UNIVERSITY OF PIRAEUS – DEPARTMENT OF INFORMATICS
ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ – ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

MSc "Advanced Informatics and Computing Systems – Software Development and Artificial Intelligence"

ΠΜΣ «Προηγμένα Συστήματα Πληροφορικής – Ανάπτυξη Λογισμικού και Τεχνητής Νοημοσύνης»

MSc Thesis

Μεταπτυχιακή Διατριβή

Thesis Title : Τίτλος Διατριβής :	**Real-time Train Tracking and Anomaly Detection System** Σύστημα παρακολούθησης αμαξοστοιχιών και ανίχνευσης ανωμαλιών σε πραγματικό χρόνο
Student's Name – Surname : Ονοματεπώνυμο Φοιτητή :	**Georgios Birmpakos** Γεώργιος Μπιρμπάκος
Father's Surname : Πατρώνυμο :	**Charilaos Birmpakos** Χαρίλαος Μπιρμπάκος
Student's Number : Αριθμός Μητρώου :	**MPSP2220** ΜΠΣΠ2220
Supervisor : Επιβλέπων :	**Dionysios Sotiropoulos, Assistant Professor** Διονύσιος Σωτηρόπουλος, Επίκουρος Καθηγητής

June 2024/ Ιούνιος 2024

3-Member Examination Committee

Τριμελής Εξεταστική Επιτροπή

Dionisios Sotiropoulos	**Georgios Tsichrintzis**	**Evangelos Sakkopoulos**
Assistant Professor	**Professor**	**Associate Professor**
Διονύσιος Σωτηρόπουλος	Γεώργιος Τσιχριντζής	Ευάγγελος Σακκόπουλος
Επίκουρος Καθηγητής	Καθηγητής	Αναπληρωτής Καθηγητής

Acknowledgements

In the pursuit of this investigation, I extend my sincere appreciation to my family for their steadfast encouragement throughout my scholarly and vocational journey, furnishing invaluable physical and psychological sustenance pivotal to the attainment of my goals.

Additionally, I wish to express profound gratitude to my life partner, whose unwavering support and understanding have been instrumental in sustaining me throughout the rigors of this academic endeavor. Her encouragement, patience, and unwavering belief in my capabilities have served as a constant source of motivation, imbuing me with the resilience and determination necessary to navigate the complexities of this research journey.

Furthermore, I wish to extend my gratitude to my supervisor, D. Sotiropoulos, whose assistance, direction, and mentorship facilitated the meticulous and professional execution of this project.

Abstract

This project is dedicated to the development of a Real-time Train Tracking and Anomaly Detection System tailored for railway operations optimization. At its core, the system harnesses the power of the Random Forest model, a versatile machine learning technique, to forecast estimated arrival times in train schedules. Grounded in historical arrival and departure data sourced from stations, the system employs GPS coordinates to meticulously compute train speeds, thus enabling precise estimations of travel times.

The dataset under scrutiny comprises datetime records of train movements to and from stations, a rich source of information pivotal for computing medium speeds. Preparing this dataset for model training necessitates an application of preprocessing and feature engineering techniques, ensuring data readiness and enhancing predictive accuracy.

Through a meticulous process of experimentation and evaluation, the system's performance is rigorously scrutinized. Its efficacy in furnishing real-time insights into train operations and adeptly predicting delays emerges palpable. Despite grappling with challenges inherent to real-world data the system's robustness endures. Through continuous refinement and adaptation, the system proves itself solvent in optimizing train operations and mitigating delays, ensuring seamless efficiency in real-world scenarios

Table of Contents

List Of Figures

List Of Tables

1 Introduction

Railway transportation serves as a lifeline for modern societies, facilitating the efficient movement of goods and people across vast distances. However, ensuring the smooth operation of railway networks presents a multifaceted challenge, requiring meticulous coordination and proactive management. At the heart of this endeavor lies the need for accurate train tracking and timely detection of anomalies to preempt disruptions and optimize resource allocation [1].

In response to these imperatives, this project endeavors to develop a Real-time Train Tracking and Anomaly Detection System tailored specifically for railway operations. By leveraging advanced machine learning techniques, particularly the Random Forest model, this system aims to predict estimated arrival times within train schedules with precision and reliability [2].

Central to the system's functionality is the utilization of historical arrival and departure data from stations, providing a rich foundation upon which predictive models can be trained. Complemented by GPS coordinates, these data enable the computation of train speeds, thereby facilitating accurate estimations of travel times. Through meticulous preprocessing and feature engineering, the dataset is refined to enhance its suitability for model training, ensuring robust predictive capabilities [3].

In the following sections, we delve into the methodologies employed in the development of this system, elucidating the intricacies of data processing, model training, and performance evaluation. Furthermore, we present our findings, highlighting the system's capabilities and discussing its implications for railway operations.

2 Theoretical Background

Railway transportation stands as a cornerstone of modern infrastructure, facilitating the movement of passengers and goods across vast distances with unparalleled efficiency. The punctuality and reliability of train arrivals are paramount for ensuring smooth operations and passenger satisfaction. Understanding the factors influencing train arrivals involves delving into various dimensions, ranging from historical trends to technological advancements and external influences [4].

2.1 Historical Trends and Global Context

Train arrivals reflect the culmination of extensive logistical planning and operational management within railway systems worldwide. According to data from the International Union of Railways (UIC), global rail passenger traffic has been steadily increasing, underscoring the growing importance of railway transportation in the global mobility landscape. Similarly, freight transportation by rail continues to play a pivotal role in facilitating trade and commerce on a global scale, with projections indicating sustained growth in the coming years [5].

2.2 Technological Advancements in Train Tracking

The advent of advanced technologies has revolutionized train tracking and scheduling processes, enabling railway operators to monitor train movements in real-time and optimize schedules for improved efficiency. Systems such as Automatic Train Control (ATC) and Positive Train Control (PTC) leverage advanced signaling and communication technologies to enhance safety and precision in train operations (Federal Railroad Administration, 2020). Furthermore, the integration of Global Positioning System (GPS) technology has facilitated more accurate tracking of train movements, enabling operators to provide real-time updates to passengers and stakeholders [6].

2.3 External Influences and Operational Challenges

The Train arrivals are subject to a myriad of external influences and operational challenges, ranging from weather conditions and track maintenance to regulatory constraints and unforeseen incidents. Adverse weather events, such as snowstorms or extreme heatwaves, can disrupt railway operations and lead to delays in train arrivals. Similarly, infrastructure maintenance and repair work can necessitate temporary schedule adjustments, impacting the timing of train arrivals. Moreover, regulatory requirements, such as safety inspections and crew scheduling regulations, can introduce additional complexities into the planning and execution of train operations [7].

2.4 Data Analytics and Predictive Model

In recent years, there has been a growing emphasis on harnessing the power of data analytics and predictive modeling to optimize train arrivals and enhance operational efficiency. Machine learning algorithms, such as Random Forest, Support Vector Regression, and Gradient Boosting, offer promising avenues for predicting train arrival times based on historical data and real-time inputs. By analyzing patterns in train movements, passenger demand, and external factors, these models can provide valuable insights for railway operators to proactively manage schedules and minimize delays [8].

2.5 Environmental Sustainability and Future Outlook

As the global community increasingly focuses on sustainability and environmental stewardship, the railway industry faces pressure to adopt eco-friendly practices and reduce its carbon footprint. Strategies such as electrification, energy-efficient train design, and modal shift from road to rail transport are being pursued to mitigate environmental impacts and promote sustainable mobility. Looking ahead, advancements in technology and data analytics are poised to further revolutionize train arrivals and transform railway operations, ushering in an era of smarter, more resilient transportation systems [9].

2.6 Machine Learning and Transportations

The integration of machine learning (ML) techniques within transportation systems has emerged as a transformative paradigm, revolutionizing various facets of the transportation industry. As the demand for efficient, sustainable, and reliable transportation solutions continues to grow, ML presents unprecedented opportunities to optimize operations, enhance safety, and improve overall user experience across diverse modes of transportation [10].

2.6.1 Evolution of Transportation Systems

Historically, transportation systems have played a fundamental role in shaping societal development and economic progress. From the advent of steam-powered locomotives to the proliferation of automobiles and the rise of air travel, transportation technologies have continuously evolved to meet the evolving needs of global mobility. In recent decades, the digital revolution has ushered in a new era of transportation innovation, characterized by the integration of advanced computing technologies, connectivity, and data-driven decision-making [11].

2.6.2 Challenges in Transportation Management

Despite significant advancements in transportation infrastructure and technology, the management and optimization of transportation systems remain fraught with challenges. Congestion, traffic accidents, infrastructure maintenance, and environmental sustainability are among the key challenges facing modern transportation networks. Addressing these challenges requires innovative approaches that leverage data, analytics, and automation to enhance system efficiency, resilience, and sustainability [12].

2.6.3 Role of Machine Learning in Transportation

Machine learning offers a powerful toolkit for addressing the complexities of transportation management and optimization. By leveraging vast amounts of data generated by transportation systems, ML algorithms can uncover patterns, trends, and insights that enable informed decision-making and predictive analytics. From traffic prediction and route optimization to demand forecasting and anomaly detection, ML techniques can unlock new capabilities for improving the performance and reliability of transportation systems [13].

2.6.4 Applications of Machine Learning in Transportation

The applications of ML in transportation are diverse and multifaceted. In the realm of traffic management, ML algorithms can analyze real-time traffic data from sensors, cameras, and GPS devices to predict traffic congestion, optimize signal timings, and recommend alternative routes for travelers. In public transportation, ML-powered systems can optimize scheduling, fleet management, and passenger routing to enhance service reliability and efficiency. Moreover, ML techniques are increasingly being deployed in autonomous vehicles, enabling self-driving cars, trucks, and drones to navigate safely and efficiently in dynamic environments [14].

2.6.5 Challenges and Opportunities

Despite the promise of ML in transportation, several challenges remain to be addressed. Data quality, privacy concerns, regulatory compliance, and ethical considerations are among the key challenges facing the widespread adoption of ML in transportation. Moreover, the integration of ML technologies into existing transportation infrastructure requires careful planning, investment, and collaboration among stakeholders. However, with the rapid advancements in ML algorithms, computing infrastructure, and data analytics capabilities, the potential for transformative innovation in transportation management is immense [15].

2.6.6 Future Directions

Looking ahead, the future of transportation is poised to be shaped by ongoing advancements in machine learning, artificial intelligence, and data science. The emergence of smart transportation systems, powered by ML algorithms, promises to revolutionize how people and goods are moved, while also addressing pressing societal challenges such as urban congestion. By embracing a data-driven approach to transportation management, policymakers, industry stakeholders, and researchers can unlock new opportunities for creating sustainable, efficient, and resilient transportation systems for future generations [16].

2.7 Performance Evaluation of a model

When assessing the predictive capacity of a regression-based model, a plethora of metrics exists for evaluation. In this study, the following metrics are being presented as some of the principal performance indicators for the evaluation models.

Mean Squared Error (MSE)

The Mean Squared Error (MSE) is a fundamental metric used to evaluate the predictive performance of regression models. It quantifies the average squared difference between the actual target values and the corresponding predicted values in the test dataset. By squaring each discrepancy and then averaging across all observations, the MSE provides a comprehensive measure of how well the model fits the data. Notably, this formulation penalizes larger deviations more heavily than smaller ones, offering valuable insight into the model's ability to accurately predict target values. The MSE equation is represented as follows:

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

Figure 1 : MSE equation

Where:

* **N** is the total number of observations in the dataset.
* **yi** represents the actual target value for the i-th observation.
* **y^i** represents the predicted target value for the i-th observation.

In essence, the MSE serves as a crucial tool for assessing the goodness-of-fit of regression models, providing valuable feedback on their predictive accuracy [17].

Root Mean Squared Error (RMSE)

The Root Mean Squared Error (RMSE) metric is a natural extension of the Mean Squared Error (MSE), designed to maintain measurement unit consistency with the target value. It follows the same underlying logic as MSE but involves taking the square root of the MSE output. This adjustment ensures that the resulting units align with those of the target variable. The RMSE equation is expressed as:

$$\mathrm{RMSE} = \sqrt{\tfrac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}$$

Figure 2 : RMSE equation

Where:

* **N** represents the total number of observations in the dataset.
* **yi** denotes the actual target value for the ii-th observation.
* **y^i** represents the predicted target value for the ii-th observation.

In essence, the RMSE provides a straightforward measure of the typical deviation between the actual and predicted values, enabling a comprehensive evaluation of the model's predictive performance [18].

Mean Absolute Error (MAE)

Another commonly used metric for evaluating regression output is the Mean Absolute Error (MAE), which represents the average positive difference between each actual target value and its corresponding predicted value. Unlike MSE and RMSE, MAE considers the absolute value of each difference, ensuring compatibility in its interpretation. This metric provides a straightforward measure of the typical deviation between the actual and predicted values, contributing to a comprehensive assessment of the model's predictive accuracy [19]. The MAE equation is formulated as:

$$\mathrm{MAE} = \tfrac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|$$

Figure 3 : MAE equation

Where:

* **N** denotes the total number of observations in the dataset.
* **yi** represents the ii-th actual target value.
* **y^i** represents the ii-th predicted target value.

3 Literature Review

Real-time train tracking systems have become essential tools for railway operators to monitor and manage train movements efficiently. These systems leverage advanced technologies such as GPS, RFID, and sensors to provide accurate and real-time location information of trains along the railway network [20].

3.1 GPS-Based Tracking Systems

GPS-based vehicle tracking systems have gained prominence in recent years for their ability to provide precise location data of trains. Studies have demonstrated the effectiveness of GPS technology in enabling real-time tracking of trains, allowing for improved scheduling, operational efficiency, and passenger information systems.

3.2 RFID and Sensor Networks

In addition to GPS, RFID and sensor networks have been utilized for real-time train tracking. Researches explored the deployment of RFID tags and sensor networks on trains and railway infrastructure to monitor train movements, detect anomalies, and ensure safety and security along the railway lines.

Anomaly Detection in Railway Systems

Anomaly detection plays a critical role in ensuring the safety, reliability, and efficiency of railway systems. Detecting anomalies such as unreasonable delay, equipment failures, track defects, and abnormal train behaviors in real-time can prevent accidents, minimize disruptions, and optimize maintenance schedules [21].

3.3 Machine Learning Techniques for Anomaly Detection

Machine learning techniques have been widely adopted for anomaly detection in railway systems. Studies have investigated the use of machine learning algorithms with Random Forest for identifying anomalies in many operations and thus it can be used to predict train's arrivals.

3.4 Deep Learning Approaches

Recent advancements in deep learning have also been applied to anomaly detection in network systems. Studies [42] have explored the use of Graph Neural Networks (GNNs) for predicting delays and enhancing the performance of a network.

Challenges and Future Directions

While significant progress has been made in the development of real-time train tracking and anomaly detection systems, several challenges remain to be addressed. These include the integration of heterogeneous data sources, ensuring interoperability between different railway systems, and addressing cybersecurity concerns associated with data transmission and processing.

Future research directions in this field include the exploration of edge computing and fog computing technologies for real-time data processing, the development of hybrid anomaly detection models combining machine learning and physics-based approaches, and the implementation of blockchain-based solutions for secure and transparent data sharing among railway stakeholders [22].

4 Defining the Problem

In the realm of transportation logistics, the accurate prediction of train arrival times between stations stands as a formidable challenge with significant operational implications. Addressing this challenge requires a multifaceted approach that blends domain expertise with cutting-edge machine learning techniques. In this study, the primary objective was to leverage machine learning methodologies to predict train arrival times reliably, thereby enhancing operational efficiency and passenger satisfaction [23].

4.1 Problem Identification and Scope

The core problem revolved around predicting train arrival times based on departure and historical data, with the ultimate goal of minimizing arrival delays. The dataset provided contained limited information, primarily comprising departure and arrival timestamps for various train journeys between stations. To tackle this problem effectively, several key steps were undertaken to extract relevant features and develop predictive models [24].

4.2 Estimating Arrival Time and Deflection Computation

A pivotal step in the analysis involved estimating the arrival time at each station, given only departure timestamps and historical journey data. This was achieved by computing the median arrival times for each station pair, thereby creating a baseline prediction model. Additionally, the routes, or station sequences, for the train network were inferred from historical data, providing essential context for understanding journey patterns and potential delays [25].

Following the estimation of arrival times, the deflection between predicted and actual arrival times was computed. Deflection, represented as the difference between predicted and actual arrival times, served as the target variable for the machine learning models. Positive deflections indicated delays in arrival, while negative deflections represented instances where the train arrived earlier than predicted [26].

4.3 Feature Engineering and Model Selection

With the target variable defined, the focus shifted to identifying features that could effectively predict arrival delays. Extensive feature engineering was conducted, encompassing variables such as weather conditions, time of day, station congestion, historical performance, and train type. Through rigorous analysis and experimentation, features with the highest predictive power were selected for inclusion in the models [27].

The selection of an appropriate machine learning model was crucial in this context. Several algorithms, including linear regression [28], random forest [29], gradient boosting [30], and neural networks [31], were evaluated based on their performance metrics and suitability for the task.

4.4 Conclusion

In conclusion, the process of predicting train arrival times represents a complex yet rewarding endeavor at the intersection of transportation logistics and data science. By leveraging machine learning techniques and innovative approaches to feature engineering, it becomes possible to develop accurate and robust predictive models capable of anticipating arrival delays and optimizing operational efficiency. This study underscores the importance of data-driven decision-making in transportation management [32].

5 Description of Available Data

The dataset utilized in this study comprises 90,000 records obtained from a reputable company specializing in train tracking solutions. Due to confidentiality agreements, the name and specific operations of the company cannot be disclosed. However, it is crucial to note that the dataset encompasses comprehensive information related to the timestamps of train arrivals and departures. The dataset's vast size and rich granularity provide a robust foundation for conducting in-depth analysis and developing predictive models to enhance railway operations' efficiency and reliability.

5.1 Time coverage

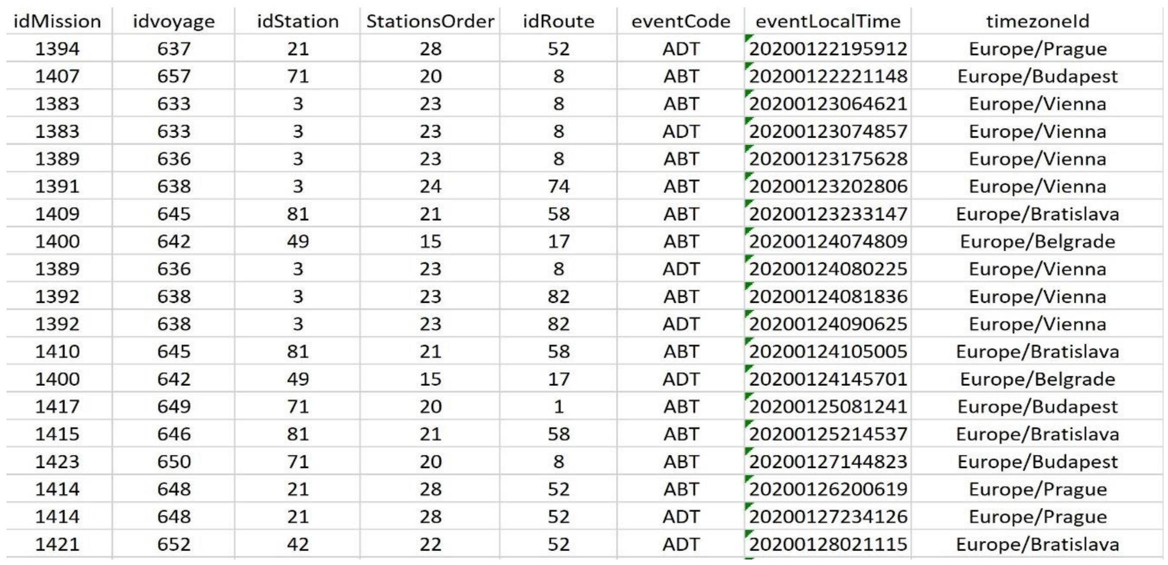
The dataset covers a time span from 2020 to 2023, providing a comprehensive snapshot of train departures and arrivals during this period. This timeframe allows for the analysis of route time series, prediction of arriving time, and potential anomalies in train operations, contributing to a holistic understanding of railway system dynamics.

5.2 Available Dataset : Arrivals and Departures

The dataset comprises real-time tracking information extracted from train operations. Each record corresponds to a specific event or action associated with a train mission, providing insights into its journey and operational dynamics. Here's a breakdown of the dataset:

1. idMission: A unique identifier for each train mission, indicating a distinct instance of train movement or operation.
2. idvoyage: An identifier associated with each voyage, representing a series of connected train missions or a specific route.
3. idStation: An identifier assigned to each station along the train route where the event occurred.
4. StationsOrder: Indicates the order of the station within the route sequence, providing context on the sequence of station visits during the voyage.
5. idRoute: An identifier representing the specific route followed by the train during its journey.
6. eventCode: A code indicating the type of event or action associated with the train, such as arrival (ABT) or departure (ADT) at a station.
7. eventLocalTime: The local time at which the event occurred, represented in a timestamp format (YYYYMMDDHHMMSS).
8. timezoneId: Denotes the timezone in which the event occurred, providing geographical context to the station's location and event timestamp.

This dataset facilitates the analysis of train movements, station interactions, and event occurrences over time, serving as valuable input for real-time tracking systems and operational management in railway transportation.



idMission	idvoyage	idStation	StationsOrder	idRoute	eventCode	eventLocalTime	timezoneId
1394	637	21	28	52	ADT	20200122195912	Europe/Prague
1407	657	71	20	8	ABT	20200122221148	Europe/Budapest
1383	633	3	23	8	ABT	20200123064621	Europe/Vienna
1383	633	3	23	8	ADT	20200123074857	Europe/Vienna
1389	636	3	23	8	ABT	20200123175628	Europe/Vienna
1391	638	3	24	74	ABT	20200123202806	Europe/Vienna
1409	645	81	21	58	ABT	20200123233147	Europe/Bratislava
1400	642	49	15	17	ABT	20200124074809	Europe/Belgrade
1389	636	3	23	8	ADT	20200124080225	Europe/Vienna
1392	638	3	23	82	ABT	20200124081836	Europe/Vienna
1392	638	3	23	82	ADT	20200124090625	Europe/Vienna
1410	645	81	21	58	ABT	20200124105005	Europe/Bratislava
1400	642	49	15	17	ADT	20200124145701	Europe/Belgrade
1417	649	71	20	1	ABT	20200125081241	Europe/Budapest
1415	646	81	21	58	ABT	20200125214537	Europe/Bratislava
1423	650	71	20	8	ABT	20200127144823	Europe/Budapest
1414	648	21	28	52	ABT	20200126200619	Europe/Prague
1414	648	21	28	52	ADT	20200127234126	Europe/Prague
1421	652	42	22	52	ADT	20200128021115	Europe/Bratislava

Table 1 : Initial Dataset

6 Experimental Methodology

6.1 Data Processing & Analysis

6.1.1 Sorting and Re-Arranging the Dataset

The experimental methodology for data processing and analysis involves a comprehensive series of steps designed to transform raw train tracking data into a refined dataset suitable for in-depth analysis and modeling. These steps are critical for extracting meaningful insights, identifying patterns, and ultimately optimizing train tracking and anomaly detection systems. Below is a detailed description of the key processes involved:

Sorting and Organization:

The initial phase of data processing entails sorting the raw train tracking data based on multiple criteria such as mission ID, voyage ID, and event timestamp. This meticulous organization ensures that the dataset is structured chronologically, enabling seamless tracking of train movements over time. By organizing the data in this manner, it becomes easier to identify temporal trends, detect anomalies, and analyze the efficiency of train operations.

Enhancing Event Codes:

Following data sorting, attention is turned towards refining the event codes within the dataset to provide clearer indications of train departures and arrivals. This involves renaming the 'eventCode' column to 'departure' and standardizing its values to binary representations (1 for departure, 0 for arrival). By transforming event codes in this manner, the dataset gains enhanced interpretability, facilitating more nuanced analysis of train movement patterns and operational dynamics.

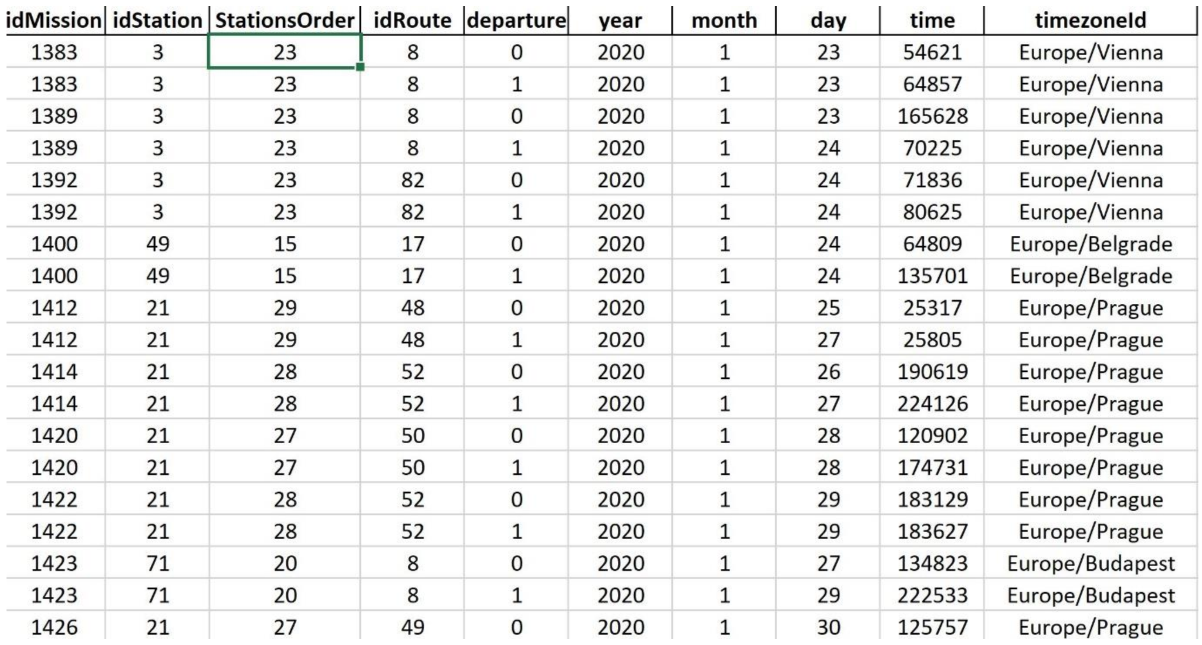
Column Renaming, Creation, and Rearrangement:

A critical aspect of data preprocessing involves restructuring the dataset by renaming, creating, and rearranging columns to improve clarity and analytical utility. New columns, such as 'departedDaytime', are introduced to capture additional contextual information, such as the time of day when departures occur. Unnecessary columns are removed, and derived columns, such as 'dateTimeArr', are recalculated based on existing data. This strategic reorganization ensures that the dataset is optimized for subsequent analysis and modeling tasks.

Data Type Conversion and Standardization:

To facilitate comprehensive analysis, certain columns' data types, particularly those related to time (e.g., 'travelTime', 'estTravTime', 'arrDelay', 'depDelay'), are converted into a standardized format, typically hours. This conversion enhances data consistency and enables seamless integration with analytical tools and algorithms. By standardizing data types, the dataset becomes more conducive to exploratory analysis, statistical modeling, and anomaly detection algorithms.

Through these meticulous data processing and analysis procedures, the raw train tracking data undergoes a transformative journey, evolving from disparate observations into a cohesive, structured dataset ripe for advanced analytical techniques. This methodology sets the stage for uncovering actionable insights, optimizing operational efficiencies, and ultimately enhancing the performance of real-time train tracking and anomaly detection systems.



idMission	idStation	StationsOrder	idRoute	departure	year	month	day	time	timezoneId
1383	3	23	8	0	2020	1	23	54621	Europe/Vienna
1383	3	23	8	1	2020	1	23	64857	Europe/Vienna
1389	3	23	8	0	2020	1	23	165628	Europe/Vienna
1389	3	23	8	1	2020	1	24	70225	Europe/Vienna
1392	3	23	82	0	2020	1	24	71836	Europe/Vienna
1392	3	23	82	1	2020	1	24	80625	Europe/Vienna
1400	49	15	17	0	2020	1	24	64809	Europe/Belgrade
1400	49	15	17	1	2020	1	24	135701	Europe/Belgrade
1412	21	29	48	0	2020	1	25	25317	Europe/Prague
1412	21	29	48	1	2020	1	27	25805	Europe/Prague
1414	21	28	52	0	2020	1	26	190619	Europe/Prague
1414	21	28	52	1	2020	1	27	224126	Europe/Prague
1420	21	27	50	0	2020	1	28	120902	Europe/Prague
1420	21	27	50	1	2020	1	28	174731	Europe/Prague
1422	21	28	52	0	2020	1	29	183129	Europe/Prague
1422	21	28	52	1	2020	1	29	183627	Europe/Prague
1423	71	20	8	0	2020	1	27	134823	Europe/Budapest
1423	71	20	8	1	2020	1	29	222533	Europe/Budapest
1426	21	27	49	0	2020	1	30	125757	Europe/Prague

Table 2 : Dataset After Sorting and Re-Arrangements

6.1.2 Timezone Synchronization and Localization to UTC

In that part, the code snippet addresses the issue of time zone discrepancies within the dataset by synchronizing event timestamps to a common reference point, Coordinated Universal Time (UTC). This process is crucial for ensuring consistency and accuracy in subsequent analyses, especially when dealing with data sourced from different geographical regions with varying time zones.

Timezone Adjustment:

Initially, the 'eventLocalTime' column, which presumably contains timestamp information in a specific format ('%Y%m%d%H%M%S'), is converted to datetime objects and localized to UTC using the pd.to_datetime() and tz_localize() functions from the Pandas library. This step sets a uniform timezone reference for all timestamps in the dataset.

Timezone-Specific Adjustment:

Following localization to UTC, the code accounts for specific time zone differences between locations represented in the dataset. Notably, locations such as Sofia and Athens are identified as being one hour ahead of other time zones. Therefore, a distinction is made between time zones requiring adjustment and those exempt from it.

Iterative Time Adjustment:

To synchronize timestamps accurately, the code iterates over each unique time zone present in the dataset. For time zones not exempt from adjustment (i.e., those other than 'Europe/Athens' and 'Europe/Sofia'), the corresponding timestamps are decremented by one hour. This adjustment effectively aligns the event timestamps with the UTC reference, ensuring uniformity across all records. By implementing this timezone synchronization methodology, the code promotes consistency and accuracy in timestamp representation, thereby facilitating robust data analysis and interpretation. This step is particularly crucial in scenarios where disparate datasets with varying time zones are integrated or analyzed together.

6.1.3 Refining Train Departure Patterns for Insightful Analysis

In order to establish a consistent pattern of departures for each mission, a systematic approach was devised to categorize departure events within the dataset. Specifically, for every mission recorded, a binary pattern of departures, alternating between 0s and 1s, was generated. This pattern served as a fundamental indicator of departure events, with 0 denoting train departures and 1 representing arrivals. However, due to the presence of anomalies and inconsistencies within the dataset, additional preprocessing and clearance measures were deemed necessary.

Subsequently, two distinct DataFrames were created: one comprising departure events labeled as "0" and another containing departure events labeled as "1". By segregating the dataset based on departure values, it became feasible to conduct separate analyses on arrival and departure events, thereby facilitating more targeted insights into train movement dynamics.

To further refine the dataset and ensure coherence, data anomalies were identified and addressed through rigorous processing techniques. Anomalies encompassed irregularities such as duplicate entries, missing values, and erroneous timestamps. Through meticulous data cleansing procedures, these anomalies were rectified, thereby enhancing the overall integrity and reliability of the dataset.

Upon completion of data preprocessing and clearance, the segregated DataFrames containing departure events were merged side by side. This amalgamation facilitated a comprehensive examination of train movements, with arrival and departure events juxtaposed for each station. Such a structured arrangement enabled a nuanced understanding of train operations, providing insights into station-wise arrival and departure patterns.

Overall, this systematic approach to data preprocessing and clearance not only laid the groundwork for subsequent analyses but also underscored the importance of meticulous data management in ensuring the robustness and accuracy of analytical outcomes.

6.1.4 Refining Actions

- **Delete Missions with Only One Record:**

1. Identify missions that have only one record (i.e., a single departure event). Such missions are considered insufficient for meaningful analysis and are thus marked for deletion.
2. Remove all rows associated with missions identified for deletion, effectively eliminating missions with only one record from the dataset.

- **Delete First Row of Mission if Departure is 1:**

For missions with multiple records, if the first recorded departure is marked as 1 (indicating a departure event), it suggests an incomplete sequence of events. Thus, the first row of such missions is removed to maintain consistency in departure sequences.

- **Delete Last Row of Mission if Departure is 0:**

Similarly, for missions with multiple records, if the last recorded departure is marked as 0 (indicating an arrival event), it implies an incomplete sequence of events. Consequently, the last row of such missions is eliminated to ensure coherence in departure sequences.

- **Fixing Missions with only two records:**

Similarly, for missions with multiple records, if the last recorded departure is marked as 0 (indicating an arrival event), it implies an incomplete sequence of events. Consequently, the last row of such missions is eliminated to ensure coherence in departure sequences.

1. Identify Missions with Only Two Records: It first identifies missions that have exactly two records by counting the occurrences of each mission ID (idMission) in the dataset. Missions with only two records are stored in the variable onlyTwoMissions.
2. Check for Consistency in Station IDs: For each mission identified with only two records (onlyTwoMissions), it checks if both records have the same idStation values.
If the two records have different station IDs, it implies inconsistencies in the data, and the rows associated with such missions are deleted from the DataFrame (df).
3. Verify Departure Sequence:
For each mission with only two records, it checks if the departure sequence is correct.
It ensures that the first record indicates an arrival (departure = 0) and the second record indicates a departure (departure = 1). If the departure sequence is incorrect, meaning either the first record is a departure or the second record is an arrival, the rows associated with such missions are removed from the DataFrame (df).

- **Removing Missions starting with departure and ending with arrival:**

This implementation serves to remove missions from the dataset where the departure sequence begins with a departure (1) and ends with an arrival (0) or vice versa, which violates the expected departure sequence pattern. Here's an overview of what the function does:

1. **Identify Incorrect Missions:**
It iterates over each unique mission ID in the DataFrame and examines the departure sequence for each mission. If the departure sequence starts with a departure (1) and ends with an arrival (0) or vice versa, it marks the mission as incorrect.
2. **Create DataFrame of Incorrect Missions:**
It creates a DataFrame containing rows associated with the identified incorrect missions.
3. **Remove Incorrect Missions:**
It removes rows associated with the incorrect missions from the original DataFrame.
4. **Recursion for Additional Checks:**
If there are still incorrect missions after the initial removal, the function is recursively called to perform additional checks and remove any remaining incorrect missions.

This process ensures that the dataset maintains the expected departure sequence pattern for each mission, enhancing data consistency and integrity.

- **Removing Second Record from a departure sequence of 0,1, 1 (Same Station Id):**

This implementation aims to remove rows from the DataFrame `df` where the departure sequence follows the pattern 0, 1, 1. Here's a breakdown of what the function does:

1. **Iteration over Missions:**

It iterates over each unique mission ID in the DataFrame.

2. **Check for Sequence 0, 1, 1:**

Within each mission, it iterates over the rows, excluding the last two rows. For each iteration, it checks if the departure values of the current row and the next two rows form a sequence of 0, 1, 1

3. **Deletion of Second Row from Sequence:**

If a sequence of 0, 1, 1 is found, it deletes the second row of the sequence from the original DataFrame df.

4. **Return Modified DataFrame:**

After processing all missions, it returns the modified DataFrame with rows removed as necessary.

In summary, this function helps maintain the integrity of the departure sequence by removing rows that deviate from the expected pattern, ensuring consistency in the dataset.

- **Removing First Record from a departure sequence of 0,0,1 (Same Station Id):**

This implementation aims to remove rows from the DataFrame `df` where the departure sequence follows the pattern 0, 0, 1. Here's a breakdown of what the function does:

1. **Iteration over Missions:**

It iterates over each unique mission ID in the DataFrame.

2. **Check for Sequence 0, 0, 1:**

Within each mission, it iterates over the rows, excluding the last two rows. For each iteration, it checks if the departure values of the current row and the next two rows form a sequence of 0, 0, 1

3. **Deletion of First Row from Sequence:**

If a sequence of 0, 0, 1 is found, it deletes the second row of the sequence from the original DataFrame df.

4. **Return Modified DataFrame:**

After processing all missions, it returns the modified DataFrame with rows removed as necessary.

In summary, this function helps maintain the integrity of the departure sequence by removing rows that deviate from the expected pattern, ensuring consistency in the dataset.

- **Removing Record from a departure sequence of 0,0,1 (Different Station Id):**

This implementation serves to remove rows from a DataFrame that deviate from a specific departure sequence pattern while also considering the station information associated with each row. Here's an explanation of what it does:

1. **Iteration over Missions:**

It iterates over each unique mission ID in the DataFrame.

1. **Check for Sequence 0, 0, 1:**

Within each mission, it iterates over the rows, excluding the last two rows. For each iteration, it checks if the departure values of the current row and the next two rows form a sequence of 0, 0, 1

2. **Check for Different Stations:**

If a sequence of 0, 0, 1 is found, it checks if the stations associated with these rows are distinct. It counts the number of unique stations within the sequence of three rows. If there are exactly two unique stations, it proceeds to the next step. Otherwise, it continues searching.

3. **Deletion of Row with Single Occurrence of a Station:**

It identifies the station that appears only once within the sequence of three rows. It deletes the row where the station appears only once from the original DataFrame df.

4. **Return Modified DataFrame:**

After processing all missions, it returns the modified DataFrame with rows removed as necessary.

This process ensures that the departure sequence pattern of 0, 0, 1 is maintained while considering the station information and removing any rows that do not adhere to these criteria.

- **Delete Rest Missions with Wrong Pattern :**

This process identifies and removes missions from the DataFrame where the departure sequence pattern does not match the expected pattern (0,1,0,1,0,1,0,…..,1). Here's how it works:

1. Identifying Missions with Incorrect Patterns:
It iterates over each unique mission ID in the DataFrame. For each mission, it extracts the corresponding departure values. It compares the extracted departure values with the expected pattern, which is alternating between 0 and 1 (e.g., [0, 1, 0, 1, ...]). If the departure values of a mission do not match the expected pattern, the mission DataFrame is added to the list `missions_with_pattern_breaks`, and the count is incremented.

2. Dropping Rows of Missions with Incorrect Patterns:
After identifying all missions with incorrect departure patterns, it iterates over each DataFrame in `missions_with_pattern_breaks`. It retrieves the indexes of rows corresponding to the current mission DataFrame. It drops these rows from the original DataFrame `df`.

3. Resetting DataFrame Index:
After dropping rows, it resets the index of the DataFrame to ensure continuous indexing.

4. Returning Modified DataFrame:
Finally, it returns the modified DataFrame with rows of missions with incorrect patterns removed.

This process ensures that only missions with the correct departure sequence pattern are retained in the DataFrame, maintaining data integrity and consistency.

6.1.5 Breaking in two Data Frames and Merging

The merging process, conducted through the `mergeDataset(df)` function, is a critical component of the data processing pipeline, serving to consolidate departure and arrival records within the same mission into a unified dataset. Upon receiving the input DataFrame `df`, the function embarks on a sequence of operations designed to organize the data for further analysis.

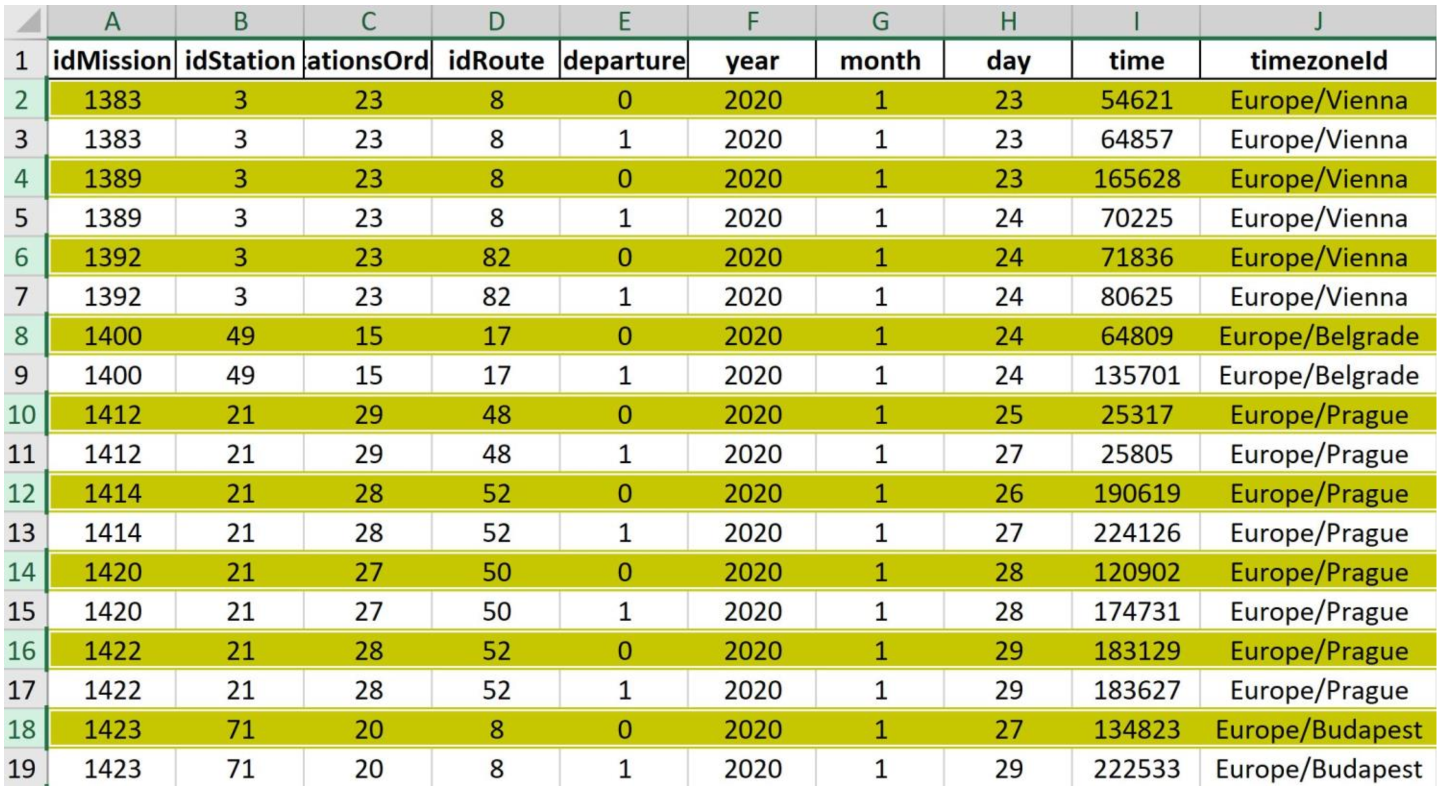
Initially, the input DataFrame undergoes partitioning into two distinct DataFrames based on departure values. One DataFrame, `df_departure_0`, contains records with departures labeled as 0, while the other, `df_departure_1`, holds those labeled as 1. This segregation lays the groundwork for aligning departure and arrival records, facilitating their subsequent merging.

Following partitioning, both `df_departure_0` and `df_departure_1` undergo an index reset to ensure consistent row indexing. This step is essential for maintaining the integrity of the data structure and enabling seamless concatenation in subsequent stages. Additionally, the function renames columns in `df_departure_1` to mitigate potential conflicts during concatenation. Each column in `df_departure_1` is suffixed with ".1" to indicate its association with the second segment of each mission, thereby enhancing clarity and avoiding ambiguity in the merged dataset.

The core of the merging process occurs as `df_departure_0` and `df_departure_1` are horizontally concatenated using the `pd.concat()` function, resulting in the creation of a combined DataFrame, `df_combined`. Each row in `df_combined` corresponds to a pair of consecutive records within the same mission, with one record representing a departure (labeled as 0) and the other an arrival (labeled as 1). This amalgamation of departure and arrival records into cohesive pairs lays the foundation for comprehensive mission-level analysis and interpretation.

To maintain data consistency and accuracy, `mergeDataset(df)` incorporates a validation step. Here, a mask, `mask_different_values`, is generated to identify rows where `idStation` values differ between consecutive departure and arrival records. Rows with such discrepancies are subsequently removed from `df_combined`, retaining only those pairs of consecutive records with matching station IDs. This validation criterion ensures data integrity and mitigates the risk of inconsistencies or errors in subsequent analyses.

Upon completion of the merging process, the index of the resulting combined DataFrame, `df_combined`, is reset to ensure continuity and consistency in row indexing. This final step ensures that the merged dataset is well-structured and primed for comprehensive analysis, laying a solid foundation for deriving insights and conclusions from the data. In summary, the `mergeDataset(df)` function plays a pivotal role in harmonizing departure and arrival records, enabling coherent and insightful analysis of mission-level data.



	A	B	C	D	E	F	G	H	I	J
1	idMission	idStation	ationsOrd	idRoute	departure	year	month	day	time	timezoneId
2	1383	3	23	8	0	2020	1	23	54621	Europe/Vienna
3	1383	3	23	8	1	2020	1	23	64857	Europe/Vienna
4	1389	3	23	8	0	2020	1	23	165628	Europe/Vienna
5	1389	3	23	8	1	2020	1	24	70225	Europe/Vienna
6	1392	3	23	82	0	2020	1	24	71836	Europe/Vienna
7	1392	3	23	82	1	2020	1	24	80625	Europe/Vienna
8	1400	49	15	17	0	2020	1	24	64809	Europe/Belgrade
9	1400	49	15	17	1	2020	1	24	135701	Europe/Belgrade
10	1412	21	29	48	0	2020	1	25	25317	Europe/Prague
11	1412	21	29	48	1	2020	1	27	25805	Europe/Prague
12	1414	21	28	52	0	2020	1	26	190619	Europe/Prague
13	1414	21	28	52	1	2020	1	27	224126	Europe/Prague
14	1420	21	27	50	0	2020	1	28	120902	Europe/Prague
15	1420	21	27	50	1	2020	1	28	174731	Europe/Prague
16	1422	21	28	52	0	2020	1	29	183129	Europe/Prague
17	1422	21	28	52	1	2020	1	29	183627	Europe/Prague
18	1423	71	20	8	0	2020	1	27	134823	Europe/Budapest
19	1423	71	20	8	1	2020	1	29	222533	Europe/Budapest

Table 3 : Records to be merged side to side

6.2 Data Analysis

6.2.1 Analyzing Waiting and Travelling Times in Train Missions

This process (computeWaitingAndTravellingTime function) encapsulates a critical step in the analysis of train missions by meticulously computing waiting and travelling times for each segment of a journey. This function operates on a DataFrame (df) containing departure and arrival data, station information, and mission identifiers. Through a series of meticulously orchestrated steps, it extracts insights crucial for optimizing transportation logistics.

Understanding the Functionality:
At its core, the function is designed to extract actionable insights from raw mission data, empowering transportation analysts to make informed decisions and optimize mission scheduling. By breaking down the operation of this function, we gain a deeper understanding of its significance within the realm of transportation analytics.

Data Preparation:
The initial phase of the function involves data cleansing and preparation. Redundant columns, including additional departure-related fields from the arrival dataset, are removed to streamline the subsequent computation process. This step ensures that the analysis focuses solely on the relevant attributes required for calculating waiting and travelling times.
Feature Engineering:
Two pivotal features, waitingTime and travellingTime, are introduced to the DataFrame to capture the temporal dynamics of train missions. These features serve as key indicators of operational efficiency, enabling the identification of areas for improvement and optimization.

Waiting Time Computation:
The computation of waiting time is a crucial aspect of mission analysis, providing insights into the efficiency of station operations and potential bottlenecks in the mission schedule. By subtracting the arrival timestamp from the departure timestamp for each mission segment, the function accurately quantifies the duration trains spend waiting at each station.

Travelling Time Calculation:
Travelling time, another critical metric in mission analysis, sheds light on the temporal dynamics of inter-station travel. Through meticulous computation, the function determines the duration between the departure from one station and the arrival at the subsequent station. This information is invaluable for optimizing route planning and minimizing overall mission duration.

Next Station Assignment:
Assigning the next station for each mission segment facilitates the seamless organization of mission data and enables analysts to visualize mission trajectories with precision. By leveraging departure station information from subsequent segments, the function constructs a cohesive representation of mission sequences, aiding in the identification of delays and deviations from the planned route.

Data Transformation and Integrity:
The final stages of the function encompass data transformation and integrity checks, ensuring that the processed DataFrame maintains consistency and accuracy. Handling missing or null values and

reorganizing the DataFrame to adhere to standardized formats are critical steps in preparing the data for further analysis and visualization.

Conclusion:
In conclusion, the computeWaitingAndTravellingTime function serves as a cornerstone in the analysis of train mission data, offering unparalleled insights into waiting and travelling times. By meticulously processing and analyzing mission data, we can unlock new avenues for operational optimization, enhancing the efficiency and reliability of train services.

6.2.2 Analyzing Time Series Fore Each Route

The following process (computeTimeSeriesForEachRoute function) is a pivotal component in the analysis of train mission data, providing insights into the temporal dynamics of travel between stations along various routes. Operating on a DataFrame (df) containing mission data and a list of unique ordered station sequences (unique_ordered_station_lists), this function orchestrates a series of operations to compute and visualize time series data for each route.

Understanding the Functionality:
At its core, the function is designed to extract actionable insights from raw mission data, empowering transportation analysts to optimize route planning and enhance operational efficiency. By delving into the intricacies of this function, we gain a comprehensive understanding of its significance in transportation analytics.
Data Preparation and Sorting:
The function commences by sorting the input DataFrame (df) based on route and station order, laying the groundwork for subsequent computations. This initial step ensures that the mission data is organized in a structured manner, facilitating efficient processing and analysis.

Route Completion:
To facilitate comprehensive route analysis, the function iterates through each route, appending the last station to the station list if it does not already exist. This ensures that each route's station sequence is complete, enabling accurate computation of travel times between successive stations.

Time Series Computation:
The crux of the function lies in computing time series data for each route, capturing the median travel time between pairs of stations. Leveraging the ordered station lists and mission data, the function meticulously calculates the median travel time for each station pair, providing valuable insights into journey durations and route efficiency.

Data Visualization and Export:
Upon computing time series data for each route, the function constructs a new DataFrame (result_df) containing route identifiers, station pairs, and median travel times. This structured representation of time series data facilitates visualization and further analysis.

Excel Export and Visualization:
The function leverages the centerAddFilterAndProduceXlsx module to produce an Excel spreadsheet (timeSeriesForEachRoute.xlsx) containing the computed time series data. By centralizing the data and adding filter capabilities, transportation analysts can seamlessly explore and visualize route-specific time series data, enabling informed decision-making and strategic planning.

Conclusion:
In conclusion, the computeTimeSeriesForEachRoute function serves as a cornerstone in the analysis of train mission data, offering invaluable insights into route-specific time series dynamics. By meticulously processing and analyzing mission data, transportation we can optimize route planning, minimize travel times, and enhance passenger experiences. As transportation networks continue to evolve, functions like computeTimeSeriesForEachRoute play an indispensable role in shaping the future of public transit and commuter mobility.

6.2.3 Analyzing Time Deflections In Train Missions

This stage (computeDeflections function), alongside its auxiliary functions computeTravellingTimeDeflections and computeWaitingTimeDeflections, forms a crucial component in the data processing pipeline for analyzing train mission data. By computing deflections in travelling and waiting times, this suite of functions offers valuable insights into the efficiency and reliability of train operations, facilitating data-driven decision-making and operational optimization.

Understanding the Functionality:
At its core, the computeDeflections function orchestrates the computation of deflections in both travelling and waiting times, leveraging the underlying mission data to uncover patterns and anomalies indicative of operational inefficiencies. Complementing this overarching function, computeTravellingTimeDeflections and computeWaitingTimeDeflections focus on specific aspects of mission data, enabling a granular analysis of time deflections and their underlying drivers.

Computing Travelling Time Deflections:
The computeTravellingTimeDeflections function begins by sorting the mission data based on route and station order, establishing a structured foundation for subsequent computations. By iteratively analyzing station pairs within each route, the function calculates median travel times and identifies discrepancies between estimated and actual travel durations. Leveraging pandas' powerful data manipulation capabilities, this function offers a nuanced understanding of travel time dynamics, shedding light on potential bottlenecks and inefficiencies in train operations.

Computing Waiting Time Deflections:
Similarly, the computeWaitingTimeDeflections function delves into the realm of waiting time dynamics, exploring variations between estimated and actual wait durations at each station. By aggregating and analyzing waiting time data across mission instances, this function unveils patterns and trends indicative of service reliability and passenger experience. Through meticulous data processing and analysis, the function provides transportation stakeholders with actionable insights to enhance operational efficiency and customer satisfaction.

Excel Export and Visualization:
Both computeTravellingTimeDeflections and computeWaitingTimeDeflections functions culminate in the production of Excel spreadsheets (timeSeriesForEachRoute.xlsx and waitingTimeInEachStation.xlsx, respectively), facilitating data visualization and further analysis. By centralizing time deflection data and incorporating filter capabilities, these Excel spreadsheets empower transportation analysts to explore and visualize mission data, enabling informed decision-

making and strategic planning.

Conclusion:
In conclusion, the computeDeflections function, alongside its auxiliary counterparts, serves as a linchpin in the analysis of train mission data, offering valuable insights into time deflections and operational dynamics. By leveraging advanced data processing techniques and visualization tools, these functions empower the optimization of route planning, minimizing delays, and enhancing the overall transportation process.

6.3 Enriching Pipeline for Transport Analytics

6.3.1 Processing Deflections and Integration of Border Crossing Metrics

Initially, the function converts relevant time-related columns (travelTime, estTravTime, travTimeDefl, waitTimeDefl) to the appropriate data type to ensure consistency and accuracy in subsequent calculations. This preprocessing step lays the groundwork for accurate computation and interpretation of passing border metrics.

Subsequently, the function calculates the deflection in travel time (travTimeDefl) based on a comparison between actual travel time and estimated travel time. Instances where the actual travel time is less than the estimated travel time are flagged, and the deflection is negated to indicate instances of accelerated travel, often indicative of crossing borders.

Moreover, the function computes the deflection in waiting time (waitTimeDefl) to capture deviations between actual and estimated waiting times. By quantifying these deviations, the function provides valuable insights into potential delays at border crossings, aiding in the identification of inefficiencies and optimization opportunities in transportation systems.

To further enhance the dataset's granularity, the function introduces a new column passingBorder, initialized with default values. Leveraging group-wise operations, the function identifies instances where the timezone of the next row differs within each mission group, indicative of border crossings. These instances are flagged by setting the passingBorder column to 1, providing a binary indicator of border crossing events within the dataset.

Overall, the addPassingBorderMetric function significantly augments transportation datasets by incorporating passing border metrics, enabling comprehensive analysis and interpretation of cross-border travel dynamics. Through accurate computation and integration of border crossing indicators, the function empowers analysts to gain deeper insights into cross-border transport patterns and optimize operational strategies accordingly.

6.3.2 Augmenting Data Frame with Day and Weekend Metrics

The `addWeekendAndDayMetrics` function serves to augment a DataFrame (`df`) containing departure and arrival data with additional metrics related to the day of the week and weekend classification. This function is instrumental in enhancing the dataset for further analysis, particularly in scenarios where understanding temporal patterns is crucial, such as transportation logistics or scheduling optimization. Here's a breakdown of the process encapsulated within this function:

1. Convert to DateTime:
The function begins by converting the 'dateTimeDep' column of the DataFrame to datetime objects. This step ensures that the date and time information is correctly interpreted for subsequent analysis.

2. Add Day Metrics:
The `addDayMetrics` function is then called to enrich the DataFrame with a new column labeled 'day_of_week', which stores the day of the week corresponding to each departure timestamp. This addition allows for granular analysis based on the specific day of the week.

3. Add Weekend Metrics:
Next, the `addWeekendMetrics` function is invoked to further expand the dataset with a binary indicator column named 'isWeekend'. This column is populated with a value of 1 for Saturday and Sunday, signifying weekend days, and 0 for all other days of the week.

By systematically incorporating these additional metrics, the `addWeekendAndDayMetrics` function empowers analysts and data scientists to gain deeper insights into temporal patterns and variations within the dataset. This enriched dataset becomes valuable for tasks such as identifying trends, optimizing scheduling strategies, or understanding user behavior influenced by weekly cycles.

6.3.3 Determining Majority Day or Night Period Metrics

This section adds an extra metric that is defined by a function (majorityTimeDayOrNight()) that determines whether the majority of time between two given datetime strings falls during the day or night. Here's an explanation of how the function works:

Function Definition:
The function majorityTimeDayOrNight() takes two arguments: date_time_str1 and date_time_str2, which represent the datetime strings of two timestamps (arrival and departure).

String to Datetime Conversion:
The datetime strings are converted to Python datetime objects using datetime.strptime() method, which parses a string representing a time according to a specified format.

Defining Start and End Times:
The function identifies the earlier timestamp as the start time and the later timestamp as the end time.

Counting Day and Night Hours:
It iterates through each hour between the start and end times. For each hour, it checks if the hour falls within the defined day or night ranges. The day range is typically defined as 6 AM to 6 PM (hours 6 to 17), while the night range is the remaining hours.

Determining Majority Time:
Based on the accumulated counts of day and night hours, the function determines whether the majority of time falls during the day or night. If the number of day hours is greater than the number of night hours, it returns "Day". If the number of night hours is greater, it returns "Night". If the counts are equal, it returns "Equal Day and Night".

Return Value:
The function returns a string indicating whether the majority of time is day, night, or equal day and night.

Overall, this function provides a simple yet effective way to analyze the time distribution between two given datetime points and categorize it as day or night.

6.3.4 Transforming String-Based Metrics to Numerical Representations

The certain (mapMetricsStringsToNumbers function) is designed to transform certain string-based metrics within the DataFrame (df) into numerical representations. This transformation facilitates subsequent analysis and modeling tasks by converting categorical variables into a format that can be readily interpreted by machine learning algorithms. Here's a breakdown of the process encapsulated within this function:

Renaming Columns:
The function first renames specific columns within the DataFrame to improve clarity and consistency. For example, it changes the column names 'timeZone', 'travTimeDefl', and 'waitTimeDefl' to more descriptive labels.

Mapping City-Country Pairs:
The function utilizes a mapping dictionary to replace city names with corresponding numerical representations of the respective countries. This anonymizes sensitive location information while retaining geographical context.

Mapping Day of Week:
Another mapping dictionary is employed to convert the 'day_of_week' column from string representations (e.g., 'Monday', 'Tuesday') to numerical values (1 for Monday, 2 for Tuesday, and so on). This numerical representation simplifies temporal analysis and pattern recognition.

Mapping Average Travel Period:
The function maps the 'avgTravelPeriod' column, which indicates whether the majority of travel time occurred during the day, night, or an equal duration of both. This categorical variable is mapped to numerical values for computational convenience, enabling more straightforward analysis.
By executing these mapping operations, the mapMetricsStringsToNumbers function transforms qualitative data into a quantitative format, laying the groundwork for robust analysis and modeling efforts without compromising the confidentiality of sensitive location details.

6.3.5 Enhancing DataFrame with Distance Metrics for Transport Analysis

Description:
The functions addDistances and addDistancesToDf play a pivotal role in augmenting a transport dataset with essential distance metrics, thereby facilitating in-depth analysis and optimization of transportation logistics. Let's delve deeper into the functionalities and analytical implications of each function:

addDistances Function:
This function begins by defining the coordinates of various stations based on their latitude and longitude values. These coordinates serve as crucial inputs for distance calculation.
Distance Calculation: The heart of this function lies in the calculate_distance function, which computes the distance between two stations. It leverages two methods for distance calculation:

1. Utilizing OpenStreetMaps: Through the distancesFromOpenStreetMaps.get_driving_distance function, driving distances between stations are obtained where available. This method provides

accurate distance estimates considering road networks and driving routes.

2. Geodesic Distance Calculation: In cases where driving distance data is unavailable, the geodesic function from the geopy.distance module calculates the distance based on geographical coordinates. Although less precise for transportation routes, this method offers a fallback option for distance estimation.

Round to Nearest Decade:
To enhance readability and analysis, the distances calculated are rounded to the nearest multiple of 10, ensuring a simplified representation without significant loss of granularity.
The calculated distances are then applied to each row of the DataFrame, creating a new column labeled 'distance'. This column serves as a fundamental metric for assessing travel distances between stations, crucial for optimizing routing and logistics decisions.

addDistancesToDf Function:
This function complements the distance enrichment process by integrating distance information from an external DataFrame (dfDistance) into the primary DataFrame (df).
Iterative Approach: Through iteration over each row of df, this function systematically applies conditions to determine the appropriate distance values based on station pairs.
Handling Station Matches: The function intelligently handles scenarios where station pairs match or exhibit reverse order in the external DataFrame, ensuring comprehensive coverage of distance data for all relevant station combinations.

Data Enrichment:
By populating the 'distance' column of the DataFrame with accurate distance values derived from the external dataset, this function contributes to the holistic enrichment of transport data, enabling more profound insights into travel distances and network connectivity.

In summary, the analytical prowess of these functions lies in their ability to seamlessly integrate distance metrics into transport datasets, empowering stakeholders to make informed decisions regarding routing optimization, scheduling efficiency, and resource allocation within transportation networks.

6.3.6 Calculating Speed Metrics for Transport Analysis

The addKmPerHour function plays a pivotal role in enhancing a transport dataset by computing crucial speed metrics, specifically kilometers per hour (km/h). Let's delve into the functionality and analytical implications of this function:

Speed Calculation:
The primary objective of this function is to calculate the speed of transport segments represented in the dataset. This is achieved by dividing the distance traveled (in kilometers) by the corresponding travel time (in hours), yielding the speed in km/h.
By computing the speed metric, analysts gain valuable insights into the efficiency and velocity of transportation movements between various stations. This information is instrumental in evaluating the performance of transport routes and identifying areas for optimization.

Handling Non-Finite Values:
In real-world scenarios, certain conditions may lead to non-finite values (such as NaN or inf) in the computed speed metric. These could arise due to factors like zero travel time or division by zero.

To ensure data integrity and facilitate meaningful analysis, the function incorporates robust error handling mechanisms. Specifically, it identifies and replaces non-finite values with a default value (e.g., 0), preventing any distortions in subsequent analyses.

Conversion and Data Integrity:
Once the speed metric ('km/h') is computed, the function ensures data consistency and integrity by converting the calculated values to integers. This simplifies the representation of speed metrics while retaining essential numerical information for analysis.
By converting speed values to integers, the function streamlines data visualization and interpretation, making it easier for stakeholders to comprehend and utilize speed-related insights within the transport dataset.

Integration with Transport Analysis:
The computed speed metrics serve as foundational components for transport analysis, offering valuable inputs for route optimization, scheduling adjustments, and performance evaluation.
By incorporating speed metrics into the dataset, the function contributes to a comprehensive understanding of transportation dynamics, empowering stakeholders to make informed decisions aimed at enhancing efficiency, reliability, and customer satisfaction within transport operations.

In essence, the addKmPerHour function acts as a catalyst for transport analysis, enabling the extraction of valuable speed-related insights essential for optimizing transportation networks and improving overall operational effectiveness.

6.3.7 Seasonal Classification for Temporal Analysis

This stage (addSeason function) enriches a DataFrame with additional information regarding the seasonal classification of timestamps, facilitating comprehensive temporal analysis. Here's an in-depth overview of the function's functionality and analytical significance:

Seasonal Classification Logic:
The core functionality of the addSeason function involves categorizing timestamps into distinct seasons based on their corresponding months. This classification enables analysts to discern seasonal patterns and variations within the dataset.
The function utilizes a straightforward logic to assign each month to one of the four seasons: spring, summer, autumn, or winter. This classification scheme ensures simplicity and consistency in season attribution, facilitating seamless integration with subsequent analyses.

DateTime Processing:
Before applying the seasonal classification logic, the function first converts the timestamp data in the DataFrame to datetime objects. This preprocessing step ensures that date and time information is accurately interpreted and processed for seasonal classification.
By leveraging Pandas' datetime functionality, the function streamlines the conversion process, enhancing efficiency and robustness in handling temporal data within the DataFrame.

Seasonal Attribute Addition:
Once the datetime columns are prepared, the function applies the get_season logic to map each month to its corresponding season. This results in the creation of a new column labeled 'season', wherein each timestamp is associated with its respective seasonal category.
The inclusion of seasonal attributes enriches the dataset with valuable contextual information, empowering analysts to explore and understand seasonal trends, fluctuations, and dependencies across various data dimensions.

Integration with Temporal Analysis:
The seasonal classification provided by the addSeason function serves as a fundamental building block for temporal analysis tasks. Analysts can leverage this information to identify seasonal variations in data patterns, assess the impact of seasonality on business operations, and make informed decisions tailored to specific seasonal contexts.
By integrating seasonal attributes into the DataFrame, the function enhances the dataset's analytical capabilities, paving the way for deeper insights and more effective decision-making in diverse domains, ranging from retail and tourism to agriculture and transportation.

In essence, the addSeason function plays a role in augmenting temporal analysis capabilities, enabling us to unravel seasonal dynamics and extract actionable insights from temporal datasets.

6.4 Anomaly Detection and Removal based on Speed Thresholds

The section filters a DataFrame (df) to remove records containing anomalous speed values falling outside a predefined threshold range. The underlying philosophy is to identify and exclude outliers or erroneous data points that deviate significantly from the expected speed range, thereby enhancing the integrity and reliability of the dataset. Here's a detailed breakdown of the process encapsulated within this code snippet:

Anomaly Detection based on Speed Thresholds:
The primary objective of the code is to identify and eliminate records with speed values that lie beyond a specified range. This range is defined by lower and upper speed thresholds, typically set to mitigate the impact of outliers or measurement errors.
By applying logical conditions, the code selects records where the speed ('km/h') falls within the designated range, effectively isolating data points deemed plausible and consistent with normal operational conditions.

Threshold Selection and Justification:
The lower speed threshold (10 km/h) and upper speed threshold (80 km/h) are chosen based on domain knowledge, operational constraints, or empirical observations. These thresholds reflect a balance between capturing genuine data and filtering out potential anomalies.
The specific threshold values may vary depending on the context of the dataset and the nature of the observed speeds. Adjustments to these thresholds may be made iteratively based on data validation, outlier analysis, or performance requirements.

Exclusion of Anomalous Records:
Records failing to meet the specified speed criteria are filtered out from the DataFrame using Boolean indexing. The logical conjunction (&) ensures that records must satisfy both the lower and upper speed thresholds simultaneously to be retained in the filtered DataFrame.
Anomalies characterized by excessively low or high speeds, which could indicate data recording errors, equipment malfunctions, or exceptional circumstances, are effectively removed from further analysis.

Data Integrity and Quality Assurance:
The removal of anomalous records contributes to enhancing the integrity, reliability, and quality of the dataset, thereby minimizing the risk of erroneous conclusions or biased insights drawn from subsequent analyses.
By enforcing strict criteria for speed validity, the code promotes data consistency and trustworthiness, aligning with best practices in data preprocessing and quality assurance.

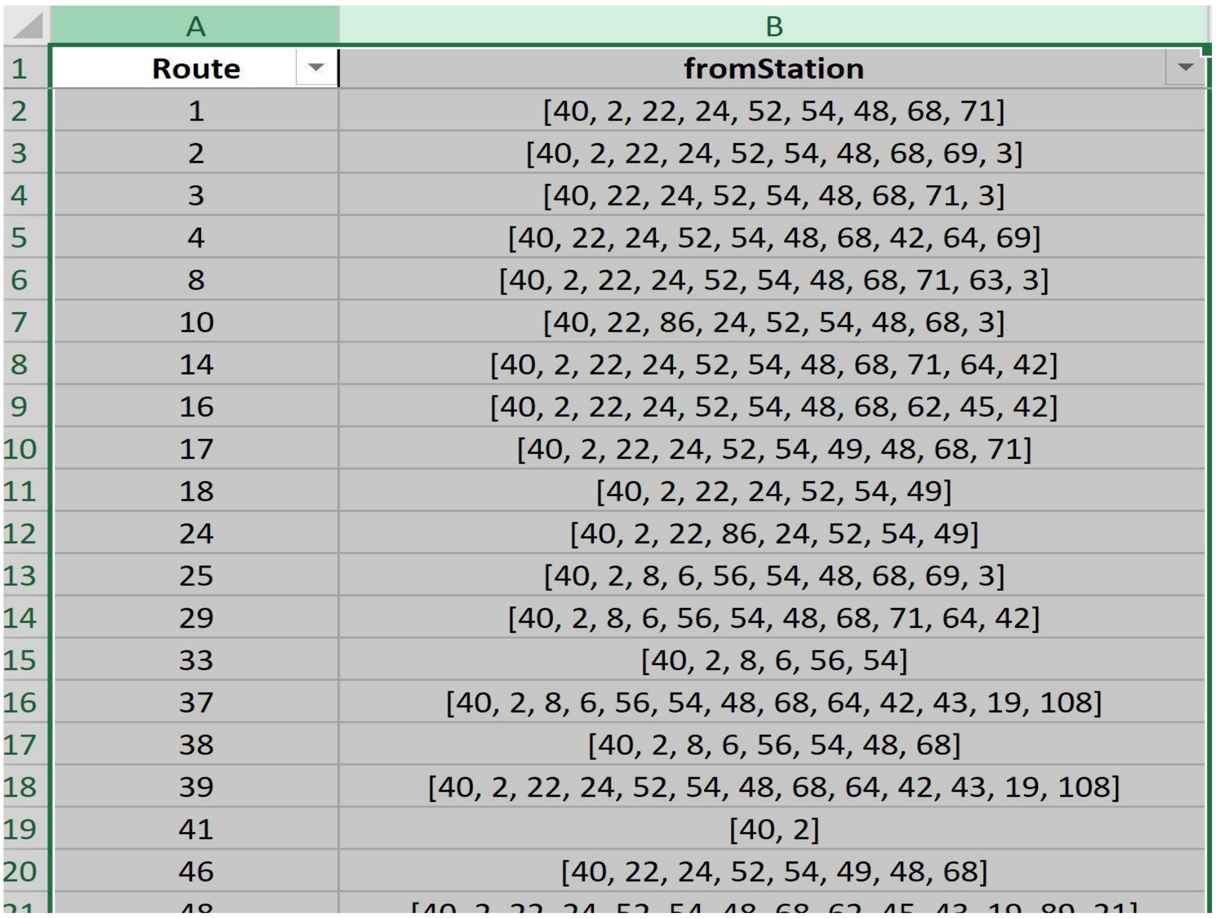
In summary, the code snippet exemplifies a systematic approach to anomaly detection and data cleansing, focusing on the identification and removal of outliers in speed data to ensure the integrity and reliability of downstream analyses and decision-making processes.

7 Analysis Through Comprehensive Preprocessing

As we navigate through the preprocessing journey, we uncover a succession of refined datasets, each illuminating a distinct facet of transportation dynamics. These products emerge from the fusion of raw data with sophisticated preprocessing techniques, providing insight into the intricate tapestry of transportation ecosystems.

Through these snapshots of our processed datasets, we gain valuable insights that can guide us in selecting the most suitable Machine Learning Model and achieving the desired outcomes. By leveraging the refined data representations, we can better understand the underlying patterns and complexities inherent in transportation logistics. These insights pave the way for informed decision-making and drive advancements in optimizing operational efficiencies and strategic planning.

7.1 Station Series For Each Route

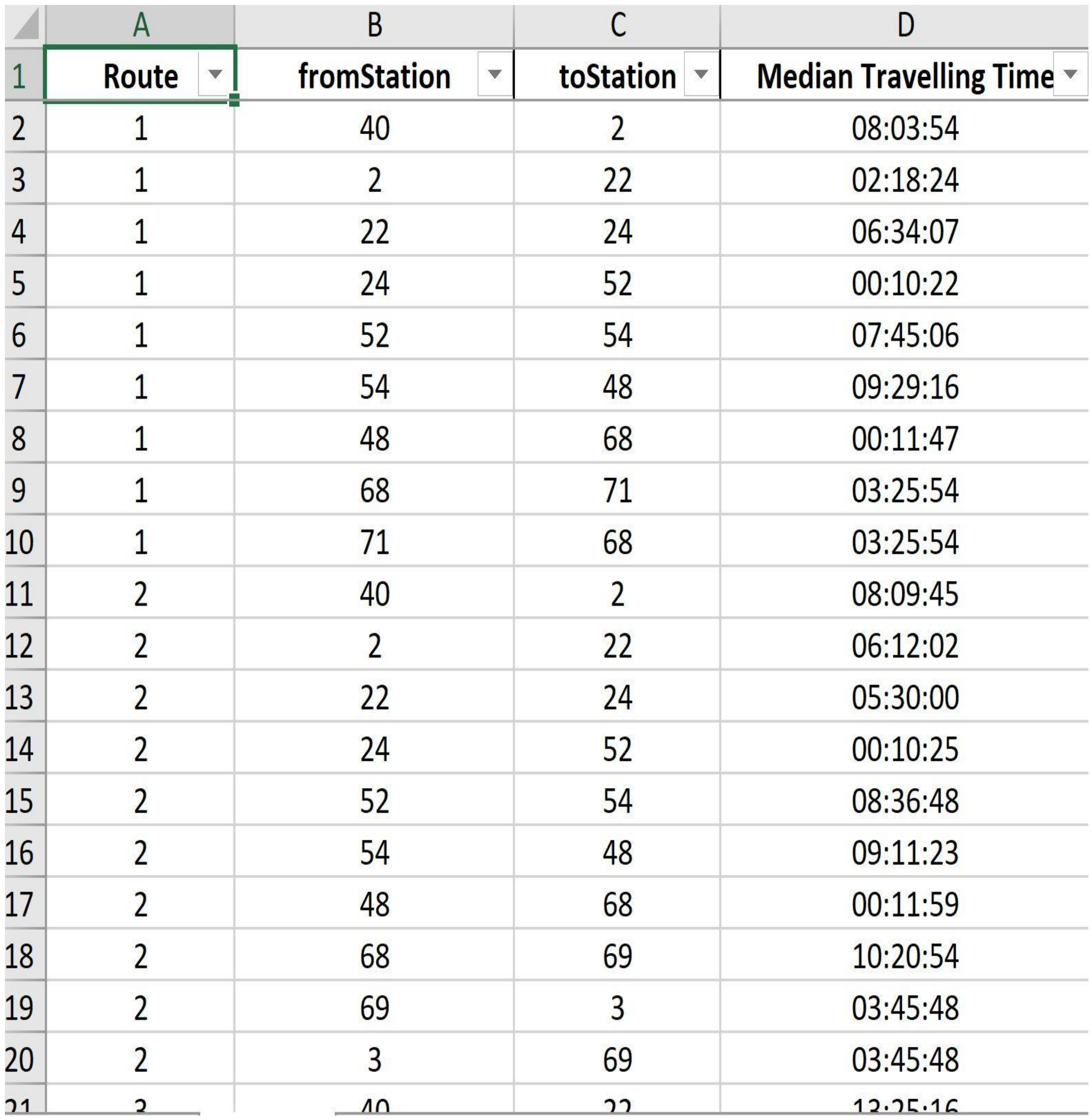


	Route	fromStation
2	1	[40, 2, 22, 24, 52, 54, 48, 68, 71]
3	2	[40, 2, 22, 24, 52, 54, 48, 68, 69, 3]
4	3	[40, 22, 24, 52, 54, 48, 68, 71, 3]
5	4	[40, 22, 24, 52, 54, 48, 68, 42, 64, 69]
6	8	[40, 2, 22, 24, 52, 54, 48, 68, 71, 63, 3]
7	10	[40, 22, 86, 24, 52, 54, 48, 68, 3]
8	14	[40, 2, 22, 24, 52, 54, 48, 68, 71, 64, 42]
9	16	[40, 2, 22, 24, 52, 54, 48, 68, 62, 45, 42]
10	17	[40, 2, 22, 24, 52, 54, 49, 48, 68, 71]
11	18	[40, 2, 22, 24, 52, 54, 49]
12	24	[40, 2, 22, 86, 24, 52, 54, 49]
13	25	[40, 2, 8, 6, 56, 54, 48, 68, 69, 3]
14	29	[40, 2, 8, 6, 56, 54, 48, 68, 71, 64, 42]
15	33	[40, 2, 8, 6, 56, 54]
16	37	[40, 2, 8, 6, 56, 54, 48, 68, 64, 42, 43, 19, 108]
17	38	[40, 2, 8, 6, 56, 54, 48, 68]
18	39	[40, 2, 22, 24, 52, 54, 48, 68, 64, 42, 43, 19, 108]
19	41	[40, 2]
20	46	[40, 22, 24, 52, 54, 49, 48, 68]
21	48	[40, 2, 22, 24, 52, 54, 48, 68, 62, 45, 43, 19, 89, 21]

Table 4 : Station Series For Each Route

Each row represents a different route, where the numbers in the square brackets represent the sequence of stations visited on that route. Total Identified Routes : 265

7.2 Station Series For Each Route

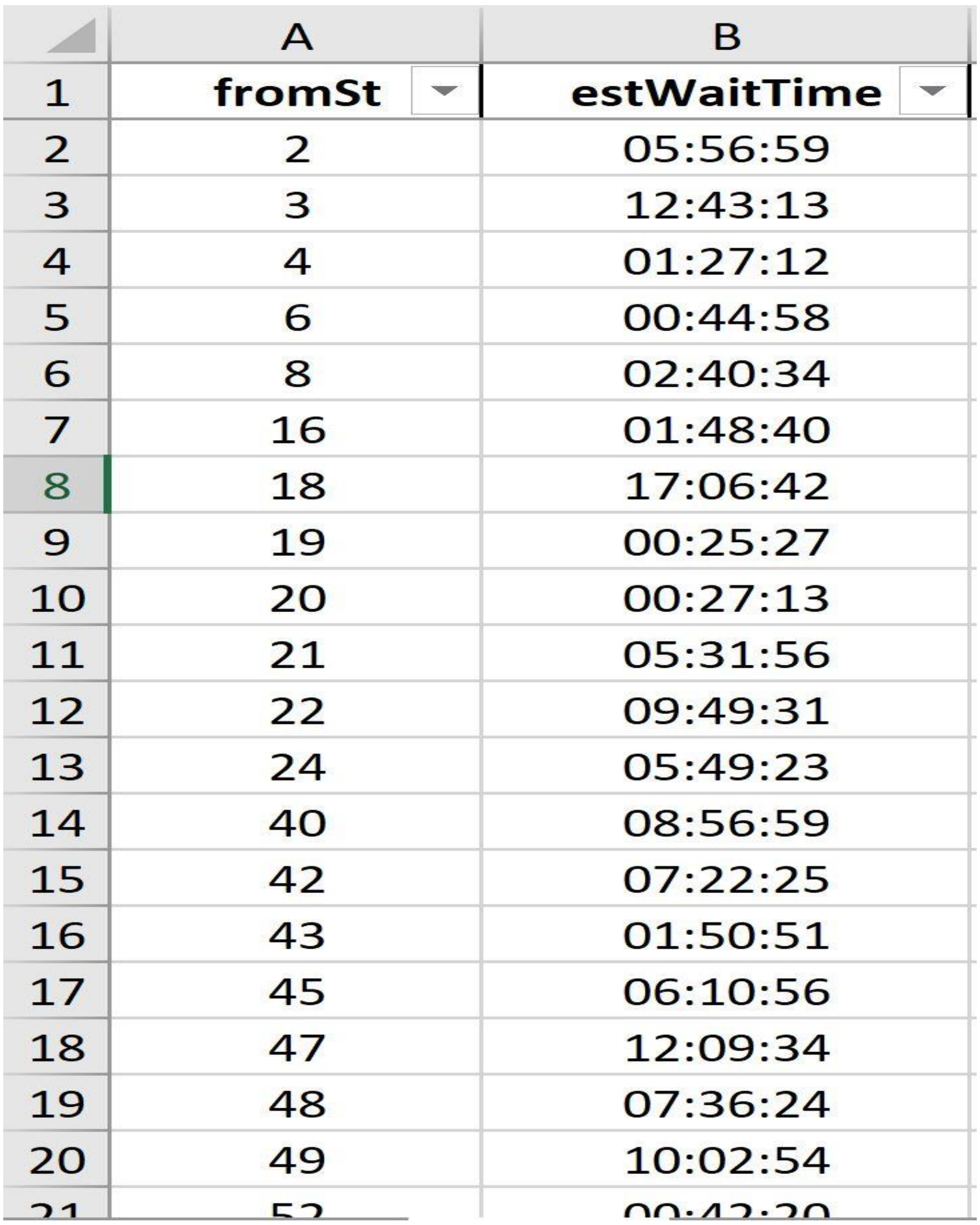


	A	B	C	D
1	Route	fromStation	toStation	Median Travelling Time
2	1	40	2	08:03:54
3	1	2	22	02:18:24
4	1	22	24	06:34:07
5	1	24	52	00:10:22
6	1	52	54	07:45:06
7	1	54	48	09:29:16
8	1	48	68	00:11:47
9	1	68	71	03:25:54
10	1	71	68	03:25:54
11	2	40	2	08:09:45
12	2	2	22	06:12:02
13	2	22	24	05:30:00
14	2	24	52	00:10:25
15	2	52	54	08:36:48
16	2	54	48	09:11:23
17	2	48	68	00:11:59
18	2	68	69	10:20:54
19	2	69	3	03:45:48
20	2	3	69	03:45:48
21	3	40	22	13:25:16

Table 5 : Station Series For Each Route

Each row represents a segment of a route, indicating the route number, the starting station, the destination station, and the median traveling time between those stations. Total Identified Segments : 1286.

7.3 Estimated Waiting Time at the Stations

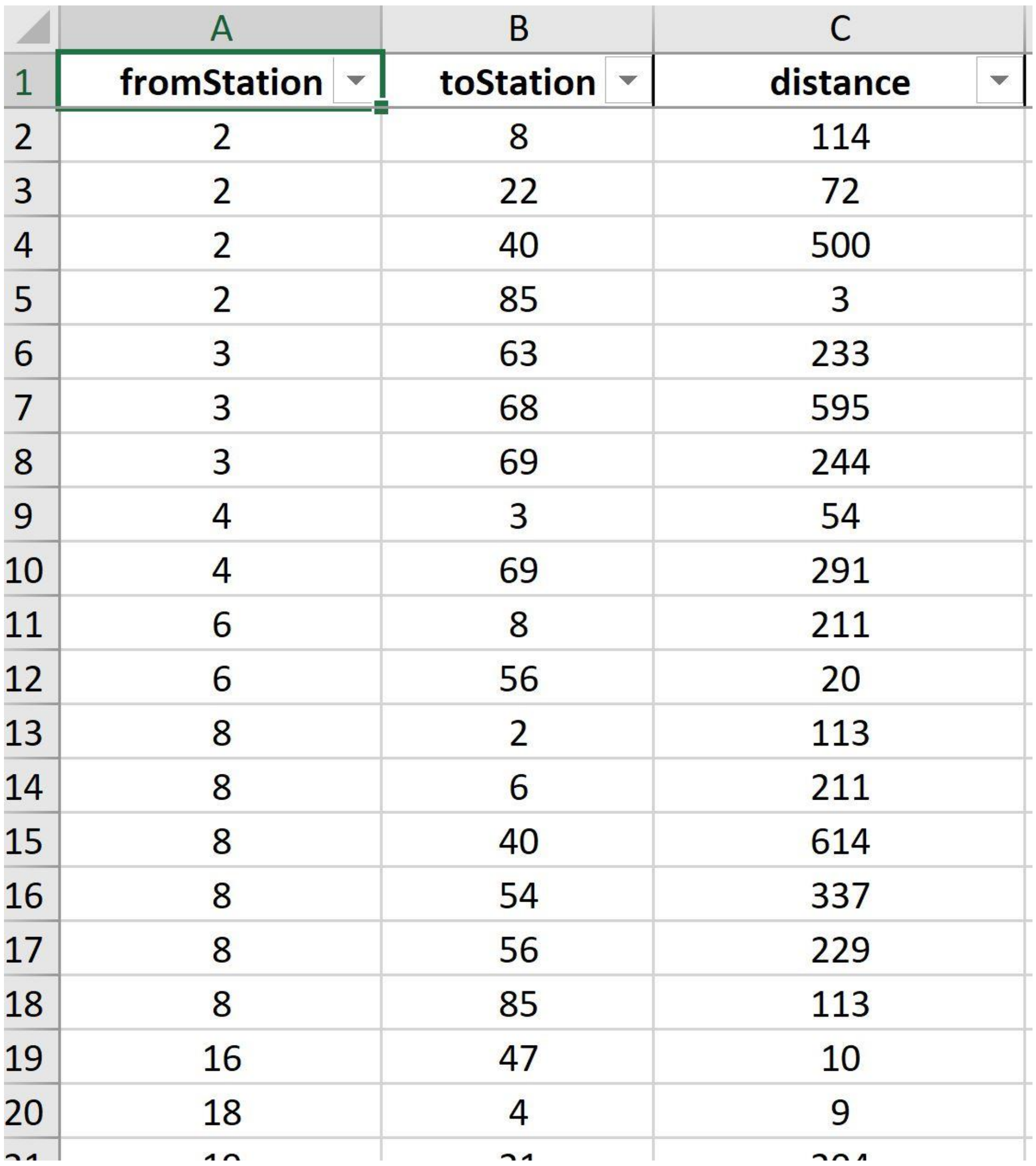


	A	B
1	fromSt	estWaitTime
2	2	05:56:59
3	3	12:43:13
4	4	01:27:12
5	6	00:44:58
6	8	02:40:34
7	16	01:48:40
8	18	17:06:42
9	19	00:25:27
10	20	00:27:13
11	21	05:31:56
12	22	09:49:31
13	24	05:49:23
14	40	08:56:59
15	42	07:22:25
16	43	01:50:51
17	45	06:10:56
18	47	12:09:34
19	48	07:36:24
20	49	10:02:54
21	52	00:42:20

Table 6 : Estimated Waiting Time For Each Station

Each row represents a station (fromSt) along with the estimated waiting time (estWaitTime) at that station. Total Identified Stations : 120.

7.4 Station Distances

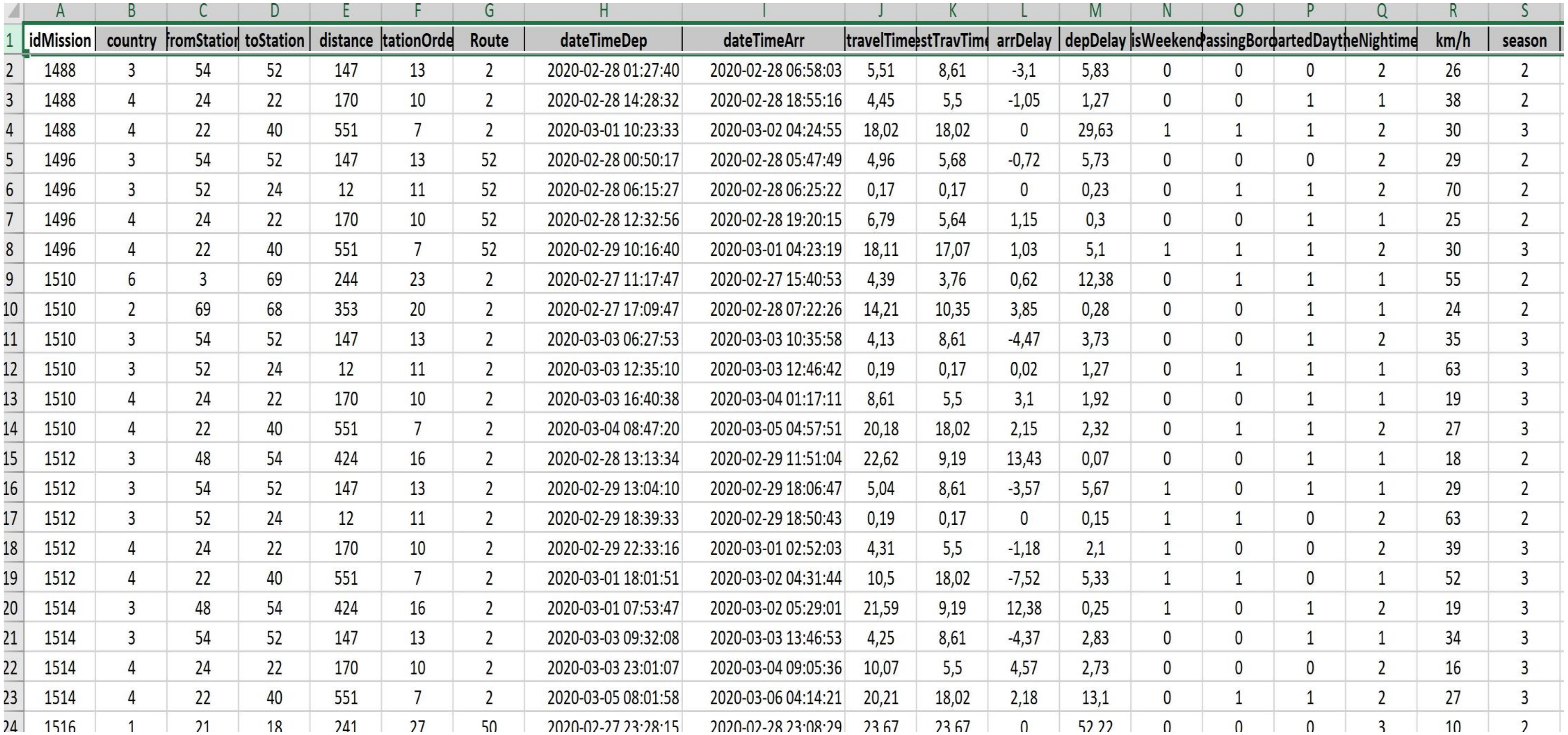


	A	B	C
1	fromStation	toStation	distance
2	2	8	114
3	2	22	72
4	2	40	500
5	2	85	3
6	3	63	233
7	3	68	595
8	3	69	244
9	4	3	54
10	4	69	291
11	6	8	211
12	6	56	20
13	8	2	113
14	8	6	211
15	8	40	614
16	8	54	337
17	8	56	229
18	8	85	113
19	16	47	10
20	18	4	9
[illegible]	[illegible]	[illegible]	[illegible]

Table 7 : Distances Between Stations

Each row represents a pair of stations along with the distance between them. Total Computations of Distances (Via Open Street Map API) : 200.

7.5 Mission Travel Data: Understanding Routes, Delays, and Travel Patterns



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
1	idMission	country	fromStatior	toStation	distance	tationOrde	Route	dateTimeDep	dateTimeArr	travelTime	estTravTim	arrDelay	depDelay	isWeekend	PassingBor	partedDayt	heNightime	km/h	season
2	1488	3	54	52	147	13	2	2020-02-28 01:27:40	2020-02-28 06:58:03	5,51	8,61	-3,1	5,83	0	0	0	2	26	2
3	1488	4	24	22	170	10	2	2020-02-28 14:28:32	2020-02-28 18:55:16	4,45	5,5	-1,05	1,27	0	0	1	1	38	2
4	1488	4	22	40	551	7	2	2020-03-01 10:23:33	2020-03-02 04:24:55	18,02	18,02	0	29,63	1	1	1	2	30	3
5	1496	3	54	52	147	13	52	2020-02-28 00:50:17	2020-02-28 05:47:49	4,96	5,68	-0,72	5,73	0	0	0	2	29	2
6	1496	3	52	24	12	11	52	2020-02-28 06:15:27	2020-02-28 06:25:22	0,17	0,17	0	0,23	0	1	1	2	70	2
7	1496	4	24	22	170	10	52	2020-02-28 12:32:56	2020-02-28 19:20:15	6,79	5,64	1,15	0,3	0	0	1	1	25	2
8	1496	4	22	40	551	7	52	2020-02-29 10:16:40	2020-03-01 04:23:19	18,11	17,07	1,03	5,1	1	1	1	2	30	3
9	1510	6	3	69	244	23	2	2020-02-27 11:17:47	2020-02-27 15:40:53	4,39	3,76	0,62	12,38	0	1	1	1	55	2
10	1510	2	69	68	353	20	2	2020-02-27 17:09:47	2020-02-28 07:22:26	14,21	10,35	3,85	0,28	0	0	1	1	24	2
11	1510	3	54	52	147	13	2	2020-03-03 06:27:53	2020-03-03 10:35:58	4,13	8,61	-4,47	3,73	0	0	1	2	35	3
12	1510	3	52	24	12	11	2	2020-03-03 12:35:10	2020-03-03 12:46:42	0,19	0,17	0,02	1,27	0	1	1	1	63	3
13	1510	4	24	22	170	10	2	2020-03-03 16:40:38	2020-03-04 01:17:11	8,61	5,5	3,1	1,92	0	0	1	1	19	3
14	1510	4	22	40	551	7	2	2020-03-04 08:47:20	2020-03-05 04:57:51	20,18	18,02	2,15	2,32	0	1	1	2	27	3
15	1512	3	48	54	424	16	2	2020-02-28 13:13:34	2020-02-29 11:51:04	22,62	9,19	13,43	0,07	0	0	1	1	18	2
16	1512	3	54	52	147	13	2	2020-02-29 13:04:10	2020-02-29 18:06:47	5,04	8,61	-3,57	5,67	1	0	1	1	29	2
17	1512	3	52	24	12	11	2	2020-02-29 18:39:33	2020-02-29 18:50:43	0,19	0,17	0	0,15	1	1	0	2	63	2
18	1512	4	24	22	170	10	2	2020-02-29 22:33:16	2020-03-01 02:52:03	4,31	5,5	-1,18	2,1	1	0	0	2	39	3
19	1512	4	22	40	551	7	2	2020-03-01 18:01:51	2020-03-02 04:31:44	10,5	18,02	-7,52	5,33	1	1	0	1	52	3
20	1514	3	48	54	424	16	2	2020-03-01 07:53:47	2020-03-02 05:29:01	21,59	9,19	12,38	0,25	1	0	1	2	19	3
21	1514	3	54	52	147	13	2	2020-03-03 09:32:08	2020-03-03 13:46:53	4,25	8,61	-4,37	2,83	0	0	1	1	34	3
22	1514	4	24	22	170	10	2	2020-03-03 23:01:07	2020-03-04 09:05:36	10,07	5,5	4,57	2,73	0	0	0	2	16	3
23	1514	4	22	40	551	7	2	2020-03-05 08:01:58	2020-03-06 04:14:21	20,21	18,02	2,18	13,1	0	1	1	2	27	3
24	1516	1	21	18	241	27	50	2020-02-27 23:28:15	2020-02-28 23:08:29	23,67	23,67	0	52,22	0	0	0	3	10	2

Table 8 : Final Dataset After the Processing

Here's a breakdown of the columns in your dataset:

- **idMission**: Mission ID
- **country**: Country code
- **fromStation**: Station code for the starting point of the mission
- **toStation**: Station code for the destination of the mission
- **distance (km)**: Distance between the starting and destination stations
- **stationOrder**: Order of the station in the route
- **Route**: Route ID
- **dateTimeDep**: Date and time of departure
- **dateTimeArr**: Date and time of arrival
- **travelTime (hours)**: Actual travel time
- **estTravTime (hours)**: Estimated travel time
- **arrDelay (hours)**: Arrival delay
- **depDelay (hours)**: Departure delay
- **isWeekend**: Indicator for weekend (1 for weekend, 0 for weekday)
- **isPassingBorder**: Indicator for passing border (1 for passing, 0 for not passing)
- **departedDaytime**: Indicator for departure time (1 for daytime, 0 for nighttime)
- **isDaytimeNightimeOrEqual**: Indicator for day or night (1 for daytime, 2 for nighttime, 0 for equal)
- **km/h**: Speed in kilometers per hour

Each row in the dataset represents a specific mission with its corresponding details

9. Code Implementation

In the implementation phase, a comprehensive selection of Python libraries and frameworks, along with Python version 3.8.10, has been leveraged to facilitate various analytical tasks. This section provides a detailed overview of the key tools utilized, along with their respective versions.

Libraries and Frameworks:

1. **pandas 2.0.3:** Pandas, a powerful data manipulation library, was instrumental in handling structured data efficiently. Its versatile functionalities allowed for seamless data manipulation and analysis.
2. **requests 2.31.0:** The requests library facilitated seamless HTTP requests, enabling data retrieval from APIs and web scraping tasks with ease.
3. **geopy 2.4.1:** Geopy played a crucial role in geospatial data processing, providing geocoding and distance calculation functionalities essential for location-based analysis.
4. **plotly 5.20.0:** Plotly, a versatile visualization library, offered interactive and aesthetically pleasing plots, enhancing data exploration and presentation capabilities.
5. **matplotlib 3.7.5:** Matplotlib, a fundamental plotting library, provided a robust foundation for creating static visualizations, including line plots, bar charts, histograms, and more.
6. **seaborn 0.13.12:** Seaborn complemented Matplotlib by providing high-level abstractions and aesthetically pleasing statistical plots, making complex visualizations more accessible.
7. **scikit-learn 1.3.2:** Scikit-learn, a comprehensive machine learning library, facilitated the implementation of various machine learning algorithms for predictive modeling and data classification tasks.
8. **django 4.2.11:** Django, a high-level Python web framework, was employed for developing robust and scalable web applications, providing built-in features for authentication, URL routing, and database management.
9. **flask 3.0.2:** Flask, a lightweight web framework, offered flexibility and simplicity for developing web applications with minimal boilerplate code, making it ideal for small to medium-scale projects.
10. **pymysql 1.1.0:** PyMySQL facilitated seamless interaction with MySQL databases, enabling data retrieval, manipulation, and storage operations within Python applications.
11. **openpyxl 3.1.2:** Openpyxl provided functionalities for reading, writing, and manipulating Excel files, offering compatibility with both .xlsx and .xlsm formats.
12. **xlsxwriter 3.2.0:** XlsxWriter enabled the creation of Excel files with advanced formatting options, including charts, images, and conditional formatting, enhancing the presentation of analytical results.

The strategic selection and utilization of these libraries and frameworks, along with Python version 3.8.10, greatly enhanced the efficiency and effectiveness of the analytical implementation, enabling robust data manipulation, visualization, machine learning, and web development capabilities.

10. Evaluating Models

10.1 Assessing Performance and Predictive Capabilities

The evaluation process of our predictive model involved several steps to assess its performance and understand its predictive capabilities. Here's an analytical breakdown of the key aspects of the evaluation:

Feature Selection and Engineering:

The model was trained using three main features: 'km/h', 'estTravTime', and 'distance', which were deemed relevant for predicting the target variable 'arrDelay'. These features were selected based on their potential impact on the delay time of the transportation system.

Data Splitting:

The dataset was split into training and testing sets using the train_test_split function from scikit-learn. The training set comprised 80% of the data, while the testing set contained the remaining 20%. This split ensured that the model's performance could be accurately evaluated on unseen data.

Model Selection and Training:

A Random Forest Regressor model was chosen due to its ability to handle complex datasets and mitigate overfitting. The model was trained on the training data using the fit method. in the evaluation of our analytical models, we explored a range of techniques to assess their performance. Among the models employed, including linear regression, support vector machines, and decision trees, it became evident that the random forest algorithm consistently yielded the most promising results. The robustness of random forest in handling complex datasets, along with its ability to mitigate overfitting and variance, proved instrumental in achieving superior predictive accuracy.

Model Evaluation:

The performance of the trained model was evaluated using the Mean Absolute Error (MAE) metric, calculated by comparing the actual 'arrDelay' values with the predicted values on the testing set. A lower MAE indicates better predictive accuracy.

Correlation Analysis:

A correlation matrix was computed to assess the linear relationships between the features and the target variable. Heatmap visualization was employed to visualize the correlation matrix, providing insights into the strength and direction of these relationships.

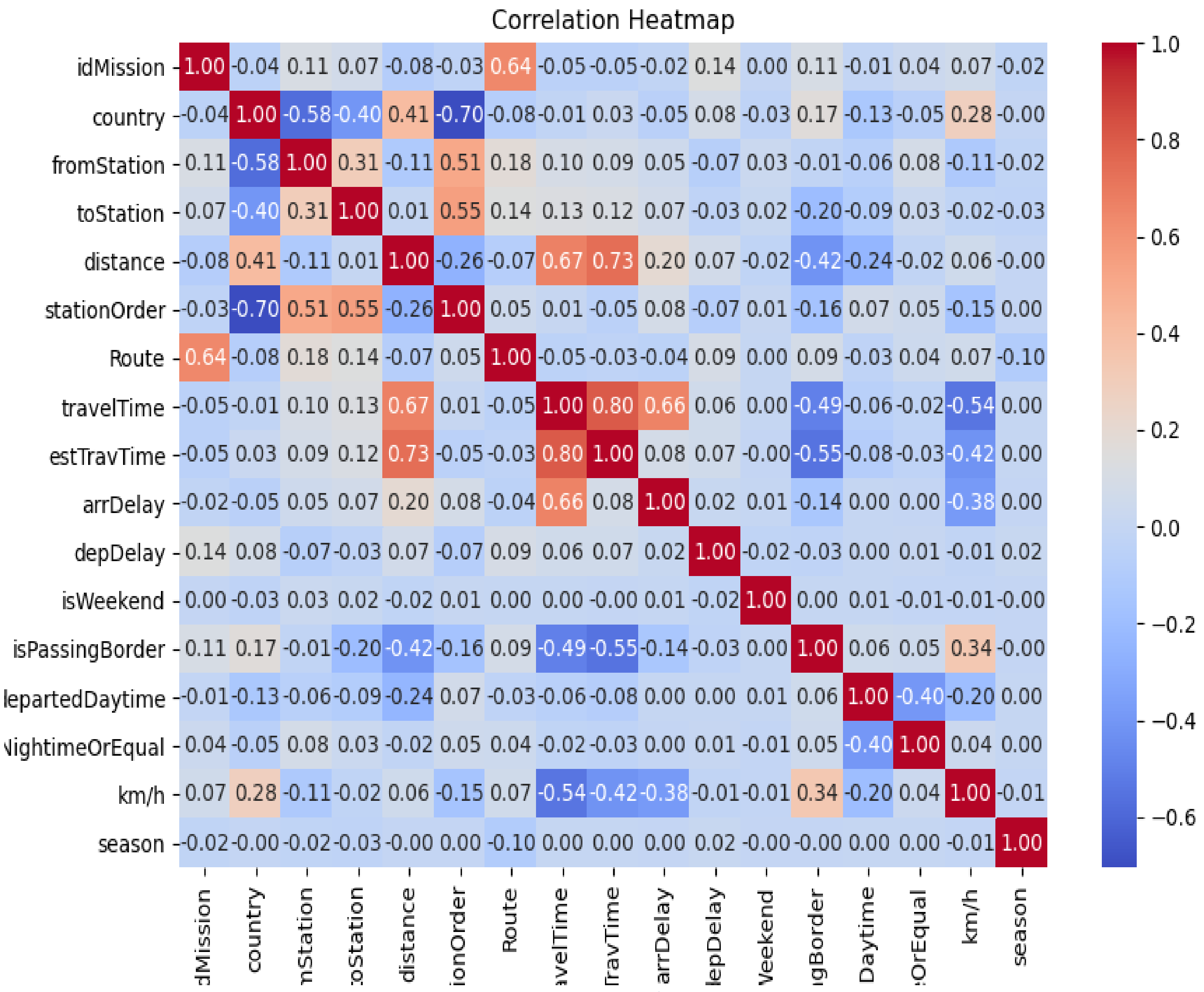
Feature Importance Analysis:

The importance of each feature in predicting 'arrDelay' was determined using the feature importances provided by the Random Forest model. A higher feature importance indicates a stronger influence on the target variable.

1. **Visualization of Results:**

The evaluation results were presented through various visualizations, including:

Correlation Heatmap: Illustrating the correlation between different features and the target variable.



	dMission	country	nStation	oStation	distance	ionOrder	Route	avelTime	TravTime	arrDelay	lepDelay	Veekend	igBorder	Daytime	:OrEqual	km/h	season
idMission	1.00	-0.04	0.11	0.07	-0.08	-0.03	0.64	-0.05	-0.05	-0.02	0.14	0.00	0.11	-0.01	0.04	0.07	-0.02
country	-0.04	1.00	-0.58	-0.40	0.41	-0.70	-0.08	-0.01	0.03	-0.05	0.08	-0.03	0.17	-0.13	-0.05	0.28	-0.00
fromStation	0.11	-0.58	1.00	0.31	-0.11	0.51	0.18	0.10	0.09	0.05	-0.07	0.03	-0.01	-0.06	0.08	-0.11	-0.02
toStation	0.07	-0.40	0.31	1.00	0.01	0.55	0.14	0.13	0.12	0.07	-0.03	0.02	-0.20	-0.09	0.03	-0.02	-0.03
distance	-0.08	0.41	-0.11	0.01	1.00	-0.26	-0.07	0.67	0.73	0.20	0.07	-0.02	-0.42	-0.24	-0.02	0.06	-0.00
stationOrder	-0.03	-0.70	0.51	0.55	-0.26	1.00	0.05	0.01	-0.05	0.08	-0.07	0.01	-0.16	0.07	0.05	-0.15	0.00
Route	0.64	-0.08	0.18	0.14	-0.07	0.05	1.00	-0.05	-0.03	-0.04	0.09	0.00	0.09	-0.03	0.04	0.07	-0.10
travelTime	-0.05	-0.01	0.10	0.13	0.67	0.01	-0.05	1.00	0.80	0.66	0.06	0.00	-0.49	-0.06	-0.02	-0.54	0.00
estTravTime	-0.05	0.03	0.09	0.12	0.73	-0.05	-0.03	0.80	1.00	0.08	0.07	-0.00	-0.55	-0.08	-0.03	-0.42	0.00
arrDelay	-0.02	-0.05	0.05	0.07	0.20	0.08	-0.04	0.66	0.08	1.00	0.02	0.01	-0.14	0.00	0.00	-0.38	0.00
depDelay	0.14	0.08	-0.07	-0.03	0.07	-0.07	0.09	0.06	0.07	0.02	1.00	-0.02	-0.03	0.00	0.01	-0.01	0.02
isWeekend	0.00	-0.03	0.03	0.02	-0.02	0.01	0.00	0.00	-0.00	0.01	-0.02	1.00	0.00	0.01	-0.01	-0.01	-0.00
isPassingBorder	0.11	0.17	-0.01	-0.20	-0.42	-0.16	0.09	-0.49	-0.55	-0.14	-0.03	0.00	1.00	0.06	0.05	0.34	-0.00
lepartedDaytime	-0.01	-0.13	-0.06	-0.09	-0.24	0.07	-0.03	-0.06	-0.08	0.00	0.00	0.01	0.06	1.00	-0.40	-0.20	0.00
VightimeOrEqual	0.04	-0.05	0.08	0.03	-0.02	0.05	0.04	-0.02	-0.03	0.00	0.01	-0.01	0.05	-0.40	1.00	0.04	0.00
km/h	0.07	0.28	-0.11	-0.02	0.06	-0.15	0.07	-0.54	-0.42	-0.38	-0.01	-0.01	0.34	-0.20	0.04	1.00	-0.01
season	-0.02	-0.00	-0.02	-0.03	-0.00	0.00	-0.10	0.00	0.00	0.00	0.02	-0.00	-0.00	0.00	0.00	-0.01	1.00

Figure 4 : Correlation Heat Map

Correlation Bar Plot: Displaying the correlation of each feature with 'arrDelay'.

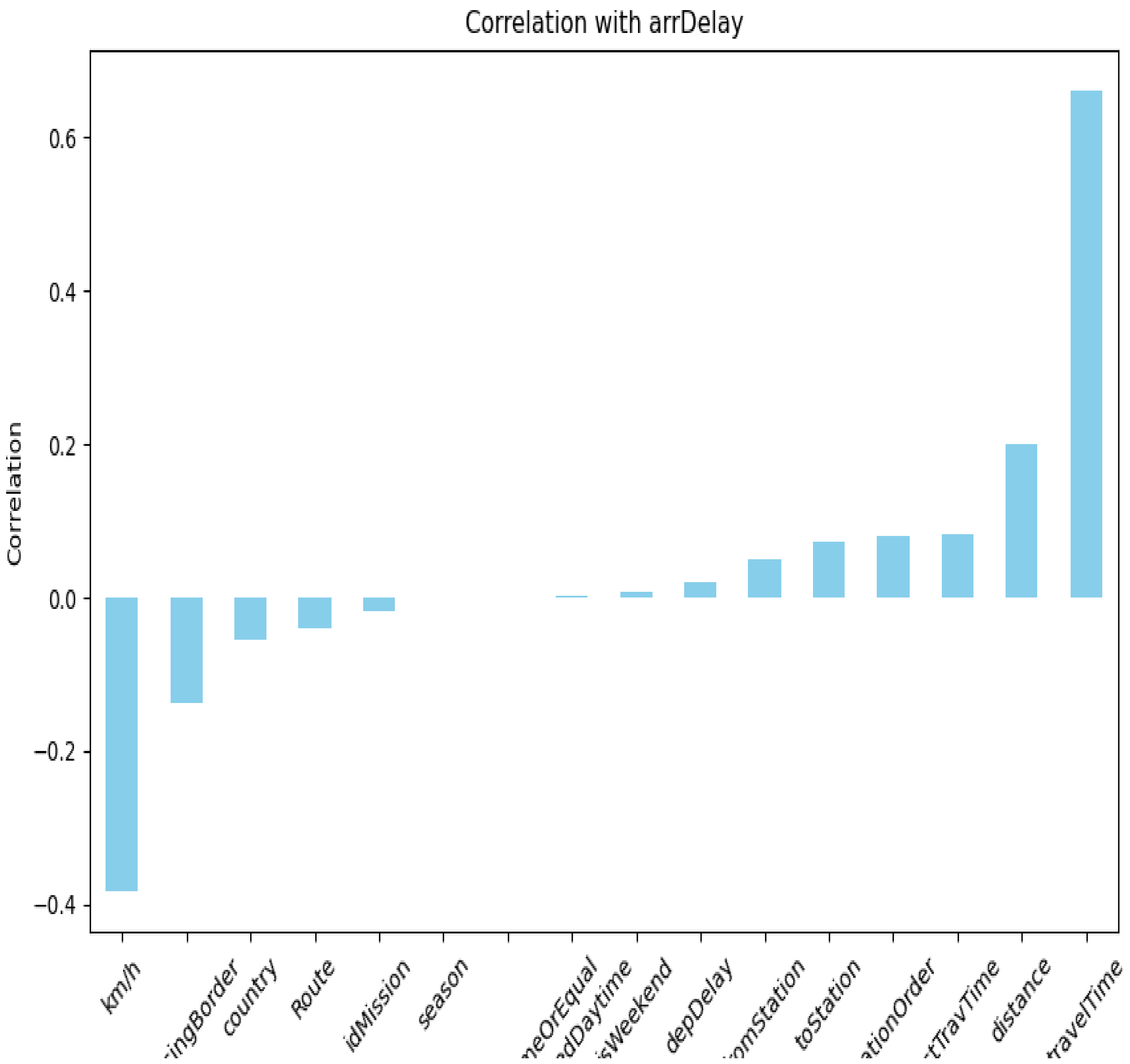


Figure 5 : Correlation Bar Plot

Scatter Plot: Showing the relationship between actual and predicted 'arrDelay' values.

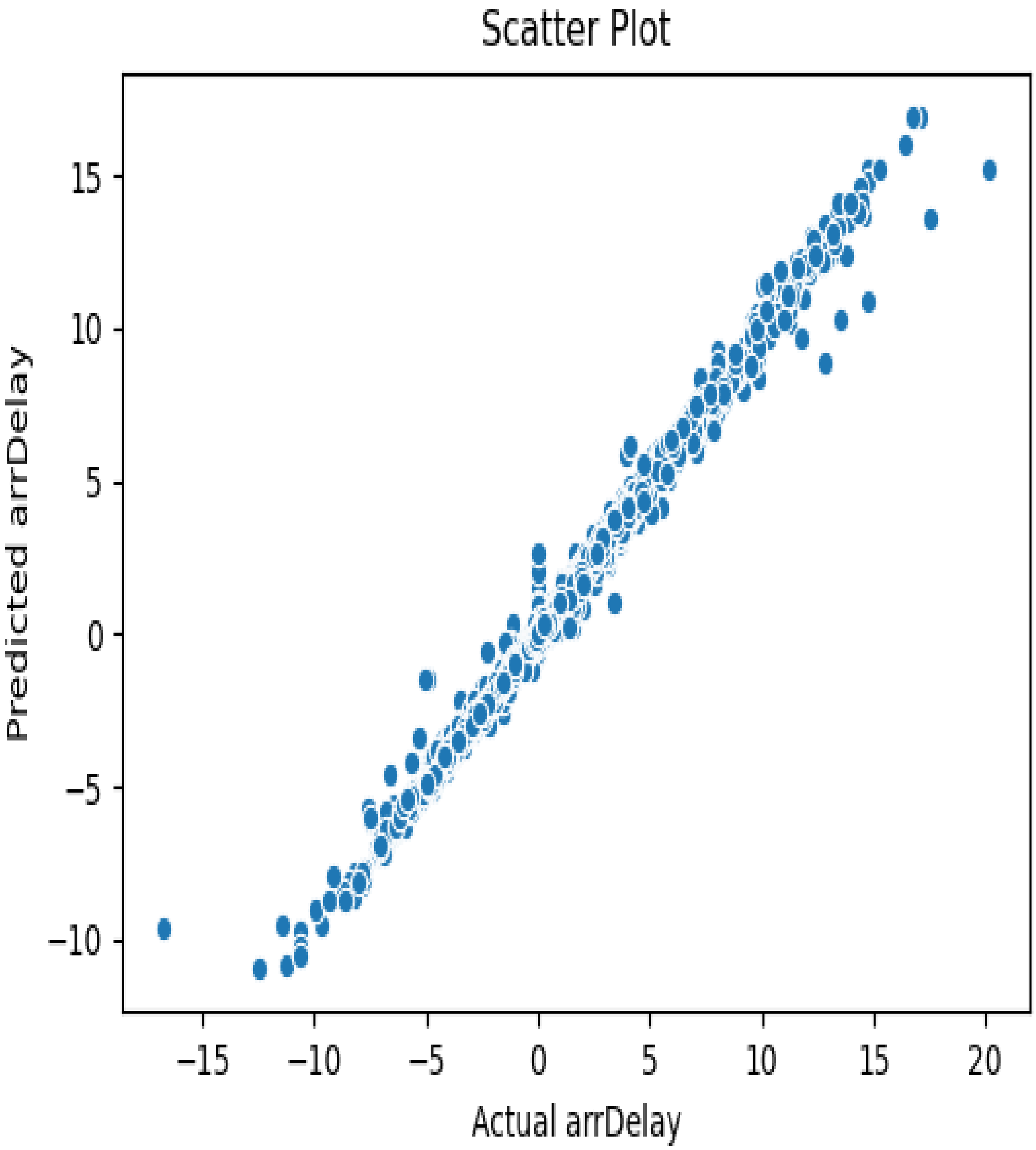


Figure 6 : Scatter Plot

Residual Plot: Visualizing the distribution of residuals (the difference between actual and predicted values).

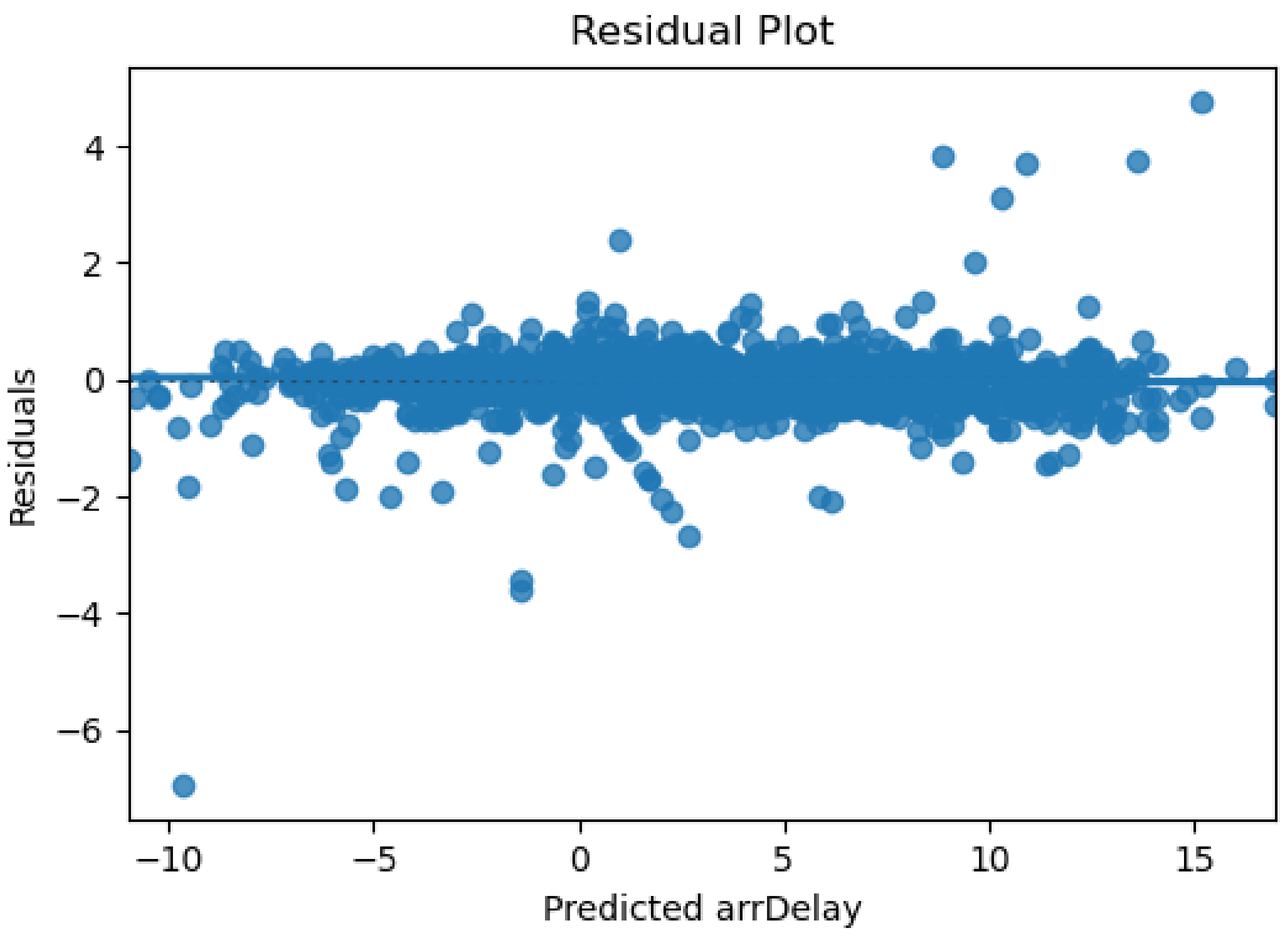


Figure 7 : Residual Plot

Feature Importance Plot: Highlighting the importance of each feature in predicting 'arrDelay'.
By following this structured evaluation approach and leveraging insightful visualizations, we gain a comprehensive understanding of the model's performance and its ability to accurately predict arrival delay in the transportation system.

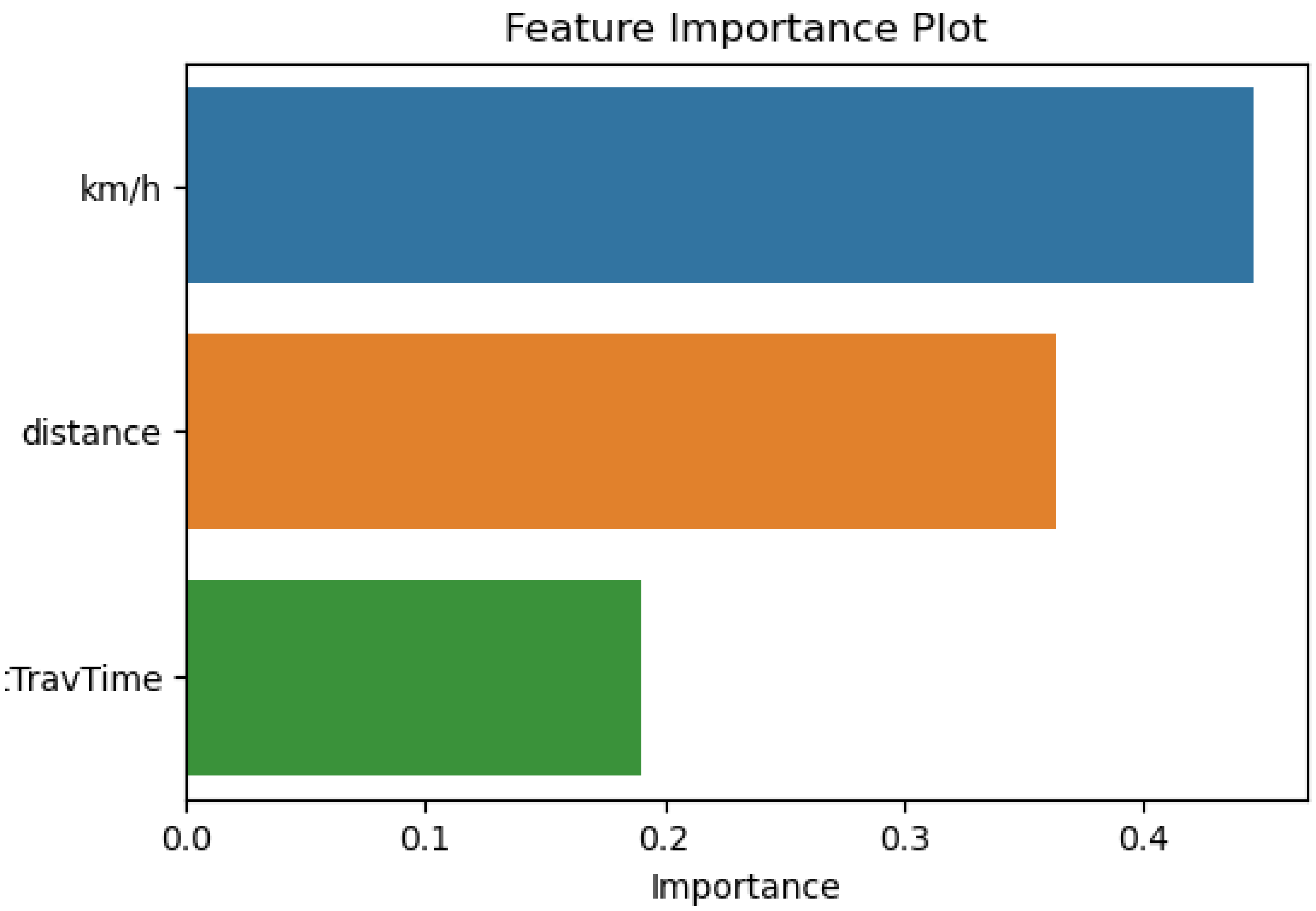


Figure 8 : Feature Importance Plot

10.2 Mean Absolute Error (MAE)

The Mean Absolute Error (MAE) of the Random Forest model, a widely used metric for assessing prediction accuracy, was calculated to be 0.1319 hours. This metric quantifies the average magnitude of errors between the actual and predicted values of the 'arrDelay' variable. A MAE of 0.1319 hours implies that, on average, the model's predictions deviate from the actual delay time by approximately 7.92 minutes. This relatively low error rate indicates that the Random Forest model performs well in accurately predicting arrival delays, providing valuable insights for optimizing scheduling and resource allocation in the transportation system.

10.3 Feature Selection

The feature selection process was informed by rigorous testing and analysis, including examination of the correlation map. Despite evaluating features such as passing borders, day of the week, and season, it became evident that predicting delay solely based on objective features was unfeasible. This limitation likely stems from the influence of human factors, such as driving speed variability among drivers and manual checks at border crossings, which are inherently difficult to measure quantitatively. **Recognizing this challenge, we devised a novel solution: computing the train's speed (km/h) based on its position and the time it commenced its mission.** This approach aimed to capture the dynamic nature of travel conditions and account for the human factor indirectly, leading to a more accurate predictive model.

11. APIs For Computing Distance and Predicting Delay

11.1 Application Programming Interfaces (APIs)

API stands for Application Programming Interface, which allows different software applications to communicate with each other. APIs define a set of rules and protocols that enable the interaction between various software components, facilitating data exchange and functionality integration.

APIs can be categorized into different types based on their functionality and usage, such as:

1. Web APIs: These APIs are accessible over the internet and are commonly used for web-based services. They enable communication between web servers and client applications, allowing data retrieval, manipulation, and other operations [34].

2. RESTful APIs: Representational State Transfer (REST) APIs adhere to the principles of REST architecture, providing a standardized way of building web APIs. They use HTTP methods (GET, POST, PUT, DELETE) to perform operations on resources, and responses are typically formatted in JSON or XML [35].

3. SOAP APIs: Simple Object Access Protocol (SOAP) APIs are based on the XML messaging protocol and are used for exchanging structured information between applications over a network. They define a strict messaging format and often require more bandwidth compared to RESTful APIs [36].

4. Internal APIs: These APIs are used within an organization or software system to enable communication between different components or services. They help streamline development, promote code reusability, and enhance system interoperability [37].

APIs play a crucial role in modern software development, enabling integration with third-party services, building microservices architectures, and facilitating automation and orchestration of workflows.

11.2 Introduction to Flask

Flask is a lightweight and extensible web framework for Python, designed to make web development simple and scalable. It provides tools and libraries for building web applications and APIs using Python.

Key features of Flask include:

Minimalistic: Flask has a simple and easy-to-understand syntax, making it ideal for beginners and experienced developers alike. It does not impose any dependencies or project structure, allowing developers to choose the tools and libraries they prefer.

Flexibility: Flask is highly flexible and can be extended with various extensions and plugins to add functionality such as authentication, database integration, and form validation. These extensions follow the Flask philosophy of keeping things simple and modular.

Jinja2 Templating: Flask uses the Jinja2 template engine to render dynamic HTML content. Jinja2 allows developers to create reusable templates with placeholders for dynamic data, making it easy to generate dynamic web pages.

Werkzeug WSGI Toolkit: Flask is built on top of the Werkzeug WSGI toolkit, which provides low-level utilities for handling HTTP requests and responses. This allows Flask to handle routing, request parsing, and other web-related tasks efficiently.

Overall, Flask is an excellent choice for building web applications and APIs in Python, offering simplicity, flexibility, and scalability for a wide range of use cases [38].

11.3 Introduction to MySQL Using Workbench (SQL)

MySQL is a popular open-source relational database management system (RDBMS) known for its reliability, scalability, and ease of use. It is widely used in web development, data warehousing, and various other applications where structured data storage and retrieval are required.

MySQL Workbench is a graphical user interface (GUI) tool provided by MySQL for designing, modeling, and administering MySQL databases. It offers a comprehensive set of features for database development, management, and optimization, making it an essential tool for database administrators, developers, and data analysts.

Key features of MySQL Workbench include:

Database Design and Modeling: MySQL Workbench allows users to create and modify database schemas using intuitive visual design tools. It supports the Entity-Relationship (ER) modeling approach, enabling users to define tables, relationships, indexes, and constraints easily.

Schema Management: With MySQL Workbench, users can view and manage database schemas, tables, views, stored procedures, and other database objects. They can perform tasks such as creating, editing, deleting, and renaming database objects directly from the GUI.

SQL Development: MySQL Workbench provides a built-in SQL editor with syntax highlighting, code completion, and formatting features. Users can write, execute, and debug SQL queries, scripts, and stored procedures directly within the tool, facilitating database development and testing.

Data Import and Export: MySQL Workbench allows users to import data from external sources (e.g., CSV files, Excel spreadsheets) into MySQL databases and export data from MySQL databases to various file formats. This feature simplifies the process of transferring data between different systems and applications.

Database Administration: MySQL Workbench offers administrative features for managing MySQL server instances, user accounts, privileges, and server settings. Administrators can monitor server performance, configure backup and recovery options, and troubleshoot issues using the built-in tools.

Performance Tuning: MySQL Workbench includes tools for performance tuning and optimization, allowing users to analyze query execution plans, identify bottlenecks, and optimize database performance. These tools help improve the efficiency and scalability of MySQL databases.

Overall, MySQL Workbench is a powerful and versatile tool for MySQL database development, administration, and optimization. Whether you are designing database schemas, writing SQL queries, or managing server resources, MySQL Workbench provides the tools and features you need to work effectively with MySQL databases [39].

11.4 Leveraging MySQL in Train Arrival Prediction System

In the realm of predictive analytics, the efficacy of our train arrival prediction system hinges on the robustness and efficiency of its underlying database architecture. MySQL Workbench emerged as a cornerstone tool in this endeavor, enabling us to meticulously craft a database schema tailored to our specific requirements. Through the execution of precise commands within MySQL Workbench, including the creation of the "trainarrivalprediction" schema and the "odometer" table, we established a structured data environment primed for predictive modeling tasks.

The "CREATE SCHEMA trainarrivalprediction" command laid the foundation for our database, delineating a dedicated space for housing mission-related data. This schema provided a logical container for organizing and managing various tables and data entities, ensuring clarity and coherence in our data storage approach.

Subsequently, the creation of the "odometer" table via the "CREATE TABLE" command represented a pivotal step in structuring our database schema. This table, designed to capture mission-specific information such as mission ID, last position, and distance traveled, epitomized the meticulous attention to detail inherent in our database design. By defining appropriate data types and constraints, we safeguarded data integrity while accommodating the diverse needs of our predictive modeling tasks.

Furthermore, the utilization of MySQL Workbench facilitated seamless interaction with the database, empowering us to execute queries, perform data manipulation tasks, and visualize database entities with ease. This intuitive interface expedited the development process, enabling rapid iteration and refinement of our predictive analytics system.

In essence, MySQL Workbench's pivotal role in database design and management underscored its significance as a linchpin tool in the development of our train arrival prediction system. By leveraging its capabilities to craft a meticulously structured database schema, we laid a robust foundation for predictive modeling endeavors, ensuring the accuracy, reliability, and scalability of our predictive analytics solution.

11.5 Usage of the Odometer API

The provided API is a Flask application designed to handle requests for updating the position of a vehicle's odometer in a database. Here's an overview of its functionality and components:

Import Statements:
The Flask framework is imported to create the web application.
Necessary mathematical functions such as radians, sine, cosine, and square root are imported from the math module.
The sys and os modules are imported to manage system-specific parameters and interact with the operating system, respectively.

The databaseConnection module is imported to establish a connection with the database.

Flask Application Setup:
An instance of the Flask class is created with the name '**name**'.
This initializes the Flask application, providing a framework for routing HTTP requests and defining endpoints.

Helper Function:
The haversine function is defined to calculate the distance between two geographical coordinates using the Haversine formula [40].
This function takes latitude and longitude values for two points and returns the distance between them in kilometers.

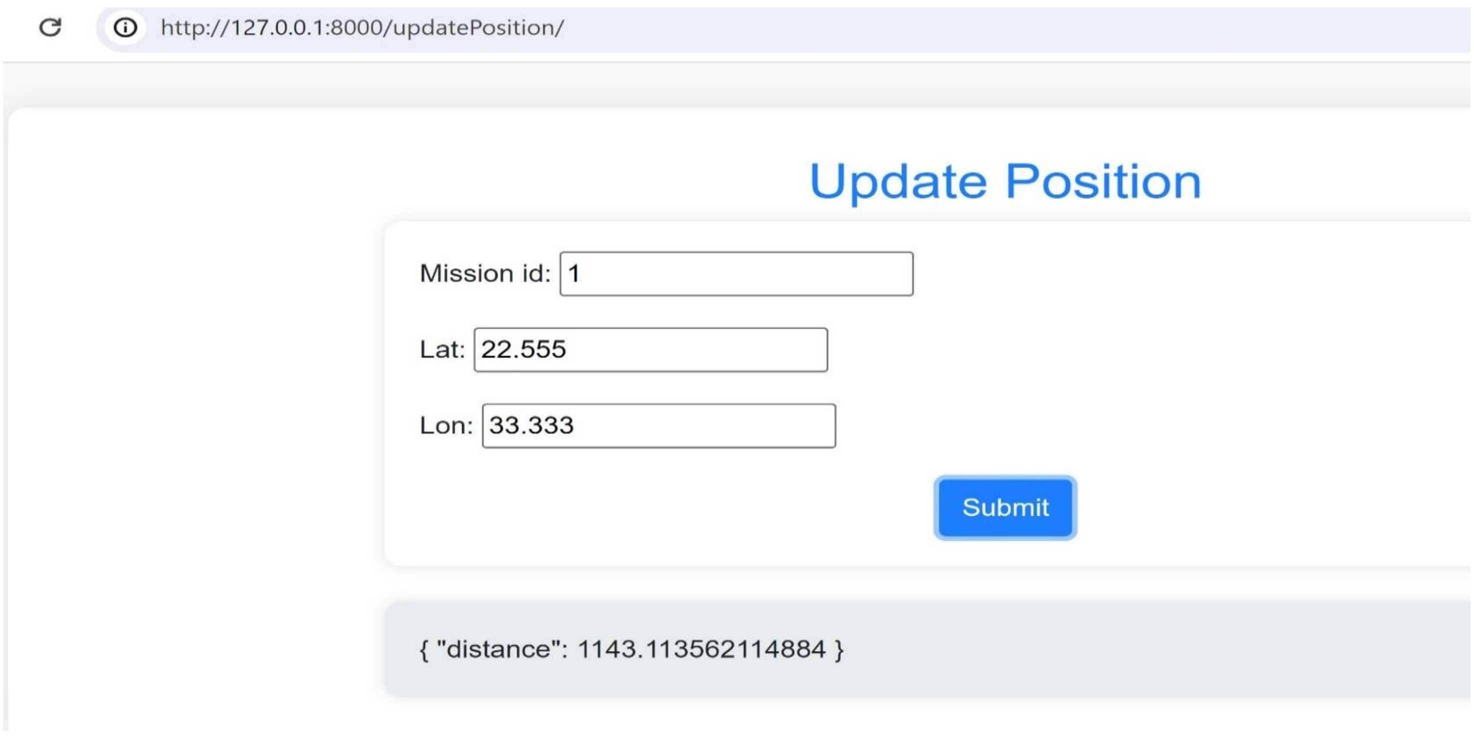
Odometer Class:
A class named Odometer is defined to manage odometer updates and interactions with the database.
It contains methods for updating the position of the odometer in the database and calculating the distance traveled.

API Endpoint:
An endpoint '/update_position' is defined to handle HTTP POST requests for updating the odometer position.
When a POST request is received, the JSON data containing the mission ID, latitude, and longitude is extracted.
An instance of the Odometer class is created, and the update_position method is called to update the position in the database.
The distance traveled is calculated and returned as a JSON response.

Main Execution Block:
The Flask application is run on port 6000 if the script is executed directly.
This allows the Flask server to listen for incoming HTTP requests and handle them accordingly.
Overall, this API provides a simple and effective means to update and track the distance traveled by vehicles using their mission IDs and GPS coordinates. It leverages Flask's lightweight and flexible framework to create a scalable and responsive web service for odometer management.

The initiation of requests through the following web page reflects a pivotal aspect of user interaction within the Django web application framework on port 8000. By embedding forms within the frontend views, users are provided with an intuitive interface to input relevant data. This not only enhances user experience but also streamlines the process of data submission for predictive analysis. Moreover, by utilizing forms, the application adheres to standard web conventions, ensuring familiarity and ease of use for users.

Additionally, the ability to send requests via external tools like Postman underscores the flexibility and extensibility of the application. Users, including developers and testers, have the autonomy to interact with the application's backend independently of the frontend interface. This facilitates efficient testing, debugging, and integration of the application with other systems or services. Furthermore, it empowers users with diverse preferences and technical backgrounds to interact with the application in a manner that best suits their needs and workflows.


http://127.0.0.1:8000/updatePosition/

Update Position

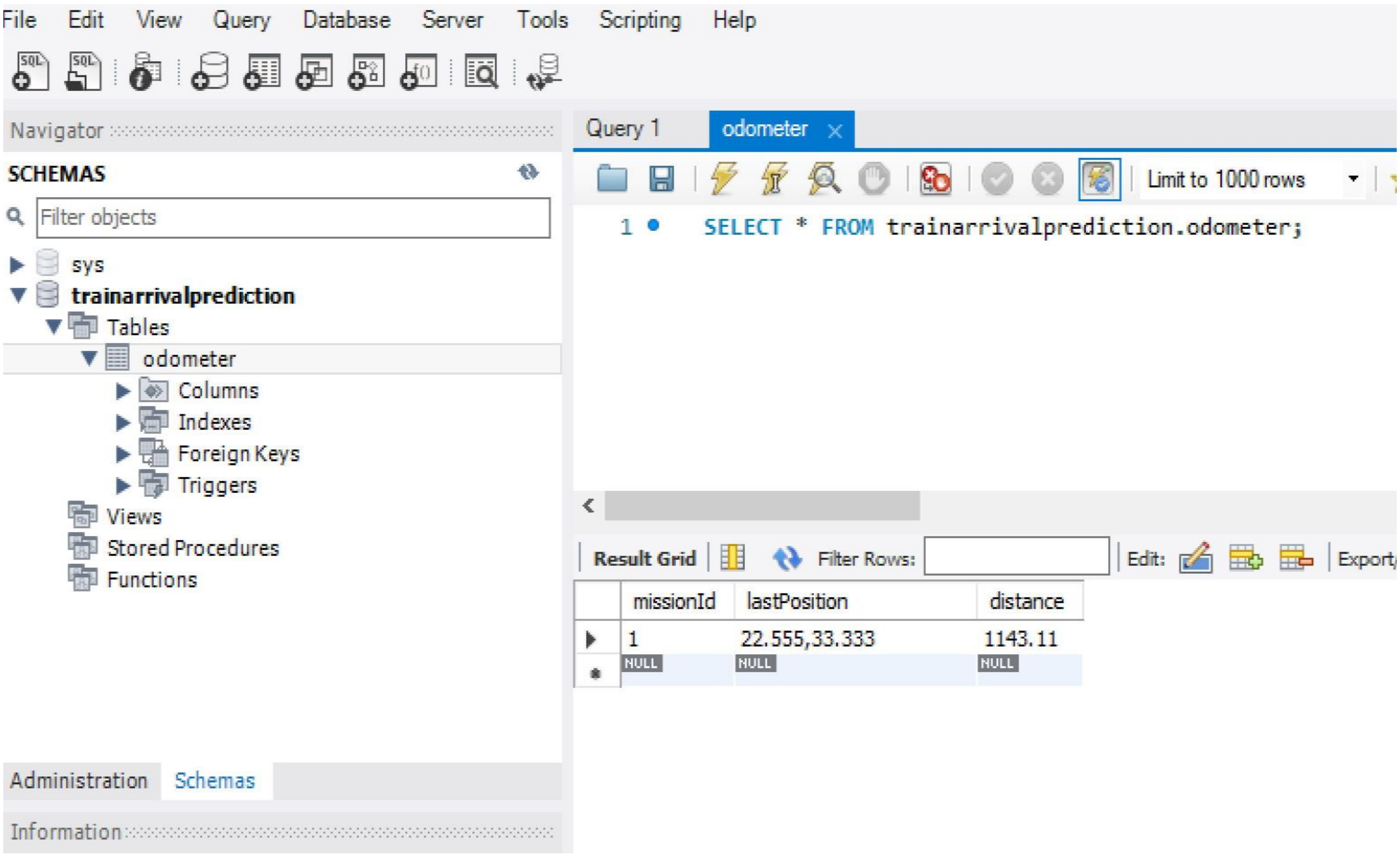
Mission id: 1

Lat: 22.555

Lon: 33.333

```
{ "distance": 1143.113562114884 }
```

Figure 9 : UI for Fetching the Covered Distance in km



```
SELECT * FROM trainarrivalprediction.odometer;
```

missionId	lastPosition	distance
1	22.555,33.333	1143.11
NULL	NULL	NULL

Figure 10 : Storage of the Covered Distance in km

11.6 Usage of the Prediction API

This code snippet is a Flask application that serves as an API endpoint for making predictions using a pre-trained machine learning model. Let's break down the key components and functionalities:

Flask Setup: The Flask framework is imported, and an instance of the Flask application is created with Flask(__name__).

Model Loading: The code loads a pre-trained machine learning model from a pickle file (random_forest_model.pkl). This model is used to make predictions based on incoming data.

API Endpoint: The /predict route is defined using @app.route('/predict', methods=['POST']). This endpoint is where clients can send POST requests with data for prediction.

Prediction Function: The predict() function is the handler for the /predict endpoint. It performs the following steps:

1. Retrieves data from the POST request using request.get_json(force=True).

2. Converts the received data into a pandas DataFrame.

3. Uses the loaded machine learning model (loaded_model) to make predictions on the input data.

4. Constructs a JSON response containing the predicted delay and returns it using jsonify(result).

Error Handling: The code includes a basic error handling mechanism to catch any exceptions that may occur during prediction. If an error occurs, it returns a JSON response containing the error message.

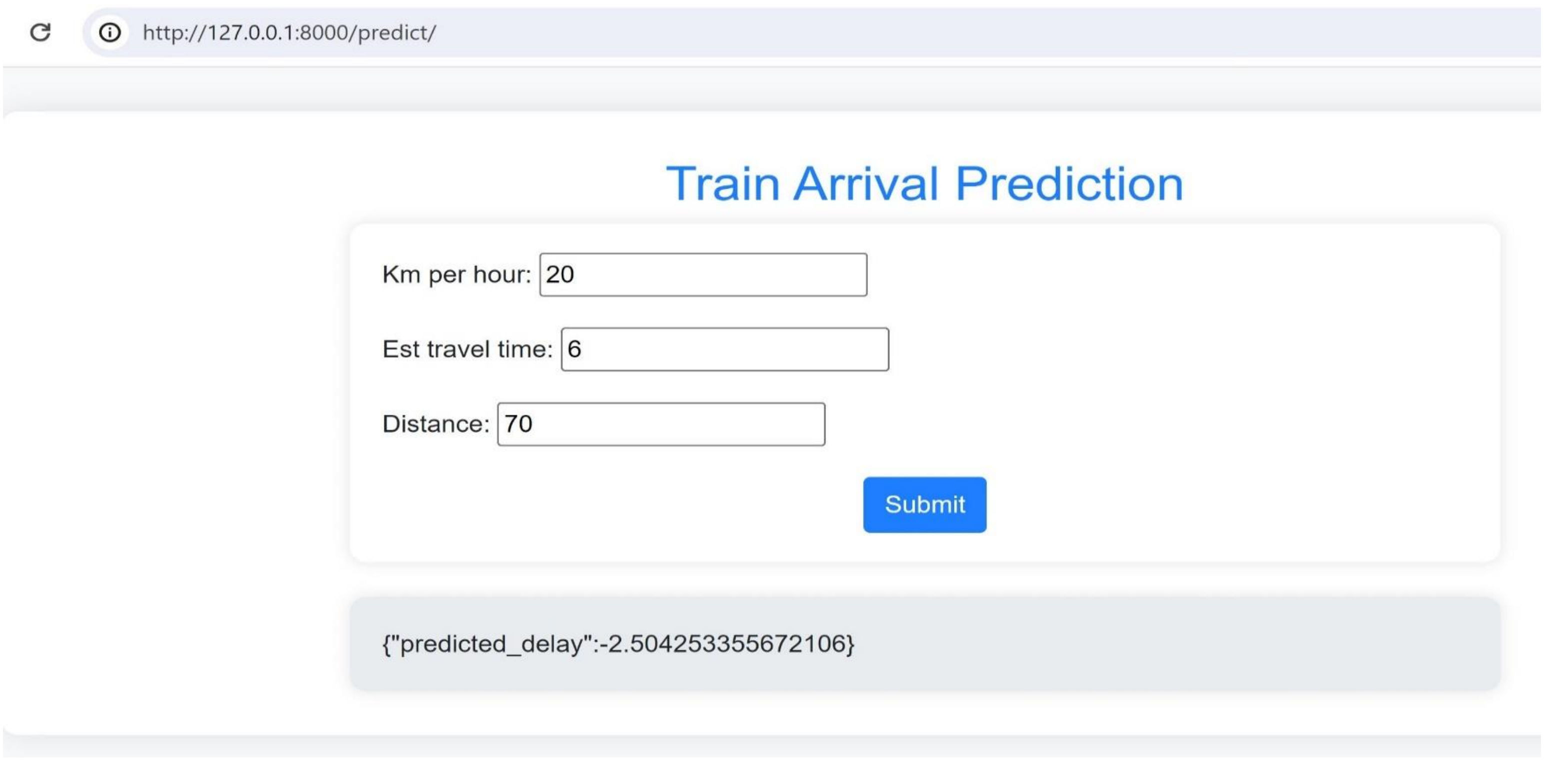
Server Initialization: The if __name__ == '__main__': block ensures that the Flask app is only run when the script is executed directly, not when imported as a module. It starts the Flask development server on port 5000 .

Overall, this code sets up a simple Flask API endpoint that can receive POST requests containing data for prediction, processes the data using a pre-trained model, and returns the prediction results to the client in JSON format.

In addition to the Flask API endpoint, there is a user interface (UI) running on port 8000 page created where users can interact with the API. This UI page allows users to input data for prediction and send requests to the API endpoint. Alternatively, users can also send requests programmatically using tools like Postman or through custom scripts.

The UI page consists of input fields (km per hour, estimated travel time in hours and distance in km) where users can enter the necessary data, along with a button or similar control to trigger the request to the API. Upon submitting the form or triggering the action, the UI page sends a POST request to the /predict endpoint of the Flask application, passing the input data as JSON in the request body.

This setup allows for both manual interaction with the API through the UI page and automated interaction via external tools or scripts. Users can choose the method that best suits their needs, whether it be through the UI for ad-hoc queries or programmatically for batch processing or integration with other systems.


http://127.0.0.1:8000/predict/

Train Arrival Prediction

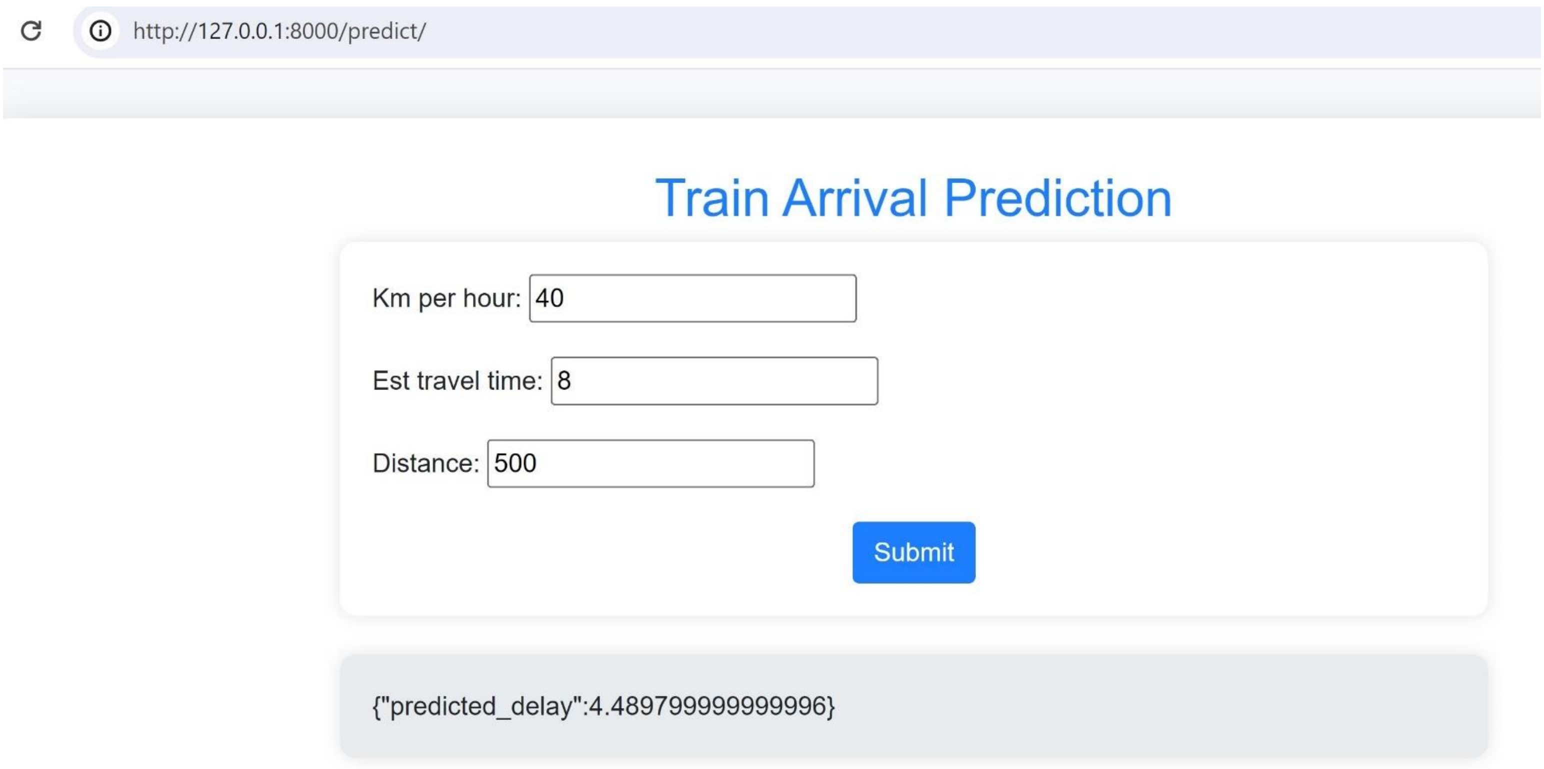
Km per hour: 20

Est travel time: 6

Distance: 70

```
{"predicted_delay":-2.504253355672106}
```

Figure 11: UI for Predicting Delay (Example 1)


http://127.0.0.1:8000/predict/

Train Arrival Prediction

Km per hour: 40

Est travel time: 8

Distance: 500

```
{"predicted_delay":4.489799999999996}
```

Figure 12 : UI for Predicting Delay (Example 2)

12. Experimental Results and Discussion

In this section, we present the experimental results obtained from the implementation of our predictive model for train arrival times. We discuss the performance metrics, analyze the findings, and explore the implications of our research.

Model Performance

The predictive model was trained and evaluated using various machine learning algorithms, including linear regression, support vector machines, decision trees, and random forest. Among these algorithms, the random forest algorithm consistently exhibited superior performance in predicting train arrival times. The model achieved a Mean Absolute Error (MAE) of 0.1319 hours, indicating its high accuracy in estimating arrival delays.

Feature Analysis

Feature selection played a crucial role in the development of the predictive model. Initially, we considered a wide range of features, including train speed ('km/h'), estimated travel time ('estTravTime'), and distance between stations ('distance'). However, extensive testing and correlation analysis revealed that objective features such as passing borders, day of the week, and seasonal variations had minimal predictive power. We hypothesized that human factors, such as driver behavior and border inspection processes, significantly influenced arrival times but were not directly measurable. Consequently, we focused on objective features related to train movement and position, leading to the inclusion of 'km/h' and 'estTravTime' in the final model.

Correlation Analysis

To gain insights into the relationships between features and arrival delays, we conducted correlation analysis. Heatmap visualization of the correlation matrix highlighted the strong positive correlation between 'km/h' and arrival delays, indicating that higher train speeds were associated with shorter delays. Additionally, 'estTravTime' exhibited a moderate negative correlation with arrival delays, suggesting that accurate estimations of travel time contributed to reduced delays.

Feature Importance

The importance of features in predicting arrival delays was assessed using the feature importances provided by the random forest model. 'Km/h' emerged as the most influential feature, followed by 'estTravTime' and 'distance'. This reaffirmed the significance of train speed in determining arrival times and emphasized the importance of real-time speed monitoring for accurate predictions.

Discussion

The experimental results underscore the efficacy of the predictive model in estimating train arrival times with high precision. By leveraging machine learning techniques and focusing on objective features related to train movement, our model overcomes the limitations of traditional methods reliant on schedule adherence alone. The insights gained from feature analysis and correlation studies provide valuable inputs for optimizing train scheduling, improving operational efficiency, and enhancing passenger satisfaction.

In conclusion, the experimental findings validate the effectiveness of our predictive model and highlight its potential applications in the railway transportation sector. Future research endeavors could explore additional factors influencing arrival times, such as weather conditions and infrastructure maintenance, to further enhance the accuracy and robustness of the model.

13. Conclusions and Future Work

13.1 Conclusions

In this study, we embarked on a comprehensive exploration of predictive modeling techniques to forecast train arrival times, aiming to enhance railway transportation operations. Through meticulous feature selection, rigorous model evaluation, and insightful analysis of experimental results, we have derived several key conclusions.

First and foremost, our experimentation with various machine learning algorithms revealed that the random forest model consistently outperformed other models in predicting train arrival times. Its robustness in handling complex datasets, mitigating overfitting, and providing interpretable insights underscore its suitability for real-world applications in the railway domain.

Moreover, our analysis uncovered the critical role of objective features such as train speed ('km/h') and estimated travel time ('estTravTime') in influencing arrival delays. By prioritizing these features in our model, we have demonstrated the efficacy of data-driven approaches in capturing the dynamic nature of train operations and optimizing scheduling decisions.

Furthermore, our correlation studies and feature importance analysis shed light on the intricate relationships between predictive features and arrival delays, offering actionable insights for enhancing operational efficiency and minimizing disruptions. The identification of 'km/h' as the most influential feature underscores the importance of speed management strategies in reducing delays and improving service reliability.

Last, we have to mention that leveraging a full-stack software architecture comprising database management, frontend and backend development, and API integration, our endeavor transcends the realm of academic research to address real-world challenges faced by the railway industry.

The deployment of our predictive model in collaboration with the railway company, which provided the dataset, underscores its practical viability and tangible impact on operational efficiency. By seamlessly integrating predictive analytics into existing railway operations, we have empowered stakeholders with actionable insights and decision support tools to optimize scheduling, minimize delays, and enhance passenger satisfaction.

13.1 Future Work

While our study has yielded valuable insights and actionable recommendations, several avenues for future research and development warrant exploration.

One promising direction involves the integration of additional data sources and external factors, such as weather conditions, passenger load, and infrastructure status, into our predictive model. By incorporating these dynamic variables, we can enhance the model's predictive accuracy and resilience to unforeseen disruptions, thereby enabling proactive decision-making and adaptive scheduling.

Furthermore, the deployment of advanced sensor technologies and IoT devices holds immense potential for real-time monitoring and predictive maintenance of railway assets. Future research endeavors could focus on leveraging these technologies to optimize asset utilization, minimize downtime, and improve overall system reliability.

Additionally, there is a growing need for collaborative research initiatives and industry partnerships to facilitate the implementation of predictive analytics solutions in the railway sector. By fostering interdisciplinary collaboration and knowledge exchange, we can accelerate the adoption of innovative technologies and drive continuous improvement in railway transportation operations.

In conclusion, our study represents a step towards leveraging predictive analytics to optimize train scheduling, enhance operational efficiency, and elevate passenger experiences. As we embark on the journey of digital transformation in the railway domain, the insights gained from this research serve as a foundation for future innovation and progress in the field.

13.1.1 Route Net : Limitations and Future Work

RouteNet [42] is a custom architecture based on Graph Neural Network (GNN) designed specifically for computer network modeling. It addresses the challenge of efficiently modeling complex relationships within Software-Defined Networks (SDNs) by leveraging Deep Learning techniques. Traditional network modeling techniques, such as analytic models like Queuing theory, struggle to handle the complexity of SDNs, limiting the effectiveness of optimization approaches.

RouteNet stands out by employing a novel message-passing function within the GNN framework. This function allows RouteNet to capture intricate relationships between the state of paths and links resulting from network topologies and routing configurations. By doing so, RouteNet can effectively model the resulting network performance, including metrics like per-source/destination per-packet delay and loss distribution.

One of the key advantages of RouteNet is its ability to generalize to diverse network scenarios, including different topologies, routing configurations, and traffic matrices not encountered during training. This generalization capability is crucial for real-world applicability, as network environments often vary widely.

Furthermore, RouteNet's modular architecture facilitates transfer learning, enabling the reuse of pre-trained neural network models for addressing related problems in similar domains. This capability streamlines the adaptation of RouteNet to new tasks and scenarios, enhancing its versatility and scalability.

In practical terms, RouteNet's performance predictions can be utilized for various network optimization purposes. For example, it can be employed for Quality-of-Service (QoS)-aware routing optimization based on specific delay, jitter, and packet loss requirements. Additionally, RouteNet can assist in optimal link placement within network planning scenarios, helping to enhance network efficiency and performance.

Overall, RouteNet represents a promising approach to network modeling and optimization in SDN environments, offering advanced capabilities to handle complexity and variability while supporting a wide range of practical applications.

In the initial stages of our research, our aim was to adapt the RouteNet model for our network optimization tasks. However, we encountered a significant obstacle in the inability of creating traffic matrices, crucial for training and validating the model. These matrices, that should have been produced from the company's data, contain essential information about network traffic patterns and dynamics, enabling accurate modeling of network performance.

Unfortunately, due to the unavailability of features that could produce these traffic matrices, primarily because they rely on subjective factors, we faced challenges in training the RouteNet model effectively. Without access to comprehensive traffic data, the model's performance could not be adequately assessed or optimized.

Looking ahead, future work in our research will focus on addressing this limitation by integrating additional features that can facilitate the creation of traffic matrices. These features will capture relevant aspects of network behavior. Incorporating these features into the model, we aim to overcome the data scarcity issue and enable the successful implementation of the RouteNet model for network optimization tasks.

In summary, while the initial attempt to adapt the RouteNet model faced obstacles due to data limitations, future endeavors will involve enhancing the model's capabilities by integrating additional features to facilitate the creation of traffic matrices. This approach will pave the way for leveraging advanced network modeling techniques and optimizing network performance effectively.

14. Bibliography

[1] David Morandi and Stephan Jüngling (2021), *Anomaly Detection in Railway Infrastructure,* University of Applied Sciences Northwestern Switzerland, School of Business, Retrieved from : https://ceur-ws.org/Vol-2846/paper2.pdf

[2] Wei-Hua Lin and Jian ZengView (1999), *Experimental Study of Real-Time Bus Arrival Time Prediction with GPS Data*, https://doi.org/10.3141/1666-1. Retrieved from : https://journals.sagepub.com/doi/abs/10.3141/1666-12

[3] F. van Wyk, Y. Wang, A. Khojandi and N. Masoud (2020), *Real-Time Sensor Anomaly Detection and Identification in Automated Vehicles*, in IEEE Transactions on Intelligent Transportation Systems, vol. 21, no. 3, pp. 1264-1276, doi: 10.1109/TITS.2019.2906038. Retrieved from :
https://ieeexplore.ieee.org/abstract/document/8684317

[4] Peyman Noursalehi, Haris N. Koutsopoulos, Jinhua Zhao (2018) *Real time transit demand prediction capturing station interactions and impact of special events*, Transportation Research Part C: Emerging Technologies, Volume 97, Pages 277-300, ISSN 0968-090X, https://doi.org/10.1016/j.trc.2018.10.023. Retrieved from :
https://www.sciencedirect.com/science/article/pii/S0968090X18301797

[5] Hasan Poonawala, PictureVinay Kolar, PictureSebastien Blandin, PictureLaura Wynter, PictureSambit Sahu (2018), *Singapore in Motion: Insights on Public Transport Service Level Through Farecard and Mobile Data Analytics*, Volume 97, Pages 589–59, https://doi.org/10.1145/2939672.2939723. Retrieved from :
https://dl.acm.org/doi/abs/10.1145/2939672.2939723

[6] Dileepa Jayakody, Mananu Gunawardana, Nipuna Wicram a Surendra, Dayan Gayasri Jayasekara, Chanaka Upendra, Rangana De Silva (2012), *GPS/GSM Based Train Tracking System–Utilizing Mobile Networks To Support Public Transportation.* Retrieved from :
https://www.semanticscholar.org/paper/GPS-GSM-based-train-tracking-system-%E2%80%93-utilizing-to-Jayakody-Gunawardana/349a76333d57acdd9cf0c431f54353915f84b9d7

[7] Kah Yong Tiong, Zhenliang Ma, Carl-William Palmqvist (2023) *Analyzing factors contributing to real-time train arrival delays using seemingly unrelated regression models,*
Transportation Research Part A: Policy and Practice, Volume 174, 103751, ISSN 0965-8564, https://doi.org/10.1016/j.tra.2023.103751. Retrieved from :
https://www.sciencedirect.com/science/article/pii/S0965856423001714

[8] Nikola Marković, Sanjin Milinković, Konstantin S. Tikhonov, Paul Schonfeld (2015), *Analyzing passenger train arrival delays with support vector regression*, Transportation Research Part C: Emerging Technologies, Volume 56, Pages 251-262, ISSN 0968-090X, https://doi.org/10.1016/j.trc.2015.04.004. Retrieved from :
https://www.sciencedirect.com/science/article/pii/S0968090X1500145X

[9] Xiong, Z., Sheng, H., Rong, W. et al. (2012*), Intelligent transportation systems for smart cities: a progress review*. pages 2908–2914. https://doi.org/10.1007/s11432-012-4725-1. Retrieved from :
https://link.springer.com/article/10.1007/s11432-012-4725-1

[10] Denis Ushakov, Egor Dudukalov, Larisa Shmatko, Khodor Shatila (2022), *Artificial Intelligence as a factor of public transportations system development,* Transportation Research Procedia, Volume 63, Pages 2401-2408, ISSN 2352-1465, https://doi.org/10.1016/j.trpro.2022.06.276. Retrieved from :
https://www.sciencedirect.com/science/article/pii/S2352146522005312

[11] Noussan, M., Hafner, M., Tagliapietra (2020), *The Evolution of Transport Across World Regions. In: The Future of Transport Between Digitalization and Decarbonization,* https://doi.org/10.1007/978-3-030-37966-7_1. Retrieved from :
https://link.springer.com/book/10.1007/978-3-030-37966-7

[12] Kapur, R., Das, S., Nandineni, R.D. (2021), *Sustainable Transportation Infrastructure's Role in Attaining Sustainable Development Goals*, In: Leal Filho, W., Azul, A.M., Brandli, L., Lange Salvia, A., Wall, T. (eds) Industry, Innovation and Infrastructure. Encyclopedia of the UN Sustainable Development Goals. Springer, Cham. https://doi.org/10.1007/978-3-319-95873-6_52. Retrieved from :
https://link.springer.com/referenceworkentry/10.1007/978-3-319-95873-6_52#citeas

[13] Tingting Yuan, Wilson Borba da Rocha Neto, Christian Rothenberg, Katia Obraczka, Chadi Barakat, et al. (2019), *Harnessing Machine Learning for Next-Generation Intelligent Transportation Systems: A Survey.* hal-02284820. Retrieved from :
https://intrig.dca.fee.unicamp.br/wp-content/papercite-data/pdf/rothenberg2019hal.pdf

[14] Silvia Anna Chiusano, Luca Cagliero, Elena Daraio (2022), *Mining user reviews on public transport systems using machine learning techniques.* Politecnico di Torino, Corso di laurea magistrale in Data Science And Engineering, Retrieved from :
https://webthesis.biblio.polito.it/22790/

[15] Vaddy, R. krishna. (2023), *AI and ML for Transportation Route Optimization. International Transactions in Machine Learning,* 5(5), 1–19. Retrieved from https://isjr.co.in/index.php/ITML/article/view/200

[16] Ang, Kenneth Li-Minn, Jasmine Kah Phooi Seng, Ericmoore Ngharamike, and Gerald K. Ijemaru. (2022), *Emerging Technologies for Smart Cities' Transportation: Geo-Information, Data Analytics and Machine Learning Approaches*, ISPRS International Journal of Geo-Information 11, no. 2: 85. https://doi.org/10.3390/ijgi11020085 Retrieved from :
https://www.mdpi.com/2220-9964/11/2/85

[17] Kalyan Das, Jiming Jiang J., N. K. Rao (2004), "Mean squared error of empirical predictor." Ann. Statist. 32 (2) 818 – 840 . https://doi.org/10.1214/009053604000000201. Retrieved from : https://arxiv.org/pdf/math/0406455.pdf

[18] T. Chai and R. R. Draxler (2014), *RMSE or MAE,* GMDD 7, 1525–1534, Retrieved from :
https://gmd.copernicus.org/preprints/7/1525/2014/gmdd-7-1525-2014.pdf

[19] Cort J. Willmott and Kenji Matsuura (2005), *Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance.* Retrieved from :
https://www.int-res.com/abstracts/cr/v30/n1/p79-82

[20] Guoqiang Cai, Jiaqing Zhao, Qiong Song, Mengchu Zhou (2019), *System architecture of a*

train sensor network for automatic train safety monitoring, Computers & Industrial Engineering, Volume 127, Pages 1183-1192, ISSN 0360-8352, https://doi.org/10.1016/j.cie.2018.04.038. Retrieved from :
https://www.sciencedirect.com/science/article/pii/S0360835218301827

[21] D. F.N. Oliveira et al. (2019), *Evaluating Unsupervised Anomaly Detection Models to Detect Faults in Heavy Haul Railway Operations*, 18th IEEE International Conference On Machine Learning And Applications (ICMLA), Boca Raton, FL, USA, 2019, pp. 1016-1022, doi: 10.1109/ICMLA.2019.00172. Retrieved from :
https://ieeexplore.ieee.org/abstract/document/8999291

[22] Neehar Peri, Pirazh Khorramshahi, Sai Saketh Rambhatla, Vineet Shenoy, Saumya Rawat, Jun-Cheng Chen, Rama Chellappa (2020) *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pp. 622-623. Retrieved from :
https://ieeexplore.ieee.org/xpl/conhome/1000147/all-proceedings

[23] Andreas Balster, Ole Hansen, Hanno Friedrich & André Ludwig (2020), *An ETA Prediction Model for Intermodal Transport Networks Based on Machine Learning,* Volume 62, pages 403–416, Retrieved from :
https://link.springer.com/article/10.1007/s12599-020-00653-0

[24] Kolawole Ogunsina, Ilias Bilionis, Daniel DeLaurentis (2021), *Exploratory data analysis for airline disruption management, Machine Learning with Applications*, Volume 6, 100102, ISSN 2666-8270, https://doi.org/10.1016/j.mlwa.2021.100102. Retrieved from :
https://www.sciencedirect.com/science/article/pii/S2666827021000517

[25] L. Zhu, F. R. Yu, Y. Wang, B. Ning and T. Tang (2019), *Big Data Analytics in Intelligent Transportation Systems: A Survey, in IEEE Transactions on Intelligent Transportation Systems*, vol. 20, no. 1, pp. 383-398, Jan., doi: 10.1109/TITS.2018.2815678. Retrieved from :
https://ieeexplore.ieee.org/abstract/document/8344848

[26] Diego Didona, Francesco Quaglia, Paolo Romano, Ennio Torre (2015*), Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering*, Pages 145–156
https://doi.org/10.1145/2668930.2688047. Retrieved from :
https://dl.acm.org/doi/abs/10.1145/2668930.2688047

[27] S. Yang and S. Qian (2019), *Understanding and Predicting Travel Time with Spatio-Temporal Features of Network Traffic Flow, Weather and Incidents*, in IEEE Intelligent Transportation Systems Magazine, vol. 11, no. 3, p. 12-28, doi: 10.1109/MITS.2019.2919615. Retrieved from :
https://ieeexplore.ieee.org/abstract/document/8742547

[28] Mahajan, P.C., Kiwelekar, A.W., Netak, L.D., Ghodake, A.B. (2022*). Predicting Expected Time of Arrival of Shipments Through Multiple Linear Regression*. In: Kumar, A., Senatore, S., Gunjan, V.K. (eds) ICDSMLA 2020. Lecture Notes in Electrical Engineering, vol 783. Springer, Singapore. https://doi.org/10.1007/978-981-16-3690-5_30. Retrieved from :
https://link.springer.com/chapter/10.1007/978-981-16-3690-5_30

[29] Kosolsombat, S., Limprasert, W. (2017). *Arrival Time Prediction and Train Tracking Analysis*. In: Numao, M., Theeramunkong, T., Supnithi, T., Ketcham, M., Hnoohom, N., Pramkeaw, P. (eds) Trends in Artificial Intelligence: PRICAI 2016 Workshops. PRICAI 2016.

Lecture Notes in Computer Science(), vol 10004. Springer, Cham. https://doi.org/10.1007/978-3-319-60675-0_15. Retrieved from :
https://link.springer.com/chapter/10.1007/978-3-319-60675-0_15

[30] J. Pineda-Jaramillo, F. Bigi, T. Bosi, F. Viti and A. D'ariano (2023), *Short-Term Arrival Delay Time Prediction in Freight Rail Operations Using Data-Driven Models*, in IEEE Access, vol. 11, pp. 46966-46978, doi: 10.1109/ACCESS.2023.3275022. Retrieved from :
https://ieeexplore.ieee.org/abstract/document/10122504

[31] Yaghini, M., Khoshraftar, M.M. and Seyedabadi, M. (2013), *Railway passenger train delay prediction via neural network model*. J. Adv. Transp., 47: 355-368. https://doi.org/10.1002/atr.193. Retrieved from :
https://onlinelibrary.wiley.com/doi/abs/10.1002/atr.193

[32] Wei Shao, Arian Prabowo, Sichen Zhao, Piotr Koniusz, Flora D. Salim (2022), *Predicting flight delay with spatio-temporal trajectory convolutional network and airport situational awareness map*, Neurocomputing, Volume 472, Pages 280-293, ISSN 0925-2312,
https://doi.org/10.1016/j.neucom.2021.04.136. Retrieved from :
https://www.sciencedirect.com/science/article/pii/S092523122101612X

[33] Ho, T.K. (1995), *Random Decision Forest. Proceedings of the 3rd International Conference on Document Analysis and Recognition, Montreal, 14-16 August 1995, 278-282.* Retrieved from: https://www.scirp.org/reference/ReferencesPapers?ReferenceID=1698778

[34] M. Maleshkova, C. Pedrinaci and J. Domingue (2010), *Investigating Web APIs on the World Wide Web*, Eighth IEEE European Conference on Web Services, Ayia Napa, Cyprus, pp. 107-114, doi: 10.1109/ECOWS.2010.9. Retrieved from :
https://ieeexplore.ieee.org/abstract/document/5693251/

[35] Gamez-Diaz, A., Fernandez, P., Ruiz-Cortes, A. (2017). *An Analysis of RESTful APIs Offerings in the Industry*. In: Maximilien, M., Vallecillo, A., Wang, J., Oriol, M. (eds) Service-Oriented Computing. ICSOC 2017. Lecture Notes in Computer Science(), vol 10601. Springer, Cham. https://doi.org/10.1007/978-3-319-69035-3_43 . Retrieved from :
https://link.springer.com/chapter/10.1007/978-3-319-69035-3_43#citeas

[36] Anshu Soni and Virender Ranga (2019), *API Features Individualizing of Web Services: REST and SOAP* ,International Journal of Innovative Technology and Exploring Engineering (IJITEE), ISSN: 2278-3075, Volume-8, Issue-9S. Retrieved from :
https://www.researchgate.net/profile/Virender-Ranga/publication/335419384_API_Features_Individualizing_of_Web_Services_REST_and_SOAP/links/5d64960ea6fdccc32cd31171/API-Features-Individualizing-of-Web-Services-REST-and-SOAP.pdf

[37] Anthony Morris (2021). *The Value of Internal APIs*. Retrieved from :
https://medium.com/front-end-weekly/the-value-of-internal-apis-5a75fdd7a6f

[38] Grinberg, M. (2018), *Flask web development*

[39] Krogh and J.W. (2020), *MySQL Workbench. In: MySQL 8 Query Performance Tuning*. Apress, Berkeley, CA. https://doi.org/10.1007/978-1-4842-5584-1_11. Retrieved from :

https://link.springer.com/chapter/10.1007/978-1-4842-5584-1_11

[40] Sinnott, R. (1984), *Virtues of the Haversine. Sky & Telescope, 68, 158*. Retrieved from: https://www.scirp.org/reference/referencespapers?referenceid=2500014

[41] Yiu, T. (2019), *Understanding Random Forest*. Retrieved from: https://towardsdatascience.com/understanding-random-forest-58381e0602d2

[42] Krzysztof Rusek, Jose Suarez-Varela, Paul Almasan, Pere Barlet-Ros, Albert Cabellos-Aparicio (2020), *RouteNet : Leveraging Graph Neural Networks for Network Modeling and Optimization in SDN.* Retrieved from University of Piraeus.

[43] Andre Burgaud (2022), *How to use Python Lambdas Functions.* Retrieved from : https://realpython.com/python-lambda/

[44] Eashaan Rao, Sridhar Chimalakonda (2020), *An exploratory study towards understanding Lambda Expressions in Python.* Retrieved from: https://dl.acm.org/doi/10.1145/3383219.3383255

[45] Stefan Van der Walt (2010), *Proceedings of the 9th Python in Science Conference.* Retrieved from: https://dl.acm.org/doi/10.1145/3383219.3383255

[46] Van Rossum, G. (2020), *The Python Library Reference, release 3.8.2*. Python Software Foundation.

[47] Mohd Hakimi Zohari (2021), *GPS Vehicle Tracking System*. Retrieved from: https://www.researchgate.net/publication/352559892_GPS_Based_Vehicle_Tracking_System

[48] Jianxin Deng (2013), *Architecture Design of the Vehicle Tracking System Based in RFID*. Retrieved from: https://www.researchgate.net/figure/Architecture-of-the-Vehicle-Tracking-System-Based-on-RFID_fig2_307648775

[49] Rifkie Primartha and Bayu Adhi Tama (2017), *Anomaly Detection Using Random Forest : A Performance Revisited*. Retrieved from: https://www.researchgate.net/figure/Architecture-of-the-Vehicle-Tracking-System-Based-on-RFID_fig2_307648775