WIRTSCHAFTS UNIVERSITÄT WIEN VIENNA UNIVERSITY OF ECONOMICS AND BUSINESS

EFMD EQUIS ACCREDITED

Bachelor Thesis

Data Interoperability in European Railway Information Exchange: A Study of GTFS and NeTEx

Fredrik Massmann

Date of Birth: 26.09.2001
Student ID: h12117343

Subject Area: Information Business

Studienkennzahl: h12117343

Supervisors: Sharom Hosseini Sohi and Dr. Amin Anjomshoaa

Date of Submission: 30. September 2025

Institute of Data, Process & Knowledge Management, Vienna University of Economics and Business, Welthandelsplatz 1, 1020 Vienna, Austria

DEPARTMENT FÜR WIRTSCHAFTSINFORMATIK UND OPERATIONS MANAGEMENT DEPARTMENT OF INFORMATION SYSTEMS AND OPERATIONS MANAGEMENT

Contents

List of Figures

List of Tables

Abstract

This thesis examines the interoperability of European railway data by comparing the General Transit Feed Specification (GTFS), a widely used global standard, with the Network Timetable Exchange (NeTEx), the official EU standard. While rail is the most sustainable mode of long-distance travel, fragmented data standards have the potential to reduce its attractiveness compared to air and road transport. The study utilised datasets from Austria, Norway and Luxembourg to analyse stations and trips, with a view to evaluating structural differences, data quality and identifier consistency. The methods employed include data extraction, merging, and cross-validation with a third source (Wikidata) to verify accuracy. The results demonstrate that while GTFS offers simplicity and global adoption, NeTEx provides richer detail but suffers from inconsistent implementation across countries. Key issues include mismatched identifiers, inaccurate coordinates, and incomplete station referencing. The findings emphasise the necessity for harmonised identifiers, such as UIC codes, and coordinated efforts to enhance data quality...

1 Introduction

Travelling by train across national borders within Europe is a task that presents a considerable challenge. The following factors are cited by passengers as justifications for their preference to travel by car or plane rather than by rail: delays, inflexibility, unsatisfactory customer service, longer travel time, and the unintegrated system of rail travel across European countries. In order to illustrate the issue of customer complaints, the following data will be presented: the journey time from Amsterdam to Copenhagen by plane is 1 hour and 20 minutes, whereas the journey time by train is 12 hours and 10 minutes. It is evident that there is a substantial discrepancy in terms of travel time and customer comfort. Nevertheless, it is clear that rail travel is the most environmentally sustainable mode of transportation. It is particularly salient to compare long-distance travel with air travel. It is therefore essential to ensure that the rail network offers greater appeal to passengers in order to facilitate the transition towards a more sustainable future. Ovenhagen, 2021

The European Commission has announced its intention to enhance the passenger rail system by leveraging the existing EU regulation and policy framework for rail. The implementation of the a single european rail area within all member nations is a key objective of the European Union. In order to achieve this ambitious objective, the European Commission adopted an action plan that encompasses the modifications of the Trans-European Transport Network (TEN-T). “Action Plan to Boost Passenger Rail - European Commission”, 2025 In order to achieve the objective of integrating each individual transportation system into a unified entity, there is a necessity for standardisation. “Trans-European Transport Network (TEN-T) - European Commission”, 2025

On the one side, there was a requirement for standardisation from companies and an early user perspective. According to Goldstein’s and Dyson’s book, in 2005 major tech companies in the USA were already offering navigation applications such as Google Maps, MapQuest and Yahoo, which were designed to assist users. However, it appears that the platforms had a lack of public-available data. Following the development of standards by Google and the subsequent global rollout of the standard for public transit data, it has become a worldwide standard. The standard developed by Google has

become an open source project of mulitple companies and communitys, this standard is today known as GTFS. Goldstein and Dyson, 2013

On the other side, the EU's approach differs in that it is not only interested in creating a standard for all member state, but in establishing a comprehensive information infrastructure. The National Access Points initiative is a data infrastructure project that aims to make public data available from designated national access points. These infrastructures are designed to provide data on all types of transportation. "National Access Points - European Commission", 2025 The objective is to establish a standardised, open data space within the EU for the efficient exchange of mobility data. "Unlocking the Potential of Mobility Data | Shaping Europe's Digital Future", 2025 As outlined in the Commission's delegated regulation of May 2017, the use of data formats such as NeTEx and the underlying data system Transmodel is required for National Access Points. European Commission, 2017

The following formats are utilised for the purpose of such data exchange: GTFS and NeTEx. The primary function of the first one is the exchange of public transport timetables and stops, while the second is designed for the transfer of more complex public transport data. Skille, 2024 In this Thesis both data standards NeTEx and GTFS will be analyzed in terms of their useability and quality of their relying information.

2 The Benefits of Data Standards

Data standards play a crucial role in the European vision to achieve climate neutrality in the railway sector by 2050. Standardisation is intended to integrate disparate mobility sectors, thereby enhancing the efficiency and sustainability of the European transport system and promoting interconnectedness. The implementation of a robust standard that is widely adopted throughout Europe has the potential to enhance the size of the internal market, while concurrently ensuring legal certainty and maintaining the global or regional leadership in technology. From an economic perspective, the fundamental purpose of standards is to facilitate the development of more straightforward and accessible interfaces for European companies. Nevertheless, over the course of the previous decade, there has been an increase in the pressure exerted by companies from non-European countries throughout standards. "International Standardisation: The European Rail Associations Vison", 2021

2.1 Data Standards in the European Union

Since 2014, NeTEx has been in development. According to the official NeTEx Transmodel website, it has already been implemented in 15 European states and the United Kingdom. “NeTEx – Transmodel”, 2025 “Data Models”, 2025 Nevertheless, a more detailed perspective will provide a report for National Access Points by Napcore. According to Napcore, the NeTEx is already in operation in 10 European Union states . The remainder of the project is still in various stages of development. “Activity WG3 NAP Content and Accessibility | NAPCORE”, 2024 In comparison, GTFS has been implmeneted in over 100 countries and 10,000 agencies on a global scale. “Why Use GTFS? - General Transit Feed Specification”, 2025 The term is frequently cited as a de facto standard, given its extensive utilisation in millions of application since 2006. Antrim and Barbeau, 2017

In the following discussion, a concise overview of the two distinct standards, NeTEx and GTFS , will be provided. The principal differences between them will be identified, as well as their respective benefits. The present study will concentrate on the utilisation of both standards in the analysis of stations and trips, as well as an international comparison of GTFS and NeTEx data via three member states of the European Union. The countries under discussion here are Austria, Norway and Luxembourg.

Austria and Norway are already using NeTEx data files on an operational level and Luxembourg is in an unknown state of developing according to Napcore. “Activity WG3 NAP Content and Accessibility | NAPCORE”, 2024. Nevertheless, the three states in question have consistently published NeTEx and GTFS files on their designated national access point, national rail operator or european websites. “ÖBB Open Data Datensätze”, 2025 “MVO - Datenbereitstellungsplattform”, 2025 “Stops- and Timetable Data”, 2025 “Data.Europa.Eu”, 2025 “Horaires et Arrêts Des Transport Publics (GTFS) - Portail Open Data”, 2025

2.2 NeTEx - the EU standard

NeTEx is an Extensible Markup Language (XML)-based open data standard that was developed by the Comité Européen de Normalisation (CEN) with a view to facilitating the exchange of public transport schedules and associated data. The following information is to be converted into XML format: public transport network topology, scheduled timetables, fare information, european passenger information profile, alternative modes exchange format

and european passenger information accessibility profile. Skille, 2024 "NeTEx – Transmodel", 2025

The NeTEx framework is based on three distinct models: Transmodel, Standard Interface for Real-time Information (SIRI) and Identification of Fixed Objects in Public Transport (IFOPT). The initial Transmodel constitutes a conceptual data model for public transport, thus providing an architectural framework for comprehending the information within public transport. It is argued that this data model should represent a more simple architecture for public service companies and operators. The Transmodel has the capacity to accommodate a wide range of public transport operations, encompassing various modes such as buses, trolleybuses and light rail systems, including subways. Soares and Martins, 2013

The IFOPT model is a reference model that is utilised for the identification of fixed objects in the real world. It accomplishes this by providing an identifier for the object in question, determining its function, establishing a topology for the objects in relation to each other, linking attributes and properties to them, and localising them unambiguously by their coordinates. The fundamental function of the IFOPT is encompassing the scheduling of timetables through stop identification, journey planning, guidance, accessibility, real-time information and navigation. This encompasses stop finding, interchange paths and walking to points of interest. Soares and Martins, 2013

NeTEx is intended to serve as a model for the representation of public transport concepts. This objective should include enhancing the efficiency and updateability of the system, facilitating a complex exchange of data between two systems, and utilising it in modern web services architectures. The latter would render it more usable for passenger information and operational applications. Soares and Martins, 2013 In accordance with the principles outlined in the 04 NeTEx Framework, NeTEx is designed to exhibit a high degree of flexibility. This adaptability is a key advantage, allowing it to meet the diverse requirements of different organisations. It is important to note that different sections of an organisation may be responsible for different parts of the data. Nicholas JS Knowles, 2015 For instance, the organisation of the regional Austrian railway system is the responsibility of seven different regional operators, each of which is responsible for the management of their respective region and the provision of data pertaining to that region. "Verkehrsverbünde in Österreich", 2025

2.2.1 NeTEx Structure

The structure of NeTEx is an XML file which is more flexible and adoptable since the nodes of the netex tree structure Knowles, 2024 are adaptable and modular for different stake holders. NeTEx is equipped with a framework containing frames which defines basic components. Each Frame defines the components. Each component must contain a specific set of elements, as defined by the frame. Each component can be augmented with additional elements as required, provided that the base elements of the component remain unchanged. Each Frame is specialized on a specific function the site frame for stop data, the timetable frame for timetable and the fare frame for fara data. Nicholas JS Knowles, 2015 Therefore, each analysed data set from the different states could provide a different structure, with some cases containing more information than others. For instance Austria has two separate files for Stations and Journeys as well as different of sources. The ÖBB provides the Geo Data set, which focuses on geometric data "ÖBB Open Data Datensätze", 2025, while the Mobilitäts Verbünde Österreich's data set contains journeys, including ÖBB journeys "MVO - Datenbereitstellungsplattform", 2025. The Norway netex data sets and Luxembourg data sets contains all information in one data set "Stops- and Timetable Data", 2025 "Data.Europa.Eu", 2025.

As NeTEx is an XML structured file and designed by frames within a framework, each element has a domain in which it can be distinctly identified. Therefore each element can be called by a specific domain. Fabrizio Arneodo, 2015 For instance, the domain used for this analysis was "http://www.netex.org.uk/netex//netex:". In addition, it was necessary to add an element name to the end of the link to specify the domain.

2.3 GTFS: the defacto standard

GTFS is an open standard that provides a standardised data format for public transit agencies. The GTFS format is employed for the purpose of describing data by the inclusion of information such as stops or fares. Its utilisation is most prevalent in the domain of trip planning. GTFS comprises two constituent elements. The GTFS Schedule and GTFS Realtime.

The first one contains information about routes, schedules, fares, and geographic transit details. The GTFS schedule is a simple text file format that is contained within a zip file. According to the MobilityData, the utilisation of this system is expected to be more straightforward, as it does not require the use of proprietary software. The second one comprises information that

is capable of being updated with regard to vehicle position, service alerts and trip changes. The software utilises the Protocol Buffers format, which represents an alternative version of XML files "Protocol Buffers", 2025. There is a symbiotic relationship between GTFS Realtime and GTFS Schedule. "What Is GTFS? - General Transit Feed Specification", 2025 "Overview - General Transit Feed Specification", 2025

3 Methodes

The analysis of both data formats was conducted using Python, a programming language that supports the processing of fundamental data formats such as XML and CSV. "Xml.Etree.ElementTree — The ElementTree XML API", 2025 "Pandas.Read_csv — Pandas 2.3.2 Documentation", 2025

3.1 Loading the Data

Initially, it is imperative to load the data into the programming environment in order to facilitate its processing and utilisation. Nevertheless, an initial discrepancy emerges between the NeTEx and GTFS formats. GTFS is stored in the form of a single zip file, which contains multiple text files. "Reference - General Transit Feed Specification", 2025 The text files are processed through a looping procedure, whereby they are converted into dataframes and stored in a dictionary, with their corresponding name tags. NeTEx 6.2

In comparison, NeTEx files consist of multiple XML files or zipped folders containing further XML files or folders. It appears that the information regarding the manner in which NeTEx files are stored remains undisclosed. In order to address the challenges posed by these customised and random network file structures, a recursive loop is required. This loop selects every entry in the data file. In the case of an XML file, it will be added a list. Conversely, if the file is a folder, the function recursively calls itself to add all XML files. The NeTEx XML files are stored into a list to preprocess them for the next step the extraction of information. 6.2

3.2 Extraction of Stations

The process of extracting GTFS information involves the utilisation of a "stop" named dataframe, which serves as the repository for the stored infromation of rail stations "Reference - General Transit Feed Specification", 2025 The utilisation of a simple name filter, facilitated by the process of looping

through the zipped file, enables the selection of the desired text file. As the GTFS files have already been converted into a dataframe, the extraction process is straightforward. However, it should be noted that GTFS does not include an identification to determine the type of stop in question. For instance, it is not possible to distinguish between a bus stop and a train station by only examining the stop dataframe. As the present thesis is exclusively concerned with a consideration of train stations, it is necessary to filter the "stop" dataframe. The GTFS format provides a link between the "stop" file and the "route" file, with the "route type" identifying the type of route. The "route type"field comprises an ID ranging from zero to seven and eleven and twelve, which provides a means of distinguishing between the various vehicle types that are utilised to operate specific routes. "Reference - General Transit Feed Specification", 2025 However, it appears that an extended version of these IDs also exists. According to the provided definition, all "route types" that commence with the numerals 1 or 4 and consist of three characters are considered to be associated with rail transport. "Extended GTFS Route Types | Static Transit", 2025 In the context of the present study, numerical values beginning with 1 or 4 and comprising three characters, in addition to the numbers 1, 2, 5, 7 and 12, have been identified as correlating with a vehicle associated with any kind of rail transport. 6.2

In order to establish a connection between the designated "route type" and the designated stops, it is necessary to refer to the link provided below. The route dataframe comprises a "route type", which is connected to a "route id". The "route id" in question refers to the "trip" dataframe. The "trip" dataframe is linked to the "stop times" dataframe by their "Trip id". The "stop times" field contains a "Trip id" and a "stop id", which refer to the stop dataframe. "Reference - General Transit Feed Specification", 2025 Following fig is illustrated above desribed connection 1. All the necessary dataframes were loaded, after which the merge functions were utilised in order to establish the previously described filter. 6.2

In comparison to GTFS, the NeTEx extraction process is different due to the different XML format. The XML format is distinguished by its provision of a deep nested structure, in contrast to the more flat structure characteristic of the Comma Separated Values) (CSV) format. Both formats are illustrated 2. Knowles, 2024

As previously stated, the NeTEx file structure is more complex, since the result of the loading function was a list of multiple XML files, with a number reaching into the thousands. In order to extract all the necessary information, it is necessary to iterate through the list of XML files and search for

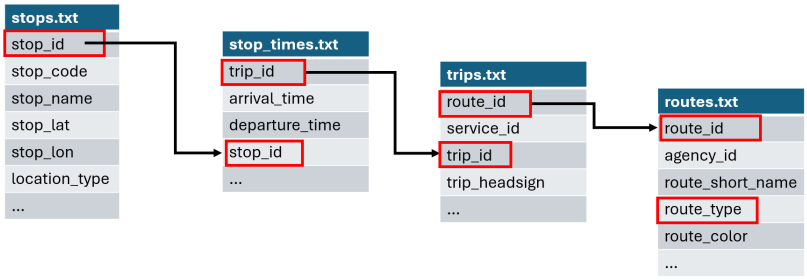
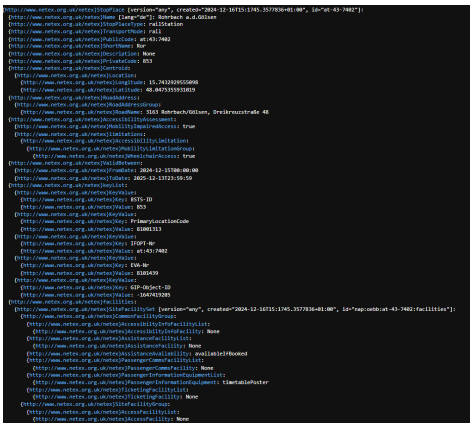


Figure 1: [Illustrated Link between Stops and Route in GTFS. Source: Self-created graphic, based on descriptions "Reference - General Transit Feed Specification", 2025



(a) NeTEx Structure of StopPlaces

	stop_id	stop_name	stop_lat	stop_lon	zone_id	location_type	parent_station	level_id	platform_code
2109	at:43:7402:0:1	Rohrbach an der Gölsen Bahnhof	48.047868	15.742544	NaN	NaN	Pat:43:7402	Level 0	1

(b) GTFS Structure already transformed into a dataframe because it is already a csv file

Figure 2: Comparison of two data structures: Nested Structure vs. Flatt Structure Source: Screenshot made from Notebook

the following “StopPlace“ Key. As mentioned above, we searched through the XML files using the domain links to find our key. Nicholas JS Knowles, 2015 It is important to note that all XML files are defined as containing fundamental and indispensable information. For instance, the coordinates of the station are stored in a nested structure into the “StopPlace“ node, firstly in the “Centroid“ node, and secondly in the “Location“ node. It is evident that the latitude and longitude elements within the “Location“ node contain the requested values. “Stops - Håndbok N801 (SIRI/NeTEX) - Entur”, 2024 “Stop Places - NeTEx Profil Österreich - Mobilitätsverbünde”, 08/09/2025, 18:06:33 However, it has been observed that supplementary nodes and elements can be appended and that these contain further information in addition to other data sets. For instance, the Austria Geo Data set contains a greater number of elements per station than the other data set. “Datenbeschreibun Österreich Netex-XML”, 2024 “MVO - Datenbereitstellungsplattform”, 2025 The process entails the extraction of each individual XML file, with the values then being stored in a dictionary. The dictionaries are stored in lists. Following the extraction process, all files are systematically compiled into a comprehensive list. This list is then flattened, which means that the multiple lists of dictionaries are consolidated into one. The objective is to create a dataframe that facilitates the reading of both formats and enhances their comparability. Following the extraction process, two data frames have been obtained. These will now be subjected to a comparative process known as the station dataframe. 6.2

3.2.1 Extracted Information

As demonstrated in the subsequent table, the extraction process yielded the following information. As previously stated, this does not imply that all of these values are present within each data set. 6.2

3.3 Comparison of Stations

In order to perform a comparison between the GTFS and NeTEx dataframes, a unique identifier is required. The objective is to identify a station in both data sets with absolute clarity. Nevertheless, it would seem that the key is not always identical in both data formats, with the type of value differing and being non-comparable with that of other countries. “Stops- and Timetable Data”, 2025 “Data.Europa.Eu”, 2025 “Horaires et Arrêts Des Transport Publics (GTFS) - Portail Open Data”, 2025 “MVO - Datenbereitstellungsplattform”, 2025 “ÖBB Open Data Datensätze”, 2025 “Datensätze”, 2025

Table 1: Comparison of GTFS and NeTEx attributes

GTFS	**NeTEx**
1. Station ID (`id_gtfs`)	1. Station ID (`id_netex`)
2. Route type (`route_type_gtfs`)	2. Reference ID (`ref_id_netex`)
3. Station name (`name_gtfs`)	3. StopPlace type (`StopPlaceType_netex`)
4. Latitude (`lat_gtfs`)	4. Station name (`name_netex`)
5. Longitude (`lon_gtfs`)	5. Latitude (`lat_netex`)
6. Stop description (`stop_desc`)	6. Longitude (`lon_netex`)
7. Location type (`location_type`)	7. EVA number (Interne Bahnhofsnummer (IBNR)) (`EVA_Nr_netex`)
8. Parent station (`parent_station_gtfs`)	8. UIC code (`UIC_Code_netex`)
9. Wheelchair boarding possibility (`wheelchair_boarding_gtfs`)	9. Quay IDs (`Quay_ids_netex`)
10. Time zone (`stop_timezone`)	10. Wheelchair access (`WheelchairAccess_netex`)
11. Platform (`platform_code_gtfs`)	11. Assistance facility (`AssistanceFacility_netex`)
12. Vehicle type (`vehicle_type`)	12. Assistance availability (`AssistanceAvailability_netex`)
	13. Access facility (`AccessFacility_netex`)

3.3.1 Key Issues to Identify the Stations in Both Formats

As can be seen from the Austria Data sets, the identifiers are similar to each other. The Austrian NeTEx dataframe contains an IFOPT, which has been intended for integration into the Transmodel since 2006. Nevertheless, since 2009, the development of the NeTEx format has incorporated these types of IDs within its structure. The IFOPT has been developed for the purpose of identifying fixed objects, including stops and points of interest. “History – Transmodel”, 10/09/2025, 11:57:38 As demonstrated in the Austrian NeTEx file, the IFOPT is present as “43-7402“. In contrast, the GTFS dataframe exhibits the following IFOPT structure: “at:48:134:0:2“. The GTFS data format uses a different structure because, rather than referring to a train station, each entry refers to a track at a station. “Key:Ref:IFOPT – OpenStreetMap Wiki”, 2025 However it has been observed that there is a similarity in the pattern of the IDs. Each station in the netx file makes reference to an identical IFOPT as the GTFS file, such as “at:43:7402“. “ÖBB Open Data Datensätze”, 2025 “Datensätze”, 2025 It is possible to disregard the final two digits in order to facilitate a comparison of both IDs whilst maintaining reference to the same station. 6.2

In the Luxembourg data set, an alternative form of identification is employed. In the GTFS data set, the ID is expressed as a sequence of nine integral digits, for example, “500000079“. The NeTEx dataframe comprises a sequence of numbers embedded within a string, for instance “DE::StopPlace:220401001_::“.
It is noticable that the identifiers vary in terms of their data type. “Data.Europa.Eu”, 2025 “Horaires et Arrêts Des Transport Publics (GTFS) - Portail Open Data”, 2025 In order to merge the two data sets, the pattern was identified and the number was extracted from the string by identifying a sequence of numbers with the length of nine. The newly created integer type was saved for the purpose of comparing both data sets. 6.2

The Norway data sets do have the same identifiers for their stations. Both identifiers are regarded as string types and manifest the following pattern: “NSR:StopPlace:1“. However, it should be noted that the structure of the GTFS file differs from that of the Luxembourg’s GTFS file and is similar to the Austria file. Each track line, known as a “Quay“, is stored as an entry for a train station. The actual station ID is specified in the “parent station“ column. “Stops- and Timetable Data”, 2025 Therefore, the GTFS data set needed to be summarised so that multiple “Quay“ IDs would refer to a single entry of a rail station. Nevertheless, this constitutes a new format that does not correspond with the Luxembourg type of ID or the IFOPT from the

Austrian data sets. 6.2

3.3.2 Merging Process

Initially, the pattern of the ID will be subjected to analysis and filtered according to the previously delineated conditions and characteristics. Following the identification of the pattern, measures will be implemented to ensure both IDs are made identifiable. The IFOPT number in the Austria GTFS data set will be reduced by their final two characters, while in the Luxembourg NeTEx data set, the ID within the string will be extracted and in the norway GTFS data set the "Quays" will be changed to stations IDs. Following the execution of the abovementioned processes, an outer merge function will be executed on each dataframe. The utilisation of the outer merge ensures that the dataframes themselves remain unaffected. It is evident that, in the event of a comparison being made between the stations, no entry will be filtered or deleted "Pandas.DataFrame.Merge — Pandas 2.3.2 Documentation", 2025. 6.2

3.4 Wiki Data Base

In order to verify the correctnesss for each stations within the data formats, it is necessary to consult a third source. The Wiki database comprises millions of entries within a knowledge graph, and it is readily accessible due to its free availability. Structured data is stored within the knowledge base, and can be accessed using SPARQL queries. Furthermore, it is possible to edit the SPARQL query request in a highly detailed manner. It is possible to set multiple conditions specific to the task at its core and gather only the data that is required for the specific purpose in question. Bielefeldt et al., 2018

3.4.1 How to Connect the Third Source with NeTEx or GTFS

In order to connect the Wiki database and the GTFS or NeTEx data sets, a key is required. As previously stated, the NeTEx and GTFS data frames have been connected. The next step is to identify another key that can be used to distinguish each station in all three or at least two data sets. According to the data description for the Austrian NeTEx data set, there is one key-value pair called "EVA-Nr". This short discription is matching with the Description for the IBNR according to the Deutschlandtarifsverbund GmbH. The IBNR is used for their electronic processing and fare calculation in the sales systems. Deutschlandtarifverbund GmbH, 2025 "Datenbeschreibun Österreich Netex-XML", 2024 As stated in the internal documents of

the Verkehrsbund Bremen/Niedersachsen, the “EVA Nr“ and the IBNR IDs are typically synchronised within the Hafas System. Raffael Rittmeier, 2016 Hafas is a system developed by Hacon that is utilised by prominent rail operators such as the Deutsche Bahn “Hacon - A Siemens Company”, 2025 and by the ÖBB Scotty App. “Dreifache Auszeichnung Für ÖPNV-Apps von Hacon”, 2025 It plays a crucial role in the management of their timetables and the conversion of data from diverse sources into a unified format. “HAFAS Rohdaten Format (HRDF) – Open Data-Plattform Mobilität Schweiz”, 2025 “HAFAS.Engine_english”, 2025 These only indicates a connection between the “EVA-Nr“ and theIBNR. Nevertheless, the wiki database provides such IBNR for stations “IBNR ID”, 2025. It is therefore hypothesised that a connection can be established between the wiki database and the Austrian NeTEx data set from the “EVA-Nr“ to the IBNR. 6.1

The Norway data NeTEx set contains a so called Union Internationale des Chemins de fer (UIC) country code. A UIC code is a unique identifier used to refer to railway stations. The first two digits denote the country of origin, while the remaining numbers are used to identify a specific station. railways, 2015 “Open Data about Railway Stations”, 2025. It appears also in the Wiki data base “UIC Station Code”, 2025, therefore a connection can be established between the Wiki data and Norway NeTEx data.

Two of the three distinct NeTEx data sets from different states contain identifiers that match those of the Wiki database. However non of the GTFS data sets contians any of those identifiers which might match with the Wiki data base identifiers. “Stops- and Timetable Data”, 2025 “Horaires et Arrêts Des Transport Publics (GTFS) - Portail Open Data”, 2025 “Datensätze”, 2025 Nonetheless, in the context of the SPARQL query, an attempt was made to retrieve the IFOPT because the Wiki data base provides it as an object “Identification of Fixed Objects in Public Transport”, 2025. It would be an attempt to compare the Austrian NeTEx and GTFS data sets with the wiki data base.

Given the nature of the wiki database as a knowledge graph comprising multiple entities, the collection of IFOPT, IBNR, UIC and coordinates is essential. For example the Vienna Main Station consist all of these entities. “Wien Hauptbahnhof”, 2025 An alternative option would be to utilise data from OpenStreetMap. However, it appears that the Open Street Maps knowledge graph contains solely UICcode, with no IBNR. “Key:Uic_ref – OpenStreetMap Wiki”, 2025 The undertaking of a comparison would be complicated for countries that do not utilise UIC codes, such as Austria. As

stated in the Österreichische Bundes Bahnen (ÖBB) data description for their Geo Data set, the netex data set includes an "EVA" number, which is equivalent to an IBNR. "Datenbeschreibun Österreich Netex-XML", 2024 "ÖBB Open Data Datensätze", 2025 6.1

The fundamental concept was to extract all subjects that met the following criteria per country in a single SPARQL request: train stations, tram stops or small train stations that have an UIC code, IFOPT or IBNR, and that have coordinates. Train stations are represented as object "Q55488" "Railway Station", 2025 and any kind of subclass like small train stations by "P31" and "P2790". "Subclass Of", 2025 "Instance Of", 2025 In addition, it was imperative to ensure the inclusion of all requested stations. To this end, the limit of collected stations was set at a value greater than the actual number for each country. To illustrate this, for Austria, the number of stations was set at 2,000, whereas the actual number is 1,031. "Zahlen, Daten, Fakten", 2025 Following SPARK-QL Querry is seen below. 6.1

```
SPARK-QL Querry Request:

 SELECT ?station ?stationLabel ?coordinate ?ifopt ?ibnr ?uic WHERE
 {{
 ?station wdt:P31/wdt:P279* wd:Q55488 ;
 # instance or subclass of train station
 wdt:P17 wd:Q{country_code} .
 # for located in Austria (Q40)

OPTIONAL {{ ?station wdt:P7824 ?ifopt. }}  # IFOPT code
OPTIONAL {{ ?station wdt:P954 ?ibnr. }}    # IBNR
OPTIONAL {{ ?station wdt:P722 ?uic. }}     # UIC
OPTIONAL {{ ?station wdt:P625 ?coordinate. }}  # Coordinates

SERVICE wikibase:label {{ bd:serviceParam wikibase:language
"[AUTO_LANGUAGE],de,en". }}}}
LIMIT {limit}
```

3.5 Distance Calculation

In the event of there being a match between the keys from the Wiki database and the keys from the various data sets, a method is required to verify whether the information regarding the coordinates of the stations is also matching. In the case of there being a difference between the two sets, it is

necessary to calculate the discrepancy between them. Therefore a comparison was made between the GTFS or NeTEx and the Wiki data set. The calculation of the distance between the two coordinates was performed by implementing the so-called Haversine formula. The haversine formula is a mathematical technique used to calculate the distance between two coordinates on a circle. It operates under the assumption that the radius of the Earth is 6,367.45 kilometres. It is evident that the Harversine forumla does not take into account the surface of the Earth. Maria et al., 2020 My approach to the Haversine formula in Python was informed by the interpretation via Java Script. "Calculate Distance and Bearing between Two Latitude/Longitude Points Using Haversine Formula in JavaScript", 2025 6.4

For each unique matching entry of the merged dataframe of Wiki data and the NeTEx and GTFS data frame, the coordinates were applied to the haversine function. The results were saved into a list and then added as a column to the merged data frame. 6.4

3.6 Extraction Trips

In order to extract the information relevant to trips, the following minimum data elements should be considered for extraction: the stops of a trip, in particular the start and end destinations. However, it is already known that NeTEx and GTFS stored their information differently due to the nature of their different formats. Soares and Martins, 2013

3.6.1 GTFS Trips Extraction

The GTFS trips are stored in the "trip" section of the GTFS file, and the stops of the trips are linked within the "stop times" section via "Trip ID" to the trip data frame. "Reference - General Transit Feed Specification", 2025. A straightforward merge process has the ability to establish a connection between the two data frames by way of the "Trip ID" in "stop times". However, it is important to note that the Norway and Luxembourg data set also contains bus trips, which must be filtered before. For instance the norway data set contains 345.019 entries after the extraction process. "Stops- and Timetable Data", 2025 "Horaires et Arrêts Des Transport Publics (GTFS) - Portail Open Data", 2025 As previously stated in the Chapter, entitled 'Extraction of Stations' 3.2, a similar filter was applied; this time an alternative approach was adopted. As seen in figure 3. Therefore, it is possible to apply a filter to the "trips" dataframe by selecting only those "route IDs" for which the train "route type" is specified. The "trips" and the "routes" dataframes

are connected by their “route id“. In order to select the appropriate stops for each journey, the stop times dataframe was filtered using the remaining trips id from the filtering process of the trip dataframe. “Reference - General Transit Feed Specification”, 2025 For instance in the norway data set remains 47.439 entries after the filtering process “Stops- and Timetable Data”, 2025. As previously stated, the accurate identification of a train journey in the route dataframe is not uniform across all states. Therefore, all route IDs were filtered on the basis that the first character was 1 or 4, and that the ID had three characters and the numbers 2, 12, 5, 7, as was the case for the stations stated in Chapter 3.2. “Reference - General Transit Feed Specification”, 2025“Extended GTFS Route Types | Static Transit”, 2025. The target information are the “stop ids“ within the “stop times“ dataframe. The “stop ID“ refere to the “stops“ dataframe. “Reference - General Transit Feed Specification”, 2025 In this case, it is linked to the already extracted train stations. By combining these information a trip can be illustrated by using the “stop times“ and the refered “stops“. “Reference - General Transit Feed Specification”, 2025

Following the preparation of all three data frames, the “Trip ID“ data frame

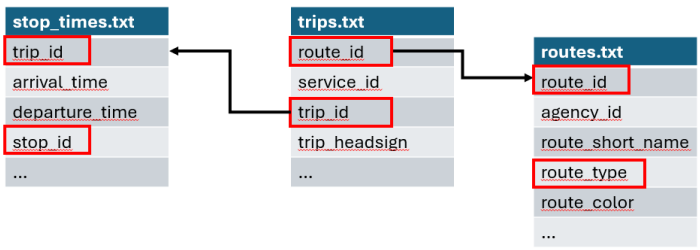


Figure 3: [Illustrated Link between Stops Times, Routes and Trips, Source: Self-created graphic, based on descriptions “Reference - General Transit Feed Specification”, 2025

was iterated. It is vital to ensure that each “Trip ID“ is taken and verified as it appears in the stop times dataframe. If it was found in the stop times data frame, a small data frame that had been filtered by the current looped “Trip ID“ was extracted and added to the matching “Trip ID“ inside a dictionary. The dictionary was then included as a new “Stop on Trip“ column in the “trips “ dataframe. In the event that a matching “Trip ID“ was not found in the “stop times “ dataframe, a “None“ value was added to a dictionary, which

was then applied to the “Stops on Trips“ column. This process was applied to the Austrian and Luxembourg data sets. 6.2

However, it should be noted that the values of the Norway data “stop ID“ set differ slightly. This is due to the fact that, while the Norwegian “stop times“structure remains the same, the “stop ID“ is not linked to a direct “Stop Place ID“, as is the case for the other states. Instead, it refers to a quay, which is a track of a train station. In order to address this issue, it is necessary to extract the stations dataframe and identify the quays and matching stations, as outlined in chapter "Key Issues to identify the Stations in Both Formats" 3.3.1. Each quay stored in the nested dataframe structure will then be searched in the station dataframe. If a match is found, it will be added to the current entry. If no match is found, a 'None' value will be added. “Stops- and Timetable Data”, 2025 “Datensätze”, 2025 “Horaires et Arrêts Des Transport Publics (GTFS) - Portail Open Data”, 2025

3.6.2 NeTex Journey Extraction

According to the Framework paper, the GTFS way of extracting is also present in the NeTEx files. As specified in the GTFS files, the “trip“ and “stop times“ should correspond to the “vehicle journey“ and “Call“ in NeTEx. As previously outlined, the “stop times“ contained within the GTFS files are the key source of information regarding the “Stop ID“, which is the target data. As illustrated in the following figure 5 the “Service Journey“ is linked to the “Scheduled Stop Points“ via “Call“, which contain the relevant information regarding the “Stop Place“. Knowles, 2024

Nevertheless, the documented process of converting the GTFS Trips to the NeTEx Journey did not work for any of the data sets analysed. NeTEx is a data format that can be used at multiple operational levels. It is characterised by its flexibility, allowing users to select the structure that best suits their needs. Nicholas JS Knowles, 2015 Therefore, it cannot be guaranteed that the described connections are present in the data sets. Another approach is needed.

As per the findings of the analysis, the link from “Service Journey“ to the “Stop Place“ was discovered as illustrated in figure 5. According to the relevant handbooks for the Austrian and Norway Netex formats, information regarding “Stop Places“ is to be found in the “Stop Assignment“ or “Scheduled Stop Point“ section. Both sections refer directly to a “Stop Place“ via a “Stop Place Ref“. It is for this reason that they act as our objective data.

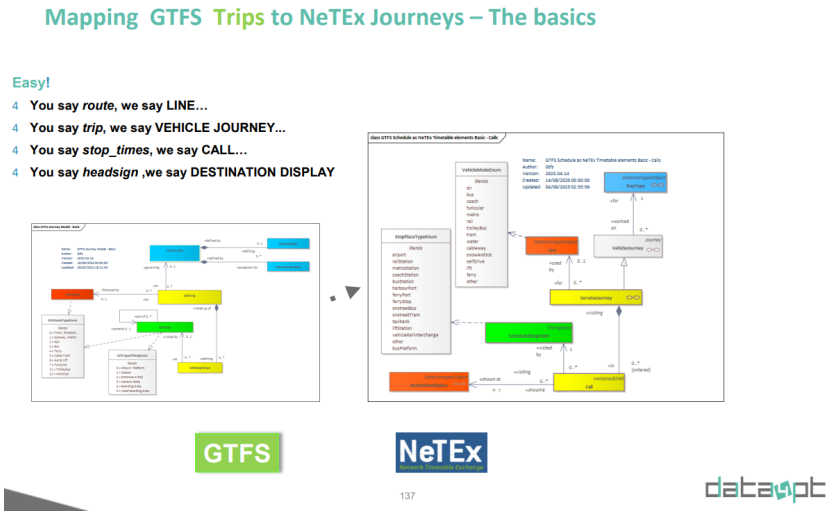


Figure 4: Mapping GTFS Trips to Netex Journey - The basics, Source: Knowles, 2024

However, navigating from the “Service Journey“ to the target information is a complex, multi-tiered process that involves traversing the nested layers of the NeTEx files. A “Service Journey“ comprises a “Timetable Passing Time“ node within its designated section. The “Timetable Passing Time’s“ entries comprise multiple entries, which are the actual made stops of the journey. Each stop in the “Journey Pattern“ refers to a “Service Journey“ or “Journey Pattern“. The “Service Journey Pattern“ provides a framework for each journey, including a node called “Stop Point“ in “Journey Pattern“. This node features multiple entries, similar to those found in the “Journey Pattern“. Each “Stop Point“ in “Journey Pattern“ entry contains a “Stop Point in the Journey Pattern“ ID, which can be connected with the same ID from the “Service Journey“ section. Furthermore, each “Stop Point“ in the “Journey Pattern“ entry contains a “Scheduled Stop Place Ref“. This reference is directly linked to the “Scheduled Stop Point“ section or “Passenger Stop Assignment“. The first one contains a direct “Stop Place Ref“, and the second one also contains a direct “Stop Place Ref“ and a “Quay Ref“. The “Quay Reference“ is also linked to the “Stop Place“ via multiple “Quay References“ for one “Stop Place“. “Timetable - Håndbok N801 (SIRI/NeTEX) - Entur”, 2025 “JourneyPattern (Abstract in EPIP), ServicePattern - NeTEx Profil Österreich - Mobilitätsverbünde”, 2025 “PointinJourneyPattern (Abstract in EPIP), StopPointInJourneyPattern - NeTEx Profil Österreich - Mobilitätsverbünde”, 2025 “ScheduledStopPoint, ServiceLink - NeTEx Profil Österreich - Mobilitätsverbünde”, 2025

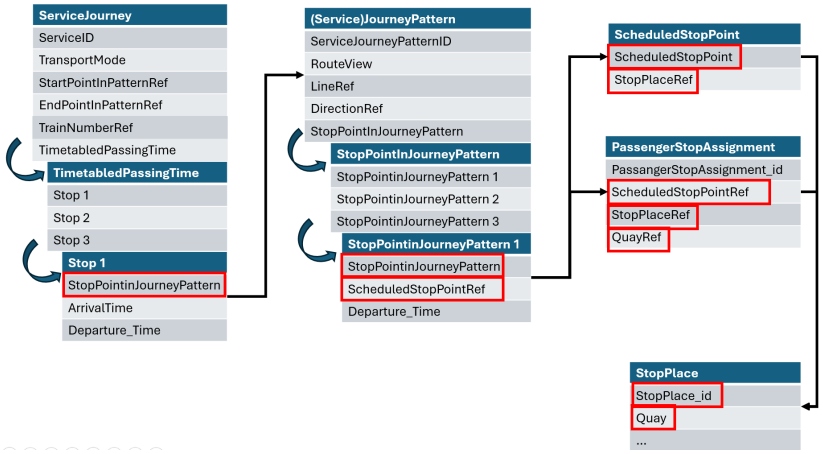


Figure 5: Link from Service Journey to the Stop Places in Netex, Source: Self-created graphic, based on descriptions “Timetable - Håndbok N801 (SIRI/NeTEX) - Entur”, 2025 “JourneyPattern (Abstract in EPIP), ServicePattern - NeTEx Profil Österreich - Mobilitätsverbünde”, 2025 “PointinJourneyPattern (Abstract in EPIP), StopPointInJourneyPattern - NeTEx Profil Österreich - Mobilitätsverbünde”, 2025 “ScheduledStopPoint, ServiceLink - NeTEx Profil Österreich - Mobilitätsverbünde”, 2025

It is important to note that a variety of methods are employed by different nation states for the structuring of their netex files. It is notable that all data sets share the path from “Service Journey“ to “Passenger Stop Assignment“. In the case of Austria and Luxembourg, the Netex files are to be used, with the “Stop Point Ref“ to be referenced directly. Nevertheless, the Norwegian NeTEx approach involves the use of the path from “Quay Ref“ to “Stop Places“, thereby resulting in a process that is marginally more complex. “Stops- and Timetable Data”, 2025 “Data.Europa.Eu”, 2025 “MVO - Datenbereitstellungsplattform”, 2025

Four distinct extraction functions were developed to efficiently extract and collate the necessary information from the four separate sections. Each function accepts the input of a single XML file. Prior to the selection of an extraction method, a selection process will be implemented to determine which XML files are to be extracted by which function. This selection process systematically iterates through all XML files, until all five names of the root are found or not. Should a root be found, such as "ServiceJourney", "Service-

JourneyPattern", "JourneyPattern", "ScheduledStopPoint" or "PassengerStopAssignment", the relevant extraction function will be called to extract the necessary data. As the names of the nodes vary between the Netex files, we require “ServiceJourney“ exclusively for the Norway data set “Timetable - Håndbok N801 (SIRI/NeTEX) - Entur”, 2025. Following the extraction process, four data frames were created and applied to the merging process. 6.2

First, part in merging process is filtering all “Service Journeys“ where their “transport mode“or “journey type netex“ is rail. However, in the case of the luxembourg data set this transport mode is not correctly implementated therefore we need to prove wether the transport model does include None values. As mentioned previsously the Norway and the other two state's structure differs from each other, therefore we define wether the data set is norwagian or not by simply apply a variable into the function which includes the name of the country. Additionaly, each extracted data frame of the “Scheduled Stop Point“ does not contain any values about the “Stop Place Ref“. Therefore, the “Passenger Stop Assignment“ is the only one that can be used. 6.2

With regard to the non-Norwegian data sets from the “Service Journey Pattern“ dataframe, every “Stop Point“ in the “Journey Pattern“ and its corresponding “Scheduled Stop Point Ref“ was extracted from each nested entry of the “Stop Point In Journey Pattern“ column to a dictionary called “stop_point_to_scheduled“. The “Passenger Stop Assignment“ dataframe was then processed using a grouping process that selected only the “Scheduled Stop Points“ and the “Stop Place Reference“into a dictionary called “scheduled_to_stop_place“. As seen from the dictionaries, the connections from “Service Journey Pattern“ to the “Passenger Stop Assignment“ and from the “Passenger Stop Assignment“ to “Stop Place“ are represented in the two create dictionaries “stop_point_to_scheduled“ and “scheduled_to_stop_place“. An iteration was then applied to the “Service Journey“, with each row of the “Timetabled Passing Time“ column being selected. Furthermore, a second iteration was applied to all entries within the row. Each designated “Stop Point in Journey Pattern“ or “Stop“ has been selected and verified to ensure its inclusion in the “stop_point_to_scheduled“, representing the connection from the “Service Journey Pattern“ to the “Passenger Stop Assignment“. If the “Stop Point in Journey Pattern“ or “Stop“ was found in the dictionary, the “Scheduled StopPoint Ref“ was used to add the matching value of the “StopPlaceRef“ from the second dictionary “scheduled_to_stop_place“ to the entry of the current loop. If not match was found an None value was added. This process is illustrated in following figure 6. 6.2

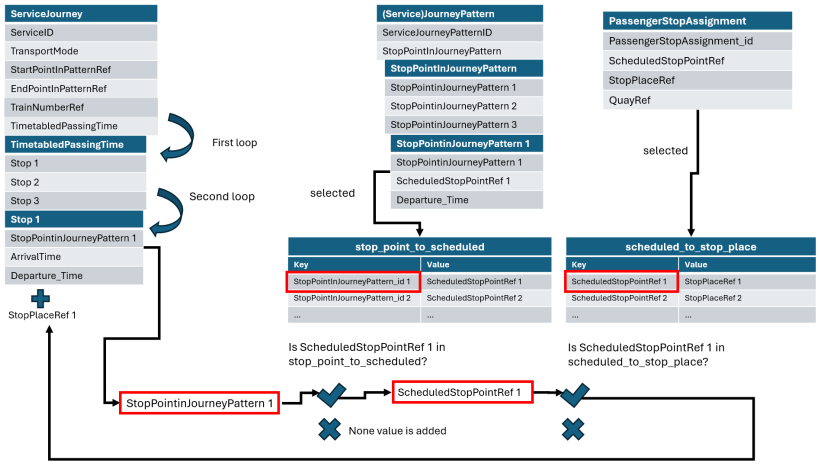


Figure 6: Code Illustration Journey Merging for Austria and Luxembourg, Source: Self-created graphic

In case of the Norway data set an different but in principle similar approach was used as seen in figure 7. As the Norway NeTEx structured uses the “Quay“ path to the “StopPlaces“, the station dataframe from the previous chapter, ’Extraction of Station’ 3.2, is required. As the station dataframe allows for multiple “Quays“, given its presented cardinality of one to multiple, only the “Passenger Stop Assignment“ can be used. “Stops - Håndbok N801 (SIRI/NeTEX) - Entur”, 2025 “Timetable - Håndbok N801 (SIRI/NeTEX) - Entur”, 2025. Therefore, a flattening process is required to change the current structure, whereby each “Quay“ is associated with a station in its own designated row. The new Station dataset and the “PassengerStopAssignment“ are then merged together, as they both share a “Quay Reference“. The new merged dataframe has been named “merged_1“. Furthermore, a dataframe was created by extracting information from the Service Journey Pattern dataframe. This dataset includes all “Stop Point in Journey Pattern“ IDs with their corresponding “Scheduled Stop Point Ref“. Following this, a second merging process is applied to the new dataframe and the already merged data frame “merge_1“, using their shared “Scheduled Stop Point Ref“. This dataframe is labelled “merge_2“. In the final step, the initial iteration is applied to the “Service Journey“, which selects each value in the “TimetabledPassingTime_netex“ column. The second iteration is then applied to each nested value of the first iterated entry. In the event of the “Stop Point in Journey Pattern“ ID being found in the “merge_2“ dataframe, the corresponding “Stop Place Id“ is to be added to the current looped entry.

In the event of no match being found, a None Value was added. 6.2

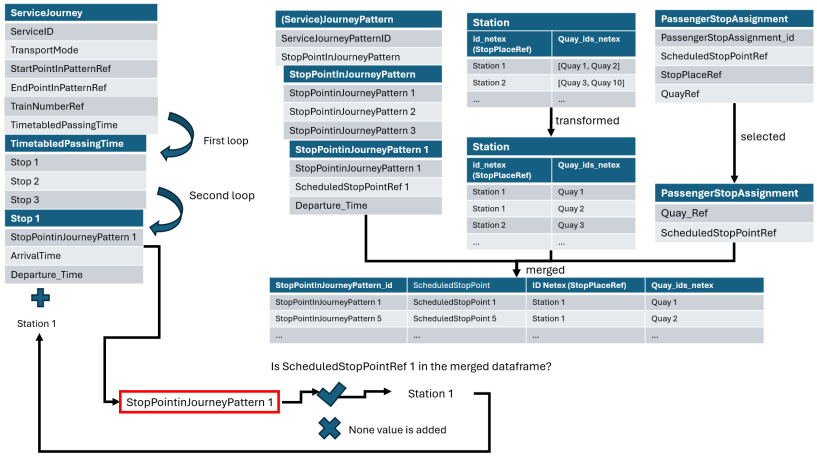


Figure 7: Code Illustration Journey Merging for Norway (simplified), Source: Self-created graphic

As can already be seen, the different NeTEx data formats within the European Union differ from each other “Stops- and Timetable Data”, 2025 “Data.Europa.Eu”, 2025 “Horaires et Arrêts Des Transport Publics (GTFS) - Portail Open Data”, 2025 “MVO - Datenbereitstellungsplattform”, 2025 “ÖBB Open Data Datensätze”, 2025 “Datensätze”, 2025. Therefore, different methods must be used to ensure that the data can be used for the thesis analysis. However, in addition to the extraction guarantee, it is important to note that processing time is another key factor. The different methods also allow results to be obtained despite limited computational power. 6.2

4 Analysis

In the preprocess for each of the three chosen states Luxembourg, Austria and Norway, the following data has been collated: a Wiki data set of all train stations in the state, a station data set from the GTFS and NeTEx, a trip GTFS data set and a NeTEx journey data set. The following questions must therefore be posed: The objective of this study is to determine the number of stations that are matching within the NeTEx and GTFS data set, and to analyse the extent to which these stations are shared with the wiki data set. With regard to the trips and journeys, it is necessary to establish the number of trips that refer to a station within each own data set. Without a reference, the trip would have contained missing stations. While there is a possibility that an ID exists, it could not be located in the station data

frame. Additionally, it is assumed that each state has a single provider for the two data formats, and therefore the sources of information are supposed to be the same. In the Austrian case, the Österreichische Bundesbahnen (ÖBB) are responsible for the stations, netex and GTFS sets, and the netex timetables from the Mobilitätsverbund Österreich data set, which includes the ÖBB-provided timetables. "ÖBB Open Data Datensätze", 2025 "Datensätze", 2025 "MVO - Datenbereitstellungsplattform", 2025. The data sets from Luxembourg are both provided by the Administration of Transport Publics "Horaires et Arrêts Des Transport Publics (GTFS) - Portail Open Data", 2025 "Data.Europa.Eu", 2025. Entur, the national register operator for all public transport, is responsible for the provision of the Norway data sets. "Stops- and Timetable Data", 2025. Therefore, this analysis also considers whether this assumption is true.

4.1 Station Comparison - How many stations are shared?

As outlined in the chapter entitled 'Key Issues to identify the stations in both formats' 3.3.1 with regard to the stations, the NeTEx and GTFS station data sets have already been merged using the provided ID in the data set. As we employed an outer merge process, it was possible to visualise the shares of stations that are shared in the two formats and those that are not. 8 9 9 6.2. The venn diagrams show how many stations are shared between the two formats. Additionally, the number of stations per state is given as a reference value for the supposed minimum number of shared stations. 6.3

The official number for the Austrian station is 1031 "Zahlen, Daten, Fakten", 2025. However, the GTFS data set and the NeTEx data set both contain more than 1031. NeTEx contains 1056 stations and GTFS even more, with 1103 stations in total. It should be noted that both data sets feature a significant number of stations in common with 1001. In comparison to the Norway data set. The number of stations in both data formats is not exact. The expected number is approximately 400 "NORWAY Train Travel Information | Railcc", 2025. The NeTEx data set comprises 509 elements, whereas the GTFS set includes 901, thus showing a significant difference. Both data sets shares 456 stations. For the Luxembourg data sets, the opposite is true. Luxembourg is said to have 70 train stations "JUIL 2025_Carte Reseau CFL_EN_A3_PRINT", 2025. However, the NeTEx data set contains a significantly larger number of stations within the country, with 187 stations listed and the GTFS data set does show 79 stations. It is noteable that there are no shared stations between GTFS and NeTEx formats from Luxembourg. It should be noted that the data sets contain a greater number of stations than there are in the state, as they also include stations from

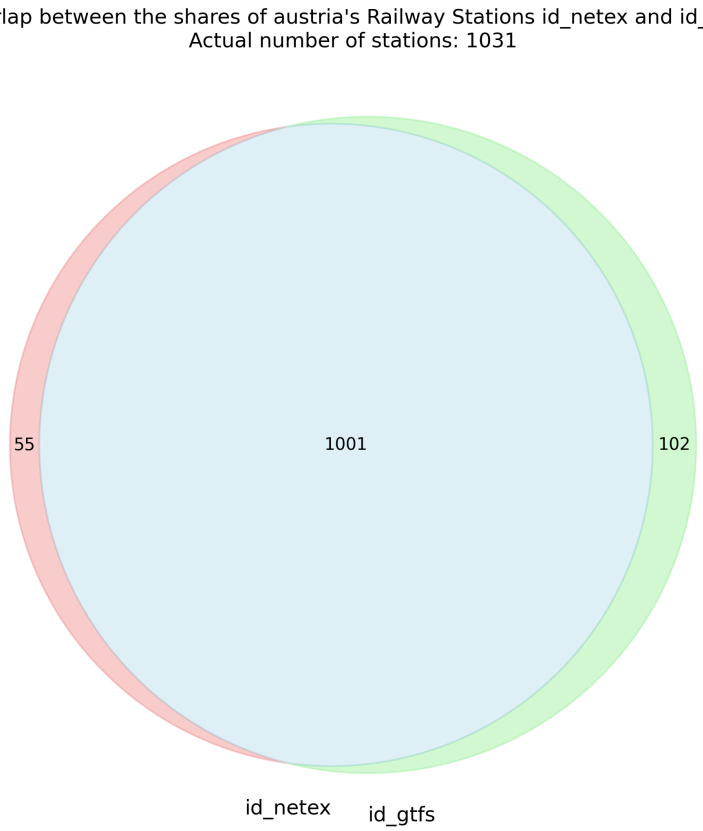


Figure 8: Overlap between the shares of Austria Railway Stations GTFS and Netex, Source: "ÖBB Open Data Datensätze", 2025 "Datensätze", 2025

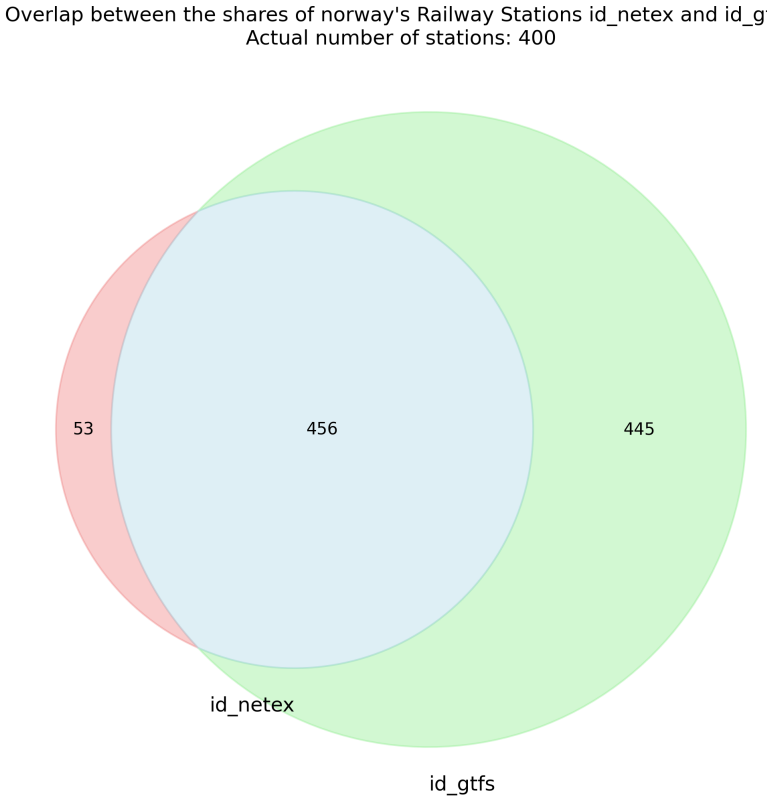


Figure 9: Overlap between the shares of Norway Railway Stations GTFS and Netex, Source: “Stops- and Timetable Data”, 2025

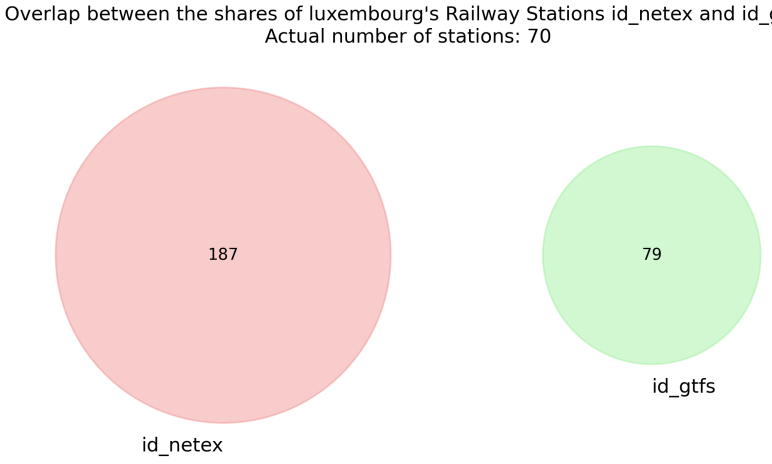


Figure 10: Overlap between the shares of Luxembourg Railway Stations GTFS and Netex, Source: “Horaires et Arrêts Des Transport Publics (GTFS) - Portail Open Data”, 2025 “Data.Europa.Eu”, 2025

neighbouring states. For instance, the Austrian data set includes stations from Hungary. “Stops- and Timetable Data”, 2025 “Data.Europa.Eu”, 2025 “Horaires et Arrêts Des Transport Publics (GTFS) - Portail Open Data”, 2025 “MVO - Datenbereitstellungsplattform”, 2025 “ÖBB Open Data Datensätze”, 2025 “Datensätze”, 2025

Following a thorough evaluation, it was determined that there was no clear explanation for the absence of stations shared between the GTFS and NeTEx files, at least in Norway and Austria. As illustrated in the accompanying maps the not shared station was mapped in each states 11 12. In Norway and Austria, there is a notable absence of significant overlap between the blue NeTEx and red GTFS not shared stations, indicating potential missing stations and whole regions in both the GTFS and NeTEx data sets. However, in Luxembourg none of the NeTEx stations are within Luxembourg's borders on the map 13. This suggests that the station's coordinates are completely inaccurate.

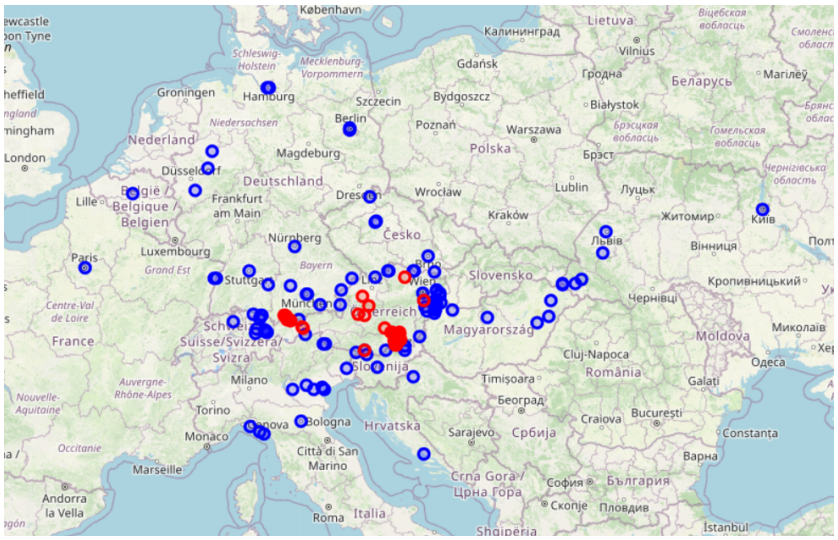


Figure 11: Austria's not shared stations between GTFS (blue) and NeTEx (red), Source: “ÖBB Open Data Datensätze”, 2025 “Datensätze”, 2025

4.1.1 Station Verfication via third Source

As was stated in the preceding chapter, entitled 'How to connect the third source with NeTEx or GTFS 3.4.1, a key was selected. The selection of the

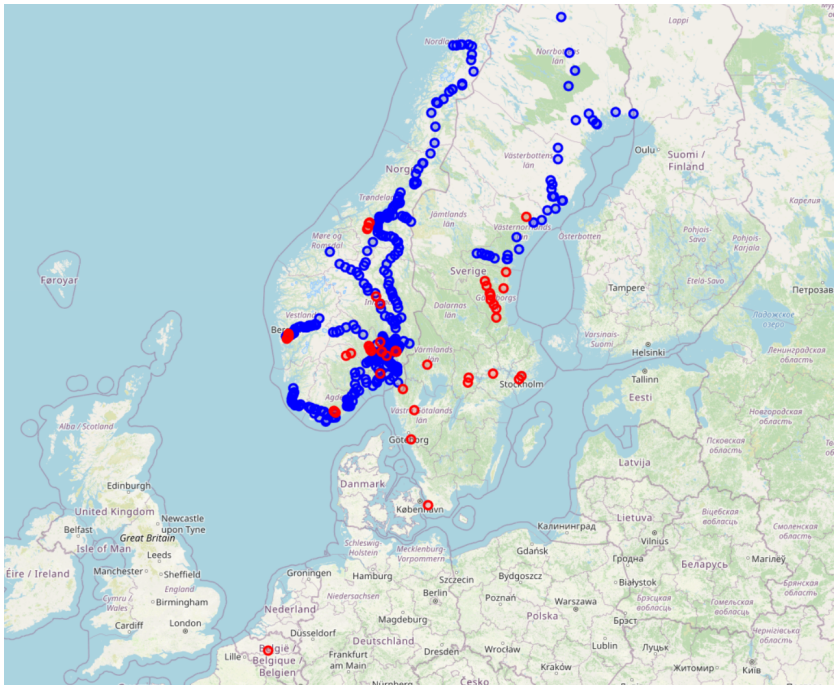


Figure 12: Norway not shared stations between GTFS (blue) and NeTEx (red), Source: “Stops- and Timetable Data”, 2025

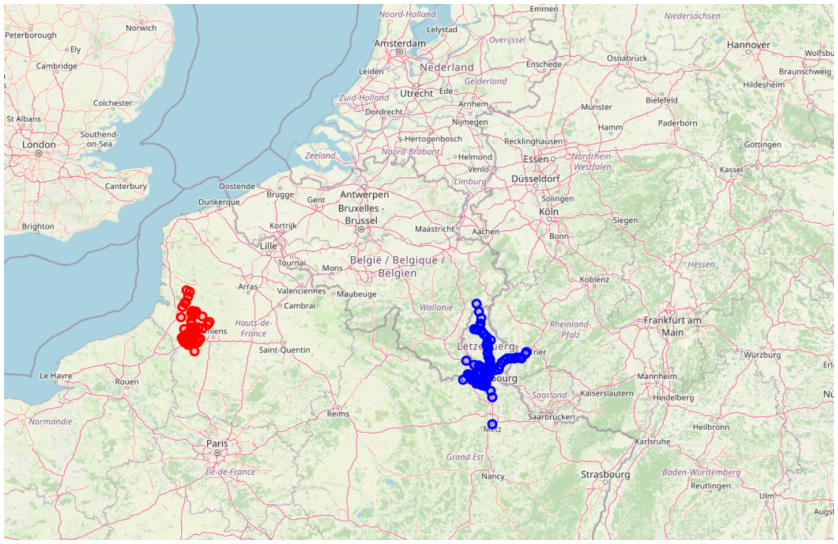


Figure 13: Luxembourg’s not shared stations between GTFS (blue) and NeTEx (red), Source: “Horaires et Arrêts Des Transport Publics (GTFS) - Portail Open Data”, 2025 “Data.Europa.Eu”, 2025

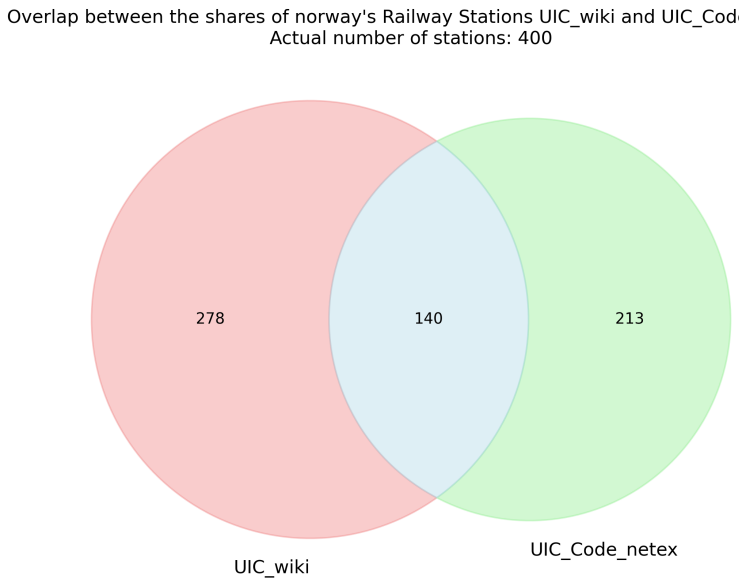


Figure 14: Overlap between the shares of Norway's Railway Stations Wiki and NeTEx, Source: "Stops- and Timetable Data", 2025

IBNR was made in the Austrian data sets due to the fact that, in the majority of cases, each station refers to such an IBNR in the wiki collected data. With regard to the Norway and Luxembourg data set, the UIC code was selected, with the majority of stations also adopting this code. 6.1 However, it is noteworthy that none of the GTFS data sets under analysis contain any of the specified keys. The NeTEx data sets from Norway and Austria are the only ones to include the IBNR or UIC codes. It is therefore only possible to make a direct comparison with the NeTEx and Wiki data set of Norway and Austria. The Luxembourg data sets do not contain any of the keys wether in the NeTEx or GTFS formats. The merging function was applied to the wiki data set and the NeTEx data set by the shared key IDs. The resulting shares are illustrated in the following figures 15 14.

The wiki data will be used as the third source to verify the correctness of the stations. The majority of the Austrian stations have been verified using the wiki data set. A total of 1,008 stations are shared between the NeTEx and Wiki data sets. It should be noted that 197 stations of the Wiki data set, and 48 stations of the NeTEx data set, have not been shared. For the Norwagian data sets the opposite is the case. The minority of 140 could be verified by the Wiki data set. 278 stations from the Wiki data set and 213 stations from the Norway data set are not shared. Not all train stations in

the Norwegian NeTEx data set appear to contain a UIC code. It is evident that there is a discrepancy in the number of stations identified in the two comparisons as illustrated in the figures. 9 14. The number of stations in the Wiki comparison is listed below the station number in Norway, which is from 353 to 400. In comparison to the initial NeTEx and GTFS analysis, a discrepancy of 146 stations is noteworthy. The higher number of stations within the Wiki data set can be explained by the fact that the used SPARQL query does not filter historical stations, which may still be in the Wiki database. 6.1

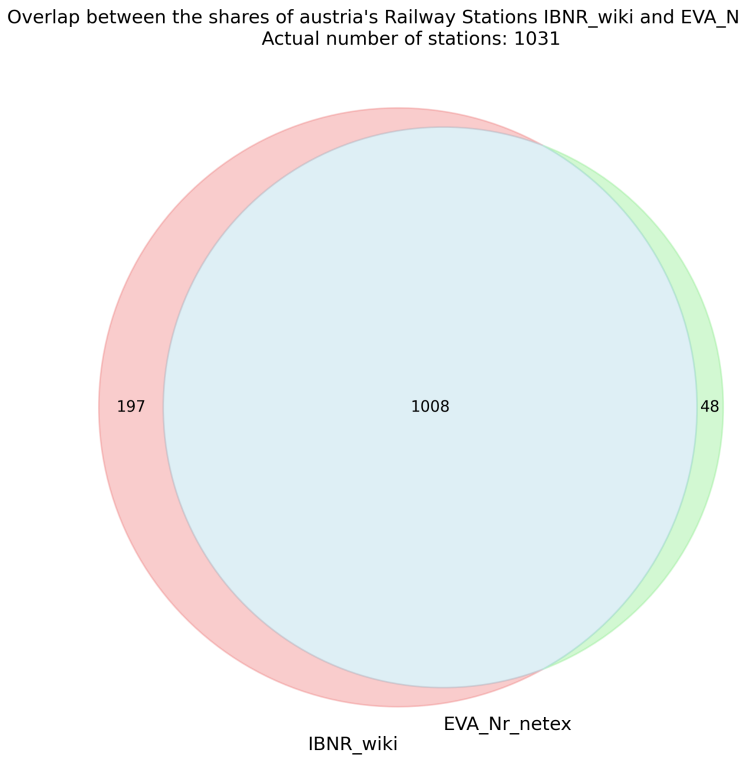


Figure 15: Overlap between the shares of Austria's Railway Stations Wiki and Netex, Source: "ÖBB Open Data Datensätze", 2025 "Datensätze", 2025

4.2 Data Quality - Distance between Stations

As outlined in the previous chapter, two of the three stations' dataframes were successfully verified. In order to verify the data quality, a comparison is made between the Wsiki data coordinates and the coordinates of the GTFS

and NeTEx data set. The distance was calculated using the Wiki data coordinates as a reference point. The distance from the Wiki coordinates to the NeTEx or GTFS coordinates was then calculated. The results are seen in the following figures 16 17. Each bar chart represents the distance of one station, with the height of the bar representing the distance between the NeTEx or GTFS format and the Wiki Data coordinates. The bar charts for both formats are marked in different colours: orange for NeTEx and blue for GTFS. The greater the height, the greater the discrepancy between the format coordinates and the Wiki coordinates.

For Norway, the mean distance between the Wiki coordinates and the GTFS coordinates is 87.95 metres, and for NeTEx, it is 90.95 metres. In comparison to Austria, the average distance from the Wiki coordinates is slightly lower, at 43.15 and 43.12, respectively. As the figures clearly demonstrate, the majority of stations in each state are far below 200 metres away from the coordinates provided by Wiki. In both cases, there are outliers with a distance greater than 900 metres. Despite the NeTEx and GTFS data sets being intended to be equivalent, there is a discrepancy in the coordinates when compared to the Wiki coordinates. However, it is notable that the Wiki coordinates are not as accurate as the other data sets in some cases. Each format, including the wiki data, has its own issues with coordinates. As illustrated in the following figures, one station was selected for inspection, with a distance greater than 400 metres from the wiki coordinates 18. In

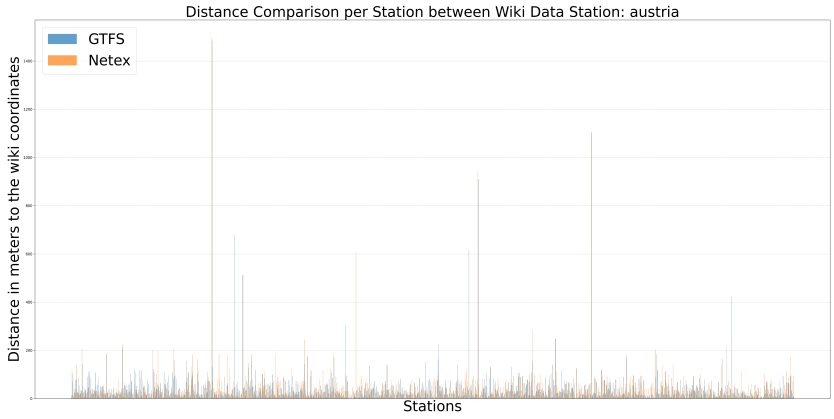


Figure 16: Distance Comparison per Austrian's Station between Wiki data, netex and GTFS, Source: "ÖBB Open Data Datensätze", 2025 "Datensätze", 2025

this instance, the Wiki and GTFS data sets are identical in terms of coordinates, whereas the NeTEx data set differs from both of them. Therefore, it

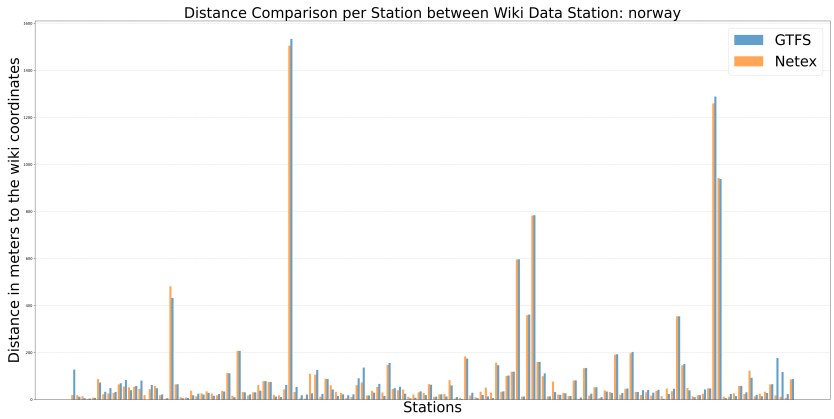


Figure 17: Distance Comparison per Norway's Station between Wiki data, netex and GTFS, Source: "Stops- and Timetable Data", 2025

Figure 18: Example Station Weitlanbrunn - different coordinates between Netex (red), GTFS (blue) and Wiki (green), Source: "ÖBB Open Data Datensätze", 2025 "Datensätze", 2025

is not clear which of these two formats performs better in terms of distances, since the third Wiki source also does not contain an accurate value. This raises the question of how much the distance differs between the two formats. In addition, the following figures illustrate the significant differences between the GTFS and NeTEx coordinates 19 20.

It is evident that there is a discrepancy in the distances between the GTFS and NeTEx in both sets. The majority of the distances are still below 100 metres. However, it should be noted that despite the fact that these data are provided by the same provider in Norway and Austria, "Stops- and Timetable Data", 2025 "ÖBB Open Data Datensätze", 2025 "Datensätze", 2025, there is a discrepancy indicating that the NeTEx and GTFS data sources are not based on the same underlying database. Otherwise, the distance between the NeTEx and GTFS stations should be in close approximation to zero. This would suggest that the coordinates used in NeTEx and GTFS are from the same source. However, this is not the case.

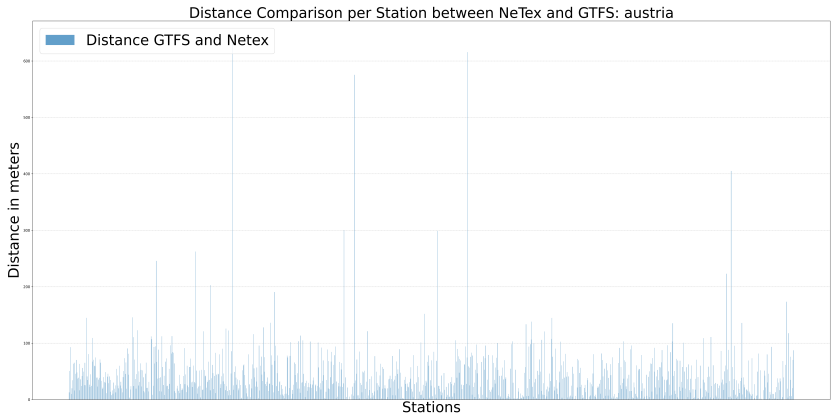


Figure 19: Distance Comparison per Austrian's Station between GTFS and NeTex, Source: "ÖBB Open Data Datensätze", 2025 "Datensätze", 2025

4.3 Trips and Journeys - Identifable stations

In order to measure how many stops for a trip are actually referring to a real stop ID, the following steps were made. Firstly, the station dataframe and the GTFS trip and NeTEx journey dataframes were loaded. Each trip and journey is contained within a list of stops, with the GTFS referring to these as Stops and the NeTEx referring to them as Stop Places. The stops IDs were selected and then compared with the stations collected in the stations dataframe. A comprehensive count and illustration of all stops per trip and

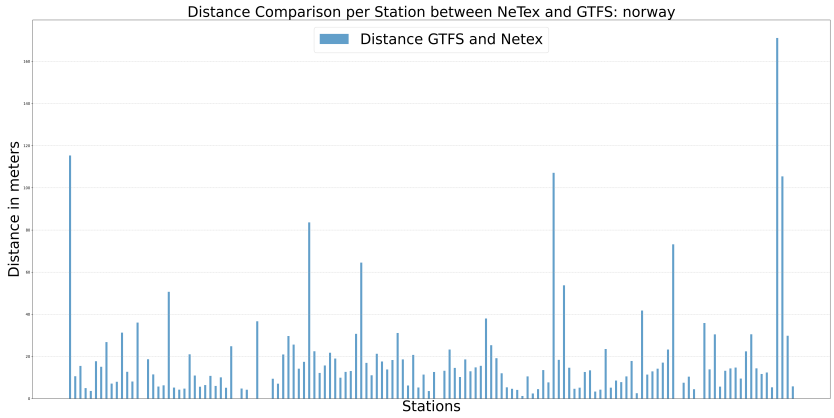


Figure 20: Distance Comparison per Austrian's Station between NeTex and GTFS, Source: "Stops- and Timetable Data", 2025

non-referable stops was performed for each individual state. Each bar chart represents a single trip or journey. The height of the bar chart indicates the number of stops for each trip and journey. The green bar chart illustrates the number of stops, while the blue bar chart shows the number of stops that have been successfully verified by the station's data frame. If a stop has not been refered, the trip or journey will contain a stop that cannot be located therefore it is a station without a name or any other information about this stop. It is important to note that the greater the number of stops that are successfully referenced by the station's data frame, the less incomplett the resulting trip or journey will be. 6.3

The Austrian data sets are illustrated in the following figures which shows that the the majority of the stops in both formats are refering to station from the station dataframe21 22. It has been noted that the successfully referred stops vary between NeTEx and GTFS for each trip. It was discovered that the Austrian GTFS data sets did not contain accurate references for 52 trips from 9325. From these 52 trips, an average of 3.4 stops were missing per trip, with an average of 4.7 stops per trip. In comparison with the NeTEx data set, 18,602 journeys were not accurately referred from a total of 36,835. In the 18,602 journeys analysed, an average of 4.35 stops were refered, with an average of 6.9 stops per journey. "MVO - Datenbereitstellungsplattform", 2025 "ÖBB Open Data Datensätze", 2025 "Datensätze", 2025

In the case of Luxembourg, there is a significant difference in the number of successfully referred stops across both formats. All stops on 2,292 trips in the GTFS could be verified using the station dataframe. In compari-

son, from all NeTEx stops, not a single complete journey from in total of 1731 could be verified by the station dataframe. As demonstrated in the figures, the substantial discrepancy is evident 23 24. On average, it was possible to refer successfully from only 1 out of a total of 9 stops per journey. “Data.Europa.Eu”, 2025 “Horaires et Arrêts Des Transport Publics (GTFS) - Portail Open Data”, 2025 In the case of Norway, all trips and journeys in

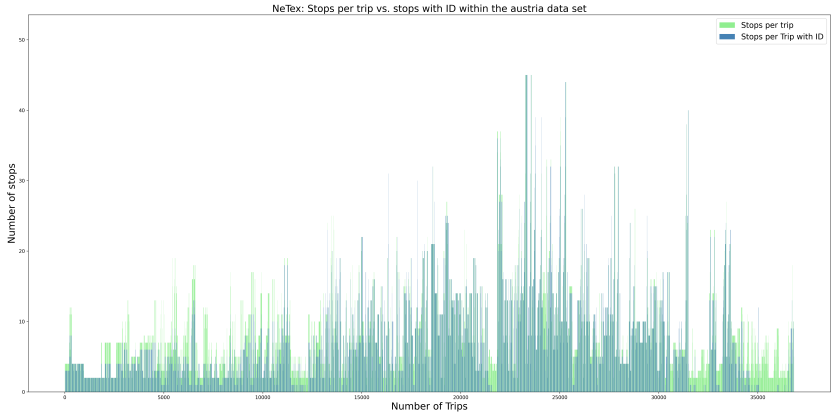


Figure 21: Netex: Stops with refering ID and Stops without in Austria, Source: “Datensätze”, 2025

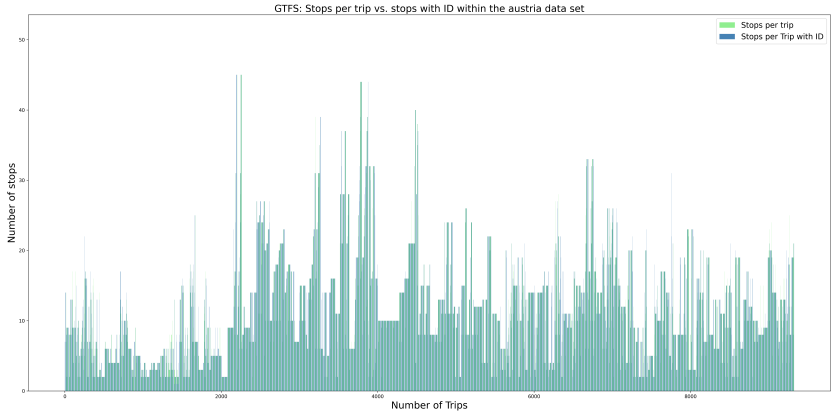


Figure 22: Netex: Stops with refering ID and Stops without in Austria, Source: “MVO - Datenbereitstellungsplattform”, 2025

both formats contain a successfully verified station IDs. Each stop is referred to as a legitimate station in the station data frame. A total of 47,439 trips have been recorded for GTFS and 26,852 for NeTEx. As illustrated in the figures26 25, it is the only state where no stops are missing in both formats.

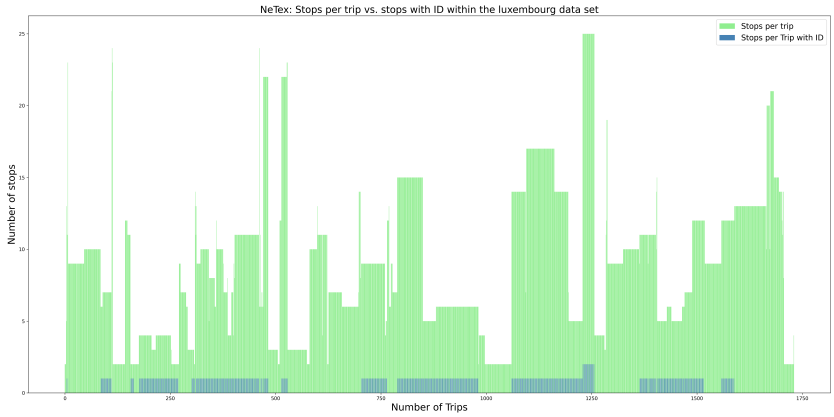


Figure 23: Netex: Stops with refering ID and Stops without in Luxembourg, Source: “Data.Europa.Eu”, 2025

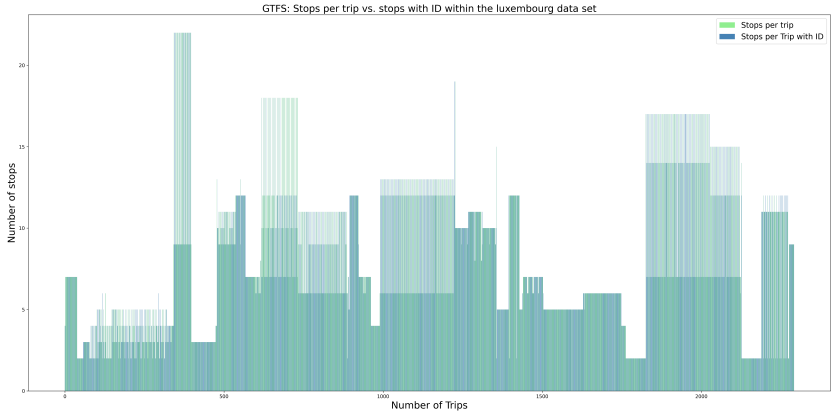


Figure 24: GTFS: Stops with refering ID and Stops without in Luxembourg, Source: “Horaires et Arrêts Des Transport Publics (GTFS) - Portail Open Data”, 2025

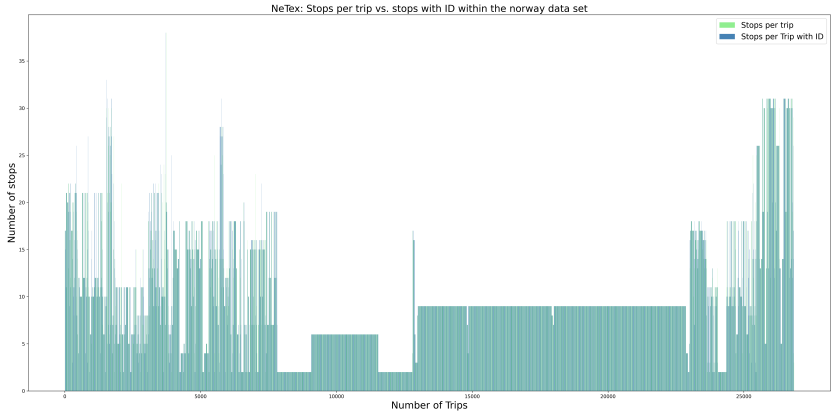


Figure 25: Netex: Stops with refering ID and Stops without in Norway, Source: “Stops- and Timetable Data”, 2025

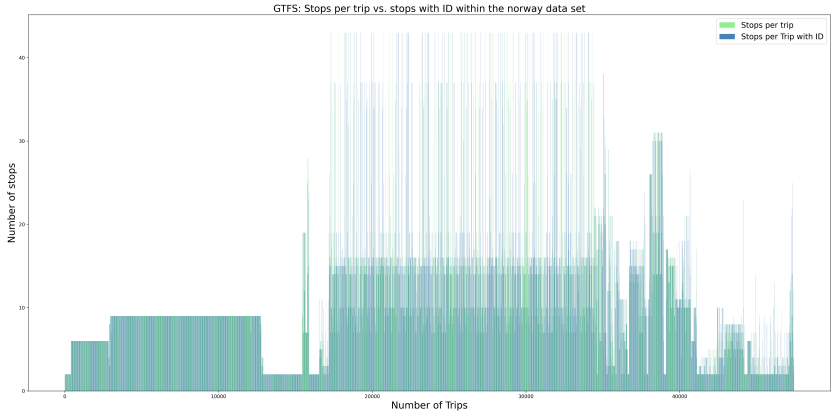


Figure 26: GTFS: Stops with refering ID and Stops without in Norway, Source: “Stops- and Timetable Data”, 2025

“Stops- and Timetable Data”, 2025

In terms of stop verification, a lack of reference is evident, especially in the NeTEx format. It appears in the GTFS format for Austria. However, this is less significant than the absence of references in NeTEx. As previosuly in the beginning of this chapter outlined the lack of stop reference demonstrates the unacurracy and imcompletnesses of the data. As outlined at the beginning of this chapter, the lack of a stop reference demonstrates the inaccuracy and incompleteness of the data. This is particularly significant for Austria and Luxembourg, where the most references are missing. “Stops- and Timetable Data”, 2025 “Data.Europa.Eu”, 2025 “Horaires et Arrêts Des Transport Publics (GTFS) - Portail Open Data”, 2025 “MVO - Datenbereitstellungsplattform”, 2025 “ÖBB Open Data Datensätze”, 2025 “Datensätze”, 2025

4.4 Results of the Analysis

Notably, it is complex to compare the two data formats, NeTEx and GTFS, with each other. The assumption was that both data sets should come from the same provider in each of the three states. It was also assumed that they should use the same data and information sources.“Datensätze”, 2025 “MVO - Datenbereitstellungsplattform”, 2025 “Horaires et Arrêts Des Transport Publics (GTFS) - Portail Open Data”, 2025 “Data.Europa.Eu”, 2025 “Stops- and Timetable Data”, 2025. The results of this analysis suggest that this is not the case.

The lack of shared stations in both data formats suggests that they do not originate from the same source. For example, examining the stations that are not shared reveals no clear pattern. They appear to be chosen at random for Norway and Austria. Verifying the stations using a third source is difficult since the GTFS format does not include any additional information, such as IBNR or UIC codes “Reference - General Transit Feed Specification”, 2025. Therefore, only a comparison with stations shared in NeTEx and GTFS was possible. As the verification of the stations by Wiki Data fluctuates significantly between Norway and Austria, it is possible that the information is incorrect in all three formats. For example, there are significant differences in the coordinates between the third source and GTFS and NeTEx, as well as between the two formats. 4.1 This indicates that, even if the provider is the same, the two formats do not share the same source of information. If they shared the same source, the analysis would measure a fluctuation much closer to zero, since the coordinates are supposed to be the same for each

station. 4.2

In terms of incomplete stations, the NeTEx and GTFS formats differ in the parts they share, as not all stations are shared between the two formats. The number of stations shared in GTFS and NeTEx for Austria is slightly lower than the actual number of stations in Austria "MVO - Datenbereitstellungsplattform", 2025 "ÖBB Open Data Datensätze", 2025 "Datensätze", 2025 "Zahlen, Daten, Fakten", 2025. Norwegian shared stations cover a greater number of stations in the country; however, there is a greater difference in the number of stations in each format "Stops- and Timetable Data", 2025 "NORWAY Train Travel Information | Railcc", 2025. In Luxembourg, no stations were shared because none of the stations in the Netex dataset are in Luxembourg, as can be seen in the figure 13. 4.1

In the case of incomplete trip references, the GTFS format achieved significantly better results than the NeTEx format in all three states. In all three states, the GTFS dataset could successfully reference the majority of its stops, and in two of the three states, it could reference all of them. By contrast, Netex data was significantly more incomplete. Only one of the three analysed states could reference all stops on a journey. The other two states performed particularly poorly in this analysis. Only a small minority of journeys in Luxembourg and around half of those in Austria could successfully reference all stops. 4.3 "Stops- and Timetable Data", 2025 "Data.Europa.Eu", 2025 "Horaires et Arrêts Des Transport Publics (GTFS) - Portail Open Data", 2025 "MVO - Datenbereitstellungsplattform", 2025 "ÖBB Open Data Datensätze", 2025 "Datensätze", 2025

4.4.1 Possible Explanations

According to the Napcore report, the NeTEx standard is the most used exchange standard for static for Multi Model Travel Information (MMTIS) realted information within the european union. It is used for the exchange of data, supporting location search, supporting detailed common standard and special fare queries and providing insight into existing trip plans and auxiliaryaspects and supporting trip plan computation. However, the standard is not yet fully implemented on a wide scale. "Activity WG3 NAP Content and Accessibility | NAPCORE", 2024 One potential reason for this could be the high level of effort required to implement such a NeTEx standard. The NeTEx structure is more complex than the GTFS structure Knowles, 2024. Furthermore, implementing NeTEx as a standard requires the integration of a comprehensive ecosystem. It is part of the Transmodel system, which incor-

porates a range of models including SIRI and NeTEx. Christophe Duquesne, 2023. Additionaly, the Transmodel ecosystem including NeTEx represents more than a standard, it is a comprehensive, harmonised concept for an open data space concerning European rail travel. One of the most significant challenges that train operators must overcome is the financial burden associated with the implementation and training required to manage and maintain such systems. It is vital that the benefits of such implementation outweigh the cost. "Cen-Tc-278_n5072_europeanfarerailprofilenetex-Callforexperts", 2024 These could be a factor in the decision of many European states to not implement a full NeTEx solution.

In the case of GTFS, there is an possible additional reason why it performs better in the analysis: this format is already used by passengers applications. GTFS was developed by Google for the specific purpose of utilising Google Maps. Goldstein and Dyson, 2013 In the world of map and navigation applications, Google Maps is the clear market leader in terms of popularity. It is reported that 67 per cent of smartphone users prefer this navigation app. wtw, 2025 It is therefore logical that a significant number of train operators publish their data in GTFS format. This format is straightforward "Activity WG3 NAP Content and Accessibility | NAPCORE", 2024 and simpler to implement "Create - General Transit Feed Specification", 2025, and many people have an application that can access the information "Publish - General Transit Feed Specification", 2025 such as Google Maps.

5 Summary

To summarise, the thesis analysed GTFS and NeTEx in terms of their stations and trips. A comparison was made between the stations in both formats, as well as with stations from each member state, using the knowledge graph from the Wiki database. Furthermore, the distance between the shared stations was calculated using the coordinates of the sources. In both formats, the trips were extracted and analysed to determine the number of stops per trip that are referenceable with the stations within the same dataframe. It was argued that both data formats should be provided by the same national provider and therefore would theoretically share the same source of information. However, this is not the case.

Following a thorough analysis and preparation, it was determined that both formats present certain issues. With regard to the international standardisation of data formats, there are a number of issues to consider. Problems

arise from the differing definitions of route types used in the various GTFS data sets “Extended GTFS Route Types | Static Transit”, 2025 “Reference - General Transit Feed Specification”, 2025. Furthermore, the standard GTFS structure for stops is not shared, since each entry in the stop section of the GTFS files appears in the Austrian and Norwegian data sets as a track line, rather than a station, as in the Luxembourg data set. “Stops- and Timetable Data”, 2025 -–Datensaetze2025 “Horaires et Arrêts Des Transport Publics (GTFS) - Portail Open Data”, 2025

According to the NeTEx convention, the identifier for train stations supposed to be the IFOPT or at least refering to it within the format Soares and Martins, 2013. However, it should be noted that this applies exclusively to the Austrian data sets. Norway and Luxembourg utilise their own national IDs system , which can result in a more complex analysis of each process. Furthermore, the significant absence of data in the NeTEx data sets is a notable issue. For instance, the Luxembourg data set does not include any stations in Luxembourg . Furthermore, approximately only half of the Austrian journeys, including referable stops. Should a referable stop not be included, the journey will contain missing information regarding the referencing station of the stop. “MVO - Datenbereitstellungsplattform”, 2025 “ÖBB Open Data Datensätze”, 2025 “Datensätze”, 2025 “Data.Europa.Eu”, 2025 “Stops- and Timetable Data”, 2025 4.3

The concept of a unified European rail network incorporating an open data space within the European Union is an ambitious objective. Following a detailed review of the available data formats, it has been determined that in order to achieve this objective, it will be necessary to implement an enhanced version of those data standards that are not currently available through GTFS or NeTEx. To summarise, a key issue common to both data formats in the three states is the requirement for a unique identifier for all stations across Europe. For instance, the IFOPT and the UIC code are designed to provide a solution to the problem. However, the current state of implementation is suboptimal, as the analysis shows. Despite the majority of train operators in Norway, Austria and Luxembourg being UIC partners, “UIC Vademecum”, 2025 further improvements can be made, such as implementing the UIC code for each station. It is evident that the current state of data format maintenance is not sufficiently developed, particularly in the case of NeTEx standard, due to a lack of information.

References

Action Plan to boost passenger rail - European Commission. (2025, April 27). Retrieved April 27, 2025, from https://transport.ec.europa.eu/news-events/news/action-plan-boost-passenger-rail-2021-12-14_en

Activity WG3 NAP content and accessibility | NAPCORE. (2024, January 30). Retrieved September 8, 2025, from https://www.napcore.eu/documents/M3.5_4th_report_NAP_data_availability.pdf

Antrim, A., & Barbeau, S. J. (2017). Opening the Door to Multimodal Applications: Creation, Maintenance and Application of GTFS Data. (17–03702). Retrieved September 8, 2025, from https://trid.trb.org/View/1438473

Bielefeldt, A., Gonsior, J., & Krötzsch, M. (2018). Practical Linked Data Access via SPARQL: The Case of Wikidata.

Calculate distance and bearing between two Latitude/Longitude points using haversine formula in JavaScript. (2025, September 12). Retrieved September 12, 2025, from https://www.movable-type.co.uk/scripts/latlong.html

Cen-tc-278_n5072_europeanfarerailprofilenetex-callforexperts. (2024, December 3). Retrieved September 22, 2025, from https://www.cencenelec.eu/media/CEN-CENELEC/News/Brief%20News/2025/cen-tc-278_n5072_europeanfarerailprofilenetex-callforexperts.pdf

Christophe Duquesne. (2023, January 18). *EN_NeTEx-introduction_v.1-1*. Retrieved September 22, 2025, from https://transmodel-cen.eu/wp-content/uploads/2024/05/EN_NeTEx-introduction_v.1-1.pdf

Create - General Transit Feed Specification. (2025, September 22). Retrieved September 22, 2025, from https://gtfs.org/getting-started/create/

Data Models. (2025, September 8). Data4PT. Retrieved September 8, 2025, from https://data4pt-project.eu/data-models/

Data.europa.eu. (2025, September 8). Retrieved September 8, 2025, from https://data.europa.eu/data/datasets/horaires-et-arrets-des-transport-publics-netex?locale=en

Datenbeschreibun Österreich Netex-XML. (2024).

Datensätze. (2025). ÖBB Open Data. Retrieved September 17, 2025, from https://data.oebb.at/de/datensaetze

Deutschlandtarifverbund GmbH. (2025, September 10). Veröffentlichung des DTV- und NRW-Entfernungswerk. https://assets.static-bahn.de/dam/jcr:4a072076-d5be-41a5-b864-a3cc074e5d1e/1%20Vorbemerkungen.pdf

Dreifache Auszeichnung für ÖPNV-Apps von Hacon. (2025, September 10). Retrieved September 10, 2025, from https://www.hacon.de/news/meldungen/app-awards-2024/

European Commission. (2017, May 31). *COMMISSION DELEGATED REGULATION (EU) 2017/1926 of 31 May 2017 supplementing Directive 2010/40/EU of the European Parliament and of the Council with regard to the provision of EU-wide multimodal travel information services.* Retrieved September 19, 2025, from https://eur-lex.europa.eu/eli/reg_del/2017/1926/oj/eng
Usr_lan: EN.

Extended GTFS Route Types | Static Transit. (2025, September 8). Google for Developers. Retrieved September 8, 2025, from https://developers.google.com/transit/gtfs/reference/extended-route-types

Fabrizio Arneodo. (2015, October). *01.NeTEx-Introduction-WhitePaper_1.03.* Retrieved September 20, 2025, from https://transmodel-cen.eu/wp-content/uploads/2024/07/01.NeTEx-Introduction-WhitePaper_1.03.pdf

Goldstein, B., & Dyson, L. (Eds.). (2013). *Beyond transparency: Open data and the future of civic innovation.* Code for America Press.

Hacon - A Siemens Company. (2025, September 10). Retrieved September 10, 2025, from https://www.hacon.de/unternehmen/

HAFAS Rohdaten Format (HRDF) – Open Data-Plattform Mobilität Schweiz. (2025, September 10). Retrieved September 10, 2025, from https://opentransportdata.swiss/de/cookbook/timetable-cookbook/hafas-rohdaten-format-hrdf/

HAFAS.engine_english. (2025, September 10). Retrieved September 10, 2025, from https://www.hacon.de/fileadmin/user_upload/Portfolio/Factsheets/HAFAS/HAFAS.engine_english.pdf

History – Transmodel. (10/09/2025, 11:57:38). Retrieved September 10, 2025, from https://transmodel-cen.eu/index.php/history/

Horaires et arrêts des transport publics (GTFS) - Portail Open Data. (2025, September 8). Retrieved September 8, 2025, from https://data.public.lu/en/datasets/horaires-et-arrets-des-transport-publics-gtfs/

IBNR ID. (2025, September 10). Retrieved September 10, 2025, from https://www.wikidata.org/wiki/Property:P954

Identification of Fixed Objects in Public Transport. (2025). Retrieved September 11, 2025, from https://www.wikidata.org/wiki/Q5988215

Instance of. (2025, September 12). Retrieved September 12, 2025, from https://www.wikidata.org/wiki/Property:P31

International Standardisation: The European Rail Associations Vison. (2021, April 29). Retrieved September 8, 2025, from https://www.unife.org/

wp-content/uploads/2021/05/INTERNATIONAL-STANDARDISATION-THE-EUROPEAN-RAIL-ASSOCIATIONS-VISION.pdf

JourneyPattern (Abstract in EPIP), ServicePattern - NeTEx Profil Österreich - Mobilitätsverbünde. (2025, September 15). Retrieved September 15, 2025, from https://mobilitaetsverbuende.atlassian.net/wiki/spaces/NET/pages/180715593/JourneyPattern+Abstract+in+EPIP+ServicePattern

JUIL 2025_Carte reseau CFL_EN_A3_PRINT. (2025). Retrieved September 17, 2025, from https://www.cfl.lu/getattachment/50fc8908-06be-462f-a8ba-8c49eb1f3927/juil-2025_carte-reseau-cfl_en_a3_print.pdf

Key:ref:IFOPT – OpenStreetMap Wiki. (2025). Retrieved September 21, 2025, from https://wiki.openstreetmap.org/wiki/Key:ref:IFOPT

Key:uic_ref – OpenStreetMap Wiki. (2025, September 8). Retrieved September 8, 2025, from https://wiki.openstreetmap.org/wiki/Key:uic_ref

Knowles, N. (2024). Outline comparison and Mapping between NeTEx & GTFS. https://transmodel-cen.eu/index.php/papers/

Maria, E., Budiman, E., Haviluddin, & Taruk, M. (2020). Measure distance locating nearest public facilities using Haversine and Euclidean Methods. *Journal of Physics: Conference Series*, *1450*(1), 012080. https://doi.org/10.1088/1742-6596/1450/1/012080

MVO - Datenbereitstellungsplattform. (2025, September 8). Retrieved September 8, 2025, from https://data.mobilitaetsverbuende.at/de/data-sets

National Access Points - European Commission. (2025). Retrieved September 19, 2025, from https://transport.ec.europa.eu/transport-themes/smart-mobility/road/its-directive-and-action-plan/national-access-points_en

NeTEx – Transmodel. (2025, April 29). Retrieved April 29, 2025, from https://transmodel-cen.eu/index.php/netex/

Nicholas JS Knowles. (2015, October). *04.NeTEx-Framework-WhitePaper_1.07*. Retrieved September 8, 2025, from https://transmodel-cen.eu/wp-content/uploads/2024/07/04.NeTEx-Framework-WhitePaper_1.07.pdf

NORWAY Train Travel Information | railcc. (2025). Retrieved September 17, 2025, from https://rail.cc/norway/xno

ÖBB Open Data Datensätze. (2025, September 8). ÖBB Open Data. Retrieved September 8, 2025, from https://data.oebb.at/de/datensaetze

Open data about railway stations. (2025). Retrieved September 11, 2025, from https://www.rijdendetreinen.nl/en/open-data/stations

Ovenhagen, L. (2021). A design vision towards seamless European train journeys. Retrieved September 19, 2025, from https://repository.tudelft.nl/record/uuid:01a0e501-2e1a-469d-b1c3-03df7abae737

Overview - General Transit Feed Specification. (2025, April 30). Retrieved April 30, 2025, from https://gtfs.org/documentation/overview/

Pandas.DataFrame.merge — pandas 2.3.2 documentation. (2025, September 10). https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.merge.html

Pandas.read_csv — pandas 2.3.2 documentation. (2025). Retrieved September 29, 2025, from https://pandas.pydata.org/docs/reference/api/pandas.read_csv.html

PointinJourneyPattern (Abstract in EPIP), StopPointInJourneyPattern - NeTEx Profil Österreich - Mobilitätsverbünde. (2025). Retrieved September 15, 2025, from https://mobilitaetsverbuende.atlassian.net/wiki/spaces/NET/pages/181141601/PointinJourneyPattern+Abstract+in+EPIP+StopPointInJourneyPattern

Protocol Buffers. (2025, April 30). Retrieved April 30, 2025, from https://protobuf.dev/

Publish - General Transit Feed Specification. (2025, September 22). Retrieved September 22, 2025, from https://gtfs.org/getting-started/publish/

Raffael Rittmeier. (2016, August 24). *VDV-Schnittstellenparameter*. Retrieved September 10, 2025, from https://www.zvbn.de/media/data/06_20160824_Anlage-6-VDV-Schnittstellenparameter.pdf

Railway station. (2025, September 12). Retrieved September 12, 2025, from https://www.wikidata.org/wiki/Q55488

railways, U.-I. union of. (2015, July 30). *Country Codes*. UIC - International union of railways. Retrieved September 11, 2025, from https://uic.org/support-activities/it/article/country-codes

Reference - General Transit Feed Specification. (2025, September 8). Retrieved September 8, 2025, from https://gtfs.org/documentation/schedule/reference/

ScheduledStopPoint, ServiceLink - NeTEx Profil Österreich - Mobilitätsverbünde. (2025). Retrieved September 15, 2025, from https://mobilitaetsverbuende.atlassian.net/wiki/spaces/NET/pages/180944962/ScheduledStopPoint+ServiceLink

Skille, E. (2024). Standards for public transport data.

Soares, I., & Martins, P. M. (2013). PUBLIC TRANSPORT STANDARDIZATION.

Stop Places - NeTEx Profil Österreich - Mobilitätsverbünde. (08/09/2025, 18:06:33). Retrieved September 8, 2025, from https://mobilitaetsverbuende.atlassian.net/wiki/spaces/NET/pages/181272577/Stop+Places

Stops - Håndbok N801 (SIRI/NeTEX) - Entur. (2024, October 17). Retrieved September 8, 2025, from https://enturas.atlassian.net/wiki/spaces/PUBLIC/pages/728727661/stops#StopPlace

Stops - Håndbok N801 (SIRI/NeTEX) - Entur. (2025). Retrieved September 15, 2025, from https://enturas.atlassian.net/wiki/spaces/PUBLIC/pages/728727661/stops#StopPlace.1

Stops- and Timetable data. (2025, September 8). Retrieved September 8, 2025, from https://developer.entur.org/stops-and-timetable-data

Subclass of. (2025, September 12). Retrieved September 12, 2025, from https://www.wikidata.org/wiki/Property:P279

Timetable - Håndbok N801 (SIRI/NeTEX) - Entur. (2025, September 15). Retrieved September 15, 2025, from https://enturas.atlassian.net/wiki/spaces/PUBLIC/pages/728760393/timetable#ServiceJourney

Trans-European Transport Network (TEN-T) - European Commission. (2025). Retrieved September 8, 2025, from https://transport.ec.europa.eu/transport-themes/infrastructure-and-investment/trans-european-transport-network-ten-t_en

UIC station code. (2025, September 11). Retrieved September 11, 2025, from https://www.wikidata.org/wiki/Property:P722

UIC Vademecum. (2025). Retrieved September 11, 2025, from https://vademecum.uic.org/

Unlocking the potential of mobility data | Shaping Europe's digital future. (2025). Retrieved September 19, 2025, from https://digital-strategy.ec.europa.eu/en/policies/mobility-data

Verkehrsverbünde in Österreich. (2025, September 8). Retrieved September 8, 2025, from https://www.bmimi.gv.at/themen/mobilitaet/transport/nahverkehr/verkehrsverbuende/oesterreich.html

What is GTFS? - General Transit Feed Specification. (2025, April 30). Retrieved April 30, 2025, from https://gtfs.org/getting-started/what-is-GTFS/

Why use GTFS? - General Transit Feed Specification. (2025, September 8). Retrieved September 8, 2025, from https://gtfs.org/getting-started/why-use-GTFS/

Wien Hauptbahnhof. (2025, September 8). Retrieved September 8, 2025, from https://www.wikidata.org/wiki/Q697300

wtw. (2025, September 2). *Essential Google Maps Statistics & Trends to Watch in 2025*. Retrieved September 22, 2025, from https://www.loopexdigital.com/blog/google-maps-statistics

Xml.etree.ElementTree — The ElementTree XML API. (2025). Python documentation. Retrieved September 29, 2025, from https://docs.python.org/3/library/xml.etree.elementtree.html

Zahlen, Daten, Fakten. (2025, September 8). ÖBB-Infrastruktur AG. Retrieved September 8, 2025, from https://infrastruktur.oebb.at/de/unternehmen/zahlen-daten-fakten

List of Abbreviations

GTFS General Transit Feed Specification

NeTEx Network Timetable Exchange

ÖBB Österreichische Bundes Bahnen

IBNR Interne Bahnhofsnummer

IFOPT Identification of Fixed Objects in Public Transport

CEN Comité Européen de Normalisation

IFOPT Identification of Fixed Objects in Public Transport

SIRI Standard Interface for Real-time Information

UIC Union Internationale des Chemins de fer

MMTIS Multi Model Travel Information

TEN-T Trans-European Transport Network

XML Extensible Markup Language

CSV Comma Separated Values)

List of Aids / Tools

Table 2: Overview of aids and tools used in the thesis

Aid / Tool	**Usage**	**Relevant Sections / Chapters**	**Documentation**
Python	Data preprocessing, analysis and visualization	Chapter Methodes, Chapter Analysis	Python Docs
Jupyter Notebook	Interactive environment for executing Python code	Chapter ??	Jupyter Project
LaTeX	Thesis writing, typesetting, and references	All chapters	LaTeX Project
DeepL Write	Thesis writing	All chapters	DeepL Write
DeepL	Understanding and translation of English sources	All chapters	DeepL Translator
ChatGPT	Coding support, error detection and plot creation	Chapter Methodes, Chapter Analysis, Additional Material	ChatGPT
PowerPoint	Creating figures	Chapter Extraction of Station, Chapter GTFS Trip Extraction, Chapter NeTEx Journey Extraction	Microsoft PowerPoint

6 Additional Material

6.1 Notebook: Third Source

Wiki Data

WIKI Data set: One request per country (collecting all entries in a single request)

- extract the stations from the wiki data base with identifable ids if available
- drawbacks: if we do not have an ID which is also represented in the netex or gtfs datasets, we cannot sort the correct station
- benefits: one single request

For the Sparkql querry we only request subjects with the following entities:

- which are a train station/ train stop (small train station)
- which have an id codes (UIC, IFOPT, IBNR...)
- which have coordinates

```
from SPARQLWrapper import SPARQLWrapper, JSON
import pandas as pd

def sparl_ql_querry_request(country_code,country_name,limit):



    sparql = SPARQLWrapper("https://query.wikidata.org/sparql")
    sparql.setQuery(f"""
    SELECT ?station ?stationLabel ?coordinate ?ifopt ?ibnr ?uic WHERE
{{
      ?station wdt:P31/wdt:P279* wd:Q55488 ;  # instance or subclass
of train station
               wdt:P17 wd:Q{country_code} .              # located in
Austria (Q40)

      OPTIONAL {{ ?station wdt:P7824 ?ifopt. }}  # IFOPT code
      OPTIONAL {{ ?station wdt:P954 ?ibnr. }}    # IBNR
      OPTIONAL {{ ?station wdt:P722 ?uic. }}     # UIC
      OPTIONAL {{ ?station wdt:P625 ?coordinate. }}  # Coordinates

      SERVICE wikibase:label {{ bd:serviceParam wikibase:language
"[AUTO_LANGUAGE],de,en". }}
    }}
    LIMIT {limit}t
    """)
    sparql.setReturnFormat(JSON)
    wiki_data_query_results = sparql.query().convert()

    #print(wiki_data_query_results)

    # create a list ot store the result
```

```
    wiki_data_querry_results_list = []

    # iterate through the json format to extract information
    for result in wiki_data_query_results["results"]["bindings"]:

        # extract name
        name = result.get("stationLabel",{}).get("value")

        ifopt = result.get("ifopt",{}).get("value")

        # extract ibnr
        ibnr = result.get("ibnr", {}).get("value")

        # extract uic
        uic = result.get("uic", {}).get("value")

        # coordinates
        # make sure it is a string to apply later the replace function
otherwise issues will appear
        coordinate = str(result.get("coordinate", {}).get("value"))  #
"Point(lon lat)"

        # if the value is not None we extract the lat and lon
        if coordinate != "None":
            lon, lat = map(float, coordinate.replace("Point(",
"").replace(")", "").split(" "))

        # otherwise it will be None as value for lat and lon
        else:
            lon = lat = None

        # add to the list
        wiki_data_querry_results_list.append({"name_wiki": name,
"UIC_wiki": uic, "IBNR_wiki": ibnr, "IFOPT_wiki": ifopt,"lat_wiki":
lat, "lon_wiki": lon})

    wiki_data_querry_results_df =
pd.DataFrame(wiki_data_querry_results_list)

    # Drop rows where 'name_wiki' and 'lat_wiki'/'lon_wiki' appear
more than once, but keep the first
    # it appears that same train station which have also tram stations
have two ifopt numbers
    df_cleaned =
wiki_data_querry_results_df[~wiki_data_querry_results_df.duplicated(su
bset=['name_wiki', 'lat_wiki', 'lon_wiki'], keep='first')]


    # saving the dataframe in order to use for the second part
2_compare
```

```
    df_cleaned.to_csv(country_name+ "_" + 'wiki_data_df.csv')

    return (df_cleaned,wiki_data_query_results)



# Set up query parameters
country_code = "40"
country_name = "austria"
limit = 1031 + 1000

wiki_data_df, wiki_data_query_results =
sparl_ql_querry_request(country_code, country_name, limit)

# country codes:
# luxembourg q32
# norway q20
# austria q40
```

6.2 Notebook: Extraction

Data Extraction: Netex and GTFS

```
# load necessary packages
#!pip install partridge
#!pip install pandas
#!pip install shapely
#!pip install osmium #open street maps tool
#!pip install sparkql
#!pip install geopandas
```

GTFS load

```
# GTFS extraction
import partridge as ptg
import zipfile
import pandas as pd

def load_gtfs(gtfs_zip_path):

    # Load the entire GTFS feed (no filters)
    feed = ptg.load_feed(gtfs_zip_path, view={})

    # extract all files within the zipped

    gtfs_list = []

    # loop trough all the files and save them in a list
    with zipfile.ZipFile(gtfs_zip_path, 'r') as z:
        for file_name in z.namelist():
            if file_name.endswith('.txt'):
                #print("Reading:",file_name)

                with z.open(file_name) as f:
                    df = pd.read_csv(f,low_memory=False)
                    #print(df.head())

                    # create a tulpe to store them into a list
(dictionary does not work: df is not hashable)
                    tulpe = (file_name,df)
                    gtfs_list.append(tulpe)

    return gtfs_list
```

Netex load

- It appears that each netex file is not well standardised because the netex files of other countries have different structures and hierarchies which make it really difficult to work with them

- Therefore a more robust approach is needed: recursive function which identifies each single xml file in nested deep structures

```
import zipfile
import io
import xml.etree.ElementTree as ET
import os


def extract_xml_from_zip(zip_data, zip_path=""):
    """Recursively extracts all .xml files from a zip (bytes or file),
including nested zips."""
    xml_files = []

    with zipfile.ZipFile(io.BytesIO(zip_data)) if isinstance(zip_data,
bytes) else zipfile.ZipFile(zip_data, 'r') as zip_ref:
        for item_name in zip_ref.namelist():
            full_path = os.path.join(zip_path, item_name)

            # Skip directories
            if item_name.endswith('/'):
                continue

            with zip_ref.open(item_name) as file:
                data = file.read()

                # another xml file was identified
                if item_name.endswith('.xml'):
                    xml_files.append((full_path, data))

                # another zip folder was identified
                elif item_name.endswith('.zip'):
                    # Recursive call
                    xml_files.extend(extract_xml_from_zip(data,
zip_path=full_path))

    return xml_files


def load_netex(netex_path):
    """Load Netex .zip and extract all .xml files including those
inside nested zips and folders."""
    all_xml_files = []

    # Outer zip
    with zipfile.ZipFile(netex_path, 'r') as outer_zip:
        for file_name in outer_zip.namelist():

            # another folder was identified
            if file_name.endswith('/'):
```

```
                continue  # Skip directory names

            with outer_zip.open(file_name) as file:
                file_data = file.read()

                # another zipped was identified
                if file_name.endswith('.zip'):
                    # Recurse into nested zip

all_xml_files.extend(extract_xml_from_zip(file_data,
zip_path=file_name))

                # another xml file was identified
                elif file_name.endswith('.xml'):
                    all_xml_files.append((file_name, file_data))

    return all_xml_files
```

Extraction Station: Modul 1

GTFS extraction: Stops (Station)

Since we focus on railway, the gtfs data sets do not include any simple solution as netex to differ between bus stations and rail stations. Therefore we need to filter the data over following connection:

Stop (Stop_ID) -> Stop_times (stop_id, trip_id) -> trips (trip_id, route_id) -> route (route_id, route_type)

Our goal is the route_type which defines wether a trip is operated by bus or train

source: https://gtfs.org/documentation/schedule/reference/#routestxt

```
def stop_extract_gtfs(gtfs_list):

    # use the data set above for extraction
    for gtfs_data in gtfs_list:


        if gtfs_data[0] == "stops.txt":
            stop_df = gtfs_data[1]

        # Filtering the stop list by their type of trips


        if gtfs_data[0] == "stop_times.txt":
            stop_times_df = gtfs_data[1]
```

```
        if gtfs_data[0] == "trips.txt":
            trips_df = gtfs_data[1]


        if gtfs_data[0] == "routes.txt":
            route_df = gtfs_data[1]


    # Filter the stations which are no train staitions
    # 1. Keep only the needed columns
    stop_df_filterd = stop_df[["stop_id"]]   # or stop_df if named
like that
    stop_times_df = stop_times_df[["stop_id", "trip_id"]]
    trips_df = trips_df[["trip_id", "route_id"]]
    route_df = route_df[["route_id", "route_type"]]

        # 2. Merge step by step with reduced data
    stop_trips = pd.merge(stop_times_df, trips_df, on="trip_id",
how="inner")
    stop_routes = pd.merge(stop_trips, route_df, on="route_id",
how="inner")
    stop_modes = pd.merge(stop_routes, stop_df_filterd,on="stop_id",
how="inner")

    # 3. Final: stop_id ↔ route_type
    result = stop_modes[["stop_id", "route_type"]].drop_duplicates()
    result["route_type"] = result["route_type"].astype(str)

    # filter the results by their route_type
    # also we consider norways special type of numbers
    result_filterd = result[
    (
        result["route_type"].str.startswith(("1", "4")) &
        (result["route_type"].str.len() == 3)
    )
    |
    (result["route_type"].isin(["1","2", "12", "5", "7"]))
    ]

    stop_final_df = pd.merge(result_filterd, stop_df, on="stop_id",
how="inner")


    # rename them to make them better readable
    stop_final_df = stop_final_df.rename(columns={
        'stop_id': 'id_gtfs',
        'stop_name': 'name_gtfs',
        "stop_lat":"lat_gtfs",
        "stop_lon":"lon_gtfs",
```

```
        "parent_station":"parent_station_gtfs",
        "wheelchair_boarding":"wheelchair_boarding_gtfs",
        "platform_code":"platform_code_gtfs",
        "route_type":"route_type_gtfs"})

    return stop_final_df
```

Netex extraction: StopPlace (Station)

- Netex files can consist of multiple xml files, each single file will be applied to the function as xml_content
- as the result a list of dictionaries (each entry is a dictionary) the content is provided for each xml file
- thats why the extraction of netex is so complicated because you need to iterate over a list of xml files to extract information

```
import pandas as pd
import geopandas as gpd
from shapely.geometry import Point

def stop_extract_netex(xml_content, ns):
    netex_stops_list = []  # Collect dictionaries here


    # Parse the XML content
    tree = ET.parse(io.BytesIO(xml_content))
    root = tree.getroot()


    for stop_place in root.findall('.//netex:StopPlace', ns):

        # reset all the variables in the case that no value was found
for some variables
        netex_id = PublicCode = name = latitude = longitude =
Wheelchair_access_list = AssistanceFacility = AssistanceAvailability =
AccessFacility = EVA_Nr = None

        # extract ID
        netex_id = stop_place.attrib.get('id')

        # exract PublicCode

        PublicCode_elem = stop_place.find("netex:PublicCode",ns)
        PublicCode = PublicCode_elem.text if PublicCode_elem is not
None else None

        # extract name
        name_elem = stop_place.find('netex:Name', ns)
        name = name_elem.text if name_elem is not None else None
```

```
        # extract the StopPlace type e.g. rail, bus, taxi
        StopPlaceType_elm = stop_place.find("netex:StopPlaceType",ns)
        StopPlaceType = StopPlaceType_elm.text if StopPlaceType_elm is
not None else None

        #   {http://www.netex.org.uk/netex}StopPlaceType: railStation

        # extract coordinates

        # clear coordinate values
        latitude = longitude = None

        # extract the coordinates of the station
        centroid = stop_place.find('netex:Centroid', ns)
        if centroid is not None:
            location = centroid.find('netex:Location', ns)
            if location is not None:
                lat_elem = location.find('netex:Latitude', ns)
                lon_elem = location.find('netex:Longitude', ns)
                latitude = float(lat_elem.text) if lat_elem is not
None else None
                longitude = float(lon_elem.text) if lon_elem is not
None else None


        # extract the each referenced key from the netex data set
        keyList = stop_place.find('netex:keyList', ns)
        if keyList is not None:

            # iterate through the list of keys
            for KeyValue in keyList:
                if (KeyValue.find("netex:Key",ns)).text == "EVA-Nr":
                    EVA_Nr_elem = KeyValue.find("netex:Value",ns)

                    if EVA_Nr_elem is not None:
                        EVA_Nr = EVA_Nr_elem.text

        """
        Quays
        """
        # wheel chair access

        quays = stop_place.find('netex:quays', ns)
        if quays is not None:

            # findall to get all entries for all quays
            Quay = quays.findall('netex:Quay', ns)

            # iterate through the list to find the Quay to find
```

```
            # and if possible the uic code and quay id
            # the wheel chair information (Quay == Gleis, Plattform)

            Wheelchair_access_list = []
            quay_id_list = []

            # iterate the the entries of quay (quay = Bahngleis)
            for x in Quay:

                """
                Quay ID and UIC code
                """
                # reset each time to avoid error
                quay_id = uicCode = None


                quay_id = x.attrib.get('id')
                quay_id_list.append(quay_id)


                keyList = x.find('netex:keyList', ns)
                if keyList is not None:

                    KeyValue = keyList.findall('netex:KeyValue', ns)

                    for y in KeyValue:

                        key_elem = y.find('netex:Key', ns)

                        if key_elem is not None and key_elem.text ==
"uicCode":

                            uicCode_elem = y.find('netex:Value', ns)

                            if uicCode_elem is not None:

                                uicCode = uicCode_elem.text


                """
                Wheelchair access
                """

                # case: if no label was found but the wheelchair
access information is still found
                Label = WheelchairAccess = None

                StopPlaceSpaceGroup =
x.find('netex:StopPlaceSpaceGroup', ns)
                if StopPlaceSpaceGroup is not None:
```

```
                    # store the information where the wheelchair
access is true or unknown
                    Label_elem =
StopPlaceSpaceGroup.find('netex:Label', ns)
                    if Label_elem is not None:
                        Label = str(Label_elem.text)

                    SiteComponentGroup =
StopPlaceSpaceGroup.find('netex:SiteComponentGroup', ns)
                    if SiteComponentGroup is not None:

                        SiteElementObjectElementGroup =
SiteComponentGroup.find('netex:SiteElementObjectElementGroup', ns)
                        if SiteElementObjectElementGroup is not None:

                            SiteElementInternalGroup =
SiteElementObjectElementGroup.find('netex:SiteElementInternalGroup',
ns)
                            if SiteElementInternalGroup is not None:

                                AccessibilityAssessment =
SiteElementInternalGroup.find("netex:AccessibilityAssessment", ns)
                                if AccessibilityAssessment is not
None:

                                    limitations =
AccessibilityAssessment.find('netex:limitations', ns)
                                    if limitations is not None:

                                        AccessibilityLimitation =
limitations.find('netex:AccessibilityLimitation', ns)
                                        if AccessibilityLimitation is
not None:

                                            MobilityLimitationGroup =
AccessibilityLimitation.find('netex:MobilityLimitationGroup', ns)

                                            if MobilityLimitationGroup
is not None:

                                                WheelchairAccess_elem
= MobilityLimitationGroup.find('netex:WheelchairAccess', ns)
                                                if
WheelchairAccess_elem is not None:

                                                    WheelchairAccess =
str(WheelchairAccess_elem.text)
                                                    # store the result
```

```
as a tuple of WheelchairAccess and label (where is the wheelchair
access)

Wheelchair_access_list.append((WheelchairAccess,Label))



        """
        People with reduced mobility (PWRM)
        """


        facilities = stop_place.find("netex:facilities",ns)
        if facilities is not None:
            SiteFacilitySet =
facilities.find("netex:SiteFacilitySet",ns)

            if SiteFacilitySet is not None:

                CommonFacilityGroup =
SiteFacilitySet.find("netex:CommonFacilityGroup",ns)
                if CommonFacilityGroup is not None:

                    # AssistanceFacility: stations with staff to
provide aid for people
                    AssistanceFacilityList =
CommonFacilityGroup.find("netex:AssistanceFacilityList",ns)
                    if AssistanceFacilityList is not None:

                        AssistanceFacility_elem =
AssistanceFacilityList.find("netex:AssistanceFacility",ns)

                        if AssistanceFacility_elem is not None:
                            AssistanceFacility =
str(AssistanceFacility_elem.text)


                    # AssistanceAvailability: staff which are
providing for poeple but must be booked previously
                    AssistanceAvailability_elem =
CommonFacilityGroup.find("netex:AssistanceAvailability",ns)
                    if AssistanceAvailability_elem is not None:

                        AssistanceAvailability =
str(AssistanceAvailability_elem.text)


            # AccessFacility: if the facility is accessable with a
wheelchair (Hebeliftbühne)
```

```
            SiteFacilityGroup =
SiteFacilitySet.find("netex:SiteFacilityGroup",ns)
            if SiteFacilityGroup is not None:

                AccessFacilityList =
SiteFacilityGroup.find("netex:AccessFacilityList",ns)

                if AccessFacilityList is not None:
                    AccessFacility_elem =
AccessFacilityList.find("netex:AccessFacility",ns)

                    if AccessFacility_elem is not None:
                        AccessFacility = str(AccessFacility_elem.text)




        # add structured data
        netex_stops_list.append({
            "id_netex": netex_id,
            "ref_id_netex":PublicCode,
            "StopPlaceType_netex": StopPlaceType,
            "name_netex": name,
            "lat_netex": latitude,
            "lon_netex": longitude,
            "EVA_Nr_netex":EVA_Nr,
            "UIC_Code_netex":uicCode,
            "Quay_ids_netex": quay_id_list,
            "WheelchairAccess_netex": Wheelchair_access_list,
            "AssistanceFacility_netex": AssistanceFacility,
            "AssistanceAvailability_netex": AssistanceAvailability,
            "AccessFacility_netex": AccessFacility
            })


    # return the list
    return netex_stops_list
```

Combine netex and gtfs

- Load: first load the data -> results gtfs a list of list, netex a list of xml files
- Extraction: gtfs find the correct list in our case "stops", netex search through all the different xml files for the root you need in our case StopPlace)

Labeling issues within netex and gtfs

Is the labeling the same for each station/ stop (is the id the same) compare with Netex and other data sets?

Austria case:

GTFS has a longer version of the IFOPT number which including a more detailed entries of each location of a train stop:

- xy:41:3087:0:2 is the IFOPT ID for the Bad Sauerbrunn Bahnhof but the train station have multiple other facilitys under the same ID core (xy:41:3087) e.g. Bus Stop, Entrance, Parc and ride
- in our case we need to filter the core ids and summarize them into a single row in order to merge the netex and the gtfs data set together

General case:

- each id of each data format can be different as the austrian data set had shwon
- each id of each data format can have a different pattern
- Also we cannot assume that each id is even closely similar to each other or the same

=> Either automatic pattern recognition is required

ID issues in Netex and GTFS

```
# this function identifies the ID pattern and sort them to the right
measures

import re

def identify_id_pattern(id_str):
    if pd.isna(id_str):
        return 'unknown'

    if re.match(r'^[a-z]{2}:\d+:\d+(?::\d+)*$', id_str) or
re.match(r'^[a-z]{2}-\d+-\d+', id_str):
        return 'austria'  # ifopt_style

    elif id_str.startswith('NSR:StopPlace:') or
id_str.startswith('NSR:Quay:'):
        return 'norway'


    elif re.match(r'^\d+$', id_str):
        return 'numeric_id'  # Luxembourg, simple numeric IDs

    else:
        return 'unknown'

# special case if the ids of netex or gtfs have different data types
like number and string
import re

def extract_digit_number(id_str, length):
```

```
    if isinstance(id_str, int):
        id_str = str(id_str)

    pattern = rf'(\d{{{length}}})'  # Removed \b to allow matches next
to colons, underscores, etc.
    match = re.search(pattern, str(id_str))

    return int(match.group(1)) if match is not None else None



def combine_gtfs_netex_by_ID(gtfs_stops_df,netex_stops_df):

    """
    Identify the id pattern of GTFS and NETEX
    There will be issues within netex and gtfs because it appears the
do not use the same ids
    """

    # we take the first entry of the gtfs and netex data set
    gtfs_id_list = gtfs_stops_df["id_gtfs"]
    netex_id_list = netex_stops_df["id_netex"]

    for id_gtfs, id_netex in zip(gtfs_id_list, netex_id_list):
        print("id_type gtfs:",id_gtfs)
        print("id_type netex:",id_netex)

        # provide the id's as strings in order to identify their
patterns
        id_pattern_netex = identify_id_pattern(str(id_netex))
        id_pattern_gtfs = identify_id_pattern(str(id_gtfs))

        # in the case that the id pattern is known and the patterns
are the same in both data formats
        if id_pattern_netex != "unknown" and id_pattern_gtfs !=
"unknown" and id_pattern_netex == id_pattern_gtfs:
            # then we can proceed the following measures
            id_pattern = id_pattern_netex = id_pattern_gtfs
            break


        # in the case that one id is unknown and one id is numeric but
the patterns have different types
        # idea: the numeric values must appear in the string
        # identify the length of the id in order to find the same
length if integers within the other type of id
        elif id_pattern_netex == "numeric_id" or id_pattern_gtfs ==
"numeric_id" and id_pattern_netex != id_pattern_gtfs:
```

```
            print("IDs have different types:")
            print("GTFS:",id_pattern_gtfs)
            print("Netex:",id_pattern_netex)

            # the netex data set contains the numeric values -> the
gtfs data set must be adapted
            if id_pattern_netex == "numeric_id":

                length = len(str(id_netex))

                # call for each entry the extract_digit_number in
order to extract a number with the same length as the other id
                gtfs_stops_df["id_gtfs"] =
gtfs_stops_df["id_gtfs"].apply(lambda x: extract_digit_number(x,
length))

                # change the id patter to numeric in order to perform
the right measures
                id_pattern = "numeric_id"
                break

            # the gtfs data set contains the numeric values -> the
netex data set must be adapted
            elif id_pattern_gtfs == "numeric_id":

                # only strings have length, no integer
                length = len(str(id_gtfs))

                # call for each entry the extract_digit_number in
order to extract a number with the same length as the other id
                netex_stops_df["id_netex"] =
netex_stops_df["id_netex"].apply(lambda x: extract_digit_number(x,
length))

                # change the id patter to numeric in order to perform
the right measures
                id_pattern = "numeric_id"
                break



    """
    Measures to match the id type
    """
    print("ID type country:",id_pattern)

    """
    Austria Type ID: IFOPT
    gtfs: xy:43:1001:1:0
    NEtex: xy:43:1001
```

```
    Issue: xy:4001:1 -> also exist
    """

    if id_pattern == "austria":

        # Extract the common part (e.g., 'xy:41:3087') from GTFS and
NeTEx IDs
        # because netex and gtfs contain different stations from other
countries
        gtfs_stops_df['id_gtfs'] =
gtfs_stops_df['id_gtfs'].str.extract(r'^([a-z]{2}:\d+:\d+)',
expand=False)
        netex_stops_df['id_netex'] =
netex_stops_df['ref_id_netex'].str.extract(r'^([a-z]{2}:\d+:\d+)',
expand=False)

        # Group by this prefix and take the first entry for name, lat,
lon
        # this does only work if the gtfs data set is sorted that the
first entry is actually the correct entry
        gtfs_stops_df = gtfs_stops_df.groupby('id_gtfs',
as_index=False).first()


        # merge both data sets together by their IFOPT ID
        merged_df = pd.merge(gtfs_stops_df, netex_stops_df,
            left_on= "id_gtfs",
            right_on= "id_netex",
            how= "outer"
        )


    """
    Norway
    GTFS: NSR:Quay:100140/ NSR:StopPlace:890
    Netex: NSR:StopPlace:1 / NSR:Quay:100
    """

    if id_pattern == "norway":

        # replace all " " to make them more comparable
        gtfs_stops_df["id_gtfs"] =
gtfs_stops_df["id_gtfs"].apply(lambda x: x.replace(" ",""))
        netex_stops_df["id_netex"] =
netex_stops_df["id_netex"].apply(lambda x: x.replace(" ",""))

        # the
        gtfs_stops_df = (
            gtfs_stops_df.groupby("parent_station_gtfs").agg(
                parent_station_list_gtfs=("id_gtfs", list),
```

```
# list of quays
                route_type_gtfs=("route_type_gtfs", list),
                name_gtfs=("name_gtfs", "first"),
# keep first name
                lat_gtfs=("lat_gtfs", "first"),
                lon_gtfs=("lon_gtfs", "first"),
                stop_desc=("stop_desc", list),
                wheelchair_boarding_gtfs=("wheelchair_boarding_gtfs",
list),
                stop_timezone=("stop_timezone", list),
                vehicle_type=("vehicle_type", list),
                platform_code_gtfs=("platform_code_gtfs", list)
            )
            .reset_index()
        )
        gtfs_stops_df =
gtfs_stops_df.rename(columns={"parent_station_gtfs": "id_gtfs"})



         # merge both data sets together by their IFOPT ID
        merged_df = pd.merge(gtfs_stops_df,netex_stops_df,
                             left_on="id_gtfs",
                             right_on="id_netex",
                             how="outer")



    """
    numeric ids (Luxembourg)
    GTFS: 10172
    Netex: 1257
    """

    if id_pattern == "numeric_id":
        # convert both strings to integers

        gtfs_stops_df["id_gtfs"] =
gtfs_stops_df["id_gtfs"].apply(lambda x: int(x))
        netex_stops_df["id_netex"] =
netex_stops_df["id_netex"].apply(lambda x: int(x))


    # only if the id pattern was sorted the merging process will start
    if id_pattern != "unknown":

        # merge both data sets together by their IFOPT ID
        merged_df = pd.merge(gtfs_stops_df,netex_stops_df,
                             left_on="id_gtfs",
                             right_on="id_netex",
```

```
                                how="outer")


    return merged_df


#merged_df = combine_gtfs_netex_by_ID(gtfs_stops_reduced_df_3,
all_netex_stops_df_3)
```

Combine function

```
def combine_gtfs_netex(gtfs_zip_path, netex_path, ns, country_name):

    """
    Load
    """
    # load the gtfs data
    gtfs_list = load_gtfs(gtfs_zip_path)

    # load the netex data
    parsed_xml_files = load_netex(netex_path)

    print("Load completed")
    """
    Extraction netex

    Why do we must iterate here?
    - Netex consist of mulitple xml files stored in a list
    - We need to iterate through all the files to extract the
information of each single xml file (because the information is not
seperate as in gtfs)
    """
    # extract the netex train stop
    if len(parsed_xml_files) != 0:

        # create a list to store all results
        all_netex_stops_list = []

        # iterate through each single xml file and search for a
StopPlace
        for xml_name, xml_bytes in parsed_xml_files:

            # start the extraction function
            netex_stops_list = stop_extract_netex(xml_bytes, ns)

            # only if the extraction function find a StopPlace, we
will add it to a list
            if len(netex_stops_list) != 0:
                all_netex_stops_list.append(netex_stops_list)
                #print(netex_stops_list)
```

```
        # Flatten the list of lists into a single list because we have
list in list in lists
        # in order to proceed it into the dataframe
        flat_netex_stop_list = [item for sublist in
all_netex_stops_list for item in sublist]

        # Create a DataFrame
        all_netex_stops_df = pd.DataFrame(flat_netex_stop_list)
        print("Extraction: netex completed")
    """
    Extraction gtfs
    """

    # extract the gtfs train stop
    if len(gtfs_list) != 0:

        # gtfs extraction
        gtfs_stops_reduced_df = stop_extract_gtfs(gtfs_list)
        print("Extraction: gtfs completed")

    """
    combination of both gtfs and netex by id if possible
    """

    # elimate all double values by their ids
    gtfs_stops_reduced_df =
gtfs_stops_reduced_df.drop_duplicates(subset="id_gtfs")
    all_netex_stops_df =
all_netex_stops_df.drop_duplicates(subset='id_netex')

    # if a railStation type exist in the StopPlaceType, we will filter
the dataframe by railStation
    # we asssume here that at least on value exist the whole data set
is supposed to have such value
    if (all_netex_stops_df["StopPlaceType_netex"] ==
"railStation").any() == True:
        all_netex_stops_df =
all_netex_stops_df[all_netex_stops_df["StopPlaceType_netex"] ==
"railStation"]


    # only if both loading functions where sucessfull, the merge
function should be applied
    if (len(gtfs_list) != 0) and (len(parsed_xml_files) != 0):

        # merge both data sets
        merged_df = combine_gtfs_netex_by_ID(gtfs_stops_reduced_df,
all_netex_stops_df)
        print("Merging completed")
```

```
    # if one of the data formats was not sucessfully extracted
    else:
        merged_df = None


    # saving the dataframe in order to use for the second part
2_compare
    merged_df.to_csv(country_name + "_" + 'station_merged_df.csv')


    return merged_df, gtfs_stops_reduced_df, all_netex_stops_df
```

```
country_name = "luxembourg"

# Path to your GTFS zip file
gtfs_zip_path = "./data_sets/"+ country_name + "_gtfs.zip"

# Path to your outer zip file
netex_path = "./data_sets/"+ country_name +"_netex.zip"

# define the path of the netex xml tree in order to find the stop
places
ns = {'netex': 'http://www.netex.org.uk/netex'}


merged_df, gtfs_df, netex_df  = combine_gtfs_netex(gtfs_zip_path,
netex_path, ns, country_name)
```

```
Load completed
Extraction: netex completed
Extraction: gtfs completed
id_type gtfs: 500000079
id_type netex: DE::StopPlace:220401001_::
IDs have different types:
GTFS: numeric_id
Netex: unknown
ID type country: numeric_id
Merging completed
```

Extraction Trips: Modul 2

goals of extraction which information would be necessary for a trip

- stops of trips
- end stop
- start point
- Rail number like (RE 123)

Issue file size:

GTFS Files include often all kinds of trips like bus, train, taxi etc.

It appears that the gtfs files contain a variable to filter each trip by their mode of transportation. This means you can filter the data set over the route_mode variable which is defined by a identifable id for each mode of transport (https://ipeagit.github.io/gtfstools/reference/filter_by_route_type.html)

All numbers according starting with a 1 and have 3 characters are consider as trains

Role of Route and Trip:

Route is the blue print: which tracks exist in norway/luxenburg at all

GTFS Extraction: Trips

1. Issue: Connecting stops of a trip to a matching trip

   The stop for each trips are in a seperated list stop_times therefore we will extract the information and merge these list together

   Solution: We will extract from the stop_time dataframe all trips which have the id of trip dataframe therefore we iterate through the rows of the trip_df in order to take each single id and search for all entries with the same id in the stop_time dataframe.

2. Issue: Process time The data sets are too big to calculate every entry. Also, we do only focus on the rail ways therefore we will filter the data set before apply the Solution of 1. Issue.

E.g. the norway data set contains over 345.019 rows of trips by filtering it was reduced to 47.439 rows

```
2.1 Issue: Some data sets use the documantation provided by google the other one use the documentation provided by gtfs https://ipeagit.github.io/gtfstools/reference/filter_by_route_type.html

2.1 We also need to filter the stop_times_df because it has over 7.398.243 entries and after filtering it by the trip_id 535.075 entries which is an enormous reducing of processing time
```

How can we link the route and the trip:

The route have an column which defines the route_type in our case rail way is in our interest.

Route_Id provides the type of route Stop_times provides the stop per trip the trip provides each single trip

Route (route_id, route_type) -> trip (route_id) -> stop_times (route_id)

```
def trip_extract_gtfs(gtfs_list, country_name):

    # extract the trip dataframes
    for trip_list in gtfs_list:

        if trip_list[0] == "trips.txt":
            trip_df = trip_list[1]
            break

    # extracting the stops per trip
    for stop_times_list in gtfs_list:

        if stop_times_list[0] == "stop_times.txt":
            stop_times_df = stop_times_list[1]
            break

    # extracting the routes in order to filter the trips df
    for route_list in gtfs_list:

        if route_list[0] == "routes.txt":
            route_df = route_list[1]
            break


    # rename them to make them better readable
    # add to all columns "_gtfs"
    for column in trip_df:
        new_column_name = (column + "_gtfs")
        trip_df = trip_df.rename(columns={column: new_column_name})


    # rename them to make them better readable
    # add to all columns "_gtfs"
    for column in stop_times_df:
        new_column_name = (column + "_gtfs")
        stop_times_df = stop_times_df.rename(columns={column:
new_column_name})


    "Before we connect the stops of a trip to the an trip id we will
filter the dataframe in order to enhance the performance"

    # not all gtfs files use the same documentation to identify their
route_type
    # convert all at once instead of each single one
```

```
    route_df["route_type_str"] = route_df["route_type"].astype(str)

    route_df = route_df[
    (
        route_df["route_type_str"].str.startswith(("1", "4")) &
        (route_df["route_type_str"].str.len() == 3)
    )
    |
    (route_df["route_type_str"].isin(["2", "12", "5", "7"]))
]


    """
    We filter all trips by their route_id and then we filter all
stop_times by ther trip_id which was filter before by the route_id
    """
    # convert the column into a list to use it as a filter
    filter_for_trip = list(route_df.route_id)

    # we already changed the name with an _gfts at the end
    trip_df = trip_df[trip_df['route_id_gtfs'].isin(filter_for_trip)]

    # convert the column into a list to use it as a filter
    trip_id_filter_for_stop_times = list(trip_df.trip_id_gtfs)

    # we already changed the name with an _gfts at the end
    stop_times_df =
stop_times_df[stop_times_df['trip_id_gtfs'].isin(trip_id_filter_for_st
op_times)]


    "Now we connect the stops per trip to an trip id"

    # create a dictionary to store the stops and a unique id in order
to make it better sortable
    # create a dictionary with lists as containers
    stops_of_trips_dic = {"trip_id_gtfs": [],
                          "stops_on_trip": []}

    for index, row in trip_df.iterrows():

        # Filter all rows where trip_id is the same as from trip_df

        if not (stop_times_df[stop_times_df['trip_id_gtfs'] ==
row.trip_id_gtfs]).empty:
            # append to the lists of the dictionaries

stops_of_trips_dic["trip_id_gtfs"].append(row.trip_id_gtfs)
```

```
stops_of_trips_dic["stops_on_trip"].append(stop_times_df[stop_times_df
['trip_id_gtfs'] == row.trip_id_gtfs])
        else:

stops_of_trips_dic["trip_id_gtfs"].append(row.trip_id_gtfs)
            stops_of_trips_dic["stops_on_trip"].append(None)



    # create a dataframe of the dictionary in order to merge it with
the trip_df
    stops_of_trips_df = pd.DataFrame(stops_of_trips_dic)

    # merge both data sets together by their ID
    trip_merged_df = pd.merge(trip_df, stops_of_trips_df,
                        left_on= "trip_id_gtfs",
                        right_on= "trip_id_gtfs",
                        how= "inner")


    return trip_merged_df


"""
This function is for norways special gtfs structure: it is changing
inside the nested dataframes name from stop_id_gtfs" to "quay_id_gtfs
and
add from the stations dataframe the following Quay which are refering
to the actual stop_ids
"""

def norway_Quay_to_StopPlace(trip_merged_gtfs_df, station_df):

    def process_stops(stops_on_trip):
        # rename stop_id_gtfs -> quay_id_gtfs
        stops_on_trip = stops_on_trip.rename(columns={"stop_id_gtfs":
"quay_id_gtfs"})

        stop_id_gtfs_list = []
        for index, entry in stops_on_trip.iterrows():
            match = station_df[station_df["parent_station_list_gtfs"]
== entry.quay_id_gtfs]

            if not match.empty:
                stop_id_gtfs_list.append(match.id_gtfs.iloc[0])
            else:
                stop_id_gtfs_list.append(None)

        # assign the new list as a column
        stops_on_trip["stop_id_gtfs"] = stop_id_gtfs_list
```

```
        return stops_on_trip

    # apply the function to each row's stops_on_trip DataFrame
    trip_merged_gtfs_df["stops_on_trip"] =
trip_merged_gtfs_df["stops_on_trip"].apply(process_stops)

    return trip_merged_gtfs_df

def start_trip_extraction_gtfs(country_name,gtfs_zip_path,
station_df):

    # load gtfs
    gtfs_list = load_gtfs(gtfs_zip_path)
    print("gtfs loading completed")

    # extract the trips
    trip_merged_gtfs_df = trip_extract_gtfs(gtfs_list,country_name)
    print("gtfs extraction completed")


    # For the norway GTFS data set the refered station within the
stops_on_trip are actually Quay not a Station therefore we need to
transform them to stations
    if country_name == "norway":
        # first we prepare the station_df: defining columns and
filtering unecessary data
        station_df =
station_df[["id_gtfs","parent_station_list_gtfs"]]
        station_df =
station_df[station_df["parent_station_list_gtfs"].notna()]

                # flaten the embedded Quay list to single entries
        station_df["parent_station_list_gtfs"] =
station_df["parent_station_list_gtfs"].apply(ast.literal_eval)  #
safely convert string → list
        station_df = station_df.explode("parent_station_list_gtfs",
ignore_index=True)


        trip_merged_gtfs_df =
norway_Quay_to_StopPlace(trip_merged_gtfs_df, station_df)

    # saving the dataframe in order to use for the second part
2_compare
    trip_merged_gtfs_df.to_json(country_name + "_" +
'trip_gtfs_df.json',
                                orient = "records",  # row-wise list
of dicts
                                lines=False, # single JSON array
                                force_ascii=False) # # keep special
```

```
characters readable

    return trip_merged_gtfs_df


country_name = "norway"

gtfs_zip_path = "./data_sets/"+ country_name+"_gtfs.zip"

#trip_merged_gtfs_df =
start_trip_extraction_gtfs(country_name,gtfs_zip_path)
```

Netex extraction: SERVICE JOURNEY (Trips)

According to this: https://transmodel-cen.eu/wp-content/uploads/2024/06/2024-June_DATA4PT_GTFS-NeTEx-Mapping_vf.pdf, the trips are supposed to be located in the ServiceJourney section of the tree, however it appears that there is no guarantee that the ServiceJourney or the link to the StopPlace is included into the Netex dataframe

Where is ServiceJourney not included?

- Austria

It appears that austria is a geo data only data set which does not include any journeys. However the track line switches are included but no journey which could connect the track line switches together to a stop in a journey. There is one austria data set which includes all ÖBB journeys: https://data.mobilitaetsverbuende.at/de/data-sets

Where is the link not included?

- Luxenburg
- Norway

This should be where the information is stored in netex and gtfs

Concept	GTFS file/field	NeTEx element/attribute
Trip	`trips.txt` (`trip_id`)	`<ServiceJourney>` (with an `id`)
Stops in trip	`stop_times.txt` (`trip_id`, `stop_id`, `stop_sequence`)	`<ServiceJourneyPattern>` → `<StopPointInJourneyPattern>` (ordered list of stops)
Stop definitions	`stops.txt` (`stop_id`, lat/lon, name)	`<StopPlace>` / `<Quay>` (defines location, name, ID)
Route	`routes.txt` (`route_id`)	`<Route>`
Link trip ↔ route	`trips.txt.route_id`	`<ServiceJourney>` references a `<JourneyPattern>` which belongs to a `<Route>`

source: https://transmodel-cen.eu/wp-content/uploads/2024/06/2024-June_DATA4PT_GTFS-NeTEx-Mapping_vf.pdf

This is supposed to be the "standard" link from the Trip to StopPlace, however it does not appear in any data set:

ServiceJourney (JourneyPatternRef) -> Service JourneyPattern (ScheduledStopPointRef) -> ScheduledStopPoint (ScheduledStopPlaceRef) or PassengerStopAssignment (ScheduledStopPlaceRef, StopPlaceRef) -> StopPlace

However this connection is not given in each single xml file

How do we solve this missing connection issue?

It appears that within the netex data standard, there is no common way to structure your data. The structure for the xml file itself is the same thanks to the xml file standard however as we learned the netex files appears in many ways. There is not only one xml file where all information is stored, there can be multiple folders with different amounts of xml files. This makes the netex standard the xml files more individual and matching to the specification and purpose that multiple organizations at different levels could use netex exchange format.

https://transmodel-cen.eu/wp-content/uploads/2024/07/04.NeTEx-Framework-WhitePaper_1.07.pdf

According to the netex handbook there might be another way how to get the connection from ScheduledStopPoint to StopPlace via PassengerStopAssignment (source: https://enturas.atlassian.net/wiki/spaces/PUBLIC/pages/728563886/network)

In the following we will search for each single xml file by following tags of the xml files:

1. ServiceJourney
2. ServiceJourneyPattern
3. ScheduledStopPoint
4. PassengerStopAssignment (a link between ScheduledStopPoint and StopPlace)
5. StopPlace (would not be necessary because we already extract this part) (Norway: Quay within the Station dataframe)

```
# to discover the structure of an xml file
def explore_element(elem, level=0):
    indent = "  " * level
    attrs = ", ".join([f'{k}="{v}"' for k, v in elem.attrib.items()])
    tag_str = f"{elem.tag} [{attrs}]" if attrs else elem.tag
    print(f"{indent}{tag_str}: {elem.text.strip() if elem.text else
''}")
    for child in elem:
        explore_element(child, level + 1)
```

Extraction of each following elements:

1. ServiceJourney

```
def ServiceJourney_extraction_netex(xml_content, ns):
    ServiceJourney_list = []  # Collect dictionaries here

    # Parse the XML content
    tree = ET.parse(io.BytesIO(xml_content))
    root = tree.getroot()

    for ServiceJourney in root.findall('.//netex:ServiceJourney', ns):


        # reset all the variables in the case that no value was found
for some variables
        netex_service_id = TransportMode = StartPointInPatternRef =
EndPointInPatternRef = TrainNumberRef = TimetabledPassingTime_list =
None


        # extract ID
        netex_service_id = ServiceJourney.attrib.get('id')

        # extract the type of the journey (e.g. bus, train)
        TransportMode_elem =
ServiceJourney.find('netex:TransportMode', ns)
        if TransportMode_elem is not None:

            TransportMode = TransportMode_elem.text

        # extract the the start and end point of each journey
        noticeAssignments =
ServiceJourney.find('netex:noticeAssignments', ns)

        if noticeAssignments is not None:
            NoticeAssignment_b =
noticeAssignments.find('netex:NoticeAssignment', ns)

            StartPointInPatternRef = EndPointInPatternRef = None

            # extract the start and ending point of the journey
            if NoticeAssignment_b is not None:
                #start_elem =
NoticeAssignment_b.find('netex:StartPointInPatternRef', ns)
                #end_elem =
NoticeAssignment_b.find('netex:EndPointInPatternRef', ns)
```

```
                start_elem =
NoticeAssignment_b.find('./netex:StartPointInPatternRef', ns)
                end_elem =
NoticeAssignment_b.find('./netex:EndPointInPatternRef', ns)

                if start_elem is not None:
                    StartPointInPatternRef =
start_elem.attrib.get('ref')
                if end_elem is not None:
                    EndPointInPatternRef = end_elem.attrib.get('ref')


        # extract the train reference
        trainNumbers = ServiceJourney.find('netex:trainNumbers', ns)
        if trainNumbers is not None:

            TrainNumberRef_elem =
trainNumbers.find('netex:TrainNumberRef', ns)
            if TrainNumberRef_elem is not None:
                TrainNumberRef = TrainNumberRef_elem.attrib.get('ref')



        """
        Extraction of stops within the journey
        """


        # extract all stops of the journey
        passingTimes = ServiceJourney.find('netex:passingTimes', ns)
        if passingTimes is not None:

            # findall to get all entries for all TimetabledPassingTime
            TimetabledPassingTime =
passingTimes.findall('netex:TimetabledPassingTime', ns)


            # iterate through the list of passingtimes (stops on the
journey)

            # following information will be extracted
            # "DE::StopPointInJourneyPattern:321018_1_0::"
            # "14:33:00"

            # the results will be stored in a list of dictionaries
            TimetabledPassingTime_list = []
            for x in TimetabledPassingTime:

                # reset the values to ensure that no error appears if
one of these information is missing
```

```
                StopPointInJourneyPatternRef = ArrivalTime =
DepartureTime = None

                # extract the stop in the journey
                StopPointInJourneyPatternRef_elem =
x.find('netex:StopPointInJourneyPatternRef', ns)
                if StopPointInJourneyPatternRef_elem is not None:

                    StopPointInJourneyPatternRef =
StopPointInJourneyPatternRef_elem.attrib.get('ref')

                 # extract the arrival time of the stop
                ArrivalTime_elem = x.find('netex:ArrivalTime', ns)
                if ArrivalTime_elem is not None:

                    ArrivalTime = str(ArrivalTime_elem.text)

                # extract the departure time of the stop
                DepartureTime_elem = x.find('netex:DepartureTime', ns)
                if DepartureTime_elem is not None:

                    DepartureTime = str(DepartureTime_elem.text)

                # create a dictionary to store the values and add it
to the list


TimetabledPassingTime_list.append({"Stop":StopPointInJourneyPatternRef
,
                                                   "Arrival_Time:":
ArrivalTime,
                                                    "Departure_Time":
DepartureTime})

        # add structured data
        ServiceJourney_list.append({
            "id_service_netex": netex_service_id,
            "journey_type_netex":TransportMode, # luxenburg data set
includes this
            "start_point_netex":StartPointInPatternRef,
            "end_point_netex": EndPointInPatternRef,
            "journey_number_netex": TrainNumberRef,
            "TimetabledPassingTime_netex": TimetabledPassingTime_list
            })


    # return the list
    return ServiceJourney_list
```

2. ServiceJourneyPattern

It appears that in the Norway data set the ServiceJourneyPattern is named JourneyPattern that is why this function needs an specific tag

```
def ServiceJourneyPattern_extraction_netex(xml_content, ns,tag_name):
    ServiceJourneyPattern_list = []  # Collect dictionaries here

    # Parse the XML content
    tree = ET.parse(io.BytesIO(xml_content))
    root = tree.getroot()


    for ServiceJourneyPattern in root.findall('.//netex:'+ tag_name ,
ns):


        # reset all the variables in the case that no value was found
for some variables
        ServiceJourneyPattern_id = RouteView = LineRef = DirectionRef
= StopPointInJourneyPattern_results =  None

        # extract ID
        ServiceJourneyPattern_id =
ServiceJourneyPattern.attrib.get('id')

        # extract routeView
        RouteView_elem = ServiceJourneyPattern.find('netex:RouteView',
ns)
        if RouteView_elem is not None:

            RouteView = RouteView_elem.attrib.get('id')

            LineRef_elem = RouteView_elem.find('netex:LineRef', ns)

            if LineRef_elem is not None:
                LineRef = LineRef_elem.attrib.get('ref')

        # extract the direction if the journey
        DirectionRef_elem =
ServiceJourneyPattern.find('netex:DirectionRef', ns)

        if DirectionRef_elem is not None:

            DirectionRef = DirectionRef_elem.attrib.get('ref')

        # extract all stops of the journey
        pointsInSequence =
ServiceJourneyPattern.find('netex:pointsInSequence', ns)
```

```
        if pointsInSequence is not None:

            StopPointInJourneyPattern_list =
pointsInSequence.findall('netex:StopPointInJourneyPattern', ns)

            if StopPointInJourneyPattern_list is not None:

                # create dictionary to store results
                StopPointInJourneyPattern_results = []
                # iterate through the list of
StopPointInJourneyPattern
                for x in StopPointInJourneyPattern_list:

                    # reset the values to None to avoid errors if the
information was not found
                    ForBoarding = ScheduledStopPointRef =
StopPointInJourneyPattern_id = None

                    # extract the id of the STopPoint in
JourneyPattern
                    StopPointInJourneyPattern_id = x.attrib.get("id")

                    # extract the id of ScheduledStopPointRef
                    ScheduledStopPointRef_elem =
x.find('netex:ScheduledStopPointRef', ns)

                    if ScheduledStopPointRef_elem is not None:
                        ScheduledStopPointRef =
ScheduledStopPointRef_elem.attrib.get("ref")

                    ForBoarding_elem = x.find('netex:ForBoarding', ns)

                    if ForBoarding_elem is not None:
                        ForBoarding = ForBoarding_elem.text

                    # add new values to the list of in the dictionary

StopPointInJourneyPattern_results.append({"StopPointInJourneyPattern_i
d":StopPointInJourneyPattern_id,

"ScheduledStopPointRef": ScheduledStopPointRef,
                                                    "Departure_Time":
ForBoarding})


        # add structured data
        ServiceJourneyPattern_list.append({
            "ServiceJourneyPattern_id": ServiceJourneyPattern_id,
            "RouteView": RouteView,
            "LineRef": LineRef,
```

```
            "DirectionRef": DirectionRef,

"StopPointInJourneyPattern":StopPointInJourneyPattern_results,
            })


    # return the list
    return ServiceJourneyPattern_list
```

3. ScheduledStopPoint

```
def ScheduledStopPoint_extraction_netex(xml_content, ns):
    ScheduledStopPoint_list = []  # Collect dictionaries here

    # Parse the XML content
    tree = ET.parse(io.BytesIO(xml_content))
    root = tree.getroot()


    for ScheduledStopPoint in
root.findall('.//netex:ScheduledStopPoint', ns):


        # reset all the variables in the case that no value was found
for some variables
        ScheduledStopPoint_id = Key = Value = keyList_results = None

        # extract ID
        ScheduledStopPoint_id = ScheduledStopPoint.attrib.get('id')

        # extract all stops of the journey
        keyList = ScheduledStopPoint.find('netex:keyList', ns)
        if keyList is not None:

            KeyValue_list = keyList.findall('netex:KeyValue', ns)

            if KeyValue_list is not None:

                keyList_results = []
                for x in KeyValue_list:

                    Key_elem = x.find('netex:Key', ns)

                    Value_elem =  x.find('netex:Value', ns)

                    if Value_elem is not None and Key_elem.text ==
"StopPlaceRef":
```

```
                        Value = Value_elem.text

                        Key = Key_elem.text


                        keyList_results.append((Key, Value))


        # add structured data
        ScheduledStopPoint_list.append({
            "id_service_netex": ScheduledStopPoint_id,
            "StopPlaceRef":keyList_results
            })


    # return the list
    return ScheduledStopPoint_list
```

4. PassengerStopAssignment

```
# PassengerStopAssignment extraction

def PassengerStopAssignment_extraction_netex(xml_content, ns):
    PassengerStopAssignment_list = []  # Collect dictionaries here

    # Parse the XML content
    tree = ET.parse(io.BytesIO(xml_content))
    root = tree.getroot()


    for PassengerStopAssignment in
root.findall('.//netex:PassengerStopAssignment', ns):


        # reset all the variables in the case that no value was found
for some variables
        PassengerStopAssignment_id = ScheduledStopPointRef =
StopPlaceRef = QuayRef = None

        # extract the id
        PassengerStopAssignment_id =
PassengerStopAssignment.attrib.get('id')

        # extract scheduledstoppoint reference
        ScheduledStopPointRef_elem =
PassengerStopAssignment.find('netex:ScheduledStopPointRef', ns)

        if ScheduledStopPointRef_elem is not None:
            ScheduledStopPointRef =
ScheduledStopPointRef_elem.attrib.get("ref")
```

```
        StopPlaceRef_elem =
PassengerStopAssignment.find('netex:StopPlaceRef', ns)

        if StopPlaceRef_elem is not None:
            StopPlaceRef = StopPlaceRef_elem.attrib.get("ref")

        QuayRef_elem = PassengerStopAssignment.find('netex:QuayRef',
ns)
        if QuayRef_elem is not None:

            QuayRef = QuayRef_elem.attrib.get("ref")



        # add structured data
        PassengerStopAssignment_list.append({
            "PassengerStopAssignment_id": PassengerStopAssignment_id,
            "ScheduledStopPointRef":ScheduledStopPointRef,
            "StopPlaceRef": StopPlaceRef,
            "QuayRef":QuayRef
            })


    # return the list
    return PassengerStopAssignment_list
```

Testing and extraction: Selection of Extraction Methode

```
def testing_extraction_netex(xml_files, ns):

    # here are the tags of our interest
    tags_of_interest = {
        "ServiceJourney": ".//netex:ServiceJourney",
        "ServiceJourneyPattern": ".//netex:ServiceJourneyPattern",
        "JourneyPattern": ".//netex:JourneyPattern",
        "ScheduledStopPoint": ".//netex:ScheduledStopPoint",
        "PassengerStopAssignment":".//netex:PassengerStopAssignment"
    }

    # store results in list
    ScheduledStopPoint_all_list = []
    ServiceJourney_all_list = []
    ServiceJourneyPattern_all_list = []
    PassengerStopAssignment_all_list = []


    # we iterate through the list of xml files
    for xml_name, xml_content in xml_files:
        tree = ET.parse(io.BytesIO(xml_content))
```

```
        root = tree.getroot()

        # then we iterate through the each single xml file
        # e.g.: tag name = ServiceJourney, path =
//netex:ServiceJourneyPattern"
        for tag_name, xpath in tags_of_interest.items():
            #print("current loop:", tag_name)
            # we use the findall function to test if on of the three
paths does found a match
            found_elements = root.findall(xpath, ns)

            if found_elements:
                #print(f"Found {len(found_elements)} {tag_name}
elements")

                # optional: explore only the first one to check
structure
                #explore_element(found_elements[0])


                # run the extraction functions (collect all)
                if tag_name == "ScheduledStopPoint":
                    ScheduledStopPoint_list =
ScheduledStopPoint_extraction_netex(xml_content, ns)

                    # after each extraction we will add the results to
list
                    # Otherwise would the loop cause error because we
would try to add a result to a list which was not defined in the
current loop
                    # because it would be in the next one for example

ScheduledStopPoint_all_list.append(ScheduledStopPoint_list)

                if tag_name == "ServiceJourney":
                    ServiceJourney_list =
ServiceJourney_extraction_netex(xml_content, ns)

ServiceJourney_all_list.append(ServiceJourney_list)

                if tag_name == "ServiceJourneyPattern" or tag_name ==
"JourneyPattern":
                    ServiceJourneyPattern_list =
ServiceJourneyPattern_extraction_netex(xml_content, ns, tag_name)

ServiceJourneyPattern_all_list.append(ServiceJourneyPattern_list)

                if tag_name == "PassengerStopAssignment":
                    PassengerStopAssignment_list =
PassengerStopAssignment_extraction_netex(xml_content, ns)
```

```
PassengerStopAssignment_all_list.append(PassengerStopAssignment_list)



    # flaten each list in order to create a df

    ScheduledStopPoint_all_flat_list = [item for sublist in
ScheduledStopPoint_all_list for item in sublist]
    ServiceJourney_all_flat_list = [item for sublist in
ServiceJourney_all_list for item in sublist]
    ServiceJourneyPattern_all_flat_list = [item for sublist in
ServiceJourneyPattern_all_list for item in sublist]
    PassengerStopAssignment_all_flat_list = [item for sublist in
PassengerStopAssignment_all_list for item in sublist]


    ScheduledStopPoint_df =
pd.DataFrame(ScheduledStopPoint_all_flat_list)
    ServiceJourney_df = pd.DataFrame(ServiceJourney_all_flat_list)
    ServiceJourneyPattern_df =
pd.DataFrame(ServiceJourneyPattern_all_flat_list)
    PassengerStopAssignment_df =
pd.DataFrame(PassengerStopAssignment_all_flat_list)


    return ScheduledStopPoint_df, ServiceJourney_df,
ServiceJourneyPattern_df, PassengerStopAssignment_df
```

Merging Process of ServiceJourney and StopPlace

```
import ast

def
connecting_StopPointInJourneyPattern_to_StopPlace(ServiceJourney_df,
ServiceJourneyPattern_df, PassengerStopAssignment_df,
station_df ,country_name):


    # clean all values which does not have TimetabledPassingTime_netex
because this might causes error and takes process time
    ServiceJourney_df =
ServiceJourney_df[ServiceJourney_df["TimetabledPassingTime_netex"].not
na()]

    # sometimes we can filter the dataframe by its mode of transport
(column: journey_type_netex)
    # this is only the case if not all values in journey_type_netex
are None
    # Only then we will filter
```

```
    all_none = ServiceJourney_df["journey_type_netex"].isna().all()

    if all_none == False:  # True if all values are NaN/None, False
otherwise

        ServiceJourney_df =
ServiceJourney_df[ServiceJourney_df["journey_type_netex"] == "rail"]


    """
    Following link exists between the ServiceJourney and StopPlace:
    ServiceJourney_df (ScheduledStopPointinJourney_ref) ->
ServiceJourneyPattern_df(ScheduledStopPoint_ref) ->
PassengerStopAssignment_df (StopPlaces_ref)
    this function will connect each StopPointInJourneyPattern_id with
an matching StopPlaceRef
    """



    if country_name != "norway":

        # first we create a dictionary where we store the connection
from StopPointInJourneyPattern to ScheduledStopPoint
        stop_point_to_scheduled = {}
        for index, row in ServiceJourneyPattern_df.iterrows():

            for stop_points in row["StopPointInJourneyPattern"]:
                # add the StopPointInJourneyPattern_id as the key and
the ScheduledStopPointRef as the value in the dictionary

stop_point_to_scheduled[stop_points.get("StopPointInJourneyPattern_id"
)] = stop_points.get("ScheduledStopPointRef")

        # second we filter the connection from ScheduledStopPoint to
StopPlace
        scheduled_to_stop_place =
PassengerStopAssignment_df.groupby("ScheduledStopPointRef")
["StopPlaceRef"].first().to_dict()

        # third we will iterate through each row of the
ServiceJourney_df in order to iterate over each nested list of
dictionaries or dict
        for index, row in ServiceJourney_df.iterrows():

            for stop_entry in row.TimetabledPassingTime_netex:

                # now we iterate through the stop_entry list or
dictionary
                # each trip can consist multiple stops on a journey or
```

```
sometimes only one

                ScheduledStopPointRef =
stop_point_to_scheduled.get(stop_entry["Stop"], None)

                # if the entry was found in the
stop_point_to_scheduled dictionary and contains a value
                if ScheduledStopPointRef is not None:
                    StopPlaceRef =
scheduled_to_stop_place.get(ScheduledStopPointRef)

                if ScheduledStopPointRef is None:
                    # if there is no value, None will be added
                    StopPlaceRef = None

                stop_entry["StopPlaceRef"] = StopPlaceRef  # modifies
in place


    if country_name == "norway":

        # first we prepare the station_df: defining columns and
filtering unecessary data
        station_df = station_df[["id_netex","Quay_ids_netex"]]
        station_df = station_df[station_df["Quay_ids_netex"].notna()]


        # flaten the embedded Quay list to single entries
        station_df["Quay_ids_netex"] =
station_df["Quay_ids_netex"].apply(ast.literal_eval)  # safely convert
string → list
        station_df = station_df.explode("Quay_ids_netex",
ignore_index=True)

        # PassengerStopAssignment_df -> define the dataframe as well
        PassengerStopAssignment_df =
PassengerStopAssignment_df[["ScheduledStopPointRef","QuayRef"]]


        # merge the station_df and the PassengerStopAssignment_df by
their shared column: QuayRef/ Quay
        merge_1 = pd.merge(PassengerStopAssignment_df, station_df,
                           left_on="QuayRef",
                           right_on="Quay_ids_netex",
                           how="inner")


        # define the ServiceJourneyPattern_df_test
        ServiceJourneyPattern_df =
ServiceJourneyPattern_df[["StopPointInJourneyPattern"]]
```

```
        # create a dataframe from all the StopPointInJourneyPattern id
and matching ScheduledStopPointRef within the ServiceJourneyPattern
        records = []

        for index, row in ServiceJourneyPattern_df.iterrows():
            for sp in row["StopPointInJourneyPattern"]:
                records.append({
                    "StopPointInJourneyPattern_id":
sp.get("StopPointInJourneyPattern_id"),
                    "ScheduledStopPointRef":
sp.get("ScheduledStopPointRef")
                })

        df_flat = pd.DataFrame(records)

        ServiceJourneyPattern_df =
df_flat.drop_duplicates(subset=['StopPointInJourneyPattern_id',
'ScheduledStopPointRef'])


        # merge now the ServiceJourneyPattern with the merged
station_df and PassangerStopAssignment (merge_1)
        merge_2 = pd.merge(merge_1, ServiceJourneyPattern_df,
                           on="ScheduledStopPointRef",
                           how="inner")



        # finally: we will now iterate through each nested list of
dictionaries and check if there is a StopPlaceRef for each
StopPointinJourneyPattern
        for index, row in ServiceJourney_df.iterrows():

            for Stop_list in row["TimetabledPassingTime_netex"]:


                # if the current StopPointInJourneyPattern id in the
merge_2 where the connection from StopPointInJourneyPattern to
StopPlace is safed
                if (merge_2["StopPointInJourneyPattern_id"] ==
Stop_list.get("Stop")).any() == True:
                    # the relationship from StopPlace to
StopPointInJourneyPattern_id is 1:n
                    Stop_list["StopPlaceRef"] =
merge_2[merge_2["StopPointInJourneyPattern_id"] ==
Stop_list.get("Stop")]["id_netex"].iloc[0]

                else:
```

```
                    Stop_list["StopPlaceRef"] = None


    return ServiceJourney_df
```

Start Netex Journey extraction

```
def start_journey_extration_netex(netex_path, ns, country_name,
station_df):

    # load the netex files
    xml_files = load_netex(netex_path)
    print("netex load completed")
    # start the extraction from four different roots of each netex xml
tree
    ScheduledStopPoint_df, ServiceJourney_df,
ServiceJourneyPattern_df, PassengerStopAssignment_df =
testing_extraction_netex(xml_files, ns)
    print("netex extraction completed")
    #print("Length of ServiceJourney_df:",len(ServiceJourney_df))

    # link each topPointInJourneyPattern_id a each StopPlaceRef
    journey_netex_df =
connecting_StopPointInJourneyPattern_to_StopPlace(ServiceJourney_df,
ServiceJourneyPattern_df, PassengerStopAssignment_df,
station_df ,country_name)


    # saving the dataframe in order to use for the second part
2_compare
    journey_netex_df.to_json(country_name + "_" +
'journey_netex_df.json',
                             orient = "records",  # row-wise list
of dicts
                             lines=False, # single JSON array
                             force_ascii=False) # # keep special
characters readable

    print("netex file stored")

    return journey_netex_df
```

Start Extraction of trips/journey for netex and gtfs

```
country_name = "luxembourg"
```

GTFS: Trip Extraction

```
gtfs_zip_path = "./data_sets/"+ country_name +"_gtfs.zip"

station_df = pd.read_csv(country_name +"_station_merged_df.csv",
index_col = 0)

trip_merged_gtfs_df =
start_trip_extraction_gtfs(country_name,gtfs_zip_path,station_df)
```

```
gtfs loading completed
gtfs extraction completed
```

Netex: Journey Extraction

```
# Path to your outer zip file

if country_name != "austria":

    netex_path = "./data_sets/" + country_name + "_netex.zip"

elif country_name == "austria":

    netex_path = "./data_sets/" + country_name + "_journey_netex.zip"


station_df = pd.read_csv(country_name +"_station_merged_df.csv",
index_col = 0)


# identify the structure of the NeTEx data
ns = {'netex': 'http://www.netex.org.uk/netex'}

# start the whole process might take 7 - 10 minutes
journey_netex_df = start_journey_extration_netex(netex_path, ns,
country_name, station_df)
```

```
netex load completed
netex extraction completed
netex file stored
```

6.3 Notebook: Comparing

Compare: GTFS and Netex

- How much of the data is the same, which is not and how much is missing?

Compare Stations: Modul 1

```
# install necessary packages

#!pip install upsetplot
#!pip install matplotlib-venn
```

Merge the three sources together: Netex, GTFS and Wiki Data

```
def merge_gtfs_netex_wiki(station_df, wiki_data_df, country_name):


    # if no merging with the wiki data set is possible then the
station_df is None

    merged_wiki_df = pd.DataFrame()

    """
    Austria: IBNR
    """

    if country_name == "austria":

        # clear the data sets before merging them (drop duplicates and
non values)
        # ordered_wiki_data_df =
ordered_wiki_data_df.drop_duplicates(subset=["lat_wiki","lon_wiki"])

        ordered_station_df =
station_df.drop_duplicates(subset=["EVA_Nr_netex"])
        # Drop duplicates and NaNs safely without inplace
modifications
        ordered_wiki_data_df =
(wiki_data_df.drop_duplicates(subset=["lat_wiki",
"lon_wiki"]).dropna(subset=["lat_wiki","lon_wiki","IBNR_wiki"]))


        # Step 2: Merge only on rows with valid IBNR/EVA numbers
        merged_wiki_df = pd.merge(
            ordered_station_df,
            ordered_wiki_data_df,
            left_on="EVA_Nr_netex",
            right_on="IBNR_wiki",
```

```
            how="outer"
        )

        # drop all duplicates after the merge
        merged_wiki_df =
merged_wiki_df.drop_duplicates(subset=['EVA_Nr_netex',"IBNR_wiki"])

    """
    Luxembourg/ Norway: UIC Code
    """

    # check if we have the austrian country and whether the netex file
contains uic codes
    # If all colums are
    if country_name != "austria" and
station_df["UIC_Code_netex"].isna().all() == False:


        # clear the data sets before merging them (drop duplicates and
non values)
        # ordered_wiki_data_df =
ordered_wiki_data_df.drop_duplicates(subset=["lat_wiki","lon_wiki"])

        ordered_station_df =
station_df.drop_duplicates(subset=["UIC_Code_netex"])
        # Drop duplicates and NaNs safely without inplace
modifications
        ordered_wiki_data_df =
(wiki_data_df.drop_duplicates(subset=["lat_wiki",
"lon_wiki"]).dropna(subset=["lat_wiki","lon_wiki","UIC_wiki"]))


        # Step 2: Merge only on rows with valid IBNR/EVA numbers
        merged_wiki_df = pd.merge(
            ordered_station_df,
            ordered_wiki_data_df,
            left_on="UIC_Code_netex",
            right_on="UIC_wiki",
            how="outer"
        )
        # drop all duplicates after the merge
        merged_wiki_df =
merged_wiki_df.drop_duplicates(subset=['UIC_Code_netex',"UIC_wiki"])

    #if country_name == "luxembourg":



    return (merged_wiki_df)
```

Visualize the difference between gtfs and netex and netex and wiki data

Why can we not compare the wiki data with the gtfs data?

Austria, Luxembourg and Norway

- labeling issues: gtfs does not include a IBNR or UIC any other referenced ID for their stations, therefore it is not possible to merge them together with the wiki data set because of the missing key
- Netex contains an IBNR nummer (EVA_Nr_netex) which is really commen in wiki data for austria train station therefore we can compare them

```
import matplotlib.pyplot as plt
from matplotlib_venn import venn2


def venn_plot(merged_df, first_key, second_key, country_name,
actual_station_number, first_name, second_name):

    # IBNR_wiki and EVA_Nr_netex or gtfs and netex
    first_df = set(merged_df[first_key].dropna())
    second_df = set(merged_df[second_key].dropna())

    plt.figure(figsize=(7, 7))
    venn = venn2([first_df, second_df], set_labels=(first_key,
second_key))

    # Change colors: light red, light green, light blue

    # check if there is even a set which can be colored otherwise
error will appear
    for region, color in {'10': 'lightcoral', '01': 'lightgreen',
'11': 'lightblue'}.items():
        patch = venn.get_patch_by_id(region)
        if patch:  # only if this region exists
            patch.set_color(color)

    #venn.get_patch_by_id('10').set_color('lightcoral')   # left
circle only
    #venn.get_patch_by_id('01').set_color('lightgreen')   # right
circle only
    #venn.get_patch_by_id('11').set_color('lightblue')    #
intersection

    # Add title with two lines
    plt.title(
        f"Overlap between the shares of {country_name}'s Railway
Stations {first_key} and {second_key}.\n"
        f"Actual number of stations: {actual_station_number}"
```

```
    )

    # Save plot
    filename =
f"{country_name}_stations_{first_name}_{second_name}.png"
    plt.tight_layout()
    plt.savefig(filename, dpi=300, bbox_inches="tight")

    # Show plot
    plt.show()
    plt.close()

    # create dataframe for non matched values
    # Rows with NeTEx ID only
    wiki_only = merged_df[merged_df[first_key].notna() &
merged_df[second_key].isna()]

    # Rows with GTFS ID only
    netex_only = merged_df[merged_df[second_key].notna() &
merged_df[first_key].isna()]



    # Combine them
    not_shared_df = pd.concat([wiki_only, netex_only])

    return not_shared_df
```

Compare the stations

```
def compare_station_gtfs_netex_wiki(station_df, wiki_data_df,
                                    first_column_a, second_column_a,
                                    first_column_b, second_column_b,
                                    country_name,
actual_station_number):


    # merge the three sources together
    # if it was succesfull: merged_wiki_df is not empty
    # otherewise it is
    merged_wiki_df = merge_gtfs_netex_wiki(station_df,
wiki_data_df,country_name)


    # if the merged wiki is empty, we have no commen key between netex
and wiki
    if merged_wiki_df.empty == False:
```

```
        """
        Plot: Venn
        """
        # plot the difference via venn plot

        # difference wiki data and netex: a
        first_name_a = country_name + "_" + first_column_a
        second_name_a = country_name + "_" + second_column_a

        not_shared_wiki_netex_df = venn_plot(merged_wiki_df,
                                             first_column_a,
second_column_a, # column to compare (first wiki, second netex)
                                             country_name,
                                             actual_station_number, #
actual number

first_name_a,second_name_a)


        # saving the dataframe in order to use for the third part
3_data_quality
        merged_wiki_df.to_csv(country_name + "_station_+_wiki_df.csv")

        print(" ")


    # difference between netex and gtfs: b
    first_name_b = country_name + "_" + first_column_b
    second_name_b = country_name + "_" + second_column_b


    not_shared_netex_gtfs_df = venn_plot(station_df,
                                         first_column_b,
second_column_b,
                                         country_name, # column to
compare (netex and gtfs)
                                         actual_station_number, #
actual number of stations
                                         first_name_b,second_name_b)



    # we can only return a list of not matching values if we can merge
the wiki data with the netex or gtfs data
    if merged_wiki_df.empty == False:

        return (merged_wiki_df, not_shared_wiki_netex_df,
not_shared_netex_gtfs_df)
```

```
    else:
        not_shared_wiki_netex_df = None
        return (merged_wiki_df, not_shared_wiki_netex_df,
not_shared_netex_gtfs_df)
```

Not shared station mapping

```
import folium

def mapping_not_shared_stations(not_shared_netex_gtfs_df):
    # center map around the mean of coordinates
    center_lat = not_shared_netex_gtfs_df[["lat_gtfs",
"lat_netex"]].stack().mean()
    center_lon = not_shared_netex_gtfs_df[["lon_gtfs",
"lon_netex"]].stack().mean()

    # create base map
    m = folium.Map(location=[center_lat, center_lon], zoom_start=6)

    # add GTFS stops (blue)
    for _, row in not_shared_netex_gtfs_df.iterrows():
        if not pd.isna(row["lat_gtfs"]) and not
pd.isna(row["lon_gtfs"]):
            folium.CircleMarker(
                location=[row["lat_gtfs"], row["lon_gtfs"]],
                radius=5,
                color="blue",
                fill=True,
                fill_color="blue",
                popup=f"GTFS: {row['name_gtfs']} ({row['id_gtfs']})"
            ).add_to(m)

    # add NeTEx stops (red)
    for _, row in not_shared_netex_gtfs_df.iterrows():
        if not pd.isna(row["lat_netex"]) and not
pd.isna(row["lon_netex"]):
            folium.CircleMarker(
                location=[row["lat_netex"], row["lon_netex"]],
                radius=5,
                color="red",
                fill=True,
                fill_color="red",
                popup=f"NeTEx: {row['name_netex']}
({row['id_netex']})"
            ).add_to(m)

    # save map
    m.save(country_name+"not_shared_map.html")
```

Start the Station comparison

```
country_name = "austria"

# norway 400
# luxembourg 70
# austria 1031

actual_station_number = 1031

# load the merged gtfs and netex data set from 1_Extraction
import pandas as pd

station_df = pd.read_csv(country_name
+"_station_merged_df.csv",index_col = 0, low_memory=False)

# load the wiki data for the matching country from
1.1_Wiki_Data_Querry
wiki_data_df = pd.read_csv(country_name+"_wiki_data_df.csv",index_col
= 0)


# IBNR
if country_name == "austria":

    merged_wiki_df, not_shared_wiki_netex_df, not_shared_netex_gtfs_df
= compare_station_gtfs_netex_wiki(station_df, wiki_data_df,

"IBNR_wiki","EVA_Nr_netex",

"id_netex", "id_gtfs",

country_name, actual_station_number)
# IIC Code
if country_name != "austria":
    merged_wiki_df, not_shared_wiki_netex_df, not_shared_netex_gtfs_df
= compare_station_gtfs_netex_wiki(station_df, wiki_data_df,

"UIC_wiki","UIC_Code_netex",

"id_netex", "id_gtfs",

country_name, actual_station_number)
# map the not shared stations
mapping_not_shared_stations(not_shared_netex_gtfs_df)

not_shared_netex_gtfs_df[['id_gtfs',"name_gtfs",
'id_netex',"name_netex","lat_netex","lon_netex","lat_gtfs","lon_gtfs"]
]
```

```
not_shared_wiki_netex_df[['UIC_wiki',"name_wiki",
'UIC_Code_netex',"name_netex"]]
```

Module 2: Analyze the trips

Questions:

1. How many stops on a trip are assigned to a stop id?
    a. Is the stop assigned with an id from the matching data set?

```
import pandas as pd


# How many stops on a trip are assigned to a stop id?
def facts_about_gtfs_trips(trips_gtfs_df, station_df, country_name):

    amount_of_stops_per_trip_list = []
    amount_of_stops_with_ids_per_trip_list = []

    # iterate through the rows
    for index, row in trips_gtfs_df.iterrows():
        # we select only the stops_on_trip column
        stop_list = row["stops_on_trip"]

        amount_of_stops_per_trip = 0
        amount_of_stops_with_ids_per_trip = 0

        # iterate through the values of this column "stops_on_trip"
        for stop in stop_list:


            stop_id = stop.get("stop_id_gtfs")
            #print(stop_id)
            # each trip is iterated

            # Only if the have an ID, we test if the id is also into
the station df
            if stop_id != None:

                """
                Austria
                Each Country has its own ids:
                ID at:43:1322:0:1
                StopPlace ID: 43:1322
                """
                # if the len of the filtered station_df is greater
than 0: we have a match
                if station_df[station_df["id_gtfs"] ==
```

```
(":".join(str(stop_id).split(":")[:3]))].empty == False and
country_name == "austria":
                    amount_of_stops_per_trip = 1 +
amount_of_stops_per_trip
                    amount_of_stops_with_ids_per_trip = 1 +
amount_of_stops_with_ids_per_trip
                    #print("find:",amount_of_stops_per_trip,
amount_of_stops_with_ids_per_trip)

                    # if the len of the filtered station_df is 0: we
have we do not have a match and will add another stop
                if station_df[station_df["id_gtfs"] ==
(":".join(str(stop_id).split(":")[:3]))].empty == True and
country_name == "austria":
                    amount_of_stops_per_trip = 1 +
amount_of_stops_per_trip

                """
                Others: Luxembourg, Norway
                """

                # if the len of the filtered station_df is greater
than 0: we have a match
                if station_df[station_df["id_gtfs"] == stop_id].empty
== False and country_name != "austria":
                    amount_of_stops_per_trip = 1 +
amount_of_stops_per_trip
                    amount_of_stops_with_ids_per_trip = 1 +
amount_of_stops_with_ids_per_trip
                    #print("find:",amount_of_stops_per_trip,
amount_of_stops_with_ids_per_trip)

                            # if the len of the filtered station_df is
0: we have we do not have a match and will add another stop
                if station_df[station_df["id_gtfs"] == stop_id].empty
== True and country_name != "austria":
                    amount_of_stops_per_trip = 1 +
amount_of_stops_per_trip

        # if the second loop is over, we will add the numbers to the
list
        amount_of_stops_per_trip_list.append(amount_of_stops_per_trip)

amount_of_stops_with_ids_per_trip_list.append(amount_of_stops_with_ids
_per_trip)

        #break

    # if the loop is finsihed, we will add the list as a column to the
dataframe: trips_gtfs_df
```

```
    trips_gtfs_df["stops_per_trip"] = amount_of_stops_per_trip_list
    trips_gtfs_df["stops_per_trip_with_id"] =
amount_of_stops_with_ids_per_trip_list

    return trips_gtfs_df

# we use the same function as we used in the extraction modul:
1_Extraction

import re

def extract_digit_number(id_str, length):

    if isinstance(id_str, int):
        id_str = str(id_str)

    pattern = rf'(\d{{{length}}})'  # Removed \b to allow matches next
to colons, underscores, etc.
    match = re.search(pattern, str(id_str))

    return int(match.group(1)) if match is not None else None


import pandas as pd
def facts_about_netex_journey(journey_netex_df,station_df,
country_name):

    """
    Luxembourg Label issue: as already known for the extraction the
netex data StopPlace id appears as "DE::StopPlace:200101007_CdT::"
    For the extraction and merging process with the gtfs data I
reduced this id by extracted the number of the string like this
200101007

    Therefore we need to do the same for her as well as we did in the
ectraction part

    We assume here that the ids will always have the same length
    """

    if country_name == "luxembourg":
        # we will filter the dataset for columns which have a netex id
and choose the first entry ofter position to calculate the length of
the first entry
        length =
len(str(station_df[station_df["id_netex"].notna()].iloc[0]
["id_netex"]).replace(".0",""))

    # How many stops on a trip are assigned to a stop id?
    #def facts_about_netex_journeys(journey_netex_df):
```

```
    amount_of_stops_per_trip_list = []
    amount_of_stops_with_ids_per_trip_list = []

    # iterate through the rows
    for index, row in journey_netex_df.iterrows():
    # we select only the stops_on_trip column
        stop_list = row["TimetabledPassingTime_netex"]

        # reset each counter before each iteration of a trip
        amount_of_stops_per_trip = 0
        amount_of_stops_with_ids_per_trip = 0

        # iterate through the values of this column
        for stop in stop_list:

            StopPlaceRef = stop["StopPlaceRef"]

            # what kind of country Luxembourg as special case with
different stop ids
            if country_name == "luxembourg":
                StopPlaceRef = extract_digit_number(StopPlaceRef,
length)

            # the Stop on the trip has no stopplace reference
            if StopPlaceRef == None:
                amount_of_stops_per_trip = 1 +
amount_of_stops_per_trip

            # a StopPlaceRef exists
            if StopPlaceRef != None:

                """Luxembourg"""
                # if a StopPlaceRef contains in the station_df
                if station_df[station_df["id_netex"] ==
StopPlaceRef].empty == False and country_name == "luxembourg":
                    amount_of_stops_per_trip = 1 +
amount_of_stops_per_trip
                    amount_of_stops_with_ids_per_trip = 1 +
amount_of_stops_with_ids_per_trip

                # if the StopPlaceRef does not contain within the
station_df, we will add only a another stop
                if station_df[station_df["id_netex"] ==
StopPlaceRef].empty == True and country_name == "luxembourg":
                    amount_of_stops_per_trip = 1 +
amount_of_stops_per_trip

                """Others"""
```

```
                # if a StopPlaceRef contains in the station_df
                if station_df[station_df["id_netex"] ==
StopPlaceRef].empty == False and country_name != "luxembourg":
                    amount_of_stops_per_trip = 1 +
amount_of_stops_per_trip
                    amount_of_stops_with_ids_per_trip = 1 +
amount_of_stops_with_ids_per_trip

                # if the StopPlaceRef does not contain within the
station_df, we will add only a another stop
                if station_df[station_df["id_netex"] ==
StopPlaceRef].empty == True and country_name != "luxembourg":
                    amount_of_stops_per_trip = 1 +
amount_of_stops_per_trip

        # add the values per trip (per row) to the lists
        amount_of_stops_per_trip_list.append(amount_of_stops_per_trip)

amount_of_stops_with_ids_per_trip_list.append(amount_of_stops_with_ids
_per_trip)

        #break

    # if the loop is finsihed, we will add the list as a column to the
dataframe: trips_gtfs_df
    journey_netex_df["stops_per_trip"] = amount_of_stops_per_trip_list
    journey_netex_df["stops_per_trip_with_id"] =
amount_of_stops_with_ids_per_trip_list

    return journey_netex_df


import matplotlib.pyplot as plt

# stops_per_trip, stops_per_trip_with_id, id_service_netex
# journey_netex_df
def bar_plot_trips_with_id_or_without(dataset, trip_id, country_name,
gtfs_or_netex):



    # stops_per_trip, stops_per_trip_with_id, id_service_netex
    # journey_netex_df

    x = range(len(dataset))  # positions for bars

    plt.figure(figsize=(24, 12))

    # Background bars (stops_per_trip)
    plt.bar(
```

```
        x,
        dataset["stops_per_trip"],
        width=0.8,
        color="lightgreen",
        label="Stops per trip"
    )

    # Foreground bars (stops_per_trip_with_id)
    plt.bar(
        x,
        dataset["stops_per_trip_with_id"],
        width=0.5,
        color="steelblue",
        label="Stops per Trip with ID",
        zorder=3
    )

    # X-axis labels (trip_id_gtfs)
    #plt.xticks(x, journey_netex_df[trip_id], rotation=90)

    plt.ylabel("Number of stops", fontsize=24)
    plt.xlabel("Number of Trips", fontsize=24)
    plt.title(gtfs_or_netex + ":" + " Stops per trip vs. stops with ID
within the "+ country_name + " data set", fontsize=24)
    plt.legend(loc="upper right", fontsize=24)
    plt.tight_layout()

    filename =
f"{country_name}_{gtfs_or_netex}_ID_comparison_plot.png"
    plt.savefig(filename, dpi=300, bbox_inches="tight")

    plt.show()
    plt.close()  # closes the figure so memory isn't clogged
```

Start the Trip Analysis

```
def start_trip_analysis(journey_netex_df, trips_gtfs_df, station_df,
country_name):

    # this might take a while ca. 10 min

    # caculate the stops
    journey_netex_df = facts_about_netex_journey(journey_netex_df,
station_df,country_name)

    trips_gtfs_df = facts_about_gtfs_trips(trips_gtfs_df,
station_df,country_name)

    print("Stops count completed")
```

```
    # plot the results

    bar_plot_trips_with_id_or_without(trips_gtfs_df, "trip_id_gtfs",
country_name,"GTFS")

    bar_plot_trips_with_id_or_without(journey_netex_df,
"id_service_netex", country_name,"NeTex")


    return journey_netex_df, trips_gtfs_df

import pandas as pd
import os


country_name = "norway"
# loading the two data sets from the extraction
journey_netex_df = pd.read_json(country_name
+"_journey_netex_df.json", orient = "records")

# load the wiki data for the matching country from
1.1_Wiki_Data_Querry
trips_gtfs_df = pd.read_json(country_name +"_trip_gtfs_df.json",orient
= "records")

# load the reference data frame for the stations
station_df = pd.read_csv(country_name
+"_station_merged_df.csv",index_col = 0)

journey_netex_df, trips_gtfs_df =
start_trip_analysis(journey_netex_df, trips_gtfs_df, station_df,
country_name)
```

How many stations of the trips are missing?

```
print("Trips in total:",len(trips_gtfs_df))
print("Trips which were not successfully
refered:",len(trips_gtfs_df[trips_gtfs_df["stops_per_trip"] !=
trips_gtfs_df["stops_per_trip_with_id"]]))

not_succesfully_refered_trips =
trips_gtfs_df[trips_gtfs_df["stops_per_trip"] !=
trips_gtfs_df["stops_per_trip_with_id"]]

print("Avereage missing stops per
trip",not_succesfully_refered_trips["stops_per_trip_with_id"].mean(),"
from",len(not_succesfully_refered_trips),"trips
and",not_succesfully_refered_trips["stops_per_trip"].mean(),"stops on
```

```
average")
print("Median of Missing stops per
trip",not_succesfully_refered_trips["stops_per_trip_with_id"].median()
,"from",len(not_succesfully_refered_trips),"trips
and",not_succesfully_refered_trips["stops_per_trip"].median(),"stops
on median")
```

```
Trips in total: 47439
Trips which were not successfully refered: 0
Avereage missing stops per trip nan from 0 trips and nan stops on
average
Median of Missing stops per trip nan from 0 trips and nan stops on
median
```

```
print("Journeys in total:",len(journey_netex_df))
print("Journeys which were not successfully
refered:",len(journey_netex_df[journey_netex_df["stops_per_trip"] !=
journey_netex_df["stops_per_trip_with_id"]]))

not_succesfully_refered_journey =
journey_netex_df[journey_netex_df["stops_per_trip"] !=
journey_netex_df["stops_per_trip_with_id"]]

print("Avereage missing stops per
Journey",not_succesfully_refered_journey["stops_per_trip_with_id"].mea
n(),"from",len(not_succesfully_refered_journey),"Journey
and",not_succesfully_refered_journey["stops_per_trip"].mean(),"stops
on average")
print("Median of Missing stops per
Journey",not_succesfully_refered_journey["stops_per_trip_with_id"].med
ian(),"from",len(not_succesfully_refered_journey),"Journey
and",not_succesfully_refered_journey["stops_per_trip"].median(),"stops
on median")
```

```
Journeys in total: 26852
Journeys which were not successfully refered: 0
Avereage missing stops per Journey nan from 0 Journey and nan stops on
average
Median of Missing stops per Journey nan from 0 Journey and nan stops
on median
```

Results: Overview

```
# load the wiki data for the matching country from
1.1_Wiki_Data_Querry
wiki_data_df = pd.read_csv("luxembourg"+"_wiki_data_df.csv",index_col
= 0)
```

```
print(len(wiki_data_df[wiki_data_df["IBNR_wiki"].notna()]),
      len(wiki_data_df[wiki_data_df["UIC_wiki"].notna()]),
      len(wiki_data_df[wiki_data_df["IFOPT_wiki"].notna()]))
```

```
37 50 0
```

```
results = {"Applications": ["Extraction: Stops","Extraction: Trips",
"Wiki data", "Compare: stations", "Compare: Info trips","Data Quality:
Distance","Wiki IBNR entries","Wiki UIC entries","Wiki IFOPT
entries"],
           "Austria": [True, True, True, True, True, True,
1210,142,0],
           "Norway": [True, True, True, True, True, True, 151, 420,
0],
           "Luxembourg": [True, True, True, "Only gtfs and netex",
True, False,37 ,50 ,0],

          }

results_df = pd.DataFrame(results)

results_df
```

6.4 Notebook: Data Quality

Data Quality:

- Are the coordinates provide by netex or gtfs correct compared to a third source

```
#!pip install -U kaleido
#!pip install pyarrow
#!pip install plotly
```

Calculate distance

```
# calculate the distance between both coordinates by using the
Haversine formula
# https://www.movable-type.co.uk/scripts/latlong.html#:~:text=This
%20uses%20the%20'haversine'%20formula,where:

from math import sin, cos, sqrt, atan2, radians

def distance_between_coordinates(lat1, lon1, lat2, lon2):
    R = 6371  # Radius of Earth in kilometers

    lat1_rad = radians(lat1)
    lon1_rad = radians(lon1)
    lat2_rad = radians(lat2)
    lon2_rad = radians(lon2)

    dlon = lon2_rad - lon1_rad
    dlat = lat2_rad - lat1_rad

    a = sin(dlat / 2)**2 + cos(lat1_rad) * cos(lat2_rad) * sin(dlon /
2)**2
    c = 2 * atan2(sqrt(a), sqrt(1 - a))

    distance = R * c
    return distance
```

Plot the distance from gtfs and netex and Wiki

```
import numpy as np
import matplotlib.pyplot as plt

def distance_wiki_netex_gtfs_barchart(output_df, station_name,
country_name):
    x = np.arange(len(output_df[station_name]))  # numeric x-axis
positions
    width = 0.4  # width of each bar

    plt.figure(figsize=(32, 16))
```

```
    # Bars for Netex and GTFS

    # if the gtfs columns does contain non values -> then we only plot
the netex column
    if output_df["distancen_between_gtfs_to"].isna().all() == True:
        plt.bar(x - width/2, output_df["distancen_between_netex_to"],
width, label="Netex", alpha=0.7)

    # the gtfs columns does not contain non values
    if output_df["distancen_between_gtfs_to"].isna().all() == False:
        plt.bar(x + width/2, output_df["distancen_between_gtfs_to"],
width, label="GTFS", alpha=0.7)
        plt.bar(x - width/2, output_df["distancen_between_netex_to"],
width, label="Netex", alpha=0.7)


    plt.title("Distance Comparison per Station between Wiki Data
Station: "+ country_name, fontsize=40)
    plt.xlabel("Stations",fontsize=40)
    plt.ylabel("Distance in meters to the wiki coordinates",
fontsize=40)
    plt.legend(fontsize=40)

    # Hide x labels (too many stations)
    plt.xticks([], [])

    plt.tight_layout()
    plt.grid(axis="y", linestyle="--", alpha=0.7)
    plt.savefig(country_name +"_station_distances_barchart.png",
dpi=300)
    plt.show()
    plt.close()
```

Plot the distance from gtfs and netex

```
import numpy as np
import matplotlib.pyplot as plt

def distance_gtfs_netex(output_df, station_name, country_name):
    x = np.arange(len(output_df[station_name]))  # numeric x-axis
positions
    width = 0.4  # width of each bar

    plt.figure(figsize=(32, 16))

    # Bars for Netex and GTFS
```

```
    plt.bar(x - width/2, output_df["distancen_between_netex_to_gtfs"],
width, label="Distance GTFS and Netex", alpha=0.7)

    plt.title("Distance Comparison per Station between NeTex and GTFS:
"+ country_name, fontsize=40)
    plt.xlabel("Stations",fontsize=40)
    plt.ylabel("Distance in meters", fontsize=40)
    plt.legend(fontsize=40)

    # Hide x labels (too many stations)
    plt.xticks([], [])

    plt.tight_layout()
    plt.grid(axis="y", linestyle="--", alpha=0.7)
    plt.savefig(country_name
+"_station_distances_gtfs_netex_barchart.png", dpi=300)
    plt.show()
    plt.close()
```

Calculate the distancen between GTFS and Netex and Wiki

```
import matplotlib.pyplot as plt
import plotly.express as px


def distance_calculator(input_df, lat_name_third_source,
lon_name_third_source, station_name, country_name):


    # create a list to store the results
    distance_list_gtfs = []
    distance_list_netex = []
    distance_list_netex_gtfs = []

    # iterate through the merged dataframe to calculate each distance
between the two coordinates
    for index, row in input_df.iterrows():

        # calculate the distance between each coordinate pair
        distance_in_meter_gtfs =
distance_between_coordinates(row["lat_gtfs"], row["lon_gtfs"],
row[lat_name_third_source], row[lon_name_third_source]) *1000
        distance_in_meter_netex =
distance_between_coordinates(row["lat_netex"], row["lon_netex"],
row[lat_name_third_source], row[lon_name_third_source]) *1000
        distance_in_meter_gtfs_netex =
distance_between_coordinates(row["lat_netex"], row["lon_netex"],
row["lat_gtfs"], row["lon_gtfs"]) *1000
```

```
        # add values to list
        distance_list_gtfs.append(distance_in_meter_gtfs)
        distance_list_netex.append(distance_in_meter_netex)
        distance_list_netex_gtfs.append(distance_in_meter_gtfs_netex)

    # create a new column in a new dataframe input
    input_df.loc[:, "distancen_between_gtfs_to"] = distance_list_gtfs
    input_df.loc[:, "distancen_between_netex_to"] =
distance_list_netex
    input_df.loc[:, "distancen_between_netex_to_gtfs"] =
distance_list_netex_gtfs

    # filter all NaN values for the gtfs OR the netex
    # why do we use "or" instead of "and": because if we would use an
"and" we would filter all values because both conditions must be true
    output_df = input_df[input_df["distancen_between_gtfs_to"].notna()
| input_df["distancen_between_netex_to"].notna()]


    print("number of matches between netex, gtfs and third source by
their coordinates:", len(output_df))

    # print the result as an interactive plot
    df_melted = output_df.melt(id_vars= station_name,
                        value_vars=['distancen_between_netex_to',
'distancen_between_gtfs_to'],
                        var_name='Distance Type', value_name='Distance
in meter')

    fig = px.line(df_melted, x= station_name, y='Distance in meter',
color='Distance Type',
                  title='Distance Comparison per Station')

    fig.update_layout(xaxis_tickangle=1000, xaxis_tickfont_size=9)
    fig.show()


    # also create a bar chart
    distance_wiki_netex_gtfs_barchart(output_df, station_name,
country_name)

    # create a plot for the distance between netex and gtfs
    distance_gtfs_netex(output_df, station_name, country_name)

    # calculate the mean difference between the stations
    mean_netex = output_df["distancen_between_netex_to"].mean()
    mean_gtfs = output_df["distancen_between_gtfs_to"].mean()

    print("avearage distance netex to third source:",mean_netex)
```

```
    print("avearage distance gtfs to third source:",mean_gtfs)



    # return the df back
    return (output_df, mean_netex, mean_gtfs)
```

Mapping the stations

```
import folium

# map the stations
def mapping_far_stations(stations_over_400):

    # center map around the mean of coordinates
    center_lat = stations_over_400[["lat_gtfs",
"lat_netex","lat_wiki"]].stack().mean()
    center_lon = stations_over_400[["lon_gtfs",
"lon_netex","lon_wiki"]].stack().mean()

    # create base map
    m = folium.Map(location=[center_lat, center_lon], zoom_start=6)

    # add GTFS stops (blue)
    for _, row in stations_over_400.iterrows():
        if not pd.isna(row["lat_gtfs"]) and not
pd.isna(row["lon_gtfs"]):
            folium.CircleMarker(
                location=[row["lat_gtfs"], row["lon_gtfs"]],
                radius=5,
                color="blue",
                fill=True,
                fill_color="blue",
                popup=f"GTFS: {row['name_gtfs']} ({row['id_gtfs']})"
            ).add_to(m)

    # add NeTEx stops (red)
    for _, row in stations_over_400.iterrows():
        if not pd.isna(row["lat_netex"]) and not
pd.isna(row["lon_netex"]):
            folium.CircleMarker(
                location=[row["lat_netex"], row["lon_netex"]],
                radius=5,
                color="red",
                fill=True,
                fill_color="red",
                popup=f"NeTEx: {row['name_netex']}
({row['id_netex']})"
            ).add_to(m)
```

```
    # add wiki stops (red)
    for _, row in stations_over_400.iterrows():
        if not pd.isna(row["lat_wiki"]) and not
pd.isna(row["lon_wiki"]):
            folium.CircleMarker(
                location=[row["lat_wiki"], row["lon_wiki"]],
                radius=5,
                color="green",
                fill=True,
                fill_color="red",
                popup=f"WIKI: {row['name_wiki']})"
            ).add_to(m)

    # save map
    m.save(country_name+"_distance_map.html")

# load the merged gtfs and netex data set from 2_Compare
import pandas as pd

country_name = "norway"

station_wiki_df = pd.read_csv(country_name +
"_station_+_wiki_df.csv",index_col = 0)


merged_wiki_distance_df, mean_netex, mean_gtfs =
distance_calculator(station_wiki_df ,"lat_wiki",
"lon_wiki","name_wiki", country_name)
```

```
number of matches between netex, gtfs and third source by their
coordinates: 140
```


```
{"config":{"plotlyServerURL":"https://plot.ly"},"data":
[{"hovertemplate":"Distance
Type=distancen_between_netex_to<br>name_wiki=%{x}<br>Distance in
meter=%{y}<extra></extra>","legendgroup":"distancen_between_netex_to",
"line":{"color":"#636efa","dash":"solid"},"marker":
{"symbol":"circle"},"mode":"lines","name":"distancen_between_netex_to"
,"orientation":"v","showlegend":true,"type":"scatter","x":["Oslo
Sentralstasjon","Bryn Station","Grorud Station","Høybråten
Station","Fjellhamar Station","Strømmen Station","Lillestrøm
station","Leirsund Station","Frogner Station","Bahnhof
Kløfta","Jessheim Station","Hauerseter Station","Dal
Station","Eidsvoll station","Nordstrand station","Ljan
station","Hauketo station","Kolbotn station","Myrvoll
station","Oppegård station","Langhus station","Ås station","Vestby
station","Kambo station","Moss station","Rygge station","Råde
station","Fredrikstad station","Bahnhof Sarpsborg","Kråkstad
```

```
station","Skotbu station","Tomter station","Knapstad
station","Spydeberg station","Askim station","Slitu station","Mysen
station","Eidsberg station","Heia station","Rakkestad
station","Bahnhof Halden","Kjelsås Station","Monsrud Station","Lunner
Station","Gjøvik Station","Tangen Station","Stange Station","Bahnhof
Hamar","Brumunddal Station","Moelv Station","Lillehammer
Station","Ringebu station","Vinstra Station","Kvam Station","Otta
Station","Dovre Station","Dombås Station","Lesja Station","Lesjaverk
Station","Bjorli Station","Ilseng Station","Løten Station","Elverum
station","Rena Station","Steinvik Station","Opphus Station","Evenstad
Station","Stai Station","Koppang Station","Atna Station","Hanestad
Station","Bellingmo Station","Os Station","Bahnhof Røros","Bahnhof
Glåmos","Reitan Station","Bahnhof Ålen","Singsås Station","Hjerkinn
Station","Kongsvoll Station","Oppdal Station","Bahnhof Støren","Hovin
Station","Lundamo Station","Ler Station","Kvål Station","Melhus
Station","Heimdal station","Selsbakk station","Lademoen
station","Leangen station","Bahnhof Hell","Stjørdal station","Bahnhof
Røra","Drevvatn station","Bahnhof Bodø","Skøyen station","Lysaker
station","Stabekk station","Høvik station","Sandvika
station","Billingstad station","Hvalstad station","Asker
station","Heggedal station","Spikkestad station","Brakerøya
station","Drammen station","Holmestrand station","Hokksund
station","Vestfossen station","Darbu station","Kongsberg
station","Nordagutu station","Bø station","Neslandsvatn
station","Bahnhof Kristiansand","Bahnhof Flaten","Bøylestad
Station","Froland Station","Bråstad
Station","Stoa","Arendal","Marnardal station","Forus station","Hinna
station","Haugastøl","Bahnhof Finse","Bahnhof Myrdal","Upsete
station","Voss station","Bulken station","Evanger
station","Bolstadøyri station","Bahnhof Dale","Stanghelle
station","Vaksdal station","Trengereid station","Bahnhof
Bergen","Bahnhof Flåm"],"xaxis":"x","y":
{"bdata":"h+9EKEuBM0DMFP7lHgwzQCwzMA2LGi1Ahy9xQyR3CEDZzEF5gUYgQIbtqsA5
vVVARbqyOcZ2NkC8OdOR9fY5QMIEdEH7dDxAkK0a0KcxUEBM2/9Y9stLQC9gECtxHUpA0L
li2XVCS0DcTjECmtFGQPlBxJLBPjNAQ232az37RUBvE8phqelMQGDBGFk+/
jJAdZrHPfSXC0AC52XVuR1+QAYau5lXSFBAdfLA2Pk5I0AJ1nc7+sgiQFvFkXWL6EJANxS
0zRaKLEAUyfi3a5o6QESOP6ltm0FArCliaRxbOkAcGZa+6eQxQHSY4pVsiEJAvWPruYhiX
ECjQXfvjXcwQFvz7dL35WlAerK8jW0lQECL21Vm4/ExQAg+Es1mQD9AtEjAaJ/
NTkCPVU3GoJ5TQNzXUDHRslJAvTRQRAfqNEAFGC5oEkgyQITNTIX35UVALHtoj1eFl0BVp
OXO3S9AQJ7j9mxxFxNAN32eEu1EEUCh6EL+tnpbQPNE6zLbiFpAkkhvR4KWKEDTrDp1E0B
WQOnAJEqyiE5AiWKf7n65QEDKEGzSvBw8QAB/8++5JRhAGNUy/
8StJEBtvGZPsrtOQPxcaFLFMFJAIW/
OpaxDMUBcCG9Q0JhCQJw9hhhdNktAYEQ+vYlPPkDzTkT/
vIJiQDuOkTENl0ZAimTOJY30Q0Aim/
hvDoRFQJnV0bvh7SdAmLQuAtHfNUAATFwuDbM+QKP/
QsAwuTxA+6ZGRi96UEDtUwjtEJooQOmPMGtASTZATwGIRCsxOECzhdQB8rVUQFv3unjNZB
BAsCitl6vSH0ClYkmiX/FmQPwDn9U/NTFAweVN/
NNdJECTwSbm4MJAQN9zmzH3f0lA05sGWlbmPUCXnAKSbKRjQBR1vi8AlkBAF5vlZJYXWUD
nRR4aSr1dQA6KhRVkoYJAEzZp169rKUBHlSwc2292QFirnXYEeYhAY1MlMycAZEAjkyUto
```

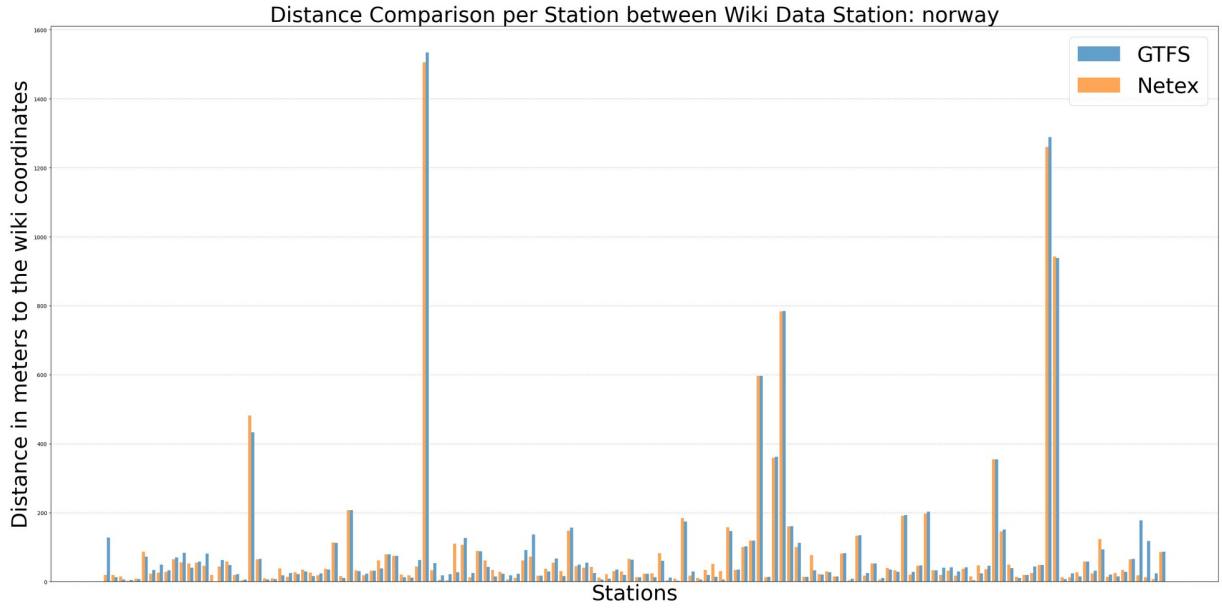
t5YQO97+kS/2itAFSAjX5whU0A+P6r8/lY1QCSYJc8DsDxALOeqAY/
KLUBQtkmqYjpUQDsLxkJFhQtAQmyRVPuxYEDLaZh7VIMxQJYQQFB4MEpAxvmNMVY7GkCwy
vLjF4RDQAUe72BsOEBAdFsx2wXjZ0DI57XB+pc0QGS4ZAYu5EZAw+POZ+qfaEAFHFE7ild
AQEIvSgjcJzNAeWRKTu12P0A+bTZoiNEwQNYEzCMMoUJASNyn4J7kLUB20WqJpURHQGCvd
mG2xkFAzjZQou4idkD1na+Y5DViQFXCK4iYukhAXx3uaOpjLEB4bnAODM0yQAW+HPUf1Dh
ALCgW+Sn4R0DT7GOsI7CTQFkWVwKTco1Ahj6k1trfKUCoW3NtLoEoQK0avODbvjpAI3onV
/v8TEBO0AUQmKU3QPIO1fre2V5AHE6bSdbqKkALEZl5AB85QMqAZCanukBAfXq3/
AczUEDO0AxTeNExQEw/
CAnReChAzixAmHRjHUAP3OeyIHtVQA==","dtype":"f8"},"yaxis":"y"},
{"hovertemplate":"Distance
Type=distancen_between_gtfs_to
name_wiki=%{x}
Distance in meter=
%{y}<extra></extra>","legendgroup":"distancen_between_gtfs_to","line":
{"color":"#EF553B","dash":"solid"},"marker":
{"symbol":"circle"},"mode":"lines","name":"distancen_between_gtfs_to",
"orientation":"v","showlegend":true,"type":"scatter","x":["Oslo
Sentralstasjon","Bryn Station","Grorud Station","Høybråten
Station","Fjellhamar Station","Strømmen Station","Lillestrøm
station","Leirsund Station","Frogner Station","Bahnhof
Kløfta","Jessheim Station","Hauerseter Station","Dal
Station","Eidsvoll station","Nordstrand station","Ljan
station","Hauketo station","Kolbotn station","Myrvoll
station","Oppegård station","Langhus station","Ås station","Vestby
station","Kambo station","Moss station","Rygge station","Råde
station","Fredrikstad station","Bahnhof Sarpsborg","Kråkstad
station","Skotbu station","Tomter station","Knapstad
station","Spydeberg station","Askim station","Slitu station","Mysen
station","Eidsberg station","Heia station","Rakkestad
station","Bahnhof Halden","Kjelsås Station","Monsrud Station","Lunner
Station","Gjøvik Station","Tangen Station","Stange Station","Bahnhof
Hamar","Brumunddal Station","Moelv Station","Lillehammer
Station","Ringebu station","Vinstra Station","Kvam Station","Otta
Station","Dovre Station","Dombås Station","Lesja Station","Lesjaverk
Station","Bjorli Station","Ilseng Station","Løten Station","Elverum
station","Rena Station","Steinvik Station","Opphus Station","Evenstad
Station","Stai Station","Koppang Station","Atna Station","Hanestad
Station","Bellingmo Station","Os Station","Bahnhof Røros","Bahnhof
Glåmos","Reitan Station","Bahnhof Ålen","Singsås Station","Hjerkinn
Station","Kongsvoll Station","Oppdal Station","Bahnhof Støren","Hovin
Station","Lundamo Station","Ler Station","Kvål Station","Melhus
Station","Heimdal station","Selsbakk station","Lademoen
station","Leangen station","Bahnhof Hell","Stjørdal station","Bahnhof
Røra","Drevvatn station","Bahnhof Bodø","Skøyen station","Lysaker
station","Stabekk station","Høvik station","Sandvika
station","Billingstad station","Hvalstad station","Asker
station","Heggedal station","Spikkestad station","Brakerøya
station","Drammen station","Holmestrand station","Hokksund
station","Vestfossen station","Darbu station","Kongsberg
station","Nordagutu station","Bø station","Neslandsvatn
station","Bahnhof Kristiansand","Bahnhof Flaten","Bøylestad

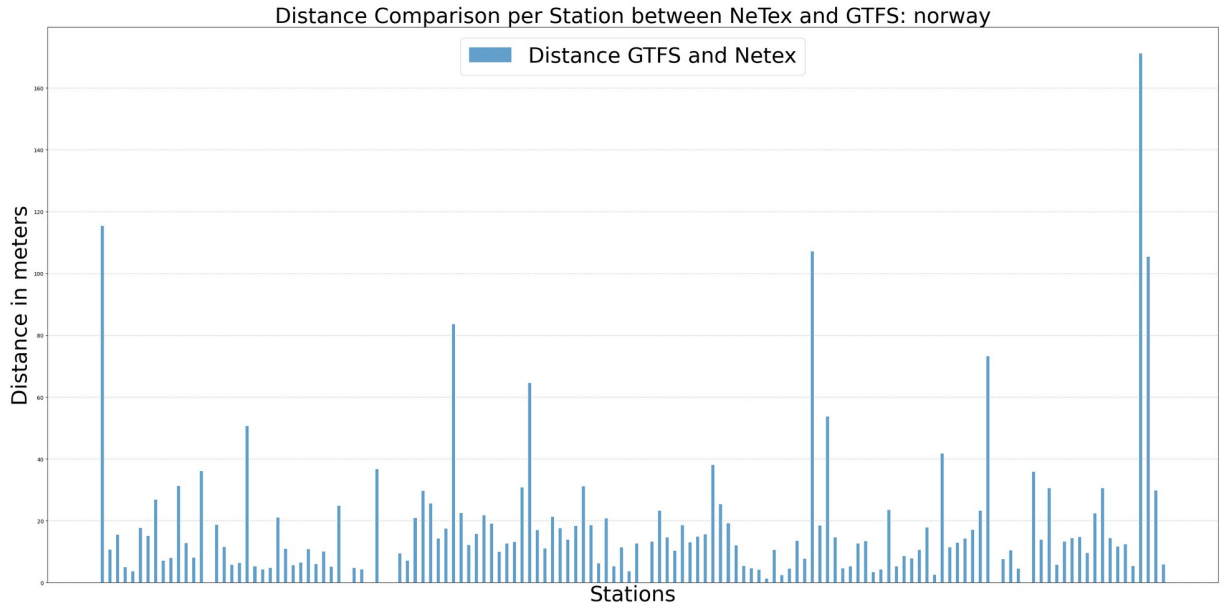
```
Station","Froland Station","Bråstad
Station","Stoa","Arendal","Marnardal station","Forus station","Hinna
station","Haugastøl","Bahnhof Finse","Bahnhof Myrdal","Upsete
station","Voss station","Bulken station","Evanger
station","Bolstadøyri station","Bahnhof Dale","Stanghelle
station","Vaksdal station","Trengereid station","Bahnhof
Bergen","Bahnhof Flåm"],"xaxis":"x","y":
{"bdata":"qNRw3zXxX0C+cJQTCF8oQL4LRqz8IxlAJfG8Ib5CE0B9rJ0RsiAeQKx2Zl+n
KVJAAC8DWFKvQECYCxBKo4tIQG1LMAPqXkBAnUUf8iZ1UUDQiKO+JNtUQMpRPaBF+ENA6o
+6Sa/+TEA0Zj511y9UQAAAAAAAAPh/
MRm6MQokT0Ak8lVlHTdIQM80JfB+wDVAzLmgcV7cGUCkYl0zmgl7QLXFa1CrTlBAn7IJe4
qoGEA0hE/SvKsbQFIFKJ7uizJA90d6V0A4OUAVCvYpBpY1QCAEvBVh4TxA/6ms/
vVfMEDw7nV9OOM3QDu3JMY9NEFAruBLMM/
1W0CRURndKE0kQFvz7dL35WlAv0d1RPXbPkAdxU34QSU2QAg+Es1mQD9Aru6+K+f4QkCPV
U3GoJ5TQNzXUDHRslJAPsBxS7saKkARyl1M3aEmQJAOnzqNIE9Ak5cN4vT2l0DAMTS63b5
KQF9YvrHppTJAjlJpGophNUDD8Dj4Ypg6QH32FTiwg19AONUA/
vBsOEBPphmONNxVQPe1sACkYkVAu9MQexhqLUA2k/
w59Uk2QHPhjAIzfDJACQTWU6KzNkB+R1qABbBWQC7t0UINEWFABTi37ulTMUAoJlQApco9
QPoJdZnGtVBA7WG5Ebg7L0D7UDBQNIZjQPwKtPKfVkhAhHO55DZGS0DfskVFARk5QNbvza
M8AxdAJJFU1rrcH0CTxE/
hP4FBQKWfPfcqMDNAWWBGESKIT0B0vbcvW5IpQOmPMGtASTZA/
sQRA56uKUCjPqniVBFOQAutcO64EyZAVc+Ct2AjBUBqXV4b38hlQBb54BPbPz1Aaaotvf+
TGUBqPMAh6R0zQFTn+vfp6ipAvR78aPIqGkC9T3AYsk9iQO7eY5legkFAQXR6oZWfWUDYP
VigB7JdQHbN3h4KpoJA93ZHqRSoKkAlz01K9Jt2QJSSR7NxgIhAaGoxfp4OZEDqYZynx/
JbQCu/kj6YXCtAZzabfkkaQEBYLGZ31fUzQPuCo2X/wjpAMS8GIyyZLkA/
+bycAXxUQFrjFHfbOSFApM5yoO7PYEBg61ebxzs5QFQQyizZMEpA/
7TdSRBZJUBxazD7LjZBQDGRG1h9oDxAA+o2D7YZaEAGsFjP1UU8QH+hukqtlUdAQsWRDjd
VaUDY+oNc4EtAQOJMP+Ng6UNAypvIySx+REA+AY4Fx6k9QEsSM82L+ERAtecbvDmSC0Dsn
tqhht03QEUiyVCSCkdAzjZQou4idkB1ACHyQ+1iQDmqMsz0lENAl/xZ/
IJcJEB4bnAODM0yQFuIVSwHjUVAMyj6Pfi+R0DpXhap0yOUQNCQ0BmGVY1AP4xiWf90GkA
O9o/
BsUE4QAoC4OBYuS1AfJGLTeX+TEAfDlDejvM+QGFybz3xN1dAqJggb9ReNEDH2V8OvuYsQ
FCjVM9/RTxAbo/
ZGLJxUEBrAx4VBSZmQDudQ3FsS11Ab835skYtN0AGSrCW3MtVQA==","dtype":"f8"},"
yaxis":"y"}],"layout":{"legend":{"title":{"text":"Distance
Type"},"tracegroupgap":0},"template":{"data":{"bar":[{"error_x":
{"color":"#2a3f5f"},"error_y":{"color":"#2a3f5f"},"marker":{"line":
{"color":"#E5ECF6","width":0.5},"pattern":
{"fillmode":"overlay","size":10,"solidity":0.2}},"type":"bar"}],"barpo
lar":[{"marker":{"line":{"color":"#E5ECF6","width":0.5},"pattern":
{"fillmode":"overlay","size":10,"solidity":0.2}},"type":"barpolar"}],"
carpet":[{"aaxis":
{"endlinecolor":"#2a3f5f","gridcolor":"white","linecolor":"white","min
orgridcolor":"white","startlinecolor":"#2a3f5f"},"baxis":
{"endlinecolor":"#2a3f5f","gridcolor":"white","linecolor":"white","min
orgridcolor":"white","startlinecolor":"#2a3f5f"},"type":"carpet"}],"ch
oropleth":[{"colorbar":
{"outlinewidth":0,"ticks":""},"type":"choropleth"}],"contour":
[{"colorbar":{"outlinewidth":0,"ticks":""},"colorscale":
```

```
[[0,"#0d0887"],[0.1111111111111111,"#46039f"],
[0.2222222222222222,"#7201a8"],[0.3333333333333333,"#9c179e"],
[0.4444444444444444,"#bd3786"],[0.5555555555555556,"#d8576b"],
[0.6666666666666666,"#ed7953"],[0.7777777777777778,"#fb9f3a"],
[0.8888888888888888,"#fdca26"],
[1,"#f0f921"]],"type":"contour"}],"contourcarpet":[{"colorbar":
{"outlinewidth":0,"ticks":""},"type":"contourcarpet"}],"heatmap":
[{"colorbar":{"outlinewidth":0,"ticks":""},"colorscale":
[[0,"#0d0887"],[0.1111111111111111,"#46039f"],
[0.2222222222222222,"#7201a8"],[0.3333333333333333,"#9c179e"],
[0.4444444444444444,"#bd3786"],[0.5555555555555556,"#d8576b"],
[0.6666666666666666,"#ed7953"],[0.7777777777777778,"#fb9f3a"],
[0.8888888888888888,"#fdca26"],
[1,"#f0f921"]],"type":"heatmap"}],"histogram":[{"marker":{"pattern":
{"fillmode":"overlay","size":10,"solidity":0.2}},"type":"histogram"}],
"histogram2d":[{"colorbar":{"outlinewidth":0,"ticks":""},"colorscale":
[[0,"#0d0887"],[0.1111111111111111,"#46039f"],
[0.2222222222222222,"#7201a8"],[0.3333333333333333,"#9c179e"],
[0.4444444444444444,"#bd3786"],[0.5555555555555556,"#d8576b"],
[0.6666666666666666,"#ed7953"],[0.7777777777777778,"#fb9f3a"],
[0.8888888888888888,"#fdca26"],
[1,"#f0f921"]],"type":"histogram2d"}],"histogram2dcontour":
[{"colorbar":{"outlinewidth":0,"ticks":""},"colorscale":
[[0,"#0d0887"],[0.1111111111111111,"#46039f"],
[0.2222222222222222,"#7201a8"],[0.3333333333333333,"#9c179e"],
[0.4444444444444444,"#bd3786"],[0.5555555555555556,"#d8576b"],
[0.6666666666666666,"#ed7953"],[0.7777777777777778,"#fb9f3a"],
[0.8888888888888888,"#fdca26"],
[1,"#f0f921"]],"type":"histogram2dcontour"}],"mesh3d":[{"colorbar":
{"outlinewidth":0,"ticks":""},"type":"mesh3d"}],"parcoords":[{"line":
{"colorbar":{"outlinewidth":0,"ticks":""}},"type":"parcoords"}],"pie":
[{"automargin":true,"type":"pie"}],"scatter":[{"fillpattern":
{"fillmode":"overlay","size":10,"solidity":0.2},"type":"scatter"}],"sc
atter3d":[{"line":{"colorbar":{"outlinewidth":0,"ticks":""}},"marker":
{"colorbar":
{"outlinewidth":0,"ticks":""}},"type":"scatter3d"}],"scattercarpet":
[{"marker":{"colorbar":
{"outlinewidth":0,"ticks":""}},"type":"scattercarpet"}],"scattergeo":
[{"marker":{"colorbar":
{"outlinewidth":0,"ticks":""}},"type":"scattergeo"}],"scattergl":
[{"marker":{"colorbar":
{"outlinewidth":0,"ticks":""}},"type":"scattergl"}],"scattermap":
[{"marker":{"colorbar":
{"outlinewidth":0,"ticks":""}},"type":"scattermap"}],"scattermapbox":
[{"marker":{"colorbar":
{"outlinewidth":0,"ticks":""}},"type":"scattermapbox"}],"scatterpolar"
:[{"marker":{"colorbar":
{"outlinewidth":0,"ticks":""}},"type":"scatterpolar"}],"scatterpolargl
":[{"marker":{"colorbar":
```

```
{"outlinewidth":0,"ticks":""}},"type":"scatterpolargl"}],"scatterterna
ry":[{"marker":{"colorbar":
{"outlinewidth":0,"ticks":""}},"type":"scatterternary"}],"surface":
[{"colorbar":{"outlinewidth":0,"ticks":""},"colorscale":
[[0,"#0d0887"],[0.1111111111111111,"#46039f"],
[0.2222222222222222,"#7201a8"],[0.3333333333333333,"#9c179e"],
[0.4444444444444444,"#bd3786"],[0.5555555555555556,"#d8576b"],
[0.6666666666666666,"#ed7953"],[0.7777777777777778,"#fb9f3a"],
[0.8888888888888888,"#fdca26"],
[1,"#f0f921"]],"type":"surface"}],"table":[{"cells":{"fill":
{"color":"#EBF0F8"},"line":{"color":"white"}},"header":{"fill":
{"color":"#C8D4E3"},"line":
{"color":"white"}},"type":"table"}]},"layout":{"annotationdefaults":
{"arrowcolor":"#2a3f5f","arrowhead":0,"arrowwidth":1},"autotypenumbers
":"strict","coloraxis":{"colorbar":
{"outlinewidth":0,"ticks":""}},"colorscale":{"diverging":
[[0,"#8e0152"],[0.1,"#c51b7d"],[0.2,"#de77ae"],[0.3,"#f1b6da"],
[0.4,"#fde0ef"],[0.5,"#f7f7f7"],[0.6,"#e6f5d0"],[0.7,"#b8e186"],
[0.8,"#7fbc41"],[0.9,"#4d9221"],[1,"#276419"]],"sequential":
[[0,"#0d0887"],[0.1111111111111111,"#46039f"],
[0.2222222222222222,"#7201a8"],[0.3333333333333333,"#9c179e"],
[0.4444444444444444,"#bd3786"],[0.5555555555555556,"#d8576b"],
[0.6666666666666666,"#ed7953"],[0.7777777777777778,"#fb9f3a"],
[0.8888888888888888,"#fdca26"],[1,"#f0f921"]],"sequentialminus":
[[0,"#0d0887"],[0.1111111111111111,"#46039f"],
[0.2222222222222222,"#7201a8"],[0.3333333333333333,"#9c179e"],
[0.4444444444444444,"#bd3786"],[0.5555555555555556,"#d8576b"],
[0.6666666666666666,"#ed7953"],[0.7777777777777778,"#fb9f3a"],
[0.8888888888888888,"#fdca26"],[1,"#f0f921"]]},"colorway":
["#636efa","#EF553B","#00cc96","#ab63fa","#FFA15A","#19d3f3","#FF6692"
,"#B6E880","#FF97FF","#FECB52"],"font":{"color":"#2a3f5f"},"geo":
{"bgcolor":"white","lakecolor":"white","landcolor":"#E5ECF6","showlake
s":true,"showland":true,"subunitcolor":"white"},"hoverlabel":
{"align":"left"},"hovermode":"closest","mapbox":
{"style":"light"},"paper_bgcolor":"white","plot_bgcolor":"#E5ECF6","po
lar":{"angularaxis":
{"gridcolor":"white","linecolor":"white","ticks":""},"bgcolor":"#E5ECF
6","radialaxis":
{"gridcolor":"white","linecolor":"white","ticks":""}},"scene":
{"xaxis":
{"backgroundcolor":"#E5ECF6","gridcolor":"white","gridwidth":2,"lineco
lor":"white","showbackground":true,"ticks":"","zerolinecolor":"white"}
,"yaxis":
{"backgroundcolor":"#E5ECF6","gridcolor":"white","gridwidth":2,"lineco
lor":"white","showbackground":true,"ticks":"","zerolinecolor":"white"}
,"zaxis":
{"backgroundcolor":"#E5ECF6","gridcolor":"white","gridwidth":2,"lineco
lor":"white","showbackground":true,"ticks":"","zerolinecolor":"white"}
},"shapedefaults":{"line":{"color":"#2a3f5f"}},"ternary":{"aaxis":
```

```
{"gridcolor":"white","linecolor":"white","ticks":""},"baxis":
{"gridcolor":"white","linecolor":"white","ticks":""},"bgcolor":"#E5ECF
6","caxis":
{"gridcolor":"white","linecolor":"white","ticks":""}},"title":
{"x":5.0e-2},"xaxis":
{"automargin":true,"gridcolor":"white","linecolor":"white","ticks":"",
"title":
{"standoff":15},"zerolinecolor":"white","zerolinewidth":2},"yaxis":
{"automargin":true,"gridcolor":"white","linecolor":"white","ticks":"",
"title":
{"standoff":15},"zerolinecolor":"white","zerolinewidth":2}}},"title":
{"text":"Distance Comparison per Station"},"xaxis":
{"anchor":"y","domain":[0,1],"tickangle":-80,"tickfont":
{"size":9},"title":{"text":"name_wiki"}},"yaxis":
{"anchor":"x","domain":[0,1],"title":{"text":"Distance in meter"}}}}
```




```
avearage distance netex to third source: 87.98512639957524
avearage distance gtfs to third source: 90.94894419626833

stations_over_400 =
merged_wiki_distance_df[(merged_wiki_distance_df["distancen_between_ne
tex_to"] > 400) |
    (merged_wiki_distance_df["distancen_between_gtfs_to"] > 400)]

mapping_far_stations(stations_over_400)
```