Key Drivers of Attractiveness of Regional Rail for Commuters: Case Study of Vienna and Lower Austria

Maciej Kilijański

11.11.2024

Data gathering

The analysis of data (Bambrick 2016)

NetEx

TODO: Until 24.11

History

Description of data model

Model transformation

GTFS

TODO: Until 30.11

History

Description of data model

Model transformation

Vagonweb

Collecting data about rolling stock assigned to certain connections is a notoriously hard task for public railway operators in Europe. There is no open data policy forced by any of the european regulators to publish such data. For the purpose of this study, the publically available information on each connection was not sufficient

Timetables only provided information on accessibility for disabled people, WiFi, availability of the 1st class, possibility to take bicycles onboard and low-floor carriages. No data on availaility of air conditioning onboard and the age of rolling stock were provided. These were however included in the scope of this study creating a need for alternative sourcing of data.

With no official sources availabe this paper resorted to webscraping of the fan-made online forum created by railway enthusiasts - vagonweb.cz.

TODO: Description and history of vagonweb

Webscraping - methodology

With lack of an official API supported by the website for downloading the data on rolling stock compositions this paper resorted to webscraping the data. The process was conducted in Python using packages described in Table 1.

Library	Description
`urllib.request`	Used for executing callouts to the website itself and receiving the data.
`urllib.parse`	Used for URL parsing for the callouts to be made.
`BeautifulSoup`	Used for parsing the data retrieved and retrieving the actual part of data necessary for the analysis from the html structure.
`ssl`	Used for ssl authentication to allow retrieving data using the https protocol.

Due to complexity of the webscraping operation, the code was then divided into two files: a controller and a performer. The controler file consisted of functions running locally, importing the list of all connections needing retrieval, controlling the pace of data querying and writing the data back locally. The performer file consisted of methods interacting with the website and processing the html content retrieved. Such setup made code debugging easier and followed code readablility best practices (**SOURCE NEEDED**).

Creating request URL

The `create_request` method creates a unique request URL for each of the train connections needing retrieval. It takes website's base URL of `https://www.vagonweb.cz/razeni/vlak.php?` and appends necessary information to it. Then a parsed string is returned. The information allowing to retrieve each train were:

- Operator - ÖBB for the case of Lower Austria and Vienna (**SOURCE NEEDED**)
- Category - S for S-Bahn, R for Regio and REX for Regional Expressess. Other categories like CAT (City Airport Train) or long distance D-Zuge, Euro/Inter-City and RailJets were not considered as the scope of this study only includes regional, commuter trains.
- Number - train number
- Year - year of the timetable.

```python
def create_request(operator: str, category: str, number: str, year=2024) -> str:
    url = baseurl
    if operator:
        url = url + "&zeme=" + parse_url.quote(operator)
    if category:
        url = url + "&kategorie=" + category
    if number:
        url = url + "&cislo=" + number
    if year:
        url = url + "&rok=" + str(year)
    return url
```

TODO: Until 06.12

Description & Problem

Webscraping: Code and performance

Pendler Dataset

The pendler dataset was joined to Raster `shp` file and mapped.

TODO: Until 06.12

Road network

https://www.data.gv.at/katalog/dataset/no-strassen-b-und-l-strassengraph-level-1#resources https://www.data.gv.at/katalog/dataset/stadt-wien_straengraphwien#resources TODO: Until 11.12

The road networks of Vienna and Lower Austria were retrieved in `shp` format and transformed into a routable network using ArcGIS Pro's Network Analyst tool.

The road network was constructed without use of elevation. Three modes of transport were added:

- Driving
- Cycling
- Walking

Model calculation

The model considered in this paper will was calculated as following. In order to measure attractiveness of physical infrastructure for each of the 250m rasters, each raster was first assigned the nearest station based on proximity to their central feature. Then the bi-directional passenger potentials were calculated for each combination of stations based on the assignment made. Thanks to the availability of GTFS transit data, these potentials were attributed to sets of morning and afternoon connections using a bi-directional network.

Each potential record was also assigned the amenities offered on both stations and rolling stock. This allowed for a *route score* calculation. The route score was then backfitted with data about potentially necessary train changes and additional time necessary for it. Moreover, each connection was assigned a service frequency score. The last factor accounted for was the time of arrival at the destination station. Following paragraphs provide a detailed description of the calculation process.

Distance raster to station

In order to assign stations to rasters, each raster had a centroid calculated, from which distance was measured to a station. The centroid calculation was conducted using the Point tool, according to Esri's documentation (https://pro.arcgis.com/en/pro-app/latest/help/editing/create-point-and-multipoint-features.htm). The raster dataset was retrieved from Statistik Austria and filtered by location to only include rasters that intersect communities within Lower Austria and Vienna (https://www.data.gv.at/katalog/dataset/566c99beb436-365e-af4f-27be6c536358). Filtering was conducted using Gemeinde Ids. This allowed the Raster dataset to shrink from over a million rows to 316317. The facility assignment tool of the Network Analyst toolbox in ArcGIS Pro requires points as features to assign, hence a centroid of each raster was calculated. For this the Feature to Point tool of ArcGIS Pro was used (https://support.esri.com/en-us/knowledge-base/how-to-find-the-centroid-of-polygons-using-calculate-ge-000021849). Assignment of each raster centroid to the nearest station was conducted using the road network for Vienna and Lower Austria and Network Analyst tool in ArcGIS Pro. The size of the dataset was posing a problem:

- 381 stations (*Facilities*)
- 316.317 rasters (*Incidents*)

Giving $381 \cdot 316.317 = 120.516.777$ possible combinations.

Such number of calculations turned out to be infeasible for ArcGIS Pro as the problem would not even construct. Therefore, heuristics became a necessity. Following the example of previous analysis (https://www.arbeiterkammer.at/infopool/wien/Verkehr_und_Infrastruktur_56.pdf), each station was assigned a distance band along the road network. The distance bands were weighted, which later allowed to assign points for closeness of a station. Still following the reasoning of (https://www.arbeiterkammer.at/infopool/wien/Verkehr_und_Infrastruktur_56.pdf), the distance bands calculated for each station were:

- 500 meters
- 1000 meters
- 1500 meters
- 2000 meters
- 4000 meters
- 6000 meters
- 8000 meters

Then, each raster was assigned to a station based on the distance band it fell into. Rasters further than 8000 meters were excluded from the analysis. This filtering was conducted by intersecting rasters with 8km distance bands. The following method allowed to construct an origin-destination matrix for all 381 stations in Lower Austria and Vienna as destinations and x rasters within 8 km from a station. Due to low quality of data on road network in Lower Austria, the bands could not be calculated according to it and were calculated as Straight-line distances.

The rasters were attributed to stations using `geopandas` package in Python. The scale of necessary calculations showed to be impossible to execute in ArcGIS Pro.

Raster to station attribution process in python

Data exported from ArcGIS Pro as `shp` files was loaded to python using the `geopandas` package.

```
import geopandas as gpd
raster_points_Wien = gpd.read_file(
    "ArcGIS/RasterPointsWien_ExportFeatures.shp"
)
raster_points_NO = gpd.read_file(
    "ArcGIS/RasterPointsNO.shp"
```

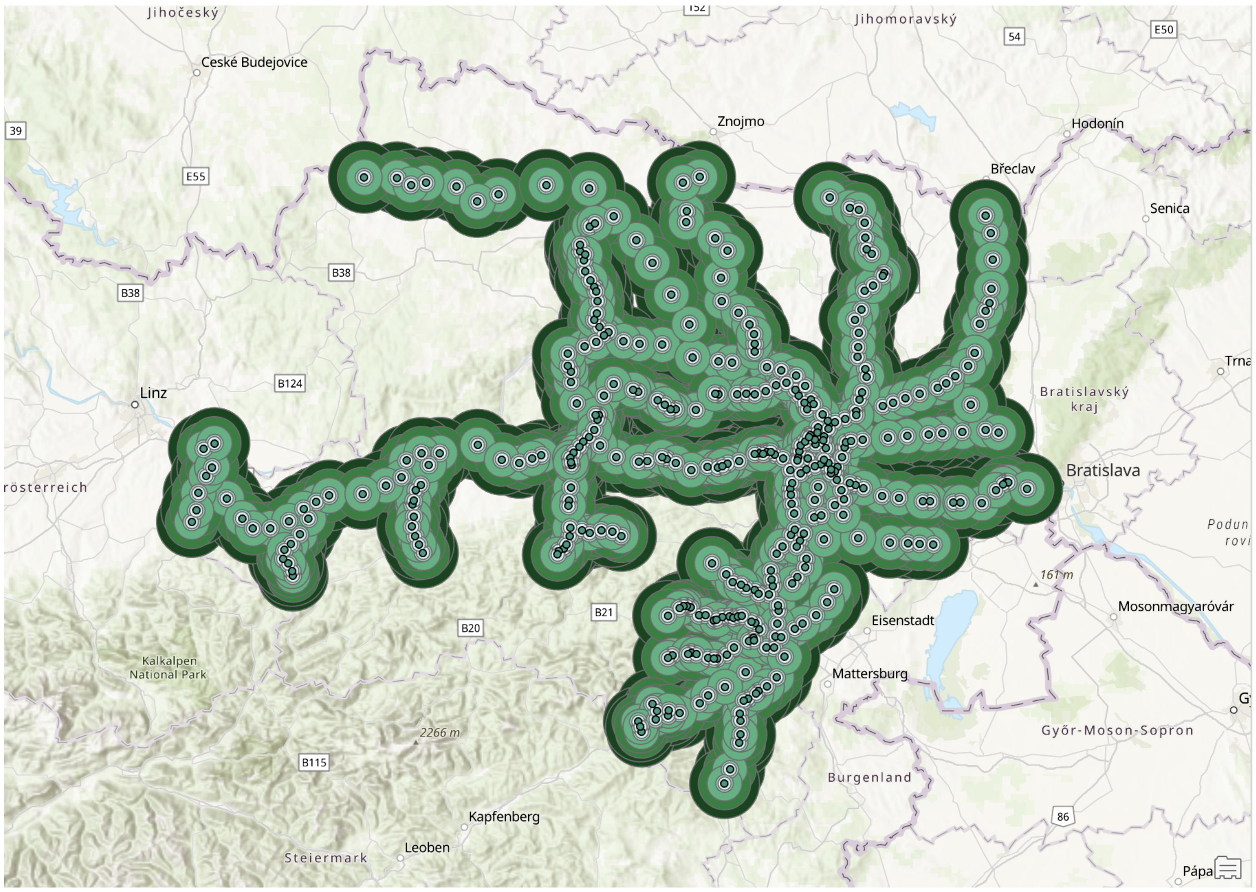

Figure 1: Distance bands around stations

Key Drivers of Attractiveness of Regional Rail for Commuters: Case Study of Vienna and Lower Austria

Maciej Kilijański

11.11.2024

Data gathering

The analysis of data (Bambrick 2016)

NetEx

TODO: Until 24.11

History

Description of data model

Model transformation

GTFS

TODO: Until 30.11

History

Description of data model

Model transformation

Vagonweb

Collecting data about rolling stock assigned to certain connections is a notoriously hard task for public railway operators in Europe. There is no open data policy forced by any of the european regulators to publish such data. For the purpose of this study, the publically available information on each connection was not sufficient

Timetables only provided information on accessibility for disabled people, WiFi, availability of the 1st class, possibility to take bicycles onboard and low-floor carriages. No data on availaility of air conditioning onboard and the age of rolling stock were provided. These were however included in the scope of this study creating a need for alternative sourcing of data.

With no official sources availabe this paper resorted to webscraping of the fan-made online forum created by railway enthusiasts - vagonweb.cz.

TODO: Description and history of vagonweb

1

Figure 2: Final Raster Selection

```
)
distance_bands_Wien = gpd.read_file(
    "ArcGIS/DistanceAreasWien.shp"
)
distance_bands_NO = gpd.read_file(
    "ArcGIS/Stops_MultipleRingBuffer_ExportFeatures.shp"
)
```

In order to execute a spacial join operation correctly, all geometries were transformed to use one referencing system

```
distance_bands_Wien = distance_bands_Wien.to_crs(raster_points_Wien.crs)
raster_points_NO = raster_points_NO.to_crs(raster_points_Wien.crs)
distance_bands_NO = distance_bands_NO.to_crs(raster_points_Wien.crs)
```

As stations in Vienna were considered separately, due to better quality of road network data, spatial join operations were conducted separately for Vienna and Lower Austria.

```
raster_join_wien = raster_points_Wien.sjoin(distance_bands_Wien, how="left")
raster_join_no = raster_points_NO.sjoin(distance_bands_NO, how="left")
```

With joins ready, the data was transformed into lists of dictionaries for Vienna and Lower Austria and then merged. All lists were transformed into pandas dataframes and saved as `csv` files for further processing.

```
def DistanceMatrixVienna(
    rasters=raster_join_wien, pattern=r"(at:\d+:\d+).*?(\d+ - \d+)"
) -> list:
    # Extract "station_code" and "range" from the "Name" column
    extracted = rasters["Name"].str.extract(pattern)
    rasters["station_code"] = extracted[0]  # Matches the first group (e.g., at:49:94)
    rasters["range"] = extracted[1]  # Matches the second group (e.g., 6000 - 8000)

    # Filter out rows where 'FromBreak' is NaN
    rasters = rasters.dropna(subset=["FromBreak"])

    # Group by 'gid' and construct dictionaries
    matrix = [
        {"gid": gid, **dict(zip(group["station_code"], group["FromBreak"]))}
        for gid, group in rasters.groupby("gid")
    ]

    return matrix



def DistanceMatrixLowerAustria(rasters=raster_join_no, stations=None) -> list:
    # Filter out rows where 'distance' is NaN
```

```python
    rasters = rasters.dropna(subset=["distance"])

    # Group by 'gid' and construct dictionaries for each group
    matrix = [
        {"gid": gid, **dict(zip(group["GStopID"], group["distance"]))}
        for gid, group in rasters.groupby("gid")
    ]

    return matrix


dist_vienna = DistanceMatrixVienna()
dist_lower_austria = DistanceMatrixLowerAustria()
OD_Vienna = pd.DataFrame(dist_vienna)
OD_Vienna.to_csv("Straßennetz/WienMatrix.csv")
OD_LoverAustria = pd.DataFrame(dist_lower_austria)
OD_LoverAustria.to_csv("Straßennetz/NOMatrix.csv")


def MergeLists(base=dist_lower_austria, to_merge=dist_vienna, key="gid"):
    # Create a dictionary to hold the merged results
    # Convert lists of dictionaries into DataFrames
    df1 = pd.DataFrame(base).set_index(key)
    df2 = pd.DataFrame(to_merge).set_index(key)

    # Merge the two DataFrames on the key, preserving all data
    merged_df = df1.combine_first(df2).reset_index()

    # Convert the merged DataFrame back to a list of dictionaries
    merged_list = merged_df.to_dict(orient="records")
    return merged_list


OD_All = pd.DataFrame(MergeLists())
OD_All.to_csv("Straßennetz/GrossraumWienMatrix.csv")
```

TODO: Until 13.12

Station to station passenger potential based on Pendler dataset

Each raster was assigned a station based on a distance from it to a station. The shorter distance was assigned to a raster. In case of many stations being within the same distance band they were assigned randomly with uniform probability. Due to the size of the dataset multicore processing was utilized in python using the `multiprocessing` package. Final code used is available in the appendix.

```python
import pandas as pd
from multiprocessing import Pool, cpu_count
from datetime import datetime

with_station_path = "Straßennetz/WithStation.csv"



def process_row(row) -> dict[str, str, int]:
    clean_row = row.drop("gid").dropna()
    lowest = clean_row.idxmin()
    return {
        "raster": row["gid"],
        "stop_id": lowest,
        "distance": clean_row[lowest],
        "finished_at": datetime.now(),
    }



def main() -> pd.DataFrame:
    matrix = pd.read_csv(with_station_path, index_col=0)

    rows = [row for _, row in matrix.iterrows()]
    num_cores = cpu_count() - 2
    start = datetime.now()
    print(f"Started at {start}")
    with Pool(num_cores) as pool:
        assignments = pool.map(process_row, rows)

    end = datetime.now()
    print(f"Executed {len(rows)} in {end-start}.")
    return pd.DataFrame(assignments)



if __name__ == "__main__":
    main().to_csv("Straßennetz/StationAssignment.csv")
```

Thanks to no use of loops and multicore processing the whole calculation executed in 12.4 seconds for 197.820 rows. The assigned station assignment was then used to aggregate station-to-station passenger flows. Each station had a sum departing and arriving passengers calculated. These sums were calculated in the same form as in the Pendler dataset from Statistik Austria, meaning a total of commuters, work commuters and school commuters. The formula for calculation was:

$$\sum_{r \in R} passengers_r$$

Where r represents a raster in Rasters R

The aggregation was conducted in Tableau Prep, due to large scale of calculations needed. An ETL tool handled necessary work in very low time, while testing in Python proved to be challenging for the tool. At this point a graphical representation of results was possible to present. This was presented in Tableau.

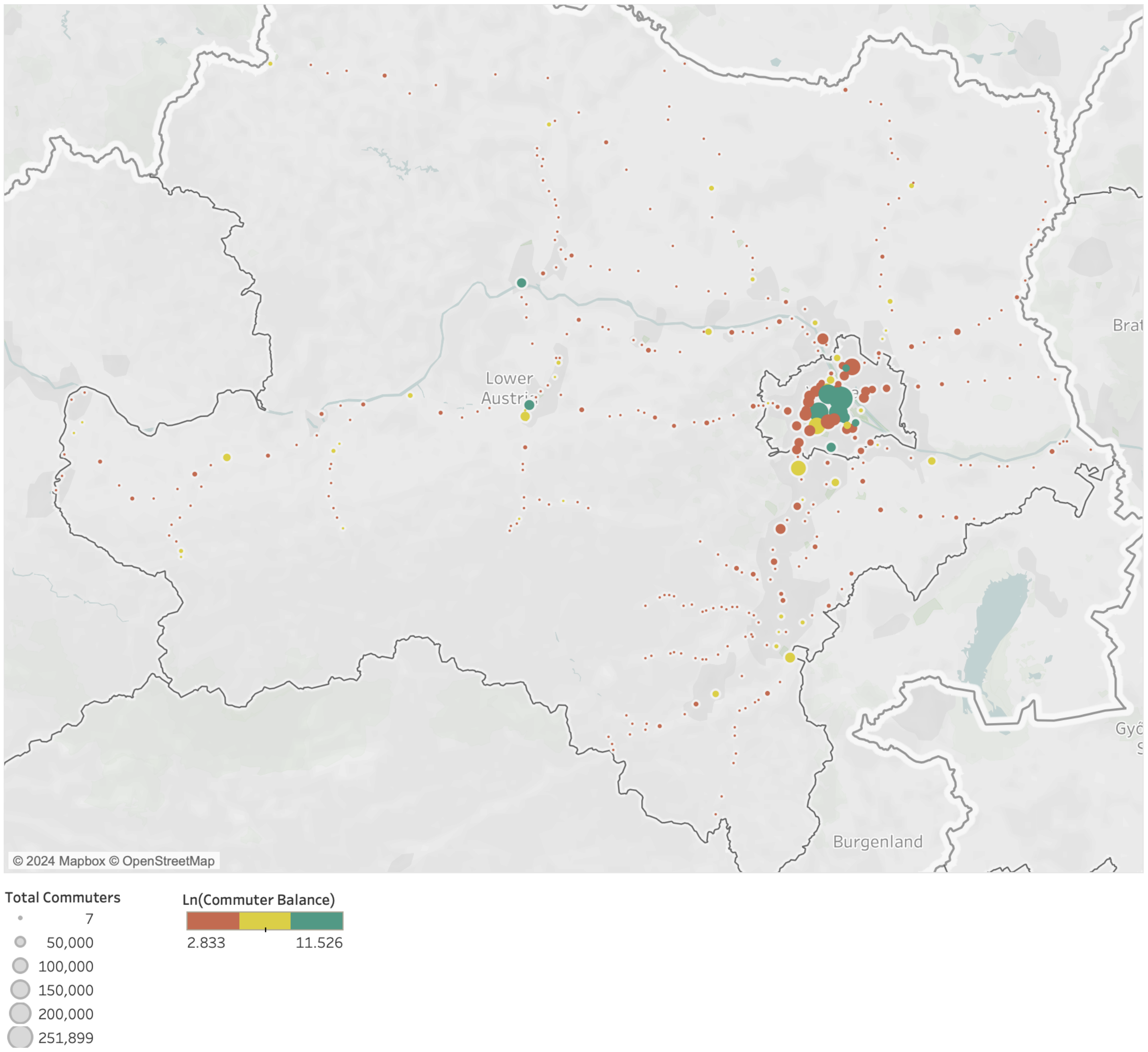

Figure 3: Commuter Balance Map

Figure above displays a map of commuter balances for stations in Vienna and Lower Austria. Size of the dot demarks total number of commuters that could potentially utilize the station, both leaving from it (from their place of residence) and arriving at it (to their place of work/study). Color denotes the balance between commuters leaving from and arriving to a station. Three categories were selected: red denotes negative commuter balance (more people leaving than arriving), yellow denotes close to equal yet positive commuter balance and green a strongly positive commuter balance.
What turns attention quickly is the concentration of strongly positive stations in Vienna city center (with

the exception of Wien Hauptbahnhof). Outside of Vienna only 14 stations had a strong positive balance: St. Pölten Hauptbahhof, Krems, Tulln, Katzelsdorf, St. Pölten Bildungscampus, Hollabrunn, Amstetten, Brunn-Maria Enzendorf, Achau, Horn, Gunstramsdorf Thallern, Korneuburg, Wieselburg, Fischamend.

Including Vienna the station potential diagram becomes dominated by the metropoly. 6 biggest potential stations are in Vienna, followed by Brunn-Maria-Enzersdorf (close to Vienna), which is then followed by another 8 stations in Vienna. This is visible in Figure x.

In order to calculate amenities available for each commute to work, necessary routings were calculated. As rasters do differ in proximity to stations, this paper once again takes inspiration from https://www.arbeiterkammer.at/infopool/wien/Verkehr_und_Infrastruktur_56.pdf and calculates necessary commutes for both SLOW and MIV modes of transport. This time we took into consideration walking time calculated with data from https://doi.org/10.1016/j.sbspro.2012.12.237. The average walking pace was 1.4 meters per second, which translates to 5 km/h. Moreover, considering Figure 6, 65% of all commuters arrive in Vienna, therefore in order to calculate the MIV transit option for the average speed of Vienese trams was taken into account. This was measured to be 15.44 km/h (https://repositum.tuwien.at/bitstream/20.500.12708/193238/1/Steinwidder%20Paul%20-%202024%20-%20Bewertungsverfahren%20zur%20Analyse%20der. . . pdf). The time to final destination was calculated for each of the distance bands, then rounded up to a full minute:

Distance [meters]	Walking time	Public Transit Time
500	6	2
1000	12	4
1500	18	6
2000	24	8
4000	48	18
6000	72	24
8000	96	32

From this table bins were calculated for time necessary to arrive before 8 - the normal work and school start hour in Austria (**SOURCE**). The bins were: 10 minutes prior, 20 minutes prior, 30 minutes prior and 40 minutes prior. Values larger than these were removed due to unlikely nature of such commute.
Station to station times were then calculated using the method described and developed in https://github.com/amitrm/shortest-path-using-gtfs/tree/master. The script was modified to accomodate necessary restriction regarding arrival times (bins). The script imported filtered and simplified GTFS data from ÖBB and calculated shortest paths for each set of stations using the Djkstra algorithm. This allowed to save data on commute between each set of stations for each arrival time for the morning rush. Results were then saved as a `json` file structured in a following way:

```
{
  "from-to-arrival": {
    "from": "str",
    "departure": "datetime",
    "to": "str",
    "arrival": "datetime",
```

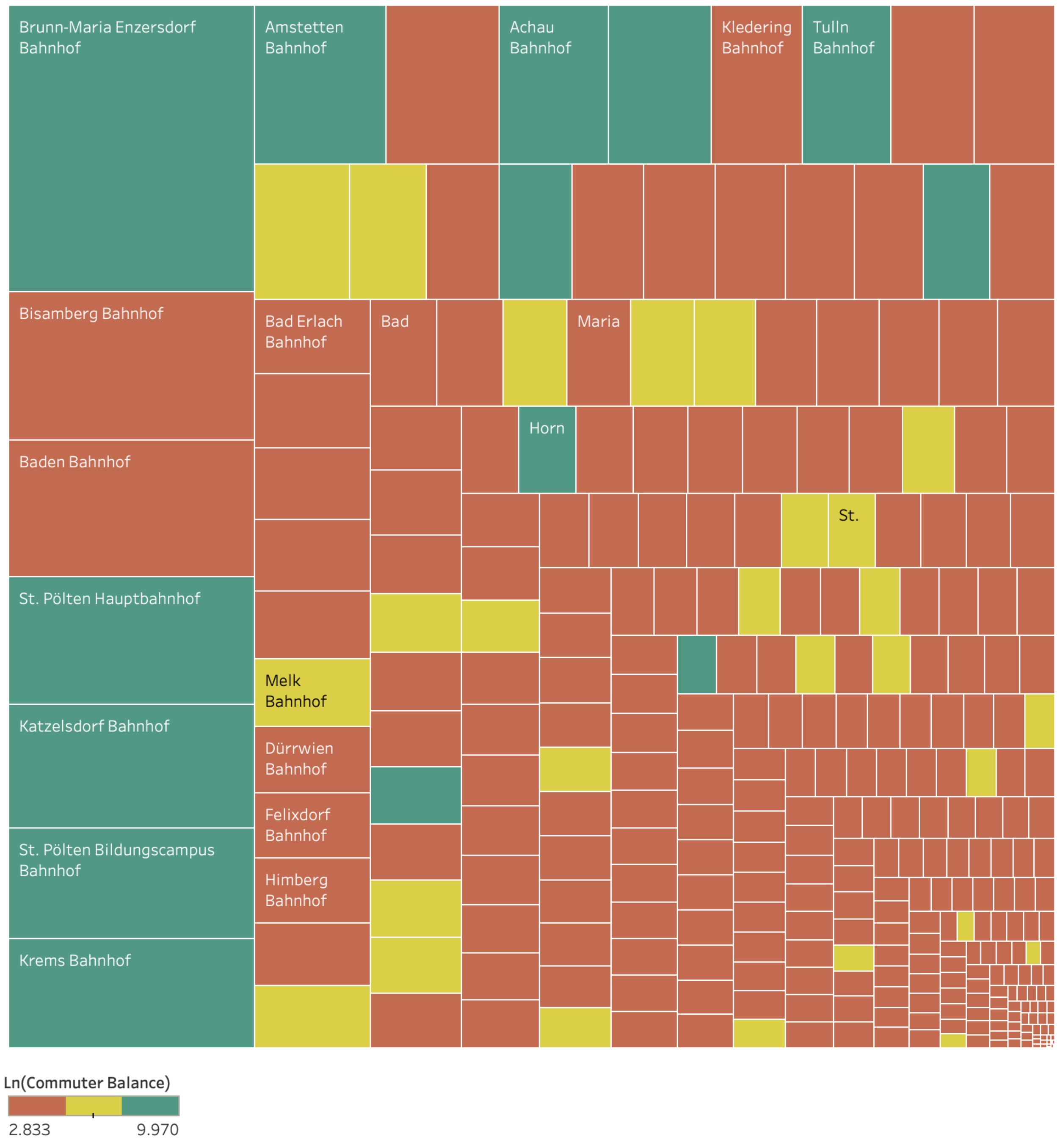

Figure 4: Biggest potential stations excluding Vienna

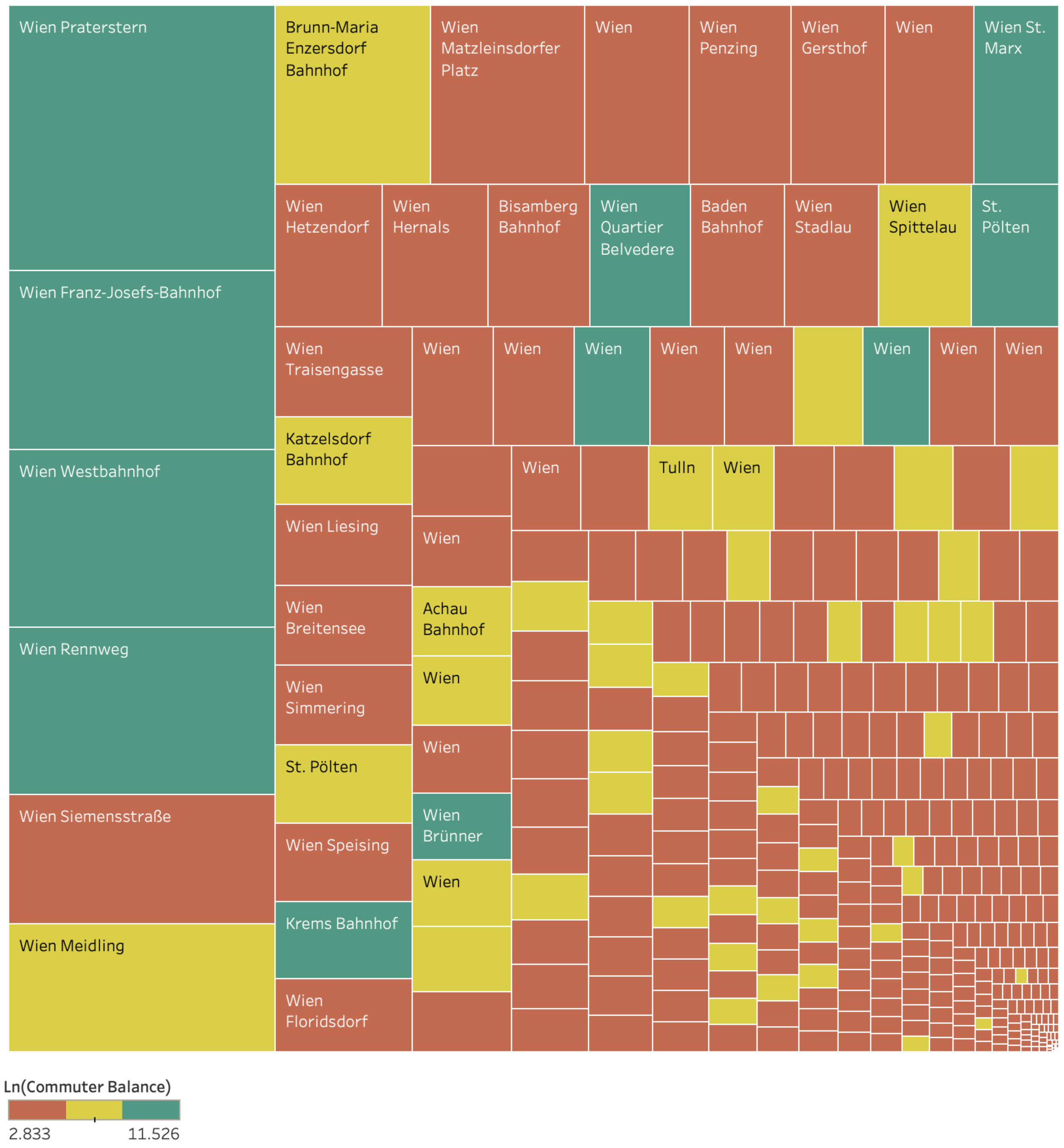

Figure 5: Biggest potential stations including Vienna

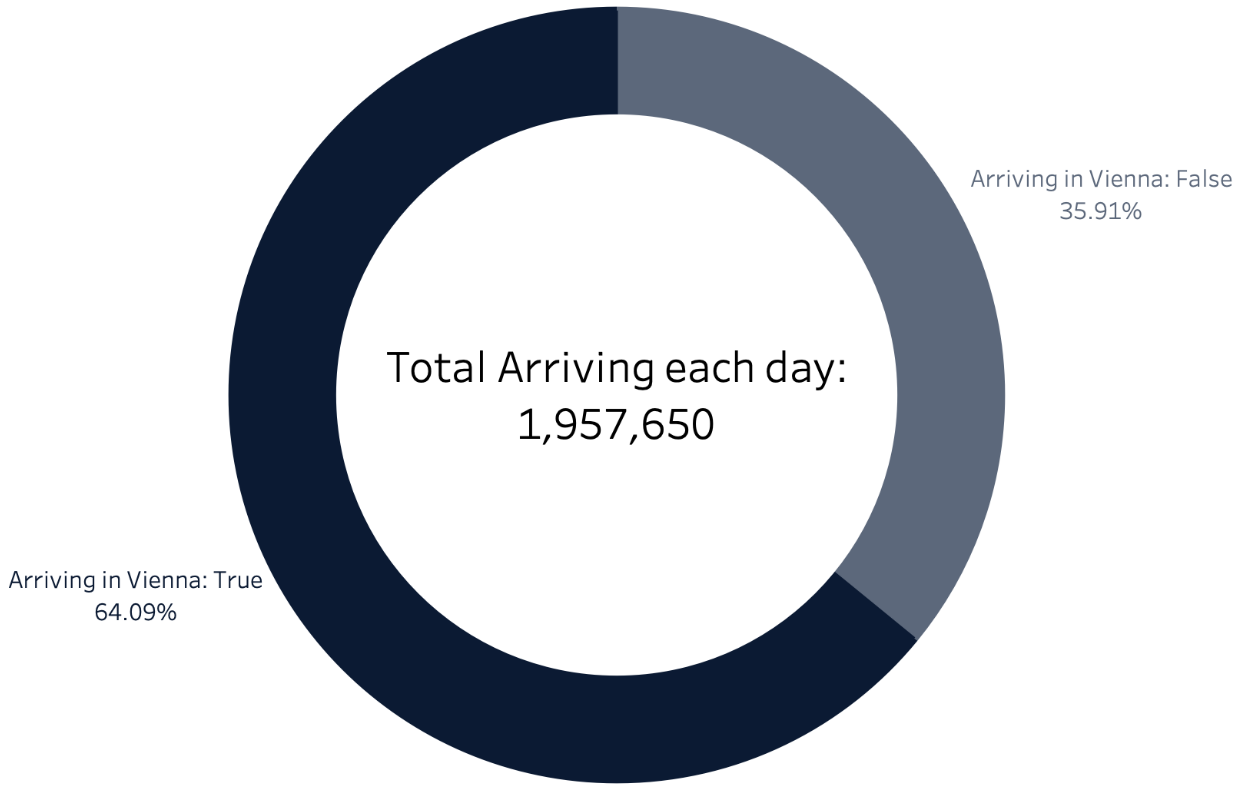

Figure 6: Commuters arriving in Vienna vs. outside of Vienna

```
    "trips": {
      "tripid": {
        "from": "str",
        "to": "str",
        "duration": "int",
        "stop_count": "int",
        "average_speed": "float",
        "via": [
          {
            "from": "str",
            "departure": "datetime",
            "to": "str",
            "arrival": "datetime",
            "duration": "int",
            "distance": "int",
            "speed": "float"
          }
        ]
      }
    },
    "duration": "int",
    "changeCount": "int",
    "changesAt": [{ "station": "str", "duration": "int", "sequence": "int" }]
  }
}
```

Such structure provided a way to calculate necessary routes once and contain all possibly necessary information for further calculations.

TODO: Until 13.12

Station to station amenities calcualtion

TODO: Until 13.12

Amenities per station score

Amenities at each station considered by the analysis (Vienna and Lower Austria only) were extracted using Python from Netex files available at https://data.oebb.at/de/datensaetze~netex-geodaten~.

Route score

Data was webscraped from vagonweb.cz. From 3748 trips handled by ÖBB in Vienna and Lower Austria, the enthustiast-run database contained data on 2092 of them (55.8%). The route score for train compositions was calculated based on data available from ÖBB https://data.oebb.at/de/datensaetze~fahrzeuge-personenverkehr~. Trips were attributed with following:

- CCTV
- Wheelchair place
- Bicycle place
- Air Conditioning
- WiFi
- Low Floor
- Year built

Route attribute assignment procedure was conducted using Python script below. Each trip from GTFS feed was retrieved from `json` file from vagonweb.cz. Data on composition were then retrieved and checked against official dataset of ÖBB. If composition contained certain amenity in any of carriages, it was marked as true, otherwise a false value was given.

```
import json
import pandas as pd
from collections import Counter
from tqdm import tqdm

path = "vagonweb/webscraping_result.json"
gtfs_path = "GTFS_Simplified/trips.txt"
def findGTFS() -> pd.DataFrame:
    trips = pd.read_csv(gtfs_path)
    trips["trip_short_name"] = trips["trip_short_name"].str.extract("(\d+)")
    return trips[["trip_short_name", "trip_id", "route_id"]]



def computeAttributes(data: dict) -> pd.DataFrame:
    attribute_list = []
    fahrzeuge = pd.read_csv("vagonweb/OBB Baureihen.csv")
    gtfs = findGTFS()
    for trip in tqdm(data, desc="Processing trips"):
        composition = data.get(trip)
        gtfs_trip_id = (
            gtfs[gtfs["trip_short_name"] == trip]["trip_id"].values[0]
            if not gtfs[gtfs["trip_short_name"] == trip].empty
            else None
        )
        gtfs_route_id = (
            gtfs[gtfs["trip_short_name"] == trip]["route_id"].values[0]
            if not gtfs[gtfs["trip_short_name"] == trip].empty
            else None
        )
        attr = {
            "trip": trip,
            "gtfs-trip-id": gtfs_trip_id,
            "gtfs-route-id": gtfs_route_id,
        }
        for carriage in composition:
            attr = {
                "trip": trip,
                "gtfs-trip-id": gtfs_trip_id,
                "gtfs-route-id": gtfs_route_id,
                "cctv": False,
                "Wheelchair": False,
                "Bicycle": False,
                "ac": False,
                "WiFi": False,
                "LowFloor": False,
                "Year": int,
            }
            attributes = fahrzeuge[fahrzeuge["Number"].isin([carriage])]
            if len(attributes) < 1:
                continue
            attr["ac"] = (
                True
                if attributes["Klimatisierter Fahrgastinnenraum"].values[0] == "ja"
                and ~attr["ac"]
                else False
            )
            attr["Bicycle"] = (
                True
                if attributes["Fahrradplätze"].values[0] == "ja" and ~attr["Bicycle"]
                else False
            )
            attr["cctv"] = (
                True
                if attributes["Videoüberwachung"].values[0] == "ja" and ~attr["cctv"]
                else False
            )
            attr["LowFloor"] = (
                True
                if attributes["Niederflurzustieg"].values[0] == "ja"
                and ~attr["LowFloor"]
                else False
            )
            attr["Wheelchair"] = (
                True
                if attributes["Rollstuhlplätze"].values[0] == "ja"
                and ~attr["Wheelchair"]
                else False
            )
            attr["WiFi"] = (
                True
                if attributes["W-LAN"].values[0] == "ja" and ~attr["WiFi"]
                else False
            )

        attribute_list.append(attr)
    return pd.DataFrame(attribute_list)


def main():
    with open(path, "r") as file:
        data = json.load(file)
    # computeStatistics(data)
    computeAttributes(data).to_csv("vagonweb/trip_attributes.csv")


if __name__ == "__main__":
    main()
```

Connection score Connection amenities were calculated based on all trips taken during a commute. They were then weighted based on travel duration on each train co create a score between 0 and 1, based on each trips' attributes. 0 meaning a complete lact of amenity during the travel and 1 meaning accessibility of amenity during the entire travel. Year built value was calculated as a weighted average and the rounded to a full year. If certain rolling stock data was not available for a connection, it was interpolated using an average score for all retrieved connections from vagonweb.cz. The average scores for all connections were:

CCTV	Wheelchair	Bicycle	AC	WiFi	Low Floor
0.6	0.77	0.97	0.88	0.27	0.6

TODO: Until 16.12

Overal route score calculation TODO: Until 16.12

Necessary switch

TODO: Until 19.12

Service frequency

TODO: Until 19.12

Time of arrival to make it on time

TODO: Until 19.12

Model outcomes

TODO: Until 01.01.2025

Bambrick, Glen. 2016. “What Is Hotspot Analysis?” Hotspot Analysis, Spatial Analysis. Geospatiality. January 21, 2016. https://glenbambrick.com/2016/01/21/what-is-hotspot-analysis/.