5_ML_TPG_Stop

September 3, 2025

1 Machine Learning with Daily Boarding/Alighting Details by Stop

In this section, I aim to carry out a **practical case study**.
The objective is to answer a concrete question such as:

“How many boardings at stop *XXX* on line *YYY* on date *ZZZ*?”

This approach allows us to test a prediction model on a real and easily interpretable situation.

However, I was not able to implement a simple function that correctly recomputes the *lag* variables for the future period.

Therefore, I chose another approach: truncate the dataset up to **June 19, 2025**, estimate the number of boardings for **June 20, 2025**, and then compare this prediction with the actual value contained in the original dataset.

[1]:
```
INSTALL_LIB = False
```

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

2 1) Loading the data

[3]:
```
file_name = "TPG_daily_df.csv"

TPG_meteo_all_df = pd.read_csv(file_name,sep=",", low_memory=False)
TPG_meteo_all_df['date'] = pd.to_datetime(TPG_meteo_all_df['date'])
display(TPG_meteo_all_df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4794267 entries, 0 to 4794266
Data columns (total 33 columns):
 #   Column                         Dtype
---  ------                         -----
```

```
 0   date                         datetime64[ns]
 1   ligne                        object
 2   ligne_type_act               object
 3   horaire_type                 object
 4   arret_code_long              object
 5   indice_semaine               int64
 6   indice_jour_semaine          int64
 7   nb_de_montees                float64
 8   nb_de_descentes              float64
 9   annee                        int64
 10  mois                         int64
 11  log_montees                  float64
 12  log_descentes                float64
 13  frequentation_totale         float64
 14  log_frequentation_totale     float64
 15  delta_montees_descentes      float64
 16  log_delta_montees_descentes  float64
 17  type_vehicule                object
 18  weather_code                 int64
 19  montees_lag_1                float64
 20  log_montees_lag_1            float64
 21  desc_lag_1                   float64
 22  log_desc_lag_1               float64
 23  montees_lag_7                float64
 24  log_montees_lag_7            float64
 25  desc_lag_7                   float64
 26  log_desc_lag_7               float64
 27  rolling_montees_7d           float64
 28  log_rolling_montees_7d       float64
 29  rolling_desc_7d              float64
 30  log_rolling_desc_7d          float64
 31  est_weekend                  int64
 32  est_ferie                    int64
dtypes: datetime64[ns](1), float64(20), int64(7), object(5)
memory usage: 1.2+ GB

None
```

[4]:
```
TPG_meteo_all_df.head()
```

[4]:
```
        date ligne ligne_type_act horaire_type arret_code_long  \
0 2021-08-08     1      PRINCIPAL     DIMANCHE          31DC00
1 2021-08-09     1      PRINCIPAL     VACANCES          31DC00
2 2021-08-10     1      PRINCIPAL     VACANCES          31DC00
3 2021-08-11     1      PRINCIPAL     VACANCES          31DC00
4 2021-08-12     1      PRINCIPAL     VACANCES          31DC00

   indice_semaine  indice_jour_semaine  nb_de_montees  nb_de_descentes  annee  \
```

```
0            31                7         88.06          70.72   2021
1            32                1        192.44         204.33   2021
2            32                2        205.16         224.59   2021
3            32                3        187.67         217.00   2021
4            32                4        214.80         217.32   2021

   …  montees_lag_7  log_montees_lag_7  desc_lag_7  log_desc_lag_7  \
0  …          86.84           4.475517       81.99        4.418720
1  …         149.95           5.016949      164.72        5.110300
2  …         199.67           5.301662      223.32        5.413074
3  …         235.60           5.466371      205.42        5.329913
4  …         219.47           5.395762      208.25        5.343530

   rolling_montees_7d  log_rolling_montees_7d  rolling_desc_7d  \
0          174.415714                5.167159       173.398571
1          180.485714                5.201177       179.057143
2          181.270000                5.205489       179.238571
3          174.422857                5.167199       180.892857
4          173.755714                5.163389       182.188571

  log_rolling_desc_7d  est_weekend  est_ferie
0            5.161343            0          0
1            5.193274            0          0
2            5.194281            0          0
3            5.203418            0          0
4            5.210516            0          0

[5 rows x 33 columns]
```

2.0.1 Action :

To build the features for **June 20, 2025** and compare the final result,
I store the data for this date in a separate dataset.

[5]:
```
TPG_20250620 = TPG_meteo_all_df[(TPG_meteo_all_df["date"] == "2025-06-20")]

#Debug, on affiche les données du 20/06/2025 pour l'arret 'RIVE00' sur la ligne␣
↪12
TPG_20250620[(TPG_20250620["arret_code_long"] == "RIVE00") &␣
↪(TPG_20250620["ligne"] == "12")].head(1)
```

[5]:
```
              date ligne ligne_type_act horaire_type arret_code_long  \
3748340 2025-06-20    12      PRINCIPAL       NORMAL          RIVE00

         indice_semaine  indice_jour_semaine  nb_de_montees  nb_de_descentes  \
3748340              25                    5        3320.14          3309.28
```

```
         annee  …  montees_lag_7  log_montees_lag_7  desc_lag_7  \
3748340   2025  …        2960.69           7.993515     3356.01

         log_desc_lag_7  rolling_montees_7d  log_rolling_montees_7d  \
3748340        8.118806         2491.258571                7.820945

         rolling_desc_7d log_rolling_desc_7d  est_weekend  est_ferie
3748340      2701.362857            7.901882            1          0

[1 rows x 33 columns]
```

2.0.2 Action :

We truncate the dataset at **June 19, 2025** in order to exclude the day of June 20.

[6]:
```python
cutoff = pd.to_datetime("2025-06-19")
TPG_meteo_all_df_cut = TPG_meteo_all_df[TPG_meteo_all_df["date"] <= cutoff]
```

[7]:
```python
# descriptive statistics
TPG_meteo_all_df_cut.describe().T
```

[7]:
```
                              count         mean         std          min  \
indice_semaine            4784731.0    26.448799   15.265995     1.000000
indice_jour_semaine       4784731.0     3.901318    1.949375     1.000000
nb_de_montees             4784731.0   170.034720  429.162980     0.000000
nb_de_descentes           4784731.0   170.099195  426.501738     0.000000
annee                     4784731.0  2023.032681    1.185014  2021.000000
mois                      4784731.0     6.510411    3.514600     1.000000
log_montees               4784731.0     3.176362    2.185523     0.000000
log_descentes             4784731.0     3.224178    2.158247     0.000000
frequentation_totale      4784731.0   340.133915  803.560279     0.000000
log_frequentation_totale  4784731.0     6.400540    4.004097     0.000000
delta_montees_descentes   4784731.0    -0.064475  294.040933 -10918.140000
log_delta_montees_descentes 4784731.0  -0.047816    1.684128    -9.086355
weather_code              4784731.0    31.431903   28.224605     0.000000
montees_lag_1             4784731.0   169.931735  428.985746     0.000000
log_montees_lag_1         4784731.0     3.175562    2.185451     0.000000
desc_lag_1                4784731.0   169.995231  426.327290     0.000000
log_desc_lag_1            4784731.0     3.223368    2.158187     0.000000
montees_lag_7             4784731.0   169.558615  428.331599     0.000000
log_montees_lag_7         4784731.0     3.172503    2.185337     0.000000
desc_lag_7                4784731.0   169.626037  425.665579     0.000000
log_desc_lag_7            4784731.0     3.220279    2.158180     0.000000
rolling_montees_7d        4784731.0   169.818729  412.354260     0.000000
log_rolling_montees_7d    4784731.0     3.299882    2.103777     0.000000
rolling_desc_7d           4784731.0   169.882173  409.400698     0.000000
log_rolling_desc_7d       4784731.0     3.351139    2.070083     0.000000
```

```
est_weekend                 4784731.0     0.295849    0.456424      0.000000
est_ferie                   4784731.0     0.000000    0.000000      0.000000

                                   25%          50%          75%  \
indice_semaine               13.000000    26.000000    40.000000
indice_jour_semaine           2.000000     4.000000     6.000000
nb_de_montees                 2.840000    22.540000   137.550000
nb_de_descentes               3.000000    24.790000   138.170000
annee                      2022.000000  2023.000000  2024.000000
mois                          3.000000     6.000000    10.000000
log_montees                   1.345472     3.158701     4.931231
log_descentes                 1.386294     3.249987     4.935696
frequentation_totale         11.730000    64.150000   308.300000
log_frequentation_totale      3.178054     6.185255     9.514157
delta_montees_descentes     -29.860000     0.000000    27.260000
log_delta_montees_descentes  -1.045502     0.000000     0.976823
weather_code                  3.000000    51.000000    61.000000
montees_lag_1                 2.830000    22.520000   137.430000
log_montees_lag_1             1.342865     3.157851     4.930365
desc_lag_1                    3.000000    24.760000   138.050000
log_desc_lag_1                1.386294     3.248823     4.934834
montees_lag_7                 2.820000    22.420000   137.010000
log_montees_lag_7             1.340250     3.153590     4.927326
desc_lag_7                    3.000000    24.660000   137.610000
log_desc_lag_7                1.386294     3.244933     4.931664
rolling_montees_7d            3.614286    25.345714   143.357143
log_rolling_montees_7d        1.529157     3.271306     4.972290
rolling_desc_7d               3.861429    28.011429   145.435714
log_rolling_desc_7d           1.581332     3.367690     4.986587
est_weekend                   0.000000     0.000000     1.000000
est_ferie                     0.000000     0.000000     0.000000

                                    max
indice_semaine                52.000000
indice_jour_semaine            7.000000
nb_de_montees               9821.710000
nb_de_descentes            10944.990000
annee                       2025.000000
mois                          12.000000
log_montees                    9.192452
log_descentes                  9.300728
frequentation_totale       17919.470000
log_frequentation_totale      18.195097
delta_montees_descentes     8326.610000
log_delta_montees_descentes    8.885554
weather_code                  75.000000
montees_lag_1               9821.710000
```

```
log_montees_lag_1                  9.192452
desc_lag_1                     10944.990000
log_desc_lag_1                     9.300728
montees_lag_7                   9821.710000
log_montees_lag_7                  9.192452
desc_lag_7                     10944.990000
log_desc_lag_7                     9.300728
rolling_montees_7d              8453.642857
log_rolling_montees_7d             9.042471
rolling_desc_7d                 9086.812857
log_rolling_desc_7d                9.114690
est_weekend                        1.000000
est_ferie                          0.000000
```

[8]:
```python
print(f"The data covers the date range from {TPG_meteo_all_df_cut['date'].
 ↪min()} to {TPG_meteo_all_df_cut['date'].max()}")
```

```
The data covers the date range from 2021-08-08 00:00:00 to 2025-06-19 00:00:00
```

[9]:
```python
df = TPG_meteo_all_df_cut.copy()
```

3 2) Splitting the data

[10]:
```python
from datetime import datetime

# Définition des bornes
train_end = pd.to_datetime("2023-12-31")
val_end = pd.to_datetime("2024-06-30")

# Split temporel
train_df = df[df["date"] <= train_end]
val_df = df[(df["date"] > train_end) & (df["date"] <= val_end)]
test_df = df[df["date"] > val_end]
```

4 3) Target and features

4.0.1 Action :

We remove the raw values from the dataset and keep only their log-transformed versions.

[11]:
```python
drop_cols = [
    "date",         # index temporel
    "nb_de_montees", "nb_de_descentes",
    "log_montees", # cible en log

    "frequentation_totale",
```

```
    "delta_montees_descentes",
    "montees_lag_1",
    "desc_lag_1",
    "montees_lag_7",
    "desc_lag_7",
    "rolling_montees_7d",
    "rolling_desc_7d"
]
```

[12]:
```
X_tr = train_df.drop(columns=drop_cols)
y_tr = train_df["log_montees"]

X_val = val_df.drop(columns=drop_cols)
y_val = val_df["log_montees"]

X_te = test_df.drop(columns=drop_cols)
y_te = test_df["log_montees"]
```

5 4) Model training

[13]:
```
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

def compute_statistics(model_name, y_te, y_te_pred_log) :
    mae_log = mean_absolute_error(y_te, y_te_pred_log)
    rmse_log = np.sqrt(mean_squared_error(y_te, y_te_pred_log))
    r2_log = r2_score(y_te, y_te_pred_log)

    print(f"{model_name} (log space):")
    print(f"  MAE : {mae_log:.4f}")
    print(f"  RMSE: {rmse_log:.4f}")
    print(f"  R²  : {r2_log:.4f}")


    y_te_pred_real = np.exp(y_te_pred_log)
    y_te_real = np.exp(y_te)


    mae_real = mean_absolute_error(y_te_real, y_te_pred_real)
    rmse_real = np.sqrt(mean_squared_error(y_te_real, y_te_pred_real))
    r2_real = r2_score(y_te_real, y_te_pred_real)

    print("\n")
    print(f"{model_name} (real space):")
    print(f"  MAE : {mae_real:.4f}")
    print(f"  RMSE: {rmse_real:.4f}")
    print(f"  R²  : {r2_real:.4f}")
```

```python
    return mae_log, rmse_log, r2_log, mae_real, rmse_real, r2_real
```

[14]:
```python
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler

# Colonnes catégorielles (OneHot)
cat_features = [
    "ligne",              # identifiant ligne
    "ligne_type_act",     # type de ligne
    "horaire_type",       # horaire (école, vacances, etc.)
    "arret_code_long",    # arrêt
    "indice_semaine",
    "indice_jour_semaine",
    "annee",
    "mois",
    "type_vehicule",      # tram, bus, etc.
    "weather_code",        # météo catégorielle
    "est_weekend",
    "est_ferie"
]

# Colonnes numériques (scalées)
num_features = [
    "log_descentes",
    "log_frequentation_totale",
    "log_delta_montees_descentes",
    "log_montees_lag_1",
    "log_desc_lag_1",
    "log_montees_lag_7",
    "log_desc_lag_7",
    "log_rolling_montees_7d",
    "log_rolling_desc_7d"
]

# Define the transformer
cat_transformer = Pipeline([("onehot", OneHotEncoder(handle_unknown='ignore'))])
num_transformer = Pipeline([("scaler", StandardScaler())])

# Apply
preprocessor = ColumnTransformer(
    transformers=[
        ('cat', cat_transformer, cat_features),
        ('num', num_transformer, num_features)
    ]
```

```
)
```

[15]:

```python
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import TimeSeriesSplit
import joblib
import time

# Create the pipeline
pipe_ridge = Pipeline([("preprocessor", preprocessor), ("ridge", Ridge())])

# Create cross-validation object
cv = TimeSeriesSplit(n_splits=5)

# Create grid for alpha
grid = {"ridge__alpha": np.logspace(-4, 4, num=20)}

# Create the grid search object
model_ridge = GridSearchCV(
    pipe_ridge,
    grid,
    cv=cv,
    return_train_score=True,
    scoring="neg_mean_absolute_error",
    verbose=1,
)

# Fit on the training set
start_time = time.time()
model_ridge.fit(X_tr, y_tr)

end_time = time.time()
print(f"Total execution time: {end_time - start_time:.2f} seconds")

print("Best Params:", model_ridge.best_params_)
print("Best Score:", -model_ridge.best_score_)

# save the model
# joblib.dump(model_ridge, "ridge_pipeline_log.pkl")
```

```
Fitting 5 folds for each of 20 candidates, totalling 100 fits
Total execution time: 451.12 seconds
Best Params: {'ridge__alpha': 0.0006951927961775605}
Best Score: 0.011389699496520364
```

[16]:

```python
# Evaluate on the test set
y_te_pred_log = model_ridge.predict(X_te)
```

```
ridge_model_name = "ridge regression"
ridge_mae_log, ridge_rmse_log, ridge_r2_log,ridge_mae_real, ridge_rmse_real,␣
↪ridge_r2_real = compute_statistics(ridge_model_name, y_te, y_te_pred_log)
```

```
ridge regression (log space):
  MAE : 0.0118
  RMSE: 0.0157
  R²  : 0.9999


ridge regression (real space):
  MAE : 1.6656
  RMSE: 5.2803
  R²  : 0.9999
```

6 5) Usage: predicting boardings for a stop

6.0.1 Observation :

We build an example of how to use the model to answer the following question:
"How many boardings will there be on 20/06/2025 at stop RIVE00 on line 12?"

6.0.2 Action :

We need to build the data row in the same format as the **X_tr** dataset in order to feed it into the model.

[17]:
```
# X_tr.info()
```

[18]:
```
from datetime import timedelta

# Colonnes attendues par ton modèle (ordre fixe)
REQ_COLS = [
    "ligne","ligne_type_act","horaire_type","arret_code_long",
    "indice_semaine","indice_jour_semaine","annee","mois",
    "log_descentes","log_frequentation_totale","log_delta_montees_descentes",
    "type_vehicule","weather_code",
    "log_montees_lag_1","log_desc_lag_1","log_montees_lag_7","log_desc_lag_7",
    "log_rolling_montees_7d","log_rolling_desc_7d",
    "est_weekend","est_ferie"
]

def make_X_pred(arret, ligne):
    # TPG_20250620 contient les données de tous les arrets et ligne pour la␣
↪journée du 20 juin.
    # il faut donc filtrer par arret et ligne
```

```
    Next_TPG = TPG_20250620[(TPG_20250620["arret_code_long"] == arret) &␣
↪(TPG_20250620["ligne"] == ligne)]

    row = {
        "ligne": Next_TPG["ligne"].iloc[0],
        "ligne_type_act": Next_TPG["ligne_type_act"].iloc[0],
        "horaire_type": Next_TPG["horaire_type"].iloc[0],
        "arret_code_long": Next_TPG["arret_code_long"].iloc[0],

        "indice_semaine": Next_TPG["indice_semaine"].iloc[0],
        "indice_jour_semaine": Next_TPG["indice_jour_semaine"].iloc[0],
        "annee": Next_TPG["annee"].iloc[0],
        "mois": Next_TPG["mois"].iloc[0],

        "log_descentes": Next_TPG["log_descentes"].iloc[0],
        "log_frequentation_totale": Next_TPG["log_frequentation_totale"].
↪iloc[0],
        "log_delta_montees_descentes": Next_TPG["log_delta_montees_descentes"].
↪iloc[0],

        "type_vehicule": Next_TPG["type_vehicule"].iloc[0],
        "weather_code": Next_TPG["weather_code"].iloc[0],

        "log_montees_lag_1": Next_TPG["log_montees_lag_1"].iloc[0],
        "log_desc_lag_1": Next_TPG["log_desc_lag_1"].iloc[0],
        "log_montees_lag_7": Next_TPG["log_montees_lag_7"].iloc[0],
        "log_desc_lag_7": Next_TPG["log_desc_lag_7"].iloc[0],
        "log_rolling_montees_7d": Next_TPG["log_rolling_montees_7d"].iloc[0],
        "log_rolling_desc_7d": Next_TPG["log_rolling_desc_7d"].iloc[0],

        "est_weekend": Next_TPG["est_weekend"].iloc[0],
        "est_ferie": Next_TPG["est_ferie"].iloc[0]
    }

    X_pred = pd.DataFrame([[row.get(c, np.nan) for c in REQ_COLS]],␣
↪columns=REQ_COLS)
    return X_pred
```

[19]:
```
# Génération de X_pred pour RIVE00 / ligne 12
arret_pred = "RIVE00"
ligne_pred = "12"
X_pred = make_X_pred(arret = arret_pred, ligne= ligne_pred)

display(X_pred.tail(7))
display(X_pred.shape)
```

```python
# Prédiction
y_pred_log = model_ridge.predict(X_pred)[0]
y_pred = np.exp(y_pred_log)
print("y_pred_log:", y_pred_log)
print("exp(y_pred_log):", np.exp(y_pred_log))

print(f"Prediction (RIVE00, L12 ) : {y_pred:,.0f} montées")
```

```
  ligne ligne_type_act horaire_type arret_code_long  indice_semaine  \
0    12      PRINCIPAL       NORMAL          RIVE00              25

   indice_jour_semaine  annee  mois  log_descentes  log_frequentation_totale  \
0                    5   2025     6       8.104788                 16.212851

   … type_vehicule weather_code  log_montees_lag_1  log_desc_lag_1  \
0  …          Tram            3           7.897572        8.046082

   log_montees_lag_7  log_desc_lag_7  log_rolling_montees_7d  \
0           7.993515        8.118806                7.820945

   log_rolling_desc_7d  est_weekend  est_ferie
0             7.901882            1          0

[1 rows x 21 columns]
```

```
(1, 21)
```

```
y_pred_log: 8.117893653279854
exp(y_pred_log): 3353.9487207395387
Prediction (RIVE00, L12 ) : 3,354 montées
```

7 6) Verification

[20]:
```python
from math import fabs
Next_TPG = TPG_20250620[(TPG_20250620["arret_code_long"] == arret_pred) &␣
 ↪(TPG_20250620["ligne"] == ligne_pred)]
y_real = Next_TPG["nb_de_montees"].iloc[0]
y_pred

abs_err = fabs(y_pred - y_real)
mape = abs_err / y_real * 100

print(f"Question: boardings at RIVE00 (L12) on 20/06/2025?")
print(f"Model prediction : {y_pred:,.2f}")
print(f"Actual value     : {y_real:,.2f}")
print(f"Absolute error   : {abs_err:,.2f}  ({mape:.2f} %)")
```

```
Question: boardings at RIVE00 (L12) on 20/06/2025?
```

```
Model prediction : 3,353.95
Actual value     : 3,320.14
Absolute error   : 33.81  (1.02 %)
```

La prediction nous donne 3354 montées contre 3320 pour le réél. Soit un écart de 1%

[21]:
```
# 7) Conclusion
```

This case study illustrates how to use a regression model to predict the number of boardings at a specific stop,
on a given date. In our example (RIVE00, line 12, June 20, 2025), the prediction is close to the actual value,
highlighting the model's ability to provide relevant estimates.

Although the approach remains simplified, it clearly demonstrates the practical application of the model trained on TPG data.

[22]:
```
#end
```