1_Presentation_Data

September 3, 2025

1 Capstone proposal by PATRICK ROMAND

2 Predicting Passenger Ridership on TPG Lines

[1]:
```
DOWNLOAD_DATA_Zenodo = True
DOWNLOAD_DATA_TPG = False
DOWNLOAD_DATA_METEO = False
INSTALL_LIB = True
```

3 1) The problem

Predicting Public Transport Ridership in Geneva Using Open Data and Machine Learning Techniques

Public transport services play a central role in sustainable urban mobility.
Understanding and anticipating ridership helps optimize resource planning, adjust supply to demand, and improve the user experience.

In this project, we focus on the Geneva Public Transport network (TPG).
The main goal is to predict **daily ridership** using historical data.
This forecasting task is complex because many factors influence it: weather conditions, day of the week, holiday periods, exceptional events, and more.

To better capture this complexity, we use a two-level approach:

Global level: predict the total daily **ridership** across the entire TPG network.
At this level, several Machine Learning models are compared to evaluate their performance and identify the most suitable approaches for this type of prediction.

Local level: focus on a specific stop in the network.
This part is presented as a concrete use case, showing how the model can be applied in operational situations with **passenger counts**.

This two-level approach highlights two complementary views:
an academic view, which compares different Machine Learning models, and a professional view, which presents a concrete use case for network management.

4 2) The data

4.1 (a) Clear overview of your data

The project relies on several data sources:

4.2 Main sources

- **TPG Open Data**
  Planned schedules, line and stop identifiers, GPS positions, ridership, available on the TPG Open Data portal.
- **Weather data**
  Temperature, precipitation, general conditions, obtained from weather sites such as Open-Meteo.

4.3 Secondary sources

- **Calendar data**
  Public holidays using the *holidays* library.
- **Real-time data** (Optional)
  Operational real-time data available via Open Transport Data Switzerland, not yet integrated but possible for future iterations.

4.3.1 TPG Data via API

From the TPG Open Data portal, data will be collected using the available API.

[2]:
```
# libraries
%matplotlib inline
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

import requests
```

[3]:
```
base_url  = "https://tpg.opendatasoft.com/api/explore/v2.1"
date_max  = "2025-06-22T23:59:59"
```

First, we need the list of available datasets:

[4]:
```
#get dataset list
API_endpoint_url = "/catalog/datasets"

url = base_url + API_endpoint_url
response = requests.get(url)
data = response.json()
for dataset in data["results"]:
    dataset_id = dataset.get("dataset_id")
    print(f"{dataset_id}")
```

```
carto-metro-couche-lignes
collisions-tpg-avec-tiers
montees-mensuelles-par-arret-par-ligne
arrets
montees-journalieres-par-arret-par-ligne-2019
kilometres-produits-journaliers-par-ligne
carto-metro-couche-point
mn_montees-par-arret-par-ligne-par-tranchehoraire
montees-par-arret-par-ligne
frequentation-journaliere-par-tranche-horaire
```

I will use the datasets **"arrets"** (stops) and **"montees-par-arret-par-ligne"** (boardings per stop per line)."

arrets:

[5]:
```
if DOWNLOAD_DATA_TPG:
    #get stops
    dataset_id = "arrets"
    API_endpoint_url = f"/catalog/datasets/{dataset_id}/exports/json"


    params = {
        "limit": -1,                # max 10000 par appel
        "lang": "fr",
        "timezone": "Europe/Zurich"
    }

    url = base_url + API_endpoint_url
    response = requests.get(url, params=params)

    data = response.json()
    print(f"Nombre de lignes : {len(data)}")


    data_df = pd.DataFrame(data)
    display(data_df.head())

    # Save data to file
    file_name = "arrets.csv"
    data_df.to_csv(file_name, index=False)

    del response
    del data
    del data_df
```

Les frequentations:

I will need the data for all lines, and then for specific lines or stops.
I will download the data for all lines, as well as the data for line 12.

[6]:
```
def download_data_TPG(filter_name, data_filter):
    dataset_id = "montees-par-arret-par-ligne"
    API_endpoint_url = f"/catalog/datasets/{dataset_id}/exports/json"

    where_clause = f"date <= '{date_max}'"
    if data_filter:
        where_clause += f" AND ligne in {data_filter}"

    params = {
        "limit": -1,                  # -1 no limit
        "lang": "fr",
        "timezone": "Europe/Zurich",
        "where": where_clause
    }

    url = base_url + API_endpoint_url
    response = requests.get(url, params=params)

    data = response.json()
    print(f"Nombre de lignes : {len(data)}")


    data_df = pd.DataFrame(data)
    # display(data_df.head())

    # Save data to file
    file_name = f"frequentations_{filter_name}.csv"
    data_df.to_csv(file_name, index=False)

    del response
    del data
    del data_df
```

[7]:
```
if DOWNLOAD_DATA_TPG:
    # download ligne 12
    filter_name = "ligne12"
    data_filter = "('12')"
    download_data_TPG(filter_name, data_filter)

    # download Tram
    # filter_name = "tram"
    # data_filter = "('12','14','15','17','18')"
    # download_data_TPG(filter_name, data_filter)
    # download All
    filter_name = "all"
    data_filter = None
    download_data_TPG(filter_name, data_filter)
```

If we look at the file sizes:
- **frequentations_ligne12.csv**: 11.1 MB
- **frequentations_tram.csv**: 64.4 MB
- **frequentations_all.csv**: 662.7 MB

And most importantly, it takes **30 minutes** to download *frequentations_all.csv*.

The TPG updated their data on **August 9**: the daily data for 2021 is no longer directly available, as it has been archived.
I kept my files from before August 2025 in order to upload them to the Zenodo platform.

```
[8]: import urllib.request
from pathlib import Path

if DOWNLOAD_DATA_Zenodo :
    FILES = {
    "arrets.csv": "https://zenodo.org/records/16880747/files/arrets.csv",
    "frequentations_ligne12.csv": "https://zenodo.org/records/16880747/files/
 ↪frequentations_ligne12.csv",
    "frequentations_all.2021.csv": "https://zenodo.org/records/16880747/files/
 ↪frequentations_all.2021.csv",
    "frequentations_all.2022.csv": "https://zenodo.org/records/16880747/files/
 ↪frequentations_all.2022.csv",
    "frequentations_all.2023.csv": "https://zenodo.org/records/16880747/files/
 ↪frequentations_all.2023.csv",
    "frequentations_all.2024.csv": "https://zenodo.org/records/16880747/files/
 ↪frequentations_all.2024.csv",
    "frequentations_all.2025.csv": "https://zenodo.org/records/16880747/files/
 ↪frequentations_all.2025.csv",
    "meteo_daily.csv": "https://zenodo.org/records/16880747/files/meteo_daily.
 ↪csv",
    "meteo_hourly.csv": "https://zenodo.org/records/16880747/files/meteo_hourly.
 ↪csv",
    }

    for name, url in FILES.items():
        urllib.request.urlretrieve(url, name)

    #concat with chatGPT
    recent = []
    for y in (2021, 2022, 2023, 2024, 2025):
        p = Path(f"frequentations_all.{y}.csv")  # fichier dans le répertoire␣
 ↪courant
```

```
        if p.exists():
            print(f"Lecture de {p}...")
            recent.append(pd.read_csv(p, low_memory=False))

    df = pd.concat(recent, ignore_index=True)

    # Save data to file
    file_name = f"frequentations_all.csv"
    df.to_csv(file_name, index=False)
```

```
Lecture de frequentations_all.2021.csv…
Lecture de frequentations_all.2022.csv…
Lecture de frequentations_all.2023.csv…
Lecture de frequentations_all.2024.csv…
Lecture de frequentations_all.2025.csv…
```

4.3.2 Weather data with Open-Meteo

The MeteoSwiss website is currently being migrated to the OGD platform, and the data is not yet available.
So I will use the website open-meteo.com.

Using https://www.gps-coordinates.net/, we can get the geographic coordinates of the city of Geneva:

DD (decimal degrees)
Latitude: 46.2017559
Longitude: 6.1466014

Lat, Long: 46.2017559, 6.1466014

DMS (degrees, minutes, seconds)
Latitude: N 46° 12’ 6.321’ ’
Longitude: E 6° 8’ 47.765’ ’

Then we can query the Open-Meteo website using the coordinates of Geneva:

```
[9]: if DOWNLOAD_DATA_METEO:
    base_url  = "https://archive-api.open-meteo.com"
    API_endpoint_url = "/v1/archive"

    params = {
        "latitude": 46.2022,
        "longitude": 6.1457,
        "start_date": "2021-08-01",
        "end_date": date_max.split("T")[0],
        "daily": "weather_code",
        "hourly": ["temperature_2m", "precipitation", "weather_code",␣
 ↪"wind_speed_10m"],
        "timezone": "auto"
    }

    url = base_url + API_endpoint_url
    # Appel API
    response = requests.get(url, params=params)
    #response.raise_for_status()  # pour voir les erreurs HTTP
    data = response.json()


    # Extract daily data
    daily_data = data.get("daily", {})
    daily_data_df = pd.DataFrame(daily_data)
    print("Daily data :")
    display(daily_data_df.head())

    # Extract hourly data
    hourly_data = data.get("hourly", {})
    hourly_data_df = pd.DataFrame(hourly_data)
    print("Hourly data:")
    display(hourly_data_df.head())

    # Save data to file
    file_name = "meteo_daily.csv"
    daily_data_df.to_csv(file_name, index=False)

    file_name = "meteo_hourly.csv"
    hourly_data_df.to_csv(file_name, index=False)

    del response
    del data, daily_data, hourly_data
    del daily_data_df, hourly_data_df
```

Clear variables to use a bit less memory:

```
[10]: import gc; gc.collect()
```

```
[10]: 0
```

4.4 (b) Plan to manage and process the data

4.5 Dataset

I have TPG data by stop, by line, and by day.
I will first run an EDA on the stops to get familiar with the network.

Then, I will run an EDA on the ridership of one line — line 12 — which I know well as a user.
After that, I will extend the analysis to all lines.

For the weather data, I have both hourly and daily values.

I can combine the two datasets: TPG (daily) and weather (daily).

4.6 Data Type

The time data needs to be converted to **Datetime**.

4.7 Data Processing Steps

4.7.1 Preliminary EDA

1. **Cleaning**
    - Handle missing values and duplicates

    - Data validation (features)
2. **Feature creation**
    - Network variables: type of vehicle
3. **Merging sources**
    - Join datasets (ridership, weather, calendar) using the date

4.7.2 Advanced EDA

4. **Temporal analysis**
    - Time-related variables: day of the week, month, holidays, vacation periods
5. **Preparation for modeling**
    - Encode categorical variables

    - Normalize numerical variables if needed

    - Split data into training and test sets based on time

[11]:
```
#end
```

[ ]:
```

```