elucidata_demonstrator_3_3

November 4, 2022

[48]:
```
import support as sp
import visualization as vis

%load_ext autoreload
%autoreload 2
```

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

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 treshold 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 is 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, transoformation 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 presented 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 answer a question such as "can we predict delays at a given train station?"

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. How well are they doing? Is it even a meaningful measure?

In order to be able to answer such questions, we will try to design 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.

[38]:
```
df_otp = sp.load_otp_data()
df_otp.head(10)
```

```
2022-10-19 14:27:50,855 [INFO ]  Using file: /Users/mdhn/Documents/starter-
kits/repository/elucidatalab.starterkits/data/SK_3_3/otp.csv
```

[38]:
```
  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:

[39]:
```
df_train_view = sp.load_train_view_data()
df_train_view.head(3)
```

```
2022-10-19 14:28:05,493 [INFO ]  Using file: /Users/mdhn/Documents/starter-
kits/repository/elucidatalab.starterkits/data/SK_3_3/trainView.csv
```

[39]:
```
  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.

[40]:
```
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` may be 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 the case in this dataset, we will count how many times a `train_id` passes through a given station on a given day in the `train_run` dataset:

[41]:
```
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.

[42]:
```
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`.

[43]:
```
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`.

[44]:
```
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.

[45]:
```
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.

[46]:
```
sp.get_otp_overview(df=df_otp)
```

[46]:

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 trians being delayed, the statement that 91% of all trains are on time (definition: no more than 5min59s delay) is questionable.

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.

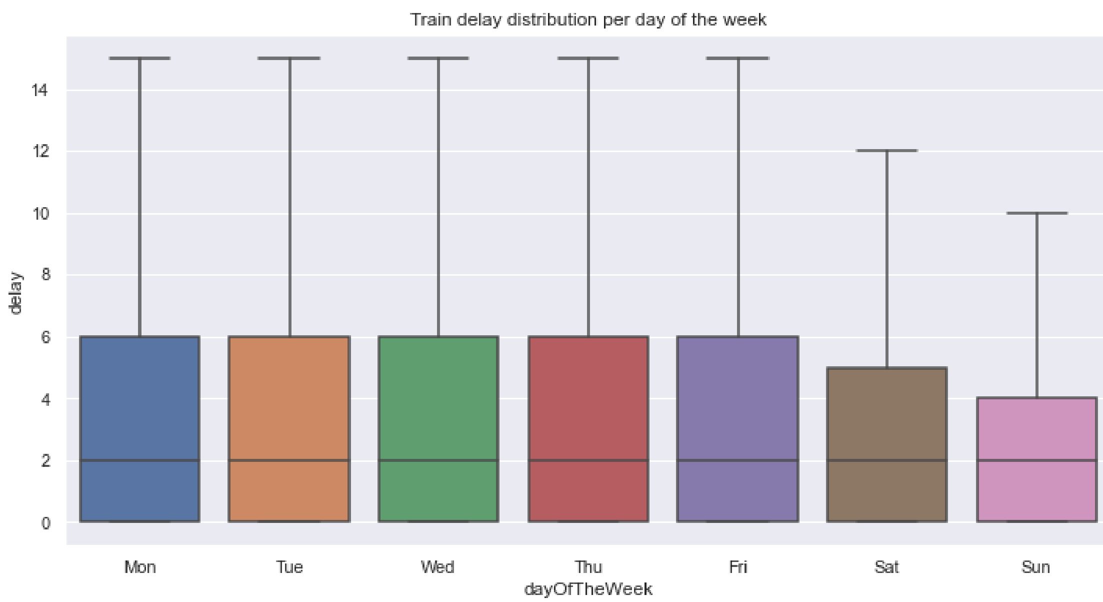
The features that we will extract are: - Day of the week - Month of the year - Rush hours - Rank the different stops in a train run - Distance between stations - Cumulative distance along a train run - Total distance of a train run - Northbound vs Southbound travels - Stations with big delay - Delay 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.

[47]:
```
vis.plot_train_delays(df=df_otp)
```



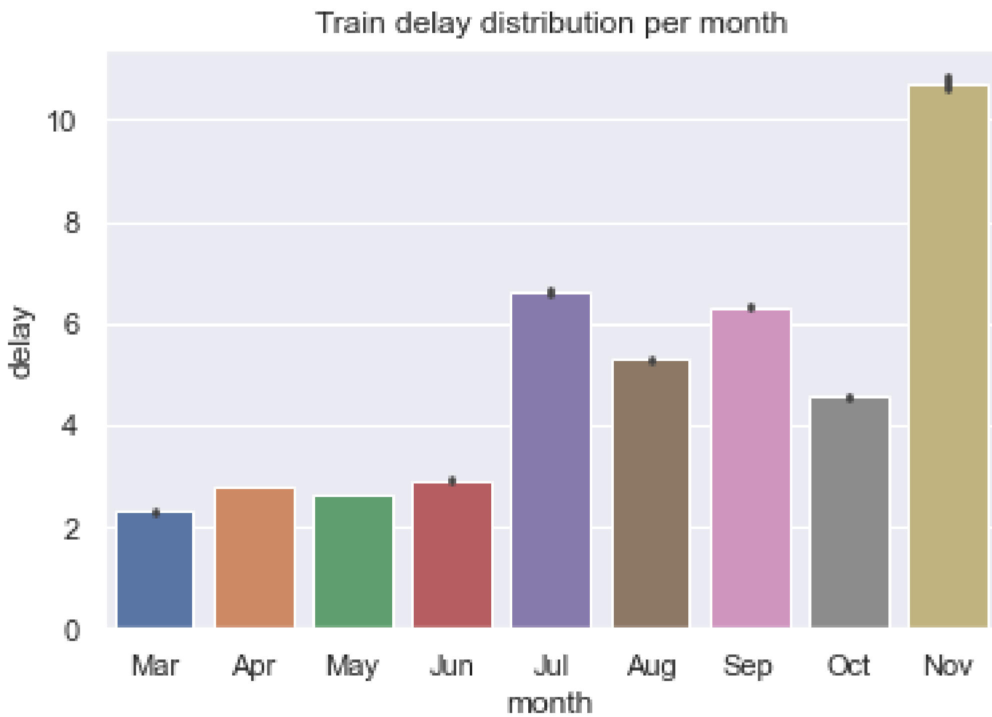
As we can see, the difference in delays in weekdays (Mon-Fri) is only minimal. We will thus categorize the day of the week into weekdays, saturdays and sundays.

[12]:
```
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 month of the year.

[13]:
```
vis.plot_train_delays(df=df_otp, kind='month')
```



Data only runs from March to November 2016. Given the large variability observed between months, we will keep this feature, rather than splitting 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.

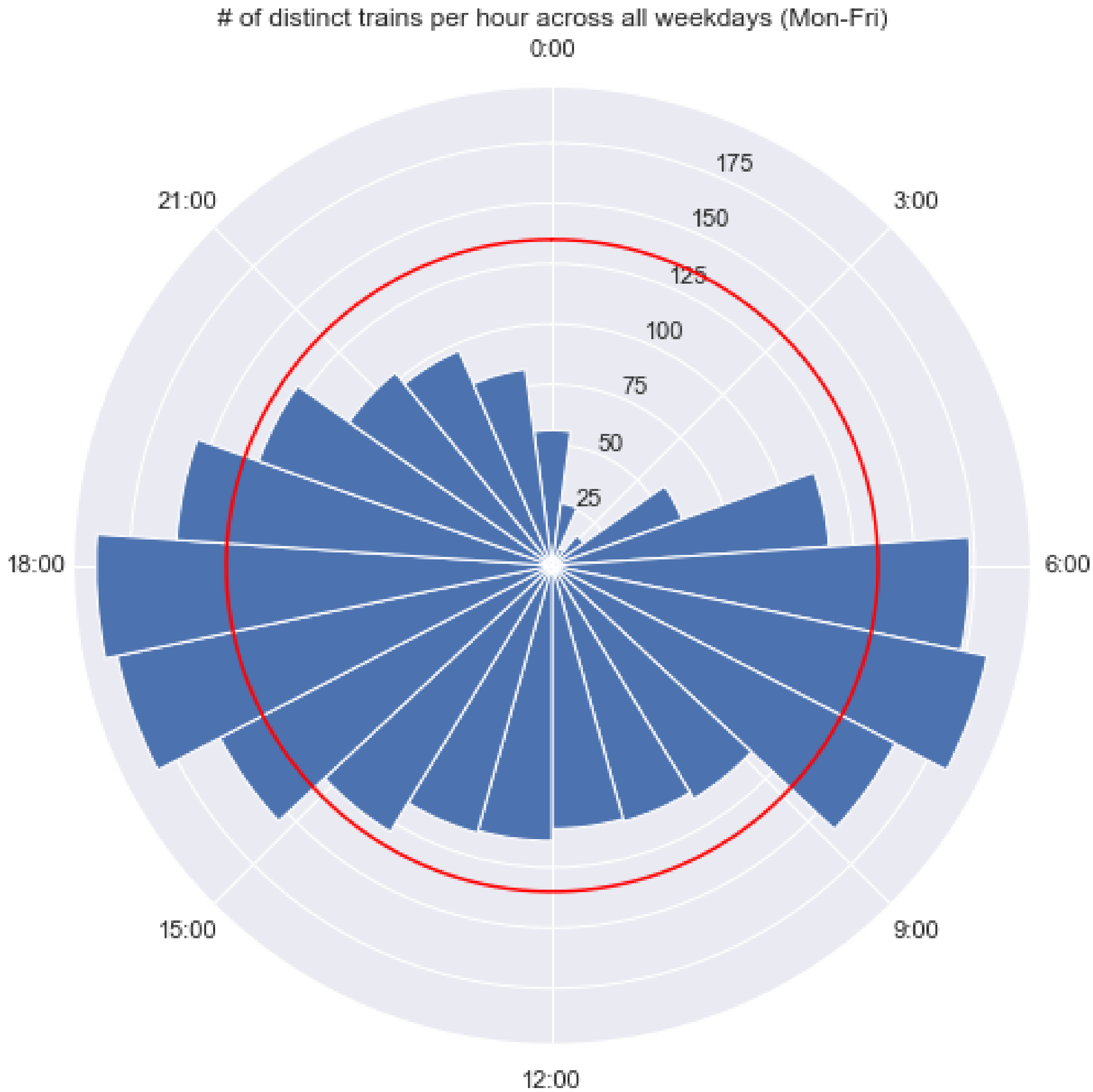
[14]:
```
df_otp = sp.extract_time_features(df=df_otp)
df_otp[['train_id', 'origin', 'date', 'hour', 'isWeekend']].sample(5)
```

[14]:

	train_id	origin	date	hour	isWeekend
1532271	533	Doylestown	2016-10-10	14	False
1116985	3535	Trent	2016-08-14	12	True
270256	584	Thorndale	2016-04-22	20	False
900279	3524	Swarthmore	2016-07-06	14	False
362051	3596	Elwyn Station	2016-05-03	16	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.

[15]:
```
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 line in the plot above.

[16]:
```
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.
```

[16]:

	train_id	origin	date	hour	dayOfTheWeek	isRushHour
894604	846	Chestnut Hill West	2016-07-05	17	Tue	True
548492	7390	Trenton	2016-05-24	19	Tue	True

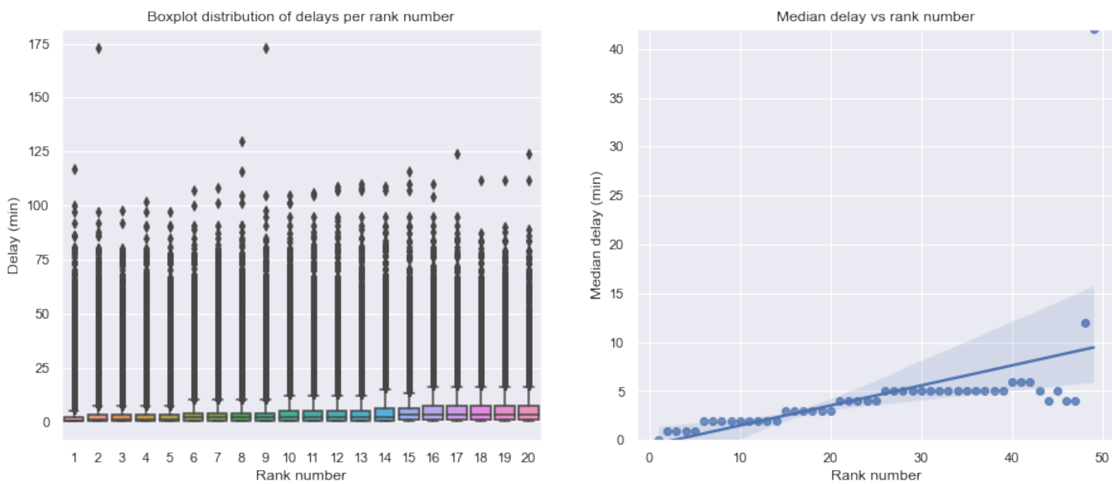
1.8.4 Rank the different stops in a train run

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.

[17]:
```
df_otp = sp.calculate_rank_stop(df=df_otp)
```

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

[18]:
```
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.

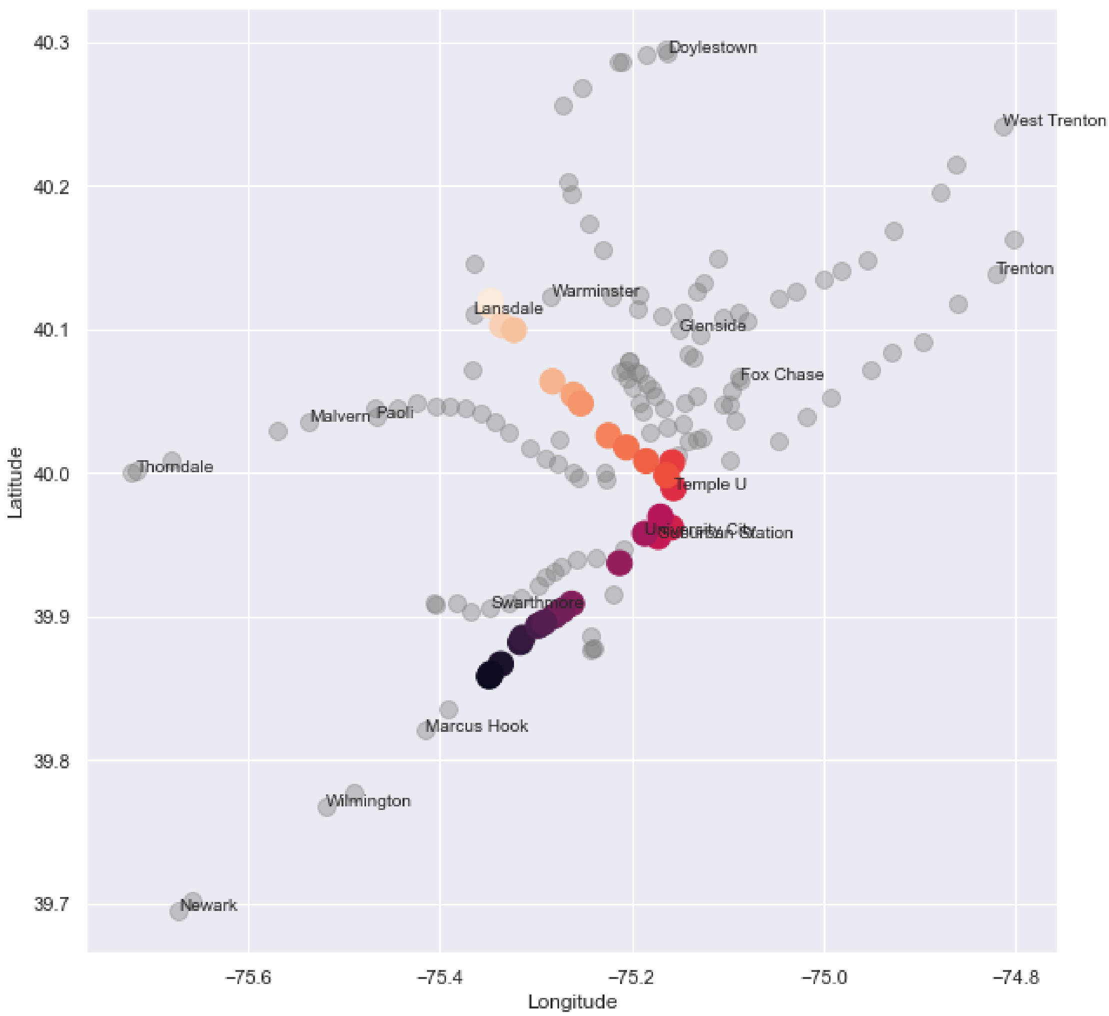
[19]:
```
uni_coords, uni_runs, df_distance, df_coords = sp.
 ↪calculate_distance_between_stations(df_otp=df_otp,␣
 ↪df_train_view=df_train_view)
df_coords[['train_id', 'date', 'rank', 'next_station', 'upcoming_station',␣
 ↪'distance']].sample(5)
```

[19]:
```
        train_id        date  rank        next_station  upcoming_station  \
1166828      552 2016-10-05  29.0             Oreland       North Hills
440093       470 2016-05-24   4.0     University City  Eastwick Station
1214037      405 2016-07-26  18.0  Airport Terminal A  Eastwick Station
872941       462 2016-09-26  11.0        Melrose Park      Fern Rock TC
1325006     9547 2016-05-25   5.0           Villanova          Rosemont

         distance
1166828  2.643977
440093   5.474109
1214037  3.798487
872941   2.424309
1325006  1.560238
```

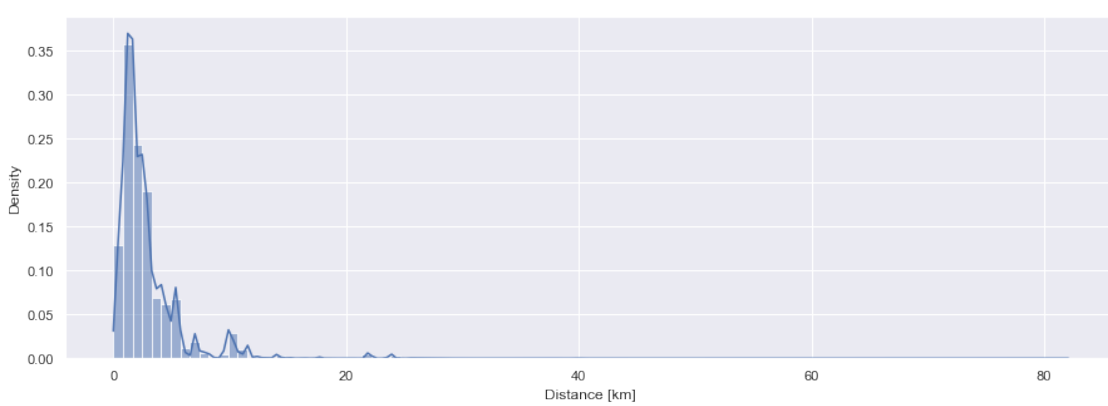
We will now visualize an example of a train run. We will plot the locations of all the train stations (in gray) with the stations of the example train run colored by the order they occur.

[20]:
```
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 trainstations we can see that that most inter-station distances are shorter than 10km.

[21]:
```
vis.plot_distance_distribution(df=df_coords)
```



1.8.6 Cumulative distance along a train run

This feature provides the cummulative 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.

[22]:
```
df_coords, random_train_run = sp.
    ↪calculate_cumulative_distance_along_train_run(df=df_coords,␣
    ↪uni_runs=uni_runs)
random_train_run
```

[22]:

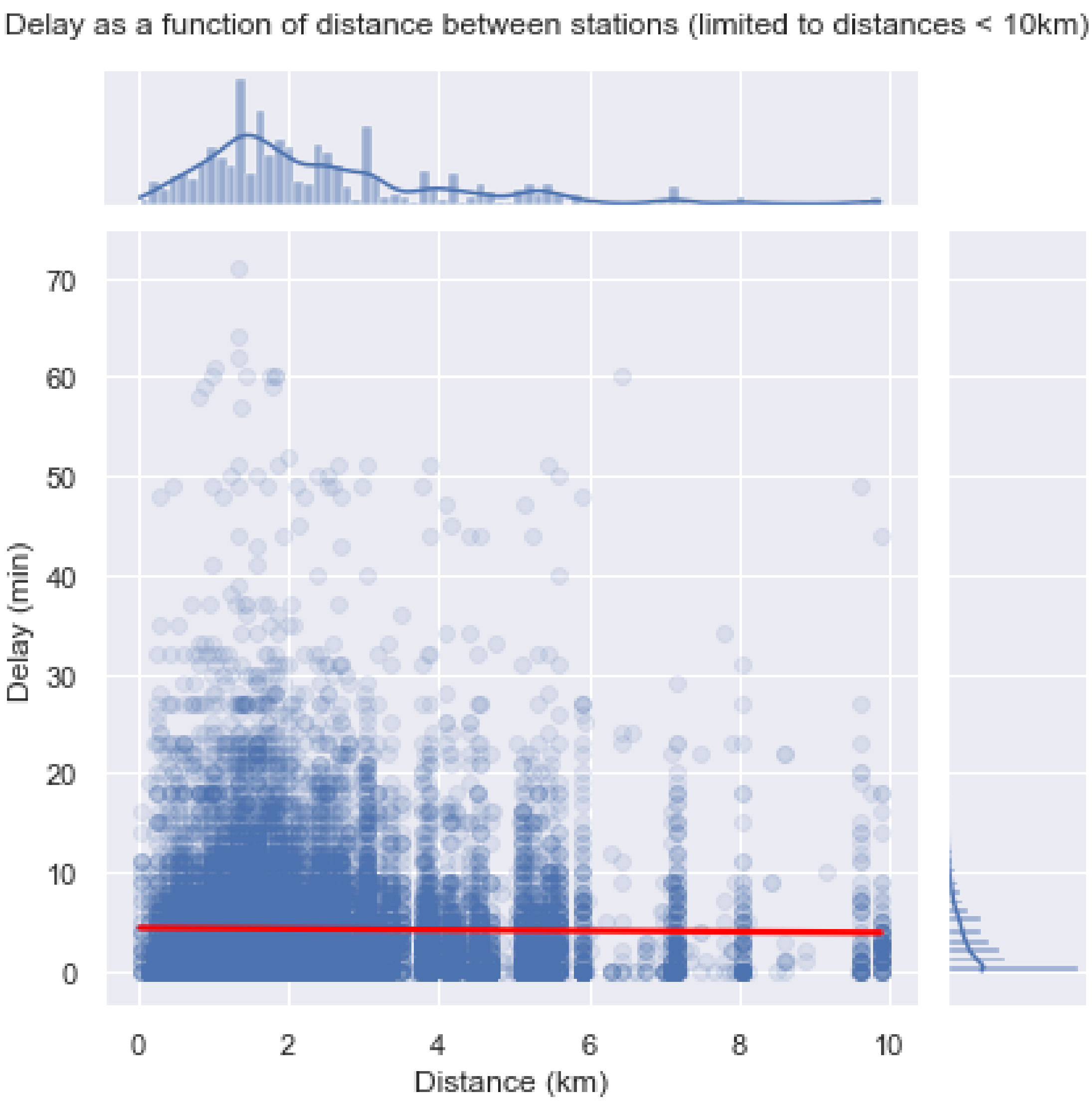
	train_id	rank	next_station	delay	upcoming_station	distance \
0	738	2.0	Wister	7	Wayne Jct	2.430122
1	738	3.0	Germantown	7	Wister	1.429974
2	738	4.0	Washington Lane	7	Germantown	1.233226
3	738	5.0	Stenton	5	Washington Lane	0.624521
4	738	6.0	Sedgwick	6	Stenton	0.572760
5	738	7.0	Mt Airy	7	Sedgwick	0.988380
6	738	8.0	Wyndmoor	9	Mt Airy	0.386719
7	738	9.0	Gravers	9	Wyndmoor	0.992990
8	738	10.0	Chestnut Hill East	10	Gravers	0.023907
9	738	11.0	Levittown	0	Chestnut Hill East	35.403964
10	738	12.0	Bristol	0	Levittown	7.101114
11	738	13.0	Croydon	0	Bristol	4.145670
12	738	14.0	Eddington	0	Croydon	2.914740
13	738	15.0	Cornwells Heights	0	Eddington	2.370927
14	738	16.0	Torresdale	0	Cornwells Heights	4.130555
15	738	17.0	Holmesburg Jct	0	Torresdale	2.646818
16	738	18.0	Tacony	0	Holmesburg Jct	3.109493
17	738	19.0	Bridesburg	0	Tacony	4.668687
18	738	20.0	North Philadelphia	0	Bridesburg	3.188490
19	738	21.0	30th Street Station	3	North Philadelphia	10.009572
20	738	22.0	Suburban Station	3	30th Street Station	1.567499
21	738	23.0	Jefferson Station	2	Suburban Station	1.344006

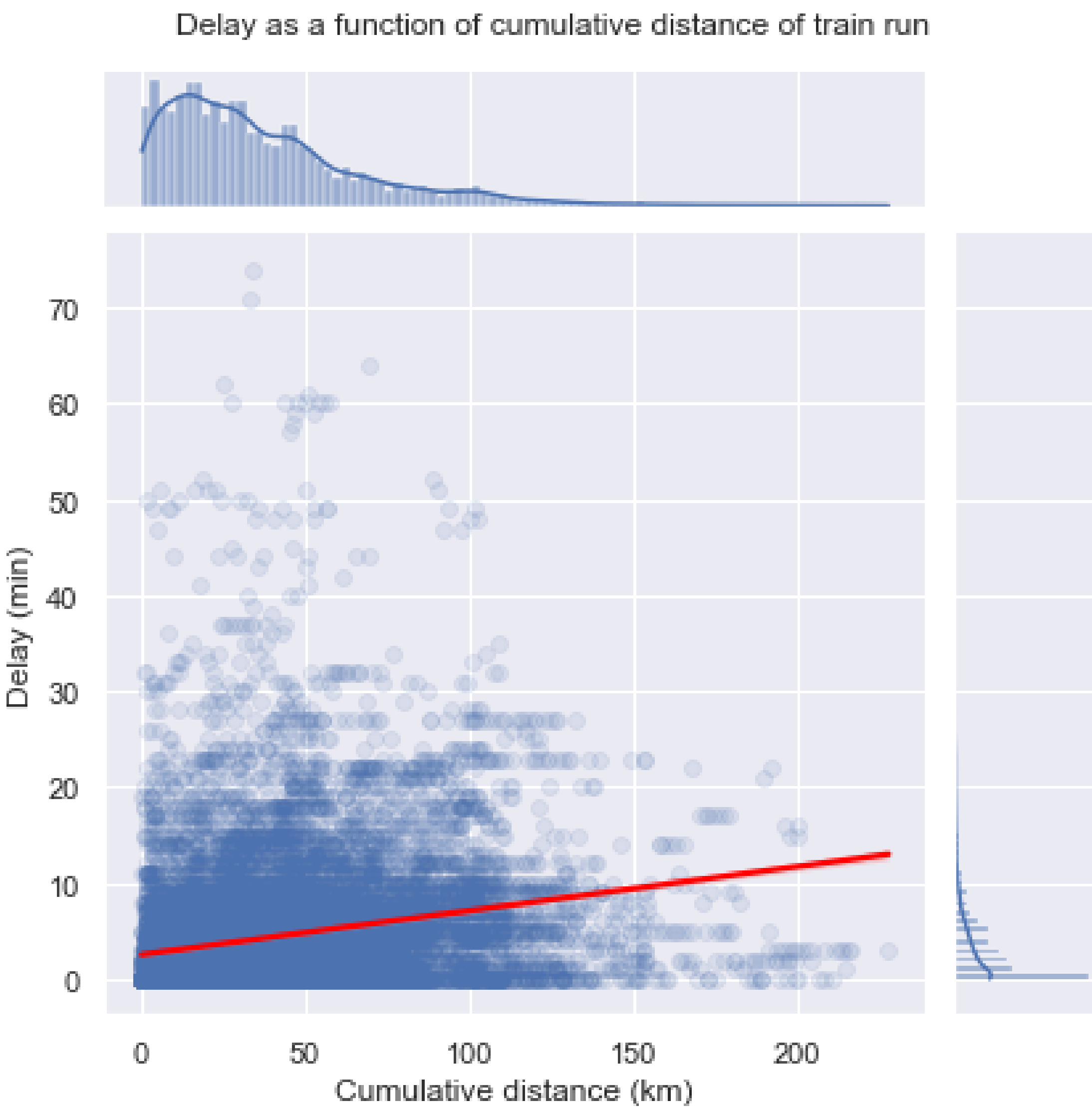
	cum_distance
0	2.430122
1	3.860096
2	5.093323
3	5.717844
4	6.290604

```
5        7.278984
6        7.665703
7        8.658693
8        8.682600
9       44.086564
10      51.187677
11      55.333347
12      58.248087
13      60.619014
14      64.749569
15      67.396387
16      70.505880
17      75.174566
18      78.363057
19      88.372629
20      89.940127
21      91.284134
```

In the scatter plot below we plot the delay on both the distance between two stops and the commulative distances within the train runs. To be able to detect a correlation between those features, we plot the regression line in red on top of the scatter plot.

[23]:
```
vis.delay_jointplot(df=df_coords, uni_runs=uni_runs)
```


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



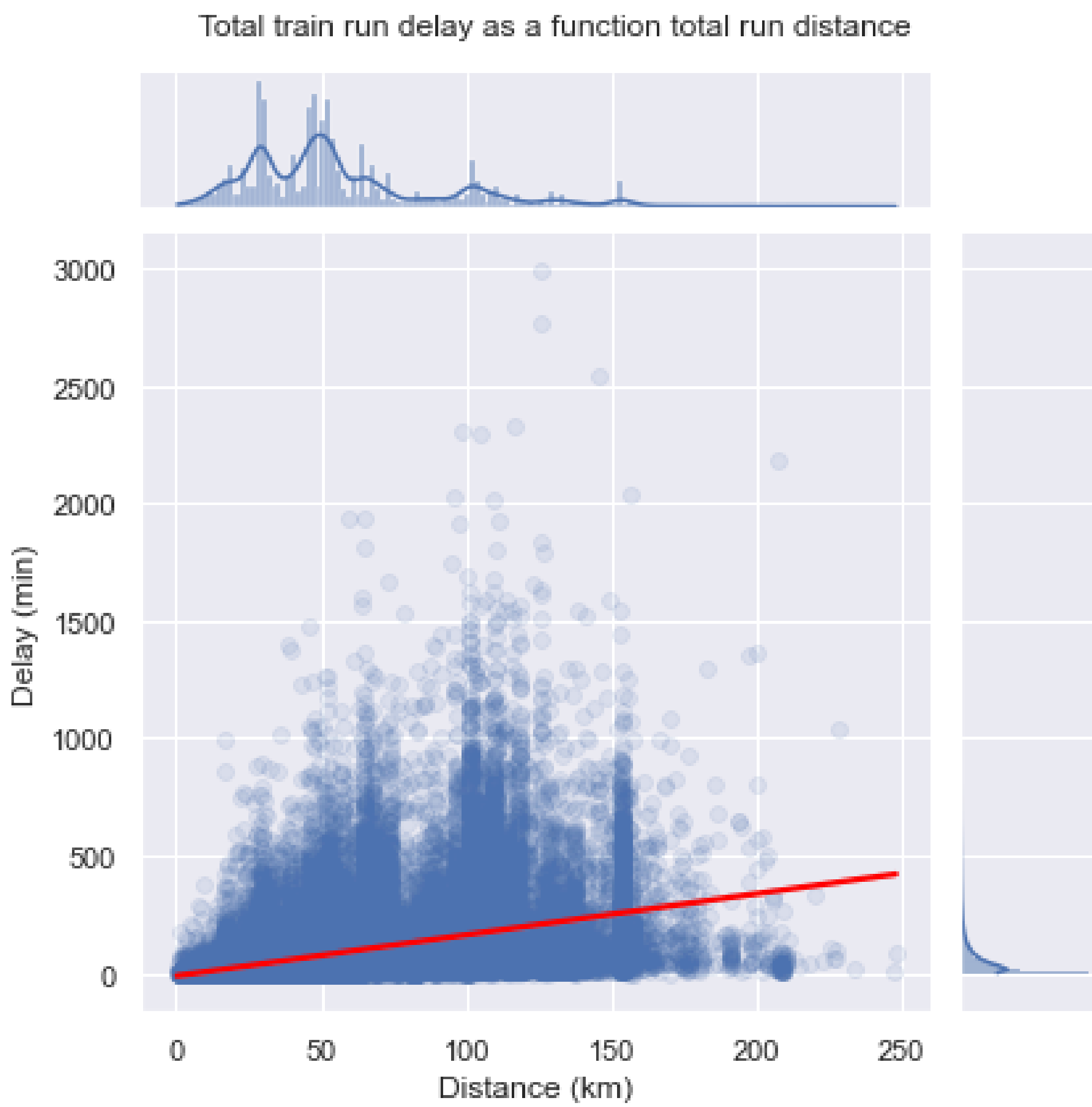
Interestingly, only cumulative distance appears to show a correlation with delay (indicated by the positive regression line). The latter is suggesting that a delay might not be caused by the distance between stations, but rather by delays incurred at stations themselves. However, the longer the train run, the longer will be the delay incurred (as we have already shown with stop rank).

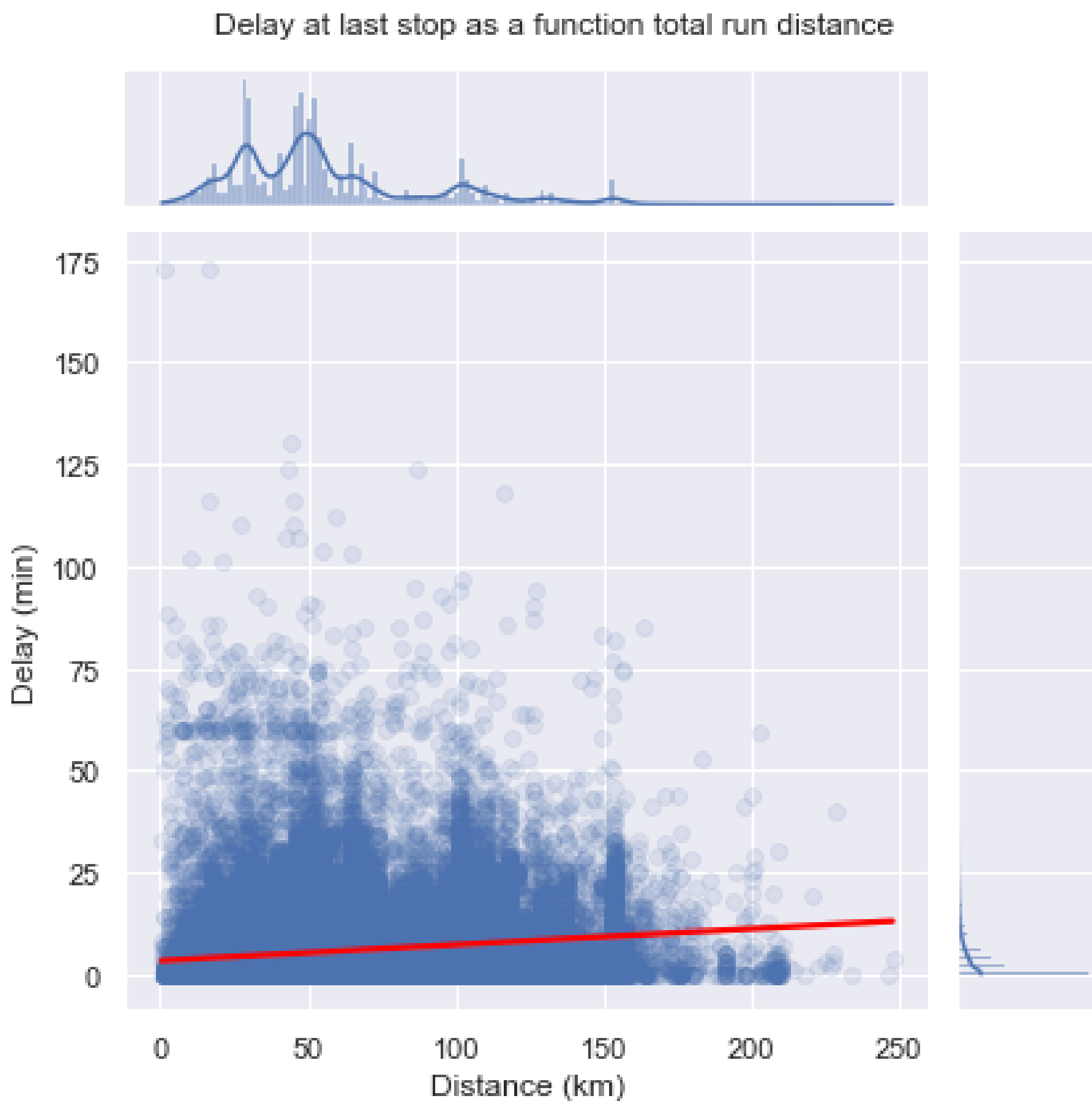
1.8.7 Total distance of a train run

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 the total train run distance. Additionally, we show plot the delay at the last stop as a function to the total train run distance. Here again, the red line indicates the regression line.

[24]:
```
vis.total_train_run_distance_joint_plot(df=df_coords)
```





We can see that train run distance clearly influences the total delay throughout the run, as well as the delay at the last stop.

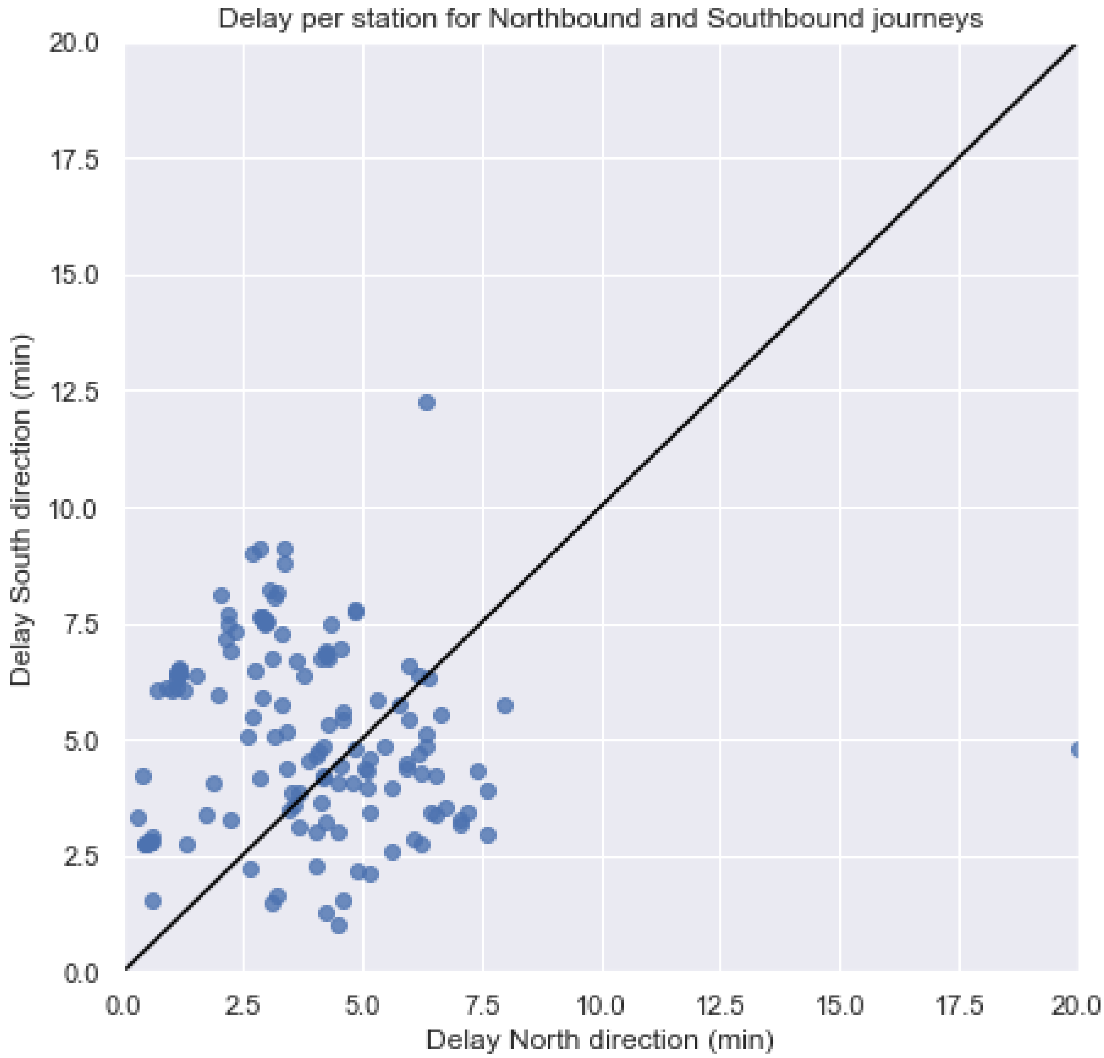
[25]:
```
df_otp = sp.add_cum_delay(df=df_otp)
df_otp = sp.add_distance(df_otp=df_otp, df_coords=df_coords)
```

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.

[26]:
```
vis.plot_northbound_southbound_dely_per_station(df=df_otp)
```

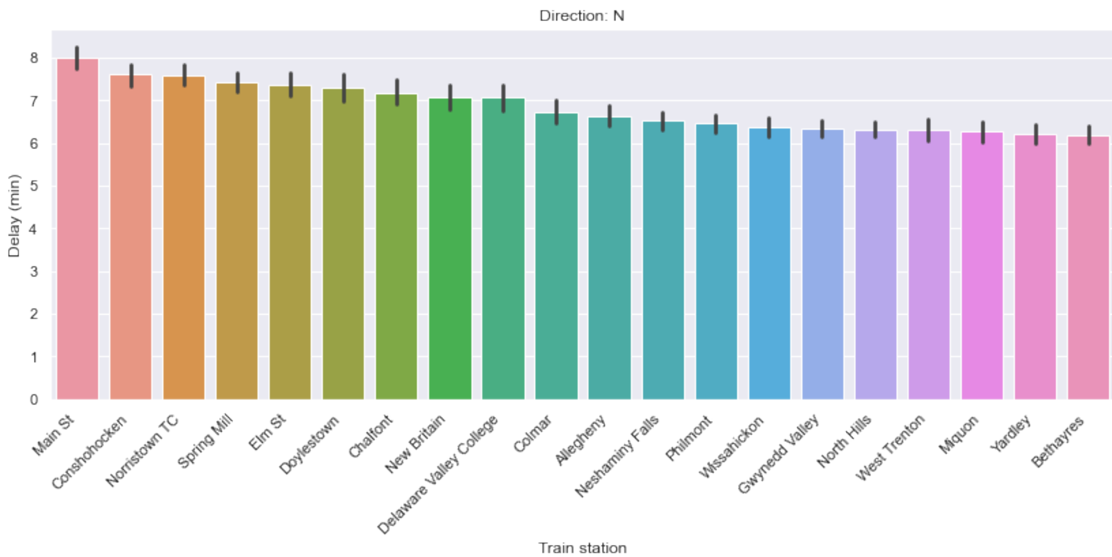


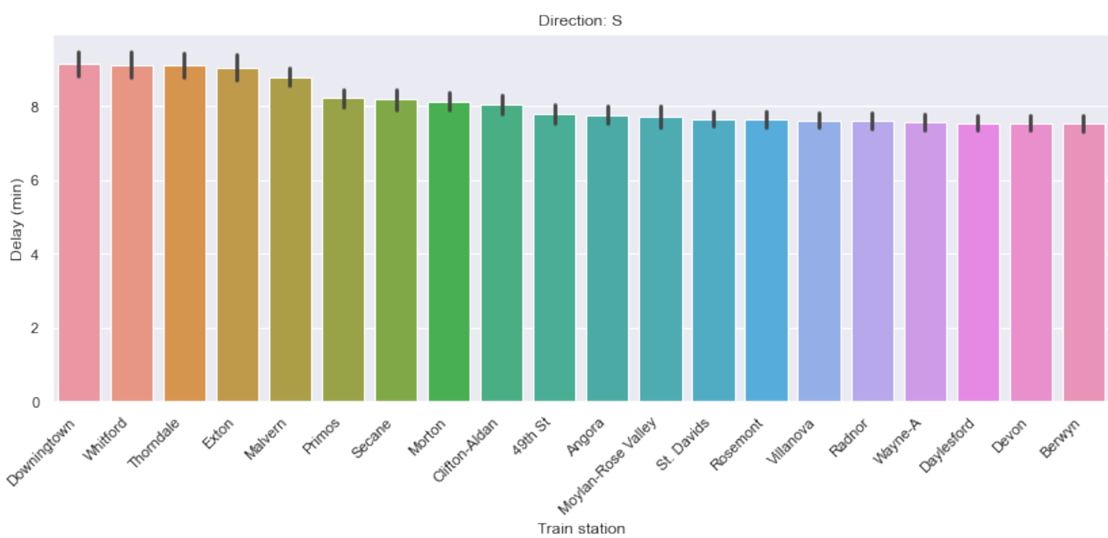
The asymmetry in the scatterplot (even if not that striking) suggests that delay is dependent on direction (as could be expected, since the distance from origin and stop number will be different according to the direction), so we will label long-duration stations per direction.

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.

[27]:
```
vis.plot_stations_with_long_delays(df=df_otp)
```





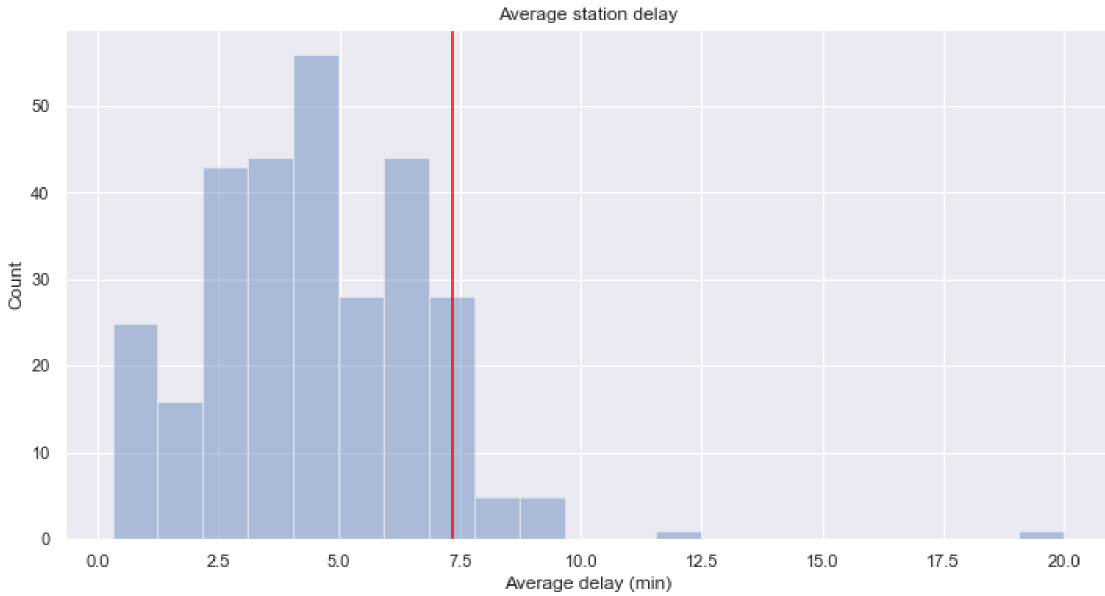
We can label those stations with an average delay above a certain threshold. In this case, we will use the 90th percentile as a threshold. Below you can see the histogram of the average delay per station.

[28]:
```
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.

[29]:
```python
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
```

[29]:

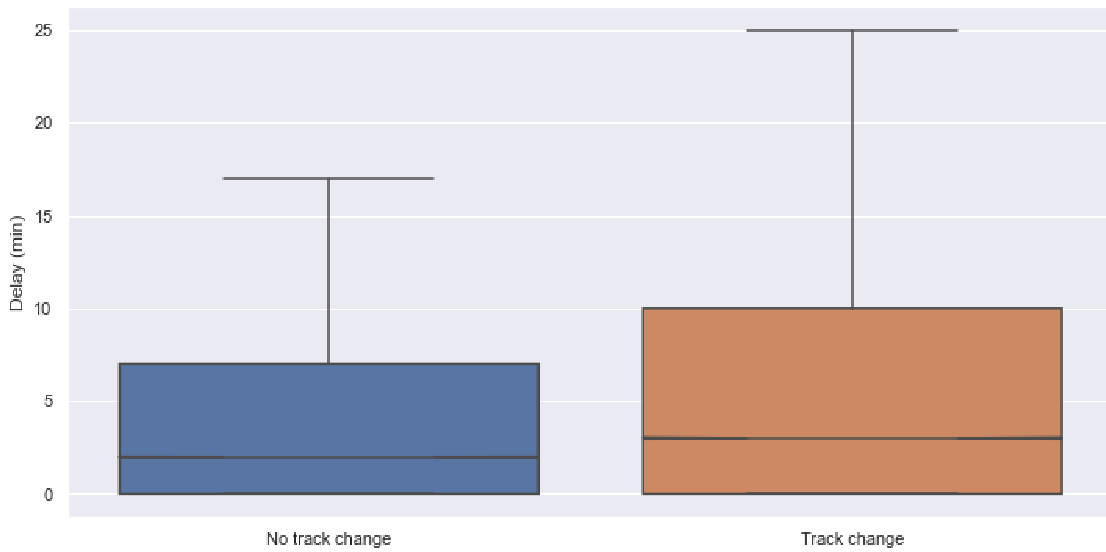
	train_id	next_station	last_week_delay	date
6278	6803	30th Street Station	116.0	2016-04-01
7028	852	30th Street Station	88.0	2016-11-02
1716	393	Yardley	86.0	2016-11-02
315	1556	Jefferson Station	85.0	2016-09-17
128	1524	Berwyn	85.0	2016-07-19
129	1524	Daylesford	85.0	2016-07-19

```
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 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.

[49]:
```
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.

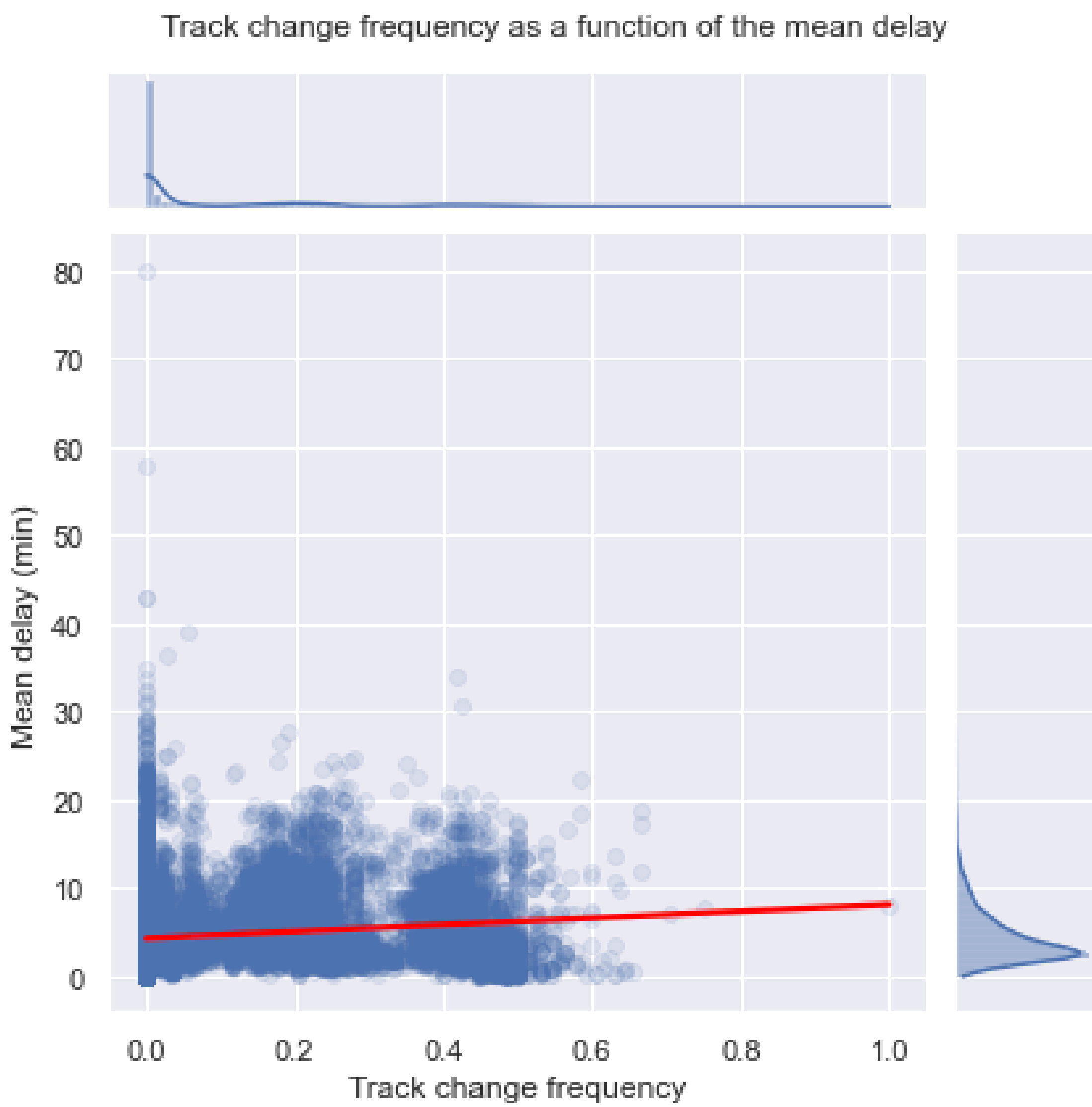
[31]:
```
track_change_frequency_date = sp.
  ↪calculate_track_changes_frequency(df=df_track_changes)
track_change_frequency = sp.
  ↪calculate_aggregated_track_changes(df=track_change_frequency_date)
```

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

[32]:

```
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.
```

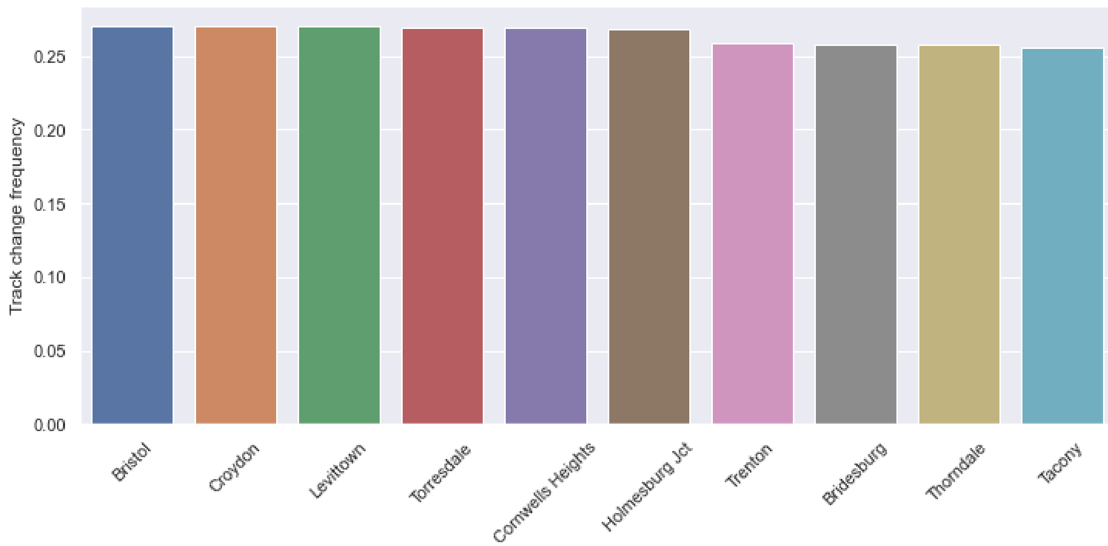


The red line indicates an upward trend, which is confirmed by the significant (i.e. the p-value > 0.05) positive Pearson correlation coefficient. The latter is suggesting that stations with higher delays incur more track changes.

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

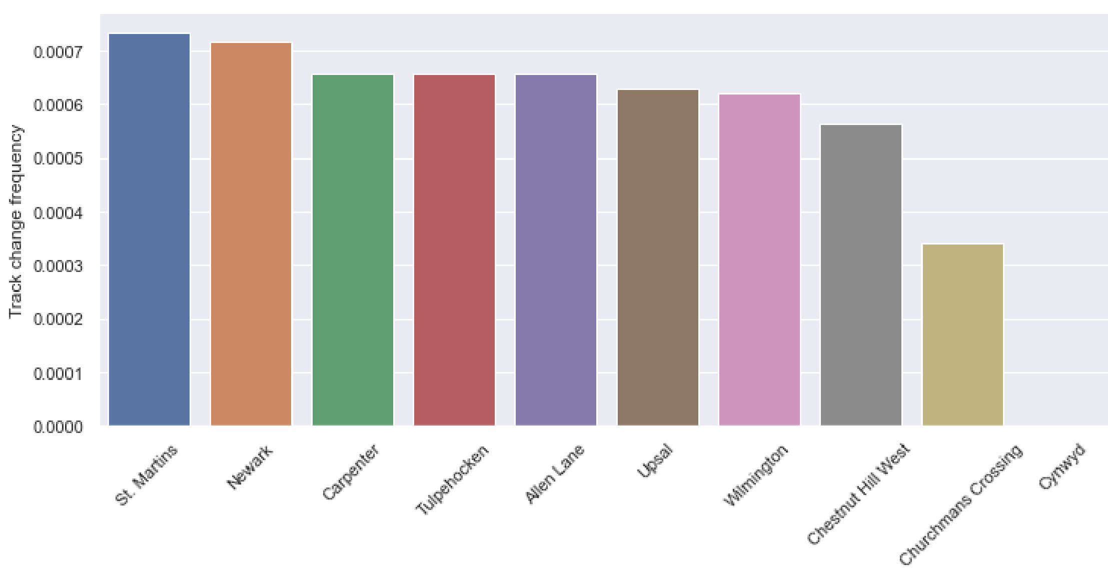
[33]:
```
# get unique combinations of station/coordinates and of train_id/dates
uni_coords = df_train_view[['next_station', 'lon', 'lat']].drop_duplicates()
uni_runs = df_otp[['train_id', 'date']].drop_duplicates()

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 enforced by the physical limitation of only a single track.

[34]:
```
vis.plot_lowest_track_change_frequency(track_change_frequency)
```



[35]:
```
# add track change frequency as a feature
df_otp = df_otp.merge(track_change_frequency, on='next_station')
```

1.9 Final dataset preparation

To facilitate further explotation of the generated features (e.g., to use them as an input for a prediction model), we will construct a single dataset with all the constructed features. An overview of this dataset by the first 5 entries of this dataset is shown below:

[36]:
```
df_output = df_otp[['train_id', 'date', 'origin', 'next_station', 'timeStamp',␣
↪'delay', 'direction', 'hour',
                    'isWeekend', 'isRushHour', 'rank', 'cum_delay', 'distance',␣
↪'cum_distance',
                    'long_delay_station', 'last_week_delay',␣
↪'track_change_frequency']]

df_output.head()
```

[36]:
```
  train_id        date             origin next_station           timeStamp  \
0      396  2016-03-23              Elwyn   Swarthmore 2016-03-23 22:11:24
1      383  2016-03-23  West Trenton Yard   Swarthmore 2016-03-23 20:40:18
2      373  2016-03-23  West Trenton Yard   Swarthmore 2016-03-23 19:53:28
3     9346  2016-03-23              Elwyn   Swarthmore 2016-03-23 09:03:10
4     4311  2016-03-23          Glenside   Swarthmore 2016-03-23 07:52:46

   delay direction hour  isWeekend  isRushHour  rank  cum_delay  distance  \
0      0         N   22      False       False   5.0          2  1.758212
1      2         S   20      False       False  30.0         22  1.772604
2      7         S   19      False        True  30.0        101  1.772604
3      0         N   09      False        True   4.0          0  1.758212
4      1         S   07      False        True  21.0         54  1.772604

   cum_distance  long_delay_station  last_week_delay  track_change_frequency
0      5.266604               False              0.0                0.050535
1     68.184001                True              0.0                0.050535
2     68.184001                True              0.0                0.050535
3      5.009131               False              0.0                0.050535
4     36.146710                True              0.0                0.050535
```

1.10 Conclusion

In this Starter Kit we demonstrated the workflow on how to construct advanced features that might be valuable in this specific question: can we predict train delays at a given station and on a given day. 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.

1.11 Additional information

This Starter Kit was developed in the context of the EluciDATA project. For more information, please contact `info@elucidata.be`.