HSLU Hochschule Luzern

Applied Information and Data Science

Data Warehouse and Data Lakes

Project Report

Traffic Lakes

Authors:
Michael Jung

Yanik Baumann

6/13/2024

1 Abstract

In this study, conducted in collaboration with the City of Zurich, we investigate the impact of weather conditions on traffic volumes and patterns. We aim to improve traffic management by integrating weather data with traffic data to produce more accurate traffic forecasts. Using the OpenWeather API and HERE traffic API, we collected weather and traffic data over a two-month period, storing and processing it in a data lake and data warehouse. Through detailed analysis and visualization using Tableau, we explored the correlation between various weather conditions and traffic behavior. Our findings indicate that extreme weather events, such as heavy snowfall and thunderstorms, significantly increase traffic congestion, while milder conditions result in smoother traffic flow. The study also reveals that traffic patterns vary significantly throughout the day and week, with higher congestion during peak commute times and on weekdays. These insights are consistent with current research and highlight the need for adaptive traffic management strategies that account for weather variations. Future research should extend the data collection period and incorporate advanced machine learning models to enhance the predictive accuracy and provide a more comprehensive understanding of the factors influencing traffic dynamics.

Content

2 Introduction

Traffic jams are a widespread source of frustration and annoyance. Simply adding lanes has proven to be ineffective in practice, as this additional capacity is quickly utilized. In addition to the immediate annoyance for those affected, congestion also leads to significant problems such as localized air pollution and loss of valuable time.

In collaboration with the City of Zurich, we want to explore a new approach to analyzing and tackling traffic problems. Our focus is on recognizing the influence of weather on traffic volumes and creating more accurate traffic forecasts based on this. By analyzing weather data and correlating it with traffic flows, we aim to gain valuable insights that go beyond the conventional view of rush-hour traffic. The aim is to lay the foundation for intelligent traffic management systems that not only reduce congestion, but also improve the quality of life for citizens.

For this, we collected weather and traffic data from two dynamic data sources. For this we used the OpenWeather API and HERE traffic API. The collected data was placed into a data lake. Within the data lake, the data is stored, structured, and processed to make it usable for further analysis. The processed data is then loaded into a data warehouse, which in this case was a database where the data was further structured using tables. From here we would perform queries to visualize the chosen data with Tableau. This enabled us to perform visual analysis and create traffic forecasts. In summary this data-driven approach allows us to precisely analyze the impact of weather on traffic activity and make well-founded predictions.

2.1 Problem Statement and Research Questions

With this project, we want to tackle the problem of unpredictable traffic volumes and investigate whether the weather has a significant influence on traffic. To this end, we have formulated the following research questions.

- How strong is the correlation between temperature and the volume of traffic?
- Do different weather events (e.g., rain, snow, extreme heat) have an influence on traffic behavior?
- How do traffic patterns vary throughout the day, week, and year?

These research questions enable us to address the current problem in a targeted manner and make a valuable contribution to research. In addition, concrete business questions for the city of Zurich can also be derived from these research questions.

2.2 Business Questions

To best offer added value for the city of Zurich with this project, the following business questions were jointly formulated based on the research questions:

- Can traffic and weather data be combined to predict potential traffic issues (e.g., accidents due to sudden downpours)?
- Which areas in Zurich experience the most significant traffic congestion during specific times of the day?
- How to traffic patterns vary on weekends versus weekdays (Saturday vs. Sunday vs Weekday)?
- Can we predict future traffic based on a given weather prediction?

2.3 Limitations

For this project, data was only collected and analyzed over a period of two months (March and April 2024). For this reason, no historical data can be used for the results. Additionally free APIs were used, therefore no advanced features such as real time traffic map overlays, historical weather data and forecasts were available.

3 State of the Art

Investigating the influence of weather on transportation activities is an increasingly relevant topic, especially in the context of urban transportation systems. Traffic jams and congestion are global problems that not only waste time and resources, but also have a significant environmental impact. Modern data warehouse and data lake technologies enable detailed analysis of large amounts of data from different sources to investigate such complex relationships. Several studies have shown that weather conditions such as precipitation, fog and snow have a significant impact on road safety. A quantitative analysis has shown that the probability of traffic accidents increases significantly in bad weather conditions. These studies often use generalized additive models to quantify the combined effects of traffic volume and meteorological parameters (Becker et al., 2022).

A study of traffic flow in urban areas has shown that weather conditions can have a significant impact on traffic flow. For example, rain and snow can reduce the average speed of vehicles and increase traffic density. This often leads to longer travel times and increased congestion. Analyses based on Traffic Performance Index (TPI) have identified patterns that vary depending on the day of the week and the weather (Zang et al., 2023).
Modern traffic forecasting systems use data from various sources, including weather APIs and traffic monitoring data. This data is stored in data lakes to make it accessible for analysis and modeling. Studies have shown that by integrating weather data into traffic models, more accurate predictions can be made, leading to more efficient traffic management strategies (Bi et al., 2023). Advanced approaches use bidirectional Long Short-Term Memory (BiLSTM) networks and Graph Neural Networks (GNN) to capture the spatiotemporal dependencies of traffic flows and weather conditions. These models have shown the ability to make accurate predictions by combining both spatial and temporal features (Alourani et al., 2023).

Studies have also investigated the impact of climate change on long-term traffic patterns. Changes in temperatures and wind speeds can affect traffic conditions, for example by altering travel times and the frequency of traffic congestion (Gratton et al., 2021). Current research and development in the field of traffic forecasting and traffic management uses advanced data engineering and data analysis techniques. Data lakes and data warehouses play a central role here. These technologies enable the storage and processing of large amounts of data from dynamic data sources such as the OpenWeather API and various traffic APIs. The integration of weather and traffic data into a data lake enables flexible and scalable data processing. This data is continuously updated and can be visualized and analyzed using tools such as Tableau. This makes it possible to recognize complex correlations between weather conditions and traffic patterns and react to them in real time. By applying machine learning algorithms to the collected data, predictive models can be developed that provide accurate forecasts of future traffic activity. These models consider various factors such as the amount of precipitation, temperature and wind speed and their influence on traffic flow and the probability of accidents (Becker et al., 2022; Alourani et al., 2023).

The insights gained from data analysis contribute to the development of intelligent traffic management systems that can reduce traffic congestion and increase road safety. Cities such as Zurich are increasingly relying on such data-driven approaches to make their transportation infrastructure more efficient and sustainable.

Therefore, the integration of weather and traffic data offers a promising opportunity to better understand and manage urban traffic. By using modern data engineering techniques, precise forecasting models can be developed that improve both the efficiency of traffic management and road safety.

4 Methodology

The following chapter describes the method chosen to answer the research and business questions. In a first step, the general technical precautions and the data sources are described. This is followed by a description of the architecture. Finally, personas are described, which are addressed with the system that has been set up.

4.1 General technical information

Zurich generally works with the services of Amazon. For this reason, we have been provided with an AWS Learners Lab, in which we have a maximum budget of 100 dollars at our disposal. The entire data lake and data warehouse environment was implemented accordingly using Amazon services. Amazon S3 was used as the data lake for storing and managing the raw data. AWS Lambda was used to contact the APIs and load the data into the data lake and the data warehouse. Amazon RDS was used as the database (PostgreSQL) for structured storage and management of the processed data.

4.2 Data Sources

Two sources were used for the project. The HERE Traffic API v7 was used for the traffic data and the OpenWeather API was used for the weather data. Both sources are discussed in more detail in the following subchapters.

4.2.1 OpenWeather API

The OpenWeather API provides a comprehensive technical interface to obtain real-time weather data as well as historical and forecast weather information. It supports various endpoints that provide different types of weather information, such as current weather data, short-term and long-term forecasts and historical weather data. The API uses the RESTful protocol and is accessible via HTTP and HTTPS. Data is provided in JSON and XML formats. An API key is required to use the API, which is available free of charge after registering with OpenWeather. The API offers several endpoints:

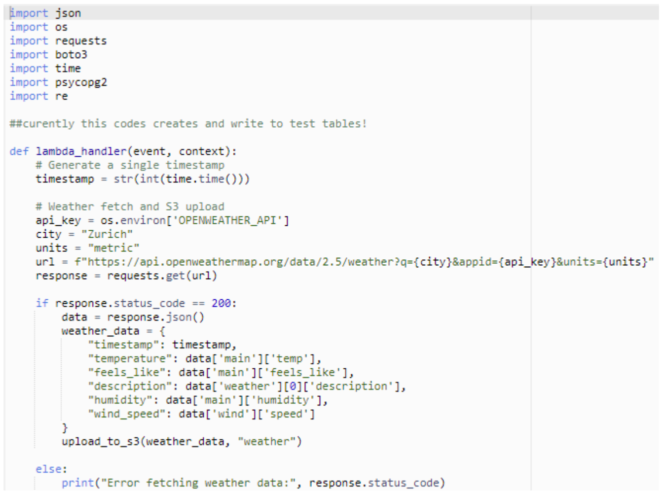
The */weather* endpoint provides current weather information for a given city or coordinates. The */forecast* endpoint provides a weather forecast in 3-hour intervals for the next 5 days, while the */forecast/daily* endpoint provides daily forecasts for up to 16 days. For historical weather data, there is the */timemachine* endpoint, which provides archived weather information for a specific location and time period.

For this project, we queried the current weather for the city of Zurich on a 20 min basis. In this query, we draw the following variables:

Variable	Description	Example
Timestamp	Unix timestamp indicating the date and time at which the weather data was collected.	1623070800
Temperature in Celsius	Current temperature in degrees Celsius.	23.4
Feels like in Celsius	Perceived temperature in degrees Celsius, based on wind speed and humidity.	22.0
Description	Description Short textual description of the current weather conditions.	"clear sky"
Humidity	Humidity in percent	56
Windspeed	Wind speed in metres per second	5.1

Table 1: Weather data

The code to fetch this data can be found in the file "lambda_fetch_store_api_data_s3_and_database.ipynb" in the Github repository linked in the annex. It can also be seen in the following screenshot:



```python
import json
import os
import requests
import boto3
import time
import psycopg2
import re

##curently this codes creates and write to test tables!

def lambda_handler(event, context):
    # Generate a single timestamp
    timestamp = str(int(time.time()))

    # Weather fetch and S3 upload
    api_key = os.environ['OPENWEATHER_API']
    city = "Zurich"
    units = "metric"
    url = f"https://api.openweathermap.org/data/2.5/weather?q={city}&appid={api_key}&units={units}"
    response = requests.get(url)

    if response.status_code == 200:
        data = response.json()
        weather_data = {
            "timestamp": timestamp,
            "temperature": data['main']['temp'],
            "feels_like": data['main']['feels_like'],
            "description": data['weather'][0]['description'],
            "humidity": data['main']['humidity'],
            "wind_speed": data['wind']['speed']
        }
        upload_to_s3(weather_data, "weather")

    else:
        print("Error fetching weather data:", response.status_code)
```

Figure 1: Code for fetching weather data

4.2.2 HERE traffic API v7

The HERE Traffic API v7 provides developers with access to comprehensive traffic information to integrate and display real-time traffic conditions, forecasts and incidents. This API enables the integration of real-time traffic data into applications to improve navigation and traffic management.

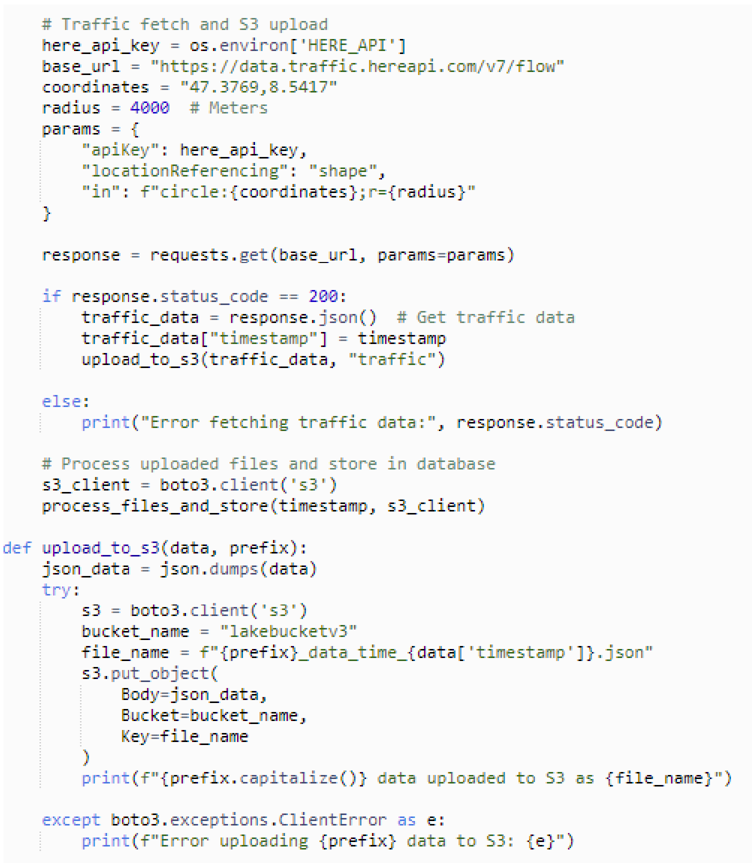
The API uses a RESTful protocol and is accessible via HTTP and HTTPS. An API key is required for use, which is provided after registration with HERE Technologies. The data is provided in JSON and XML formats. The API offers various endpoints to provide different traffic information. A key endpoint is the one for flow data:
The endpoint /traffic/6.3/flow.json provides real-time traffic flow information such as speed and traffic flow on roads. The parameters for using this endpoint include prox for the geographical position (e.g. prox=52.5308,13.3847,5000), bbox for querying a specific area (e.g. bbox=13.0884,52.3413,13.7601,52.6697) and apiKey for the API key.

For this project, we have queried the current traffic situation for the city of Zurich on a 20 min basis. In this query, we draw the following variables:

Variable	Description	Example
Jam Factor	A value that indicates the severity of a traffic jam. It ranges from 0 (no traffic jam) to 10 (complete standstill).	7.5
Confidence	The value of the confidence field indicates the proportion of real time data included in the speed calculation. It is a normalized value between 0.0 and 1.0 0.7 < confidence <= 1.0 indicates real time speeds 0.5 < confidence <= 0.7 indicates historical speeds 0.0 < confidence <= 0.5 indicates speed limit	0.4
Description	Name of the measured Road	"Limmatstrasse"
Length	The length of the affected road section in metres	520
Speed	The expected speed (in meters per second) along the roadway; will not exceed the legal speed limit.	20
Traversability	Describes whether the roadway can be driven. If the road is closed, the Jam Factor is 10 open - the roadway is open and can be driven upon. closed - the roadway is closed (blocked) and cannot be driven (jamFactor is 10.0) upon. reversibleNotRoutable - the roadway is reversible but is currently not routable.	"open"

Table 2: Traffic data

The code to fetch this data can be found in the file "lambda_fetch_store_api_data_s3_and_data-base.ipynb" in the Github repository linked in the annex. It can also be seen in the following screenshot:



```
    # Traffic fetch and S3 upload
    here_api_key = os.environ['HERE_API']
    base_url = "https://data.traffic.hereapi.com/v7/flow"
    coordinates = "47.3769,8.5417"
    radius = 4000  # Meters
    params = {
        "apiKey": here_api_key,
        "locationReferencing": "shape",
        "in": f"circle:{coordinates};r={radius}"
    }

    response = requests.get(base_url, params=params)

    if response.status_code == 200:
        traffic_data = response.json()  # Get traffic data
        traffic_data["timestamp"] = timestamp
        upload_to_s3(traffic_data, "traffic")

    else:
        print("Error fetching traffic data:", response.status_code)

    # Process uploaded files and store in database
    s3_client = boto3.client('s3')
    process_files_and_store(timestamp, s3_client)

def upload_to_s3(data, prefix):
    json_data = json.dumps(data)
    try:
        s3 = boto3.client('s3')
        bucket_name = "lakebucketv3"
        file_name = f"{prefix}_data_time_{data['timestamp']}.json"
        s3.put_object(
            Body=json_data,
            Bucket=bucket_name,
            Key=file_name
        )
        print(f"{prefix.capitalize()} data uploaded to S3 as {file_name}")

    except boto3.exceptions.ClientError as e:
        print(f"Error uploading {prefix} data to S3: {e}")
```

Figure 2: Code for fetching traffic data

4.3 Architecture

The architecture of this data processing and analysis pipeline is divided into several main components: Data Sources, Data Ingestion, Data Lake, Data Warehouse and Visualization. The data sources include the APIs of OpenWeather and HERE, which provide weather and traffic data and are accessible via developer APIs.

AWS Lambda, a Python orchestrator that runs every 20 minutes to extract the data from the OpenWeather and HERE APIs, is used for data ingestion. AWS Lambda acts as a serverless computing platform that automatically scales and only charges for the computing power used.
The extracted data is stored as JSON files in Amazon S3 (Simple Storage Service). S3 offers scalable storage space and high availability for storing large amounts of data, which enables the centralized storage and management of raw data.
The data from the Data Lake is loaded into a PostgreSQL database in Amazon RDS (Relational Database Service) using AWS Lambda. Amazon RDS offers a scalable relational database that is optimized for analytical queries and the management of structured data.
For visualization, Tableau is used to visualize the data stored in Amazon RDS. Tableau offers powerful functions for creating interactive dashboards and analyzing data. In addition, on a virtual machine outside of AWS (for cost reasons), we run the recently released 'llama3' large language model, dubbed: "The most capable openly available LLM to date", by its creator Meta (formerly Facebook). Llama3 was set up using the easy-to-use application "Ollama". The LLM is used to create traffic forecasts based on a given weather prediction. These model predictions are based on the processed and stored data in the pipeline and can be used for further analyses and decision-making processes. In detail this means that using Lambda we can extract relevant traffic and weather data to ask the model to predict the traffic flow in the future. The response as well as the prompt is fed back to our RDS database and can be visualized using Tableau.

This architecture provides an efficient and scalable solution for integrating and analyzing weather and traffic data that can be used for accurate traffic forecasting and informed traffic management decisions.

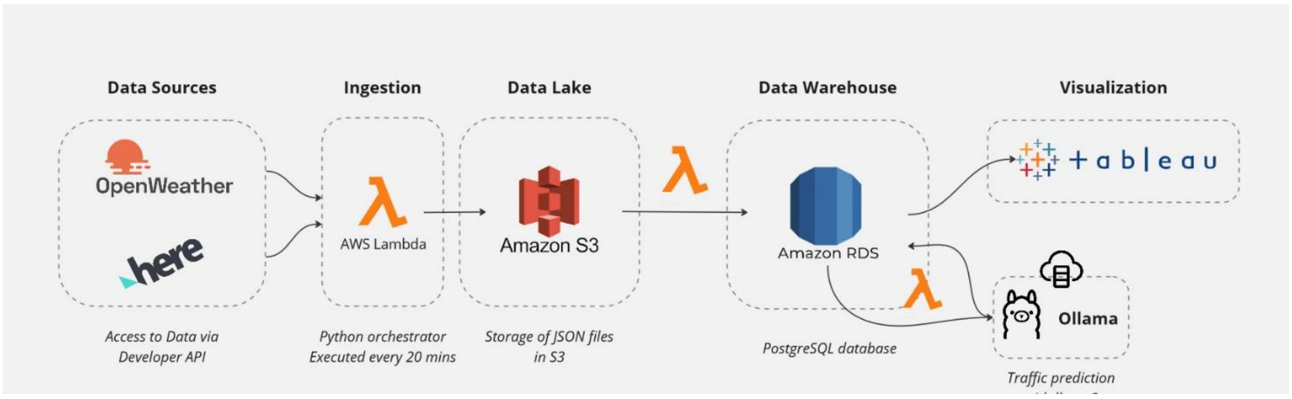


Figure 3: Architecture

4.4 Data sources: Data imputation, data cleaning

This chapter describes the data imputation and data cleaning of the two data sources. Firstly, it ensures that the data is successfully retrieved from the APIs. The code checks whether the API requests are successful (status code 200). If an error occurs, an error message is issued, and the data is not processed further. The extracted data is stored in a structured JSON format and uploaded to Amazon S3. This enables standardized storage and easy access to the raw data.

4.4.1 Traffic Data

Data imputation refers to the replacement of missing or incomplete data with estimated values. During the extraction and processing of traffic data, missing values are replaced by default values. For example, the speed is set to 0 if it is not available in the current traffic data flow. Similarly, the congestion factor, reliability and trafficability are set to 0 or 'open' if this information is missing. If the description is missing, a placeholder value 'no_description' is used. This ensures that all required fields are present and correctly structured.

```
current_flow = entry.get('currentFlow', {})
location = entry.get('location', {})
description = location.get('description', 'no_description')  # Use a placeholder if description is missing
length = location.get('length')
speed = current_flow.get('speed', 0)  # Default to 0 if speed is not available
jam_factor = current_flow.get('jamFactor', 0)  # Default to 0 if jamFactor is not available
confidence = current_flow.get('confidence', 0)  # Default to 0 if confidence is not available
traversability = current_flow.get('traversability', 'open')
```

Before inserting the data into the database, it is ensured that the required tables exist. If the tables do not exist, they are created to ensure that the structure of the database meets the requirements of the application. This is achieved by creating the tables for weather and traffic data.

```
cur.execute("""
            CREATE TABLE IF NOT EXISTS traffic_data_test (
                id SERIAL PRIMARY KEY,
                timestamp BIGINT,
                description TEXT,
                length FLOAT,
                speed FLOAT,
                jam_factor FLOAT,
                confidence FLOAT,
                traversability TEXT
            )
""")
```

During database insertion, a check is made to ensure that the required fields are present and missing values are replaced with default values. For example, for traffic data, fields such as speed, congestion factor and reliability are set to default values if they are not available.

In order to also have a timestamp for the traffic data and to be able to link it to the weather data at a later point in time, the timestamp field is created and filled in when the database is loaded.

4.4.2 Weather Data

During the extraction and processing of the weather data, the relevant fields are extracted from the API response and stored in a structured form. These fields include the timestamp, temperature, perceived temperature, weather description, humidity and wind speed. This structured storage ensures that all the required information is available and formatted correctly.

```
data = response.json()
    weather_data = {
        "timestamp": timestamp,
        "temperature": data['main']['temp'],
        "feels_like": data['main']['feels_like'],
        "description": data['weather'][0]['description'],
        "humidity": data['main']['humidity'],
        "wind_speed": data['wind']['speed']
    }
```

Before inserting the weather data into the database, the system checks whether the required table exists. If the table does not exist, it is created to ensure that the structure of the database meets the requirements of the application. During database insertion, it is ensured that all extracted fields are correctly inserted into the database. The timestamp is used as the primary key to avoid duplicate entries. The temperature, perceived temperature, weather description, humidity and wind speed are inserted into the corresponding fields of the database table.

4.5 Data Ingestion and Data Lake

We already talked about saving data to our database in the previous chapter, let's take a step back and look at the whole picture.

For the data ingestions part of this project, we used AWS Lambda. It is a computing service that allows running Python code without provisioning or managing servers and can thus be very cheap, as you pay only per execution. It's an effective solution for running event-driven code, such as periodically fetching data from an API, which is exactly what we need.

4.5.1 Process

The initial idea was to create a workflow with two lambda functions. The first function would fetch all available data from the free APIs to store it into S3, our data lake. This would include later unused features such as exact coordinates for road sections. Then the second lambda function would extract only the relevant information into our RDS database, our data warehouse. This would enable us to collect data independently and process it at a later stage, giving us more flexibility.

We faced various issues within AWS trying to perform this with two separate lambda functions. Essentially it was not possible to retrieve a given traffic data file from S3, based on a timestamp within the file name. After many hours of debugging, we chalked this up to having too many files (over 2000 per data source) in our S3 bucket and the AWS Boto3 function 'list_objects_v2' not being able to cope with this many files.

Ultimately, we decided to create one big lambda function called "get_traffic_weather_api" that would fetch the data from the two API sources, store a weather and traffic file into our S3 bucket and at the same time transform relevant information directly into respective weather and traffic tables in our database. These end result of these efforts can be found in the file "lambda_fetch_store_api_data_s3_and_database.ipynb" in the Github repository linked in the annex.

4.5.2 Lambda layers

Lambda functions can be enhanced with so-called lambda layers. These layers are essentially zip-files containing a Python installation with further python packages such as pandas, NumPy and other useful Data Science related packages not present in the default Python install.

For our project we needed the following Python packages:

- 'requests', for fetching weather and traffic data from our APIs
- 'psycopg2', for interacting with PostgreSQL databases (our Data Warehouse)
- 'boto3', required for AWS to operate between its services (S3, RDS, etc.)

We did not end up using the pandas and NumPy packages and opted instead to transform our data in the warehouse and later Tableau using SQL instead.

Creating this lambda layer was done in the virtual machine service Cloud9 in AWS. We began with the installation of Python 3.9.6 from source, ensuring a robust setup with optimization enabled. We then created a dedicated virtual environment using 'virtualenv' and installed the Python libraries mentioned above to interact with our data sources and the AWS components. Then we zipped up the directory containing the python install and could either publish it to our lambda layer directly, or in case of any issues we could push it to our S3 bucket and retrieve it there later when creating a lambda layer.

This lambda layer now can be re-used for multiple functions that require these packages. We used it for the primary data ingestion function as well as the LLM prediction function.

4.6 Data Warehouse

Our data warehouse of choice is a PostgreSQL database running on the Amazon RDS service.
Amazon RDS (Relational Database Service) provides users with a host of database technologies to chose from such as Amazon Aurora, PostgreSQL, MySQL, MariaDB, Oracle Database, and Microsoft SQL Server. RDS is designed to provide a cost-effective and reliable solution for running relational databases in a cloud environment. We found this to be the case, our credit consumption was moderate, and we didn't manage to use up the USD 100 credit over the course of the project.

We chose PostgreSQL over other alternatives due to its ease-of-use handling JSON files, our primary data files in our data lake.

Our data warehouse consists of one database called 'traffic_weather_db1'.
The database schema is comprised of three main tables:

- traffic_data
- weather_data
- traffic_prediction

These tables are designed to store traffic data, weather data retrieved from the HERE and OpenWeather APIs, as well as the traffic predictions created by our LLM.

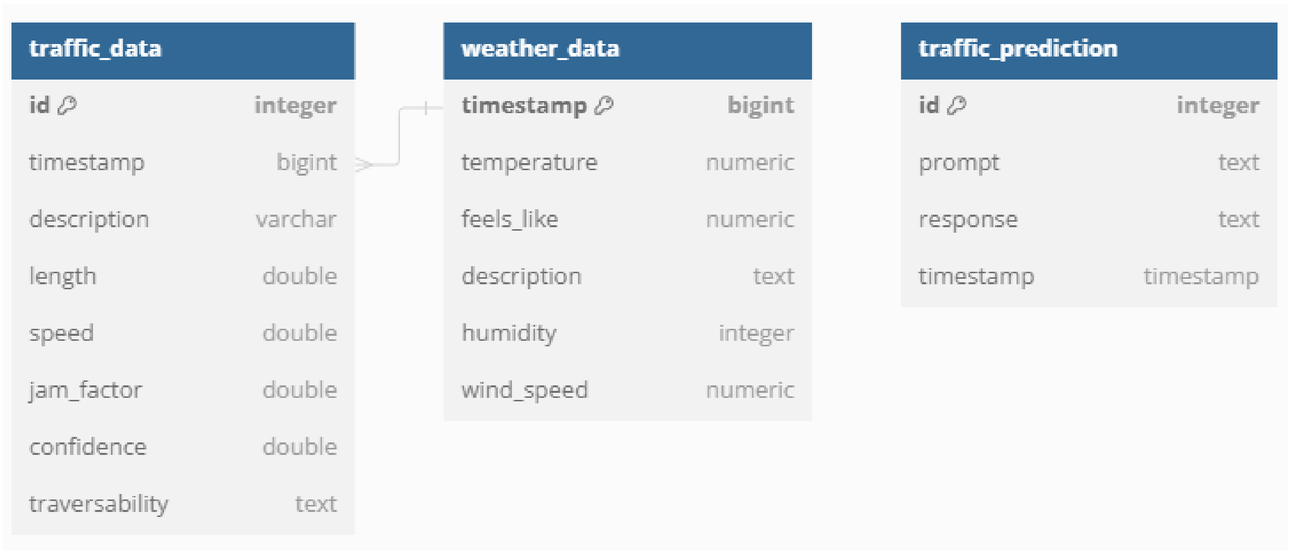
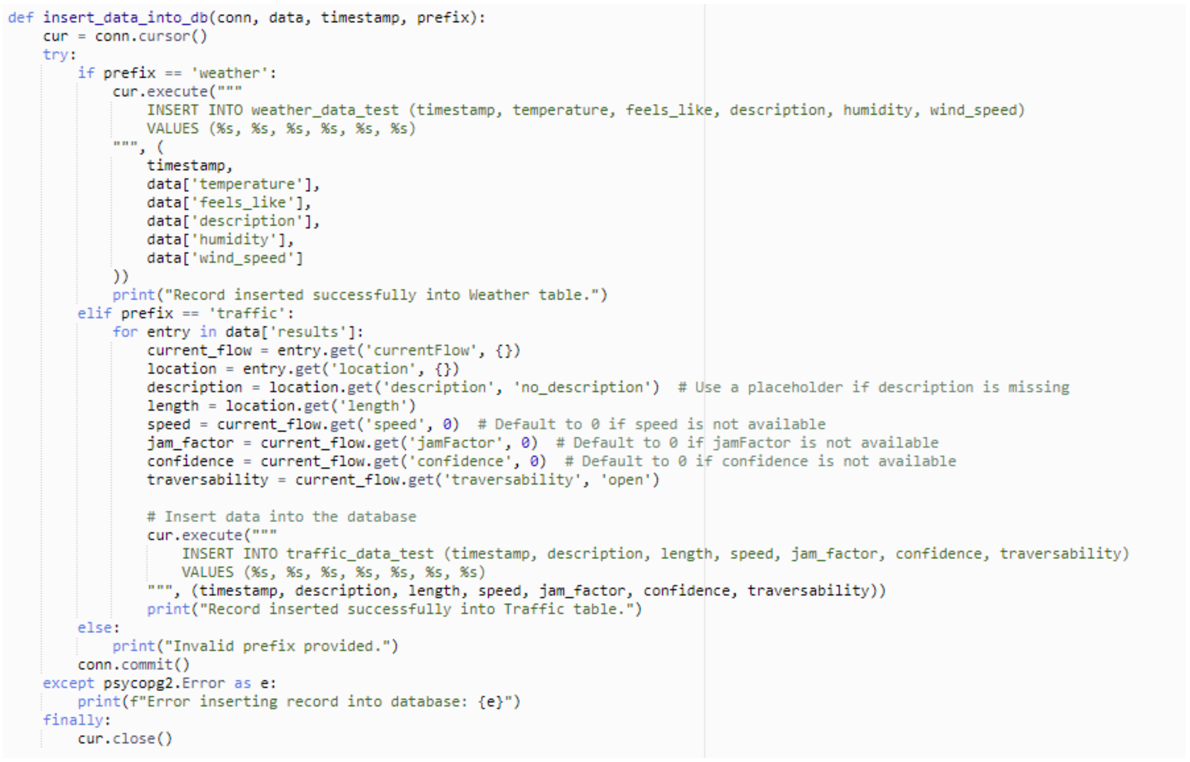


Figure 4: Database model

The cardinality of the “traffic_data” table and the “weather_data” table is one to many. A weather forecast has several traffic data. This is because there are multiple traffic_data entries for a single weather_data entry, as traffic conditions can be recorded on different road sections or at different times within the same time period. Access to the database is provided in the annex. The following screenshot shows finally how the data is inserted into the database.



```python
def insert_data_into_db(conn, data, timestamp, prefix):
    cur = conn.cursor()
    try:
        if prefix == 'weather':
            cur.execute("""
                INSERT INTO weather_data_test (timestamp, temperature, feels_like, description, humidity, wind_speed)
                VALUES (%s, %s, %s, %s, %s, %s)
            """, (
                timestamp,
                data['temperature'],
                data['feels_like'],
                data['description'],
                data['humidity'],
                data['wind_speed']
            ))
            print("Record inserted successfully into Weather table.")
        elif prefix == 'traffic':
            for entry in data['results']:
                current_flow = entry.get('currentFlow', {})
                location = entry.get('location', {})
                description = location.get('description', 'no_description')  # Use a placeholder if description is missing
                length = location.get('length')
                speed = current_flow.get('speed', 0)  # Default to 0 if speed is not available
                jam_factor = current_flow.get('jamFactor', 0)  # Default to 0 if jamFactor is not available
                confidence = current_flow.get('confidence', 0)  # Default to 0 if confidence is not available
                traversability = current_flow.get('traversability', 'open')

                # Insert data into the database
                cur.execute("""
                    INSERT INTO traffic_data_test (timestamp, description, length, speed, jam_factor, confidence, traversability)
                    VALUES (%s, %s, %s, %s, %s, %s, %s)
                """, (timestamp, description, length, speed, jam_factor, confidence, traversability))
                print("Record inserted successfully into Traffic table.")
        else:
            print("Invalid prefix provided.")
        conn.commit()
    except psycopg2.Error as e:
        print(f"Error inserting record into database: {e}")
    finally:
        cur.close()
```

Figure 5: Inserting data into our data warehouse

4.7 Traffic Prediction

One question we wanted to answer is if we could predict traffic, more precisely the jam_factor, of a given street based on previous patterns in data (e.g. time, day) and some given weather data (weather forecast).

Initially we wanted to train a regression model or use deep learning methods for our prediction, however due to time and budget constraints, we decided to simplify the approach and use a large language model to make a prediction for us. In each a case a virtual machine executing Python code would be necessary, however training a model would have been outside of the AWS scope for this project. The LLM route was for all intents and purposes a simpler approach.

We received access to a virtual machine, set up outside of AWS, by the lecturers of this course. Given more resources (i.e. compute credits) this could have also been achieved with Cloud9 in AWS. The VM was set up with Ollama, which can easily be used to install various LLM’s. We trialed a few including ‘Gemma’, ‘Phi 3’ and ‘Mistral’, however we decided to go with the recently released ‘Llama3’ which subjectively gave us the best responses in this phase.

Next, we used a lambda function, conveniently also using the layer we created earlier, to extract relevant traffic data from our database to a string, which then could be used for a prompt to the LLM. Within the lambda function we first specified a single street name and then extracted the last 500 datapoints excluding the hours from midnight to 4am. Basically, this was done to cut down on the size of the context window for the prompt. As we are probably more interested in the traffic during the day we don’t need these 4 hours of nighttime data.

LLMs tend to output a lot of (meaningless) text. In our case they would often analyze data and not answer our questions. The LLMs would create summaries and basic analysis of the given data, instead of answering the question at the end, i.e.: “based on this data, what would you predict…”

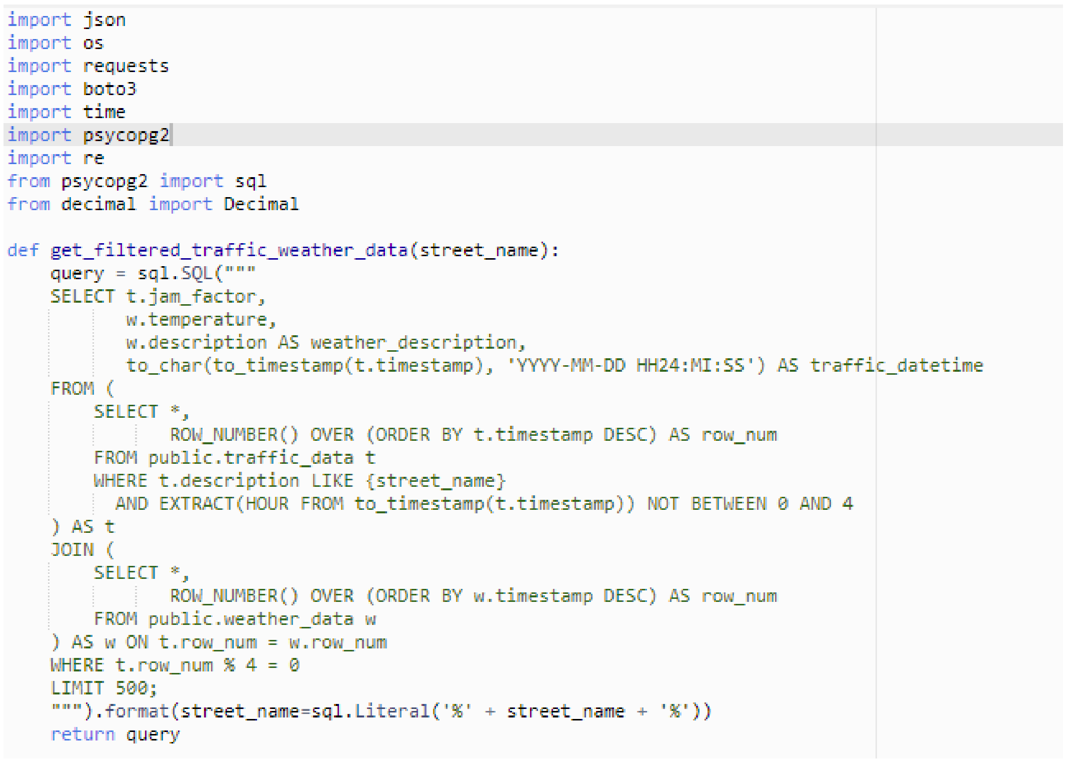
After extracting the relevant data we combined it with a carefully crafted prompt to ensure that we receive a prediction based on the data, not an analysis of the given data. The prompt was as follows:

"You are given the following information for one street in the city of Zurich. It has a jamfactor, the temperature in degrees (C) and the weather status. The jamfactor can take a value from 0 to 10 and represents how jammed the street is, 10 meaning the road is closed. Make a prediction for the jam factor of the next day based on the following weather: {temperature}C {weather_description}. Don't analyze anything else about the data, just predict the new jam factor. {data}"

The prompt could be further tuned for more different results. We could try combining multiple streets to get a prediction for a general area (e.g. around the northern lake area, or the main train station). We could also ask it to predict for a specific time of the day, as this was not specified in the prompt thus far. Also, we could try and make it aware of time and day being present in the data and ask it to consider this in its prediction.

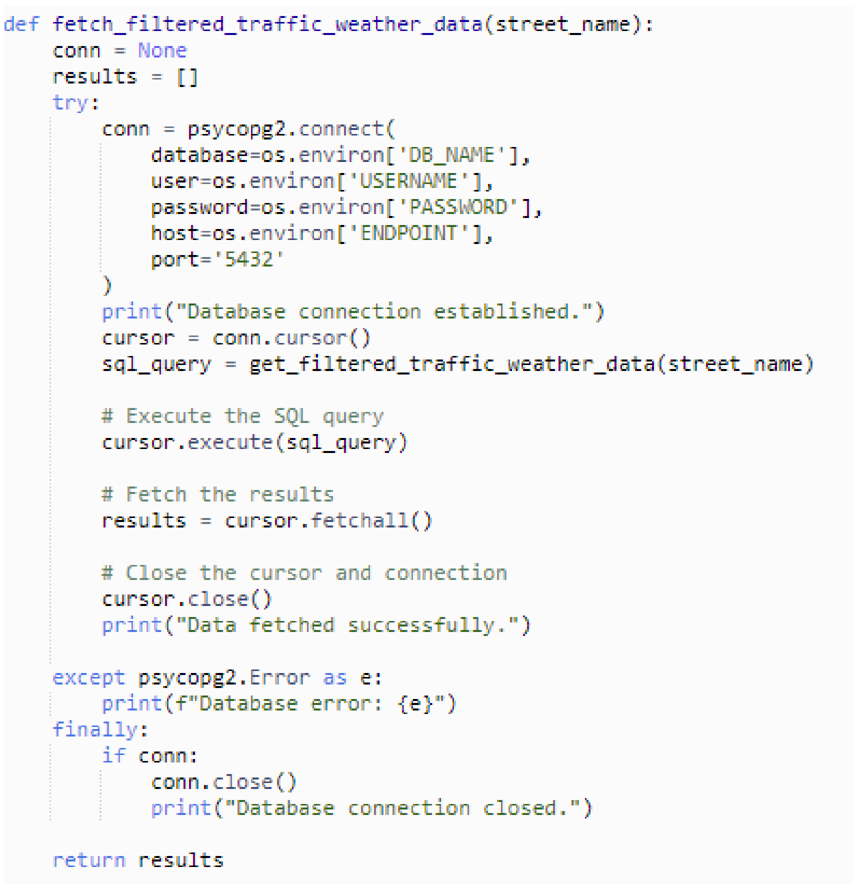
The results from the model, along with the input prompt, were then stored back into the database to maintain a historical record of the predictions, as well as making it accessible for visualizations in Tableau.

These efforts can be found found in the file "lambda_llm_prediction.ipynb" in the Github repository linked in the annex. It is the basis for the "predict_traffic" lambda function. The following screenshots show the most important parts of the implementation:



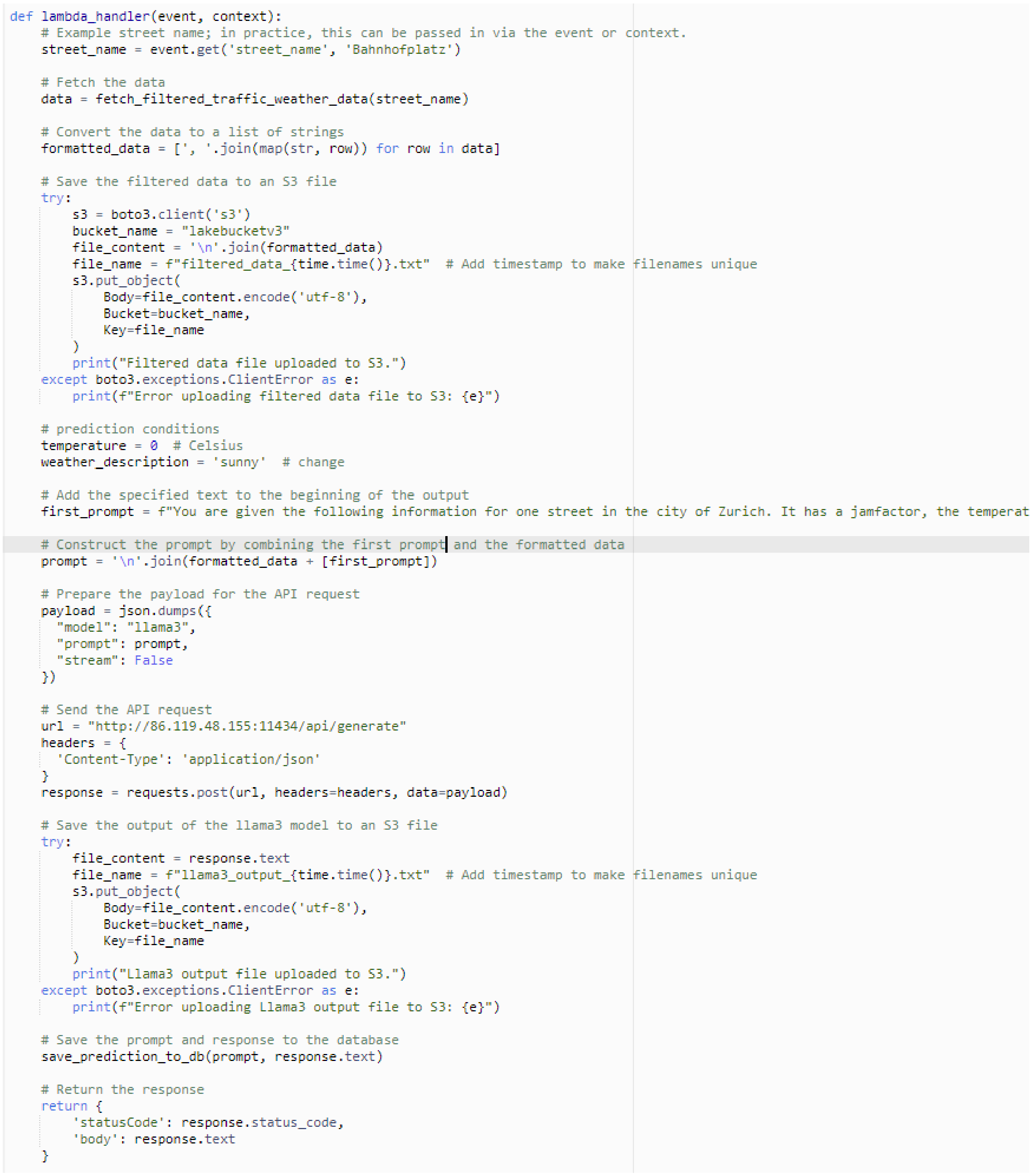
```
import json
import os
import requests
import boto3
import time
import psycopg2
import re
from psycopg2 import sql
from decimal import Decimal

def get_filtered_traffic_weather_data(street_name):
    query = sql.SQL("""
    SELECT t.jam_factor,
           w.temperature,
           w.description AS weather_description,
           to_char(to_timestamp(t.timestamp), 'YYYY-MM-DD HH24:MI:SS') AS traffic_datetime
    FROM (
        SELECT *,
               ROW_NUMBER() OVER (ORDER BY t.timestamp DESC) AS row_num
        FROM public.traffic_data t
        WHERE t.description LIKE {street_name}
          AND EXTRACT(HOUR FROM to_timestamp(t.timestamp)) NOT BETWEEN 0 AND 4
    ) AS t
    JOIN (
        SELECT *,
               ROW_NUMBER() OVER (ORDER BY w.timestamp DESC) AS row_num
        FROM public.weather_data w
    ) AS w ON t.row_num = w.row_num
    WHERE t.row_num % 4 = 0
    LIMIT 500;
    """).format(street_name=sql.Literal('%' + street_name + '%'))
    return query
```

Figure 6: code for executing LLM prediction (1)



```python
def fetch_filtered_traffic_weather_data(street_name):
    conn = None
    results = []
    try:
        conn = psycopg2.connect(
            database=os.environ['DB_NAME'],
            user=os.environ['USERNAME'],
            password=os.environ['PASSWORD'],
            host=os.environ['ENDPOINT'],
            port='5432'
        )
        print("Database connection established.")
        cursor = conn.cursor()
        sql_query = get_filtered_traffic_weather_data(street_name)

        # Execute the SQL query
        cursor.execute(sql_query)

        # Fetch the results
        results = cursor.fetchall()

        # Close the cursor and connection
        cursor.close()
        print("Data fetched successfully.")

    except psycopg2.Error as e:
        print(f"Database error: {e}")
    finally:
        if conn:
            conn.close()
            print("Database connection closed.")

    return results
```

Figure 7: code for executing LLM prediction (2)



```python
def lambda_handler(event, context):
    # Example street name; in practice, this can be passed in via the event or context.
    street_name = event.get('street_name', 'Bahnhofplatz')

    # Fetch the data
    data = fetch_filtered_traffic_weather_data(street_name)

    # Convert the data to a list of strings
    formatted_data = [', '.join(map(str, row)) for row in data]

    # Save the filtered data to an S3 file
    try:
        s3 = boto3.client('s3')
        bucket_name = "lakebucketv3"
        file_content = '\n'.join(formatted_data)
        file_name = f"filtered_data_{time.time()}.txt"  # Add timestamp to make filenames unique
        s3.put_object(
            Body=file_content.encode('utf-8'),
            Bucket=bucket_name,
            Key=file_name
        )
        print("Filtered data file uploaded to S3.")
    except boto3.exceptions.ClientError as e:
        print(f"Error uploading filtered data file to S3: {e}")

    # prediction conditions
    temperature = 0  # Celsius
    weather_description = 'sunny'  # change

    # Add the specified text to the beginning of the output
    first_prompt = f"You are given the following information for one street in the city of Zurich. It has a jamfactor, the temperat

    # Construct the prompt by combining the first prompt and the formatted data
    prompt = '\n'.join(formatted_data + [first_prompt])

    # Prepare the payload for the API request
    payload = json.dumps({
      "model": "llama3",
      "prompt": prompt,
      "stream": False
    })

    # Send the API request
    url = "http://86.119.48.155:11434/api/generate"
    headers = {
      'Content-Type': 'application/json'
    }
    response = requests.post(url, headers=headers, data=payload)

    # Save the output of the llama3 model to an S3 file
    try:
        file_content = response.text
        file_name = f"llama3_output_{time.time()}.txt"  # Add timestamp to make filenames unique
        s3.put_object(
            Body=file_content.encode('utf-8'),
            Bucket=bucket_name,
            Key=file_name
        )
        print("Llama3 output file uploaded to S3.")
    except boto3.exceptions.ClientError as e:
        print(f"Error uploading Llama3 output file to S3: {e}")

    # Save the prompt and response to the database
    save_prediction_to_db(prompt, response.text)

    # Return the response
    return {
        'statusCode': response.status_code,
        'body': response.text
    }
```

Figure 8: code for executing LLM prediction (3)

4.8 Data Lake Consumers

The following table describes four different personas that could benefit from the developed data architecture.

Personas	Description	Use Case
Traffic Planners	Traffic planners work on designing and optimizing traffic flow in urban and rural areas.	• Analyzing traffic patterns and identifying causes of congestion. • Planning road infrastructure projects. • Forecasting and modeling future traffic scenarios considering weather conditions.
City Administrators	City administrators are responsible for managing and improving urban infrastructure.	• Monitoring and evaluating the efficiency of existing traffic management measures. • Planning and implementing measures to reduce traffic congestion. • Informing the public about current traffic conditions and expected disruptions.
Emergency Services	Emergency services include police, fire, and medical response units.	• Real-time access to traffic and weather data to plan safe routes to incidents. • Predicting potential traffic issues that could impact response times. • Coordinating and planning emergency responses during extreme weather conditions.
Traffic Engineers	Traffic engineers analyze traffic flows and develop systems to improve transportation infrastructure.	• Conducting detailed analyses to improve road capacity and reduce congestion. • Developing and testing traffic control algorithms and systems. • Validating simulation models with real traffic data.

Table 3: Data lake consumers

5 Results

The following chapter describes the results of the study. In a first step, the data import is discussed and then the research questions are described using the graphical representations.

5.1 Tableau – Data import

The evaluations were all prepared and graphically displayed using Tableau. The tables were linked together in Tableau after accessing the Postgres SQL server. The link takes place via the timestamp.

The dashboards can be viewed online using this link: Traffic-Lake | Tableau Public

5.2 Correlation between temperature and the volume of traffic

The first question deals with the relationship between the temperature and the jam factor. A line diagram was used to prepare this research question graphically.

The figure 3 shows a line representing the relationship between temperature and the average jam factor (Avg. Jam Factor). The x-axis represents the temperature in degrees Celsius (from 0 to 27), while the y-axis shows the average jam factor (from 0 to 2.8). The jam factor rises noticeably between 2-8 degrees Celsius and then levels off again. Around 8 degrees there is a tiny drop before the it steadily rises agon with the temperature. In summary we observed the higher the temperature the higher the jam factor, with a special area of interest between 2 and 8 degrees.

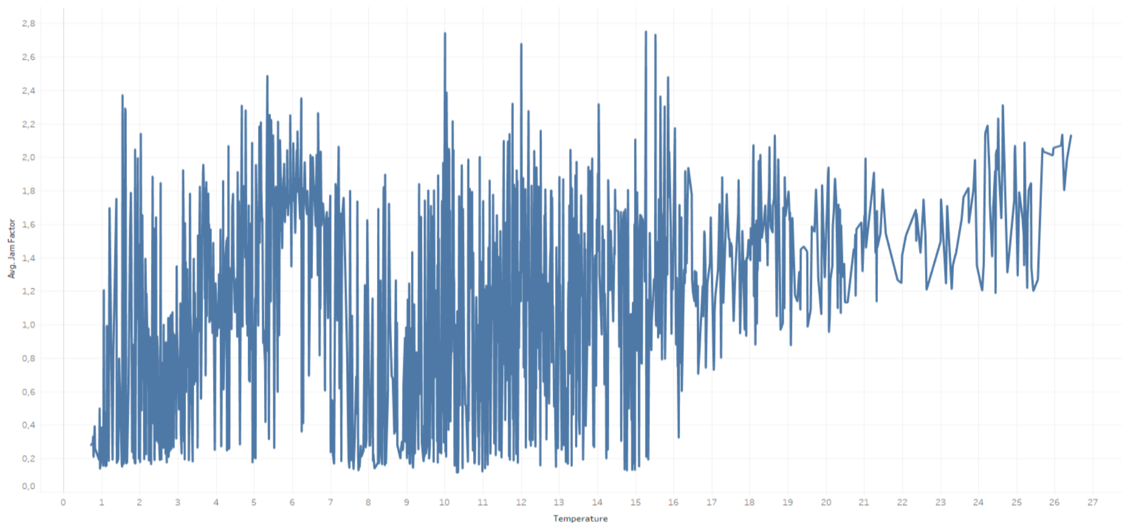


Figure 9: Correlation between temperature and the volume of traffic

5.3 Do different weather events have an influence on traffic behavior?

Two different graphics were created to answer the research question of whether the different weather events have an influence on traffic behavior.

The figure 4 shows a horizontal bar chart that illustrates the relationship between different weather conditions and the average jam factor (Avg. Jam Factor). The y-axis lists the weather conditions, while the x-axis shows the average jam factor from 0 to 2.0. The graph shows that snow has a high average jam factor, whereas a clear sky, for example, has a lower average jam factor.

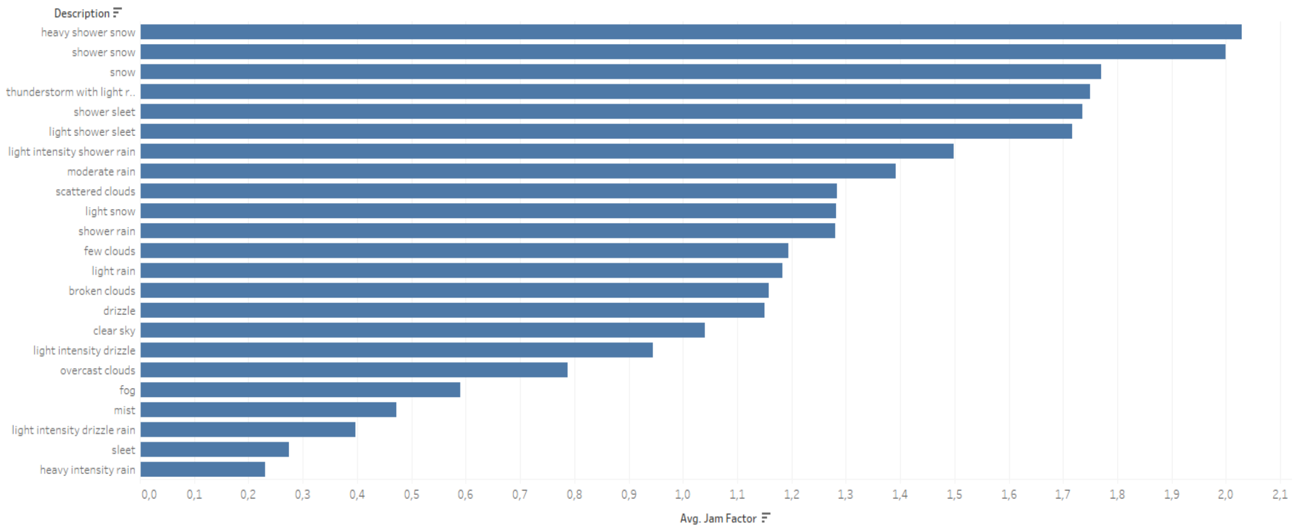


Figure 10: Influence of weather events on traffic behavior

The figure 5 shows a boxplot diagram that illustrates the relationship between different weather conditions and the jam factor. The various weather conditions are listed on the x-axis, while the y-axis shows the jam factor from 0 to 12. This representation makes it possible to compare the distribution and dispersion of the jam factor under different weather conditions.
Certain weather conditions have a higher spread of the congestion factor. For example, “Heavy intensity rain” and “Light intensity shower rain” show a large dispersion of the congestion factor with some outliers reaching values close to 10. This indicates that traffic congestion can vary greatly in these conditions. In contrast, “Fog” has a relatively low median congestion factor and less dispersion, indicating a more consistent, albeit moderate, traffic situation.
Particularly high median congestion factors are observed for conditions such as “Snow” and “Thunderstorm with light rain”. These weather conditions lead to more intense congestion more frequently. Similar trends can be seen for “Shower sleet” and “Shower snow”, which also have high median congestion factors.
On the other hand, weather conditions such as “Light intensity rain”, “Light intensity shower rain” and “Drizzle” have lower median congestion factors, indicating less intense traffic congestion. Clear conditions such as “Clear sky” and “Few clouds” show a relatively low dispersion and lower median congestion

factors, indicating an overall smoother traffic situation. Several weather conditions show outliers that are well above the upper quartile. This suggests that extreme congestion conditions may occasionally occur under certain conditions, although the overall trend may be more moderate.

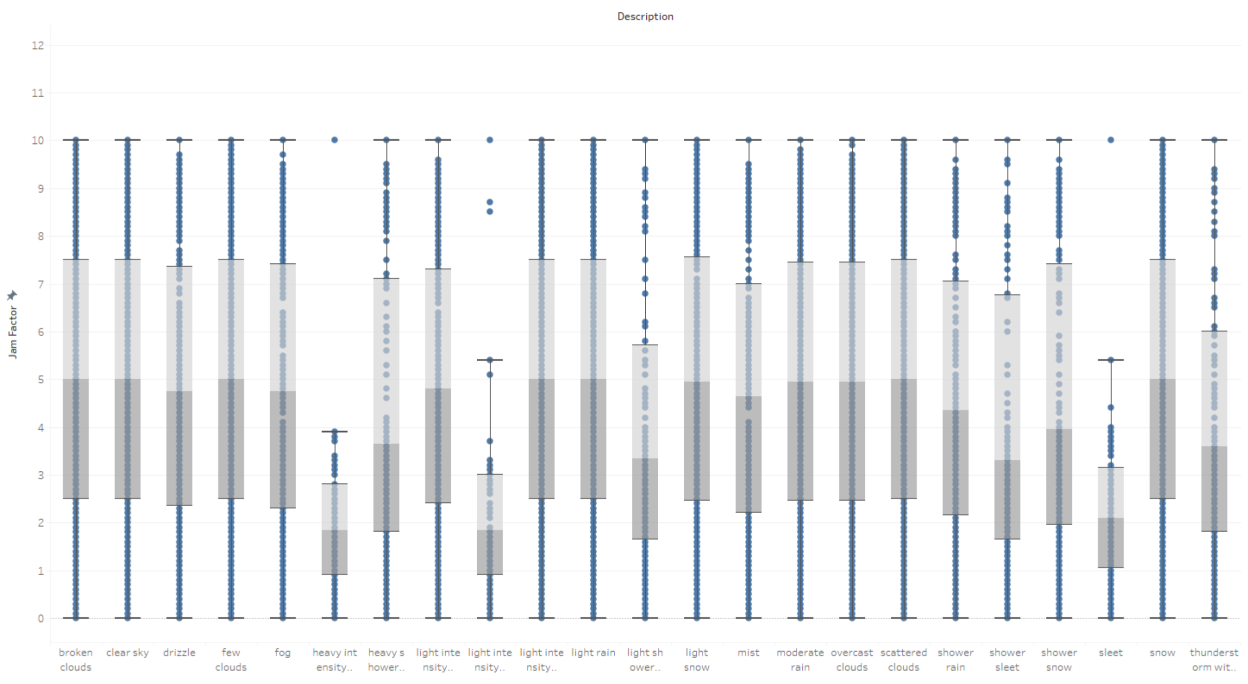


Figure 11: Box-Plot weather events and jam factor

To summarize, extreme weather conditions such as heavy snowfall, thunderstorms and heavy rain tend to result in higher and more variable congestion factors. In contrast, milder conditions such as drizzle and light cloud cover lead to lower congestion factors and an overall smoother traffic situation.

5.4 How do traffic patterns vary throughout the day, week, and year?

To answer this research question two charts were created. One of them illustrates the average jam factor (Avg. Jam Factor) over the course of a day and the other over the course of a week.

Over the course of the day, the figure 6 shows that the jam factor starts at around 0.2 in the early hours of the morning and increases significantly at around 6:00 am. This increase reaches a first peak at around 8 a.m., which indicates the morning commuting time. After a slight decrease, the congestion factor increases again and reaches its maximum value of around 2.0 between 4pm and 6pm, reflecting the afternoon and evening traffic peaks. After 6 p.m., the congestion factor drops sharply again and reaches a low value around 10pm.

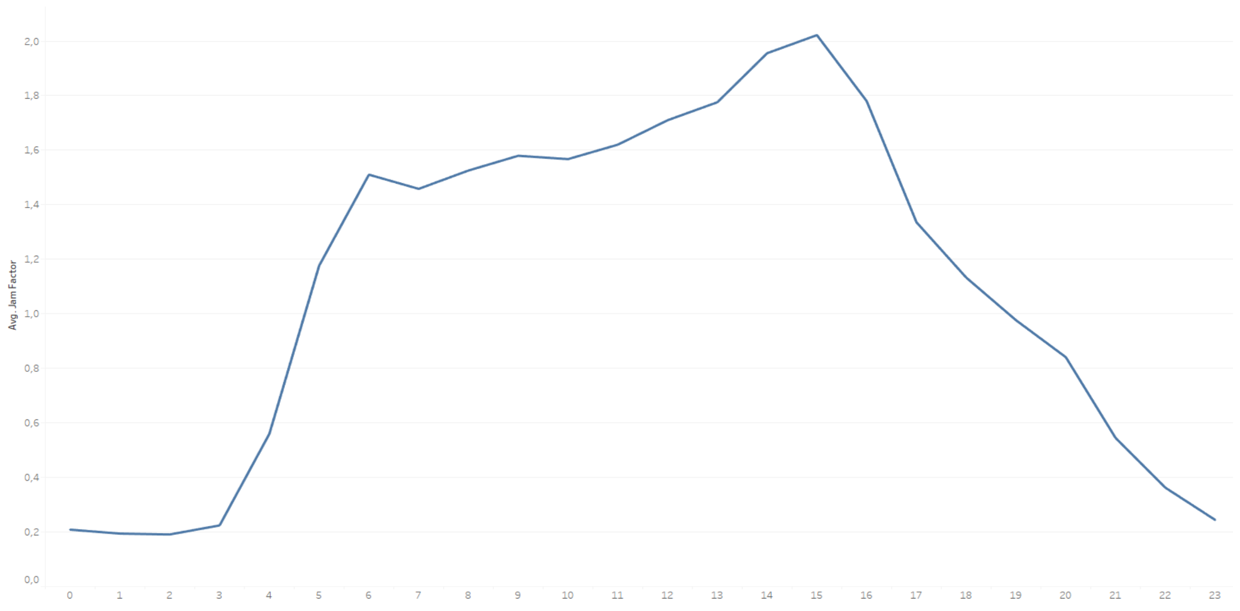


Figure 12: traffic pattern throughout the day

The figure 7 shows the congestion factor over the course of the week. It is relatively high on working days from Monday to Friday. There the congestion factor reaches its highest value in the middle of the week, particularly from Tuesday to Thursday, with a value of around 1.2. On Friday, the congestion factor drops slightly and falls significantly at the weekend to a value of around 0.7 on Sunday. This indicates that the volume of traffic at weekends is lower than on working days. The people visiting the city on weekends do not cause as much traffic (i.e. jam factor) as commuters during the work week, even if it sometimes subjectively seems to be the case.

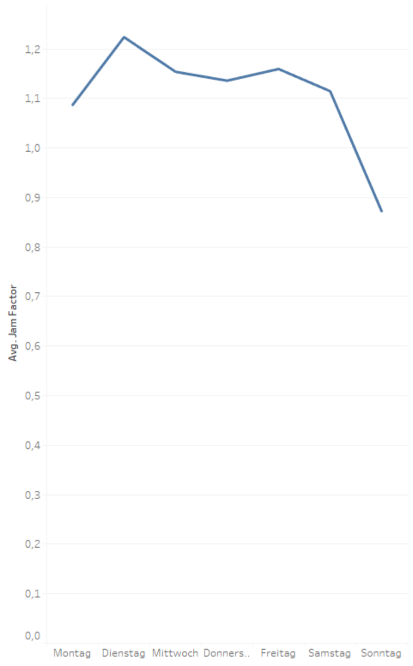


Figure 13: traffic pattern throughout the week

In summary, the diagrams show that traffic congestion is dependent on both the time of day and the day of the week. During the day, the highest levels of congestion occur in the afternoon, while weekdays are more affected by congestion than weekends. These patterns reflect typical commuter flows and work routes, which have a significant impact on traffic volumes during rush hours and on working days.

6 Interpretation

The analysis of the various graphs provides valuable insights into the relationships between weather conditions, times of day, days of the week and the average jam factor.

6.1 Influence of temperature on the traffic volume

The figure 3 shows the relationship between temperature and the average stagnation factor. It is clear that there is considerable variability in the congestion factor at different temperatures. At temperatures below 10 degrees Celsius, the congestion factors are often higher and more variable, which could indicate higher traffic volumes or more difficult driving conditions. At temperatures above 20 degrees Celsius, the average congestion factor appears to be more stable and tends to be higher, indicating that at warmer temperatures traffic is more consistent but denser. The fact that the data was only collected from February to April may have skewed the results, as higher temperatures occur less frequently during these months.
Based on the available data, it is not feasible to say whether temperature has an influence on traffic volumes. This would require further data to be collected throughout the year.

6.2 Influence of weather event on traffic behavior

The figure 4 shows a bar chart representing the average congestion factor for different weather conditions. Extreme weather conditions such as heavy snowfall ("heavy shower snow", "shower snow", "snow") and thunderstorm with light rain ("thunderstorm with light rain") lead to higher congestion factors. Milder weather conditions such as "light rain", "drizzle" and "scattered clouds", "few clouds" have lower congestion factors, indicating less intense congestion. These results could indicate that adverse weather conditions have an impact on traffic volume and congestion.

The boxplot clearly shows that the dispersion of the congestion factor is greater in extreme weather conditions such as heavy rain and snowfall, indicating a high variability in traffic congestion. Weather conditions such as fog and light drizzle have lower dispersion and lower median congestion factors, indicating more consistent, albeit moderate, traffic conditions. This distribution indicates that extreme weather events can occasionally lead to extreme congestion conditions.

Based on the available results, it can be said that the weather events have an influence on the traffic behaviour.

6.3 Variation of traffic patterns during the day and week

The analysis of the daily and weekly course of the average jam factor provides valuable insights into the traffic load at different times. Over the course of the day, the jam factor starts low in the early morning hours and increases significantly from 6:00h, which indicates the morning commuter period. The peak is reached between 16:00h and 18:00h, which reflects the evening rush hour. After 6 p.m., the congestion factor decreases again, indicating a lower traffic density in the late evening hours.

Over the course of the week, the highest congestion factors occur on working days from Monday to Friday, with the congestion factor being highest in the middle of the week (Tuesday to Thursday). At weekends, the congestion factor drops significantly, especially on Sundays, which indicates a lower volume of traffic at weekends. These patterns reflect typical commuter flows and work routes that influence traffic volumes during rush hours and on working days.

7 Conclusion

The study shows that the congestion factor varies greatly at different temperatures. At temperatures below 10 degrees Celsius, the congestion factors are often higher and more variable, which could indicate higher traffic volumes or more difficult driving conditions. At temperatures above 20 degrees Celsius, the average congestion factor is more stable and tends to be higher, indicating more consistent but denser traffic. Extreme weather conditions such as heavy snowfall and thunderstorms lead to higher congestion factors, while milder conditions such as drizzle led to lower congestion factors. This is consistent with recent research showing that adverse weather conditions have a significant impact on traffic volumes. Over the course of the day, the congestion factor increases in the morning and peaks in the afternoon before decreasing again in the evening. Over the course of the week, the highest congestion factors are observed on working days, while they are significantly lower at weekends. Our results are consistent with current research, which also demonstrates the significant impact of weather conditions on traffic behavior and shows that incorporating weather data into traffic forecasts leads to more efficient traffic management strategies. These findings are crucial for traffic planning and the management of urban infrastructures. They underline the need to develop traffic strategies aimed at relieving road congestion during peak periods, for example through flexible working hours, promotion of public transport and weather-adapted traffic management measures.

However, it must also be said that the weather alone cannot be used to make accurate traffic forecasts and that other aspects should also be considered. This is also shown by the tests carried out with LLM forecasts. The traffic jam forecasts tend to be close to the actual traffic jams, but they can also deviate significantly. Accordingly, other factors must also play a role. Another point that should be mentioned is that the average jam factor is rather low and there could therefore there is most likely an error in the data handling or aggregation. The actual data seems to be correct with constantly higher jam factors listed than seen in the chapter "Interpretation", this would have to be looked in a further study.

8 Reflection

During the project, we had the opportunity to gain extensive experience with the Amazon Cloud. These experiences were educational, but also came with some challenges.
One issue we encountered concerned the use of AWS Lambda Layers. More specifically creating a custom proved impossible to get working locally, instead we needed to use the virtual machine service Cloud9, which made local development difficult. Another aspect was that during the development process we had to constantly switch between local and cloud-based environments. Our code created for lambda would not work locally and vice versa, small adaptions were always required. At the end of the project one team mentioned a way to execute lambda code locally, this would have been helpful for us.
Another constant concern was the fear of unexpected costs. Although AWS offers many useful services, the pricing structure is complicated, and it is not immediately apparent what the costs incurred are for the products we used. This meant that we had to constantly monitor usage to ensure that the project budget was not exceeded.
User management and resource sharing in AWS also proved to be cumbersome. We had to essentially use one person's account, so that the entire team could access them same data in the AWS environment. Another hurdle was backing up the collected weather and traffic data. The number of files in our S3 buckets grew quickly and they could no longer be downloaded through the web interface. We were provided with a Jupyter notebook file to download contents from our S3 buckets ("s3_bucket_download.ipynb" linked in the GitHub repository in the Annex). Finally, the use of an LLM for forecasting should be reconsidered. Either using different machine learning methods or using a specially trained LLM for such use cases should probably be used to obtain reasonably intelligent results.

9 Outlook

This study provides important insights into the relationship between weather conditions and traffic volumes, but also lays the foundation for further research. Future studies should include a longer data period to better understand seasonal effects and long-term trends. Extending the data collection to different seasons could improve the analysis of the effects of extreme weather conditions such as heat in summer and snow in winter.
Furthermore, it would be useful to further optimize accuracy and predictive capability by using advanced machine learning models such as bidirectional Long Short-Term Memory (BiLSTM) networks and Graph Neural Networks (GNN). These models could better capture the spatio-temporal dependencies between weather conditions and traffic patterns.

Another research approach could include the integration of additional data sources, such as real-time traffic surveillance cameras and historical accident statistics, to obtain a more comprehensive picture of traffic conditions and their determinants. In addition, social and economic factors that influence traffic behavior could also be looked at to help understand traffic dynamics. Finally, simulations and scenario analyses could be developed to assess the potential impact of traffic management measures under different weather conditions and thus contribute to the development of resilient urban transport systems.

List of sources

Alourani, A., Ashfaq, F., Jhanjhi, N. Z., & Ali Khan, N. (2023). BiLSTM- and GNN-Based Spatiotemporal Traffic Flow Forecasting with Correlated Weather Data. *Journal of Advanced Transportation*, *2023*, 1–17. https://doi.org/10.1155/2023/8962283

Becker, N., Rust, H. W., & Ulbrich, U. (2022). Weather impacts on various types of road crashes: A quantitative analysis using generalized additive models. *European Transport Research Review*, *14*(1), 37. https://doi.org/10.1186/s12544-022-00561-2

Bi, H., Ye, Z., & Zhu, H. (2022). Data-driven analysis of weather impacts on urban traffic conditions at the city level. *Urban Climate*, *41*, 101065. https://doi.org/10.1016/j.uclim.2021.101065

Gratton, G. B., Williams, P. D., Padhra, A., & Rapsomanikis, S. (2022). Reviewing the impacts of climate change on air transport operations. *The Aeronautical Journal*, *126*(1295), 209–221. https://doi.org/10.1017/aer.2021.109

Zang, J., Jiao, P., Liu, S., Zhang, X., Song, G., & Yu, L. (2023). Identifying Traffic Congestion Patterns of Urban Road Network Based on Traffic Performance Index. *Sustainability*, *15*(2), 948. https://doi.org/10.3390/su15020948

List of illustrations

List of tables

Annex

Github repository

https://github.com/MJ-HSLU/dw_dl_project

Code Project Leads

Project	Yanik Baumann	Michael Jung
Weather data API fetch and store to Data Lake (S3)		x
Traffic data API fetch and store to Data Lake (S3)	x	
Setting up database (RDS – PostgreSQL)	x	x
Transform weather data from Data Lake to Data Warehouse		x
Transform traffic data from Data Lake to Data Warehouse	x	
LLM prediction code		x

Evaluation

		Traffic Lakes	
		Yanik	*Michael*
Introduction	**5**	**5**	**5**
Table of contents Topic introduction Problem definition & Goals Business/Research questions & limita-tions	*5*	x	x
State of the art	**5**	**5**	**5**
Literature review (Academic / Busi-ness)	*5*	x	x
Methodology	**50**	**45**	**40**
Data Lake Part			
Data lake architecture, System etc. in technical details	*5*		x
Design choices / trade-offs / assump-tions	*5*	x	x
Data sources definitions (technical)	*5*	x	x
Data imputation, data cleaning per data source	*5*	x	x
Data analysis per data source	*5*	x	x
Database schema, model, table de-scriptions and/or files.	*5*	x	x
Data Lake consumers (personas)	*5*	x	
Data Warehouse Part			
Data warehouse architecture, system etc. in technical details	*5*	x	x
Visualization design justification	*5*	x	x
Tableau visualization (with public link)	*5*	x	
Conclusions	**5**	**5**	**5**
Answer to research/business ques-tions Proper discussion of the solution Proper discussion of the project out-comes Future work, areas to improve in the data lake, data warehouse, what would you do different, etc.	*5*	x	x
GRADE BY STUDENT / 65	**65**	60	55