elucidata_demonstrator_3_3

November 25, 2022

```
[ ]: import support as sp
     import visualization as vis

     %load_ext autoreload
     %autoreload 2
```

1 Starter Kit 3.3: Feature Engineering

1.1 Description

Most data mining and machine learning algorithms do not work well if you just feed them your raw data: such data often contains noise and the most relevant distinguishing information is often implicit. For example, raw sensor data just contains a timestamp and a corresponding value, while interesting aspects of it might be the trends contained within or the number of times a threshold is exceeded.

Feature engineering is the process of extracting from the raw data the most relevant distinguishing characteristics that will be presented to the algorithm during modeling. It is a way of deriving new information from the existing data, so that its characteristics are more explicitly represented. The resulting features are eventually feed to the AI/ML algorithm of the model. In practice, the feature engineering step is achieved by the selection of the most appropriate parameters, or the composition of new features by manipulation, transformation and combination of the raw data.

Feature engineering is one of the most important and creative steps in the data science workflow. However, there is no clearly-defined formal process for engineering features and, consequently, this requires a lot of creativity, a good understanding of the domain and of the available data, some trial-and-error, etc. It is also desirable to have upfront an idea of the modeling and analysis task for which you want to use the resulting features, as this might help you to identify relevant features.

1.2 Business goal

The overall goal of this Starter Kit is to present some **advanced feature engineering steps** related to **feature construction** and **extraction**. By keeping in mind **what is your business question** and **what is the corresponding data science task**, you will be able to derive **valuable features** that can be used in the next stage of your analysis.

1.3 Application context

Feature engineering is one of the steps in the data science workflow with the most decisive impact on the accuracy of the model you want to develop. It is the final step before actually training the model and it defines how the input data will be fed to the model.

1.4 Starter Kit outline

In this Starter Kit we use a dataset from the Regional Rail System of Pennsylvania. Before starting the feature extraction *per se*, we will first apply some basic preprocessing to the dataset and have a high-level overview of the data. We will then derive several interesting features from the dataset that can be used to characterise train rides and delays.

1.5 Basic data understanding

In order to illustrate how to engineer features, we will use in this Starter Kit a dataset provided by the Southeastern Pennsylvania Transportation Authority (SEPTA), which can be downloaded here.

The SEPTA Regional Rail System consists of 13 branches and more than 150 stations in Philadelphia, Pennsylvania, and its suburbs and satellite cities, as depicted in the map below.

SEPTA uses On-Time Performance (OTP) to measure service reliability. OTP identifies the number of trains for all rail lines that arrive at their scheduled destination at the scheduled time. However, by industry standard, a train may arrive up to 5 minutes and 59 seconds after its scheduled time and still be considered on-time.

SEPTA has established an annual goal of 91% for Regional Rail On-Time Performance. Based on this goal, we will demonstrate how feature engineering can be used to validate SEPTA's requirement. More specifically, we will start designing features that may help us **predict** delays of a train on a given **stop** at a given **date**.

The SEPTA dataset consists of 2 CSV files that we will consider in this Starter Kit. The first file, `otp.csv`, provides OTP data, that is, information about trains and the times at which they arrive at the different stations on their lines. The table below shows an excerpt of this OTP dataset.

[ ]:
```
df_otp = sp.load_otp_data()
df_otp.head(10)
```

```
2022-11-25 11:43:36,796 [INFO ]  Using file: /Users/mdhn/Documents/starter-
kits/repository/elucidatalab.starterkits/data/SK_3_3/otp.csv
```

[ ]:

	train_id	direction	origin	next_station	date	\
0	778	N	Trenton	Stenton	2016-03-23	
1	598	N	Thorndale	Narberth	2016-03-23	
2	279	S	Elm	Ridley Park	2016-03-23	
3	476	N	Airport Terminal E-F	Suburban Station	2016-03-23	
4	474	N	Airport Terminal E-F	Jenkintown-Wyncote	2016-03-23	
5	279	S	Elm	Crum Lynne	2016-03-23	
6	778	N	Trenton	Sedgwick	2016-03-23	
7	598	N	Thorndale	Merion	2016-03-23	

```
8      395         S                 Trent             Temple U 2016-03-23
9     6464         N         Powelton Yard  30th Street Station 2016-03-23

    status           timeStamp
0    1 min 2016-03-23 00:01:47
1    1 min 2016-03-23 00:01:58
2    2 min 2016-03-23 00:02:02
3  On Time 2016-03-23 00:03:19
4  On Time 2016-03-23 00:03:35
5    2 min 2016-03-23 00:03:42
6    2 min 2016-03-23 00:03:48
7    1 min 2016-03-23 00:03:58
8  On Time 2016-03-23 00:04:17
9  On Time 2016-03-23 00:04:22
```

The different attributes in this file OTP dataset are as follows: * `train_id`: the identifier of the train * `direction`: the train direction; its values are 'N' for Northbound and 'S' for Southbound * `origin`: the station before *next_station* * `next_station`: the next station stop at *timeStamp* * `date`: the date of the journey * `status`: the current train delay; its values are 'On Time' or the delay, i.e. the amount of time above the 5min59s limit until which a train is considered to be on time (e.g. '1 min', '5 min', '10 min'); a value of 999 indicates a suspended train * `timeStamp`: the timestamp at which the train will arrive at `next_station`

The second file, `trainView.csv`, provides train tracking information:

[ ]:
```
df_train_view = sp.load_train_view_data()
df_train_view.head(3)
```

```
2022-11-25 11:43:52,015 [INFO ]  Using file: /Users/mdhn/Documents/starter-
kits/repository/elucidatalab.starterkits/data/SK_3_3/trainView.csv
```

[ ]:
```
  train_id status next_station service              dest       lon       lat  \
0    102TT      0       Radnor   LOCAL  Colmar-Link Belt -75.37250  40.04388
1    102TT      0   St. Davids   LOCAL  Colmar-Link Belt -75.38670  40.04583
2    102TT      0    Strafford   LOCAL  Colmar-Link Belt -75.42277  40.04722

  source track_change track       date           timeStamp0  \
0  Devon           -1    -1 2016-04-22  2016-04-22 13:21:07
1  Devon           -1    -1 2016-04-22  2016-04-22 13:19:11
2  Devon           -1    -1 2016-04-22  2016-04-22 13:15:04

            timeStamp1  seconds
0  2016-04-22 13:22:43       96
1  2016-04-22 13:21:01      110
2  2016-04-22 13:17:01      117
```

Its most important attributes are: * `lon`: the current GPS longitude of the train * `lat`: the current GPS latitude of the train * `timeStamp0`: the earliest timestamp at the current GPS coordinates *

`timeStamp1`: the latest timeStamp at the current GPS coordinates * `seconds`: the time difference (in seconds) between `timeStamp1` and `timeStamp0` * `track_change`: The name of the track if there was a track change, else -1

We can already observe in the previews of these two datasets format differences at the level of the `train_id` and `status` columns. We will address these in the next section.

1.6 Data preprocessing

1.6.1 Removing invalid data instances

A manual inspection of the dataset reveals that the `train_id` column from the OTP data contains 1 negative value and many values containing letters and punctuation characters. Let's filter out the rows for which the `train_id` is not a number.

[ ]:
```
df_otp = sp.remove_invalid_data_instances(df=df_otp)
```

```
We found 35964 IDs with non-numeric characters, which is 1.91% from a total of
1882015 IDs.
In addition, there are 88333 entries with next_station name = None
These entries with an unexpected train ID or no station name information will be
excluded from the dataset
```

1.6.2 Understanding train_id

`train_id` can be used an identifier for a specific train run between a departure and a destination station at a given time of the day. To confirm that this is possible, we count how many times a `train_id` passes through a given station on a given day in the `train_run` dataset:

[ ]:
```
train_run = sp.get_train_run(df=df_otp)
print("In %0.2f%% of the cases, a train_id passes only once through a station␣
 ↪on a given day." %
       (train_run.single_pass.sum() / float(len(train_run)) * 100))
```

```
In 99.81% of the cases, a train_id passes only once through a station on a given
day.
```

We will exclude from the dataset the remaining 0.2% of cases, since they correspond to exceptional situations where a `train_id` was registered more than once in a day. For consistency, we only keep in the trainView dataset those rows for which the (`train_id`, `next_station`, `date`) tuples also occur in the OTP dataset.

For the remainder of the notebook, we will refer to **train id** as an identifier for a train journey between a given *origin* and a given *destination* at a given *time* of the day. In other words, a train id can be repeated on multiple days. A **train run** on the other hand, is now defined as a train id on a specific date, as used in the trainview dataset.

[ ]:
```
df_otp, df_train_view = sp.exclude_remaining_cases(otp=df_otp,␣
 ↪train_view=df_train_view)
```

1.6.3 Turning statuses into workable delays

The current string-based format of the `status` column in the OTP dataset is not suitable for performing calculations on the delays. We will transform its values into integers (in minutes) and rename the column to `delay`.

[ ]:
```
df_otp = sp.calculate_delays_for_otp(df=df_otp)
```

```
    status  delay
0    1 min      1
1    1 min      1
2    2 min      2
3  On Time      0
4  On Time      0
```

We will also exclude missing (`None`) values present in the `status` column of the df_train_view dataset, and, as for the OTP dataset, convert the values of that column to integers and rename the column to `delay`.

[ ]:
```
df_train_view = sp.calculate_delays_for_train_view(df=df_train_view)
```

1.6.4 Removing suspended trains

Since a status of `999` represented a suspended train, we exclude those trains from the datasets. In addition, a delay of 1440 minutes corresponds to one day delay, meaning that the train was most likely canceled. We exclude the trains with those delays from the datasets as well.

[ ]:
```
df_otp, df_train_view = sp.remove_suspended_trains(df_otp=df_otp,␣
↪df_train_view=df_train_view)
```

1.7 Data overview

Let's check some basic characteristics of the (cleaned OTP) dataset.

[ ]:
```
sp.get_otp_overview(df=df_otp)
```

[ ]:
```
Number of stations                          154
Number of train ids                         995
Number of train runs                     173924
Time range                23 Mar 2016 to 06 Nov 2016
Percentage trains on time                 74.9%
```

As the table above indicates 74.9% of the trains being delayed, i.e. we demonstrate analytically that **the original requirement of having 91% of all trains on time** (definition: no more than 5min59s delay) **is far from being reached**. This is just a showcase of how feature engineering can be used in practice. In the remainder of the notebook we will present further examples to better understand the process.

1.8 Feature engineering

We are now ready to investigate several features to demonstrate the workflow and some hidden obstacles while calculating features. You might come up with other features as well, so don't hesitate to try them out.

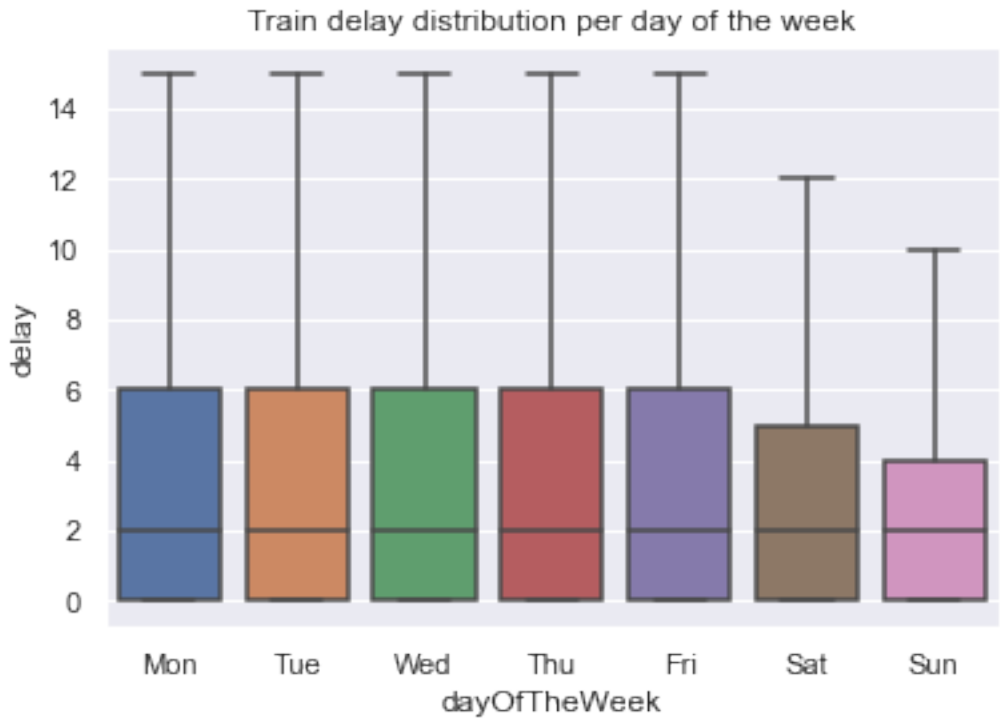
Based on our interest to characterize train rides and delays, we come up with the following features: - Day of the week - Month of the year - Rush hours - Rank the different stops in a train run - Distance between stations - Cumulative distance along a train run - Total distance of a train run - Northbound vs Southbound travels - Stations with big delay - Delay over the last 7 days - Track changes

1.8.1 Day of the week

Based on our collective experience from using public transport, we can expect that delays are more likely to happen at specific moments in a day (e.g. rush hours) and at specific days in a week (e.g. working days). In order to be able to easily verify these assumptions later on, we first extract for each entry in the OTP dataset the following features: - the hour - the name of the day - the type of day, which we define as having 3 possible values: weekday, Saturday and Sunday - and a boolean indicating whether that day is a weekday or a weekend day

Let's start by looking at the impact of day of the week on delay.

[ ]:
```
vis.plot_train_delays(df=df_otp)
```



As we can see, the difference in delays on weekdays (Mon-Fri) is only minimal. We will thus

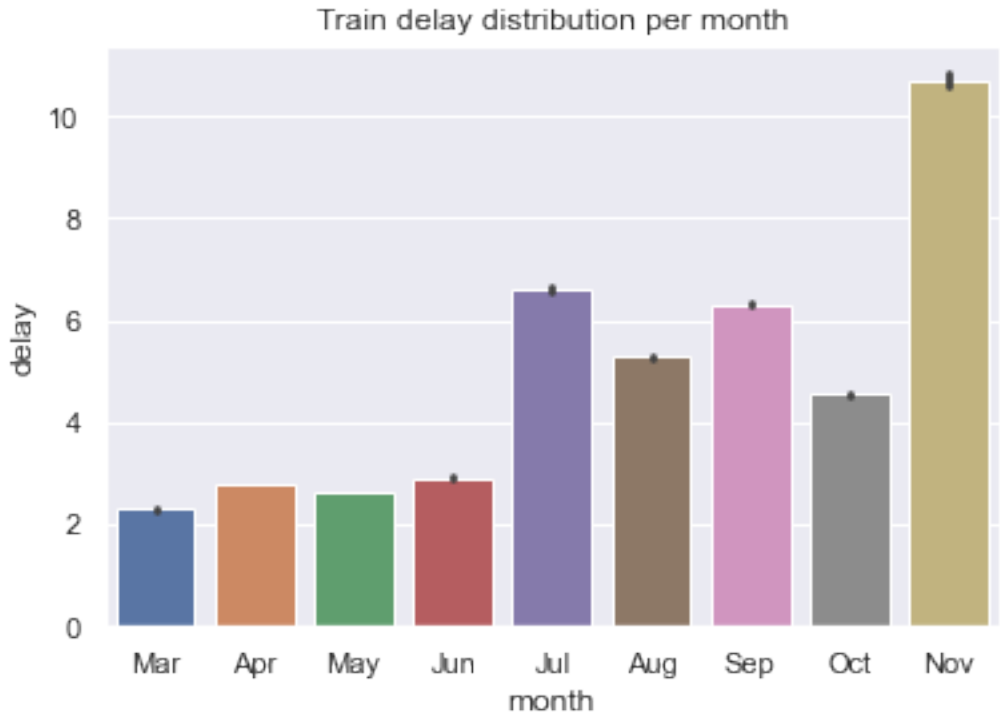
categorize the day of the week into weekdays, saturdays and sundays.

```
[ ]: df_otp = sp.get_type_of_days(df=df_otp)
```

1.8.2 Month of the year

Seasonal effects, expressed here as the month of the year, can have an impact on train performance. As an example consider the summer holidays one can expect less (commuting) traffic, albeit also reduced staff, which might affect how on-time trains are. Additionally, one might expect more (bad) weather related problems with trains or tracks during winter. Let's start by looking at delay as a function of the month of the year.

```
[ ]: vis.plot_train_delays(df=df_otp, kind='month')
```



Data only runs from March to November 2016. As we can see, there is a large variability observed between months. As a consequence, we will keep this feature, rather than grouping the months into the four seasons.

1.8.3 Rush hours

During rush hours there's an intense increase in the number of passengers and, often, of the number of trains. It is also a period where it is most critical for trains to be on-time. To identify rush hour periods, we will start by extracting some time features from the dataset: the hour and whether it is a weekend day or not.

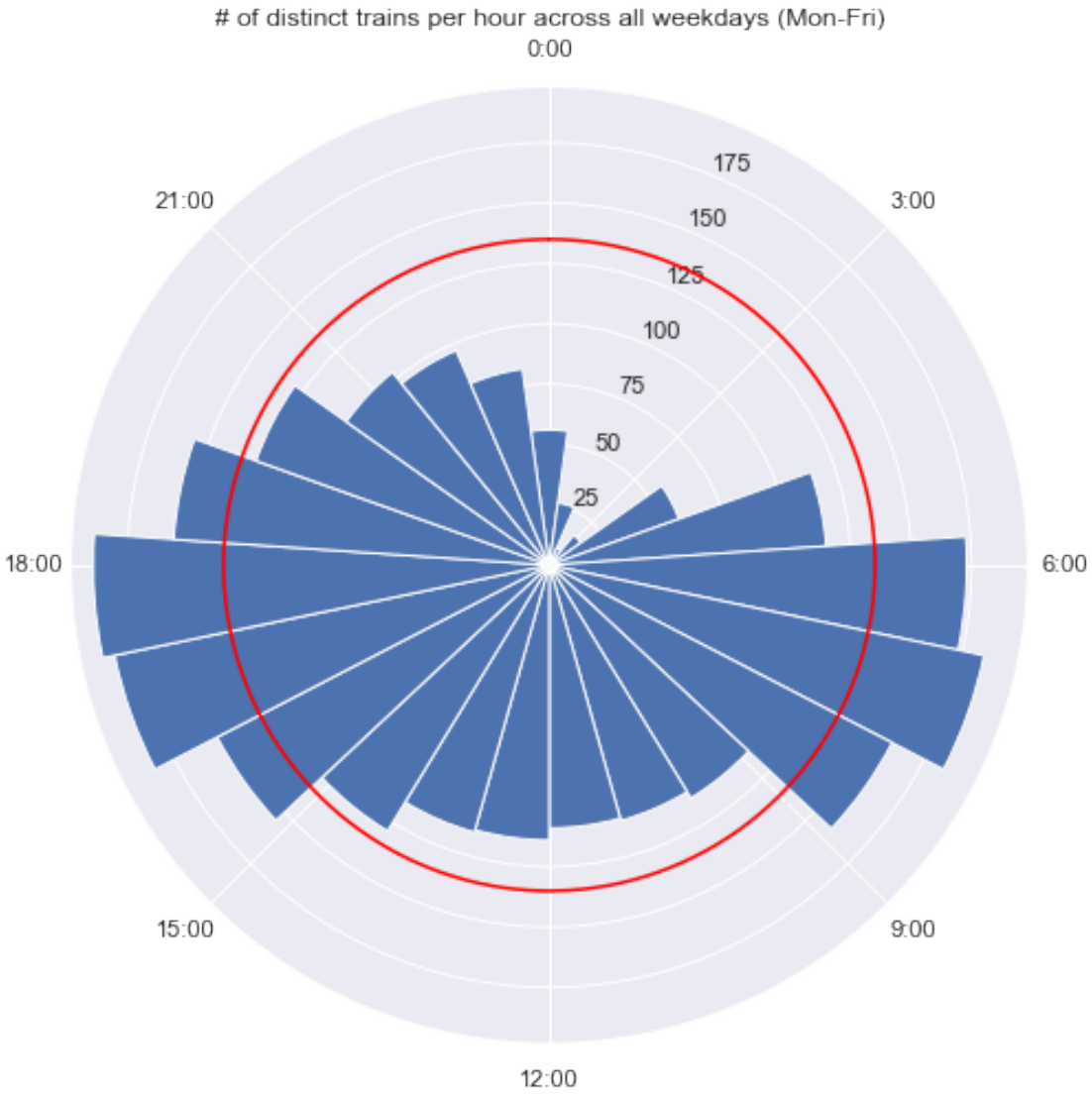
[ ]:
```
df_otp = sp.extract_time_features(df=df_otp)
df_otp[['train_id', 'origin', 'date', 'hour', 'isWeekend']].sample(5)
```

[ ]:

	train_id	origin	date	hour	isWeekend
1715296	9421	Roberts Yard	2016-11-01	09	False
1522404	9703	Roberts Yard	2016-10-09	07	True
1753194	3534	Elwyn	2016-11-05	19	True
941768	215	Elm St	2016-07-20	13	False
52691	7327	Chestnut Hill East	2016-03-29	08	False

Let's now use these features to count the number of distinct trains in the rail system per hour during weekdays, and hence have an idea of the train density per hour. We define a rush hour as one for which the number of trains per hour and across all hours and weekdays is above the 75th percentile. Let's visualize the train density per hour on weekdays with that threshold.

[ ]:
```
vis.plot_train_density(df=df_otp)
```



We will now define rush hour as all times on weekdays that go beyond the 75th percentile, which is 135 trains per hour as indicated with the red curve in the plot above. As an example, we will also provide an excerpt of the OTP data.

[ ]:
```
df_otp = sp.define_rush_hour(df=df_otp, percentile=75)
df_otp[['train_id', 'origin', 'date', 'hour', 'dayOfTheWeek', 'isRushHour']].
↪sample(5).query("isRushHour==True").head()
```

```
The threshold that we will use is of 135 trains per hour over all weekdays and
hours.
Identified rush hours are: 07, 08, 09, 16, 17, 18, 19.
```

[ ]:

	train_id	origin	date	hour	dayOfTheWeek	isRushHour
482579	521	Doylestown	2016-05-17	09	Tue	True

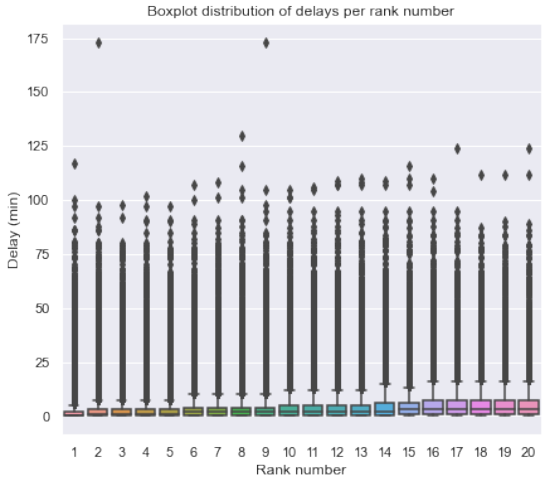
1.8.4 Rank the different stops in a train run

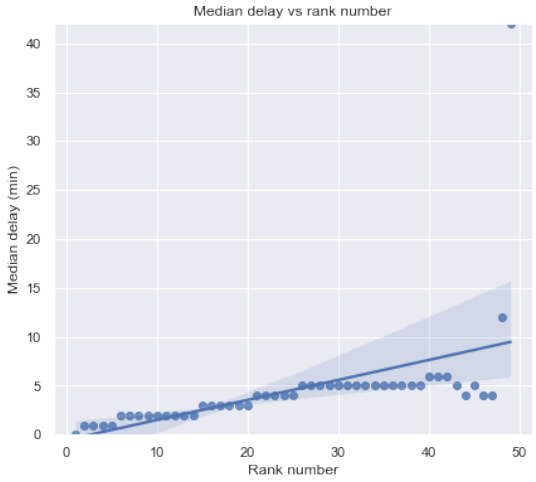
The fact that a `train_id` passes only once through a station on a given day allows us to assume that a `train_id` is defined for a specific trip (origin-destination) at a given time in the day. So the combination of a `train_id` and `date` identifies a unique train run (as we defined before). We construct the feature *rank* to identify the sequence number of each stop along the train run, i.e., 1 is origin station, 2 is the second stop, 3 is the third stop, etc.

[ ]:
```
df_otp = sp.calculate_rank_stop(df=df_otp)
```

We can now look at the delay as a function of the rank number.

[ ]:
```
vis.plot_delays_per_stop_number(df=df_otp)
```





We can see that the delay is a function of the train stop number, confirming the idea that trains accumulate delays along the run, without being able of (fully) compensating for them.

1.8.5 Distance between stations

We will now use the coordinates information to derive the distance in km between stations, the total distance per train run and the cumulative distance along a train run. Since there are multiple coordinate points for each stop (train position might be registered at different time points, e.g. when entering the station, when docking, when departing, etc.), we will take the average latitude and longitude for each station.

[ ]:
```
uni_coords, uni_runs, df_distance, df_coords = sp.
 ↪calculate_distance_between_stations(df_otp=df_otp,␣
 ↪df_train_view=df_train_view)
df_coords[['train_id', 'date', 'rank', 'next_station', 'upcoming_station',␣
 ↪'distance']].sample(5)
```

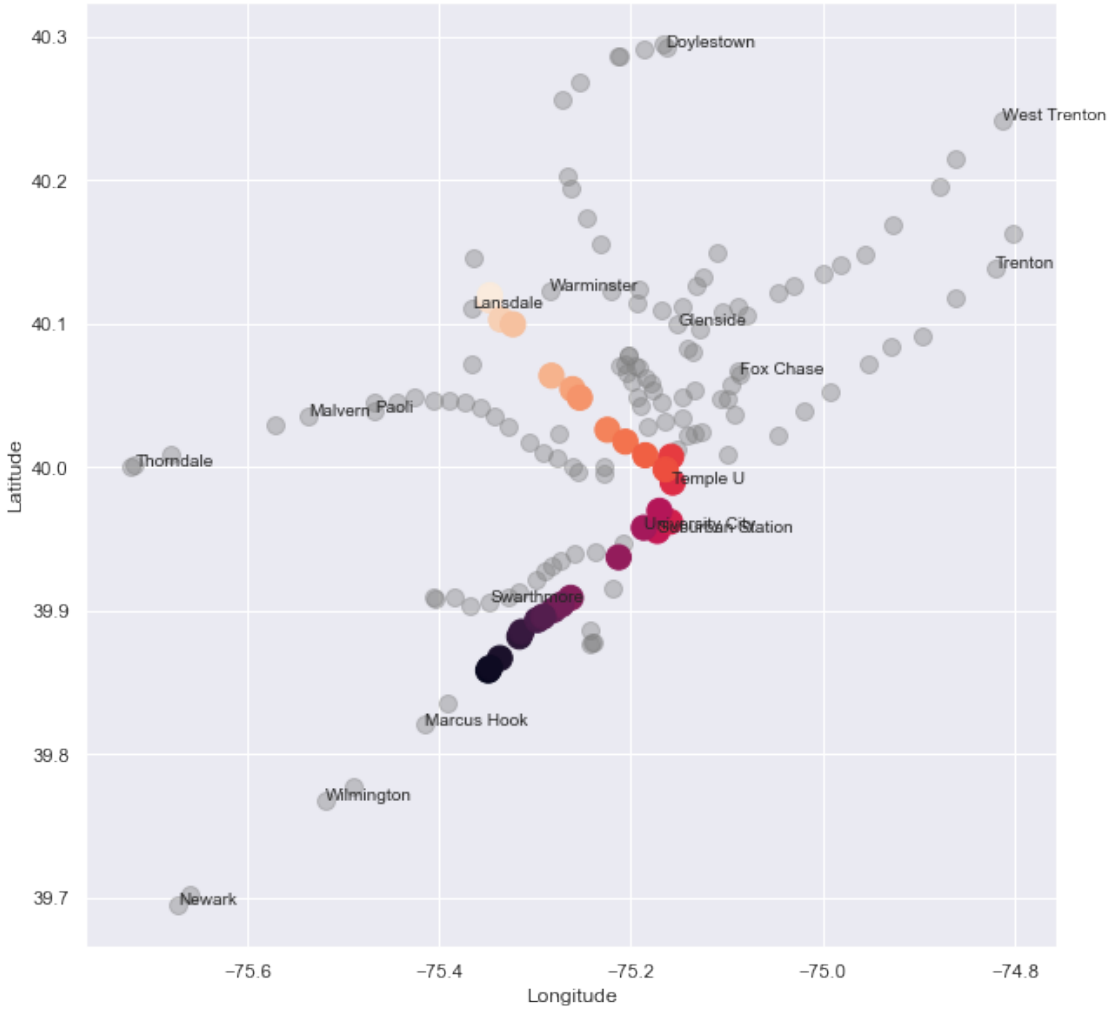
[ ]:

	train_id	date	rank	next_station	upcoming_station	\
1383031	475	2016-09-25	11.0	Eastwick Station	University City	
694135	389	2016-06-27	8.0	Philmont	Forest Hills	
1339992	521	2016-05-20	45.0	Thorndale	Downingtown	
1085449	522	2016-10-12	7.0	30th Street Station	Overbrook	
1459200	4213	2016-09-27	6.0	Glenside	Ardsley	

	distance
1383031	5.474109
694135	1.496937
1339992	0.458662
1085449	7.155336
1459200	1.410460

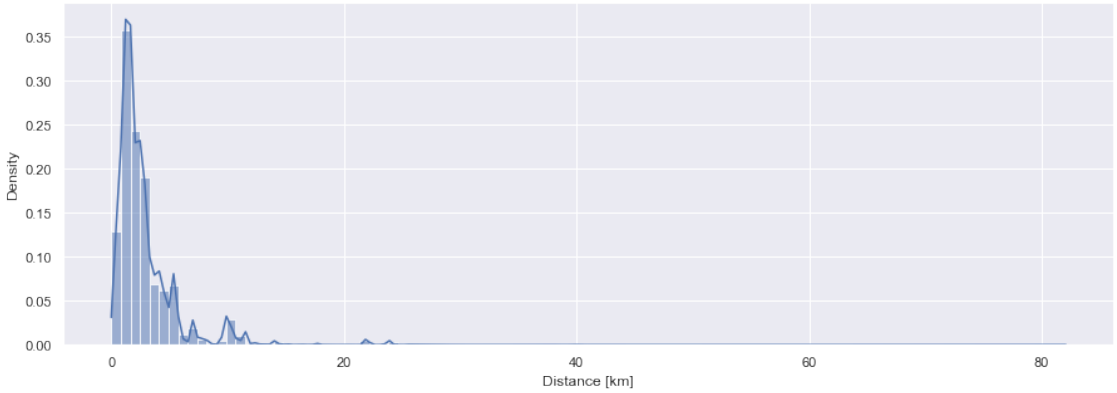
We will now visualize an example of a train run. We will plot the locations of all the train stations (in gray) with the stations of the example train run colored by the order they occur.

[ ]:
```
vis.plot_train_run(df_train_view=df_train_view, uni_coords=uni_coords,
                   uni_runs=uni_runs, df_distance=df_distance,␣
 ↪df_coords=df_coords)
```



If we plot the distribution of distances between consecutive train stations we can see that most inter-station distances are shorter than 10km.

[ ]:
```
vis.plot_distance_distribution(df=df_coords)
```



1.8.6 Cumulative distance along a train run

This feature provides the cumulative distance that a train has traveled along a given train run. It is likely correlated with the rank of a given station on a given train run. Nevertheless, it might convey different information and be useful, depending on the specific questions we will try to address, when applying this feature engineering analysis to a machine learning task.

In the table below we see the information regarding distance and cumulative distance for an example train run.

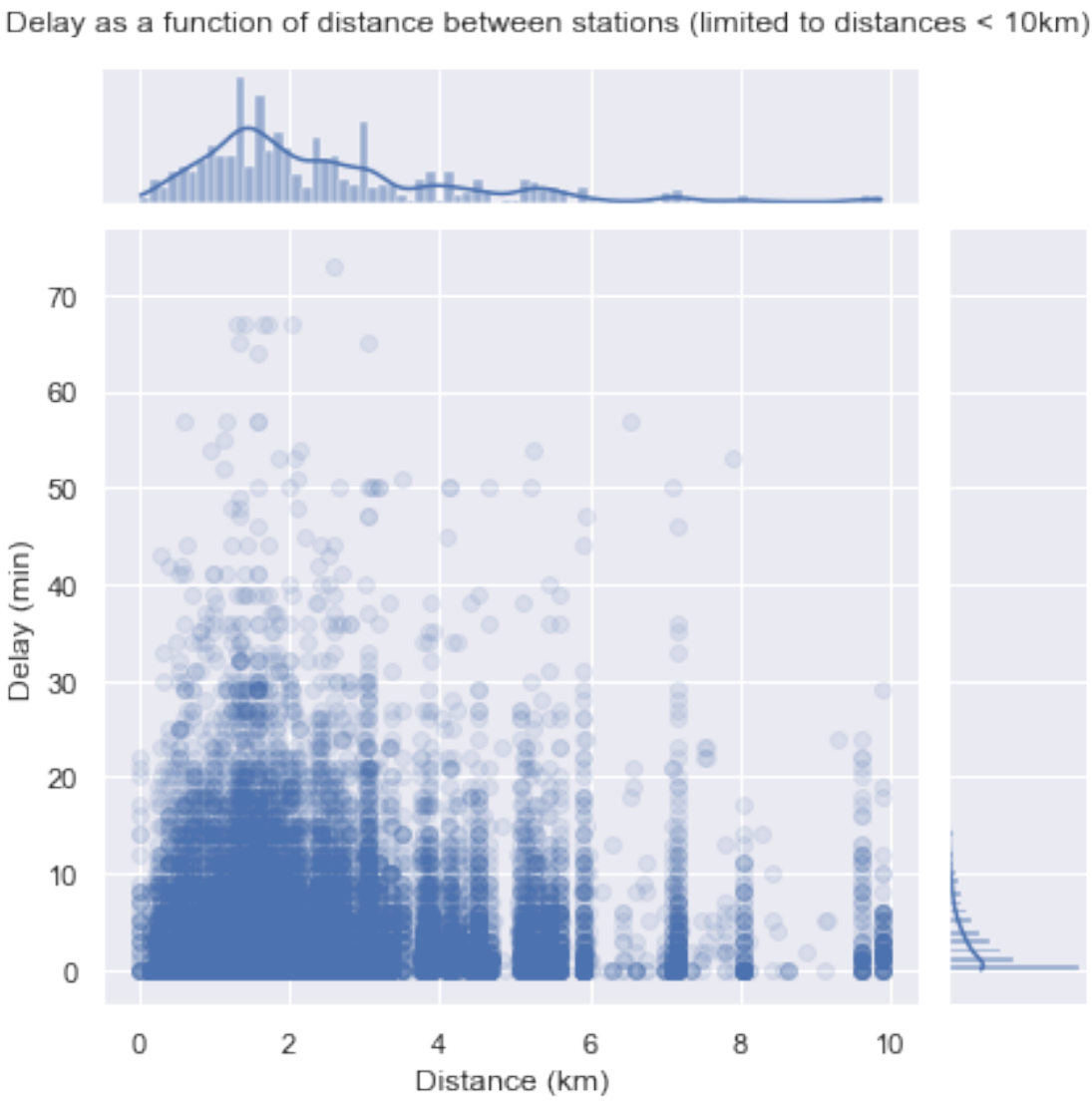
[ ]:
```
df_coords, random_train_run = sp.
  ↪calculate_cumulative_distance_along_train_run(df=df_coords,␣
  ↪uni_runs=uni_runs)
random_train_run
```

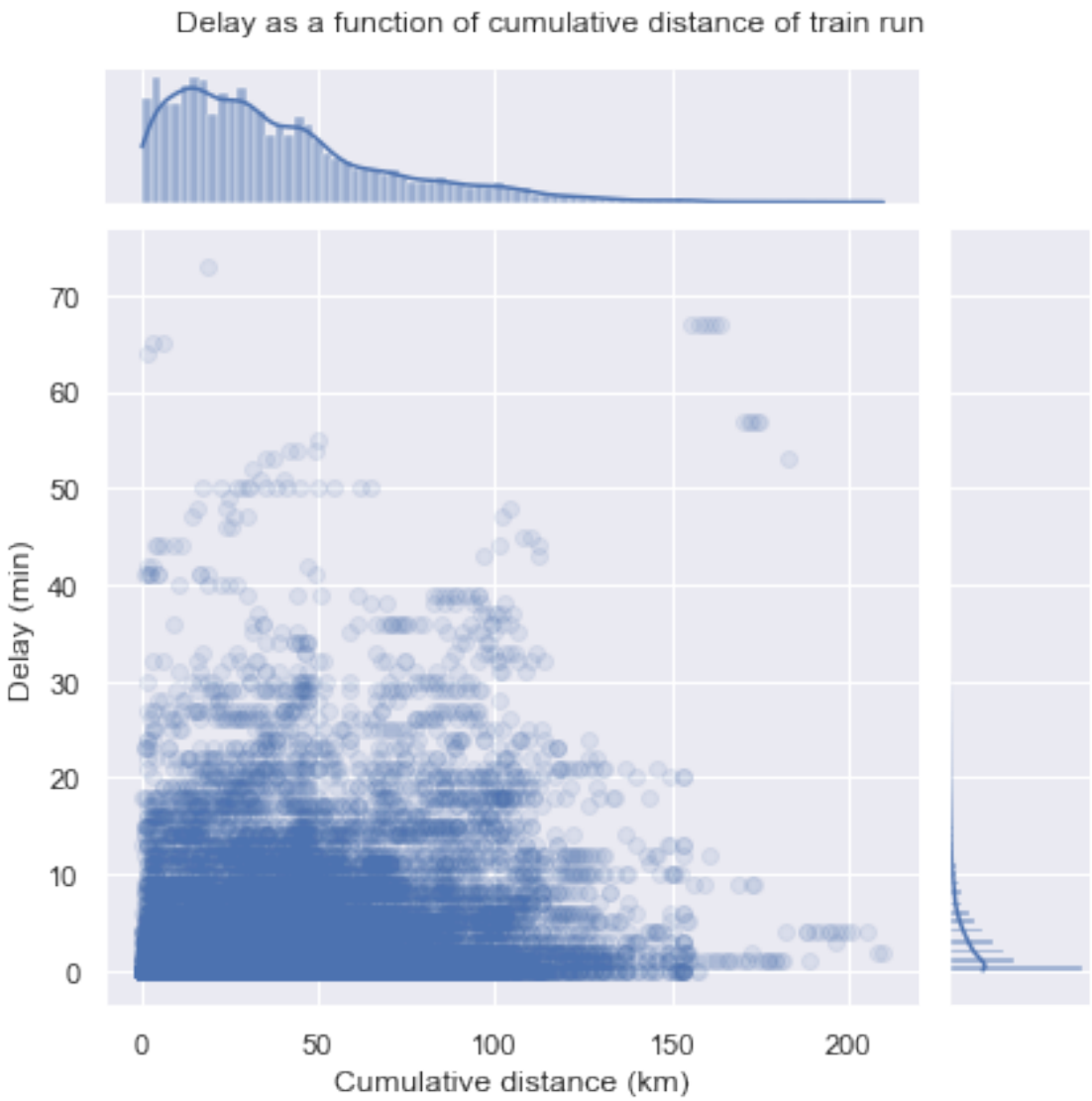
[ ]:

	train_id	rank	next_station	delay	upcoming_station	distance	cum_distance
0	426	2.0	University City	22	Eastwick Station	5.474109	5.474109
1	426	3.0	30th Street Station	23	University City	1.912928	7.387037
2	426	4.0	Suburban Station	25	30th Street Station	1.567499	8.954536
3	426	5.0	Jefferson Station	25	Suburban Station	1.344006	10.298542
4	426	6.0	Temple U	24	Jefferson Station	3.042680	13.341222
5	426	7.0	Wayne Jct	22	Temple U	2.582294	15.923516
6	426	8.0	Fern Rock TC	23	Wayne Jct	2.506551	18.430066
7	426	9.0	Melrose Park	23	Fern Rock TC	2.424309	20.854375
8	426	10.0	Elkins Park	22	Melrose Park	3.024633	23.879008
9	426	11.0	Jenkintown-Wyncote	21	Elkins Park	0.524916	24.403924

In the scatter plot below we plot the delay on both the distance between two stops and the cumulative distances within the train runs.

[ ]:
```
vis.delay_jointplot(df=df_coords, uni_runs=uni_runs)
```


Delay as a function of distance between stations (limited to distances < 10km)
Delay (min)
0
10
20
30
40
50
60
70
0
2
4
6
8
10
Distance (km)



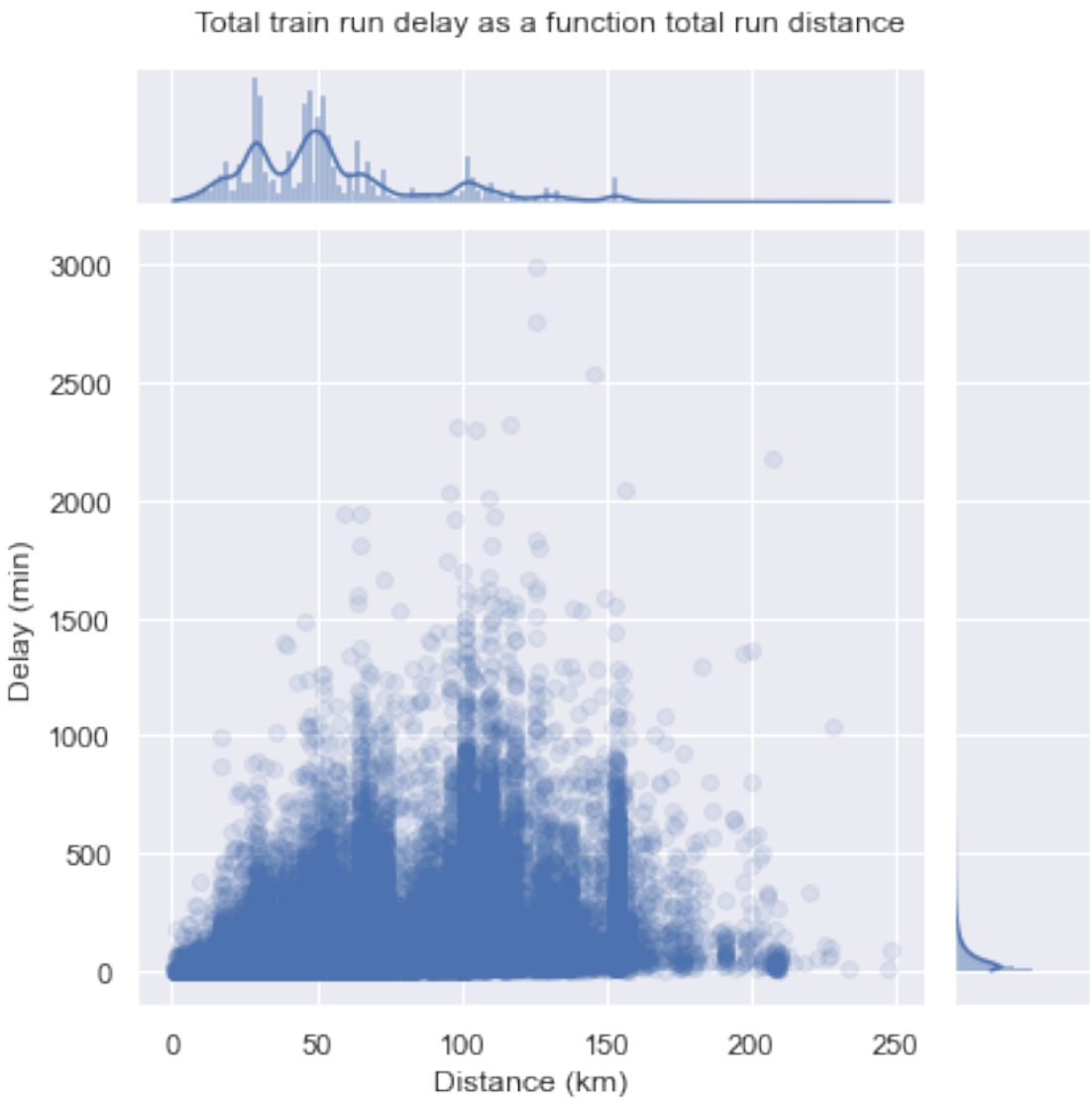
As we can visually see, in both plots there is no correlation with delay.

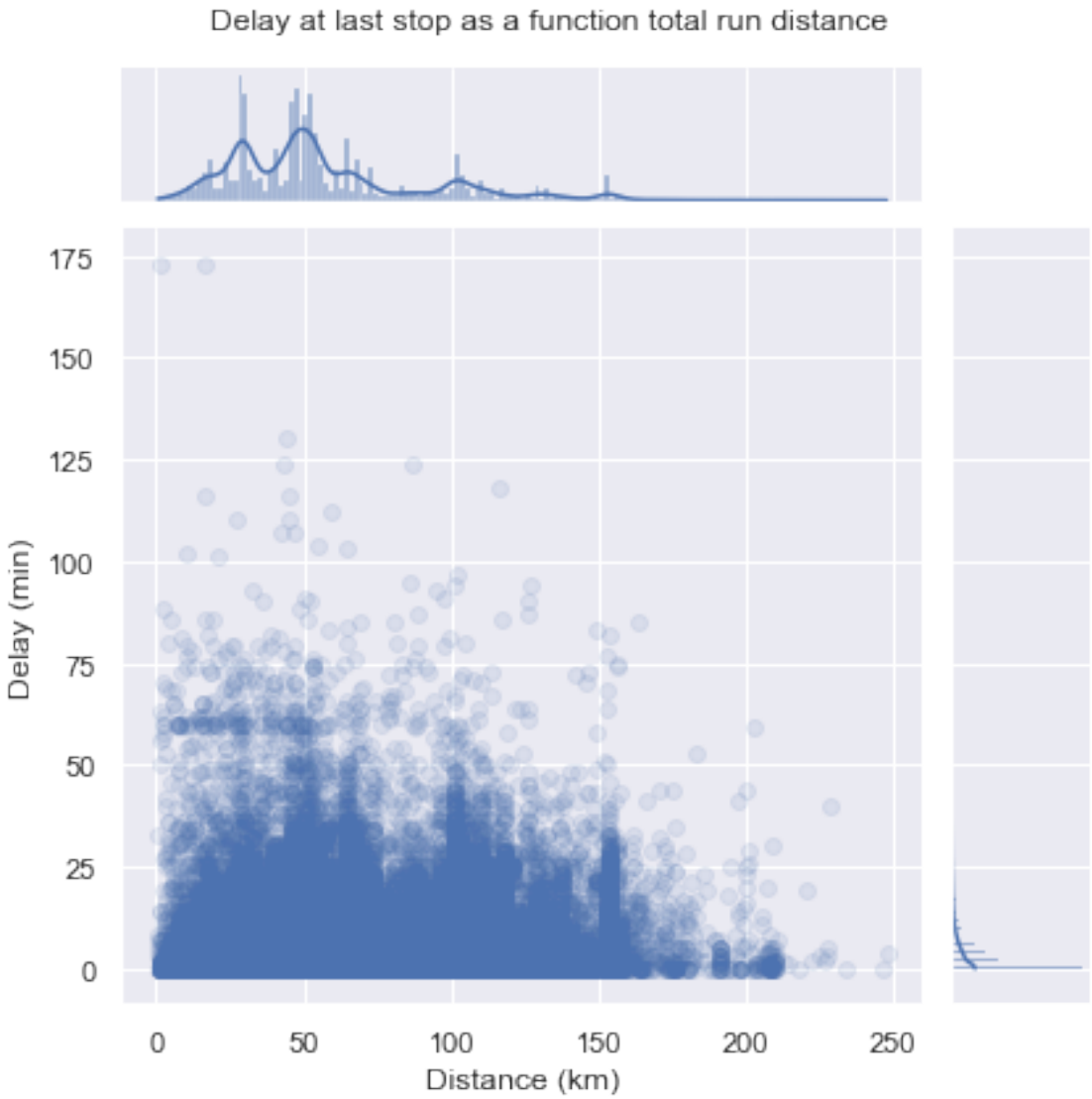
1.8.7 Total distance of a train run

The total distance traveled by the train is probably closely linked to the total delay or the delay at the last stop of a train run. Here we will look at train run total distance and delay.

In the first scatter plot below we plot the total delay as a function to the total train run distance. Additionally, we plot the delay at the last stop as a function to the total train run distance.

[ ]:
```
vis.total_train_run_distance_joint_plot(df=df_coords)
```


Total train run delay as a function total run distance
Delay (min)
3000
2500
2000
1500
1000
500
0
0
50
100
150
200
250
Distance (km)



Also in this case we cannot see a clear relationship between the train run distance and the total delay throughout the run, neither with the delay at the last stop.

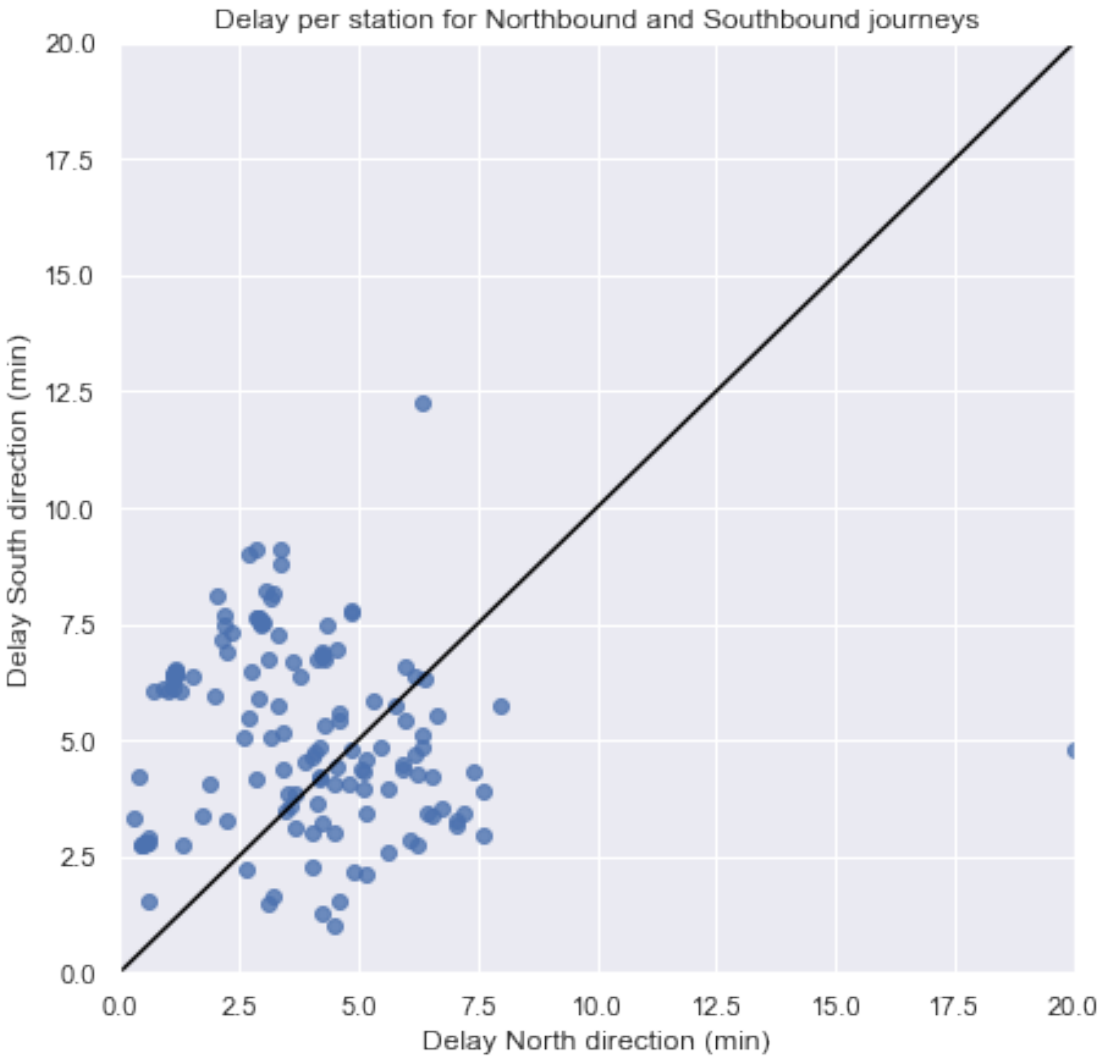
[ ]:
```
df_otp = sp.add_cum_delay(df=df_otp)
df_otp = sp.add_distance(df_otp=df_otp, df_coords=df_coords)
```

1.8.8 Northbound vs Southbound travels

This feature identifies stations which are typically problematic, in the sense that there are always delays. We can do this by checking their average delay.

We will first compare the delays of trains operating in one or the other direction.

[ ]:
```
vis.plot_northbound_southbound_dely_per_station(df=df_otp)
```

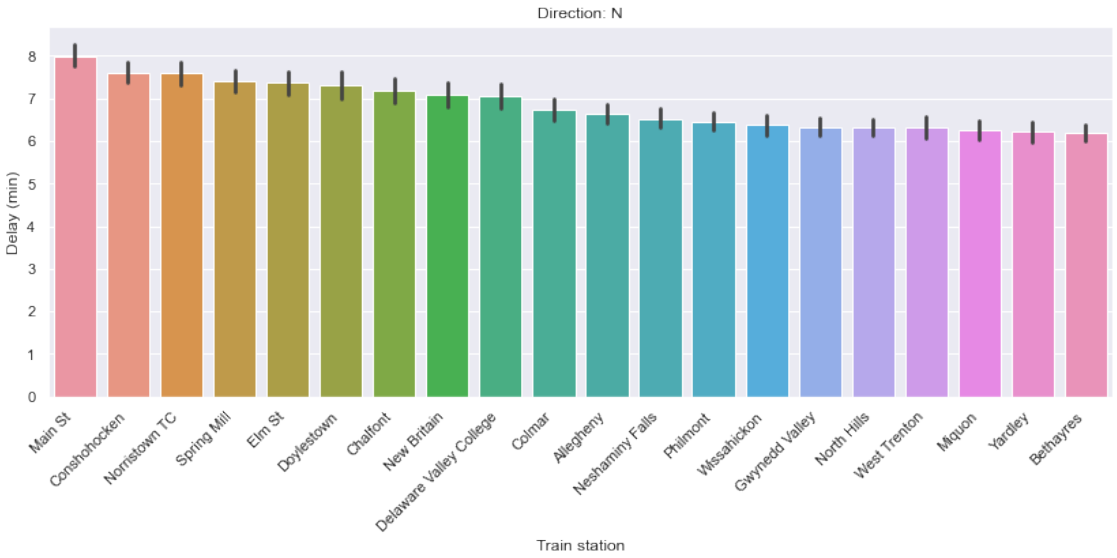


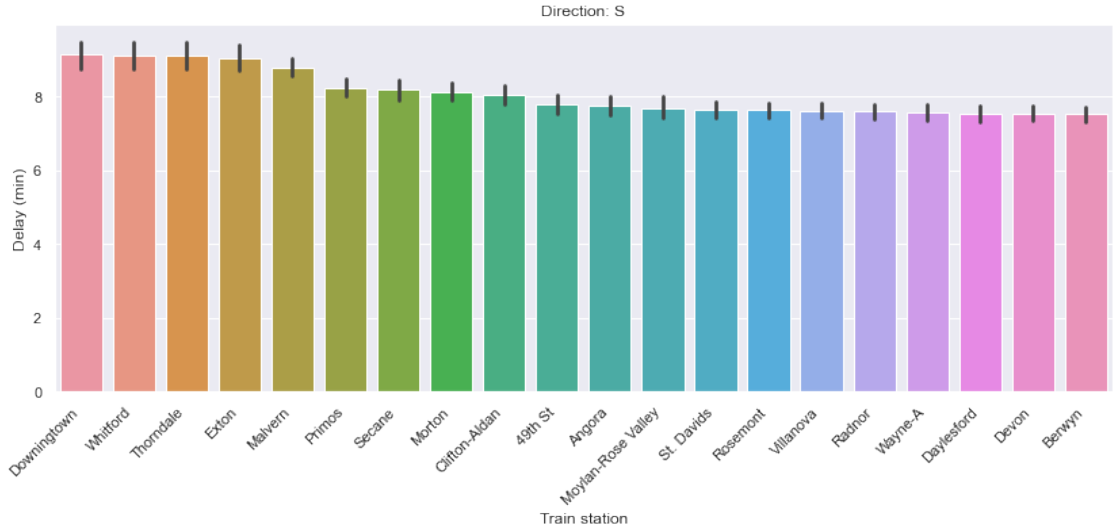
The asymmetry in the scatterplot (even if not that striking) suggests that delay is dependent on direction (as could be expected, since the distance from origin and stop number will be different according to the direction), so we will label long-duration stations per direction.

1.8.9 Stations with big delay

Some stations might be more prone to delays. This might be caused e.g. by many tracks converging to a smaller number of platforms of the station or due to long-term works on a station. We will, thus, rank the average delay for each station and label the top and bottom ones.

[ ]:
```
vis.plot_stations_with_long_delays(df=df_otp)
```





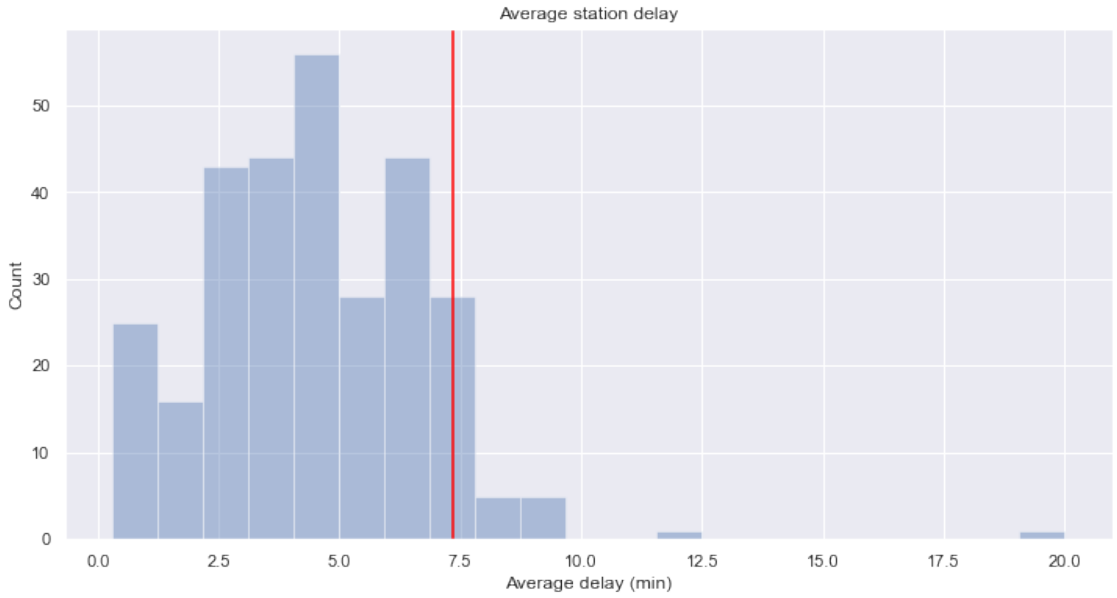
We can label those stations with an average delay above a certain threshold. In this case, we will use the 90th percentile as a threshold. Below you can see the histogram of the average delay per station.

[ ]:
```
vis.plot_average_delay(df=df_otp)

sp.print_delays_overview(df=df_otp)
df_otp = sp.label_long_delay_stations(df=df_otp)
```

```
Stations with an average delay above 7.4 minutes will be labeled:
        Direction N: Churchmans Crossing, Conshohocken, Elm St, Main St,
```

```
Norristown TC, Spring Mill
        Direction S: 49th St, Angora, Berwyn, Clifton-Aldan, Daylesford, Devon,
Downingtown, Exton, Fernwood, Malvern, Morton, Moylan-Rose Valley, Primos,
Radnor, Rosemont, Secane, St. Davids, Strafford, Swarthmore, Thorndale,
Villanova, Wayne-A, West Trenton, Whitford
```



The red line on the histogram above indicates the threshold (90th percentile) that was used to label stations with considerably long delays. Most of those stations are in direction S (southbound).

1.8.10 Delay over the last 7 days

Sometimes, uncontrollable factors can cause delays in trains for a couple of days (e.g. works on the train tracks, long duration strikes, etc.). We can calculate what was the delay over the last 7 days of each date to add that as a feature to predict delays.

[ ]:
```
df_otp, last_week_delay = sp.calculate_longest_delays_over_past_week(df=df_otp)
print('10 cases with the longest delay over the past 7 days')
last_week_delay.sort_values('last_week_delay', ascending=False).head(10)
```

```
10 cases with the longest delay over the past 7 days
```

[ ]:

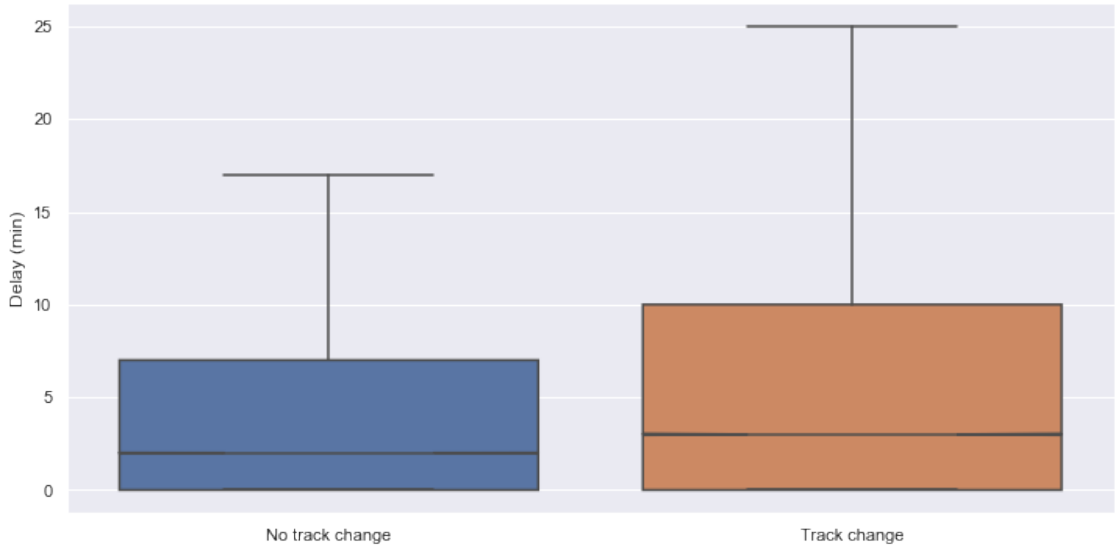
	train_id	next_station	last_week_delay	date
6278	6803	30th Street Station	116.0	2016-04-01
7028	852	30th Street Station	88.0	2016-11-02
1716	393	Yardley	86.0	2016-11-02
315	1556	Jefferson Station	85.0	2016-09-17
128	1524	Berwyn	85.0	2016-07-19
129	1524	Daylesford	85.0	2016-07-19

```
130       1524                Devon          85.0 2016-07-19
134       1524              Malvern          85.0 2016-07-19
138       1524                Paoli          85.0 2016-07-19
139       1524               Radnor          85.0 2016-07-19
```

1.8.11 Track changes

Track changes can be the cause of delays. To address that, we will compare delays in cases when there was a track change with cases when there was no track changes.

[ ]:
```
df_track_changes = sp.get_track_changes(df=df_train_view)
vis.plot_track_changes(df=df_track_changes)
```



As we can see, delays are higher when there were track changes. A feature we can add to our model is the track change frequency per station per day, defined as:

$$Trackchangefrequency_d = \frac{\sum_{tc\_tr}}{\sum_{tr}}$$

where d is a given date, tc_tr is a train arrival with a track change and tr is a train.

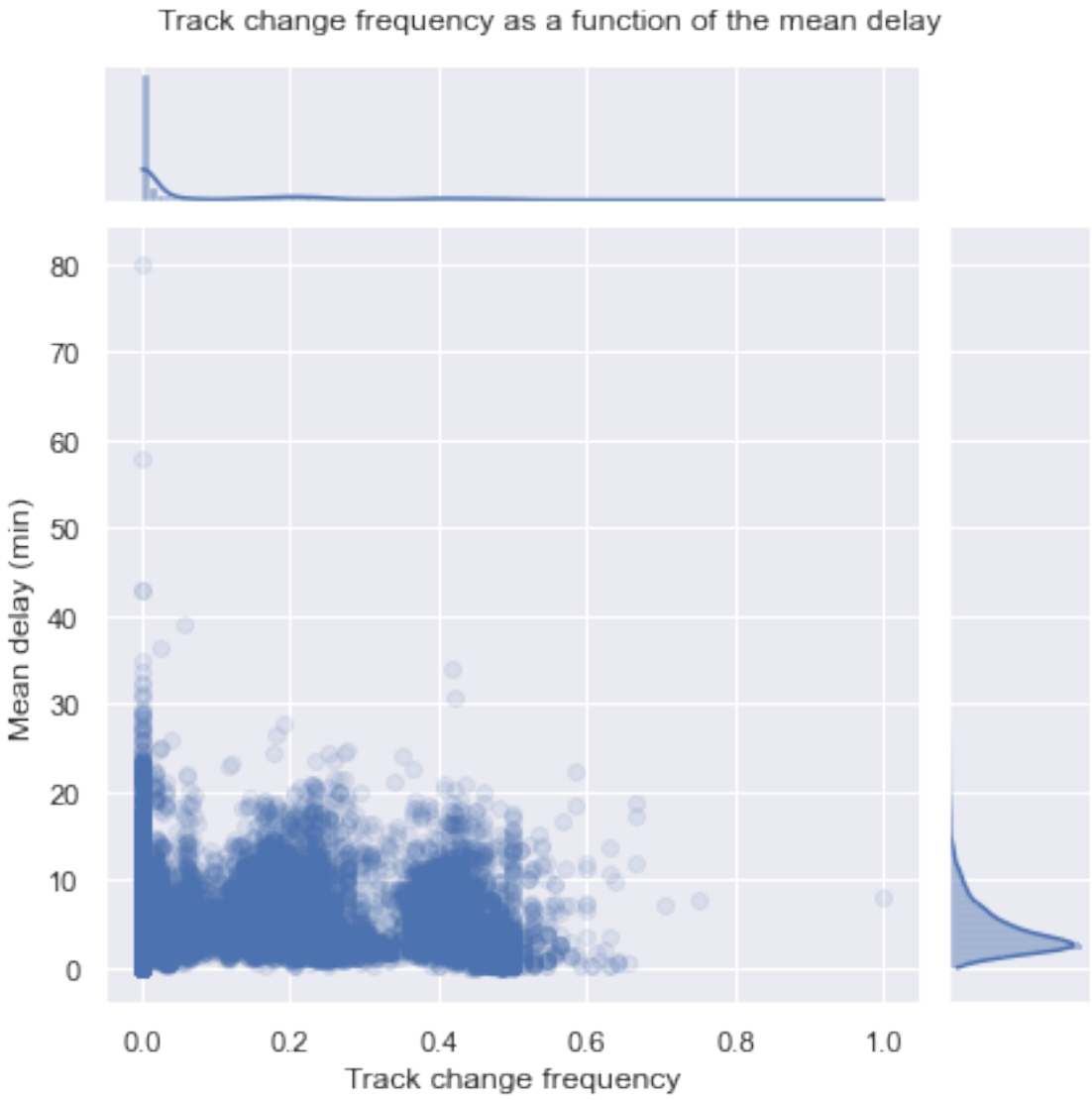
[ ]:
```
track_change_frequency_date = sp.
  ↪calculate_track_changes_frequency(df=df_track_changes)
track_change_frequency = sp.
  ↪calculate_aggregated_track_changes(df=track_change_frequency_date)
```

We can compare how delay changes as a function of track change frequency:

[ ]:

```
vis.
  ↪plot_track_change_frequency_with_delays(df_track_change_frequency=track_change_frequency_da
  ↪df_otp=df_otp)
```

```
Pearson's coefficient = 0.16 with p-value = 5.70e-178.
```

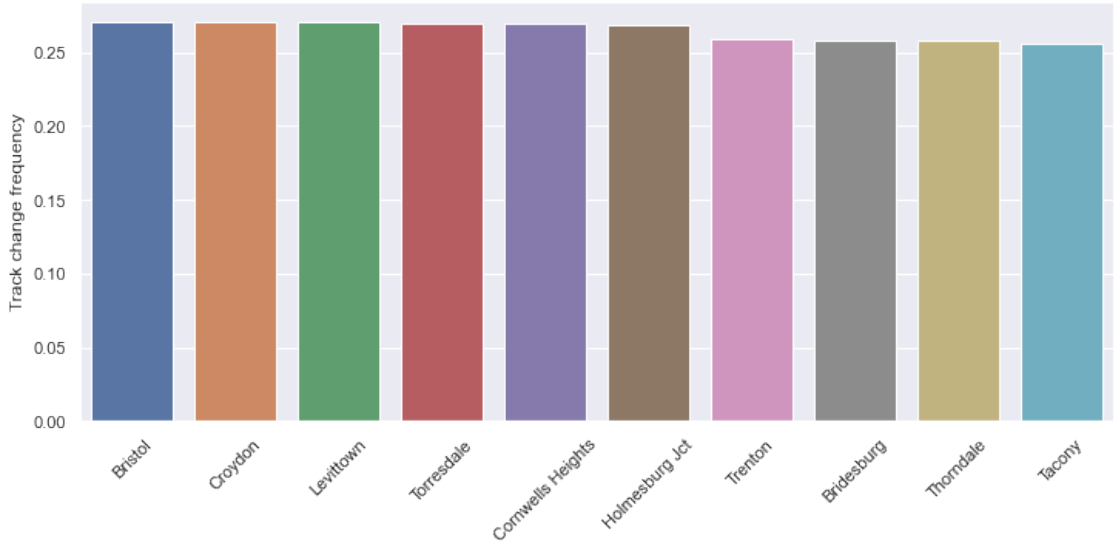


Also in this case, the scatter plot above do not show a clear relationship between delays and track changes.

In the plot below we see the track change frequency of the 10 stations with the highest track change frequency.

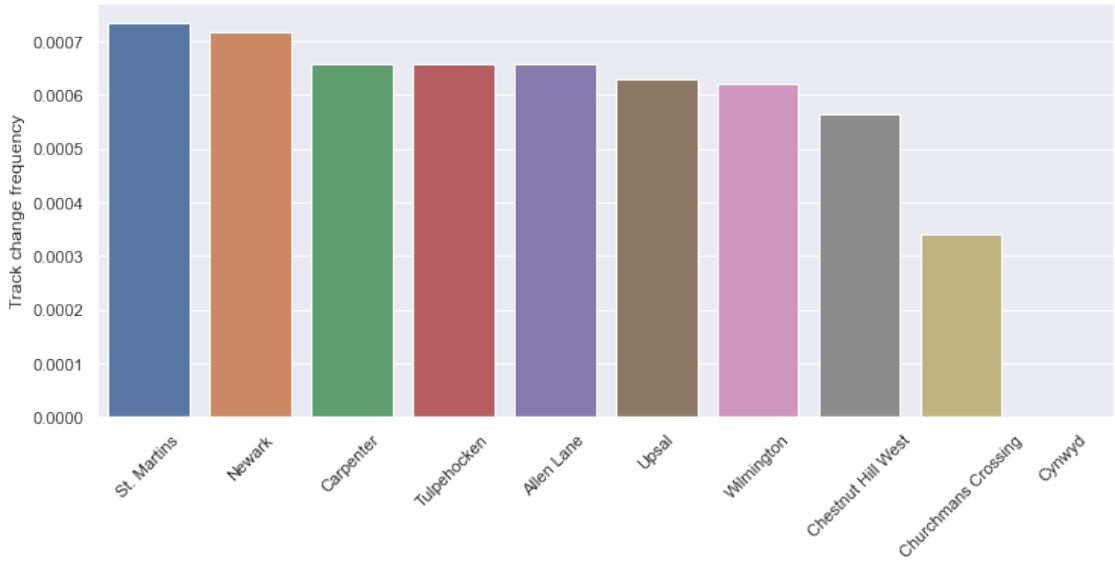
[ ]:
```
# get unique combinations of station/coordinates and of train_id/dates
uni_coords = df_train_view[['next_station', 'lon', 'lat']].drop_duplicates()
uni_runs = df_otp[['train_id', 'date']].drop_duplicates()

vis.
 ↪plot_highest_track_change_frequency(track_change_frequency=track_change_frequency)
```



In the plot below you can see the same figure, but now for the 10 stations with the lowest change frequency. Note that the change frequency for station Cynwyd is zero, which is enforced by the physical limitation of only a single track.

[ ]:
```
vis.plot_lowest_track_change_frequency(track_change_frequency)
```



[ ]:
```
# add track change frequency as a feature
df_otp = df_otp.merge(track_change_frequency, on='next_station')
```

1.9 Conclusion

In this Starter Kit we demonstrated the workflow on how to construct advanced features that are fundamental for answering company business questions such as SEPTA' performance measures. We demonstrated the need of data preparation steps before calculating features as it might simplify the work later. In addition, a wide range of complementary features are constructed, while the impact on train delay is briefly examined using statistical plots.