Schiphol-Code-Assignment Documentation

Release 0.1

Lodewic van Twillert, Qualogy

Jun 28, 2020

Contents

Contents:

CHAPTER 1

Output overview

All output can be found on Google Cloud Storage where files are created using a snakemake pipeline which was ran locally.

1.1 Data trelliscopes

As an exploratory step we can browse the data using this trelliscope,

Exploration trelliscope

– NOTE: Errors when rendering this external .html file(!) Any tips about how to include external .html files like this are welcome. –

For now, download the trelliscope as a .zip file from Google Storage bucket: https://storage.googleapis.com/lvt-schiphol-assignment-snakemake/trelliscopes.zip

Unzip the folder and open the `index.html` file to view the trelliscope.

1.2 Snakemake pipeline overviews

Be sure to open these to understand the steps in the pipeline and the order in which notebooks are executed.

- DAG
- Rulegraph
- Filegraph
- Report

The Snakemake pipeline currently looks as follows,

CHAPTER 2

Project summary

2.1 Project goal

Predictive modeling of delays of departing aircraft at Schiphol. We are provided with 2 raw data files of flights and airports.

This project assignment is used to,

1. Display technical skills
2. Understanding and flexibility when handling new datasets
3. Make a prediction model of flight delays

While the assignment description specifically calls for a prediction model and data understanding, I have focussed mostly on **1.: Display technical skills.**.

I have taken the liberty to deviate quite a bit from the assignment and understand that especially requirement **2. Understanding new dataset** is largely skipped in my results. Allow me to motivate why I have made this decision

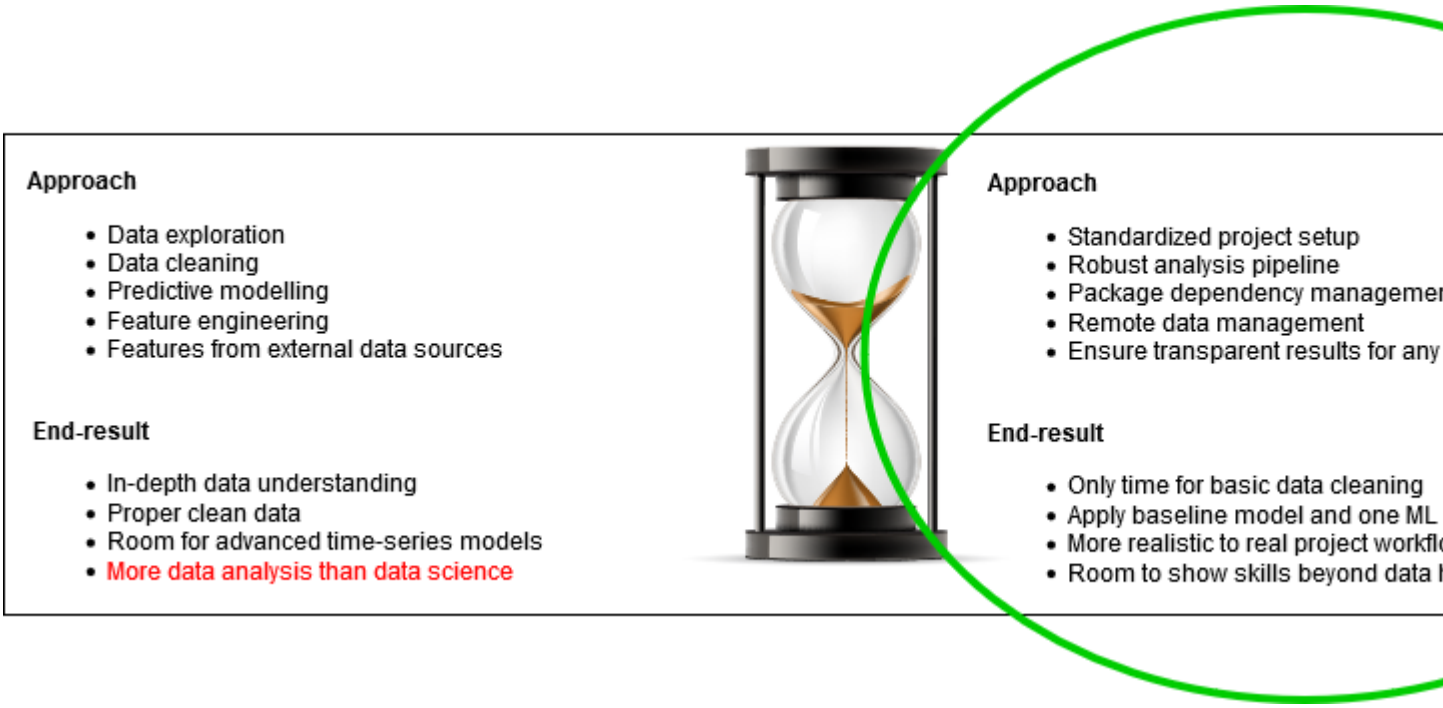
from the assignment.

2.2 Motivation for not sticking to assignment requirements

Because this assignment is a code assessment rather than a true business case I have taken the liberty of deviating from the requirements. My focus was to prove my technical skills more so than my data exploration or modelling skills. I believe that with my current results I can show my technical strengths in a way that differentiates my approach from others.

Because of time constraints I had to choose between,

1. Spend time focussing on data understanding and creating a good prediction model
2. Spend time focussing on what I believe is important project setup



I have chosen to spend most time on **2.**. Because I have not developed a deep data understanding I also do not have a great model performance. However, I do have a project setup in which adding a new better model is a breeze which would make this project much more robust in the future when moving to production.

2.2.1 Unoriginal results from previous internal results

In this assignment I felt I had a conflict of interest because my team internally already did quite some exploratory analysis of the flights data and the Schiphol API's before this assignment. We have weekly demo's so I am kind of familiar with the datasets and questions they already asked, though I did not write the code for them at the time. Then I noticed I was writing code to reproduce the answers to questions they had already answered, which I don't think would highlight my technical skill.

The first steps I took were to look for their notebooks for data understanding and API descriptions, then I realized that any exploratory analyses, external datasets or models would not be my ideas. So I only did basic exploration and modeling, and instead treated this assignment as the start of a new project within a team of data scientists.

The first steps I would take for a new project is to apply structure in a way that allows for continuous data science, where we present results to the business as soon as possible even when the results are still in an early phase and development is required to achieve good models.

A handful of strange things found in the data,

- Exceptional number of samples on some 1st day of the months
- Two timezones in the data at Schiphol we may have to deal with

2.2.2 Go beyond data analysis

The question I was asked to answer was to create a prediction model. As a data analysis project I could deep-dive into the data and make a single-use analysis and show that I could make predictions on the dataset I was given, possibly collect data from Schiphol API's to enrich the feature set and improve the results. Traditionally I would do exactly that, with a lot of inspiration from existing examples on Kaggle and blogs.

From experience I have noticed how important good project setup can be. An efficient workflow takes time to setup but will benefit a single data scientist, a team of us and especially the business. To display my coding skills I chose to setup a data analysis pipeline with,

- Package dependency management
- R and Python code use
- Reproducible Snakemake pipeline
- Cover all standard steps in modelling pipeline

From here I can show how easy it is to now add additional models, features from external data, analysis results, any steps you can write in a notebook.

2.2.3 My differentiating strength

Having worked together with data scientists in multiple teams both internally and as a consultant I have noticed that you'll find team members who are strong at modelling, visualization, dashboarding, building API's or even efficiently using unit-tests. What I was missing was motivated knowledge of project management where,

1. analysis results are reproducible
2. data is reusable across different steps in the analysis pipeline
3. models are saved from where they can be deployed

I have since taken an interest in ensuring reproducibility, traceability of results and an efficient development setup. By efficient development setup I mean that other contributing data scientists can painlessly add their ideas to the project. Now, any new idea can be implemented in a dedicated notebook that is easy to integrate in a larger pipeline. Each notebook under `/scripts` represents an analysis step, relying on previously created data if possible but not necessarily.

2.3 Approach

Predicting aircraft delays at an airport is a core problem for the airport that may affect flights scheduling, managing flow of people, departure gate assignments, etc. Because it is a complex problem to predict airplane delays effectively I have created a project workflow in which a data scientist or a team of data scientists can develop quite painlessly. Traditionally I would approach an analysis assignment like this with a handful of notebooks that explore the data

in-depth, then use that knowledge to preprocess the specific dataset I was handed and then create a model to prove how good I can make predictions. together and contribute models to an analysis pipeline.

The idea is that when you reach this stage quickly, it becomes easy to communicate new results to the business in a way that they can contribute ideas because your results are easily viewable. Combined with the fact that each analysis step is a concise rendered notebook with results that are understandable by both business and development interest.

With the current project setup I have built it has become easy to add new models and analysis steps to the pipeline while reusing earlier results, such as calculated features or train/test set splitting. All used notebooks are self-documenting and included on Git, and rendered on readthedocs.

2.4 What I focused on

2.4.1 File directory setup

Have a look at the README on GitHub for a more detailed file directory setup.

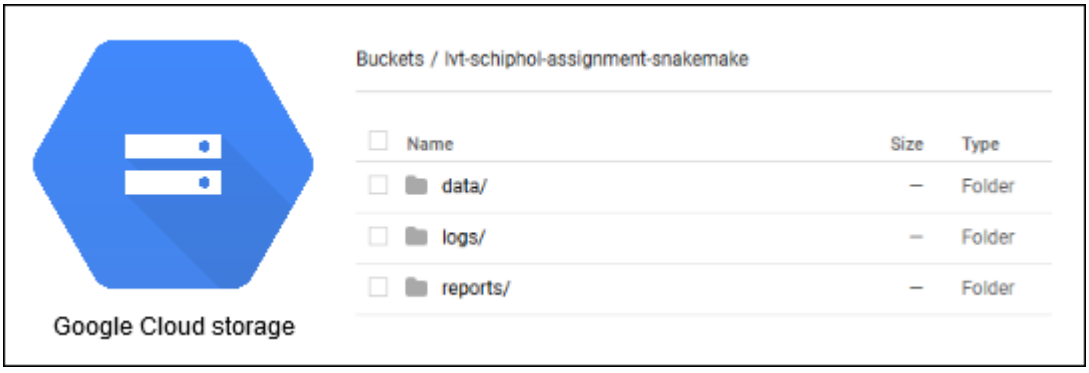
Steps taken,

- Use cookiecutter from drivendata, https://drivendata.github.io/cookiecutter-data-science/
- Scripts in the project are notebooks that can be executed with papermill
  - Note how papermill notebooks are similar to parameterized notebooks on DataBricks.
- Data is on Google Cloud Storage, with public read-access annd private write-access.
- Use `Snakemake` to structure output files, easily use local, remote bucket or even Azure DataLake
- Sphinx documentation from files under `/src` and rendered example notebooks from `/scripts`

2.4.2 Remote data source

I used Google Cloud Storage on a personal account and created a storage bucket. All project data including analysis results, rendered notebooks, models, etc. is all in the bucket!

Browse the data here: https://console.cloud.google.com/storage/browser/lvt-schiphol-assignment-snakemake/



You may need to login to GCP, but you do not need specific project access.

Steps taken,

- Create storage bucket in new GCP project
- Set public read-access and create service-account for write-access

- Define `Snakemake` pipeline to manage all output data files and structure
  - Manual data changes causes the pipeline to re-run analysis steps because files go out-of-sync
  - All data on the bucket expected to be created from pipeline, similar to managed datasets on DataLake

2.4.3 Reproducible analysis pipeline

For this I use Snakemake! Since my job at TNO I have kept an eye on Snakemake, which is sadly still not working smoothly on Windows. It is simply a Python package for which I think the syntax is easy to read and the output results are sufficient for a project like this.

As per snakemake,

```
The Snakemake workflow management system is a tool to create reproducible and␣
↪scalable data analyses. Workflows are described via a human readable, Python based␣
↪language. They can be seamlessly scaled to server, cluster, grid and cloud␣
↪environments, without the need to modify the workflow definition. Finally,␣
↪Snakemake workflows can entail a description of required software, which will be␣
↪automatically deployed to any execution environment.
```

Steps taken,

- (!) Fork Snakemake and fix recent bug on Windows, https://github.com/Lodewic/snakemake
  - Even in the Docker container we'll install my own version of the package
- Create Snakefile with pipeline definition
- Add config.yml with relevant settings
- Authenticate with Google Cloud storage to sync pipeline output
- Create rules for each script under `/scripts`
- Create multiple conda environment.yml files for different types of scripts

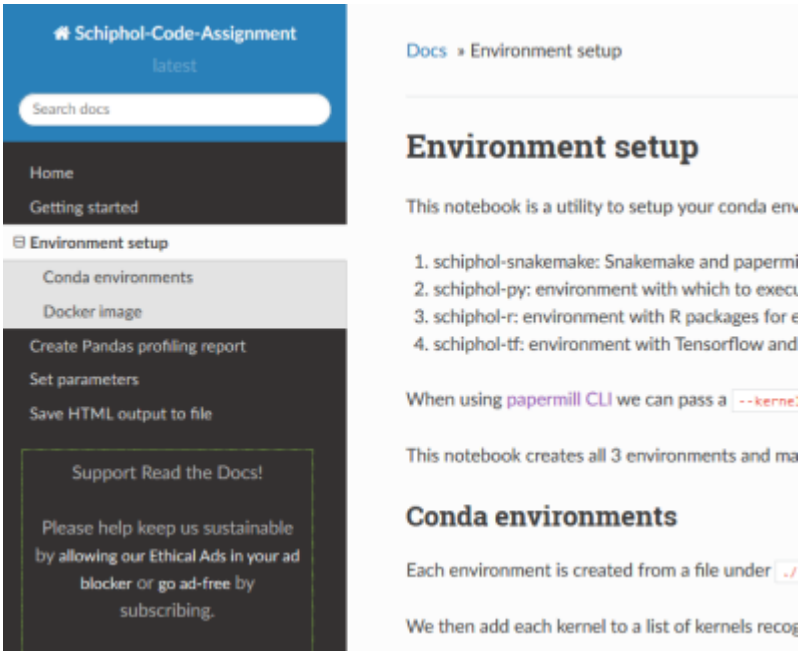
Additionally you can find more output here,

- DAG
- Rulegraph
- Filegraph
- Report

The Snakemake pipeline currently looks as follows,

2.4.4 Sphinx documentation

You can find the full documentation here, https://schiphol-assignment.readthedocs.io/en/latest/



Note that I have played with Sphinx earlier, but never to a stage of published pages. This will document the project progress, links to relevant files and most importantly code documentation.

Steps taken,

- Setup Sphinx project under `/docs`
- Set content and config of documentation pages, including rendered notebooks
- Setup build pipeline from GitHub on readthedocs.io
- Publish pages whenever master branch changes

You'll find that the `/docs` directory is fully functional and will allow you to create a basic documentation page. Cool thing is that notebooks are also included in the documentation pages, which then easily integrates our notebooks we use as scripts under `/scripts`. These script notebooks all expect a certain format, namely a description of the notebook, the input parameters and the expected output files.

Sphinx docs will include both the python modules under `/src` and the applied notebooks under `/scripts`.

2.5 What I left out

Due to time constraints I have not focussed on the data as much for the assignment, as stated earlier. So what you may be missing is,

- In-depth data analysis
- Pretty plots
- Display of object-oriented programming
- Connecting to external datasets
- Complex time-series models

CHAPTER 3

Exploration summary

As noted this part is very basic.

A great tool for a quick display of the data as first step when receiving the data.

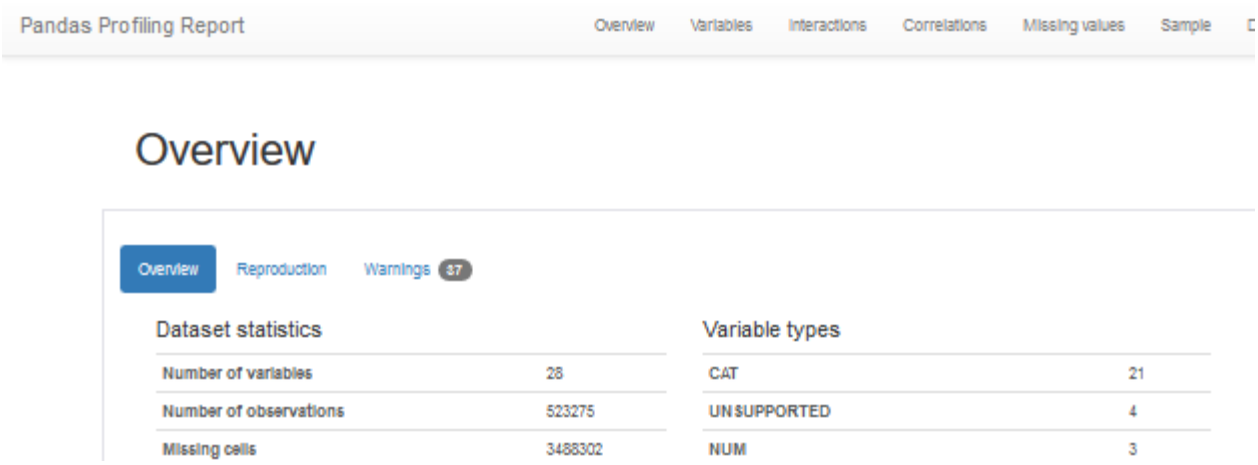
Creating these profiling reports is an easy first step, and the notebooks that create them are a good first example of using notebooks as papermill scripts.

Pandas profiling script

Output HTML files are saved to our remote bucket and is publicly available,

First thing we made was a profiling report of the data for quick results. Luckily the data is already fairly clean from the start.

Airports data profiling report HTML Flights data profiling report HTML



This summary of the flights data already teaches us a few basic things,

- Missing values of `actualOffBlockTime` where we cannot calculate the delay
- Duplicated ID values in the raw data!

- 97.2% of all flights is `serviceType==J` , passenger flights
- Some columns contain no data at all
- And more:)

There are columns which we should not use for delay prediction which I believe to only be known After the delay is known or a flight is even canceled.

- expectedTimeBoarding, expectedTimeGateClosing, expectedTimeGateOpen
  - No idea when these values are determined, but unlikely that these values are known/equal 2-hours before the scheduled flight
  - Therefore these columns not considered for prediction now

We will consider these findings downstream when we process the data for model input. Scripts to perform preprocessing are documented themselves so for the implementation of handling the data please look at the scripts.

More points to take with us from here,

- ID's are unique despite duplicates in the flights data
- By far the most flights are of `serviceType` J for Passengers
- A lot of airlines only have a handful of flights
- A lot of destinations only have a handful of flights
- Clearly categorical data with many small levels is an issue that needs to be dealt with
- Delays are anywhere from 1-hour too early to over 10 hours, with a very skewed distribution
  - Requires work to find fitting error metric for model evaluation
- Creating time-series data without data leakage will be a challenge!
  - For every flight, we know after the fact what the delay was. Be careful not to take the actual delay when at the time of prediction a plane is simply 'still delayed' but it may be unkown for how long.

The model scripts are self-documenting and examples can be found on readthedocs, rendered on Google Cloud storage under `{bucket}/data/model_output/{model_name}/{model_script}.ipynb`.

- Baseline average model
- Basic catboost model
- TODO: chained catboost model - or any other model
  - Predict quantile of delay
  - Separate within-quantile models trained and normalized within range of given quantile
  - New prediction first predicts the quantile - then makes a regression output based on that quantile's model

[ ]:

CHAPTER 4

Environment setup

This notebook is a utility to setup your conda environments for local development. For this project we will use 4 conda environments,

1. schiphol-snakemake: Snakemake and papermill to execute notebooks as scripts in a pipeline
2. schiphol-py: environment with which to execute Python notebooks with datascience tools like pandas, sklearn, xgboost, etc.
3. schiphol-r: environment with R packages for exploratory analyses and R time-series forecasting
4. schiphol-tf: environment with Tensorflow and Tensorflow-probability separate from other packages to avoid conflicts

When using papermill CLI we can pass a `--kernel` argument that specifies a kernel to use for executing a target notebook. Together with conda we can make our environments available as a `kernel`, but this requires some repetitive setup.

This notebook describes how to install all conda environments and make them available as a kernel.

4.1 Conda environments

Each environment is created from a file under `./envs/`.

We then add each kernel to a list of kernels recognized by jupyter, so that `papermill` can run notebooks with specified conda environments.

[17]:
```
from pathlib import Path
```

[16]:
```
env_file_dir = "./envs/"
conda_envs = ["schiphol-snakemake", "schiphol-py", "schiphol-r", "schiphol-tf"]

for conda_env in conda_envs:
    env_file = Path(env_file_dir, conda_env + ".yml")
    print(f"conda env create -f {env_file_dir}{conda_env}.yml")
```

(continues on next page)

(continued from previous page)

```
print()

for conda_env in conda_envs:
    env_file = Path(env_file_dir, conda_env + ".yml")

    print(
          f"conda deactivate\n"
          f"conda activate {conda_env}\n"
          f'python -m ipykernel install --user --name {conda_env} --display-name
↪"Python ({conda_env})"\n'
          f"conda deactivate\n"
    )

# check if all 4 environments are added to the kernels
!jupyter kernelspec list
```

```
conda env create -f ./envs/schiphol-snakemake.yml
conda env create -f ./envs/schiphol-py.yml
conda env create -f ./envs/schiphol-r.yml
conda env create -f ./envs/schiphol-tf.yml

conda deactivate
conda activate schiphol-snakemake
python -m ipykernel install --user --name schiphol-snakemake --display-name "Python␣
↪(schiphol-snakemake)"
conda deactivate

conda deactivate
conda activate schiphol-py
python -m ipykernel install --user --name schiphol-py --display-name "Python␣
↪(schiphol-py)"
conda deactivate

conda deactivate
conda activate schiphol-r
python -m ipykernel install --user --name schiphol-r --display-name "Python (schiphol-
↪r)"
conda deactivate

conda deactivate
conda activate schiphol-tf
python -m ipykernel install --user --name schiphol-tf --display-name "Python␣
↪(schiphol-tf)"
conda deactivate
```

4.2 Docker image

Build locally, but note that the build is time-consuming as we are installing 3 separate conda environments into the container.

```
docker build -t schiphol .
```

The Docker container will execute `snakemake` when you run the container, but for this you need to be authenticated for write access to the Google Cloud Storage where data is located. Read access is already public.

Assuming that you have a folder named `keys/` in this project root directory, you must mount it alongside the rest of the project when you run the container. Because we are mounting the service-account key with write-access we can now execute Snakemake with the Docker container.

```
docker run -v {$pwd}:/project/ schiphol
```

CHAPTER 5

Create Pandas profiling report

Using a .CSV file as input, we create a .HTML pandas profiling report.

This notebook is purposely concise and simple.

5.1 Script parameters

Parameters

- input_file: String filename as input that can be read using pandas.read_csv(...)
- output_file: String filename to save .HTML report file to

Returns

HTML pandas profiling report file saved to specified `output_file`

[2]:
```
# input parameters cell
input_file = "../data/raw/flights.csv"
output_file = "../test_output/pandas_profiling__flights.html"
profiling_config = "profiling_config.yml"
```

[ ]:
```
import pandas as pd
from pandas_profiling import ProfileReport
```

5.2 Read data

[3]:
```
df = pd.read_csv(input_file)
df.shape
```

[3]:
```
(523275, 28)
```

5.3 Create pandas-profiling reports

Default options used here. This script can easily be extended to allow custom themes or summary calculations.

[4]:
```
# create pandas profiling report and save to HTML output
profile = ProfileReport(df)
profile
```

```
HBox(children=(FloatProgress(value=0.0, description='Summarize dataset', max=42.0,␣
↪style=ProgressStyle(descrip...
```

```
HBox(children=(FloatProgress(value=0.0, description='Generate report structure',␣
↪max=1.0, style=ProgressStyle(...
```

```
HBox(children=(FloatProgress(value=0.0, description='Render HTML', max=1.0,␣
↪style=ProgressStyle(description_wi...
```

```
<IPython.core.display.HTML object>
```

[4]:

5.4 Save HTML output to file

Report output is saved to a stand-alone HTML file that can be sent to others with no further dependencies.

[5]:
```
# create pandas profiling report and save to HTML output
profile.to_file(output_file)
print(f"Written profiling report to {output_file}")
```

```
HBox(children=(FloatProgress(value=0.0, description='Export report to file', max=1.0,␣
↪style=ProgressStyle(desc...
```

```
Written profiling report to ../test_output/pandas_profiling__flights.html
```

[ ]:

[13]:
```
%load_ext autoreload
%autoreload 2

%matplotlib inline
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

CHAPTER 6

Create minimal model input

- Takes the flights data
- Processes the schedule/realized datetimes and computes the delay in seconds
- Remove observations with unknown prediction targets
- Write prediction target with minimal feature set to CSV

6.1 Parameters

- input_file: Filepath of flights data in format received from Schiphol
- output_file: Filepath to write output csv file with minimal modelling input

6.2 Returns

Output CSV file written to `output_file` with minimal model input

Example output,

```
id                  |  aircraftRegistration  |  airlineCode  |  terminal  | ␣
↪serviceType  |      scheduleDateTime        |     actualOffBlockTime      | ␣
↪scheduleDelaySeconds
124257473326719795   |    PHEXI               |     80.0      |     2.0    |       J ␣
↪     |  2018-05-01 16:35:00+02:00   |  2018-05-01 16:58:16+02:00  |         1396.0
124538476600837715   |    PHEXL               |     2481.0    |     1.0    |       J ␣
↪     |  2018-06-10 13:00:00+02:00   |  2018-06-10 13:11:25+02:00  |         685.0
123512829091050355   |    PHBQO               |     100.0     |     2.0    |       J ␣
↪     |  2018-01-15 10:15:00+01:00   |  2018-01-15 10:35:10+01:00  |         1210.0
```

(continues on next page)

(continued from previous page)

```
123786805997701057   |    PHEXG              |    2481.0    |    1.0    |       J ␣
↪     |  2018-02-23 17:45:00+01:00   |  2018-02-23 17:55:52+01:00  |          652.0
124664922607744671   |    PHBXP              |    1551.0    |    2.0    |       J ␣
↪     |  2018-06-28 20:50:00+02:00   |  2018-06-28 22:09:23+02:00  |         4763.0
```

6.2.1 File parameters

[14]:
```
# input parameters cell
input_file = "../lvt-schiphol-assignment-snakemake/data/raw/flights.csv"
output_file = "processed_flights.csv"
```

6.2.2 Imports

[15]:
```
import pandas as pd
import numpy as np

import sys
sys.path.append("../")

from src.data.google_storage_io import read_csv_data, write_csv_data
```

6.2.3 Utility functions

[16]:
```
def missing_values_percentages(df):
    """Calculate summary of missing values per column"""
    percent_missing = df.isnull().sum() * 100 / len(df)
    missing_value_df = pd.DataFrame({'column_name': df.columns,
                                     'percent_missing': percent_missing})

    missing_value_df = missing_value_df.sort_values('percent_missing',␣
↪ascending=False)
    return missing_value_df


def check_col_singular(x):
    """check if pd.Series contains more than 1 unique value excluding NaN"""
    return x.dropna().nunique() <= 1


def drop_singular_columns(df, verbose=False):
    """Drop DataFrame columns with 1 or fewer unique values excluding NaN"""
    col_singular = df.apply(check_col_singular, axis=0)

    if verbose:
        n_singular = sum(col_singular)
        print(f"Dropping {n_singular} columns")
        print(f"{col_singular[col_singular].index}")

    df_output = df[[col for col, is_singular in col_singular.items()
                    if not is_singular]]
```

(continues on next page)

(continued from previous page)

```
    return df_output


def clean_flights(df_flights, verbose=True):
    """Clean flights data by removing singular columns and formatting dates"""
    df = df_flights
    df = df.dropna(subset=["scheduleDate", "scheduleTime", "actualOffBlockTime"]).
↪reset_index(drop=True)

    # remove singular columns
    df = drop_singular_columns(df, verbose=verbose)

    # format datetime fields
    df["actualOffBlockTime"] = pd.to_datetime(df["actualOffBlockTime"], utc=True).dt.
↪tz_convert('Europe/Amsterdam')

    series_datetime_str = df['scheduleDate'].astype(str) + " " + df['scheduleTime'].
↪astype(str)
    df["scheduleDateTime"] = pd.to_datetime(series_datetime_str, format="%Y%m%d %H:%M:
↪%S").dt.tz_localize('Europe/Amsterdam')

#     calculate delay as difference between scheduled and realized departure
    df["scheduleDelaySeconds"] = pd.to_timedelta(df["actualOffBlockTime"] - df[
↪"scheduleDateTime"]).dt.total_seconds()

    return df


def read_flights_data(filename):
    """Read data local or from Google Storage bucket and clean it"""
    df = read_csv_data(input_file)
    print(f"Loaded data from: {input_file}\n"
          f"Shape of data: {df.shape}")

    df = clean_flights(df)
    print(f"Cleaned flights data\n"
          f"Shape of data: {df.shape}")

    return df
```

Read data

[17]:
```
%%time
df = read_csv_data(input_file)
df.head()
```

```
Reading file from local directory
File:   ../lvt-schiphol-assignment-snakemake/data/raw/flights.csv

Wall time: 1.93 s
```

[17]:

	actualOffBlockTime	aircraftRegistration	aircraftType.iatamain	\
0	NaN	NaN	NaN	
1	NaN	PHPXY	AW1	
2	NaN	NaN	AW1	
3	2018-01-01T03:22:00.000+01:00	PHPXB	NaN	

(continues on next page)

(continued from previous page)

```
4  2018-01-01T05:58:22.000+01:00                PHHSJ                    73H

  aircraftType.iatasub  airlineCode  baggageClaim  estimatedLandingTime  \
0                  NaN        148.0           NaN                   NaN
1                  NaN        148.0           NaN                   NaN
2                  NaN        148.0           NaN                   NaN
3                  NaN        148.0           NaN                   NaN
4                  73H        164.0           NaN                   NaN

  expectedTimeBoarding expectedTimeGateClosing expectedTimeGateOpen  ...  \
0                  NaN                     NaN                  NaN  ...
1                  NaN                     NaN                  NaN  ...
2                  NaN                     NaN                  NaN  ...
3                  NaN                     NaN                  NaN  ...
4                  NaN                     NaN                  NaN  ...

   prefixICAO publicEstimatedOffBlockTime publicFlightState.flightStates  \
0         ZXP                         NaN                       ['SCH']
1         ZXP                         NaN                       ['SCH']
2         ZXP                         NaN                       ['SCH']
3         ZXP                         NaN                       ['DEP']
4         TRA                         NaN                       ['DEP']

   route.destinations scheduleDate  scheduleTime serviceType terminal  \
0             ['AMS']   2018-01-01      03:02:07           P      NaN
1             ['AMS']   2018-01-01      03:16:00         NaN      NaN
2             ['AMS']   2018-01-01      03:16:29           P      NaN
3             ['AMS']   2018-01-01      03:30:00         NaN      NaN
4             ['SPC']   2018-01-01      06:00:00           J      1.0

  transferPositions transferPositions.transferPositions
0               NaN                                NaN
1               NaN                                NaN
2               NaN                                NaN
3               NaN                                NaN
4               NaN                                NaN

[5 rows x 28 columns]
```

[18]:
```
%%time
df = clean_flights(df)
df.head()
```

```
Dropping 5 columns
Index(['baggageClaim', 'estimatedLandingTime', 'expectedTimeOnBelt',
       'flightDirection', 'transferPositions'],
      dtype='object')
Wall time: 4.92 s
```

[18]:
```
         actualOffBlockTime aircraftRegistration aircraftType.iatamain  \
0 2018-01-01 03:22:00+01:00                PHPXB                   NaN
1 2018-01-01 05:58:22+01:00                PHHSJ                   73H
2 2018-01-01 06:00:00+01:00                PHHSG                   73H
3 2018-01-01 06:00:00+01:00                PHHSG                   73H
4 2018-01-01 06:26:34+01:00                PHHXB                   73H

  aircraftType.iatasub  airlineCode expectedTimeBoarding  \
```

(continues on next page)

(continued from previous page)

```
0                  NaN        148.0                  NaN
1                  73H        164.0                  NaN
2                  73H        100.0                  NaN
3                  73H        164.0                  NaN
4                  73H        164.0                  NaN

  expectedTimeGateClosing expectedTimeGateOpen flightName  flightNumber  ...  \
0                     NaN                  NaN     ZXP022          22.0  ...
1                     NaN                  NaN     HV5641        5641.0  ...
2                     NaN                  NaN     KL2533        2533.0  ...
3                     NaN                  NaN     HV6455        6455.0  ...
4                     NaN                  NaN     HV5801        5801.0  ...

  publicEstimatedOffBlockTime  publicFlightState.flightStates  \
0                         NaN                        ['DEP']
1                         NaN                        ['DEP']
2                         NaN                        ['DEP']
3                         NaN                        ['DEP']
4                         NaN                        ['DEP']

  route.destinations scheduleDate scheduleTime serviceType terminal  \
0            ['AMS']   2018-01-01     03:30:00         NaN      NaN
1            ['SPC']   2018-01-01     06:00:00           J      1.0
2            ['LPA']   2018-01-01     06:05:00           J      1.0
3            ['LPA']   2018-01-01     06:05:00           J      1.0
4            ['TLV']   2018-01-01     06:15:00           J      1.0

  transferPositions.transferPositions          scheduleDateTime  \
0                                 NaN 2018-01-01 03:30:00+01:00
1                                 NaN 2018-01-01 06:00:00+01:00
2                                 NaN 2018-01-01 06:05:00+01:00
3                                 NaN 2018-01-01 06:05:00+01:00
4                                 NaN 2018-01-01 06:15:00+01:00

  scheduleDelaySeconds
0               -480.0
1                -98.0
2               -300.0
3               -300.0
4                694.0

[5 rows x 25 columns]
```

6.3 Check for duplicates by id

Based on earlier findings we know there are duplicate values in the `id` column. We assume the `id` to be unique so that it can be used for indexing and merging.

Downstream it is vital that the `id` is unique and we can safely drop duplicate entries from the data.

[19]:
```python
def duplicates_by_id(df):
    df = df[df["id"].isin(df["id"][df[["id"]].duplicated()].unique())]
    return df
```

(continues on next page)

(continued from previous page)

```
def test_duplicates_by_id(df, verbose=True):
    df_duplicates_by_id = duplicates_by_id(df)
    nrows_duplicates_by_id = df_duplicates_by_id.shape[0]
    nrows_drop_duplicates_by_id = df_duplicates_by_id.drop_duplicates("id").shape[0]
    nrows_drop_duplicates_all = df_duplicates_by_id.drop_duplicates().shape[0]

    diff_duplicates_by_id = nrows_drop_duplicates_by_id - nrows_drop_duplicates_all
    if diff_duplicates_by_id == 0:
        only_full_duplicates = True
    if verbose:
        print(f"""
        The number of rows with only those ID's that are duplicated, including
        all their occurences is {nrows_duplicates_by_id}.

        Rows without duplicates only based on the `id`: {nrows_drop_duplicates_by_id}
        Rows without duplicates based on all columns: {nrows_drop_duplicates_by_id}

        If they are equal then all rows that are duplicated by `id` have no␣
↪differences
        in other columns and are exact duplicates.

        Result: {only_full_duplicates}
        """)
    return only_full_duplicates


def test_duplicates_by_id_smarter(df):
    """Test if dropping all duplicates is equivalent to dropping by `id`"""
    nrows_no_dupes = df.drop_duplicates().shape[0]
    nrows_no_dupes_by_id = df.drop_duplicates("id").shape[0]
    return nrows_no_dupes == nrows_no_dupes_by_id

# test if we have unique ids even though there are duplicates in the dataframe
ids_ok = test_duplicates_by_id(df, verbose=True)
ids_ok_smarter = test_duplicates_by_id_smarter(df)

# asserts
assert ids_ok
assert ids_ok_smarter

# drop duplicates if test passed
df = df.drop_duplicates("id")
```

```
        The number of rows with only those ID's that are duplicated, including
        all their occurences is 19847.

        Rows without duplicates only based on the `id`: 9909
        Rows without duplicates based on all columns: 9909

        If they are equal then all rows that are duplicated by `id` have no␣
↪differences
        in other columns and are exact duplicates.

        Result: True
```

6.4 Filter out data from 2017

Flight data from late 2017 has some outliers. Since it is at the very start of the data we don't take as much consideration to simply remove the first couple of observations before 2018-01-01.

[20]:
```
shape_b4 = df.shape
filter_date = '2018-01-01 00:00:00+01:00'
df = df.query(f"scheduleDateTime >= '{filter_date}'")
print(f"Removed {shape_b4[0] - df.shape[0]} rows from before {filter_date}")
```

```
Removed 2 rows from before 2018-01-01 00:00:00+01:00
```

6.5 Output prediction target

[21]:
```
df.columns
```

[21]:
```
Index(['actualOffBlockTime', 'aircraftRegistration', 'aircraftType.iatamain',
       'aircraftType.iatasub', 'airlineCode', 'expectedTimeBoarding',
       'expectedTimeGateClosing', 'expectedTimeGateOpen', 'flightName',
       'flightNumber', 'gate', 'id', 'mainFlight', 'prefixIATA', 'prefixICAO',
       'publicEstimatedOffBlockTime', 'publicFlightState.flightStates',
       'route.destinations', 'scheduleDate', 'scheduleTime', 'serviceType',
       'terminal', 'transferPositions.transferPositions', 'scheduleDateTime',
       'scheduleDelaySeconds'],
      dtype='object')
```

[22]:
```
# meta columns for utility for columns we will often merge by
output_columns = ["id", "aircraftRegistration", "airlineCode", "terminal",
                  "serviceType", "scheduleDateTime", "actualOffBlockTime",
↪"scheduleDelaySeconds"]

# DataFrame with id + merging columns + prediction target
df_target = df[output_columns]
df_target.head()
```

[22]:

	id	aircraftRegistration	airlineCode	terminal	serviceType	scheduleDateTime	actualOffBlockTime	scheduleDelaySeconds
0	123414481790510775	PHPXB	148.0	NaN	NaN	2018-01-01 03:30:00+01:00	2018-01-01 03:22:00+01:00	-480.0
1	123414479288269149	PHHSJ	164.0	1.0	J	2018-01-01 06:00:00+01:00	2018-01-01 05:58:22+01:00	-98.0
2	123414479666542945	PHHSG	100.0	1.0	J	2018-01-01 06:05:00+01:00	2018-01-01 06:00:00+01:00	-300.0
3	123414479288365061	PHHSG	164.0	1.0	J	2018-01-01 06:05:00+01:00	2018-01-01 06:00:00+01:00	-300.0
4	123414479288274329	PHHXB	164.0	1.0	J	2018-01-01 06:15:00+01:00	2018-01-01 06:26:34+01:00	694.0

6.6 Write output to CSV

Local or Google Storage is both handled

[23]:
```
# write output file
write_csv_data(df_target, output_file, index=False)
```

```
Writing file to local directory
File:   processed_flights.csv
```

[24]:
```
df_target.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 477776 entries, 0 to 487714
Data columns (total 8 columns):
 #   Column                Non-Null Count   Dtype
---  ------                --------------   -----
 0   id                    477776 non-null  int64
 1   aircraftRegistration  477773 non-null  object
 2   airlineCode           476594 non-null  float64
 3   terminal              467868 non-null  float64
 4   serviceType           473110 non-null  object
 5   scheduleDateTime      477776 non-null  datetime64[ns, Europe/Amsterdam]
 6   actualOffBlockTime    477776 non-null  datetime64[ns, Europe/Amsterdam]
 7   scheduleDelaySeconds  477776 non-null  float64
dtypes: datetime64[ns, Europe/Amsterdam](2), float64(3), int64(1), object(2)
memory usage: 32.8+ MB
```

[1]:
```
%matplotlib inline
```

CHAPTER 7

Merge base input and feature data

- Takes the flights data
- Processes the schedule/realized datetimes and computes the delay in seconds
- Remove observations with unknown prediction targets
- Write prediction target with minimal feature set to CSV

7.1 Parameters

- base_file: Filepath of base model input with at least column 'id'
- features: List of feature files or a string of feature files separated by a '+'

7.2 Returns

Output CSV file with minimal model input

```
      id              |  aircraftRegistration   |  airlineCode   |  terminal   |  ... ⌋
↪ |  year   | ...
123414481790510775    |         PHPXB           |     148.0      |     NaN     | ...  ⌋
↪ |  2018   | ...
123414479288269149    |         PHHSJ           |     164.0      |     1.0     | ...  ⌋
↪ |  2018   | ...
123414479666542945    |         PHHSG           |     100.0      |     1.0     | ...  ⌋
↪ |  2018   | ...
```

(continues on next page)

(continued from previous page)

```
123414479288365061   |         PHHSG          |     164.0     |     1.0      | ...  ␣
↪ |  2018   | ...
123414479288274329   |         PHHXB          |     164.0     |     1.0      | ...  ␣
↪ |  2018   | ...
```

7.3 File parameters

[2]:
```
# input parameters cell
base_file = "../lvt-schiphol-assignment-snakemake/data/model_input/delays_base_input.
↪csv"
features = [
    "../lvt-schiphol-assignment-snakemake/data/model_input/features/route_
↪destinations.csv",
    "../lvt-schiphol-assignment-snakemake/data/model_input/features/schedule_time_
↪features.csv"
]


output_file = "../lvt-schiphol-assignment-snakemake/data/model_input/delays_extended_
↪input.csv"
```

[3]:
```
if isinstance(features, str):
    features = features.split('+')
    print("Parsed features from string instead of List object")
    print(features)
```

[4]:
```
columns_to_ignore = [
    "scheduleDateTime", "scheduleDate", "scheduleTime", "actualOffBlockTime"
]
```

7.3.1 Libraries

[5]:
```
import pandas as pd
import numpy as np

import sys
sys.path.append("../")

from src.data.google_storage_io import read_csv_data, write_csv_data
```

Read data

[6]:
```
%%time
df_base = read_csv_data(base_file)
df_base.head()
```

```
Reading file from local directory
File:   ../lvt-schiphol-assignment-snakemake/data/model_input/delays_base_input.csv

Wall time: 533 ms
```

[6]:
```
                   id aircraftRegistration  airlineCode  terminal serviceType  \
0  123414481790510775                PHPXB        148.0       NaN         NaN
1  123414479288269149                PHHSJ        164.0       1.0           J
2  123414479666542945                PHHSG        100.0       1.0           J
3  123414479288365061                PHHSG        164.0       1.0           J
4  123414479288274329                PHHXB        164.0       1.0           J

            scheduleDateTime         actualOffBlockTime  scheduleDelaySeconds
0  2018-01-01 03:30:00+01:00  2018-01-01 03:22:00+01:00                -480.0
1  2018-01-01 06:00:00+01:00  2018-01-01 05:58:22+01:00                 -98.0
2  2018-01-01 06:05:00+01:00  2018-01-01 06:00:00+01:00                -300.0
3  2018-01-01 06:05:00+01:00  2018-01-01 06:00:00+01:00                -300.0
4  2018-01-01 06:15:00+01:00  2018-01-01 06:26:34+01:00                 694.0
```

[7]:
```
%%time

# read feature data from multiple files and merge by 'id'
print(f"Reading features from first file: {features[0]}")
df_features = read_csv_data(features[0])

if len(features) > 0:
    for feature_file in features[1:]:
        print(f"Merging features from file: {feature_file}")
        old_shape = df_features.shape
        tmp_features = read_csv_data(feature_file)
        df_features = pd.merge(
            df_features,
            tmp_features,
            on="id",
            how="inner"
        )
        print(f"Merged features. Shape {old_shape} -> {df_features.shape}")
df_features.head()
```

```
Reading features from first file: ../lvt-schiphol-assignment-snakemake/data/model_
↪input/features/route_destinations.csv
Reading file from local directory
File:   ../lvt-schiphol-assignment-snakemake/data/model_input/features/route_
↪destinations.csv

Merging features from file: ../lvt-schiphol-assignment-snakemake/data/model_input/
↪features/schedule_time_features.csv
Reading file from local directory
File:   ../lvt-schiphol-assignment-snakemake/data/model_input/features/schedule_time_
↪features.csv

Merged features. Shape (523275, 9) -> (487716, 17)
Wall time: 1.23 s
```

[7]:
```
                   id final_destination      Country                   City  \
0  123414481790510775               AMS  Netherlands              Amsterdam
1  123414479288269149               SPC        Spain  Santa Cruz De La Palma
2  123414479666542945               LPA        Spain           Gran Canaria
3  123414479288365061               LPA        Spain           Gran Canaria
4  123414479288274329               TLV       Israel               Tel-aviv

    Latitude  Longitude  Altitude DST  destination_distance  dayofweek  \
0  52.308601    4.76389     -11.0   E          4.338444e-07          0
```

(continues on next page)

(continued from previous page)

```
1  28.626499  -17.75560     107.0   E          3.267980e+01          0
2  27.931900  -15.38660      78.0   E          3.162698e+01          0
3  27.931900  -15.38660      78.0   E          3.162698e+01          0
4  32.011398   34.88670     135.0   E          3.632300e+01          0

   quarter  month  year  dayofmonth  weekofyear  hour  minutes
0        1      1  2018           1           1     3       30
1        1      1  2018           1           1     6        0
2        1      1  2018           1           1     6        5
3        1      1  2018           1           1     6        5
4        1      1  2018           1           1     6       15
```

Merge base model input with features

- One large file to pass onto model notebooks

Downside: One large file with a lot of copied values

Upside: Easier to verify downstream model notebooks use the same data

[8]:
```
df_output = pd.merge(
    df_base,
    df_features,
    on="id",
    how="inner")
print(f"Data shape: {df_output.shape}")
df_output.head()
```

```
Data shape: (487714, 24)
```

[8]:
```
                  id aircraftRegistration  airlineCode  terminal serviceType  \
0  123414481790510775                PHPXB        148.0       NaN         NaN
1  123414479288269149                PHHSJ        164.0       1.0           J
2  123414479666542945                PHHSG        100.0       1.0           J
3  123414479288365061                PHHSG        164.0       1.0           J
4  123414479288274329                PHHXB        164.0       1.0           J

            scheduleDateTime         actualOffBlockTime  scheduleDelaySeconds  \
0  2018-01-01 03:30:00+01:00  2018-01-01 03:22:00+01:00                -480.0
1  2018-01-01 06:00:00+01:00  2018-01-01 05:58:22+01:00                 -98.0
2  2018-01-01 06:05:00+01:00  2018-01-01 06:00:00+01:00                -300.0
3  2018-01-01 06:05:00+01:00  2018-01-01 06:00:00+01:00                -300.0
4  2018-01-01 06:15:00+01:00  2018-01-01 06:26:34+01:00                 694.0

  final_destination      Country  ... DST  destination_distance  dayofweek  \
0               AMS  Netherlands  ...   E          4.338444e-07          0
1               SPC        Spain  ...   E          3.267980e+01          0
2               LPA        Spain  ...   E          3.162698e+01          0
3               LPA        Spain  ...   E          3.162698e+01          0
4               TLV       Israel  ...   E          3.632300e+01          0

   quarter month  year  dayofmonth  weekofyear  hour  minutes
0        1     1  2018           1           1     3       30
1        1     1  2018           1           1     6        0
2        1     1  2018           1           1     6        5
3        1     1  2018           1           1     6        5
4        1     1  2018           1           1     6       15
```

(continues on next page)

(continued from previous page)

```
[5 rows x 24 columns]
```

Write output to CSV

Local or Google Storage is both handled

[9]:
```
# # write output file
write_csv_data(df_output, output_file, index=False)
```

```
Writing file to local directory
File:   ../lvt-schiphol-assignment-snakemake/data/model_input/delays_extended_input.
↪csv
```

7.4 Overview of the output data

[10]:
```
df_output.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 487714 entries, 0 to 487713
Data columns (total 24 columns):
 #   Column                Non-Null Count   Dtype
---  ------                --------------   -----
 0   id                    487714 non-null  int64
 1   aircraftRegistration  487711 non-null  object
 2   airlineCode           486501 non-null  float64
 3   terminal              477391 non-null  float64
 4   serviceType           482935 non-null  object
 5   scheduleDateTime      487714 non-null  object
 6   actualOffBlockTime    487714 non-null  object
 7   scheduleDelaySeconds  487714 non-null  float64
 8   final_destination     487701 non-null  object
 9   Country               486955 non-null  object
 10  City                  486955 non-null  object
 11  Latitude              486955 non-null  float64
 12  Longitude             486955 non-null  float64
 13  Altitude              486955 non-null  float64
 14  DST                   486955 non-null  object
 15  destination_distance  486955 non-null  float64
 16  dayofweek             487714 non-null  int64
 17  quarter               487714 non-null  int64
 18  month                 487714 non-null  int64
 19  year                  487714 non-null  int64
 20  dayofmonth            487714 non-null  int64
 21  weekofyear            487714 non-null  int64
 22  hour                  487714 non-null  int64
 23  minutes               487714 non-null  int64
dtypes: float64(7), int64(9), object(8)
memory usage: 93.0+ MB
```

CHAPTER 8

Create train/test split of the data

This notebook will take a DataFrame with at least `['id', 'scheduleDateTime']` and creates a train/test split

1. Get `id` column of the data and split into a train/test set
2. Strategy of either 'sample' or 'timeseries'

TODO:

- Stratification based on important groups we have yet to identify
- Decide whether validation splits should also be made here or if OK to do downstream

8.1 Parameters

- input_file: Filepath of flights data in format received from Schiphol
- output_file: Filepath to write output csv file with minimal modelling input
- strategy: One of ['sample', 'timeseries']
- test_size: (Optional) Default 0.3. Fraction to use as test data between 0 and 1
- val_size: (Optional) Default 0.1. Fraction to use as validation data between 0 and 1

If `strategy` == 'timeseries' then data is split on the `scheduleDateTime` column and takes the last `test_size` from the data as the test set.

8.2 Returns

Output format

```
id                   |   model_set |
123414481790510775   |  train      |
123414479288269149   |  train      |
123414479666542945   |  test       |
123414479288365061   |  test       |
123414479288274329   |  validation |    # validation set not currently implemented
```

8.2.1 Script parameters

[7]:
```
# parameters
input_file = "../lvt-schiphol-assignment-snakemake/data/model_input/delays_extended_
↪input.csv"
output_file = "train_test__sample__0.2.csv"
strategy = 'sample'
test_size = 0.2
```

[8]:
```
assert test_size < 1 and test_size > 0
assert strategy in ['sample', 'timeseries']
```

8.2.2 Imports

[9]:
```
import pandas as pd
from sklearn.model_selection import train_test_split

import sys
sys.path.append("../")

from src.data.google_storage_io import read_csv_data, write_csv_data
```

Load data

[10]:
```
%%time

df = read_csv_data(input_file)

# subset only columns we need
df = df[["id", "scheduleDateTime"]]
df["scheduleDateTime"] = pd.to_datetime(df["scheduleDateTime"])
df.head()
```

```
Reading file from local directory
File:   ../lvt-schiphol-assignment-snakemake/data/model_input/delays_extended_input.
↪csv

Wall time: 8.22 s
```

[10]:

	id	scheduleDateTime
0	123414481790510775	2018-01-01 03:30:00+01:00
1	123414479288269149	2018-01-01 06:00:00+01:00
2	123414479666542945	2018-01-01 06:05:00+01:00
3	123414479288365061	2018-01-01 06:05:00+01:00
4	123414479288274329	2018-01-01 06:15:00+01:00

Make train/test split

[11]:
```
print(f"Strategy: {strategy}")

if strategy == 'sample':
    # sample like usual
    train_ids, test_ids = train_test_split(df["id"], test_size=test_size)
elif strategy == 'timeseries':
    # select last fraction of data as test set
    df = df.sort_values("scheduleDateTime").reset_index()
    test_size = int(len(df) * 0.2)
    train_ids, test_ids = df.iloc[:-test_size]["id"], df.iloc[-test_size:]["id"]

df_train_test = pd.concat([
    pd.DataFrame(dict(id = train_ids.values, model_set = "train")),
    pd.DataFrame(dict(id = test_ids.values, model_set = "test"))
])

df_train_test
```

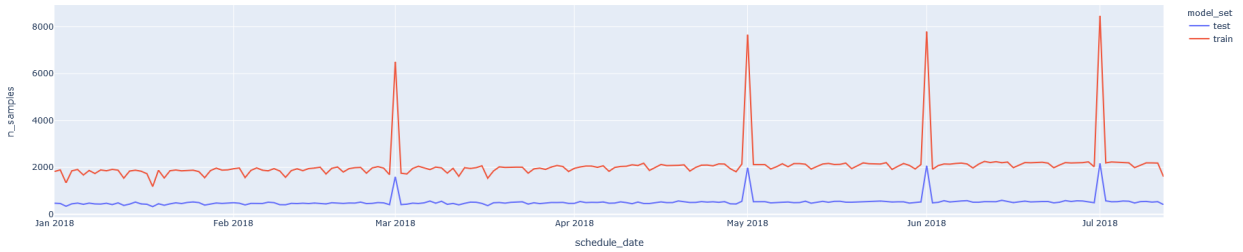
```
Strategy: sample
```

[11]:
```
                       id model_set
0      123583078680134091     train
1      124594671792954447     train
2      123990526571117207     train
3      124397975056975081     train
4      123421504625349921     train
...                   ...       ...
97538  123899201831170143      test
97539  124060775885955111      test
97540  124039700039122673      test
97541  123885150856850759      test
97542  124552523009520195      test

[487714 rows x 2 columns]
```

8.3 Visualize train/test split over time

Show number of samples per day to understand train/test distribution

[16]:
```
import plotly.express as px
df_plot = pd.merge(
        df_train_test,
        df, on="id", how="left") \
    .assign(
        schedule_date = lambda d: pd.to_datetime(d["scheduleDateTime"], utc=True).dt.
↪date) \
    .groupby(["schedule_date", "model_set"])["id"].count().reset_index(name="n_samples
↪")

px.line(df_plot, x="schedule_date", y="n_samples", color="model_set")
```



Local or Google Storage is both handled

[11]:
```
# write output file
write_csv_data(df_train_test, output_file, index=False)
```

```
Writing file to local directory
File:   processed_flights.csv
```

8.4 Overview of the output data

[13]:
```
df_train_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 487716 entries, 0 to 487715
Data columns (total 8 columns):
id                      487716 non-null int64
aircraftRegistration    487713 non-null object
airlineCode             486503 non-null float64
terminal                477391 non-null float64
serviceType             482937 non-null object
scheduleDateTime        487716 non-null datetime64[ns, Europe/Amsterdam]
actualOffBlockTime      487716 non-null datetime64[ns, Europe/Amsterdam]
scheduleDelaySeconds    487716 non-null float64
dtypes: datetime64[ns, Europe/Amsterdam](2), float64(3), int64(1), object(2)
memory usage: 29.8+ MB
```

CHAPTER 9

Time features from DateTime data

9.1 year, week_number, day_of_week, etc..

This notebook will take a DataFrame with `scheduleDateTime` or optionally a different column

1. Calculate time features from DateTime values
2. Output pd.DataFrame with id columns + time feature columns
3. Write output to CSV file

9.2 Parameters

- input_file: Filepath of flights data in format received from Schiphol
- output_file: Filepath to write output csv file with minimal modelling input
- dt_column: (Optional) Column with datetime values to create features from. Default `scheduleDateTime`
- id_column: (Optional) ID column to keep from input file. Default `id`

9.3 Returns

Output format

```
id                  |   scheduleDateTime           |  dayofweek   |  quarter   |
↪month   |  year  |   dayofmonth   |  weekofyear   |  hour   |  minutes
123414481790510775  |  2018-01-01 03:30:00+01:00   |      0       |      1     |
↪1       |  2018  |        1       |       1       |    3    |  30
123414479288269149  |  2018-01-01 06:00:00+01:00   |      0       |      1     |
↪1       |  2018  |        1       |       1       |    6    |  0
123414479666542945  |  2018-01-01 06:05:00+01:00   |      0       |      1     |
↪1       |  2018  |        1       |       1       |    6    |  5
123414479288365061  |  2018-01-01 06:05:00+01:00   |      0       |      1     |
↪1       |  2018  |        1       |       1       |    6    |  5
123414479288274329  |  2018-01-01 06:15:00+01:00   |      0       |      1     |
↪1       |  2018  |        1       |       1       |    6    |  15
```

[19]:
```
# parameters
input_file = "../lvt-schiphol-assignment-snakemake/data/model_input/delays_base_input.
↪csv"
output_file = "schedule_time_features.csv"
dt_column = "scheduleDateTime"
id_column = "id"
```

9.3.1 Imports

[1]:
```
import pandas as pd
import numpy as np

import sys
sys.path.append("../")

from src.data.google_storage_io import read_csv_data, write_csv_data
```

9.3.2 Load data

[31]:
```
%%time

df = read_csv_data(input_file)
df.head()
```

```
Reading file from local directory
File:   ../lvt-schiphol-assignment-snakemake/data/model_input/delays_base_input.csv

Wall time: 535 ms
```

[31]:

	id	aircraftRegistration	airlineCode	terminal	serviceType	\
0	123414481790510775	PHPXB	148.0	NaN	NaN	
1	123414479288269149	PHHSJ	164.0	1.0	J	
2	123414479666542945	PHHSG	100.0	1.0	J	
3	123414479288365061	PHHSG	164.0	1.0	J	
4	123414479288274329	PHHXB	164.0	1.0	J	

	scheduleDateTime	actualOffBlockTime	scheduleDelaySeconds
0	2018-01-01 03:30:00+01:00	2018-01-01 03:22:00+01:00	-480.0
1	2018-01-01 06:00:00+01:00	2018-01-01 05:58:22+01:00	-98.0
2	2018-01-01 06:05:00+01:00	2018-01-01 06:00:00+01:00	-300.0

(continues on next page)

(continued from previous page)

```
3  2018-01-01 06:05:00+01:00  2018-01-01 06:00:00+01:00      -300.0
4  2018-01-01 06:15:00+01:00  2018-01-01 06:26:34+01:00       694.0
```

Create DataFrame with daily time features

[44]:
```
%%time

keep_columns = [
        'dayofweek', 'quarter',
        'month', 'year', 'dayofmonth',
        'weekofyear'
    ]

def add_time_features(df,
                      date_column,
                      id_column = None,
                      keep_columns = [
                          'dayofweek', 'quarter',
                          'month', 'year', 'dayofmonth',
                          'weekofyear', 'hour', 'minutes']):
    """
    Creates time series features from date column

    Pass a dataframe with dates in `date_column`
    The input dataframe is returned with an id column and the  extracted
    basic time features.

    """
    original_columns = list(df.columns)

    df[date_column] = pd.to_datetime(df[date_column], utc=True).dt.tz_convert("Europe/
↪Amsterdam")

    df = df.assign(
            dayofweek = df[date_column].dt.dayofweek,
            quarter = df[date_column].dt.quarter,
            month = df[date_column].dt.month,
            year = df[date_column].dt.year,
            dayofyear = df[date_column].dt.dayofyear, # ignored
            dayofmonth = df[date_column].dt.day,
            weekofyear = df[date_column].dt.weekofyear,
            hour = df[date_column].dt.hour,
            minutes = df[date_column].dt.minute)

    if id_column is None:
        df = df.reset_index()
        id_column = "index"

    output_columns = [id_column] + keep_columns

    return df[output_columns]


df_output = add_time_features(df, date_column = dt_column, id_column = id_column)
df_output.head()
```

(continues on next page)

(continued from previous page)

```
Wall time: 448 ms
```

[44]:
```
                   id  dayofweek  quarter  month  year  dayofmonth  \
0  123414481790510775          0        1      1  2018           1
1  123414479288269149          0        1      1  2018           1
2  123414479666542945          0        1      1  2018           1
3  123414479288365061          0        1      1  2018           1
4  123414479288274329          0        1      1  2018           1

   weekofyear  hour  minutes
0           1     3       30
1           1     6        0
2           1     6        5
3           1     6        5
4           1     6       15
```

[45]:
```
df_output.describe(include='all')
```

[45]:
```
                 id      dayofweek        quarter          month  \
count  4.877160e+05  487716.000000  487716.000000  487716.000000
mean   1.241177e+17       2.940383       1.637703       3.869715
std    3.891198e+14       1.994474       0.610135       1.855526
min    1.234004e+17       0.000000       1.000000       1.000000
25%    1.237868e+17       1.000000       1.000000       2.000000
50%    1.241310e+17       3.000000       2.000000       4.000000
75%    1.244612e+17       5.000000       2.000000       5.000000
max    1.247644e+17       6.000000       4.000000      12.000000

                year     dayofmonth     weekofyear           hour  \
count  487716.000000  487716.000000  487716.000000  487716.000000
mean     2017.999996      14.799594      14.881474      13.650389
std         0.002025       8.904548       7.913090       4.649598
min      2017.000000       1.000000       1.000000       0.000000
25%      2018.000000       7.000000       8.000000       9.000000
50%      2018.000000      14.000000      15.000000      13.000000
75%      2018.000000      23.000000      22.000000      17.000000
max      2018.000000      31.000000      52.000000      23.000000

             minutes
count  487716.000000
mean       27.531055
std        17.187699
min         0.000000
25%        15.000000
50%        30.000000
75%        40.000000
max        59.000000
```

Write output to CSV

Local or Google Storage is both handled

[46]:
```
# # write output file
write_csv_data(df_output, output_file, index=False)
```

```
Writing file to local directory
File:   schedule_time_features.csv
```

9.4 Overview of the output data

[47]:
```
df_output.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 487716 entries, 0 to 487715
Data columns (total 9 columns):
id            487716 non-null int64
dayofweek     487716 non-null int64
quarter       487716 non-null int64
month         487716 non-null int64
year          487716 non-null int64
dayofmonth    487716 non-null int64
weekofyear    487716 non-null int64
hour          487716 non-null int64
minutes       487716 non-null int64
dtypes: int64(9)
memory usage: 33.5 MB
```

[ ]:

CHAPTER 10

Features from destinations

- Takes the flights data and airports data
- destinations are in a list in the flights data
- Clean the 'destination' of flights so that we list all intermediate destinations and first/last clearly
- Merge with airports data to include airport data in our model input

TODO:

- Clearly a waste of space with all the copied values for each airport ID, restructure pipeline so that isn't required
- I imagine that the airports data is pretty static and we could expect a table on DataBricks

Due to time-constraints I use this solution so that we can keep merging/indexing only by `id`. In a more sophisticated approach we could merge downstream by other columns, such as `destination` on `airport` - therefore not having to include all duplicate features and save them to disk at this stage.

10.1 Parameters

• flights_file: Filepath of flights data in format received from Schiphol - airports_file: Filepath of airports data from Schiphol - output_file: Filepath to write output csv file with minimal modelling input
Returns

Output CSV file with minimal model input

```
id                  |  aircraftRegistration  |  airlineCode  |  terminal  | ␣
↪serviceType  |      scheduleDateTime        |     actualOffBlockTime      | ␣
↪scheduleDelaySeconds
124257473326719795   |    PHEXI               |     80.0      |     2.0    |       J ␣
↪     |  2018-05-01 16:35:00+02:00   |  2018-05-01 16:58:16+02:00  |         1396.0
```

(continues on next page)

(continued from previous page)

```
124538476600837715   |    PHEXL              |    2481.0    |    1.0    |       J ␣
↪     |  2018-06-10 13:00:00+02:00   |  2018-06-10 13:11:25+02:00   |          685.0
123512829091050355   |    PHBQO              |    100.0     |    2.0    |       J ␣
↪     |  2018-01-15 10:15:00+01:00   |  2018-01-15 10:35:10+01:00   |          1210.0
123786805997701057   |    PHEXG              |    2481.0    |    1.0    |       J ␣
↪     |  2018-02-23 17:45:00+01:00   |  2018-02-23 17:55:52+01:00   |          652.0
124664922607744671   |    PHBXP              |    1551.0    |    2.0    |       J ␣
↪     |  2018-06-28 20:50:00+02:00   |  2018-06-28 22:09:23+02:00   |          4763.0
```

id | final_destination | Country | City | Latitude | Longitude | Altitude | DST | destination_distance
123414478192901837 | AMS | Netherlands | Amsterdam | 52.308601 | 4.76389 | -11.0 | E | 4.338444e-07
123414481790516475 | AMS | Netherlands | Amsterdam | 52.308601 | 4.76389 | -11.0 | E | 4.338444e-07
123414478192901991 | AMS | Netherlands | Amsterdam | 52.308601 | 4.76389 | -11.0 | E | 4.338444e-07
123414481790510775 | AMS | Netherlands | Amsterdam | 52.308601 | 4.76389 | -11.0 | E | 4.338444e-07
123414479288269149 | SPC | Spain | Santa Cruz . . . | 28.626499 | -17.75560 | 107.0 | E | 3.267980e+01

10.2 File parameters

[3]:
```
# input parameters cell
flights_file = "../lvt-schiphol-assignment-snakemake/data/raw/flights.csv"
airports_file = "../lvt-schiphol-assignment-snakemake/data/raw/airports.csv"
output_file = "processed_flights.csv"
```

10.2.1 Libraries

[4]:
```
import pandas as pd
import numpy as np

import sys
sys.path.append("../")

from src.data.google_storage_io import read_csv_data, write_csv_data
```

10.2.2 Read data

[5]:
```
%%time
df_flights = read_csv_data(flights_file)
df_airports = read_csv_data(airports_file)
```

```
Reading file from local directory
File:   ../lvt-schiphol-assignment-snakemake/data/raw/flights.csv

Reading file from local directory
File:   ../lvt-schiphol-assignment-snakemake/data/raw/airports.csv

Wall time: 1.95 s
```

10.3 Destination features

- Route to destination in a column of lists
- Get final destination and number of destination

[15]:
```
df_flights
```

[15]:
```
                  actualOffBlockTime aircraftRegistration  \
0                                NaN                   NaN
1                                NaN                 PHPXY
2                                NaN                   NaN
3       2018-01-01T03:22:00.000+01:00                 PHPXB
4       2018-01-01T05:58:22.000+01:00                 PHHSJ
...                               ...                   ...
523270                           NaN                   NaN
523271                           NaN                   NaN
523272                           NaN                   NaN
523273                           NaN                   NaN
523274                           NaN                   NaN

       aircraftType.iatamain aircraftType.iatasub  airlineCode  baggageClaim  \
0                        NaN                  NaN        148.0           NaN
1                        AW1                  NaN        148.0           NaN
2                        AW1                  NaN        148.0           NaN
3                        NaN                  NaN        148.0           NaN
4                        73H                  73H        164.0           NaN
...                      ...                  ...          ...           ...
523270                   320                  320         64.0           NaN
523271                   319                  319         64.0           NaN
523272                   73H                  73H        234.0           NaN
523273                   73W                  73W        100.0           NaN
523274                   787                  788        794.0           NaN

        estimatedLandingTime           expectedTimeBoarding  \
0                        NaN                            NaN
1                        NaN                            NaN
2                        NaN                            NaN
3                        NaN                            NaN
4                        NaN                            NaN
...                      ...                            ...
523270                   NaN  2018-07-31T21:20:00.000+02:00
523271                   NaN  2018-07-31T21:25:00.000+02:00
523272                   NaN                            NaN
523273                   NaN  2018-07-31T21:40:00.000+02:00
523274                   NaN  2018-07-31T21:35:00.000+02:00

              expectedTimeGateClosing          expectedTimeGateOpen  ...  \
0                                 NaN                           NaN  ...
1                                 NaN                           NaN  ...
2                                 NaN                           NaN  ...
3                                 NaN                           NaN  ...
4                                 NaN                           NaN  ...
...                               ...                           ...  ...
523270  2018-07-31T21:30:00.000+02:00                           NaN  ...
523271  2018-07-31T21:35:00.000+02:00                           NaN  ...
523272                            NaN                           NaN  ...
523273  2018-07-31T21:55:00.000+02:00  2018-07-31T21:10:00.000+02:00  ...
```

(continues on next page)

(continued from previous page)

```
523274  2018-07-31T22:05:00.000+02:00  2018-07-31T20:30:00.000+02:00  ...

        prefixICAO publicEstimatedOffBlockTime publicFlightState.flightStates  \
0              ZXP                         NaN                        ['SCH']
1              ZXP                         NaN                        ['SCH']
2              ZXP                         NaN                        ['SCH']
3              ZXP                         NaN                        ['DEP']
4              TRA                         NaN                        ['DEP']
...            ...                         ...                            ...
523270         EZY                         NaN                        ['SCH']
523271         EZY                         NaN                        ['SCH']
523272         RYR                         NaN                        ['SCH']
523273         KLM                         NaN                        ['SCH']
523274         AMX                         NaN                        ['SCH']

        route.destinations scheduleDate  scheduleTime serviceType terminal  \
0                ['AMS']   2018-01-01      03:02:07           P      NaN
1                ['AMS']   2018-01-01      03:16:00         NaN      NaN
2                ['AMS']   2018-01-01      03:16:29           P      NaN
3                ['AMS']   2018-01-01      03:30:00         NaN      NaN
4                ['SPC']   2018-01-01      06:00:00           J      1.0
...                  ...          ...           ...         ...      ...
523270           ['MAN']   2018-07-31      21:50:00           J      NaN
523271           ['SEN']   2018-07-31      21:55:00           J      NaN
523272           ['DUB']   2018-07-31      22:05:00           J      NaN
523273           ['NCL']   2018-07-31      22:10:00           J      2.0
523274           ['MEX']   2018-07-31      22:25:00           J      NaN

       transferPositions transferPositions.transferPositions
0                    NaN                                NaN
1                    NaN                                NaN
2                    NaN                                NaN
3                    NaN                                NaN
4                    NaN                                NaN
...                  ...                                ...
523270               NaN                                NaN
523271               NaN                                NaN
523272               NaN                                NaN
523273               NaN                                NaN
523274               NaN                                NaN

[523275 rows x 28 columns]
```

[6]:
```
%%time

# route destinations parsed as a list then calculate length and expand list to columns
df_routes = df_flights[["id", "route.destinations"]] \
    .assign(route_list = lambda d: d["route.destinations"].apply(eval)) \
    .assign(route_length = lambda d: d["route_list"].apply(len),
            first_destination = lambda d: d["route_list"].apply(lambda x: x[0]),
            final_destination = lambda d: d["route_list"].apply(lambda x: x[-1]))

# determine separate route output columns
max_route_length = df_routes["route_length"].max()
destination_columns = [f"destination_{i}" for i in range(max_route_length)]
```

(continues on next page)

(continued from previous page)

```
# unlist routes into multiple columns
df_routes[destination_columns] =  pd.DataFrame(df_routes["route_list"] \
                                               .apply(lambda x: (x + [np.nan] * max_
↪route_length)[:max_route_length]).tolist(),
                                               index= df_routes.index)
df_routes
```

```
Wall time: 4.58 s
```

[6]:

```
                        id route.destinations route_list  route_length  \
0       123414478192901837            ['AMS']      [AMS]             1
1       123414481790516475            ['AMS']      [AMS]             1
2       123414478192901991            ['AMS']      [AMS]             1
3       123414481790510775            ['AMS']      [AMS]             1
4       123414479288269149            ['SPC']      [SPC]             1
...                    ...                ...        ...           ...
523270  124896773782315507            ['MAN']      [MAN]             1
523271  124896773782912169            ['SEN']      [SEN]             1
523272  124896745325235371            ['DUB']      [DUB]             1
523273  124896745995906173            ['NCL']      [NCL]             1
523274  124896744612750419            ['MEX']      [MEX]             1

       first_destination final_destination destination_0 destination_1  \
0                    AMS               AMS           AMS           NaN
1                    AMS               AMS           AMS           NaN
2                    AMS               AMS           AMS           NaN
3                    AMS               AMS           AMS           NaN
4                    SPC               SPC           SPC           NaN
...                  ...               ...           ...           ...
523270               MAN               MAN           MAN           NaN
523271               SEN               SEN           SEN           NaN
523272               DUB               DUB           DUB           NaN
523273               NCL               NCL           NCL           NaN
523274               MEX               MEX           MEX           NaN

       destination_2 destination_3 destination_4
0                NaN           NaN           NaN
1                NaN           NaN           NaN
2                NaN           NaN           NaN
3                NaN           NaN           NaN
4                NaN           NaN           NaN
...              ...           ...           ...
523270           NaN           NaN           NaN
523271           NaN           NaN           NaN
523272           NaN           NaN           NaN
523273           NaN           NaN           NaN
523274           NaN           NaN           NaN

[523275 rows x 11 columns]
```

[7]:

```
df_airports.head()
```

[7]:

```
   Airport                                      Name          City  \
0        1                            Goroka Airport        Goroka
1        2                            Madang Airport        Madang
2        3                Mount Hagen Kagamuga Airport   Mount Hagen
3        4                            Nadzab Airport        Nadzab
4        5  Port Moresby Jacksons International Airport  Port Moresby
```

(continues on next page)

(continued from previous page)

```
            Country IATA  ICAO  Latitude   Longitude  Altitude Timezone DST  \
0  Papua New Guinea  GKA  AYGA -6.081690  145.391998      5282       10   U
1  Papua New Guinea  MAG  AYMD -5.207080  145.789001        20       10   U
2  Papua New Guinea  HGU  AYMH -5.826790  144.296005      5388       10   U
3  Papua New Guinea  LAE  AYNZ -6.569803  146.725977       239       10   U
4  Papua New Guinea  POM  AYPY -9.443380  147.220001       146       10   U

                     Tz     Type       Source
0  Pacific/Port_Moresby  airport  OurAirports
1  Pacific/Port_Moresby  airport  OurAirports
2  Pacific/Port_Moresby  airport  OurAirports
3  Pacific/Port_Moresby  airport  OurAirports
4  Pacific/Port_Moresby  airport  OurAirports
```

[8]:
```
df_airports.query("IATA == 'AMS'")
```

[8]:
```
     Airport                         Name       City      Country IATA  ICAO  \
574      580  Amsterdam Airport Schiphol  Amsterdam  Netherlands  AMS  EHAM

      Latitude  Longitude  Altitude Timezone DST               Tz     Type  \
574  52.308601    4.76389       -11        1   E  Europe/Amsterdam  airport

          Source
574  OurAirports
```

[9]:
```
def distance_to_schiphol(lat, lon):
    """euclidean distance to hard-coded coords of Schiphol"""
    schiphol_coords = np.array([52.308601, 4.76389])
    dist = np.linalg.norm(np.array([lat, lon]) -  schiphol_coords)
    return dist
```

[10]:
```
%%time

df_final_destination_features = pd.merge(
    df_routes[["id", "final_destination"]],
    df_airports[["IATA", "Country", "City", "Latitude", "Longitude", "Altitude", "DST
↪", "Type"]],
    how = "left",
    left_on = ["final_destination"],
    right_on = ["IATA"])

df_final_destination_features = df_final_destination_features \
    .assign(destination_distance = lambda d: d[["Latitude", "Longitude"]] \
                                .apply(lambda x: distance_to_schiphol(lat=x[0],␣
↪lon=x[1]), axis=1)
            )
df_final_destination_features
```

```
Wall time: 26.3 s
```

[10]:
```
                      id final_destination IATA         Country  \
0       123414478192901837               AMS  AMS     Netherlands
1       123414481790516475               AMS  AMS     Netherlands
2       123414478192901991               AMS  AMS     Netherlands
3       123414481790510775               AMS  AMS     Netherlands
4       123414479288269149               SPC  SPC           Spain
```

(continues on next page)

(continued from previous page)

```
...                    ...               ...  ...             ...
523270  124896773782315507               MAN  MAN  United Kingdom
523271  124896773782912169               SEN  SEN  United Kingdom
523272  124896745325235371               DUB  DUB         Ireland
523273  124896745995906173               NCL  NCL  United Kingdom
523274  124896744612750419               MEX  MEX          Mexico

                          City   Latitude  Longitude  Altitude DST     Type  \
0                    Amsterdam  52.308601   4.763890     -11.0   E  airport
1                    Amsterdam  52.308601   4.763890     -11.0   E  airport
2                    Amsterdam  52.308601   4.763890     -11.0   E  airport
3                    Amsterdam  52.308601   4.763890     -11.0   E  airport
4       Santa Cruz De La Palma  28.626499 -17.755600     107.0   E  airport
...                        ...        ...        ...       ...  ..      ...
523270               Manchester  53.353699  -2.274950     257.0   E  airport
523271                 Southend  51.571400   0.695556      49.0   E  airport
523272                   Dublin  53.421299  -6.270070     242.0   E  airport
523273                Newcastle  55.037498  -1.691670     266.0   E  airport
523274              Mexico City  19.436300 -99.072098    7316.0   S  airport

        destination_distance
0               4.338444e-07
1               4.338444e-07
2               4.338444e-07
3               4.338444e-07
4               3.267980e+01
...                      ...
523270          7.116003e+00
523271          4.134587e+00
523272          1.108992e+01
523273          7.008647e+00
523274          1.089151e+02

[523275 rows x 11 columns]
```

[11]:
```
# meta columns for utility for columns we will often merge by
output_columns = ["id", "final_destination", "Country", "City", "Latitude", "Longitude
↪", "Altitude", "DST", "destination_distance"]

# DataFrame with id + merging columns + prediction target
df_output = df_final_destination_features[output_columns]
df_output.head()
```

[11]:
```
                  id final_destination      Country                  City  \
0  123414478192901837               AMS  Netherlands             Amsterdam
1  123414481790516475               AMS  Netherlands             Amsterdam
2  123414478192901991               AMS  Netherlands             Amsterdam
3  123414481790510775               AMS  Netherlands             Amsterdam
4  123414479288269149               SPC        Spain  Santa Cruz De La Palma

    Latitude  Longitude  Altitude DST  destination_distance
0  52.308601    4.76389     -11.0   E          4.338444e-07
1  52.308601    4.76389     -11.0   E          4.338444e-07
2  52.308601    4.76389     -11.0   E          4.338444e-07
3  52.308601    4.76389     -11.0   E          4.338444e-07
4  28.626499  -17.75560     107.0   E          3.267980e+01
```

10.4 Write output to CSV

Local or Google Storage is both handled.ipynb_checkpoints/

[12]:
```
# # write output file
write_csv_data(df_output, output_file, index=False)
```

```
Writing file to local directory
File:   processed_flights.csv
```

10.5 Overview of the output data

[13]:
```
df_output.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 523275 entries, 0 to 523274
Data columns (total 9 columns):
 #   Column                Non-Null Count   Dtype
---  ------                --------------   -----
 0   id                    523275 non-null  int64
 1   final_destination     523275 non-null  object
 2   Country               522336 non-null  object
 3   City                  522336 non-null  object
 4   Latitude              522336 non-null  float64
 5   Longitude             522336 non-null  float64
 6   Altitude              522336 non-null  float64
 7   DST                   522336 non-null  object
 8   destination_distance  522336 non-null  float64
dtypes: float64(4), int64(1), object(4)
memory usage: 39.9+ MB
```

[ ]:

CHAPTER 11

Rolling window features of delays

This notebook will take a DataFrame with `scheduleDateTime` or optionally a different column

1. Calculate time features from DateTime values
2. Output pd.DataFrame with id columns + time feature columns
3. Write output to CSV file

11.1 Parameters

• input_file: Filepath of model input data or flights data with column `scheduleDelaySeconds` • output_file: Filepath to CSV file to write output features to - freq: Frequency of rolling window features. Default `10T` for 10-minute intervals - window: Window size to calculate rolling window features. Default `2H` for 2-hour windows
The `window` argument is passed a pandas `DateOffset` frequency string. For valid arguments to `window` look here, https://pandas.pydata.org/pandas-docs/stable/user_guide/timeseries.html#dateoffset-objects
Returns
Output CSV file with rolling window features at every timestep from the beginning to the end of the `input_file` date range, at an interval of `freq`.
At each timestamp the average delay in the previous window based on the DateTime, with window size `window`.

Output format

```
timeStamp                   |    n    |  sum_delays  |  mean_delay
-------------------------------------------------------------------
2017-12-31 15:00:00+01:00   |   1.0   |     75600.0  |  75600.000000
2017-12-31 15:10:00+01:00   |   1.0   |     75600.0  |  75600.000000
2017-12-31 15:20:00+01:00   |   1.0   |     75600.0  |  75600.000000
2017-12-31 15:30:00+01:00   |   1.0   |     75600.0  |  75600.000000
2017-12-31 15:40:00+01:00   |   1.0   |     75600.0  |  75600.000000
```

(continues on next page)

(continued from previous page)

```
  ...                       |   ...   |       ...     |        ...
2018-07-12 17:10:00+02:00   |  379.0  |    223135.0   |    588.746702
2018-07-12 17:20:00+02:00   |  380.0  |    209786.0   |    552.068421
2018-07-12 17:30:00+02:00   |  363.0  |    207260.0   |    570.964187
2018-07-12 17:40:00+02:00   |  349.0  |    185570.0   |    531.719198
2018-07-12 17:50:00+02:00   |  336.0  |    164110.0   |    488.422619
```

11.1.1 Script parameters

[3]:
```
# parameters
input_file = "gs://lvt-schiphol-assignment-snakemake/data/model_input/delays_base_
↪input.csv"
output_file = "rolling_delay_features__test.csv"
freq = "10T"
window = "30T+2H+1D"
```

11.1.2 Imports

[4]:
```
import pandas as pd
import numpy as np

import sys
sys.path.append("../")

from src.data.google_storage_io import read_csv_data, write_csv_data
```

Load data

[5]:
```
%%time

df = read_csv_data(input_file)
df.head()
```

```
Reading file from Google Storage
Bucket: lvt-schiphol-assignment-snakemake
File:   data/model_input/delays_base_input.csv

Wall time: 2.44 s
```

[5]:

	id	aircraftRegistration	airlineCode	terminal	serviceType	scheduleDateTime	actualOffBlockTime	scheduleDelaySeconds
0	123414481790510775	PHPXB	148.0	NaN	NaN	2018-01-01 03:30:00+01:00	2018-01-01 03:22:00+01:00	-480.0
1	123414479288269149	PHHSJ	164.0	1.0	J	2018-01-01 06:00:00+01:00	2018-01-01 05:58:22+01:00	-98.0
2	123414479666542945	PHHSG	100.0	1.0	J	2018-01-01 06:05:00+01:00	2018-01-01 06:00:00+01:00	-300.0
3	123414479288365061	PHHSG	164.0	1.0	J	2018-01-01 06:05:00+01:00	2018-01-01 06:00:00+01:00	-300.0
4	123414479288274329	PHHXB	164.0	1.0	J	2018-01-01 06:15:00+01:00	2018-01-01 06:26:34+01:00	694.0

Rolling functions

After more development these should go into common modules to be reused.

[6]:
```
def bin_delays_by_schedule(df, freq='D'):
    # datetime floored to 10-minute interval
    df["timeStamp"] = pd.to_datetime(df["scheduleDateTime"], utc=True).dt.tz_convert(
↪"Europe/Amsterdam").dt.floor(freq)

    # groupby each floored datetime and keep n values + total delay per bin
    df_floor = df \
        .groupby("timeStamp")["scheduleDelaySeconds"] \
        .apply(lambda x: pd.DataFrame(dict(n=len(x), sum_delays=x.sum()), index=[x.
↪name])) \
        .reset_index(level=0) \
        .set_index("timeStamp")
    return df_floor

def get_rolling_mean_delay(df_floor, freq='D', window='2D'):

    # Not every 10 minute interval has flights, in those cases we fill in timestamps
    complete_date_range = pd.date_range(df_floor.index.min(), df_floor.index.max(),
                                        freq=freq)

    # list of dates to fill because they are not in df_floor yet
    tofill_date_range = complete_date_range[~complete_date_range.isin(df_floor.index)]

    # concatenate df_floor with missing timestamps using n=0, sum_delays=0
    df_floor = pd.concat([
        df_floor,
        pd.DataFrame(dict(n=0,
                          sum_delays=0,
                          timeStamp=tofill_date_range
                    )).set_index("timeStamp")
    ]).sort_values("timeStamp")

    # calculate mean from sum and total n. If n=0, we get a NaN mean and fill with 0
    df_rolling_mean = df_floor.rolling(window).sum().assign(
        mean_delay = lambda x: (x["sum_delays"] / x["n"].replace(0, 1)))

    return df_rolling_mean
```

Note on DATA LEAKAGE

There may be data leakage introduced in this features in the current implementation(!)

By ways of example, assume `window=2hours` and `freq=10minutes`.

- The time is now 18:30 and we want to predict delays for 2hours from now at 20:30
- We want to calculate the average delay of the last 2 hours
- A flight was scheduled at 18:00 and has still not left yet at 18:30
- In our data we know that this flight at 18:00 will have a delay of say 1 hour
- At the time of prediction, the only information we have is that the flight is **at least** 30minutes delayed

When calculating the average delay in our current implementation, we are leaking knowledge by using the fact that the flight at 18:00 had a 1-hour delay, even though this is not known at the time of prediction at 18:30(!)

Alternatively we can round delays of flights that have not yet left at the time of calculating mean-delay over the last `{window}`. In real-time we can then calculate the same values and therefore not introduce data leakage.

With more time permitted you could make a prediction of delay of flights that have not yet left, based on all Other features. Then use that prediction to fill in our rolling-window timeseries.

Calculate rolling features

First implementation took forever to determine rolling windows.

Implemented this step in PySpark but did not want to force a pyspark dependency unless we need to.

Then refactored in Pandas with sufficient speed. Current implementation does the trick.

[7]:
```
%%time
df_floor = bin_delays_by_schedule(df, freq=freq)
df_floor.head()
```

```
Wall time: 25 s
```

[7]:

timeStamp	n	sum_delays
2018-01-01 03:30:00+01:00	1	-480.0
2018-01-01 06:00:00+01:00	3	-698.0
2018-01-01 06:10:00+01:00	1	694.0
2018-01-01 06:20:00+01:00	5	2193.0
2018-01-01 06:30:00+01:00	5	2121.0

[8]:
```
%%time

if isinstance(window, list):
    window_list = window
elif '+' in window:
    window_list = window.split('+')
else:
    window_list = [window]

# add rolling features for all window sizes
df_rolling_features = pd.concat([get_rolling_mean_delay(df_floor, freq=freq,␣
↪window=window).add_suffix(f"__{window}")
                                 for window in window_list], axis=1) \
                      .reset_index()
df_rolling_features
```

```
Wall time: 47 ms
```

[8]:

	timeStamp	n__30T	sum_delays__30T	mean_delay__30T	\
0	2018-01-01 03:30:00+01:00	1.0	-480.0	-480.000000	
1	2018-01-01 03:40:00+01:00	1.0	-480.0	-480.000000	
2	2018-01-01 03:50:00+01:00	1.0	-480.0	-480.000000	
3	2018-01-01 04:00:00+01:00	0.0	0.0	0.000000	
4	2018-01-01 04:10:00+01:00	0.0	0.0	0.000000	
...	...	...	...	...	
27724	2018-07-12 17:10:00+02:00	102.0	51057.0	500.558824	
27725	2018-07-12 17:20:00+02:00	71.0	15875.0	223.591549	
27726	2018-07-12 17:30:00+02:00	42.0	9169.0	218.309524	
27727	2018-07-12 17:40:00+02:00	24.0	2197.0	91.541667	

(continues on next page)

(continued from previous page)

```
27728 2018-07-12 17:50:00+02:00     2.0         -10625.0     -5312.500000

       n__2H  sum_delays__2H  mean_delay__2H   n__1D  sum_delays__1D  \
0        1.0          -480.0     -480.000000     1.0          -480.0
1        1.0          -480.0     -480.000000     1.0          -480.0
2        1.0          -480.0     -480.000000     1.0          -480.0
3        1.0          -480.0     -480.000000     1.0          -480.0
4        1.0          -480.0     -480.000000     1.0          -480.0
...      ...             ...             ...     ...             ...
27724  379.0        223135.0      588.746702  2715.0       2022369.0
27725  380.0        209786.0      552.068421  2715.0       2025047.0
27726  363.0        207260.0      570.964187  2692.0       2015030.0
27727  349.0        185570.0      531.719198  2685.0       1998027.0
27728  336.0        164110.0      488.422619  2669.0       1986413.0

       mean_delay__1D
0         -480.000000
1         -480.000000
2         -480.000000
3         -480.000000
4         -480.000000
...               ...
27724      744.887293
27725      745.873665
27726      748.525260
27727      744.144134
27728      744.253653

[27729 rows x 10 columns]
```

Check assumption that we have no NaN values in our data

[9]:
```
assert df["scheduleDelaySeconds"].shape == df["scheduleDelaySeconds"].dropna().shape
df["scheduleDelaySeconds"].shape, df["scheduleDelaySeconds"].dropna().shape
```

[9]:
```
((477776,), (477776,))
```

[10]:
```
assert df_floor["sum_delays"].shape == df_floor["sum_delays"].dropna().shape
df_floor["sum_delays"].shape, df_floor["sum_delays"].dropna().shape
```

[10]:
```
((20532,), (20532,))
```

[11]:
```
# check separately for each
assert df_rolling_features.shape == df_rolling_features.dropna().shape
df_rolling_features.shape, df_rolling_features.dropna().shape
```

[11]:
```
((27729, 10), (27729, 10))
```

Show output features

[12]:
```
import matplotlib.pyplot as plt
import plotly.express as px
```

(continues on next page)

(continued from previous page)

```
import plotly.graph_objects as go

def px_facet_variable(df, x, y_cols, title=None, width=1200, height=800, log_
↪transform=False):
    # long format with variable values in 'y' and name in 'variable'
    df_plot = pd.concat([
        df[[x, y_col]].assign(variable=y_col).rename(columns={y_col: "y"})
        for y_col in y_cols
    ]).sort_values(x)

    if log_transform:
        df_plot["y"] = np.log(df_plot["y"] + 1)

    # facet plot of each variable in a row
    fig = px.line(df_plot[[x, "y", "variable"]], x=x, y="y", facet_row="variable",
                  title=f"{title}{' [Log-transformed]' if log_transform else None}",
                  width=width, height=height)

    # Add range slider assuming 'x' is a date or datetime index
    fig.update_layout(
        xaxis=dict(
            rangeslider=dict(
                visible=True
            ),
            type="date"
        ),
        hovermode="x"
    )

    # independent y-axes
    fig.update_yaxes(matches=None)
    fig.for_each_annotation(lambda a: a.update(text=a.text.split("=")[-1]))
    return fig

# plotly figure of rolling features
flatten = lambda l: [item for sublist in l for item in sublist]
n_mean_columns = flatten([[f"n__{w}", f"mean_delay__{w}"] for w in window_list])
px_facet_variable(df_rolling_features, "timeStamp", n_mean_columns,
                  title="Rolling window features (window={w})")
```

Data type cannot be displayed: application/vnd.plotly.v1+json, text/html

11.2 Write output to CSV

Local or Google Storage is both handled

[13]:
```
# # write output file
write_csv_data(df_rolling_features, output_file, index=False)
```

```
Writing file to local directory
File:   rolling_delay_features__test.csv
```

[14]:
```
df_rolling_features.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 27729 entries, 0 to 27728
Data columns (total 10 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   timeStamp        27729 non-null  datetime64[ns, Europe/Amsterdam]
 1   n__30T           27729 non-null  float64
 2   sum_delays__30T  27729 non-null  float64
 3   mean_delay__30T  27729 non-null  float64
 4   n__2H            27729 non-null  float64
 5   sum_delays__2H   27729 non-null  float64
 6   mean_delay__2H   27729 non-null  float64
 7   n__1D            27729 non-null  float64
 8   sum_delays__1D   27729 non-null  float64
 9   mean_delay__1D   27729 non-null  float64
dtypes: datetime64[ns, Europe/Amsterdam](1), float64(9)
memory usage: 2.1 MB
```

CHAPTER 12

Get holiday data from RijksOverheid

12.1 NOTE THIS SCRIPT HAS BROKEN

- Apparently you are no longer able to retrieve data from this source from before 2019-01-01 !

...

parse dataset from url: https://opendata.rijksoverheid.nl/v1/sources/rijksoverheid/infotypes/schoolholidays

Each of these groups may be used for separate models and use different external features.

Scrapes holidays from rijksoverheid.nl and writes them to CSV

```
Output format:
   ds         | Herfstvakantie_noord | Herfstvakantie_zuid  | Meivakantie_
↪heelNederland
   2014-10-10 |                  1.0 |                  0.0 | 0.0
   2014-10-11 |                  1.0 |                  0.0 | 0.0
   2014-10-12 |                  1.0 |                  0.0 | 0.0
   2014-10-13 |                  1.0 |                  0.0 | 0.0
   2014-10-14 |                  1.0 |                  0.0 | 0.0
   2014-10-15 |                  1.0 |                  0.0 | 0.0
```

12.1.1 Set parameters

Parameters

• output_file: String filepath of .csv file to write to - start_date: String first date to include in output data. Must be possible to parse with pandas.to_datetime() - end_date: String last date to include in output data. Must be possible to parse with pandas.to_datetime()
Returns

CSV file named `{output_file}` with 1 DateTime column `ds` and additional columns with binary holiday indicator values

[18]:
```
# parameters
output_file = "tmp_dump/scraped_holidays.csv"
start_date = "2018-01-01"
end_date = "2020-12-31"   # prefix to standardized filename
```

12.1.2 Load packages

Uses **conda environment: envs/schiphol-py.yml**

[19]:
```
import re
import urllib.request
from pathlib import Path
import xml.etree.ElementTree as ET
import pandas as pd
import numpy as np
```

12.1.3 Utility functions

[20]:
```
def parse_xml_schoolholidays(xml_string):
    """
    Function to parse dataset from url: https://opendata.rijksoverheid.nl/v1/sources/
↪rijksoverheid/infotypes/schoolholidays

    args:
        xml_string (str): string of xml document

    returns:
        pandas.DataFrame containing data from xml

    """
    def get_schoolyear_from_string(x):
        schoolyear = re.findall("20[0-9]+", x)[0]
        return schoolyear

    root = ET.fromstring(xml_string) # Make ElementTree root to make parsing of data␣
↪easy


    # In the following loops data from the xml document is taken, made into a␣
↪dictionary and a DataFrame is created

    data_vacations = []
    for document in root.findall("document"):
        schoolyear_raw = document.findall("content")[0].findall("contentblock")[0].
↪findall("schoolyear")[0].text
        schoolyear = get_schoolyear_from_string(schoolyear_raw)

        for vacations in document.findall("content")[0].findall("contentblock")[0].
↪findall("vacations"):
            for vacation in vacations.findall("vacation"):
                type_vakantie = vacation.findall("type")[0].text.strip()
```

(continues on next page)

(continued from previous page)

```
                for regio in vacation.findall("regions"):
                    region = regio.findall("region")[0].text
                    region_start_dt = regio.findall("startdate")[0].text
                    region_end_dt = regio.findall("enddate")[0].text

                    dict_region = {
                        "schoolyear":schoolyear,
                        "region": region,
                        "type_vakantie":type_vakantie,
                        "region_start_dt": region_start_dt,
                        "region_end_dt": region_end_dt
                    }
                    data_vacations.append(dict_region)

    df_schoolholidays = pd.DataFrame(data_vacations)
    return(df_schoolholidays)


def transform_schoolyear_data_to_long_format(df):
    """
    Very specific function to transform schoolyear data.
    Every date get's a row for every region with an indication (vakantie_ind) if the␣
↪date is a holiday.

    From:

        region | region_end_dt              | region_start_dt           | schoolyear |␣
↪type_vakantie
        noord  | 2017-10-29T22:59:00.000Z  | 2017-10-21T22:00:00.000Z | 2017       |␣
↪schoolvakanties
        midden | 2017-10-22T21:59:00.000Z  | 2017-10-14T22:00:00.000Z | 2017       |␣
↪schoolvakanties
        zuid   | 2017-10-22T21:59:00.000Z  | 2017-10-14T22:00:00.000Z | 2017       |␣
↪schoolvakanties

    To:
        index                      | type_vakantie   | region | vakantie_ind
        2017-10-21 00:00:00+00:00 | schoolvakanties | noord  | 1.0
        2017-10-22 00:00:00+00:00 | schoolvakanties | noord  | 1.0
        2017-10-23 00:00:00+00:00 | schoolvakanties | noord  | 1.0
        ...
        2017-10-29 00:00:00+00:00 | schoolvakanties | noord  | 1.0
        2017-10-31 00:00:00+00:00 | schoolvakanties | noord  | 0.0
        2017-10-01 00:00:00+00:00 | schoolvakanties | noord  | 0.0

    """
    list_df_regions = []
    for row in df.iterrows():
        record = row[1]

        start = record["region_start_dt"]
        end = record["region_end_dt"]
        schoolyear = record["schoolyear"]
        region = record["region"]
        type_vakantie = record["type_vakantie"]
```

(continues on next page)

(continued from previous page)

```
        index = pd.date_range(start=start, end=end)

        n_periods = len(index)

        region_list = [region] * n_periods
        type_vakantie_list = [type_vakantie] * n_periods

        dict_data = {
            "type_vakantie": type_vakantie_list,
            "region": region_list,
            "vakantie_ind": 1,
        }

        df_region = pd.DataFrame(index=index, data=dict_data)

        list_df_regions.append(df_region)

    df_holidays = pd.concat(list_df_regions)

    list_df_resampled = []
    for region in df_holidays["region"].unique():
        type_vakantie = df_holidays["type_vakantie"][0]

        df = df_holidays[df_holidays["region"]== region].resample("D").max()
        df["type_vakantie"] = df["type_vakantie"].fillna(type_vakantie)
        df["region"] = df["region"].fillna(region)
        df["vakantie_ind"] = df["vakantie_ind"].fillna(0)
        list_df_resampled.append(df)

    df_holidays = pd.concat(list_df_resampled)

    return df_holidays


def pivot_df_holidays_long(df):
    """
    Specific function to pivot dataframe holidays in long format.

    Input format:
        ds         | type_vakantie  | region | vakantie_ind
        2014-10-10 | Herfstvakantie | noord  | 1.0
        2014-10-11 | Herfstvakantie | noord  | 1.0
        2014-10-12 | Herfstvakantie | noord  | 1.0
        2014-10-13 | Herfstvakantie | noord  | 1.0
        2014-10-14 | Herfstvakantie | noord  | 1.0
        2014-10-15 | Herfstvakantie | noord  | 1.0
        ...
        2022-04-08 | Herfstvakantie | heel Nederland | 0.0
        2022-04-09 | Herfstvakantie | heel Nederland | 0.0
        2022-04-10 | Herfstvakantie | heel Nederland | 0


    Output format:
        ds         | Herfstvakantie_noord | Herfstvakantie_zuid  | Meivakantie_
↪heelNederland
        2014-10-10 |                  1.0 |                  0.0 | 0.0
        2014-10-11 |                  1.0 |                  0.0 | 0.0
```

(continues on next page)

(continued from previous page)

```
        2014-10-12 |                 1.0 |                 0.0 | 0.0
        2014-10-13 |                 1.0 |                 0.0 | 0.0
        2014-10-14 |                 1.0 |                 0.0 | 0.0
        2014-10-15 |                 1.0 |                 0.0 | 0.0

    """
    list_df_pivots = []
    for i, region in enumerate(df["region"].unique()):
        df_pivot = df[df["region"]==region].pivot(index="ds", columns="type_vakantie",
↪ values=["vakantie_ind"]).fillna(0)

        renamed_columns = []
        for column in df_pivot.columns.get_level_values(1):
            column_renamed = column + "_" + region.replace(" ", "")
            renamed_columns.append(column_renamed)

        df_pivot.columns = renamed_columns

        df_pivot = df_pivot.reset_index()

        list_df_pivots.append(df_pivot)

        if i == 0:
            df_holidays_pivot = df_pivot
        else:
            df_holidays_pivot = df_holidays_pivot.merge(df_pivot, on="ds")

    df_holidays_pivot = df_holidays_pivot.fillna(0)

    return df_holidays_pivot
```

12.2 Main step to gather and write holiday data

[21]:
```
import ssl

# fix certificate errors..
# https://stackoverflow.com/questions/35569042/ssl-certificate-verify-failed-with-
↪python3
ssl._create_default_https_context = ssl._create_unverified_context
```

[22]:
```
start, end
```

[22]:
```
(Timestamp('2018-01-01 00:00:00'), Timestamp('2018-12-31 00:00:00'))
```

[23]:
```
df
```

[23]:
```
   schoolyear          region      type_vakantie           region_start_dt  \
0        2019           noord     Herfstvakantie  2019-10-19T22:00:00.000Z
1        2019          midden     Herfstvakantie  2019-10-19T22:00:00.000Z
2        2019            zuid     Herfstvakantie  2019-10-12T22:00:00.000Z
3        2019  heel Nederland      Kerstvakantie  2019-12-21T23:00:00.000Z
4        2019           noord  Voorjaarsvakantie  2020-02-15T23:00:00.000Z
..        ...             ...                ...                       ...
72       2025            zuid  Voorjaarsvakantie  2026-02-14T23:00:00.000Z
```

(continues on next page)

(continued from previous page)

```
73       2025  heel Nederland        Meivakantie  2026-04-25T22:00:00.000Z
74       2025           noord      Zomervakantie  2026-07-04T22:00:00.000Z
75       2025          midden      Zomervakantie  2026-07-18T22:00:00.000Z
76       2025            zuid      Zomervakantie  2026-07-11T22:00:00.000Z

               region_end_dt
0   2019-10-27T21:59:00.000Z
1   2019-10-27T22:59:00.000Z
2   2019-10-20T22:59:00.000Z
3   2020-01-05T22:59:00.000Z
4   2020-02-23T22:59:00.000Z
..                       ...
72  2026-02-22T22:59:00.000Z
73  2026-05-03T21:59:00.000Z
74  2026-08-16T21:59:00.000Z
75  2026-08-30T21:59:00.000Z
76  2026-08-23T21:59:00.000Z

[77 rows x 5 columns]
```

[24]:
```
start, end = pd.to_datetime(start_date), pd.to_datetime(end_date)

url_schoolvakanties = "https://opendata.rijksoverheid.nl/v1/sources/rijksoverheid/
↪infotypes/schoolholidays"

# Get XML string
with urllib.request.urlopen(url_schoolvakanties) as url:
    xml_string = url.read()

# Parse to dataframe
df = parse_xml_schoolholidays(xml_string)

# Convert to feature data format with 'ds' dates column
df_holidays_long = transform_schoolyear_data_to_long_format(df)
df_holidays_long["ds"] = pd.to_datetime(df_holidays_long.index.date)
df_holidays_long_pivot = pivot_df_holidays_long(df_holidays_long)
df_holidays_long_pivot = df_holidays_long_pivot.drop(columns=["Herfstvakantie_
↪heelNederland"])
df_holidays_long_pivot = df_holidays_long_pivot[["ds"] + [col for col in df_holidays_
↪long_pivot.columns if col != "ds"]]

print("Subset holidays from period: %s to %s" % (start_date, end_date))
df_holidays_long_pivot = df_holidays_long_pivot \
    .query("ds >= '%s' and ds <= '%s'" % (start, end))

print("Collected %d holiday features: " % df_holidays_long_pivot.shape[1],
      df_holidays_long_pivot.columns.tolist())
df_holidays_long_pivot
```

```
Subset holidays from period: 2018-01-01 to 2020-12-31
Collected 12 holiday features:  ['ds', 'Herfstvakantie_noord', 'Voorjaarsvakantie_
↪noord', 'Zomervakantie_noord', 'Herfstvakantie_midden', 'Voorjaarsvakantie_midden',
↪'Zomervakantie_midden', 'Herfstvakantie_zuid', 'Voorjaarsvakantie_zuid',
↪'Zomervakantie_zuid', 'Kerstvakantie_heelNederland', 'Meivakantie_heelNederland']
```

[24]:
```
            ds  Herfstvakantie_noord  Voorjaarsvakantie_noord  \
0   2019-12-21                   0.0                      0.0
```

(continues on next page)

(continued from previous page)

```
1   2019-12-22                  0.0                       0.0
2   2019-12-23                  0.0                       0.0
3   2019-12-24                  0.0                       0.0
4   2019-12-25                  0.0                       0.0
..         ...                  ...                       ...
372 2020-12-27                  0.0                       0.0
373 2020-12-28                  0.0                       0.0
374 2020-12-29                  0.0                       0.0
375 2020-12-30                  0.0                       0.0
376 2020-12-31                  0.0                       0.0

     Zomervakantie_noord  Herfstvakantie_midden  Voorjaarsvakantie_midden  \
0                    0.0                    0.0                       0.0
1                    0.0                    0.0                       0.0
2                    0.0                    0.0                       0.0
3                    0.0                    0.0                       0.0
4                    0.0                    0.0                       0.0
..                   ...                    ...                       ...
372                  0.0                    0.0                       0.0
373                  0.0                    0.0                       0.0
374                  0.0                    0.0                       0.0
375                  0.0                    0.0                       0.0
376                  0.0                    0.0                       0.0

     Zomervakantie_midden  Herfstvakantie_zuid  Voorjaarsvakantie_zuid  \
0                     0.0                  0.0                     0.0
1                     0.0                  0.0                     0.0
2                     0.0                  0.0                     0.0
3                     0.0                  0.0                     0.0
4                     0.0                  0.0                     0.0
..                    ...                  ...                     ...
372                   0.0                  0.0                     0.0
373                   0.0                  0.0                     0.0
374                   0.0                  0.0                     0.0
375                   0.0                  0.0                     0.0
376                   0.0                  0.0                     0.0

     Zomervakantie_zuid  Kerstvakantie_heelNederland  \
0                   0.0                          1.0
1                   0.0                          1.0
2                   0.0                          1.0
3                   0.0                          1.0
4                   0.0                          1.0
..                  ...                          ...
372                 0.0                          1.0
373                 0.0                          1.0
374                 0.0                          1.0
375                 0.0                          1.0
376                 0.0                          1.0

     Meivakantie_heelNederland
0                          0.0
1                          0.0
2                          0.0
3                          0.0
4                          0.0
..                         ...
```

(continues on next page)

(continued from previous page)

```
372                         0.0
373                         0.0
374                         0.0
375                         0.0
376                         0.0

[377 rows x 12 columns]
```

[25]:
```
print("Writing holiday feature data to: %s" % output_file)
df_holidays_long_pivot.to_csv(output_file, header=True, index=False)
```

```
Writing holiday feature data to: tmp_dump/scraped_holidays.csv
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-25-b8de810df7fe> in <module>
      1 print("Writing holiday feature data to: %s" % output_file)
----> 2 df_holidays_long_pivot.to_csv(output_file, header=True, index=False)

~\anaconda3\envs\schiphol-py\lib\site-packages\pandas\core\generic.py in to_csv(self,␣
↪path_or_buf, sep, na_rep, float_format, columns, header, index, index_label, mode,␣
↪encoding, compression, quoting, quotechar, line_terminator, chunksize, date_format,␣
↪doublequote, escapechar, decimal)
   3226             decimal=decimal,
   3227         )
-> 3228         formatter.save()
   3229
   3230         if path_or_buf is None:

~\anaconda3\envs\schiphol-py\lib\site-packages\pandas\io\formats\csvs.py in save(self)
    181                 self.mode,
    182                 encoding=self.encoding,
--> 183                 compression=self.compression,
    184             )
    185             close = True

~\anaconda3\envs\schiphol-py\lib\site-packages\pandas\io\common.py in _get_
↪handle(path_or_buf, mode, encoding, compression, memory_map, is_text)
    397         if encoding:
    398             # Encoding
--> 399             f = open(path_or_buf, mode, encoding=encoding, newline="")
    400         elif is_text:
    401             # No explicit encoding

FileNotFoundError: [Errno 2] No such file or directory: 'tmp_dump/scraped_holidays.csv
↪'
```

12.3 Assert correct output

1. Check start/end dates of output match input parameters
2. Check all date differences are exactly 1 day

[31]:
```
# Assert some assumptions about the data
df_date_diffs = df_holidays_long_pivot \
```

(continues on next page)

(continued from previous page)

```
    .sort_values("ds") \
    ["ds"].diff().dt.days

# assert start and end date
df_start_date, df_end_date = min(df_holidays_long_pivot["ds"]), max(df_holidays_long_
↪pivot["ds"])
if df_start_date == pd.to_datetime(start_date) and df_end_date == pd.to_datetime(end_
↪date):
    print("SUCCESS: Output data min/max date as expected")
else:
    # todo: show exactly where we found unexpected results
    print("FAILURE: Output data min/max date don't match specified parameters")

# assert that all dates per group are exactly 1 day apart
date_diff_ok = all(df_date_diffs.fillna(1) == 1)
if date_diff_ok:
    print("SUCCESS: Date difference between all observations exactly 1 day")
else:
    # todo: show exactly where we found unexpected results
    print("FAILURE: Date differences not all exactly 1 day!")
```

```
SUCCESS: Output data min/max date as expected
SUCCESS: Date difference between all observations exactly 1 day
```

12.4 Information about processed holiday data

[1]:
```
# unsplit data
df_holidays_long_pivot.head()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-b4a1da2c784b> in <module>
      1 # unsplit data
----> 2 df_holidays_long_pivot.head()

NameError: name 'df_holidays_long_pivot' is not defined
```

[33]:
```
# describe all columns of the unsplit data
df_holidays_long_pivot.describe(include='all')
```

[33]:

	ds	Herfstvakantie_noord	Voorjaarsvakantie_noord	\
count	1826	1826.000000	1826.000000	
unique	1826	NaN	NaN	
top	2019-07-06 00:00:00	NaN	NaN	
freq	1	NaN	NaN	
first	2017-01-01 00:00:00	NaN	NaN	
last	2021-12-31 00:00:00	NaN	NaN	
mean	NaN	0.023001	0.021906	
std	NaN	0.149948	0.146416	
min	NaN	0.000000	0.000000	
25%	NaN	0.000000	0.000000	
50%	NaN	0.000000	0.000000	
75%	NaN	0.000000	0.000000	
max	NaN	1.000000	1.000000	

(continues on next page)

(continued from previous page)

```
        Zomervakantie_noord  Herfstvakantie_midden  Voorjaarsvakantie_midden  \
count           1826.000000            1826.000000                1826.000000
unique                  NaN                    NaN                        NaN
top                     NaN                    NaN                        NaN
freq                    NaN                    NaN                        NaN
first                   NaN                    NaN                        NaN
last                    NaN                    NaN                        NaN
mean               0.118291               0.024096                   0.021906
std                0.323041               0.153390                   0.146416
min                0.000000               0.000000                   0.000000
25%                0.000000               0.000000                   0.000000
50%                0.000000               0.000000                   0.000000
75%                0.000000               0.000000                   0.000000
max                1.000000               1.000000                   1.000000

        Zomervakantie_midden  Herfstvakantie_zuid  Voorjaarsvakantie_zuid  \
count             1826.000000          1826.000000             1826.000000
unique                    NaN                  NaN                     NaN
top                       NaN                  NaN                     NaN
freq                      NaN                  NaN                     NaN
first                     NaN                  NaN                     NaN
last                      NaN                  NaN                     NaN
mean                 0.118291             0.023549                0.022453
std                  0.323041             0.151680                0.148194
min                  0.000000             0.000000                0.000000
25%                  0.000000             0.000000                0.000000
50%                  0.000000             0.000000                0.000000
75%                  0.000000             0.000000                0.000000
max                  1.000000             1.000000                1.000000

        Zomervakantie_zuid  Kerstvakantie_heelNederland  \
count          1826.000000                  1826.000000
unique                 NaN                          NaN
top                    NaN                          NaN
freq                   NaN                          NaN
first                  NaN                          NaN
last                   NaN                          NaN
mean              0.118291                     0.040526
std               0.323041                     0.197243
min               0.000000                     0.000000
25%               0.000000                     0.000000
50%               0.000000                     0.000000
75%               0.000000                     0.000000
max               1.000000                     1.000000

        Meivakantie_heelNederland
count                 1826.000000
unique                        NaN
top                           NaN
freq                          NaN
first                         NaN
last                          NaN
mean                     0.021906
std                      0.146416
min                      0.000000
25%                      0.000000
50%                      0.000000
```

(continues on next page)

(continued from previous page)

```
75%                         0.000000
max                         1.000000
```

CHAPTER 13

Fit baseline model using flight delay averages

- Takes the model input data for flight delays
- Split data based on external train/test data file
- Define baseline average model
- Evaluate model
- Save model as pickle
- – save to mlflow –
- Write prediction prediction output to csv

13.1 Parameters

- input_file: Filepath of model input data of flight delays
- train_test_file: Filepath of train/test csv file with columns [“id”, “model_set”]
- output_file: Filepath to write output csv file with minimal modelling input

13.2 Returns

Trained baseline model that simply predicts the average flight delay from the training data in all predictions.

[42]:
```
# model params
input_file = "../lvt-schiphol-assignment-snakemake/data/model_input/delays_base_input.
↪csv"
train_test_file = "../lvt-schiphol-assignment-snakemake/data/model_input/train_test__
↪0.2__sample.csv"
output_predictions = "./predictions.csv"

# mlflow params
log_mlflow = True
mlflow_tracking_uri = "../mlruns"
mlflow_experiment = "from_script"
mlflow_run = "baseline_avg"
```

[43]:
```
from pathlib import Path
output_dir = Path(output_predictions).parent.absolute()
output_dir
```

[43]:
```
WindowsPath('C:/Users/lodew/qualogy/schiphol-code-assignment/scripts')
```

13.2.1 Imports

[44]:
```
import pandas as pd
import numpy as np
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
from sklearn.base import BaseEstimator, TransformerMixin

import matplotlib.pyplot as plt
import seaborn as sns

import sys
sys.path.append("../")

from src.data.google_storage_io import read_csv_data, write_csv_data
from src.evaluation.metrics import get_regression_metrics
from src.evaluation.regression import make_regression_metrics_by_group, make_
↪regression_metrics_by_datetime
from src.evaluation.predictions import make_predictions_dataframe
```

[45]:
```
plt.rcParams["figure.figsize"] = (16, 8)
```

13.3 Read data

[46]:
```
%%time
df = read_csv_data(input_file)
train_test = read_csv_data(train_test_file)
```

```
Reading file from local directory
File:   ../lvt-schiphol-assignment-snakemake/data/model_input/delays_base_input.csv

Reading file from local directory
```

(continues on next page)

(continued from previous page)

```
File:   ../lvt-schiphol-assignment-snakemake/data/model_input/train_test__0.2__sample.
↪csv

Wall time: 727 ms
```

[47]:
```
%%time

def split_train_test(df, train_test, target="scheduleDelaySeconds"):
    # merge by `id` and group by train/test set labels
    df_set_groups = pd.merge(df, train_test, on="id", how="left").groupby("model_set")

    # get data per train/test set
    df_train, df_test = df_set_groups.get_group("train"), df_set_groups.get_group(
↪"test")

    # split target from features
    X_train, y_train = df_train.drop(columns=[target]), df_train[target]
    X_test, y_test = df_test.drop(columns=[target]),  df_test[target]

    print(f"""
        Split data shapes
        Input: {df.shape}
        Train: {X_train.shape},\t {y_train.shape}
        Test:  {X_test.shape},\t {y_test.shape}
        """)

    # assert that we haven't dropped values at this stage
    # failed assert could indicate duplicate ids found in the data
    assert (len(X_train) + len(X_test)) == len(df)

    return X_train, X_test, y_train, y_test

# split data
X_train, X_test, y_train, y_test = split_train_test(df, train_test)
```

```
        Split data shapes
        Input: (477776, 8)
        Train: (382220, 8),      (382220,)
        Test:  (95556, 8),       (95556,)

Wall time: 536 ms
```

13.4 Prediction model

13.4.1 Define model

[48]:
```
class AverageBaseline(BaseEstimator):
    def __init__(self):
        self._average_y = None
```

(continues on next page)

(continued from previous page)

```
    @property
    def average_y(self):
        return self._average_y

    def fit(self, X, y):
        """calculate the average values of `y` and save  internally"""
        self._average_y = np.mean(y)
        return self

    def predict(self, X):
        """return trained average y value for all observations in X"""
        return np.array([self.average_y] * X.shape[0])
```

13.4.2 Train model

[49]:
```
# train
avg_baseline = AverageBaseline().fit(X_train, y_train)
```

13.4.3 Evaluate model

[50]:
```
def datetime_to_date(dt):
    return pd.to_datetime(dt, utc=True).dt.date

def datetime_to_date_hour(dt):
    return pd.to_datetime(dt, utc=True).dt.floor('H')
```

[51]:
```
# create predictions on train/test sets
df_predictions = make_predictions_dataframe(avg_baseline, X_train, X_test, y_train, y_
↪test)
df_predictions
```

[51]:
```
                        id           scheduleDateTime        y         yhat  \
1       123414479288269149  2018-01-01 06:00:00+01:00   -98.0  859.825258
2       123414479666542945  2018-01-01 06:05:00+01:00  -300.0  859.825258
5       123414479666545913  2018-01-01 06:20:00+01:00   611.0  859.825258
6       123414478696233855  2018-01-01 06:20:00+01:00   611.0  859.825258
7       123414479288370681  2018-01-01 06:20:00+01:00   180.0  859.825258
...                    ...                        ...     ...         ...
477742  124763275285891683  2018-07-12 17:15:00+02:00   115.0  859.825258
477748  124763299563775951  2018-07-12 17:15:00+02:00  -144.0  859.825258
477761  124763272032454817  2018-07-12 17:20:00+02:00   423.0  859.825258
477765  124763271776654663  2018-07-12 17:25:00+02:00    80.0  859.825258
477775  124763271129903067  2018-07-12 17:50:00+02:00 -8690.0  859.825258

              error model_set
1        957.825258     train
2       1159.825258     train
5        248.825258     train
6        248.825258     train
7        679.825258     train
...             ...       ...
477742   744.825258      test
```

(continues on next page)

(continued from previous page)

```
477748  1003.825258      test
477761   436.825258      test
477765   779.825258      test
477775  9549.825258      test

[477776 rows x 6 columns]
```

13.4.4 Calculate regression metrics

[52]:
```
%%time

df_metrics_long = make_regression_metrics_by_group(df_predictions, group_cols = [
↪"model_set"])
df_daily_metrics_long = make_regression_metrics_by_datetime(df_predictions, freq="D",␣
↪alias="schedule_date")
df_hourly_metrics_long = make_regression_metrics_by_datetime(df_predictions, freq="H",
↪ alias="schedule_date")
```

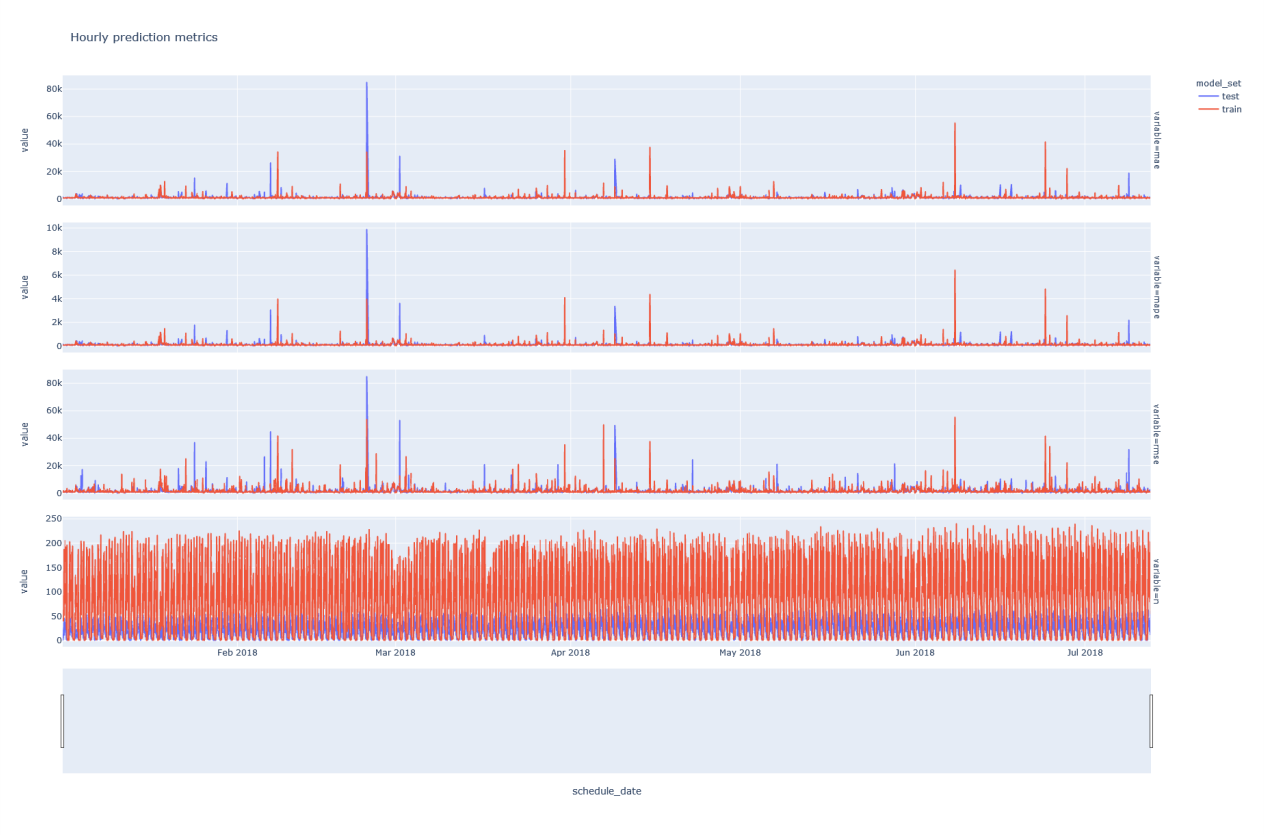
```
Wall time: 15.2 s
```

[53]:
```
df_metrics_long.head()
```

[53]:

	model_set	variable	value
0	test	mae	889.598834
1	train	mae	886.865285
2	test	mape	103.462747
3	train	mape	103.144828
4	test	rmse	2273.616920

[54]:
```
import plotly.express as px
fig = px.line(df_hourly_metrics_long, x="schedule_date", y="value", facet_row=
↪"variable", color="model_set",
              width=1200, height=1200, title="Hourly prediction metrics")
# Add range slider
fig.update_layout(
    xaxis=dict(
        rangeslider=dict(
            visible=True
        ),
        type="date"
    ),
    hovermode="x"
)
fig.update_yaxes(matches=None)
# fig.update_xaxes(matches=None)
fig.show()
```



13.5 Plot some prediction results

[55]:
```
# def predictions_daily_mean(df_predictions):
#     df_predictions["schedule_date"] = datetime_to_date(df_predictions[
↪"scheduleDateTime"])
#     df_predictions = df_predictions.drop(columns="id")
#     df_daily_mean = df_predictions.groupby(["model_set", "schedule_date"]).mean().
↪reset_index()
#     return df_daily_mean

# def predictions_hourly_mean(df_predictions):
#     df_predictions["schedule_date"] = datetime_to_date_hour(df_predictions[
↪"scheduleDateTime"])
#     df_predictions = df_predictions.drop(columns="id")
#     df_daily_mean = df_predictions.groupby(["model_set", "schedule_date"]).mean().
↪reset_index()
#     return df_daily_mean

# def get_safe_ylim(y, q=0.05, q2=None):
#     if q2 is None:
#         q2 = 1 - q
#     return (np.quantile(y, q), np.quantile(y, q2))


# df_daily_mean = predictions_daily_mean(df_predictions)
```

(continues on next page)

(continued from previous page)

```
# y_ylim = get_safe_ylim(df_daily_mean["y"])
# error_ylim = get_safe_ylim(df_daily_mean["error"])

# df_daily_mean[["schedule_date", "y", "yhat"]].plot(x="schedule_date", ylim=y_ylim)
# df_daily_mean[["schedule_date", "error"]].plot(x="schedule_date", ylim=error_ylim)

# df_hourly_mean = predictions_hourly_mean(df_predictions)
# y_ylim = get_safe_ylim(df_hourly_mean["y"])
# error_ylim = get_safe_ylim(df_hourly_mean["error"])

# df_hourly_mean[["schedule_date", "y", "yhat"]].plot(x="schedule_date", ylim=y_ylim)
# df_hourly_mean[["schedule_date", "error"]].plot(x="schedule_date", ylim=error_ylim)
```

13.6 Write output to output directory

[56]:
```
import joblib, pickle
from pathlib import Path
```

[58]:
```
model_file = str(Path(output_dir, "model.pkl"))
predictions_file = str(Path(output_dir, "predictions.csv"))
overall_metrics_file = str(Path(output_dir, "overall_metrics_long.csv"))
daily_metrics_file = str(Path(output_dir, "daily_metrics_long.csv"))
hourly_metrics_file = str(Path(output_dir, "hourly_metrics_long.csv"))
```

13.7 Pickle output files for mlflow artifacts

- Pipeline serialized with `joblib`
- Model data or sample thereof

[57]:
```
joblib.dump(avg_baseline, model_file)
```

[57]:
```
['C:\\Users\\lodew\\qualogy\\schiphol-code-assignment\\scripts\\model.pkl']
```

13.8 Write output to CSV

Local or Google Storage is both handled

[59]:
```
# write output file
write_csv_data(df_predictions,       predictions_file, index=False)
write_csv_data(df_metrics_long,       overall_metrics_file, index=False)
write_csv_data(df_daily_metrics_long, daily_metrics_file, index=False)
write_csv_data(df_hourly_metrics_long, hourly_metrics_file, index=False)
```

```
Writing file to local directory
File:   C:\Users\lodew\qualogy\schiphol-code-assignment\scripts\predictions.csv

Writing file to local directory
File:   C:\Users\lodew\qualogy\schiphol-code-assignment\scripts\overall_metrics_long.
↪csv
```

(continues on next page)

(continued from previous page)

```
Writing file to local directory
File:   C:\Users\lodew\qualogy\schiphol-code-assignment\scripts\daily_metrics_long.csv

Writing file to local directory
File:   C:\Users\lodew\qualogy\schiphol-code-assignment\scripts\hourly_metrics_long.
↪csv
```

13.9 Log to mlflow

[60]:
```
import mlflow

mlflow.set_tracking_uri(mlflow_tracking_uri)
mlflow.set_experiment(mlflow_experiment)


print(f"Logging to experiment: {mlflow_experiment}")
print(f"Run name: {mlflow_run}")

with mlflow.start_run(run_name=mlflow_run):
    mlflow.log_param("Input file", input_file)
    mlflow.log_param("Train-test file", train_test_file)

    # Model metadata
    for idx, metric_row in df_metrics_long.iterrows():
        metric_name = "__".join([metric_row["variable"], metric_row["model_set"]])
        mlflow.log_metric(metric_name, metric_row["value"])

    # log artifacts
    print("Logging artifacts")
    mlflow.log_artifact(model_file)
    mlflow.log_artifact(predictions_file)
    mlflow.log_artifact(overall_metrics_file)
    mlflow.log_artifact(daily_metrics_file)
    mlflow.log_artifact(hourly_metrics_file)
```

```
Logging to experiment: test_baseline_average
Run name: baseline_avg
Logging artifacts
```

13.10 Overview of the output data

[61]:
```
df_predictions.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 477776 entries, 1 to 477775
Data columns (total 7 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   id                477776 non-null  int64
```

(continues on next page)

(continued from previous page)

```
 1   scheduleDateTime  477776 non-null  object
 2   y                 477776 non-null  float64
 3   yhat              477776 non-null  float64
 4   error             477776 non-null  float64
 5   model_set         477776 non-null  object
 6   schedule_date     477776 non-null  datetime64[ns, UTC]
dtypes: datetime64[ns, UTC](1), float64(3), int64(1), object(2)
memory usage: 29.2+ MB
```

CHAPTER 14

Fit CatBoost model using extended features

- Takes the model input data for flight delays
- Split data based on external train/test data file
- Define catboost model
- Perform randomized search on selected parameter space
- Retrain model for more iterations using optimal parameters
- Save model as pickle
- – save to mlflow –
- Write prediction prediction output to csv

14.1 Parameters

- input_file: Filepath of model input data of flight delays
- train_test_file: Filepath of train/test csv file with columns [“id”, “model_set”]
- output_file: Filepath to write output csv file with minimal modelling input

14.2 Returns

Trained baseline model that simply predicts the average flight delay from the training data in all predictions.

[2]:
```
# model params
input_file = "../lvt-schiphol-assignment-snakemake/data/model_input/delays_extended_
↪input.csv"
train_test_file = "../lvt-schiphol-assignment-snakemake/data/model_input/train_test__
↪0.2__timeseries.csv"
output_predictions = "./predictions.csv"

# mlflow params
log_mlflow = True
mlflow_tracking_uri = "../mlruns"
mlflow_experiment = "from_script"
mlflow_run = "catboost_simple"
```

[3]:
```
from pathlib import Path
output_dir = Path(output_predictions).parent.absolute()
output_dir
```

[3]:
```
WindowsPath('C:/Users/lodew/qualogy/schiphol-code-assignment/scripts')
```

14.3 Imports

[4]:
```
import pandas as pd
import numpy as np
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
from sklearn.base import BaseEstimator, TransformerMixin

# catboost
from catboost import Pool
from catboost import CatBoostRegressor

import matplotlib.pyplot as plt
import seaborn as sns

import sys
sys.path.append("../")

from src.data.google_storage_io import read_csv_data, write_csv_data
from src.evaluation.metrics import get_regression_metrics
from src.evaluation.regression import make_regression_metrics_by_group, make_
↪regression_metrics_by_datetime
from src.evaluation.predictions import make_predictions_dataframe
```

[5]:
```
plt.rcParams["figure.figsize"] = (16, 8)
```

14.4 Read data

[6]:
```
%%time
df = read_csv_data(input_file)
train_test = read_csv_data(train_test_file)
```

```
Reading file from local directory
File:   ../lvt-schiphol-assignment-snakemake/data/model_input/delays_extended_input.
↪csv

Reading file from local directory
File:   ../lvt-schiphol-assignment-snakemake/data/model_input/train_test__0.2__
↪timeseries.csv

Wall time: 1.51 s
```

[7]:
```
%%time

def split_train_test(df, train_test, target="scheduleDelaySeconds"):
    # merge by `id` and group by train/test set labels
    df_set_groups = pd.merge(df, train_test, on="id", how="left").groupby("model_set")

    # get data per train/test set
    df_train = df_set_groups.get_group("train").drop(columns="model_set")
    df_test = df_set_groups.get_group("test").drop(columns="model_set")

    # split target from features
    X_train, y_train = df_train.drop(columns=[target]), df_train[target]
    X_test, y_test = df_test.drop(columns=[target]),  df_test[target]

    print(f"""
        Split data shapes
        Input: {df.shape}
        Train: {X_train.shape},\t {y_train.shape}
        Test:  {X_test.shape},\t {y_test.shape}
        """)

    # assert that we haven't dropped values at this stage
    # failed assert could indicate duplicate ids found in the data
    assert (len(X_train) + len(X_test)) == len(df)

    return X_train, X_test, y_train, y_test

# split data
X_train, X_test, y_train, y_test = split_train_test(df, train_test)
```

```
        Split data shapes
        Input: (487714, 24)
        Train: (389439, 23),     (389439,)
        Test:  (98275, 23),      (98275,)

Wall time: 1.1 s
```

14.4.1 Prediction model

14.5 Define model

[8]:
```
class CatBoostChain(BaseEstimator, TransformerMixin):
    """
    Catboost estimator that uses .transform() to append output predictions
    """
    def __init__(self, catboost_kwargs, early_stopping_rounds=None):
        self.catboost = CatBoostRegressor(**catboost_kwargs)
        self.early_stopping_rounds = early_stopping_rounds

    def fit(self, X, y, **kwargs):
        """
        Expects 'y' to be the race finish times- not avg_speed_left
        """
        # calculate avg speed left
        y =  ((X["distance"] - X["Passed"]) / (y.values - X["seconds"])) \
            .replace([np.inf, -np.inf], np.nan) \
            .fillna(0).values

        X_pool_train, X_pool_eval, y_pool_train, y_pool_eval = train_test_split(X, y,␣
↪test_size=0.2)

        train_pool = Pool(data=X_pool_train,
                          label=y_pool_train,
                          cat_features=self.catboost.get_param("cat_features"))
#                                  baseline=X_dist_train["yhat_finish_time"])

        eval_pool = Pool(data=X_pool_eval,
                         label=y_pool_eval,
                         cat_features=self.catboost.get_param("cat_features"))

        self.catboost.fit(train_pool, eval_set=eval_pool, early_stopping_rounds=self.
↪early_stopping_rounds, **kwargs)

        return self
```

14.6 Select features for catboost model

[9]:
```
columns_to_drop = ["id", "scheduleDateTime", "actualOffBlockTime", "year", "month",
↪"quarter"]
X_train_meta = X_train[["id", "scheduleDateTime"]]
X_test_meta = X_test[["id", "scheduleDateTime"]]

X_train = X_train[[col for col in X_train.columns if col not in columns_to_drop]]
X_test = X_test[[col for col in X_test.columns if col not in columns_to_drop]]

# type categorical features for catboost
cat_features = [
    'aircraftRegistration', 'airlineCode', 'terminal', 'serviceType',
    'final_destination', 'Country', 'City', 'DST',
    'dayofweek', 'dayofmonth', 'weekofyear', 'hour', 'minutes'
]
```

(continues on next page)

(continued from previous page)

```
X_train[cat_features] = X_train[cat_features].astype(str).astype('category')
X_test[cat_features] = X_test[cat_features].astype(str).astype('category')

assert all(X_test.columns == X_train.columns)

X_train.columns
```

[9]:
```
Index(['aircraftRegistration', 'airlineCode', 'terminal', 'serviceType',
       'final_destination', 'Country', 'City', 'Latitude', 'Longitude',
       'Altitude', 'DST', 'destination_distance', 'dayofweek', 'dayofmonth',
       'weekofyear', 'hour', 'minutes'],
      dtype='object')
```

14.7 Perform random search for optimal parameters

- Random search over grid search for faster results

[10]:
```
# create catboost input data
X_pool_train, X_pool_eval, y_pool_train, y_pool_eval = train_test_split(X_train, y_
↪train, test_size=0.2)

train_pool = Pool(data=X_pool_train,
                  label=y_pool_train,
                  cat_features=cat_features)

eval_pool = Pool(data=X_pool_eval,
                 label=y_pool_eval,
                 cat_features=cat_features)

# set initial catboost kwargs for random search
catboost_kwargs={
        "verbose": 1,
        "iterations": 10,
        "depth": 4,
        "learning_rate": 1,
        "loss_function": "MAE",
        "l2_leaf_reg": 4,
        "train_dir": str(Path(output_dir, "catboost_random_search")),
        "cat_features": cat_features}

# sensible values for random search after trial-error
grid = {'learning_rate': [0.1, 0.3, 0.5],
        'depth': [4, 6, 10],
        'l2_leaf_reg': [1, 3, 5, 7, 9]}
search_model = CatBoostRegressor(**catboost_kwargs)
randomized_search_result = search_model.randomized_search(grid,
                                                          X=train_pool,
                                                          plot=True)
```

```
<IPython.core.display.HTML object>
```

```
MetricVisualizer(layout=Layout(align_self='stretch', height='500px'))
```

```

bestTest = 804.7577782
```

(continues on next page)

(continued from previous page)

```
bestIteration = 9

0:      loss: 804.7577782       best: 804.7577782 (0)   total: 953ms    remaining: 8.
↪57s

bestTest = 763.8192678
bestIteration = 9

1:      loss: 763.8192678       best: 763.8192678 (1)   total: 1.7s     remaining: 6.
↪79s

bestTest = 757.4576161
bestIteration = 9

2:      loss: 757.4576161       best: 757.4576161 (2)   total: 2.42s    remaining: 5.
↪64s

bestTest = 763.8192678
bestIteration = 9

3:      loss: 763.8192678       best: 757.4576161 (2)   total: 3.13s    remaining: 4.
↪69s

bestTest = 757.4576161
bestIteration = 9

4:      loss: 757.4576161       best: 757.4576161 (2)   total: 3.85s    remaining: 3.
↪85s

bestTest = 763.8192678
bestIteration = 9

5:      loss: 763.8192678       best: 757.4576161 (2)   total: 4.59s    remaining: 3.
↪06s

bestTest = 757.4576161
bestIteration = 9

6:      loss: 757.4576161       best: 757.4576161 (2)   total: 5.29s    remaining: 2.
↪27s

bestTest = 800.7015278
bestIteration = 9

7:      loss: 800.7015278       best: 757.4576161 (2)   total: 6.15s    remaining: 1.
↪54s

bestTest = 756.1623348
bestIteration = 9

8:      loss: 756.1623348       best: 756.1623348 (8)   total: 7.01s    remaining:␣
↪779ms

bestTest = 792.3006764
bestIteration = 9

9:      loss: 792.3006764       best: 756.1623348 (8)   total: 8.13s    remaining: 0us
```

(continues on next page)

(continued from previous page)

```
Estimating final quality...
```

14.8 Train model

[11]:
```
# update catboost parameters with randomized search results
updated_catboost_kwargs = dict(
    catboost_kwargs,
    iterations = 200,
    train_dir = str(output_dir),
    **randomized_search_result['params'])

# create new catboost object with updated parameters
model = CatBoostRegressor(**updated_catboost_kwargs)
# train final model
model.fit(train_pool, eval_set = eval_pool, early_stopping_rounds=50, plot=True)
```

```
<IPython.core.display.HTML object>
```

```
MetricVisualizer(layout=Layout(align_self='stretch', height='500px'))
```

```
0:      learn: 756.4992915      test: 736.1896539       best: 736.1896539 (0)   total:
↪ 486ms    remaining: 1m 36s
1:      learn: 716.4095588      test: 689.1406306       best: 689.1406306 (1)   total:
↪ 1.01s    remaining: 1m 40s
2:      learn: 697.8411741      test: 668.3668963       best: 668.3668963 (2)   total:
↪ 1.27s    remaining: 1m 23s
3:      learn: 692.7714155      test: 663.3020848       best: 663.3020848 (3)   total:
↪ 1.63s    remaining: 1m 19s
4:      learn: 678.4410044      test: 645.9118846       best: 645.9118846 (4)   total:
↪ 2.06s    remaining: 1m 20s
5:      learn: 665.8758141      test: 631.6815334       best: 631.6815334 (5)   total:
↪ 2.36s    remaining: 1m 16s
6:      learn: 659.9978535      test: 624.8699020       best: 624.8699020 (6)   total:
↪ 2.82s    remaining: 1m 17s
7:      learn: 657.8774503      test: 622.9807212       best: 622.9807212 (7)   total:
↪ 3.06s    remaining: 1m 13s
8:      learn: 655.4896950      test: 620.3435796       best: 620.3435796 (8)   total:
↪ 3.36s    remaining: 1m 11s
9:      learn: 653.2327483      test: 618.2358122       best: 618.2358122 (9)   total:
↪ 3.6s     remaining: 1m 8s
10:     learn: 651.1965779      test: 616.2311812       best: 616.2311812 (10)  total:
↪ 3.94s    remaining: 1m 7s
11:     learn: 650.5140676      test: 615.5689544       best: 615.5689544 (11)  total:
↪ 4.26s    remaining: 1m 6s
12:     learn: 649.0516013      test: 613.7866925       best: 613.7866925 (12)  total:
↪ 4.63s    remaining: 1m 6s
13:     learn: 648.4775319      test: 613.2001549       best: 613.2001549 (13)  total:
↪ 4.95s    remaining: 1m 5s
14:     learn: 647.8060426      test: 612.5047631       best: 612.5047631 (14)  total:
↪ 5.35s    remaining: 1m 5s
15:     learn: 647.1291244      test: 611.8994895       best: 611.8994895 (15)  total:
↪ 5.63s    remaining: 1m 4s
16:     learn: 645.8213327      test: 610.5029021       best: 610.5029021 (16)  total:
↪ 5.98s    remaining: 1m 4s
17:     learn: 644.8812865      test: 609.5610178       best: 609.5610178 (17)  total:
↪ 6.39s    remaining: 1m 4s
```

(continues on next page)

(continued from previous page)

```
18:     learn: 644.5974207      test: 609.3279597       best: 609.3279597 (18)  total:
↪ 6.79s    remaining: 1m 4s
19:     learn: 644.1965673      test: 608.9084781       best: 608.9084781 (19)  total:
↪ 7.21s    remaining: 1m 4s
20:     learn: 642.3113860      test: 607.1499557       best: 607.1499557 (20)  total:
↪ 7.51s    remaining: 1m 3s
21:     learn: 640.8101524      test: 605.1263016       best: 605.1263016 (21)  total:
↪ 7.88s    remaining: 1m 3s
22:     learn: 639.1655080      test: 603.3016212       best: 603.3016212 (22)  total:
↪ 8.24s    remaining: 1m 3s
23:     learn: 638.9469651      test: 603.1433498       best: 603.1433498 (23)  total:
↪ 8.56s    remaining: 1m 2s
24:     learn: 638.7524524      test: 602.9733992       best: 602.9733992 (24)  total:
↪ 8.89s    remaining: 1m 2s
25:     learn: 638.4115124      test: 602.6926753       best: 602.6926753 (25)  total:
↪ 9.43s    remaining: 1m 3s
26:     learn: 637.9200366      test: 602.2545671       best: 602.2545671 (26)  total:
↪ 9.87s    remaining: 1m 3s
27:     learn: 637.7586167      test: 602.1181496       best: 602.1181496 (27)  total:
↪ 10.1s    remaining: 1m 2s
28:     learn: 637.6390371      test: 602.0260108       best: 602.0260108 (28)  total:
↪ 10.4s    remaining: 1m 1s
29:     learn: 637.4954822      test: 601.9018914       best: 601.9018914 (29)  total:
↪ 10.8s    remaining: 1m
30:     learn: 637.3643015      test: 601.7997421       best: 601.7997421 (30)  total:
↪ 11.1s    remaining: 1m
31:     learn: 637.2449287      test: 601.6952385       best: 601.6952385 (31)  total:
↪ 11.5s    remaining: 1m
32:     learn: 636.9031647      test: 601.3616772       best: 601.3616772 (32)  total:
↪ 11.8s    remaining: 59.9s
33:     learn: 636.6963216      test: 601.2079507       best: 601.2079507 (33)  total:
↪ 12.3s    remaining: 59.9s
34:     learn: 636.2495884      test: 600.8071639       best: 600.8071639 (34)  total:
↪ 12.6s    remaining: 59.3s
35:     learn: 635.9025845      test: 600.4896005       best: 600.4896005 (35)  total:
↪ 13s      remaining: 59.1s
36:     learn: 635.7190227      test: 600.3220853       best: 600.3220853 (36)  total:
↪ 13.3s    remaining: 58.4s
37:     learn: 635.5344418      test: 600.1087458       best: 600.1087458 (37)  total:
↪ 13.7s    remaining: 58.5s
38:     learn: 634.8079986      test: 599.4618361       best: 599.4618361 (38)  total:
↪ 14.1s    remaining: 58.4s
39:     learn: 634.4043165      test: 599.0290034       best: 599.0290034 (39)  total:
↪ 14.6s    remaining: 58.5s
40:     learn: 634.3035510      test: 598.9302567       best: 598.9302567 (40)  total:
↪ 15s      remaining: 58s
41:     learn: 634.1347057      test: 598.7614733       best: 598.7614733 (41)  total:
↪ 15.4s    remaining: 58s
42:     learn: 633.9306509      test: 598.6202432       best: 598.6202432 (42)  total:
↪ 15.7s    remaining: 57.4s
43:     learn: 633.7490183      test: 598.4112566       best: 598.4112566 (43)  total:
↪ 16.1s    remaining: 57.1s
44:     learn: 633.2496051      test: 598.0410102       best: 598.0410102 (44)  total:
↪ 16.6s    remaining: 57.1s
45:     learn: 632.7809645      test: 597.6645594       best: 597.6645594 (45)  total:
↪ 16.9s    remaining: 56.6s
46:     learn: 632.6504159      test: 597.5792268       best: 597.5792268 (46)  total:
↪ 17.3s    remaining: 56.5s
```

(continues on next page)

(continued from previous page)

```
47:     learn: 632.3499691      test: 597.2903086       best: 597.2903086 (47)  total:
↪ 17.8s     remaining: 56.2s
48:     learn: 632.0798885      test: 597.0893884       best: 597.0893884 (48)  total:
↪ 18.1s     remaining: 55.8s
49:     learn: 631.9518132      test: 596.9704911       best: 596.9704911 (49)  total:
↪ 18.6s     remaining: 55.8s
50:     learn: 631.8371254      test: 596.8837289       best: 596.8837289 (50)  total:
↪ 19s       remaining: 55.5s
51:     learn: 631.6555558      test: 596.7125866       best: 596.7125866 (51)  total:
↪ 19.4s     remaining: 55.2s
52:     learn: 631.3500733      test: 596.5806829       best: 596.5806829 (52)  total:
↪ 19.7s     remaining: 54.8s
53:     learn: 631.2160011      test: 596.4982592       best: 596.4982592 (53)  total:
↪ 20.1s     remaining: 54.4s
54:     learn: 631.1155251      test: 596.4052152       best: 596.4052152 (54)  total:
↪ 20.5s     remaining: 54.1s
55:     learn: 630.9322951      test: 596.2872326       best: 596.2872326 (55)  total:
↪ 20.9s     remaining: 53.7s
56:     learn: 630.8313626      test: 596.2052357       best: 596.2052357 (56)  total:
↪ 21.3s     remaining: 53.4s
57:     learn: 630.5880527      test: 596.0234432       best: 596.0234432 (57)  total:
↪ 21.6s     remaining: 52.8s
58:     learn: 630.4350416      test: 595.9243026       best: 595.9243026 (58)  total:
↪ 22s       remaining: 52.5s
59:     learn: 630.1765985      test: 595.6232071       best: 595.6232071 (59)  total:
↪ 22.3s     remaining: 52s
60:     learn: 629.9420989      test: 595.3269454       best: 595.3269454 (60)  total:
↪ 22.7s     remaining: 51.8s
61:     learn: 629.7855614      test: 595.1980082       best: 595.1980082 (61)  total:
↪ 23.1s     remaining: 51.5s
62:     learn: 629.6821856      test: 595.1140363       best: 595.1140363 (62)  total:
↪ 23.5s     remaining: 51.1s
63:     learn: 629.3721379      test: 594.7708591       best: 594.7708591 (63)  total:
↪ 23.9s     remaining: 50.8s
64:     learn: 629.2935584      test: 594.7005343       best: 594.7005343 (64)  total:
↪ 24.3s     remaining: 50.4s
65:     learn: 629.1437203      test: 594.5599510       best: 594.5599510 (65)  total:
↪ 24.6s     remaining: 50s
66:     learn: 629.0597314      test: 594.4965610       best: 594.4965610 (66)  total:
↪ 25s       remaining: 49.6s
67:     learn: 628.9604825      test: 594.4300063       best: 594.4300063 (67)  total:
↪ 25.3s     remaining: 49.2s
68:     learn: 628.6144429      test: 594.0584278       best: 594.0584278 (68)  total:
↪ 25.8s     remaining: 49s
69:     learn: 628.3600205      test: 593.7500512       best: 593.7500512 (69)  total:
↪ 26.3s     remaining: 48.8s
70:     learn: 627.9630414      test: 593.2422692       best: 593.2422692 (70)  total:
↪ 26.8s     remaining: 48.6s
71:     learn: 627.8194149      test: 593.1220129       best: 593.1220129 (71)  total:
↪ 27.1s     remaining: 48.1s
72:     learn: 627.6032628      test: 592.9470811       best: 592.9470811 (72)  total:
↪ 27.4s     remaining: 47.7s
73:     learn: 627.0955542      test: 592.5350740       best: 592.5350740 (73)  total:
↪ 27.8s     remaining: 47.3s
74:     learn: 626.9570505      test: 592.4246023       best: 592.4246023 (74)  total:
↪ 28.3s     remaining: 47.1s
75:     learn: 626.8580474      test: 592.3451068       best: 592.3451068 (75)  total:
↪ 28.6s     remaining: 46.6s
```

(continues on next page)

(continued from previous page)

```
76:     learn: 626.6240930      test: 592.1134100       best: 592.1134100 (76)  total:
↪ 29.2s    remaining: 46.6s
77:     learn: 626.5238310      test: 591.9830204       best: 591.9830204 (77)  total:
↪ 29.7s    remaining: 46.5s
78:     learn: 626.2930463      test: 591.7926196       best: 591.7926196 (78)  total:
↪ 30.1s    remaining: 46.2s
79:     learn: 626.1785614      test: 591.7165412       best: 591.7165412 (79)  total:
↪ 30.5s    remaining: 45.7s
80:     learn: 626.1020838      test: 591.6535367       best: 591.6535367 (80)  total:
↪ 30.8s    remaining: 45.3s
81:     learn: 626.0103928      test: 591.5821192       best: 591.5821192 (81)  total:
↪ 31.3s    remaining: 45s
82:     learn: 625.8423460      test: 591.4126421       best: 591.4126421 (82)  total:
↪ 31.8s    remaining: 44.8s
83:     learn: 625.7572063      test: 591.3751926       best: 591.3751926 (83)  total:
↪ 32.2s    remaining: 44.5s
84:     learn: 625.6648629      test: 591.2986017       best: 591.2986017 (84)  total:
↪ 32.6s    remaining: 44.1s
85:     learn: 624.8146807      test: 590.5119781       best: 590.5119781 (85)  total:
↪ 33s      remaining: 43.8s
86:     learn: 624.5953493      test: 590.2107524       best: 590.2107524 (86)  total:
↪ 33.4s    remaining: 43.3s
87:     learn: 624.3865336      test: 590.0259422       best: 590.0259422 (87)  total:
↪ 33.7s    remaining: 42.9s
88:     learn: 624.1381544      test: 589.7534605       best: 589.7534605 (88)  total:
↪ 34.2s    remaining: 42.7s
89:     learn: 624.0491247      test: 589.7121752       best: 589.7121752 (89)  total:
↪ 34.8s    remaining: 42.5s
90:     learn: 623.7640891      test: 589.4765438       best: 589.4765438 (90)  total:
↪ 35.2s    remaining: 42.1s
91:     learn: 623.6698931      test: 589.3804272       best: 589.3804272 (91)  total:
↪ 35.5s    remaining: 41.7s
92:     learn: 623.5908364      test: 589.3442882       best: 589.3442882 (92)  total:
↪ 35.8s    remaining: 41.2s
93:     learn: 623.5306758      test: 589.3090298       best: 589.3090298 (93)  total:
↪ 36.2s    remaining: 40.8s
94:     learn: 623.3227559      test: 589.0772829       best: 589.0772829 (94)  total:
↪ 36.5s    remaining: 40.3s
95:     learn: 623.1951940      test: 588.9860376       best: 588.9860376 (95)  total:
↪ 36.9s    remaining: 40s
96:     learn: 622.9879567      test: 588.8807494       best: 588.8807494 (96)  total:
↪ 37.2s    remaining: 39.5s
97:     learn: 622.8090526      test: 588.7342637       best: 588.7342637 (97)  total:
↪ 37.6s    remaining: 39.1s
98:     learn: 622.5838785      test: 588.7075079       best: 588.7075079 (98)  total:
↪ 37.9s    remaining: 38.7s
99:     learn: 622.4825836      test: 588.6086717       best: 588.6086717 (99)  total:
↪ 38.2s    remaining: 38.2s
100:    learn: 622.2791306      test: 588.3271311       best: 588.3271311 (100) total:
↪ 38.7s    remaining: 37.9s
101:    learn: 622.2020300      test: 588.2731782       best: 588.2731782 (101) total:
↪ 39.2s    remaining: 37.6s
102:    learn: 621.9820212      test: 588.1600497       best: 588.1600497 (102) total:
↪ 39.6s    remaining: 37.3s
103:    learn: 621.6717145      test: 588.0134407       best: 588.0134407 (103) total:
↪ 40s      remaining: 36.9s
104:    learn: 621.4919530      test: 587.8427733       best: 587.8427733 (104) total:
↪ 40.4s    remaining: 36.6s
```

(continues on next page)

(continued from previous page)

```
105:    learn: 621.4406313      test: 587.7976833       best: 587.7976833 (105) total:
↪ 40.8s    remaining: 36.2s
106:    learn: 620.8423283      test: 587.2740215       best: 587.2740215 (106) total:
↪ 41.2s    remaining: 35.8s
107:    learn: 620.7610185      test: 587.1424969       best: 587.1424969 (107) total:
↪ 41.5s    remaining: 35.4s
108:    learn: 620.6532462      test: 587.0451113       best: 587.0451113 (108) total:
↪ 41.9s    remaining: 34.9s
109:    learn: 620.5812943      test: 587.0190482       best: 587.0190482 (109) total:
↪ 42.3s    remaining: 34.6s
110:    learn: 619.9161984      test: 586.3519279       best: 586.3519279 (110) total:
↪ 42.6s    remaining: 34.1s
111:    learn: 619.8554406      test: 586.2735098       best: 586.2735098 (111) total:
↪ 43s      remaining: 33.8s
112:    learn: 619.7443666      test: 586.2191013       best: 586.2191013 (112) total:
↪ 43.4s    remaining: 33.4s
113:    learn: 619.6755077      test: 586.1544571       best: 586.1544571 (113) total:
↪ 43.8s    remaining: 33.1s
114:    learn: 619.5537904      test: 586.0764185       best: 586.0764185 (114) total:
↪ 44.3s    remaining: 32.8s
115:    learn: 619.2903614      test: 586.0431799       best: 586.0431799 (115) total:
↪ 44.7s    remaining: 32.4s
116:    learn: 619.2025000      test: 585.9869016       best: 585.9869016 (116) total:
↪ 45.1s    remaining: 32s
117:    learn: 618.7112464      test: 585.3193213       best: 585.3193213 (117) total:
↪ 45.5s    remaining: 31.6s
118:    learn: 618.5342821      test: 585.1961176       best: 585.1961176 (118) total:
↪ 45.9s    remaining: 31.3s
119:    learn: 618.3582463      test: 585.0500864       best: 585.0500864 (119) total:
↪ 46.3s    remaining: 30.9s
120:    learn: 618.2471939      test: 584.9732303       best: 584.9732303 (120) total:
↪ 46.8s    remaining: 30.5s
121:    learn: 618.0731285      test: 584.8395377       best: 584.8395377 (121) total:
↪ 47.2s    remaining: 30.2s
122:    learn: 618.0000942      test: 584.7620554       best: 584.7620554 (122) total:
↪ 47.8s    remaining: 29.9s
123:    learn: 617.9257270      test: 584.7316075       best: 584.7316075 (123) total:
↪ 48.3s    remaining: 29.6s
124:    learn: 617.7787591      test: 584.6011831       best: 584.6011831 (124) total:
↪ 48.8s    remaining: 29.3s
125:    learn: 617.6997120      test: 584.5398819       best: 584.5398819 (125) total:
↪ 49.2s    remaining: 28.9s
126:    learn: 617.5244009      test: 584.3529823       best: 584.3529823 (126) total:
↪ 49.6s    remaining: 28.5s
127:    learn: 617.2962930      test: 584.1669826       best: 584.1669826 (127) total:
↪ 50s      remaining: 28.1s
128:    learn: 617.1825785      test: 584.1385620       best: 584.1385620 (128) total:
↪ 50.4s    remaining: 27.7s
129:    learn: 617.1099402      test: 584.0210716       best: 584.0210716 (129) total:
↪ 50.8s    remaining: 27.3s
130:    learn: 617.0114904      test: 583.9671104       best: 583.9671104 (130) total:
↪ 51.2s    remaining: 27s
131:    learn: 616.8188259      test: 583.8113980       best: 583.8113980 (131) total:
↪ 51.5s    remaining: 26.5s
132:    learn: 616.7738320      test: 583.7953743       best: 583.7953743 (132) total:
↪ 51.9s    remaining: 26.2s
133:    learn: 616.5926336      test: 583.7213538       best: 583.7213538 (133) total:
↪ 52.3s    remaining: 25.8s
```

(continues on next page)

(continued from previous page)

```
134:    learn: 616.5084778      test: 583.7059402       best: 583.7059402 (134) total:
↪ 52.7s    remaining: 25.4s
135:    learn: 616.4312025      test: 583.6415595       best: 583.6415595 (135) total:
↪ 53.3s    remaining: 25.1s
136:    learn: 616.3240607      test: 583.5173499       best: 583.5173499 (136) total:
↪ 53.7s    remaining: 24.7s
137:    learn: 616.2783275      test: 583.5019564       best: 583.5019564 (137) total:
↪ 54.2s    remaining: 24.3s
138:    learn: 616.2333421      test: 583.4770307       best: 583.4770307 (138) total:
↪ 54.6s    remaining: 23.9s
139:    learn: 616.2008139      test: 583.4749795       best: 583.4749795 (139) total:
↪ 55s      remaining: 23.6s
140:    learn: 616.0859712      test: 583.3754610       best: 583.3754610 (140) total:
↪ 55.5s    remaining: 23.2s
141:    learn: 616.0157786      test: 583.3133219       best: 583.3133219 (141) total:
↪ 55.9s    remaining: 22.8s
142:    learn: 615.9341868      test: 583.2771837       best: 583.2771837 (142) total:
↪ 56.4s    remaining: 22.5s
143:    learn: 615.8869803      test: 583.2243588       best: 583.2243588 (143) total:
↪ 57s      remaining: 22.2s
144:    learn: 615.8266985      test: 583.2070106       best: 583.2070106 (144) total:
↪ 57.4s    remaining: 21.8s
145:    learn: 615.7423962      test: 583.1519266       best: 583.1519266 (145) total:
↪ 57.7s    remaining: 21.4s
146:    learn: 615.4880554      test: 582.9063992       best: 582.9063992 (146) total:
↪ 58.2s    remaining: 21s
147:    learn: 615.3819690      test: 582.8068397       best: 582.8068397 (147) total:
↪ 58.5s    remaining: 20.6s
148:    learn: 615.3397580      test: 582.7699536       best: 582.7699536 (148) total:
↪ 59s      remaining: 20.2s
149:    learn: 615.1919542      test: 582.6876877       best: 582.6876877 (149) total:
↪ 59.4s    remaining: 19.8s
150:    learn: 615.1626818      test: 582.6696281       best: 582.6696281 (150) total:
↪ 59.8s    remaining: 19.4s
151:    learn: 615.0243761      test: 582.5860315       best: 582.5860315 (151) total:
↪ 1m       remaining: 19s
152:    learn: 614.8813938      test: 582.5034830       best: 582.5034830 (152) total:
↪ 1m       remaining: 18.6s
153:    learn: 614.8605463      test: 582.4908016       best: 582.4908016 (153) total:
↪ 1m 1s    remaining: 18.2s
154:    learn: 614.7724205      test: 582.4161990       best: 582.4161990 (154) total:
↪ 1m 1s    remaining: 17.8s
155:    learn: 614.7300791      test: 582.4021538       best: 582.4021538 (155) total:
↪ 1m 1s    remaining: 17.4s
156:    learn: 614.3910751      test: 582.1378535       best: 582.1378535 (156) total:
↪ 1m 1s    remaining: 17s
157:    learn: 614.2645807      test: 582.0170679       best: 582.0170679 (157) total:
↪ 1m 2s    remaining: 16.5s
158:    learn: 614.1563803      test: 581.9316131       best: 581.9316131 (158) total:
↪ 1m 2s    remaining: 16.1s
159:    learn: 613.9578124      test: 581.7698210       best: 581.7698210 (159) total:
↪ 1m 2s    remaining: 15.7s
160:    learn: 613.7906887      test: 581.5912194       best: 581.5912194 (160) total:
↪ 1m 3s    remaining: 15.3s
161:    learn: 613.7044189      test: 581.5036286       best: 581.5036286 (161) total:
↪ 1m 3s    remaining: 14.9s
162:    learn: 613.5849392      test: 581.4266365       best: 581.4266365 (162) total:
↪ 1m 4s    remaining: 14.6s
```

(continues on next page)

(continued from previous page)

```
163:    learn: 613.3722696      test: 581.2292450       best: 581.2292450 (163) total:
↪ 1m 4s    remaining: 14.2s
164:    learn: 613.1556289      test: 581.1037576       best: 581.1037576 (164) total:
↪ 1m 5s    remaining: 13.8s
165:    learn: 613.0290732      test: 580.9696302       best: 580.9696302 (165) total:
↪ 1m 5s    remaining: 13.4s
166:    learn: 612.9695643      test: 580.9225562       best: 580.9225562 (166) total:
↪ 1m 5s    remaining: 13s
167:    learn: 612.9182516      test: 580.8944932       best: 580.8944932 (167) total:
↪ 1m 6s    remaining: 12.6s
168:    learn: 612.8759733      test: 580.8487442       best: 580.8487442 (168) total:
↪ 1m 6s    remaining: 12.2s
169:    learn: 612.7284283      test: 580.7110513       best: 580.7110513 (169) total:
↪ 1m 7s    remaining: 11.8s
170:    learn: 612.6469981      test: 580.6450549       best: 580.6450549 (170) total:
↪ 1m 7s    remaining: 11.4s
171:    learn: 612.4922876      test: 580.4813941       best: 580.4813941 (171) total:
↪ 1m 7s    remaining: 11s
172:    learn: 612.3682492      test: 580.4154322       best: 580.4154322 (172) total:
↪ 1m 8s    remaining: 10.6s
173:    learn: 612.1525769      test: 580.2475661       best: 580.2475661 (173) total:
↪ 1m 8s    remaining: 10.2s
174:    learn: 612.0985593      test: 580.2149161       best: 580.2149161 (174) total:
↪ 1m 8s    remaining: 9.85s
175:    learn: 611.9614065      test: 580.0693887       best: 580.0693887 (175) total:
↪ 1m 9s    remaining: 9.46s
176:    learn: 611.8774389      test: 580.0103718       best: 580.0103718 (176) total:
↪ 1m 9s    remaining: 9.07s
177:    learn: 611.7511840      test: 579.9108673       best: 579.9108673 (177) total:
↪ 1m 10s   remaining: 8.68s
178:    learn: 611.7252624      test: 579.9017319       best: 579.9017319 (178) total:
↪ 1m 10s   remaining: 8.29s
179:    learn: 611.6226393      test: 579.8035861       best: 579.8035861 (179) total:
↪ 1m 11s   remaining: 7.89s
180:    learn: 611.4549767      test: 579.6788475       best: 579.6788475 (180) total:
↪ 1m 11s   remaining: 7.51s
181:    learn: 611.3454769      test: 579.5541309       best: 579.5541309 (181) total:
↪ 1m 12s   remaining: 7.13s
182:    learn: 611.3054737      test: 579.5266276       best: 579.5266276 (182) total:
↪ 1m 12s   remaining: 6.73s
183:    learn: 611.2498650      test: 579.4875620       best: 579.4875620 (183) total:
↪ 1m 12s   remaining: 6.33s
184:    learn: 611.1533009      test: 579.3775673       best: 579.3775673 (184) total:
↪ 1m 13s   remaining: 5.92s
185:    learn: 610.9020615      test: 579.1870376       best: 579.1870376 (185) total:
↪ 1m 13s   remaining: 5.54s
186:    learn: 610.8316611      test: 579.0995204       best: 579.0995204 (186) total:
↪ 1m 14s   remaining: 5.16s
187:    learn: 610.7659888      test: 579.0790918       best: 579.0790918 (187) total:
↪ 1m 14s   remaining: 4.76s
188:    learn: 610.7128587      test: 579.0734995       best: 579.0734995 (188) total:
↪ 1m 15s   remaining: 4.37s
189:    learn: 610.6606708      test: 579.0397298       best: 579.0397298 (189) total:
↪ 1m 15s   remaining: 3.97s
190:    learn: 610.6550951      test: 579.0353334       best: 579.0353334 (190) total:
↪ 1m 15s   remaining: 3.58s
191:    learn: 610.5822829      test: 578.9660564       best: 578.9660564 (191) total:
↪ 1m 16s   remaining: 3.18s
```

(continues on next page)

(continued from previous page)

```
192:    learn: 610.5296158      test: 578.9261770       best: 578.9261770 (192) total:
↪ 1m 16s   remaining: 2.78s
193:    learn: 610.4417085      test: 578.8428978       best: 578.8428978 (193) total:
↪ 1m 17s   remaining: 2.39s
194:    learn: 610.3924358      test: 578.8206193       best: 578.8206193 (194) total:
↪ 1m 17s   remaining: 1.99s
195:    learn: 610.1421457      test: 578.5998934       best: 578.5998934 (195) total:
↪ 1m 17s   remaining: 1.59s
196:    learn: 610.0201709      test: 578.4677762       best: 578.4677762 (196) total:
↪ 1m 18s   remaining: 1.19s
197:    learn: 609.8806418      test: 578.3458600       best: 578.3458600 (197) total:
↪ 1m 18s   remaining: 795ms
198:    learn: 609.8425944      test: 578.3160699       best: 578.3160699 (198) total:
↪ 1m 18s   remaining: 397ms
199:    learn: 609.7698163      test: 578.3045065       best: 578.3045065 (199) total:
↪ 1m 19s   remaining: 0us

bestTest = 578.3045065
bestIteration = 199
```

[11]:
```
<catboost.core.CatBoostRegressor at 0x1a6c7b74c48>
```

14.9 Evaluate model

[12]:
```
def datetime_to_date(dt):
    return pd.to_datetime(dt, utc=True).dt.date

def datetime_to_date_hour(dt):
    return pd.to_datetime(dt, utc=True).dt.floor('H')
```

[13]:
```
# create predictions on train/test sets
df_predictions = make_predictions_dataframe(model, X_train, X_test, y_train, y_test,
                                            meta_train = X_train_meta,
                                            meta_test = X_test_meta)
df_predictions
```

[13]:

	id	scheduleDateTime	y	yhat	error	model_set
0	123414481790510775	2018-01-01 03:30:00+01:00	-480.0	-197.459328	282.540672	train
1	123414479288269149	2018-01-01 06:00:00+01:00	-98.0	-100.360379	-2.360379	train
2	123414479666542945	2018-01-01 06:05:00+01:00	-300.0	-120.601478		
3	123414479288365061	2018-01-01 06:05:00+01:00	-300.0	-121.494374		
4	123414479288274329	2018-01-01 06:15:00+01:00	694.0	176.538920		
...	...	...	...	...		
487709	124763270719624901	2018-07-12 17:25:00+02:00	80.0	280.244578		
487710	124763272032451639	2018-07-12 17:25:00+02:00	80.0	281.817452		
487711	124763270368084713	2018-07-12 17:25:00+02:00	12.0	278.095160		
487712	124763270625998761	2018-07-12 17:45:00+02:00	-1935.0	-2.120094		
487713	124763271129903067	2018-07-12 17:50:00+02:00	-8690.0	31.526172		

(continues on next page)

(continued from previous page)

```
2         179.398522     train
3         178.505626     train
4        -517.461080     train
...              ...       ...
487709    200.244578      test
487710    201.817452      test
487711    266.095160      test
487712   1932.879906      test
487713   8721.526172      test

[487714 rows x 6 columns]
```

[14]:
```
%%time

df_metrics_long = make_regression_metrics_by_group(df_predictions, group_cols = [
↪"model_set"])
df_daily_metrics_long = make_regression_metrics_by_datetime(df_predictions, freq="D",␣
↪alias="schedule_date")
df_hourly_metrics_long = make_regression_metrics_by_datetime(df_predictions, freq="H",
↪ alias="schedule_date")
```

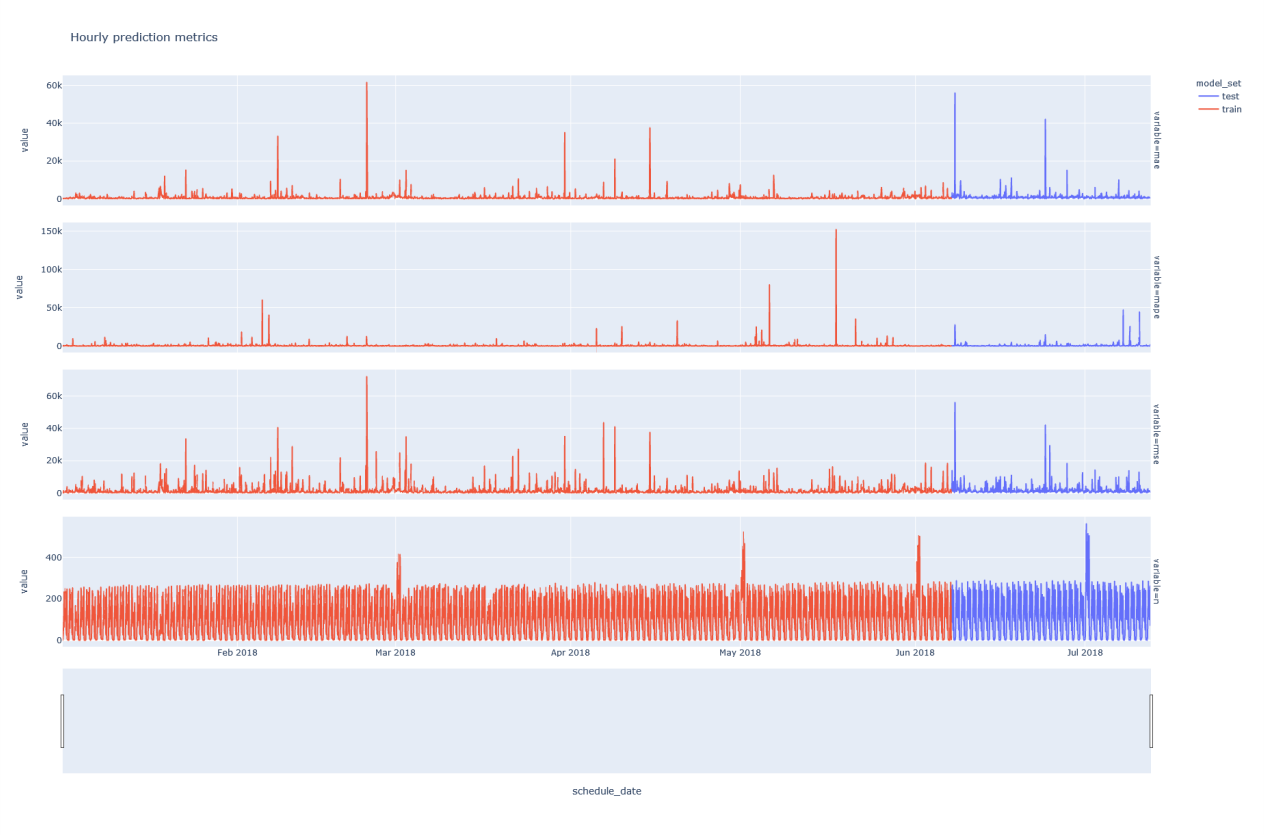
```
Wall time: 9.12 s
```

[15]:
```
df_metrics_long.head()
```

[15]:

	model_set	variable	value
0	test	mae	841.293334
1	train	mae	558.490217
2	test	mape	365.325530
3	train	mape	448.172767
4	test	rmse	2429.517313

[16]:
```
import plotly.express as px
fig = px.line(df_hourly_metrics_long, x="schedule_date", y="value", facet_row=
↪"variable", color="model_set",
              width=1200, height=1200, title="Hourly prediction metrics")
# Add range slider
fig.update_layout(
    xaxis=dict(
        rangeslider=dict(
            visible=True
        ),
        type="date"
    ),
    hovermode="x"
)
fig.update_yaxes(matches=None)
# fig.update_xaxes(matches=None)
fig.show()
```



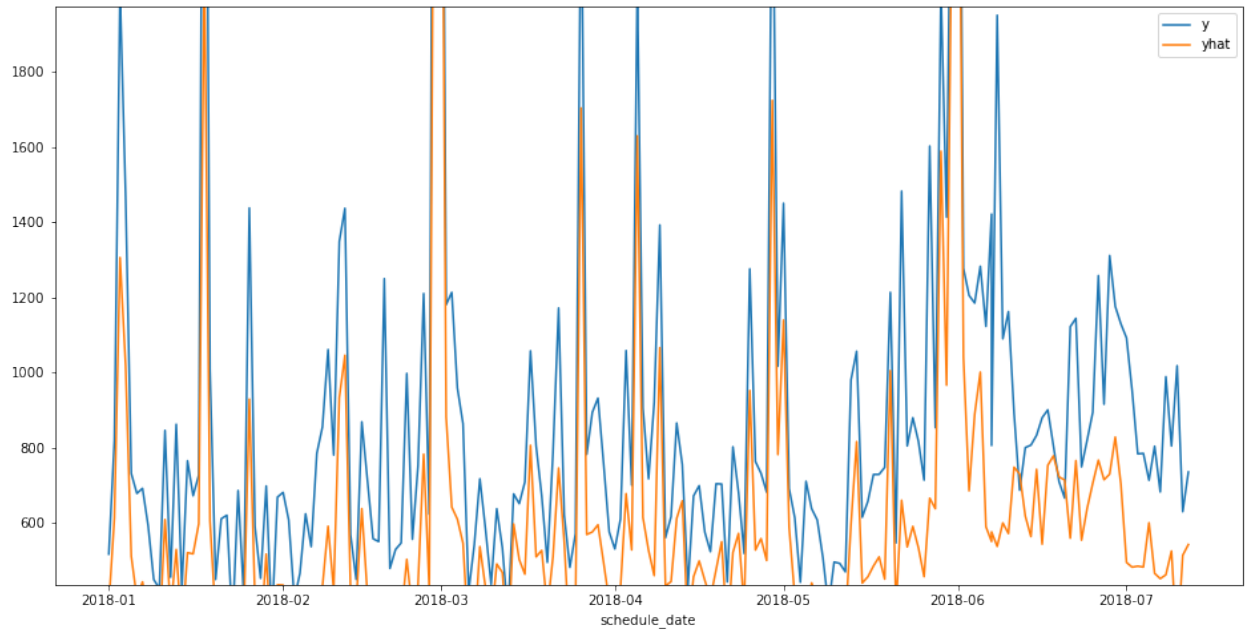
14.9.1 Plot some prediction results

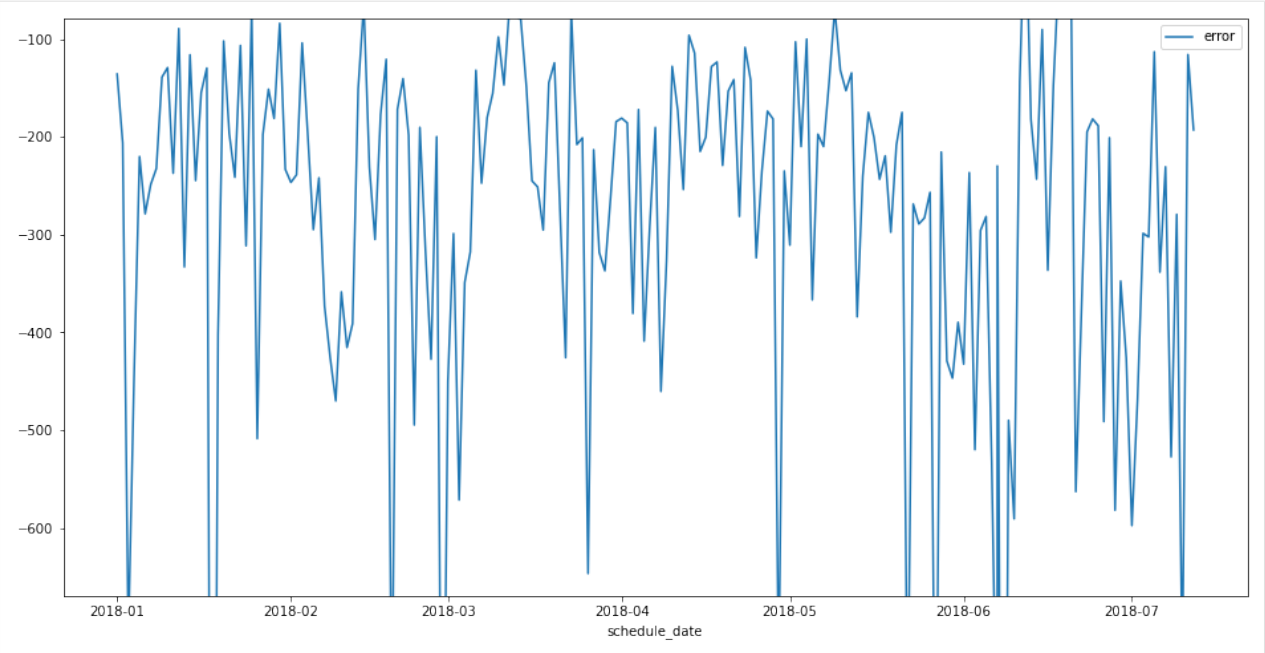
[17]:
```
def predictions_daily_mean(df_predictions):
    df_predictions["schedule_date"] = datetime_to_date(df_predictions[
↪"scheduleDateTime"])
    df_predictions = df_predictions.drop(columns="id")
    df_daily_mean = df_predictions.groupby(["model_set", "schedule_date"]).mean().
↪reset_index()
    return df_daily_mean

def predictions_hourly_mean(df_predictions):
    df_predictions["schedule_date"] = datetime_to_date_hour(df_predictions[
↪"scheduleDateTime"])
    df_predictions = df_predictions.drop(columns="id")
    df_daily_mean = df_predictions.groupby(["model_set", "schedule_date"]).mean().
↪reset_index()
    return df_daily_mean

def get_safe_ylim(y, q=0.05, q2=None):
    if q2 is None:
        q2 = 1 - q
    return (np.quantile(y, q), np.quantile(y, q2))


df_daily_mean = predictions_daily_mean(df_predictions)
y_ylim = get_safe_ylim(df_daily_mean["y"])
```

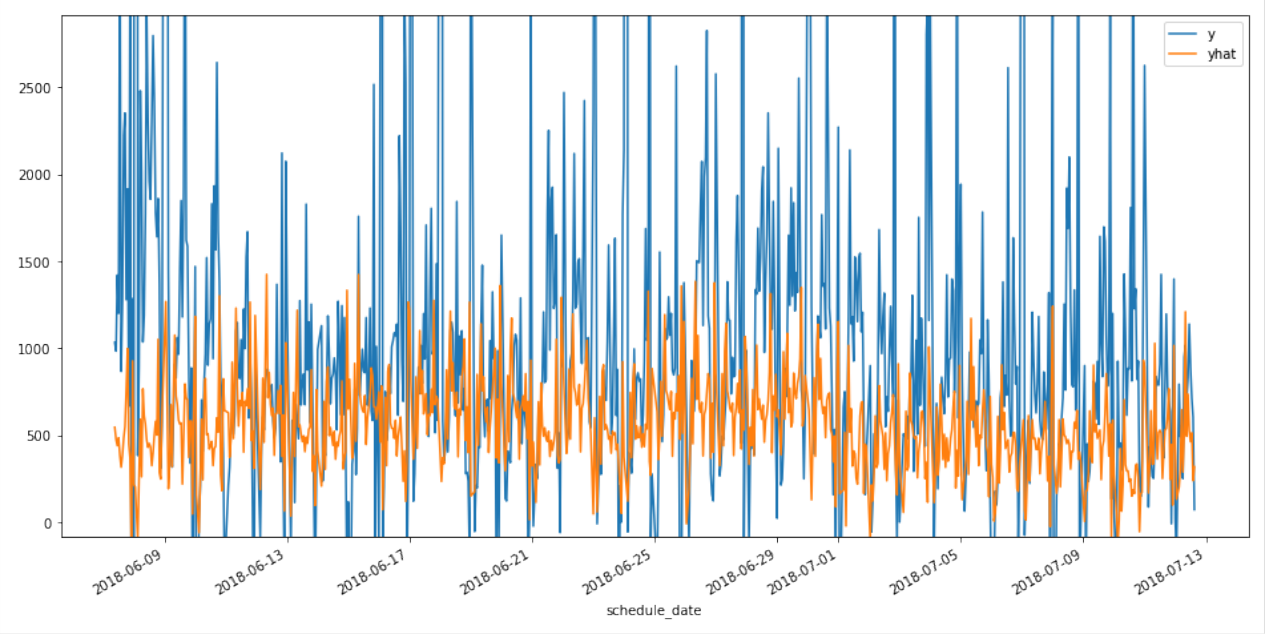
(continues on next page)

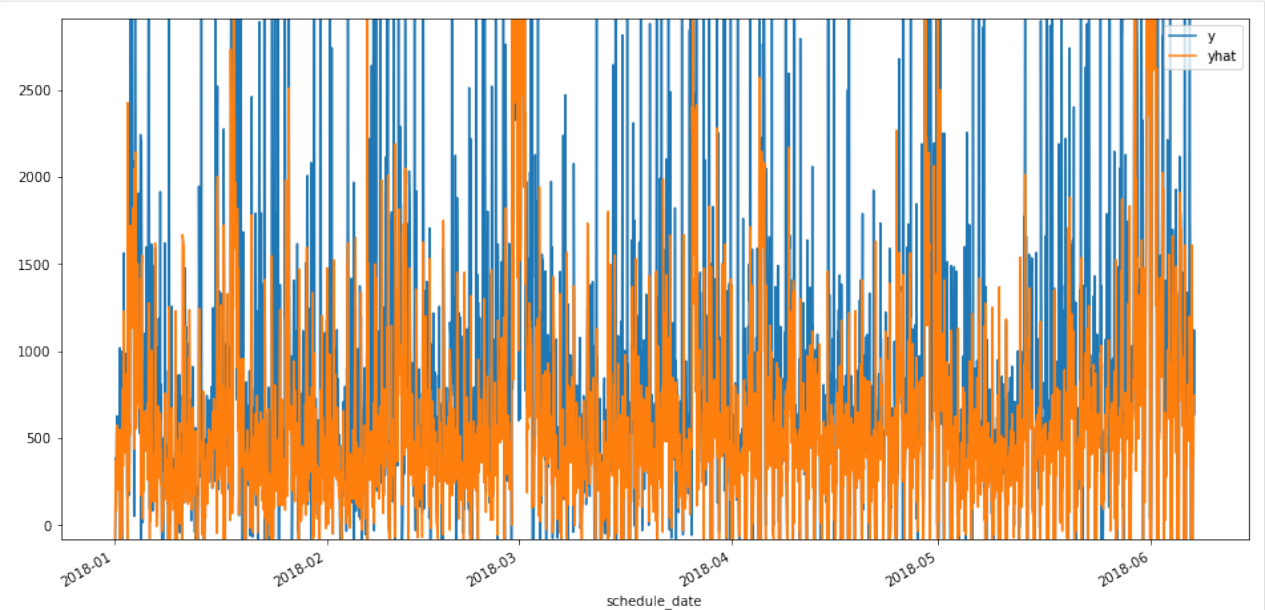
(continued from previous page)

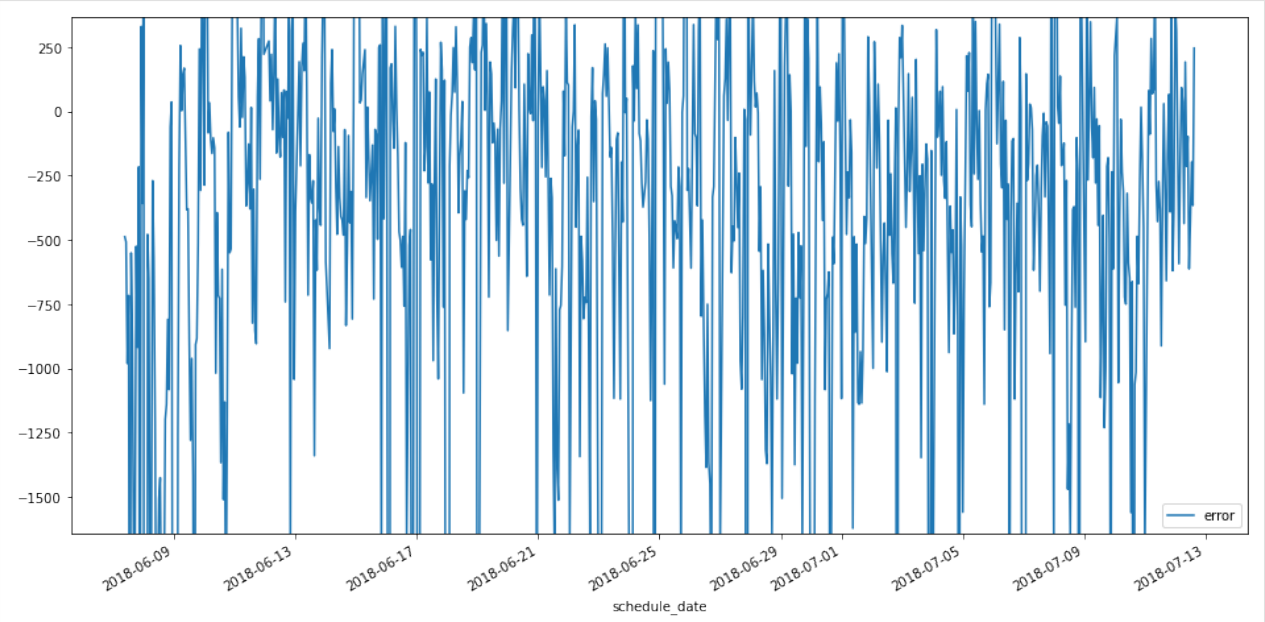
```
error_ylim = get_safe_ylim(df_daily_mean["error"])

df_daily_mean[["schedule_date", "y", "yhat", "model_set"]].plot(x="schedule_date",␣
↪ylim=y_ylim)
df_daily_mean[["schedule_date", "error", "model_set"]].plot(x="schedule_date",␣
↪ylim=error_ylim)

df_hourly_mean = predictions_hourly_mean(df_predictions)
y_ylim = get_safe_ylim(df_hourly_mean["y"])
error_ylim = get_safe_ylim(df_hourly_mean["error"])

df_hourly_mean[["schedule_date", "y", "yhat", "model_set"]].groupby("model_set").
↪plot(x="schedule_date", ylim=y_ylim)
df_hourly_mean[["schedule_date", "error", "model_set"]].groupby("model_set").plot(x=
↪"schedule_date", ylim=error_ylim)

df_hourly_mean[["schedule_date", "y", "yhat", "error", "model_set"]]
plt.show()

fig = px.line(df_hourly_mean, x="schedule_date", y="error", color="model_set",
              width=1200, height=1200, title="Hourly total prediction error")
fig.update_yaxes(matches=None)
fig.update_xaxes(matches=None)
fig.show()
```

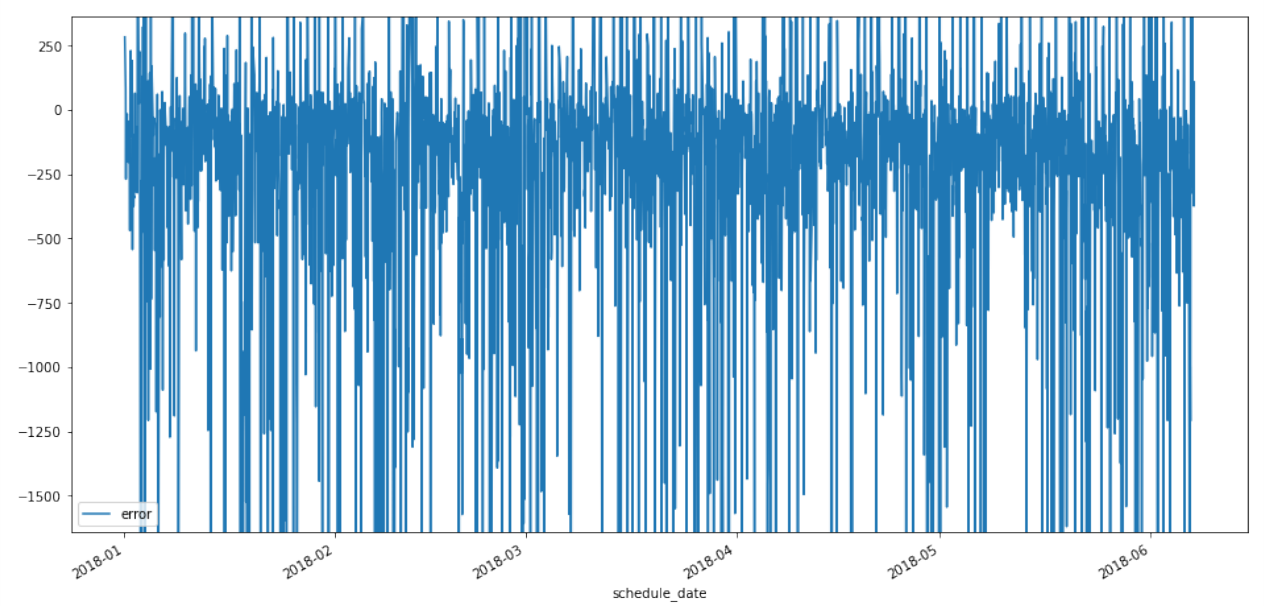


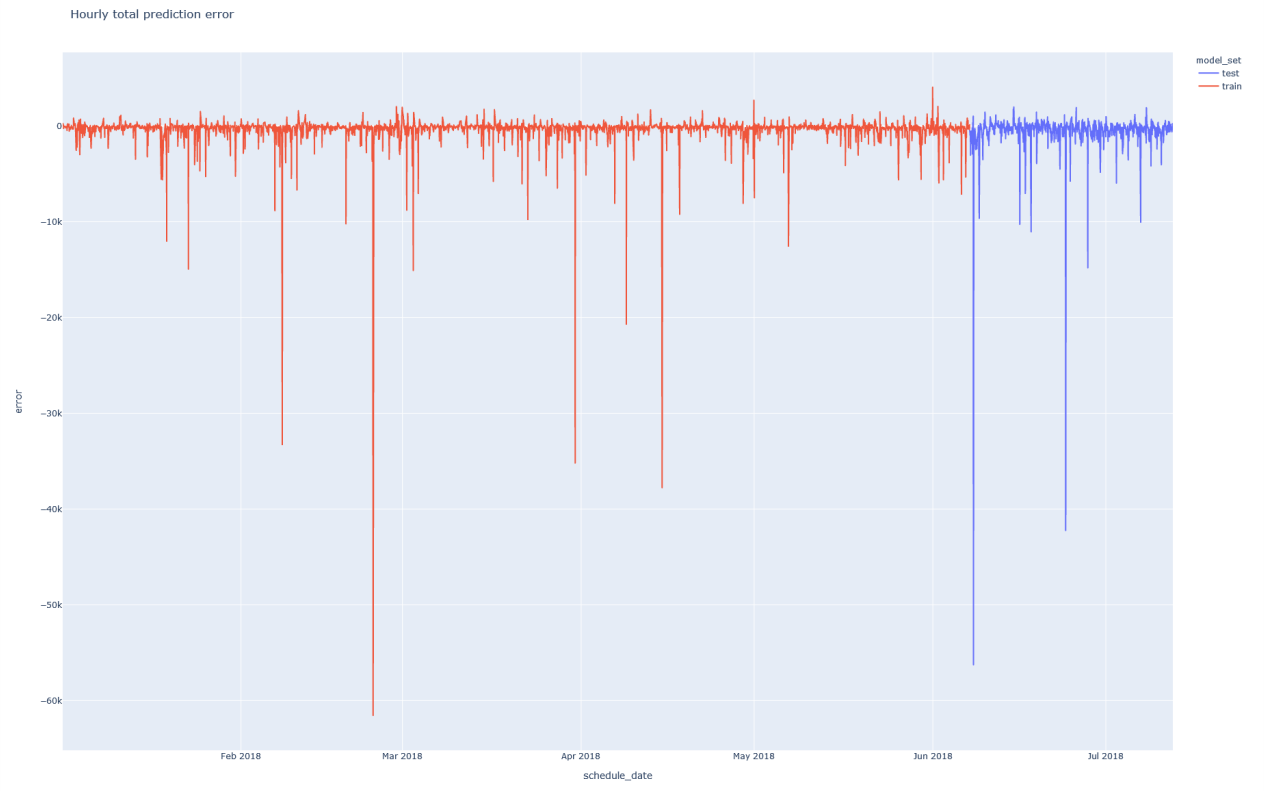

error
−100
−200
−300
−400
−500
−600
2018-01
2018-02
2018-03
2018-04
2018-05
2018-06
2018-07
schedule_date


y
yhat
2500
2000
1500
1000
500
0
2018-06-09
2018-06-13
2018-06-17
2018-06-21
2018-06-25
2018-06-29
2018-07-01
2018-07-05
2018-07-09
2018-07-13
schedule_date


y
yhat
2500
2000
1500
1000
500
0
2018-01
2018-02
2018-03
2018-04
2018-05
2018-06
schedule_date


250
0
−250
−500
−750
−1000
−1250
−1500
error
2018-06-09
2018-06-13
2018-06-17
2018-06-21
2018-06-25
2018-06-29
2018-07-01
2018-07-05
2018-07-09
2018-07-13
schedule_date





14.10 Write output to output directory

[18]:
```
import joblib, pickle
from pathlib import Path
```

```
[ ]: model_file = str(Path(output_dir, "model.pkl"))
predictions_file = str(Path(output_dir, "predictions.csv"))
overall_metrics_file = str(Path(output_dir, "overall_metrics_long.csv"))
daily_metrics_file = str(Path(output_dir, "daily_metrics_long.csv"))
hourly_metrics_file = str(Path(output_dir, "hourly_metrics_long.csv"))
```

14.10.1 Pickle output files for mlflow artifacts

- Pipeline serialized with `joblib`
- Model data or sample thereof

```
[19]: joblib.dump(model, Path(output_dir, model_file))
```

```
[19]: ['C:\\Users\\lodew\\qualogy\\schiphol-code-assignment\\scripts\\model.pkl']
```

14.10.2 Write output to CSV

Local or Google Storage is both handled

```
[20]: # write output file
write_csv_data(df_predictions,       predictions_file, index=False)
write_csv_data(df_metrics_long,       overall_metrics_file, index=False)
write_csv_data(df_daily_metrics_long, daily_metrics_file, index=False)
write_csv_data(df_hourly_metrics_long, hourly_metrics_file, index=False)
```

```
Writing file to local directory
File:   C:\Users\lodew\qualogy\schiphol-code-assignment\scripts\predictions.csv

Writing file to local directory
File:   C:\Users\lodew\qualogy\schiphol-code-assignment\scripts\overall_metrics_long.
↪csv

Writing file to local directory
File:   C:\Users\lodew\qualogy\schiphol-code-assignment\scripts\daily_metrics_long.csv

Writing file to local directory
File:   C:\Users\lodew\qualogy\schiphol-code-assignment\scripts\hourly_metrics_long.
↪csv
```

14.11 Log to MLFlow

```
[21]: import mlflow

mlflow.set_tracking_uri(mlflow_tracking_uri)
mlflow.set_experiment(mlflow_experiment)

print(f"Logging to experiment: {mlflow_experiment}")
print(f"Run name: {mlflow_run}")

with mlflow.start_run(run_name=mlflow_run):
```

(continues on next page)

(continued from previous page)

```
    mlflow.log_param("Input file", input_file)
    mlflow.log_param("Train-test file", train_test_file)

    # Model metadata
    for idx, metric_row in df_metrics_long.iterrows():
        metric_name = "__".join([metric_row["variable"], metric_row["model_set"]])
        mlflow.log_metric(metric_name, metric_row["value"])

    # log artifacts
    print("Logging artifacts")
    mlflow.log_artifact(predictions_file)
    mlflow.log_artifact(overall_metrics_file)
    mlflow.log_artifact(daily_metrics_file)
    mlflow.log_artifact(hourly_metrics_file)
```

```
INFO: 'from_script' does not exist. Creating a new experiment
Logging to experiment: from_script
Run name: catboost_simple
Logging artifacts
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-21-a6e1f7e0ce6b> in <module>
     18     # log artifacts
     19     print("Logging artifacts")
---> 20     mlflow.log_artifact(predictions_file)
     21     mlflow.log_artifact(overall_metrics_file)
     22     mlflow.log_artifact(daily_metrics_file)

NameError: name 'predictions_file' is not defined
```

14.12 Overview of the output data

[ ]:
```
df_predictions.info()
```

CHAPTER 15

Indices and tables

- genindex
- modindex
- search