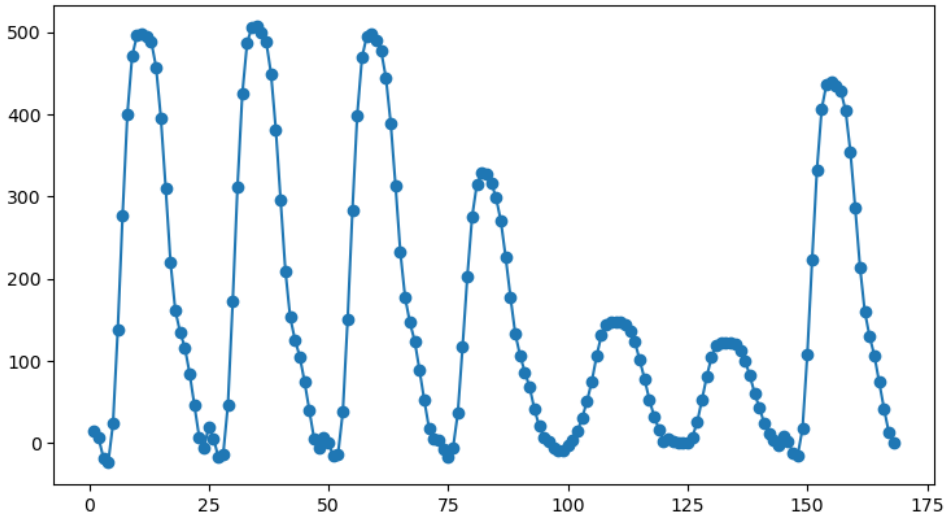

Figure 17: 7 days prediction of occupied parking spots

Finally, I save the predictions to a CSV file, including the table name, parking ID, and current date. This file serves as a record of the predicted occupancy values and will be used by web application in the next chapter. Through the use of the LSTM model and the incorporation of historical parking data and weather conditions, the Model Prediction subsection enables accurate forecasting of parking occupancy, facilitating effective management and planning of parking resources. Although I have focused on forecasting parking data in this chapter, it is important to note that the same principles and methodologies have been applied to the

prediction of tram delays. The parking data was only used to describe and explain the principles.

8 Web Application

In the Web Application chapter I focus on visualizing the predicted traffic data on a web page. This interactive web application allows users to explore and understand the forecasted traffic patterns in the city.

8.1 Pages

The web application is built using Streamlit, a Python library designed for creating data-driven web applications. The page layout is configured to provide an optimal user experience, with a wide layout, a captivating title, and an appropriate description. The web application is divided into four distinct pages, each serving a specific purpose and providing valuable insights into the predicted traffic patterns. In the following subsections, I will explain the functionality and features of each page in detail.

8.1.1 Overview

The main feature of the web application is the visualization of percentile predictions for the city's traffic density over the next seven days, see figure 18. The data used for visualization is loaded from a CSV file containing the predicted traffic values. To optimize data loading and improve performance, caching techniques are applied to store the data in memory.

Upon loading the data, it undergoes preprocessing steps. The timestamp column is converted to the appropriate data type, and the data is indexed by the timestamp for easier manipulation. The data is then grouped by date, and the maximum traffic value for each date is determined.

The visualization consists of two components. First, a set of metric cards is displayed, representing the maximum traffic values for each day. The traffic values are transformed into percentile values, indicating the level of traffic density. Descriptive labels are assigned to each percentile range, such as "Very light traffic,"

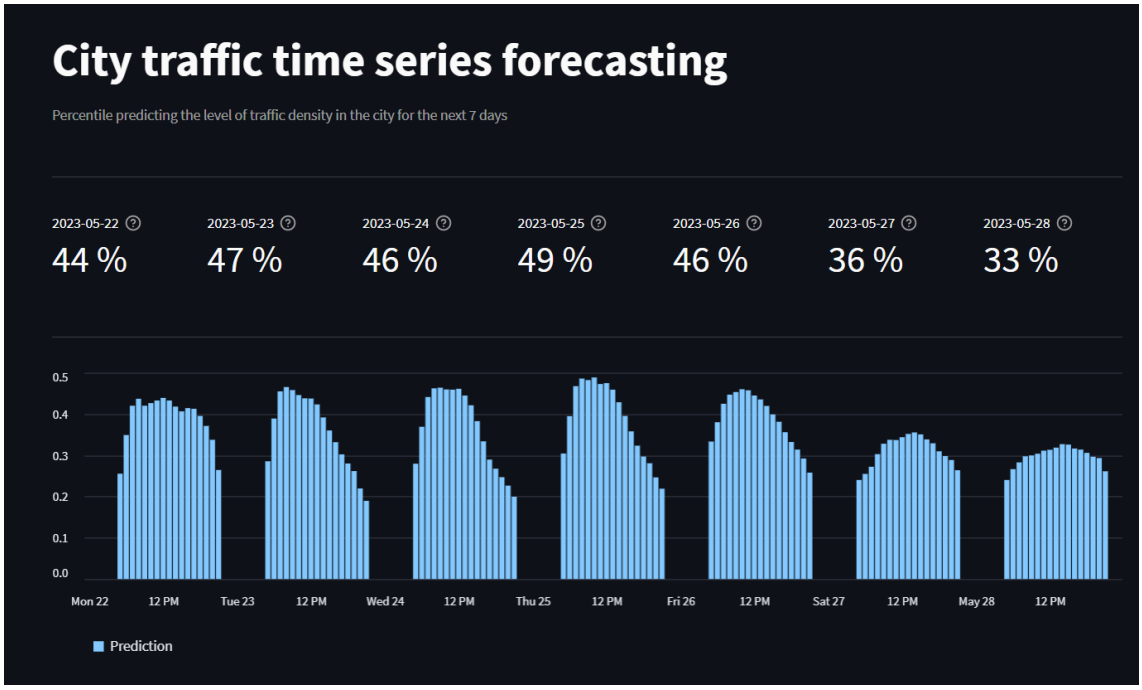

Figure 18: 7 days forecasting of traffic density

"Light traffic," "Regular traffic," "Heavy traffic," and "Very heavy traffic." This allows users to quickly interpret the predicted traffic conditions for each day.

In addition to the metric cards, a bar chart is presented, visualizing the predicted traffic values over time. The height of the bars represents the traffic intensity, providing a visual comparison of traffic patterns across different days.

Figure 19: Description of the different levels of traffic intensity

To further assist users in understanding the different levels of traffic intensity, a help section is included in figure 19. This section displays five metrics, ranging from "0% - 20%" to "80% - 100%," each accompanied by a corresponding description of traffic intensity.

8.1.2 Parking Predictions

The next page of the web application focuses on forecasting the number of available parking spaces in P+R parking houses. It provides valuable insights into

the future availability of parking spots. The page consists of an interactive map powered by PyDeck, allowing users to visualize the occupancy levels in real-time, figure 20.

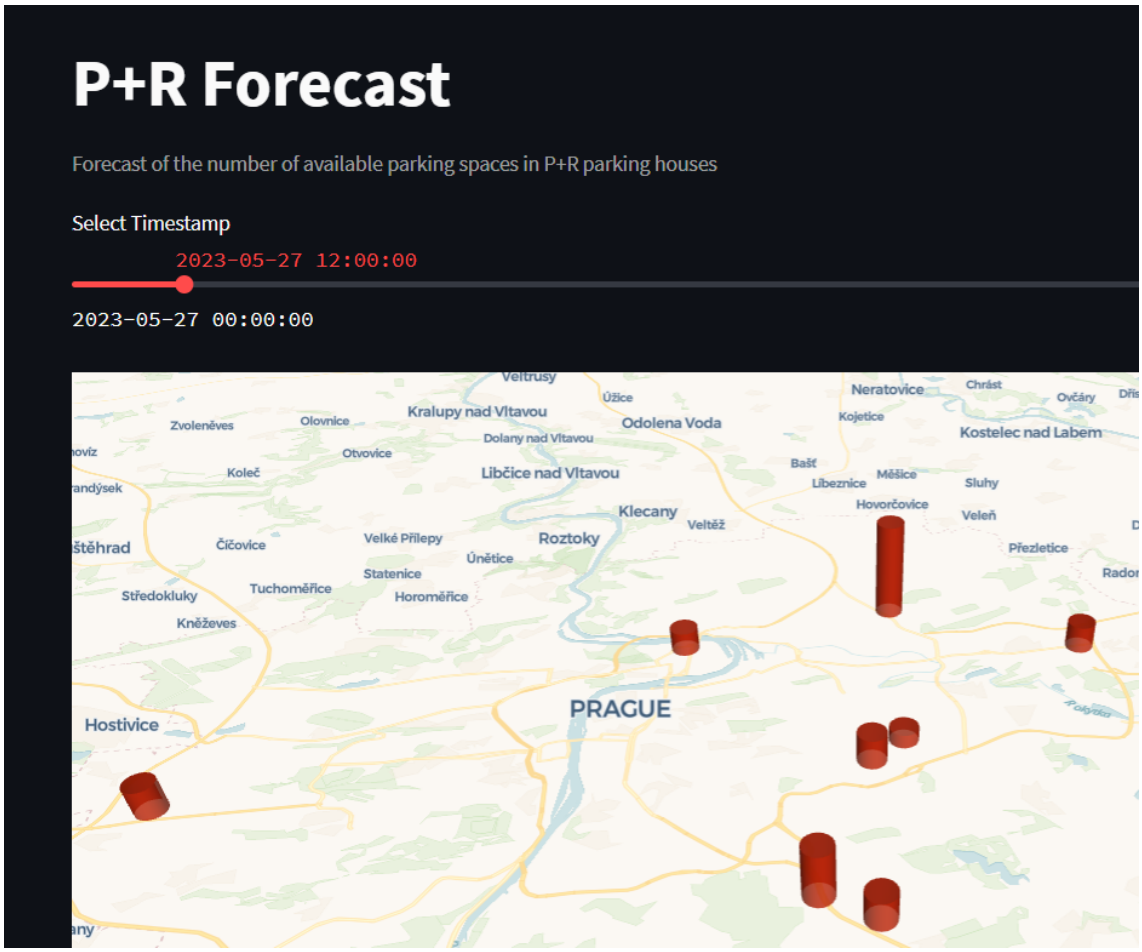

Figure 20: Map with parking occupancy level forecasts

Using the available data, the application generates predictions for the occupancy levels at different timestamps. Users can adjust the timestamp using a slider to explore the forecasts for specific time intervals. The map displays the parking houses as individual columns, with the height representing the predicted occupancy level. A tooltip, on figure 21, provides detailed information about each P+R parking house, including the number of occupied spots.

By providing visual representations of parking availability, this page offers users a comprehensive view of the predicted parking conditions. They can make informed decisions based on these forecasts, ensuring a hassle-free parking experience.

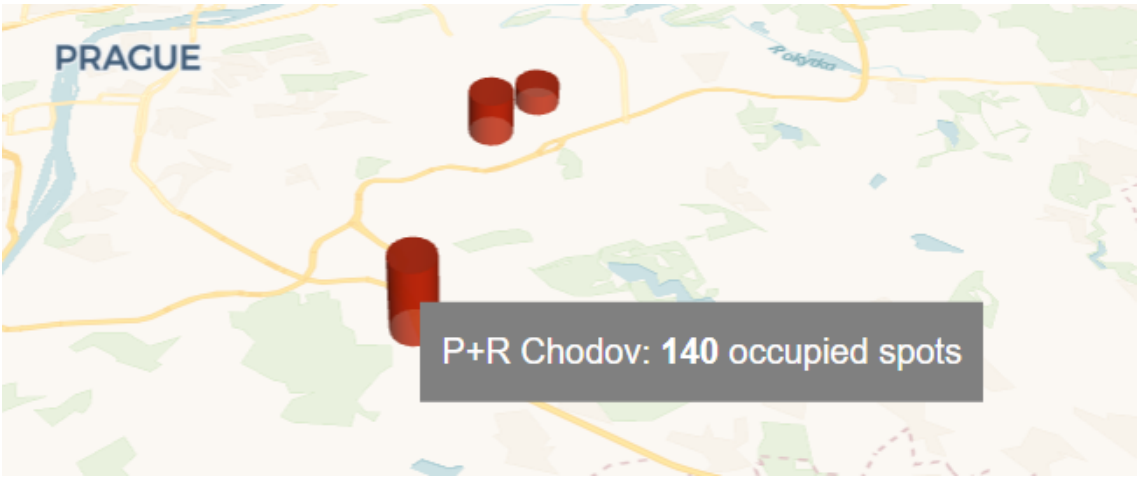

Figure 21: Tooltip with detailed information

8.1.3 Parking Predictions Details

The next page of the web application focuses on providing detailed forecasts for the number of available parking spaces in various P+R parking houses. The page consists of separate sections for each parking house, allowing users to explore the forecasts and raw data for each location.

For each P+R parking house, the application displays a bar chart showing the forecasted occupancy levels for the next 7 days, figure 22. Users can gain insights into the expected availability of parking spaces and plan their trips accordingly. Additionally, a checkbox is provided to view the raw data associated with each parking house, providing transparency and allowing users to verify the accuracy of the forecasts.

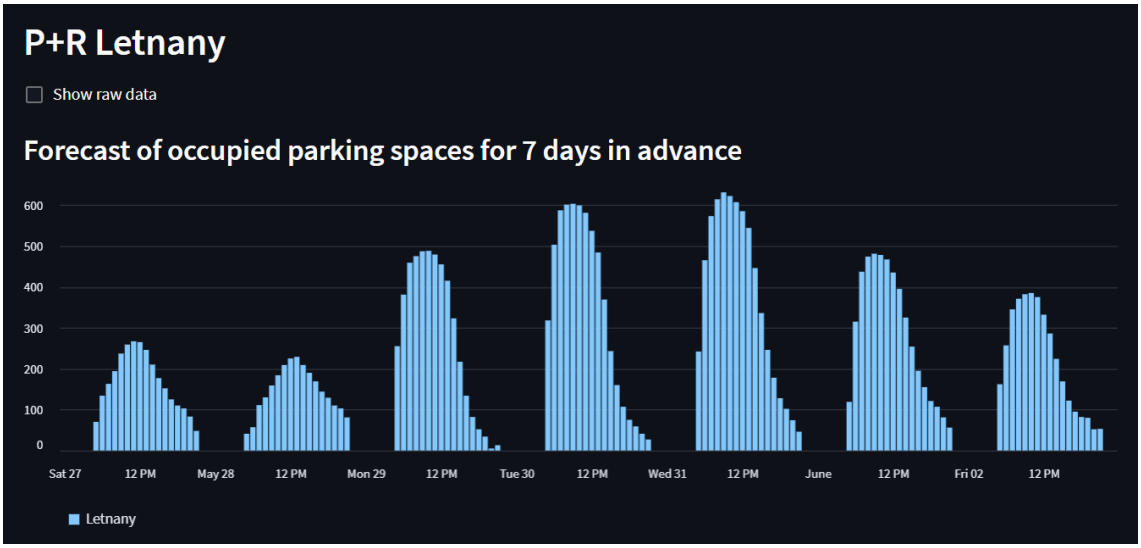

Figure 22: 7 days forecast of occupied parking spaces

By providing specific forecasts for multiple P+R parking houses, this page enables users to make informed decisions about their parking choices based on the predicted occupancy levels. It enhances the user experience by offering comprehensive and detailed information about parking availability at various locations.

8.1.4 Public Transport Predictions

The following and last page of the web application focuses on providing detailed forecasts for tram delays for specific tram numbers. Users can select a tram number from the available options and view the forecasted delays in seconds for the next 7 days, figure 23.

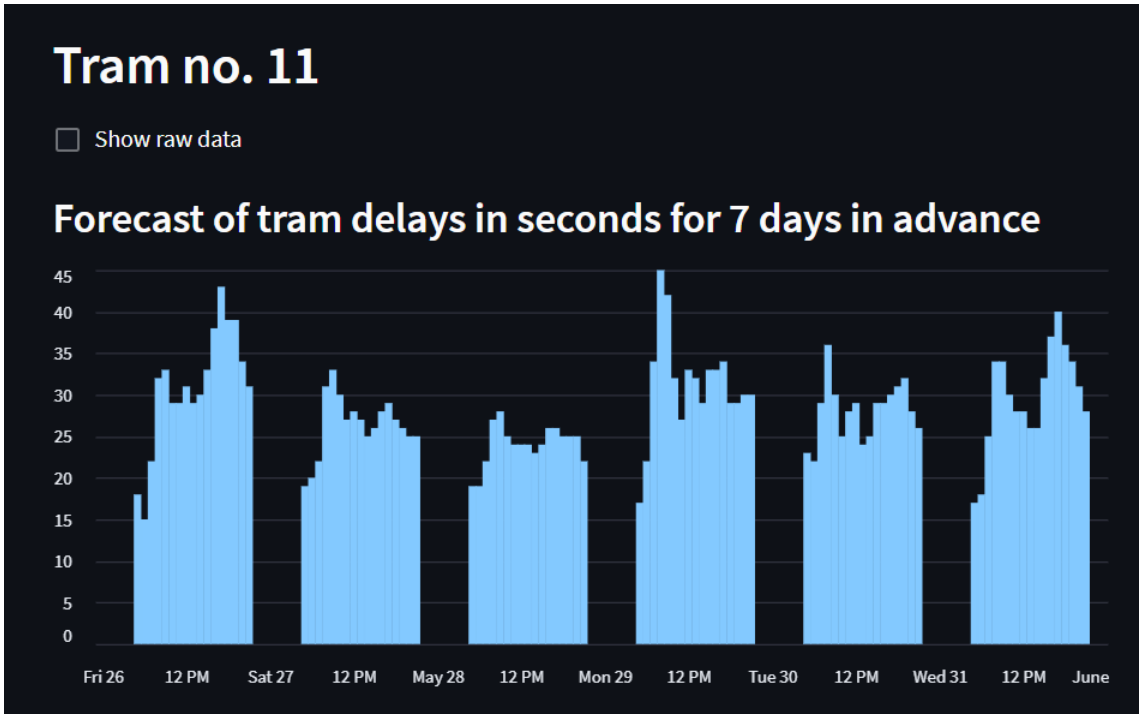

Figure 23: 7 day forecast of tram 11 delay

The page is divided into separate sections for each tram number, allowing users to easily navigate and explore the forecasts for different trams. For each tram number, a bar chart is displayed, illustrating the forecasted delays over time. Users can gain insights into the expected delays and plan their journeys accordingly.

Additionally, the application provides a checkbox to display the raw data associated with each tram number. This allows users to verify the accuracy of the forecasts and gain a deeper understanding of the underlying data.

9 Conclusion

In conclusion, the aim of this thesis was to develop a comprehensive urban mobility prediction system that incorporates various aspects such as urban traffic and parking occupancy and tram delays. The aim was also to provide the most accurate and timely forecasts that would support the decision making of both the key people who manage the operation of urban transport and the ordinary people who flow into the city every day.

Several key findings and achievements were made during the research and development process. Firstly, a robust data collection and pre-processing system has been developed that uses data from a variety of sources including public transport vehicle locations and information, parking sensors and weather data. This provided a rich and diverse data set for training and evaluating forecasting models.

Second, state-of-the-art machine learning models, specifically LSTM-based models, were implemented to predict urban traffic, parking occupancy and tram delays. These models have proven their effectiveness in capturing temporal dependencies and generating accurate forecasts. The integration of external factors such as weather data further improved the accuracy of the forecasts and provided valuable insights into the relationships between different urban mobility variables.

I would describe the results of the models as successful, as clear patterning can be observed in the models, where for example it is possible to tell from the parking occupancy prediction whether it is a weekday or a weekend. The prediction also shows a clear trend that occurs during the day, where the car park is less occupied in the morning and evening hours and, conversely, peaks during midday. Another conclusion is that, as predicted, the recurrent LSTM neural network achieved better results than the classical ARIMA statistical model and the convolutional neural network.

Furthermore, a web application was developed to visualise the forecast data and facilitate interaction with users. The application provides an intuitive interface for exploring and analyzing the predictions, allowing users to make informed decisions and plan their journeys accordingly. The inclusion of raw data display and

visualisation capabilities increased the transparency and credibility of the forecast system.

This thesis has contributed to the field of urban mobility forecasting by providing a comprehensive solution that addresses multiple aspects of urban transport, offering a new way of forecasting by aggregating multiple data sources. The developed forecasting system offers valuable information on urban traffic patterns, parking availability and tram delays, allowing commuters and city planners to make informed decisions and optimize their mobility.

There is still a lot of room for further improvements and advancements. Forecasting models can be improved and fine-tuned to capture even more complex patterns and increase the accuracy of predictions. With the ever-increasing capabilities in the industry, it can also be expected that more powerful and appropriate predictive models will emerge within a few years. In addition, the incorporation of real-time data and integration with smart city infrastructure can increase the speed of response and reliability of the system.

In conclusion, this thesis lays the foundation for a data-driven intelligent urban mobility forecasting system and contributes to the ongoing efforts to create sustainable and efficient transportation networks. The research and development carried out in this thesis opens the door for future innovations and advances in urban mobility forecasting, ultimately leading to a more seamless and optimised urban transport experience for all.

List of Literature

1. TRINH, Ngoc-Phap; TRAN, Anh-Khoa N.; DO, Trong-Hop. Traffic Flow Forecasting using Multivariate Time-Series Deep Learning and Distributed Computing. 2022, pp. 665–670. Available from DOI: 10.1109/RIVF55975.2022.10013796.
2. HOU, Yue; DENG, Zhiyuan; CUI, Hanke. Short-Term Traffic Flow Prediction with Weather Conditions: Based on Deep Learning Algorithms and Data Fusion. 2021. Available from DOI: 10.1155/2021/6662959.
3. VILLARROYA, Cristian; CALAFATE, Carlos T.; ONAINDIA, Eva; CANO, Juan-Carlos; MARTINEZ, Francisco J. Neural Network-based Model for Traffic Prediction in the City of Valencia. *Procedia Computer Science*. 2022, vol. 207, pp. 552–562. ISSN 1877-0509. Available from DOI: https://doi.org/10.1016/j.procs.2022.09.110. Knowledge-Based and Intelligent Information Engineering Systems: Proceedings of the 26th International Conference KES2022.
4. LIANG, Zilu; WAKAHARA, Yasushi. City traffic prediction based on real-time traffic information for Intelligent Transport Systems. 2013, pp. 378–383. Available from DOI: 10.1109/ITST.2013.6685576.
5. HASTIE, Trevor; TIBSHIRANI, Robert; FRIEDMAN, Jerome. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2009.
6. AULD, Travis. Data: A Fundamental Building Block for Predictive Models. *SSRN Electronic Journal*. 2019, pp. 1–23. Available from DOI: 10.2139/ssrn.3355024.
7. GOODFELLOW, Ian; BENGIO, Yoshua; COURVILLE, Aaron. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.
8. SIVIA, D. S. *Data analysis: a Bayesian tutorial*. Oxford University Press, 2006.
9. GOLEMIO.CZ. *Prague Public API*. City of Prague, 2021. Available also from: https://api.golemio.cz/v2/pid/docs/openapi/.

10. OPEN-METEO. *Open-Meteo API Documentation* [https://open-meteo.com/en/docs]. 2023. [Online; accessed 04-April-2023].

11. TIAN, Hui; HE, Qing; LIU, Wen; WANG, Wei; ZHANG, Yingjie; ZOU, Le; QIN, Rongjun; WU, Jianjun; GUO, Jianping. Importance of Weather Variables for Time Series Forecasting of Traffic Flow: A Case Study of Melbourne, Australia. *Journal of Advanced Transportation*. 2021, vol. 2021. Available from DOI: 10.1155/2021/6662959.

12. PYTHON SOFTWARE FOUNDATION. *Python Software Foundation*. Python Software Foundation, 2023. Available also from: https://www.python.org/.

13. GOOGLE CLOUD. *Google Cloud Platform*. 2023. Available also from: %5Curl%7Bhttps://cloud.google.com/%7D. [Online; accessed 04-April-2023].

14. AIRFLOW, Apache. *Apache Airflow*. 2023. Available also from: %5Curl%7Bhttps://airflow.apache.org/%7D. [Online; accessed 04-April-2023].

15. THULASIRAMAN, K.; SWAMY, M. N. S. 5.7 Acyclic Directed Graphs. In: *Graphs: Theory and Algorithms*. John Wiley and Sons, 1992, p. 118. ISBN 978-0-471-51356-8.

16. DATE, C. J. SQL and Relational Theory. 2004.

17. ELMASRI, R.; NAVATHE, S. B. Fundamentals of database systems. 2016.

18. CHAMBERLIN, D. D.; BOYCE, R. F. SEQUEL 2: A unified approach to data definition, manipulation, and control. *IBM research report*. 1974, vol. 1974, no. 1.

19. WIDENIUS, Michael. MySQL: An Open Source Relational Database Management System. *ACM SIGMOD Record*. 2004, vol. 33, no. 1, pp. 23–33.

20. GAMAUF, Stefan; SCHNEIDER, Markus. MySQL security: a practical guide for DBAs. *Security and Communication Networks*. 2011, vol. 4, no. 4, pp. 365–382.

21. BOX, George EP; JENKINS, Gwilym M; REINSEL, Gregory C; LJUNG, GM. *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.

22. MURPHY, Kevin P. *Machine learning: a probabilistic perspective*. MIT press, 2012.

23. GOODFELLOW, Ian; BENGIO, Yoshua; COURVILLE, Aaron. *Deep learning*. MIT press, 2016.

24. SHUMWAY, Robert; STOFFER, David. *Time Series Analysis and Its Applications: With R Examples*. Springer, 2017.

25. BROCKWELL, P.J.; DAVIS, R.A. *Introduction to time series and forecasting*. Springer, 2016.

26. MAKRIDAKIS, Spyros; SPILIOTIS, Evangelos; ASSIMAKOPOULOS, Vassilios. Statistical and machine learning forecasting methods: Concerns and ways forward. *PloS one*. 2018, vol. 13, no. 3, e0194889.

27. HYNDMAN, Rob J; ATHANASOPOULOS, George. Forecasting: principles and practice. *OTexts*. 2018.

28. QUINLAN, J Ross. Induction of decision trees. *Machine learning*. 1986, vol. 1, no. 1, pp. 81–106.

29. BREIMAN, Leo. *Classification and Regression Trees*. 1st. Routledge, 1984. Available from DOI: 10.1201/9781315139470.

30. BREIMAN, Leo. Random forests. *Machine learning*. 2001, vol. 45, no. 1, pp. 5–32.

31. BIAU, Gerard. Random forests: recent developments. *Mathematics and computers in simulation*. 2016, vol. 130.

32. CHANG, Chih-Chung; LIN, Chih-Jen. Libsvm: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)*. 2011, vol. 2, no. 3, p. 27.

33. HUANG, Shupei; AN, Haizhong; GAO, Xiangyun. Forecasting crude oil market volatility: further evidence with jumps. *Energy Economics*. 2014, vol. 44, pp. 169–176.

34. GUO, Xiaoxia; WANG, Jianzhou; ZHU, Yongwei. Short-term load forecasting using extreme gradient boosting: A case study of residential electricity consumption. *Energy*. 2017, vol. 133, pp. 366–374.

35. FENG, Xia; GUO, Shuyu; ZHANG, Liwei. Predicting stock prices using gradient boosting machine and support vector regression. *Journal of Risk and Financial Management*. 2018, vol. 11, no. 4, p. 64.

36. SINGH, Mandeep; SINGH, Sukhpreet; KAUR, Harwinder. Time series prediction using deep learning: A comparative study. *International Journal of Innovative Technology and Exploring Engineering (IJITEE)*. 2019, vol. 8, no. 11S, pp. 200–205.

37. YU, Fei; HUANG, Zhenyu; WANG, Shaohua. Long-term load forecasting using a deep learning framework. *IEEE Transactions on Smart Grid*. 2017, vol. 8, no. 1, pp. 413–421.

38. LIU, Yang; WANG, Wei; GHADIMI, Noradin. Electricity load forecasting by an improved forecast engine for building level consumers. *Energy*. 2017, vol. 139, pp. 18–30. ISSN 0360-5442. Available from DOI: https://doi.org/10.1016/j.energy.2017.07.150.

39. CORANI, Giorgio. Air quality prediction in Milan: feed-forward neural networks, pruned neural networks and lazy learning. *Ecological Modelling*. 2005, vol. 185, no. 2, pp. 513–529. ISSN 0304-3800. Available from DOI: https://doi.org/10.1016/j.ecolmodel.2005.01.008.

40. SURESH, Vishnu; JANIK, Przemyslaw; REZMER, Jacek; LEONOWICZ, Zbigniew. Forecasting Solar PV Output Using Convolutional Neural Networks with a Sliding Window Algorithm. *Energies*. 2020, vol. 13, p. 723. Available from DOI: 10.3390/en13030723.

41. TSANTEKIDIS, Avraam; PASSALIS, Nikolaos; TEFAS, Anastasios; KANNIAINEN, Juho; GABBOUJ, Moncef; IOSIFIDIS, Alexandros. Forecasting Stock Prices from the Limit Order Book Using Convolutional Neural Networks. 2017, vol. 01, pp. 7–12. Available from DOI: 10.1109/CBI.2017.23.

42. (USER:IXNAY), Wikimedia Commons contributors. *Compressed (left) and unfolded (right) basic recurrent neural network*. Available also from: https://en.wikipedia.org/wiki/Recurrent_neural_network#/media/File:Recurrent_neural_network_unfold.svg. Accessed on 2023/05/04.

43. (USER:IXNAY), Wikimedia Commons contributors. *A diagram for a one-unit Long Short-Term Memory (LSTM)*. Available also from: https://commons.wikimedia.org/wiki/User:Ixnay#/media/File:Long_Short-Term_Memory.svg. Accessed on 2023/05/04.

44. MA, Xiaolei; TAO, Zhimin; WANG, Yinhai; YU, Haiyang; WANG, Yunpeng. Long short-term memory neural network for traffic speed prediction using remote microwave sensor data. *Transportation Research Part C: Emerging Technologies*. 2015, vol. 54, pp. 187–197. ISSN 0968-090X. Available from DOI: https://doi.org/10.1016/j.trc.2015.03.014.

45. LI, Xiang; PENG, Ling; YAO, Xiaojing; CUI, Shaolong; YOU, Chengzeng; CHI, Tianhe. Long short-term memory neural network for air pollutant concentration predictions: Method development and evaluation. *Environmental pollution (Barking, Essex : 1987)*. 2017, vol. 231, pp. 997–1004. Available from DOI: 10.1016/j.envpol.2017.08.114.

46. KUMAR, Jitendra; GOOMER, Rimsha; SINGH, Ashutosh Kumar. Long Short Term Memory Recurrent Neural Network (LSTM-RNN) Based Workload Forecasting Model For Cloud Datacenters. *Procedia Computer Science*. 2018, vol. 125, pp. 676–682. ISSN 1877-0509. Available from DOI: https://doi.org/10.1016/j.procs.2017.12.087. The 6th International Conference on Smart Computing and Communications.

47. GERS, Felix A.; SCHRAUDOLPH, Nicol N.; SCHMIDHUBER, Jürgen. Learning Precise Timing with LSTM Recurrent Networks. *Journal of Machine Learning Research*. 2002, vol. 3, pp. 115–143. Available also from: https://www.jmlr.org/papers/volume3/gers02a/gers02a.pdf. Submitted 5/01; Revised 3/02; Published 8/02.

48. GOOGLE CLOUD PLATFORM. *Set up a MySQL server on a Compute Engine instance* [https://cloud.google.com/compute/docs/instances/sql-server/setup-mysql]. 2023. [Online; accessed May 4, 2023].

List of Figures

List of Attachments

Attachment no. 1: CD
Attachment no. 2: Git repository - https://github.com/martin-krecek/diploma

List of Software used

Python 3.10.9 + libraries
SQL
Google Cloud Platform
Apache Airflow 2.6.1
MySQL 8.0
Streamlit 1.23
Git 2.34.1
Overleaf LaTeX editor