Notebook-1: EDA & Preprocessing

by Qianjing Liang (8434350), Daniel Barbie (4939697), Jacob Umland (8436406), Fan Jia (8436217), Jan Faulstich (8439328)

In this notebook, we take a closer look at the data provided (EDA) and lay out several preprocessing steps for making the data more suitable for certain kinds of models.

Imports and Data Download

In [1]:

```
# imports
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import pickle
from sklearn.preprocessing import MinMaxScaler

from google.colab import drive
drive.mount('/content/gdrive')
%mkdir /content/gdrive/MyDrive/Electricity_Data
%cd /content/gdrive/MyDrive/Electricity_Data
```

```
Mounted at /content/gdrive
mkdir: cannot create directory ‘/content/gdrive/MyDrive/Electricity_Data’: File exis
ts
/content/gdrive/MyDrive/Electricity_Data
```

In [2]:

```
# Demonstration data used in this excercise is already preprocessed and split into
# training, validation, and test sets

# Use wget to download the data stored in csv format.
import itertools

# Define what files to download; download all of the preprocessed data
# Note that the data are already split into Train, Validation, and Test sets.
# The predictor data are denoted with 'X', the target by 'y'
data_download = {}
data_download["window_size"] = [5, 15]
data_download["data_type"] = ["train", "valid", "test"]
data_download["predictor_or_target"] = ["X", "y"]

# Prepare the combinations of the window sizes and the data types
keys, values = zip(*data_download.items())
data_download_combinations = [dict(zip(keys, v)) for v in itertools.product(*values)
display('The kind of data to be downloaded:', data_download_combinations)

print("Downloading started...")
for data_download_param in data_download_combinations:
  file_to_download = "https://frankfurt-school-dataset.s3.eu-central-1.amazonaws.com
                     .format(data_download_param["window_size"], data_download_param

  # the actual downloading
  !wget "$file_to_download"

print("Downloading has finished")
```

```
'The kind of data to be downloaded:'
```

```
[{'data_type': 'train', 'predictor_or_target': 'X', 'window_size': 5},
 {'data_type': 'train', 'predictor_or_target': 'y', 'window_size': 5},
 {'data_type': 'valid', 'predictor_or_target': 'X', 'window_size': 5},
 {'data_type': 'valid', 'predictor_or_target': 'y', 'window_size': 5},
 {'data_type': 'test', 'predictor_or_target': 'X', 'window_size': 5},
 {'data_type': 'test', 'predictor_or_target': 'y', 'window_size': 5},
 {'data_type': 'train', 'predictor_or_target': 'X', 'window_size': 15},
 {'data_type': 'train', 'predictor_or_target': 'y', 'window_size': 15},
 {'data_type': 'valid', 'predictor_or_target': 'X', 'window_size': 15},
 {'data_type': 'valid', 'predictor_or_target': 'y', 'window_size': 15},
 {'data_type': 'test', 'predictor_or_target': 'X', 'window_size': 15},
 {'data_type': 'test', 'predictor_or_target': 'y', 'window_size': 15}]
Downloading started...
--2021-11-09 20:03:58--  https://frankfurt-school-dataset.s3.eu-central-1.amazonaws.
com/Sept2021/window_size_5_time_encoding_True/X_train_window_size_5_time_encoding_Tr
ue.csv
Resolving frankfurt-school-dataset.s3.eu-central-1.amazonaws.com (frankfurt-school-d
ataset.s3.eu-central-1.amazonaws.com)... 52.219.47.92
Connecting to frankfurt-school-dataset.s3.eu-central-1.amazonaws.com (frankfurt-scho
ol-dataset.s3.eu-central-1.amazonaws.com)|52.219.47.92|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 53997719 (51M) [text/csv]
Saving to: ‘X_train_window_size_5_time_encoding_True.csv’

X_train_window_size 100%[===================>]  51.50M  18.2MB/s    in 2.8s

2021-11-09 20:04:02 (18.2 MB/s) - ‘X_train_window_size_5_time_encoding_True.csv’ sav
ed [53997719/53997719]

--2021-11-09 20:04:02--  https://frankfurt-school-dataset.s3.eu-central-1.amazonaws.
com/Sept2021/window_size_5_time_encoding_True/y_train_window_size_5_time_encoding_Tr
ue.csv
Resolving frankfurt-school-dataset.s3.eu-central-1.amazonaws.com (frankfurt-school-d
ataset.s3.eu-central-1.amazonaws.com)... 52.219.47.92
Connecting to frankfurt-school-dataset.s3.eu-central-1.amazonaws.com (frankfurt-scho
ol-dataset.s3.eu-central-1.amazonaws.com)|52.219.47.92|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1505402 (1.4M) [text/csv]
Saving to: ‘y_train_window_size_5_time_encoding_True.csv’

y_train_window_size 100%[===================>]   1.44M  2.10MB/s    in 0.7s

2021-11-09 20:04:03 (2.10 MB/s) - ‘y_train_window_size_5_time_encoding_True.csv’ sav
ed [1505402/1505402]

--2021-11-09 20:04:03--  https://frankfurt-school-dataset.s3.eu-central-1.amazonaws.
com/Sept2021/window_size_5_time_encoding_True/X_valid_window_size_5_time_encoding_Tr
ue.csv
Resolving frankfurt-school-dataset.s3.eu-central-1.amazonaws.com (frankfurt-school-d
ataset.s3.eu-central-1.amazonaws.com)... 52.219.72.213
Connecting to frankfurt-school-dataset.s3.eu-central-1.amazonaws.com (frankfurt-scho
ol-dataset.s3.eu-central-1.amazonaws.com)|52.219.72.213|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 2931637 (2.8M) [text/csv]
Saving to: ‘X_valid_window_size_5_time_encoding_True.csv’

X_valid_window_size 100%[===================>]   2.79M  3.97MB/s    in 0.7s

2021-11-09 20:04:04 (3.97 MB/s) - ‘X_valid_window_size_5_time_encoding_True.csv’ sav
ed [2931637/2931637]

--2021-11-09 20:04:04--  https://frankfurt-school-dataset.s3.eu-central-1.amazonaws.
com/Sept2021/window_size_5_time_encoding_True/y_valid_window_size_5_time_encoding_Tr
ue.csv
Resolving frankfurt-school-dataset.s3.eu-central-1.amazonaws.com (frankfurt-school-d
```

```
ataset.s3.eu-central-1.amazonaws.com)... 52.219.72.213
Connecting to frankfurt-school-dataset.s3.eu-central-1.amazonaws.com (frankfurt-scho
ol-dataset.s3.eu-central-1.amazonaws.com)|52.219.72.213|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 82567 (81K) [text/csv]
Saving to: ‘y_valid_window_size_5_time_encoding_True.csv’

y_valid_window_size 100%[===================>]  80.63K   416KB/s    in 0.2s

2021-11-09 20:04:05 (416 KB/s) - ‘y_valid_window_size_5_time_encoding_True.csv’ save
d [82567/82567]

--2021-11-09 20:04:05--  https://frankfurt-school-dataset.s3.eu-central-1.amazonaws.
com/Sept2021/window_size_5_time_encoding_True/X_test_window_size_5_time_encoding_Tru
e.csv
Resolving frankfurt-school-dataset.s3.eu-central-1.amazonaws.com (frankfurt-school-d
ataset.s3.eu-central-1.amazonaws.com)... 52.219.72.213
Connecting to frankfurt-school-dataset.s3.eu-central-1.amazonaws.com (frankfurt-scho
ol-dataset.s3.eu-central-1.amazonaws.com)|52.219.72.213|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 6479846 (6.2M) [text/csv]
Saving to: ‘X_test_window_size_5_time_encoding_True.csv’

X_test_window_size_ 100%[===================>]   6.18M  6.33MB/s    in 1.0s

2021-11-09 20:04:07 (6.33 MB/s) - ‘X_test_window_size_5_time_encoding_True.csv’ save
d [6479846/6479846]

--2021-11-09 20:04:07--  https://frankfurt-school-dataset.s3.eu-central-1.amazonaws.
com/Sept2021/window_size_5_time_encoding_True/y_test_window_size_5_time_encoding_Tru
e.csv
Resolving frankfurt-school-dataset.s3.eu-central-1.amazonaws.com (frankfurt-school-d
ataset.s3.eu-central-1.amazonaws.com)... 52.219.72.213
Connecting to frankfurt-school-dataset.s3.eu-central-1.amazonaws.com (frankfurt-scho
ol-dataset.s3.eu-central-1.amazonaws.com)|52.219.72.213|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 181827 (178K) [text/csv]
Saving to: ‘y_test_window_size_5_time_encoding_True.csv’

y_test_window_size_ 100%[===================>] 177.57K   452KB/s    in 0.4s

2021-11-09 20:04:08 (452 KB/s) - ‘y_test_window_size_5_time_encoding_True.csv’ saved
[181827/181827]

--2021-11-09 20:04:08--  https://frankfurt-school-dataset.s3.eu-central-1.amazonaws.
com/Sept2021/window_size_15_time_encoding_True/X_train_window_size_15_time_encoding_
True.csv
Resolving frankfurt-school-dataset.s3.eu-central-1.amazonaws.com (frankfurt-school-d
ataset.s3.eu-central-1.amazonaws.com)... 52.219.171.98
Connecting to frankfurt-school-dataset.s3.eu-central-1.amazonaws.com (frankfurt-scho
ol-dataset.s3.eu-central-1.amazonaws.com)|52.219.171.98|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 41978632 (40M) [text/csv]
Saving to: ‘X_train_window_size_15_time_encoding_True.csv’

X_train_window_size 100%[===================>]  40.03M  16.7MB/s    in 2.4s

2021-11-09 20:04:11 (16.7 MB/s) - ‘X_train_window_size_15_time_encoding_True.csv’ sa
ved [41978632/41978632]

--2021-11-09 20:04:11--  https://frankfurt-school-dataset.s3.eu-central-1.amazonaws.
com/Sept2021/window_size_15_time_encoding_True/y_train_window_size_15_time_encoding_
True.csv
Resolving frankfurt-school-dataset.s3.eu-central-1.amazonaws.com (frankfurt-school-d
ataset.s3.eu-central-1.amazonaws.com)... 52.219.171.98
```

```
Connecting to frankfurt-school-dataset.s3.eu-central-1.amazonaws.com (frankfurt-scho
ol-dataset.s3.eu-central-1.amazonaws.com)|52.219.171.98|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 531355 (519K) [text/csv]
Saving to: ‘y_train_window_size_15_time_encoding_True.csv’

y_train_window_size 100%[===================>] 518.90K  1.04MB/s    in 0.5s

2021-11-09 20:04:12 (1.04 MB/s) - ‘y_train_window_size_15_time_encoding_True.csv’ sa
ved [531355/531355]

--2021-11-09 20:04:12--  https://frankfurt-school-dataset.s3.eu-central-1.amazonaws.
com/Sept2021/window_size_15_time_encoding_True/X_valid_window_size_15_time_encoding_
True.csv
Resolving frankfurt-school-dataset.s3.eu-central-1.amazonaws.com (frankfurt-school-d
ataset.s3.eu-central-1.amazonaws.com)... 52.219.171.98
Connecting to frankfurt-school-dataset.s3.eu-central-1.amazonaws.com (frankfurt-scho
ol-dataset.s3.eu-central-1.amazonaws.com)|52.219.171.98|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 2262426 (2.2M) [text/csv]
Saving to: ‘X_valid_window_size_15_time_encoding_True.csv’

X_valid_window_size 100%[===================>]   2.16M  3.08MB/s    in 0.7s

2021-11-09 20:04:13 (3.08 MB/s) - ‘X_valid_window_size_15_time_encoding_True.csv’ sa
ved [2262426/2262426]

--2021-11-09 20:04:13--  https://frankfurt-school-dataset.s3.eu-central-1.amazonaws.
com/Sept2021/window_size_15_time_encoding_True/y_valid_window_size_15_time_encoding_
True.csv
Resolving frankfurt-school-dataset.s3.eu-central-1.amazonaws.com (frankfurt-school-d
ataset.s3.eu-central-1.amazonaws.com)... 52.219.75.96
Connecting to frankfurt-school-dataset.s3.eu-central-1.amazonaws.com (frankfurt-scho
ol-dataset.s3.eu-central-1.amazonaws.com)|52.219.75.96|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 27675 (27K) [text/csv]
Saving to: ‘y_valid_window_size_15_time_encoding_True.csv’

y_valid_window_size 100%[===================>]  27.03K  --.-KB/s    in 0.1s

2021-11-09 20:04:13 (278 KB/s) - ‘y_valid_window_size_15_time_encoding_True.csv’ sav
ed [27675/27675]

--2021-11-09 20:04:14--  https://frankfurt-school-dataset.s3.eu-central-1.amazonaws.
com/Sept2021/window_size_15_time_encoding_True/X_test_window_size_15_time_encoding_T
rue.csv
Resolving frankfurt-school-dataset.s3.eu-central-1.amazonaws.com (frankfurt-school-d
ataset.s3.eu-central-1.amazonaws.com)... 52.219.75.96
Connecting to frankfurt-school-dataset.s3.eu-central-1.amazonaws.com (frankfurt-scho
ol-dataset.s3.eu-central-1.amazonaws.com)|52.219.75.96|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 5152033 (4.9M) [text/csv]
Saving to: ‘X_test_window_size_15_time_encoding_True.csv’

X_test_window_size_ 100%[===================>]   4.91M  5.58MB/s    in 0.9s

2021-11-09 20:04:15 (5.58 MB/s) - ‘X_test_window_size_15_time_encoding_True.csv’ sav
ed [5152033/5152033]

--2021-11-09 20:04:15--  https://frankfurt-school-dataset.s3.eu-central-1.amazonaws.
com/Sept2021/window_size_15_time_encoding_True/y_test_window_size_15_time_encoding_T
rue.csv
Resolving frankfurt-school-dataset.s3.eu-central-1.amazonaws.com (frankfurt-school-d
ataset.s3.eu-central-1.amazonaws.com)... 52.219.75.96
Connecting to frankfurt-school-dataset.s3.eu-central-1.amazonaws.com (frankfurt-scho
```

```
ol-dataset.s3.eu-central-1.amazonaws.com)|52.219.75.96|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 61541 (60K) [text/csv]
Saving to: 'y_test_window_size_15_time_encoding_True.csv'

y_test_window_size_ 100%[===================>]  60.10K   313KB/s    in 0.2s

2021-11-09 20:04:16 (313 KB/s) - 'y_test_window_size_15_time_encoding_True.csv' save
d [61541/61541]

Downloading has finished
```

In [3]:

```
# Load data for both window size 5
X_train_window_size_5 = pd.read_csv('X_train_window_size_5_time_encoding_True.csv')
X_valid_window_size_5 = pd.read_csv('X_valid_window_size_5_time_encoding_True.csv')
X_test_window_size_5 = pd.read_csv('X_test_window_size_5_time_encoding_True.csv')
y_train_window_size_5 = pd.read_csv('y_train_window_size_5_time_encoding_True.csv')
y_valid_window_size_5 = pd.read_csv('y_valid_window_size_5_time_encoding_True.csv')
y_test_window_size_5 = pd.read_csv('y_test_window_size_5_time_encoding_True.csv')

# Load data for both window size 15
X_train_window_size_15 = pd.read_csv('X_train_window_size_15_time_encoding_True.csv'
X_valid_window_size_15 = pd.read_csv('X_valid_window_size_15_time_encoding_True.csv'
X_test_window_size_15 = pd.read_csv('X_test_window_size_15_time_encoding_True.csv')
y_train_window_size_15 = pd.read_csv('y_train_window_size_15_time_encoding_True.csv'
y_valid_window_size_15 = pd.read_csv('y_valid_window_size_15_time_encoding_True.csv'
y_test_window_size_15 = pd.read_csv('y_test_window_size_15_time_encoding_True.csv')
```

EDA

What are we dealing with?

The data at hand is about electricity contracts trades.

There are two different sets of data, looking at different numbers of timesteps (trading hours) in the past, either 5 or 15 steps (i.e., window size) for a lot of different contracts.

At this point, this is all you need to know.

Using the following steps, we will go through the actual features and the datasets' structure in order to explore the data and get familiar with it.

- a) Data structure
- b) Candle Plots
- c) Correlation matrix

a) Data structure

When we look at the columns (see below), we can identify several features:

- **total_hours**: hours from the first trade we know for a given contract until delivery start of the same contract
- **dlvry_weekend**: whether the electricity is to be delivered on a weekend (binary)
- **dlvry_bank_holiday**: whether the electricity is to be delivered on a bank holiday (binary)
- **dlvry_day_sin** & **dlvry_day_cos**: sine/cosine-transformed day of delivery
- **dlvry_weekday_sin** & **dlvry_weekday_cos**: sine/cosine-transformed weekday of delivery
- **dlvry_hour_sin** & **dlvry_hour_cos**: sine/cosine-transformed hour of delivery

- **lasttrade_weekend**: whether most recent trading hour was on a weekend (binary)
- **lasttrade_bank_holiday**: whether most recent trading hour was on a holiday (binary)
- **lasttrade_day_sin** & **lasttrade_day_cos**: sine/cosine-transformed day of most recent trading hour
- **lasttrade_weekday_sin** & **lasttrade_weekday_cos**: sine/cosine-transformed weekday of most recent trading hour
- **lasttrade_hour_sin** & **lasttrade_hour_cos**: sine/cosine-transformed hour of most recent trading hour

All the aforementioned features, are the same for the both window sizes (5 and 15).

In [4]:

```
X_train_window_size_5.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 91512 entries, 0 to 91511
Data columns (total 41 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   total_hours            91512 non-null  float64
 1   dlvry_weekend          91512 non-null  float64
 2   dlvry_bank_holiday     91512 non-null  float64
 3   dlvry_day_sin          91512 non-null  float64
 4   dlvry_day_cos          91512 non-null  float64
 5   dlvry_weekday_sin      91512 non-null  float64
 6   dlvry_weekday_cos      91512 non-null  float64
 7   dlvry_hour_sin         91512 non-null  float64
 8   dlvry_hour_cos         91512 non-null  float64
 9   lasttrade_weekend      91512 non-null  float64
 10  lasttrade_bank_holiday 91512 non-null  float64
 11  lasttrade_day_sin      91512 non-null  float64
 12  lasttrade_day_cos      91512 non-null  float64
 13  lasttrade_weekday_sin  91512 non-null  float64
 14  lasttrade_weekday_cos  91512 non-null  float64
 15  lasttrade_hour_sin     91512 non-null  float64
 16  lasttrade_hour_cos     91512 non-null  float64
 17  0                      91512 non-null  float64
 18  1                      91512 non-null  float64
 19  2                      91512 non-null  float64
 20  3                      91512 non-null  float64
 21  4                      91512 non-null  float64
 22  5                      91512 non-null  float64
 23  6                      91512 non-null  float64
 24  7                      91512 non-null  float64
 25  8                      91512 non-null  float64
 26  9                      91512 non-null  float64
 27  10                     91512 non-null  float64
 28  11                     91512 non-null  float64
 29  12                     91512 non-null  float64
 30  13                     91512 non-null  float64
 31  14                     91512 non-null  float64
 32  15                     91512 non-null  float64
 33  16                     91512 non-null  float64
 34  17                     91512 non-null  float64
 35  18                     91512 non-null  float64
 36  19                     91512 non-null  float64
 37  20                     91512 non-null  float64
 38  21                     91512 non-null  float64
 39  22                     91512 non-null  float64
 40  23                     91512 non-null  float64
dtypes: float64(41)
memory usage: 28.6 MB
```

Okay, those features are quite intuitive, but what about all those integer-named columns?

They represent 6 additional features for each time step within for the specified window size. As they are a series of values obtained at successive times, these features can be seen as time series. More specifically, the values represent the differences in values between time steps, which is why we have `window size - 1` values for each time series feature. This is why the data set with window size 5 has 24 time series columns (`6 features * 4 time steps`) and the window size 15 data set has 84 (`6 features * 14 time steps`).

Using the overview below, we grasp the mapping of integer-named columns and their meaning more intuitively.

Open (1st diff)		Col - 0		Col - 6		Col - 12		Col - 18	
High (1st diff)		Col - 1		Col - 7		Col - 13		Col - 19	
Low (1st diff)		Col - 2		Col - 8		Col - 14		Col - 20	
Close (1st diff)		Col - 3		Col - 9		Col - 15		Col - 21	
Volume (1st diff)		Col - 4		Col - 10		Col - 16		Col - 22	
Min left until dlvry starts		Col - 5		Col - 11		Col - 17		Col - 23	
Time Line	**T-4**		**T-3**		**T-2**		**T-1**		**T**
	for window_size = 5								

Open (1st diff)		Col - 0		Col - 6		Col - 12		Col - 18		Col - 24		Col - 30		Col - 36		Col - 42		Col - 48		Col - 54		Col - 60		Col - 66		Col - 72		Col - 78	
High (1st diff)		Col - 1		Col - 7		Col - 13		Col - 19		Col - 25		Col - 31		Col - 37		Col - 43		Col - 49		Col - 55		Col - 61		Col - 67		Col - 73		Col - 79	
Low (1st diff)		Col - 2		Col - 8		Col - 14		Col - 20		Col - 26		Col - 32		Col - 38		Col - 44		Col - 50		Col - 56		Col - 62		Col - 68		Col - 74		Col - 80	
Close (1st diff)		Col - 3		Col - 9		Col - 15		Col - 21		Col - 27		Col - 33		Col - 39		Col - 45		Col - 51		Col - 57		Col - 63		Col - 69		Col - 75		Col - 81	
Volume (1st diff)		Col - 4		Col - 10		Col - 16		Col - 22		Col - 28		Col - 34		Col - 40		Col - 46		Col - 52		Col - 58		Col - 64		Col - 70		Col - 76		Col - 82	
Min left until dlvry starts		Col - 5		Col - 11		Col - 17		Col - 23		Col - 29		Col - 35		Col - 41		Col - 47		Col - 53		Col - 59		Col - 65		Col - 71		Col - 77		Col - 83	
Time Line	**T-14**		**T-13**		**T-12**		**T-11**		**T-10**		**T-9**		**T-8**		**T-7**		**T-6**		**T-5**		**T-4**		**T-3**		**T-2**		**T-1**		**T**
	for window_size = 15																												

So, let's give meaningful names to the columns with integer column names, in line with the structure provided above.

In [5]:

```
# create function for changing the integer-based column names for window size 5
def rename_time_series_5(df):
    labels = ["open", "high", "low", "close", "volume", "minutes"]
    new_cols = list(df.columns[:17])
    for i in range(4, 0, -1):
        for label in labels:
            new_cols.append(label+f"_{i}_{i-1}")

    df_original = df.copy()
    df.columns = new_cols

    return df
```

In [6]:

```
# create function for changing the integer-based column names for window size 15
def rename_time_series_15(df):
    labels = ["open", "high", "low", "close", "volume", "minutes"]
    new_cols = list(df.columns[:17])
    for i in range(14, 0, -1):
        for label in labels:
            new_cols.append(label+f"_{i}_{i-1}")

    df_original = df.copy()
    df.columns = new_cols

    return df
```

In [7]:

```
# apply renaming functions to all X datasets
X_train_window_size_5 = rename_time_series_5(X_train_window_size_5)
```

```
X_valid_window_size_5 = rename_time_series_5(X_valid_window_size_5)
X_test_window_size_5 = rename_time_series_5(X_test_window_size_5)

# Load data for both window size 15
X_train_window_size_15 = rename_time_series_15(X_train_window_size_15)
X_valid_window_size_15 = rename_time_series_15(X_valid_window_size_15)
X_test_window_size_15 = rename_time_series_15(X_test_window_size_15)
```

In [8]:

```
X_train_window_size_5.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 91512 entries, 0 to 91511
Data columns (total 41 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   total_hours            91512 non-null  float64
 1   dlvry_weekend          91512 non-null  float64
 2   dlvry_bank_holiday     91512 non-null  float64
 3   dlvry_day_sin          91512 non-null  float64
 4   dlvry_day_cos          91512 non-null  float64
 5   dlvry_weekday_sin      91512 non-null  float64
 6   dlvry_weekday_cos      91512 non-null  float64
 7   dlvry_hour_sin         91512 non-null  float64
 8   dlvry_hour_cos         91512 non-null  float64
 9   lasttrade_weekend      91512 non-null  float64
 10  lasttrade_bank_holiday 91512 non-null  float64
 11  lasttrade_day_sin      91512 non-null  float64
 12  lasttrade_day_cos      91512 non-null  float64
 13  lasttrade_weekday_sin  91512 non-null  float64
 14  lasttrade_weekday_cos  91512 non-null  float64
 15  lasttrade_hour_sin     91512 non-null  float64
 16  lasttrade_hour_cos     91512 non-null  float64
 17  open_4_3               91512 non-null  float64
 18  high_4_3               91512 non-null  float64
 19  low_4_3                91512 non-null  float64
 20  close_4_3              91512 non-null  float64
 21  volume_4_3             91512 non-null  float64
 22  minutes_4_3            91512 non-null  float64
 23  open_3_2               91512 non-null  float64
 24  high_3_2               91512 non-null  float64
 25  low_3_2                91512 non-null  float64
 26  close_3_2              91512 non-null  float64
 27  volume_3_2             91512 non-null  float64
 28  minutes_3_2            91512 non-null  float64
 29  open_2_1               91512 non-null  float64
 30  high_2_1               91512 non-null  float64
 31  low_2_1                91512 non-null  float64
 32  close_2_1              91512 non-null  float64
 33  volume_2_1             91512 non-null  float64
 34  minutes_2_1            91512 non-null  float64
 35  open_1_0               91512 non-null  float64
 36  high_1_0               91512 non-null  float64
 37  low_1_0                91512 non-null  float64
 38  close_1_0              91512 non-null  float64
 39  volume_1_0             91512 non-null  float64
 40  minutes_1_0            91512 non-null  float64
dtypes: float64(41)
memory usage: 28.6 MB
```

For example, open_3_2 is the difference from the open price in t-3 compared to the open price in t-2.

Now that we got an understaning of the columns, let's focus on the rows.

The underlying data-generating entities are different contracts. Multiple rows could, in theory, stem from the same contract, as its trading history can be split into multiple sliding windows.

We can use the total_hours as a proxy primary key for identifying the underlying contract for each row of the data.

In [9]:

```
# group by total_hours features
X_train_window_size_5_grouped = X_train_window_size_5.groupby("total_hours").size().
X_train_window_size_5_grouped.rename(columns={0: "rowcount"}, inplace=True)

print(f"The minimum amount of rows attributed to a specific contract for window size
print(f"The maximum amount of rows attributed to a specific contract for window size
print(f"The mean amount of rows attributed to a specific contract for window size 5
```

```
The minimum amount of rows attributed to a specific contract for window size 5 is: 1
The maximum amount of rows attributed to a specific contract for window size 5 is: 4
4
The mean amount of rows attributed to a specific contract for window size 5 is: 12
```

In [10]:

```
# group by total_hours features
X_train_window_size_15_grouped = X_train_window_size_15.groupby("total_hours").size(
X_train_window_size_15_grouped.rename(columns={0: "rowcount"}, inplace=True)

print(f"The minimum amount of rows attributed to a specific contract for window size
print(f"The maximum amount of rows attributed to a specific contract for window size
print(f"The mean amount of rows attributed to a specific contract for window size 15
```

```
The minimum amount of rows attributed to a specific contract for window size 15 is:
1
The maximum amount of rows attributed to a specific contract for window size 15 is:
24
The mean amount of rows attributed to a specific contract for window size 15 is: 7
```

This also highlights the difficulty that the data is not a continuous time series, but rather is composed of many time series that can occur in parallel.

b) Candles plots

Another way of looking at the data and spur an understanign is to use the visualization as candles.

In [11]:

```
# create helper function to plot candles
def plot_subplots(dataframe: pd.DataFrame, title: str):
    """
    Draws one subplot for each of the columns in the DataFrame.
    """

    fig, axes = plt.subplots(nrows=len(dataframe.columns), figsize=(20,20))
    labels = list(dataframe.columns)

    for idx, row in enumerate(axes):
        y = dataframe.iloc[:, idx]
        row.plot(y, label=y.name)
        row.set_ylabel(labels[idx])

    fig.set_facecolor("grey")
    fig.suptitle(title, size=15, y=0.9)
    plt.show()
```

In [12]:

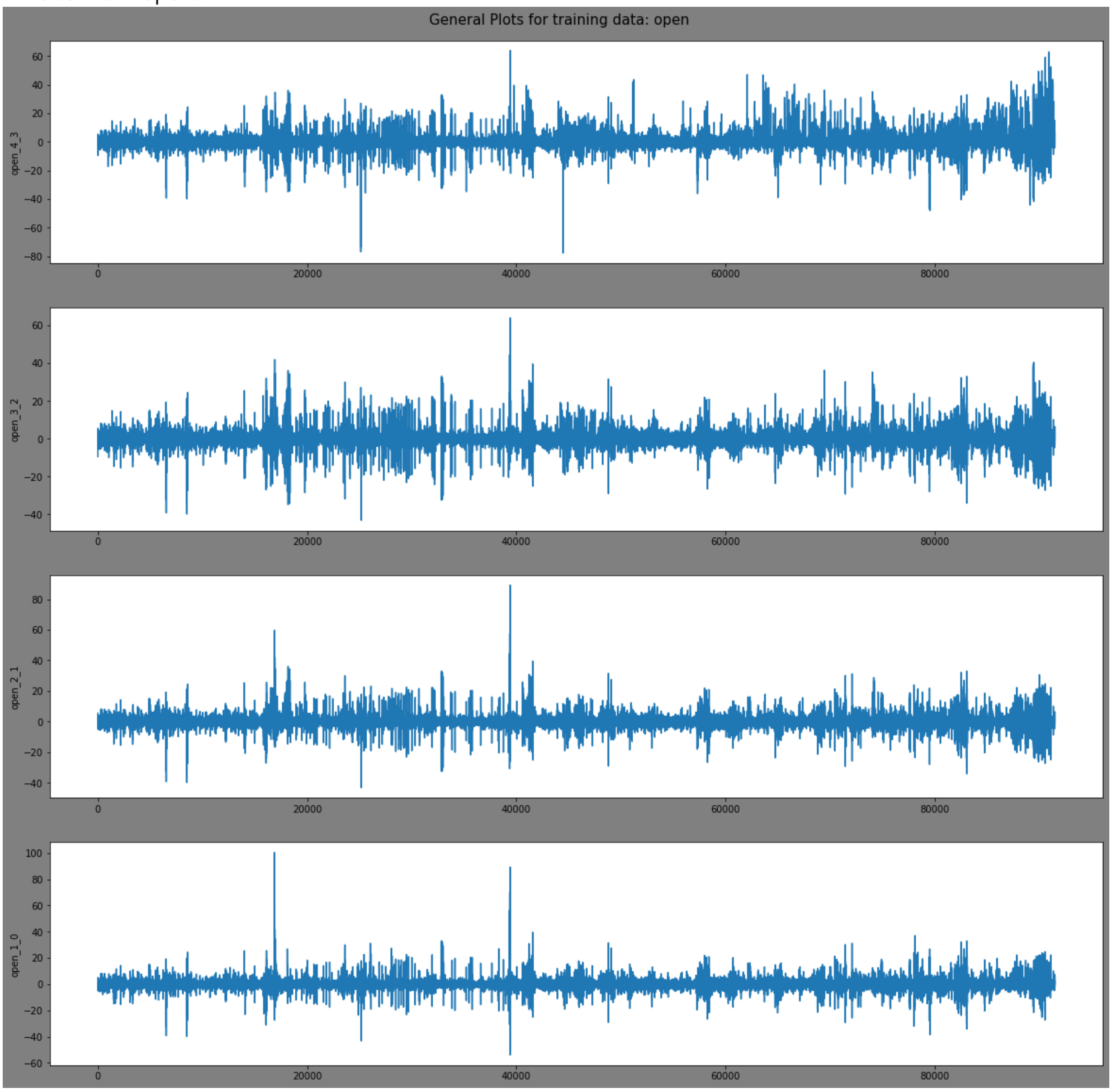
```
# create helper function to plot candles
def plot_subplots_wrapper(
    dataframe: pd.DataFrame,
    title: str,
    labels: list = ["open", "high", "low", "close", "volume"]
    ):
    """
    Calls the 'plot_subplots' function for each label provided in labels and selects
    """

    #df = dataframe.copy()
    for label in labels:
        print(f"Plots for {label}:")
        title = title + f" {label}"
        plot_subplots(dataframe.loc[:,dataframe.columns.str.startswith(label)], titl
```

Let's look at the differenced feature open, high, low, close and volume in the training data for window size 5.

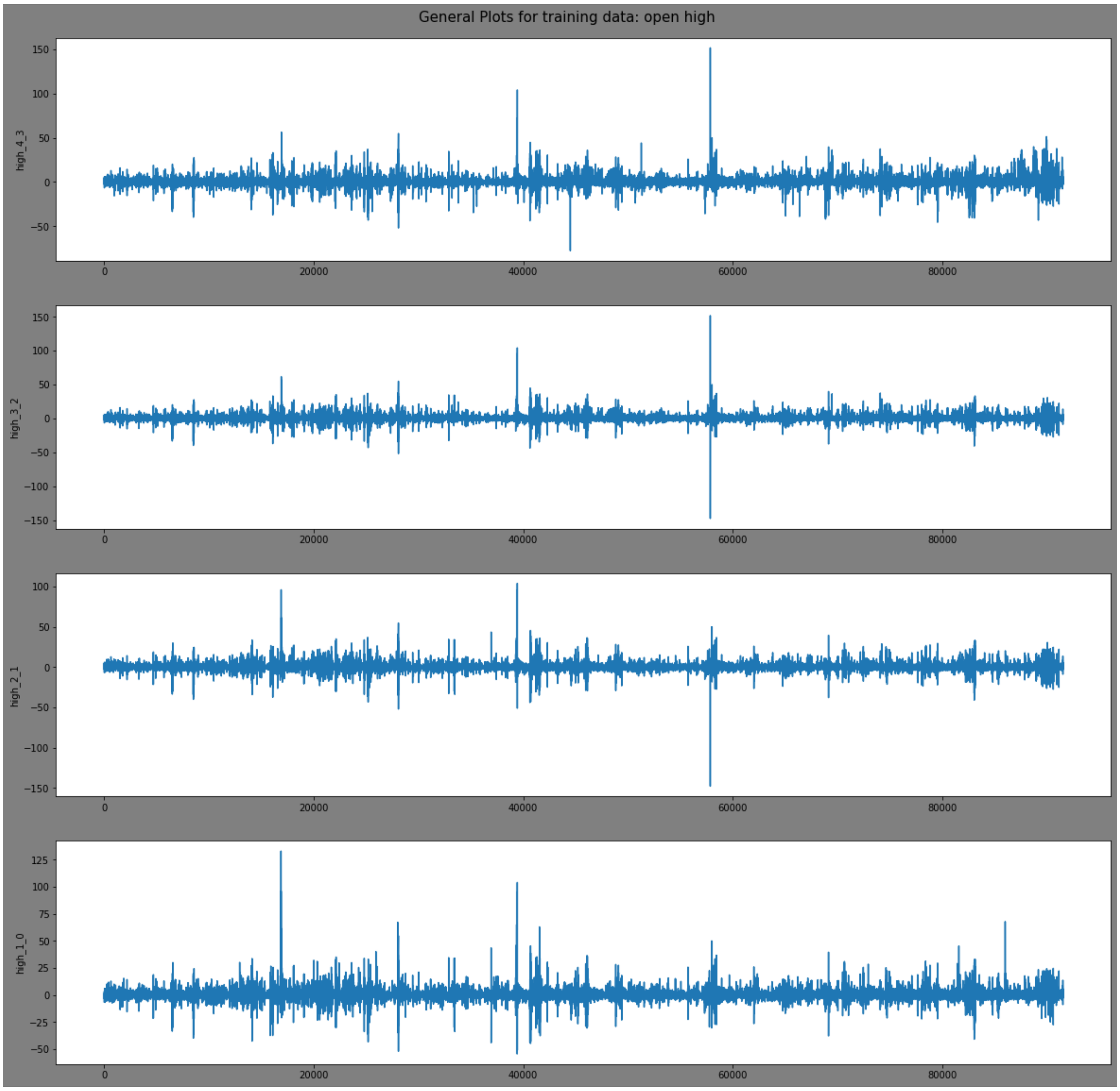
In [13]:

```
df_to_plot = X_train_window_size_5.copy()
title = "General Plots for training data:"
plot_subplots_wrapper(df_to_plot, title)
```

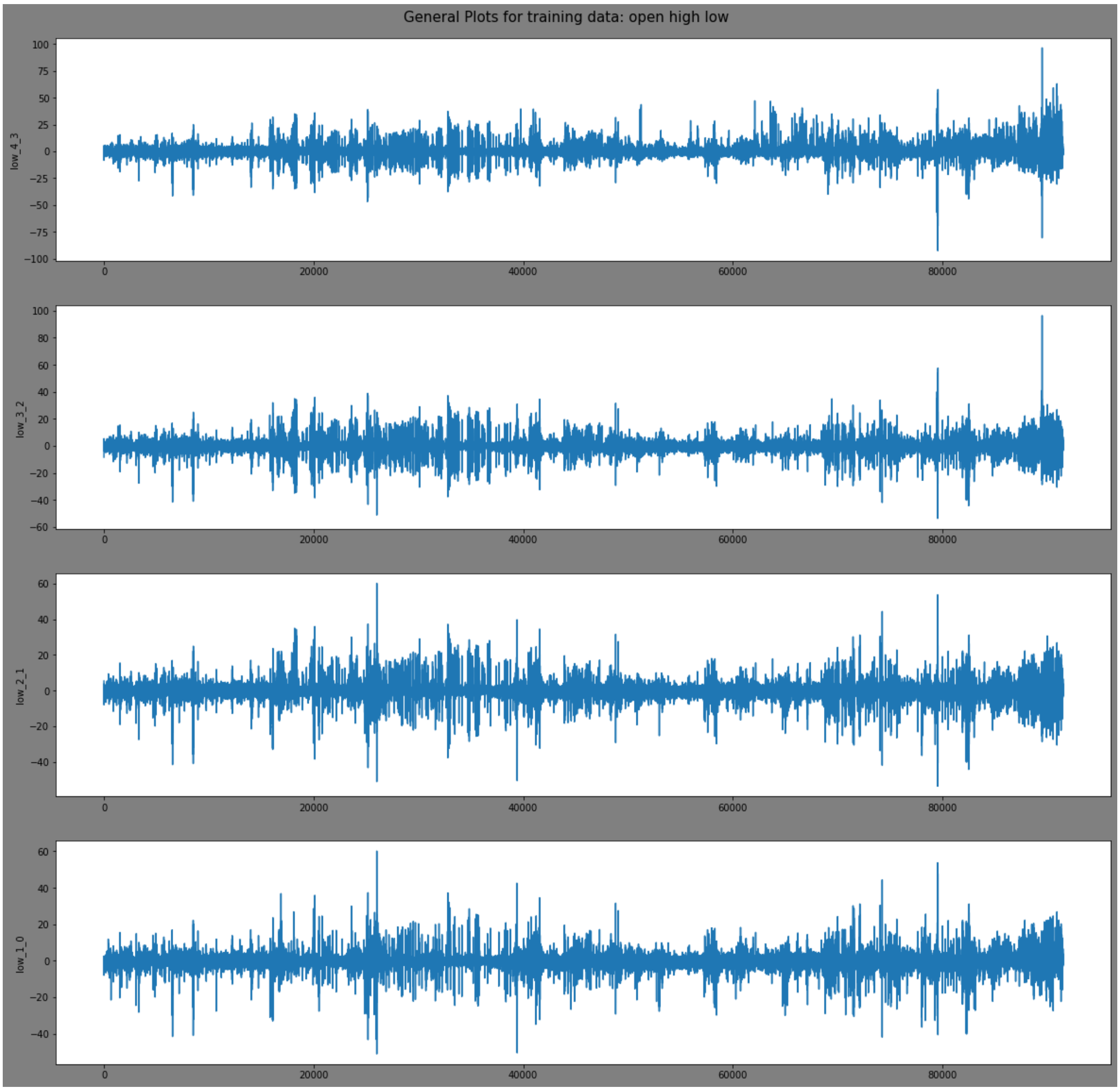
```
Plots for open:
```



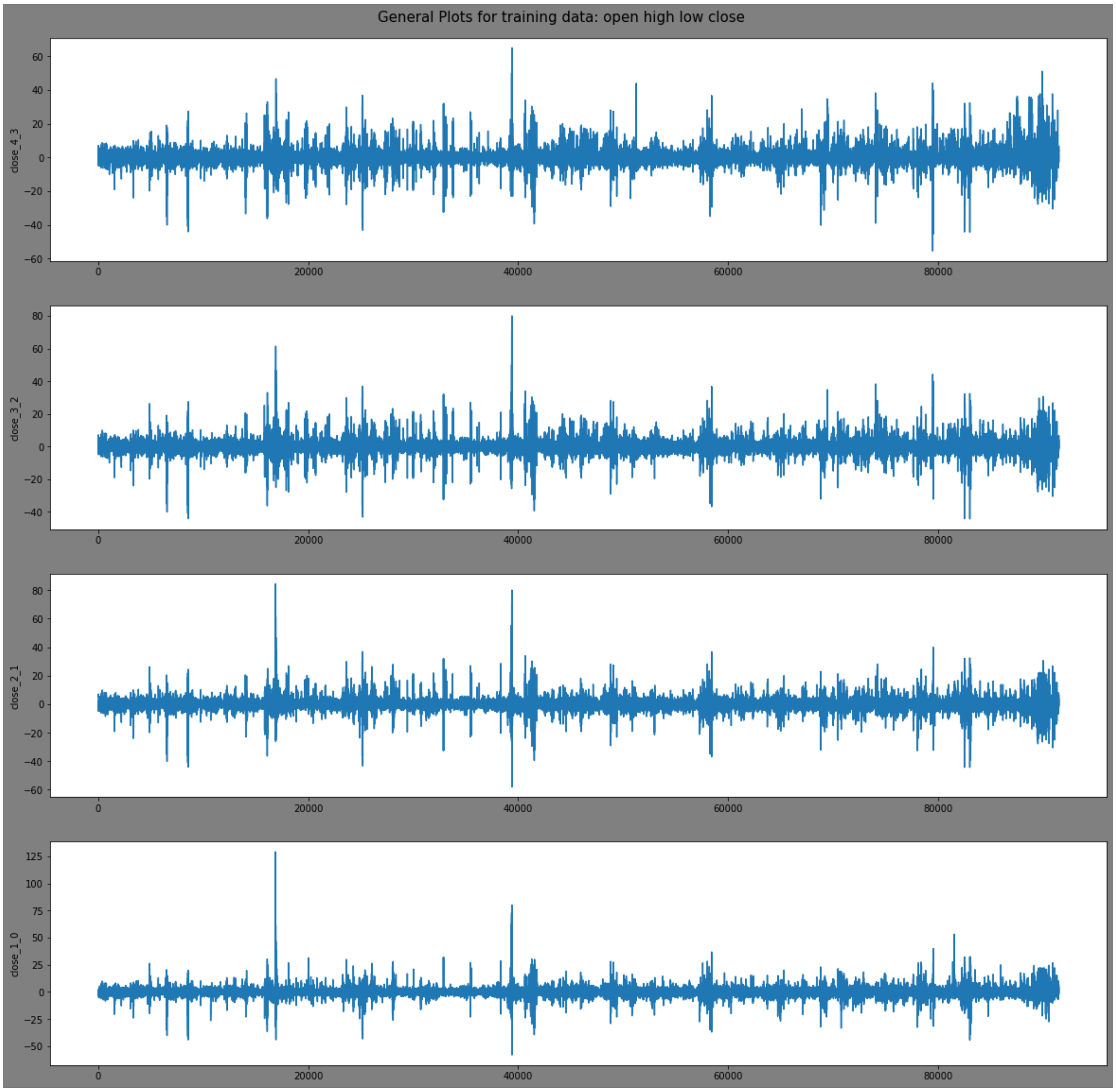

```
Plots for high:
```



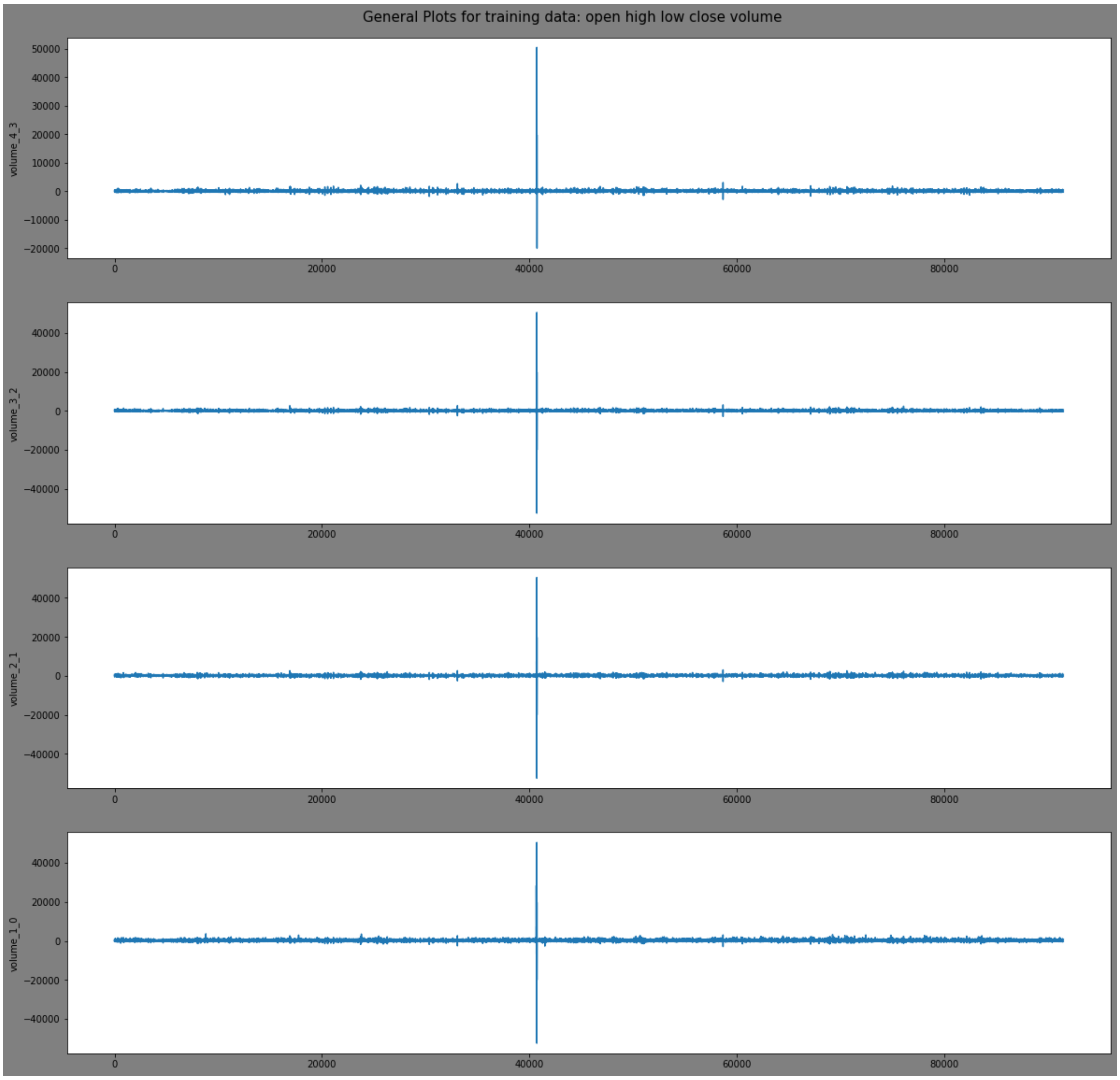
Plots for low:




```
Plots for close:
```



Plots for volume:



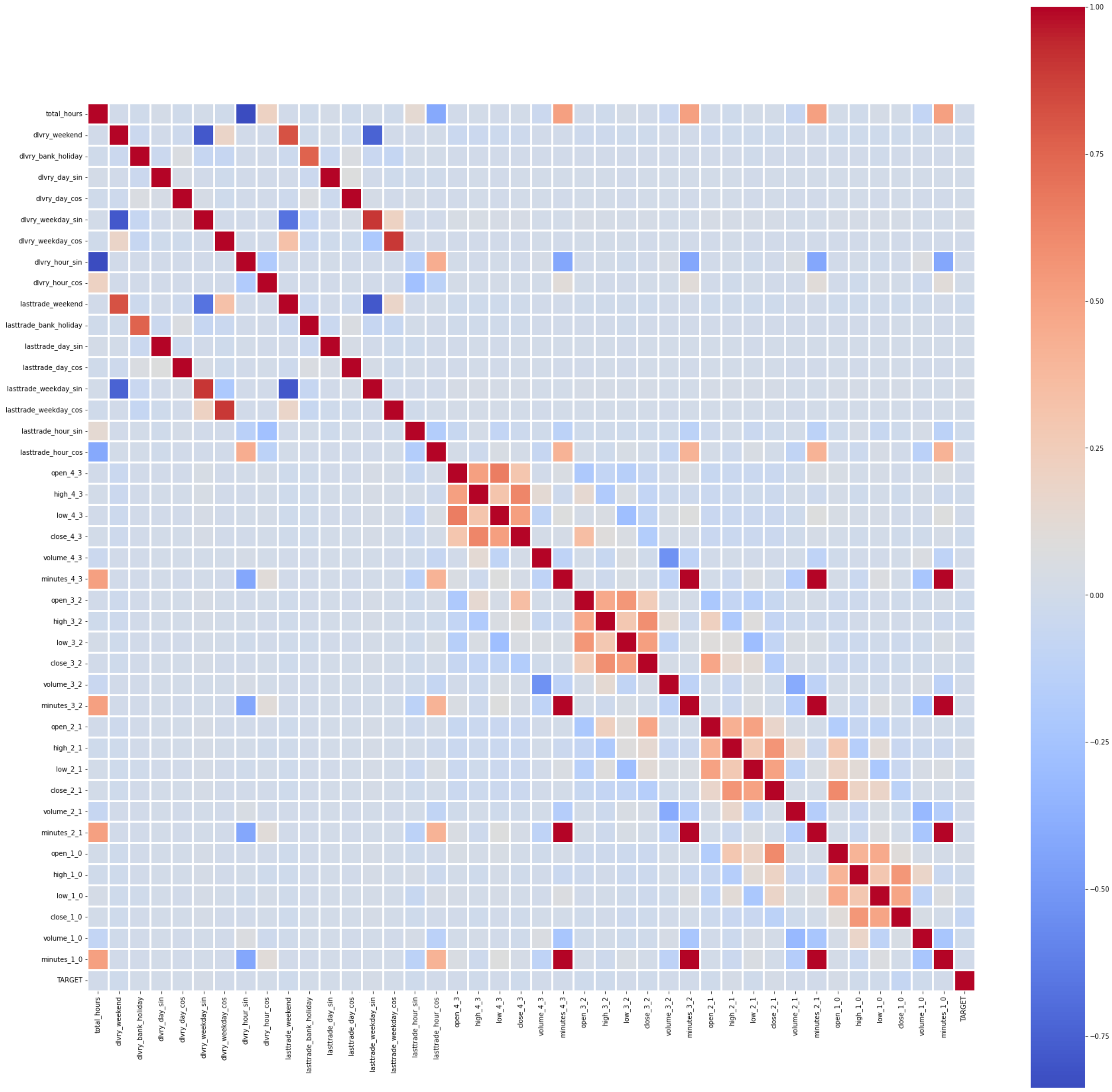
The fact that the resulting plots seem to be centered around 0 (similar to the representations of sound wave recordings) already suggests that positive differences are followed by negative differences and vice versa. Please keep in mind that this is just a crude concatenation of the open, high, low, close and volume values, across all the different contracts included in the data.

c) Correlation matrix

Finally, we can check whether there are significant linear relationships between the features using the correlation matrix. Linear relationships can easily be picked up by baseline and advanced models.

In [14]:

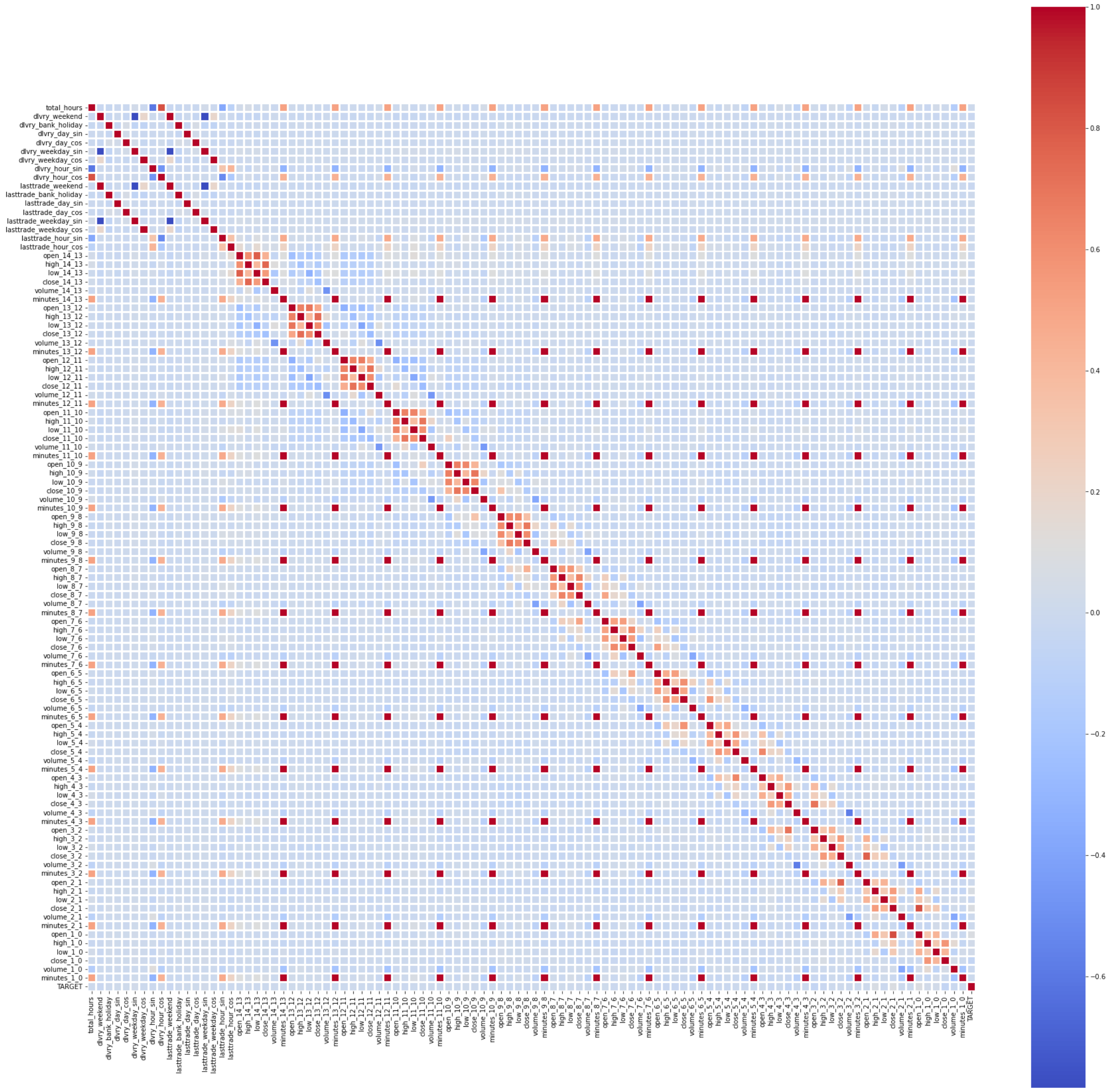
```
plt.figure(figsize=(30,30))
full_train_window_size_5 = X_train_window_size_5.copy()
full_train_window_size_5["TARGET"] = y_train_window_size_5["y"]
sns.heatmap(full_train_window_size_5.corr(), annot=False, linewidths = 2, square= Tr
plt.show()
```



The pattern seen for window size 5 continues for the additional time series features for the window size 15 data set.

In [15]:

```
plt.figure(figsize=(30,30))
full_train_window_size_15 = X_train_window_size_15.copy()
full_train_window_size_15["TARGET"] = y_train_window_size_15["y"]
sns.heatmap(full_train_window_size_15.corr(), annot=False, linewidths = 2, square= T
plt.show()
```



Takeaways from correlation matrix for windows size 5 and 15:

- The target variable is not significantly correlated (linear) with any of the predictors
- Most of the features are uncorrelated
- There are correlation patterns between the same features at different time steps and between the open, high, low and close prices differences within a specific time step

Preprocessing

Although the data already came in a preprocessed format, there are several preprocessing measures to be taken, given the reasons listed below.

- a) **Retransforming sine/cosine-transformed cyclical features** in order to provide tree-based models (splitting on one column per split) with suitable data. Since the sine/cosine transformation creates separate sine and cosine columns per cyclical feature, tree-based approaches may split on the cosine column only, which disregards the corresponding sine column for representing the actual value.
- b) **Reshaping the data from 2D to 3D** in order to make it compatible with RNNs while keeping the time series structure intact.

- c) **Applying normalization** in order to align feature scales and prevent scale-sensitive models (e.g., SVM) from taking into account differently scaled features with different weightage.

These 3 preprocessing steps will result in different data sets that are to be used for the training of different models (see other notebeooks).

a) Retransforming sine/cosine-transformed cyclical features

Tree-based approaches split nodes into sub-nodes on the basis of one feature at the time.

As we saw above, several features (i.e., dlvry_day, dlvry_weekday, dlvry_hour, lasttrade_day, lasttrade_weekday, lasttrade_hour) are sine/cosine-encoded, which means that they have a sine value and cosine value in separate columns for each feature respectively.

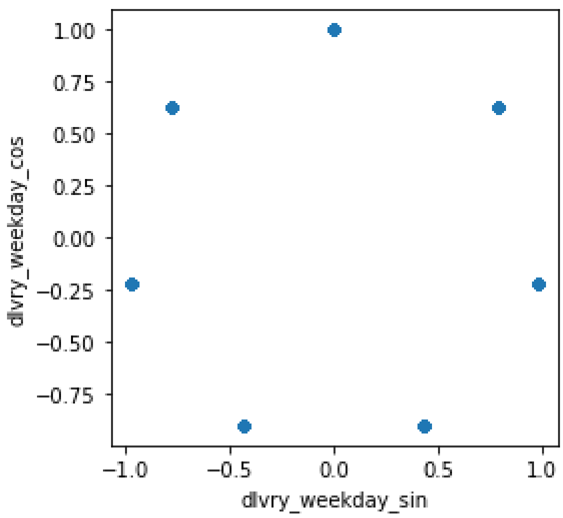
This is why, in the following, we will try to reproduce the mapping from sine/cosine values to original categorial values, assigning ordinal (e.g., Monday = 0, Tuesday = 1, ...) values. This approach, however, will erase the sine/cosine encoding's advantage of keeping the same distance between subsequent days. This is why the resulting data shall only be used for tree-based approaches.

As one cannot simply solve the underlying sine/cosine transformation formula to yield the original value (sine and cosine functions have many x-values for the same y-value), we have to reproduce the transformation in order to get the original values.

_weekday features

When we look at the values for the dlvry_weekday, we can see that there are 7 unique values, likely to correspond to the days of the week.

In [16]:
```
# plot sample of values
X_train_window_size_5.sample(500).plot.scatter('dlvry_weekday_sin','dlvry_weekday_co
```



In [17]:
```
# look at unique values
X_train_window_size_5.groupby(['dlvry_weekday_sin', 'dlvry_weekday_cos']).size().res
```

Out[17]:

	dlvry_weekday_sin	dlvry_weekday_cos	count
0	-0.974928	-0.222521	12722

	dlvry_weekday_sin	dlvry_weekday_cos	count
1	-0.781831	0.623490	12738
2	-0.433884	-0.900969	13163
3	0.000000	1.000000	12923
4	0.433884	-0.900969	13283
5	0.781831	0.623490	13156
6	0.974928	-0.222521	13527

With that information, we can calculate our own sine and cosine features and create a mapping which is to be used for creating the original feature column for days of the week.

In [18]:

```
# from observation above
days_in_a_week = 7

mapping_day_of_week = {}

# apply cosine and sine transformaiton for every day of the week
for i in range(0, days_in_a_week):
    mapping_day_of_week[(round(np.sin(2*np.pi*i/days_in_a_week), 5), round(np.cos(2*
```

In [19]:

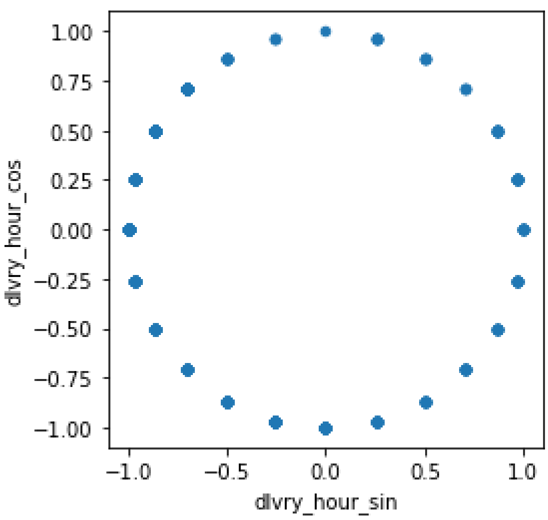
```
# create function to convert sine- and cosine-transformed values back
def convert_weekday(sin, cos):
    """Return original value from sine- and cosine-transformed values

    Arguments:
    sin -- sine-transformed value
    cos -- cosine-transformed value"""
    weekday = mapping_day_of_week[(round(sin, 5), round(cos, 5))]
    return weekday
```

_hour features

When we look at the values for the dlvry_hour, we can see that there are most likely 24 possible values (points are distributed like a 24h clock, corresponding to the hours of the day.

In [20]:

```
# plot sample of values
X_train_window_size_5.sample(500).plot.scatter('dlvry_hour_sin','dlvry_hour_cos').se
```



Again, we can reproduce a mapping as done before.

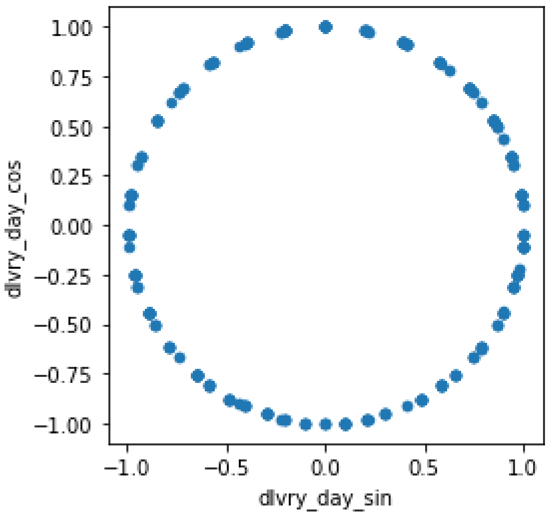
In [21]:
```
hours_in_a_day = 24

mapping_hour_of_day = {}

# apply cosine and sine transformaiton for every hour of the day
for i in range(0, hours_in_a_day):
    mapping_hour_of_day[(round(np.sin(2*np.pi*i/hours_in_a_day), 5), round(np.cos(2*
```

In [22]:
```
# create function to convert sine- and cosine-transformed values back
def convert_hour(sin, cos):
    hour = mapping_hour_of_day[(round(sin, 5), round(cos, 5))]
    return hour
```

_day features

When we look at the values for the dlvry_day, we can see that there is no obvious pattern like we observed for the _hour and _weekday features.

In [23]:
```
# plot sample of values
X_train_window_size_5.sample(500).plot.scatter('dlvry_day_sin','dlvry_day_cos').set_
```



In [24]:
```
# look at unique values
year_days = X_train_window_size_5.groupby(['dlvry_day_sin','dlvry_day_cos']).size().
year_days = year_days[['dlvry_day_sin','dlvry_day_cos']].round(5)
```

Intuitively, we might assume that the transformation is based on 365 days in a year, but creating the respective mapping does not yield matches (see below), so we couldn't reproduce the sine and cosine features for the _day features.

In [25]:
```
days_in_a_year = 365

mapping_day = {}

for i in range(0, days_in_a_year):
    mapping_day[(round(np.sin(2*np.pi*i/days_in_a_year), 5), round(np.cos(2*np.pi*i/

# check whether there is a match with randomly picked values from the unique values
try:
    mapping_day[(0.98847, 0.15143)]
```

```
except KeyError:
    print("There is no match")
```

```
There is no match
```

Let's create the tree-friendly data.

In [26]:

```
def tree_transform(df, drop_day=True):
    """Return tree-friendly data with cos and sin features transformed back to origi

    Arguments:
    df -- dataset
    drop_day -- _day features cannot be transformed back, can be dropped or kept"""
    # create new columns for original values
    data = df.copy()
    data["dlvry_weekday"] = data.apply(lambda x: convert_weekday(x['dlvry_weekday_si
    data["lasttrade_weekday"] = data.apply(lambda x: convert_weekday(x['lasttrade_we
    data["dlvry_hour"] = data.apply(lambda x: convert_hour(x['dlvry_hour_sin'], x['d
    data[["lasttrade_hour"]] = data.apply(lambda x: convert_hour(x['lasttrade_hour_s

    # drop redundant sine and cosine features
    data = data.drop(['dlvry_weekday_sin', 'dlvry_weekday_cos', 'lasttrade_weekday_s
                      'lasttrade_weekday_cos', 'dlvry_hour_sin', 'dlvry_hour_cos',
                      'lasttrade_hour_sin', 'lasttrade_hour_cos'], axis=1)
    # drop _day features as we couldn't transform them back
    if drop_day:
        data = data.drop(['dlvry_day_cos', 'dlvry_day_sin', 'lasttrade_day_sin', 'la

    return data
```

In [27]:

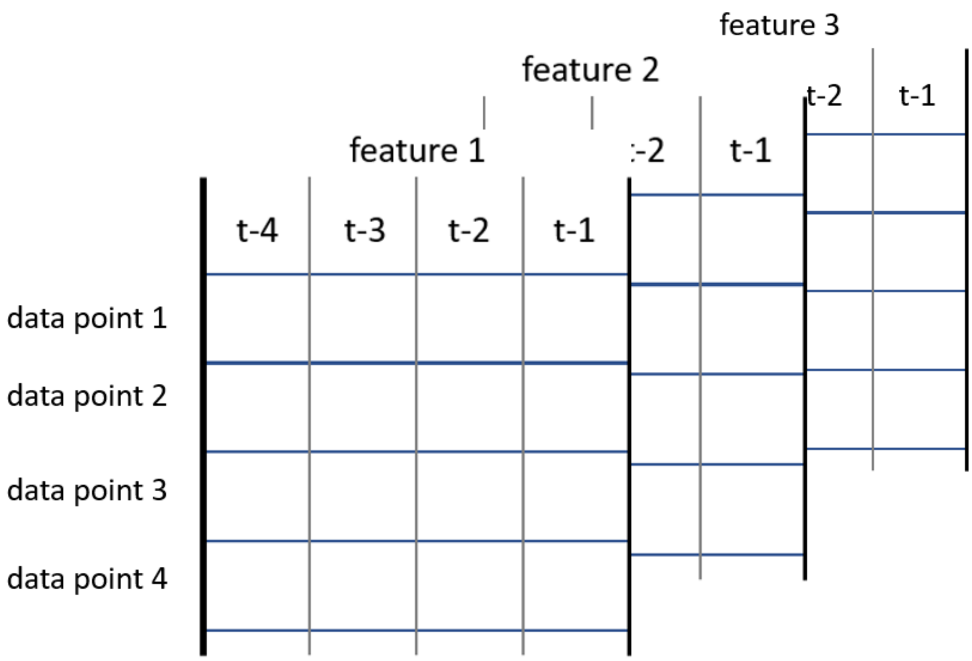
```
# create tree-friendly datasets
X_train_window_size_5_tree = tree_transform(X_train_window_size_5)
X_valid_window_size_5_tree = tree_transform(X_valid_window_size_5)
X_test_window_size_5_tree = tree_transform(X_test_window_size_5)
X_train_window_size_15_tree = tree_transform(X_train_window_size_15)
X_valid_window_size_15_tree = tree_transform(X_valid_window_size_15)
X_test_window_size_15_tree = tree_transform(X_test_window_size_15)
```

b) Reshaping the data from 2D to 3D

RNNs require inout to be 3D with the second dimension being the time steps and the third dimension being the number of features.

In the following, we will reshape all of the features as well as only the open, high, low, close and volume features as 3D arrays.

The stacking-of-2D-tables approach can be shown with the image below.



In [28]:

```
def unflatten_ohlcv(data):
    """Return 3D data for open, high, low, close and volume (ohlcv)
    to make the data suitable for recurrent neural networks

    Arguments:
    data -- predictor data after renaming and normalization
    """

    # only keep open, high, low, close and volume features
    X = data.iloc[:,17:]
    # drop minutes left feature as it is a linear function of time
    minute_col = [col for col in X.columns if 'minutes' in col]
    X = X.drop(minute_col, axis = 1)

    # separate data per feature
    open_col = [col for col in X.columns if 'open' in col]
    X_open = X[open_col]
    high_col = [col for col in X.columns if 'high' in col]
    X_high = X[high_col]
    low_col = [col for col in X.columns if 'low' in col]
    X_low = X[low_col]
    close_col = [col for col in X.columns if 'close' in col]
    X_close = X[close_col]
    vol_col = [col for col in X.columns if 'volume' in col]
    X_vol = X[vol_col]

    # stack data
    X = np.stack([X_open, X_high, X_low, X_close, X_vol], axis = 2)

    return X
```

In [29]:

```
# transform window size 5 data
X_train_unflatten_5_ohlcv = unflatten_ohlcv(X_train_window_size_5)
X_valid_unflatten_5_ohlcv = unflatten_ohlcv(X_valid_window_size_5)
X_test_unflatten_5_ohlcv = unflatten_ohlcv(X_test_window_size_5)
print("Check dimensions. Second dimension should be 4.")
np.shape(X_train_unflatten_5_ohlcv), np.shape(X_valid_unflatten_5_ohlcv), np.shape(X
```

```
Check dimensions. Second dimension should be 4.
```

Out[29]:
```
((91512, 4, 5), (4817, 4, 5), (10704, 4, 5))
```

In [30]:
```
# transform window size 15 data
X_train_unflatten_15_ohlcv = unflatten_ohlcv(X_train_window_size_15)
X_valid_unflatten_15_ohlcv = unflatten_ohlcv(X_valid_window_size_15)
X_test_unflatten_15_ohlcv = unflatten_ohlcv(X_test_window_size_15)
print("Check dimensions. Second dimension should be 14.")
np.shape(X_train_unflatten_15_ohlcv), np.shape(X_valid_unflatten_15_ohlcv), np.shape
```

```
Check dimensions. Second dimension should be 14.
```

Out[30]:
```
((29888, 14, 5), (1574, 14, 5), (3496, 14, 5))
```

For features other than open, high, low, close and volume there does not exist a (different) value for each time step, so we copied the respective features to be in line with the dimensionality requirements (4 or 14 times depending on the data set).

In [31]:
```
def unflatten_all(data, window_size):
    """Return 3D data for all features (except minutes left)
    to make the data suitable for recurrent neural networks

    Arguments:
    data -- predictor data after renaming and normalization
    window_size -- either 5 or 15
    """

    timesteps = window_size - 1

    X_no_ts = []

    # copy every non time series feature to get right shape
    for feature in data.iloc[:,:17].columns:
      X_feat = pd.concat([data[feature]]*timesteps, axis=1)
      X_no_ts.append(X_feat)

    # only keep open, high, low, close and volume features
    X = data.iloc[:,17:]
    # drop minutes left feature as it is a linear function of time
    minute_col = [col for col in X.columns if 'minutes' in col]
    X = X.drop(minute_col, axis = 1)

    # separate data per feature
    open_col = [col for col in X.columns if 'open' in col]
    X_open = X[open_col]
    high_col = [col for col in X.columns if 'high' in col]
    X_high = X[high_col]
    low_col = [col for col in X.columns if 'low' in col]
    X_low = X[low_col]
    close_col = [col for col in X.columns if 'close' in col]
    X_close = X[close_col]
    vol_col = [col for col in X.columns if 'volume' in col]
    X_vol = X[vol_col]

    # all seperate data
    X_all_feat = [X_open, X_high, X_low, X_close, X_vol]
    X_all_feat.extend(X_no_ts)

    # stack data
    X = np.stack(X_all_feat, axis = 2)

    return X
```

In [32]:
```
# transform window size 5 data
X_train_unflatten_all_5 = unflatten_all(X_train_window_size_5, 5)
X_valid_unflatten_all_5 = unflatten_all(X_valid_window_size_5, 5)
X_test_unflatten_all_5 = unflatten_all(X_test_window_size_5, 5)
print("Check dimensions. Second dimension should be 4.")
np.shape(X_train_unflatten_all_5), np.shape(X_valid_unflatten_all_5), np.shape(X_tes
```

```
Check dimensions. Second dimension should be 4.
```
Out[32]:
```
((91512, 4, 22), (4817, 4, 22), (10704, 4, 22))
```

In [33]:
```
# transform window size 15 data
X_train_unflatten_all_15 = unflatten_all(X_train_window_size_15, 15)
X_valid_unflatten_all_15 = unflatten_all(X_valid_window_size_15, 15)
X_test_unflatten_all_15 = unflatten_all(X_test_window_size_15, 15)
print("Check dimensions. Second dimension should be 14.")
np.shape(X_train_unflatten_all_15), np.shape(X_valid_unflatten_all_15), np.shape(X_t
```

```
Check dimensions. Second dimension should be 14.
```
Out[33]:
```
((29888, 14, 22), (1574, 14, 22), (3496, 14, 22))
```

c) Applying normalization

Large differences in the scales of different features (think for example height in meters vs. number of blood cells) can cause problems for many models which might assign higher wieghting to large numbers. As the underlying unit should be irrelevant, we can align the features' scale without distorting the differences between values within the ranges, thereby not losing information.

Please note, that scale-robust models such as tree-based approaches should not perform worse when the input data is normalized due to the aforementioned characteristis of not changing the relative values within the range.

In the following we will apply the MinMaxScaler to the time series features of the data sets.

In [34]:
```
scaler5 = MinMaxScaler()
X_train_window_size_5[X_train_window_size_5.columns[17:]] = scaler5.fit_transform(X_
X_valid_window_size_5[X_valid_window_size_5.columns[17:]] = scaler5.transform(X_vali
X_test_window_size_5[X_test_window_size_5.columns[17:]] = scaler5.transform(X_test_w

scaler15 = MinMaxScaler()
X_train_window_size_15[X_train_window_size_15.columns[17:]] = scaler15.fit_transform
X_valid_window_size_15[X_valid_window_size_15.columns[17:]] = scaler15.transform(X_v
X_test_window_size_15[X_test_window_size_15.columns[17:]] = scaler15.transform(X_tes
```

Save all datasets

In [35]:
```
# tree-friendly data
# read back using:
#       <name> = pd.read_csv("<name>"", index_col=0)
X_train_window_size_5_tree.to_csv("X_train_window_size_5_tree.csv")
X_valid_window_size_5_tree.to_csv("X_valid_window_size_5_tree.csv")
X_test_window_size_5_tree.to_csv("X_test_window_size_5_tree.csv")
X_train_window_size_15_tree.to_csv("X_train_window_size_15_tree.csv")
X_valid_window_size_15_tree.to_csv("X_valid_window_size_15_tree.csv")
X_test_window_size_15_tree.to_csv("X_test_window_size_15_tree.csv")
```

```python
# normalized data
# read back using:
#      <name> = pd.read_csv("<name>"", index_col=0)
X_train_window_size_5.to_csv("X_train_window_size_5.csv")
X_valid_window_size_5.to_csv("X_valid_window_size_5.csv")
X_test_window_size_5.to_csv("X_test_window_size_5.csv")
X_train_window_size_15.to_csv("X_train_window_size_15.csv")
X_valid_window_size_15.to_csv("X_valid_window_size_15.csv")
X_test_window_size_15.to_csv("X_test_window_size_15.csv")

y_train_window_size_5.to_csv('y_train_window_size_5.csv')
y_valid_window_size_5.to_csv('y_valid_window_size_5.csv')
y_test_window_size_5.to_csv('y_test_window_size_5.csv')
y_train_window_size_15.to_csv('y_train_window_size_15.csv')
y_valid_window_size_15.to_csv('y_valid_window_size_15.csv')
y_test_window_size_15.to_csv('y_test_window_size_15.csv')

# 3D data has to be saved using pickle as it's 3-dimensional
# read back using:
#     pkl_file = open("<name>", 'rb')
#     <name> = pickle.load(pkl_file)
datasets_3D = {"X_train_unflatten_5_ohlcv": X_train_unflatten_5_ohlcv,
               "X_valid_unflatten_5_ohlcv": X_valid_unflatten_5_ohlcv,
               "X_test_unflatten_5_ohlcv": X_test_unflatten_5_ohlcv,
               "X_train_unflatten_15_ohlcv": X_train_unflatten_15_ohlcv,
               "X_valid_unflatten_15_ohlcv": X_valid_unflatten_15_ohlcv,
               "X_test_unflatten_15_ohlcv": X_test_unflatten_15_ohlcv,
               "X_train_unflatten_all_5": X_train_unflatten_all_5,
               "X_valid_unflatten_all_5": X_valid_unflatten_all_5,
               "X_test_unflatten_all_5": X_test_unflatten_all_5,
               "X_train_unflatten_all_15": X_train_unflatten_all_15,
               "X_valid_unflatten_all_15": X_valid_unflatten_all_15,
               "X_test_unflatten_all_15": X_test_unflatten_all_15}

for data in datasets_3D:
    output = open(data, 'wb')
    pickle.dump(datasets_3D[data], output)
    output.close()
```