1. Obtain and review raw data

The subject was reviewing his running styles, training habits, and achievements when he suddenly realized that he could take an in-depth analytical look at my training. He had been using a popular GPS fitness tracker called Runkeeper for years and decided it was time to analyze the running data to have an in-depth review of his performance.

I have exported seven years worth of the subject's training data, from 2012 through 2018. The data is a CSV file where each row is a single training activity.

```
# Import pandas
import pandas as pd

# Define file containing dataset
# runkeeper_file = 'datasets/cardioActivities.csv' #For jupyter notebook
runkeeper_file = '/content/cardioActivities.csv'   #For google colabratory

# Create DataFrame with parse_dates and index_col parameters
df_activities = pd.read_csv(runkeeper_file, parse_dates=['Date'], index_col='Date')

# First look at exported data: select sample of 3 random rows
display(df_activities.sample(3))

# # Print DataFrame summary
print(df_activities.info())
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
Cell In[1], line 9
      6 runkeeper_file = '/content/cardioActivities.csv'   #For google colabratory
      8 # Create DataFrame with parse_dates and index_col parameters
----> 9 df_activities = pd.read_csv(runkeeper_file, parse_dates=['Date'], index_col='Date')
     11 # First look at exported data: select sample of 3 random rows
     12 display(df_activities.sample(3))

File c:\Users\Atharva Deorukhkar\AppData\Local\Programs\Python\Python39\lib\site-packages\pandas-2.1.2-py3.9-win-amd64.egg\pandas\io\parsers\readers.py:948,
in read_csv(filepath_or_buffer, sep, delimiter, header, names, index_col, usecols, dtype, engine, converters, true_values, false_values, skipinitialspace,
skiprows, skipfooter, nrows, na_values, keep_default_na, na_filter, verbose, skip_blank_lines, parse_dates, infer_datetime_format, keep_date_col, date_parser,
date_format, dayfirst, cache_dates, iterator, chunksize, compression, thousands, decimal, lineterminator, quotechar, quoting, doublequote, escapechar,
comment, encoding, encoding_errors, dialect, on_bad_lines, delim_whitespace, low_memory, memory_map, float_precision, storage_options, dtype_backend)
    935 kwds_defaults = _refine_defaults_read(
    936     dialect,
    937     delimiter,
   (...)
    944     dtype_backend=dtype_backend,
    945 )
    946 kwds.update(kwds_defaults)
--> 948 return _read(filepath_or_buffer, kwds)

File c:\Users\Atharva Deorukhkar\AppData\Local\Programs\Python\Python39\lib\site-packages\pandas-2.1.2-py3.9-win-amd64.egg\pandas\io\parsers\readers.py:611,
in _read(filepath_or_buffer, kwds)
    608 _validate_names(kwds.get("names", None))
    610 # Create the parser.
--> 611 parser = TextFileReader(filepath_or_buffer, **kwds)
    613 if chunksize or iterator:
    614     return parser

File c:\Users\Atharva Deorukhkar\AppData\Local\Programs\Python\Python39\lib\site-packages\pandas-2.1.2-py3.9-win-amd64.egg\pandas\io\parsers\readers.py:1448,
in TextFileReader.__init__(self, f, engine, **kwds)
   1445     self.options["has_index_names"] = kwds["has_index_names"]
   1447 self.handles: IOHandles | None = None
-> 1448 self._engine = self._make_engine(f, self.engine)

File c:\Users\Atharva Deorukhkar\AppData\Local\Programs\Python\Python39\lib\site-packages\pandas-2.1.2-py3.9-win-amd64.egg\pandas\io\parsers\readers.py:1705,
in TextFileReader._make_engine(self, f, engine)
   1703     if "b" not in mode:
   1704         mode += "b"
-> 1705 self.handles = get_handle(
   1706     f,
   1707     mode,
   1708     encoding=self.options.get("encoding", None),
   1709     compression=self.options.get("compression", None),
   1710     memory_map=self.options.get("memory_map", False),
   1711     is_text=is_text,
   1712     errors=self.options.get("encoding_errors", "strict"),
   1713     storage_options=self.options.get("storage_options", None),
   1714 )
   1715 assert self.handles is not None
   1716 f = self.handles.handle

File c:\Users\Atharva Deorukhkar\AppData\Local\Programs\Python\Python39\lib\site-packages\pandas-2.1.2-py3.9-win-amd64.egg\pandas\io\common.py:863, in
get_handle(path_or_buf, mode, encoding, compression, memory_map, is_text, errors, storage_options)
    858 elif isinstance(handle, str):
    859     # Check whether the filename is to be opened in binary mode.
    860     # Binary mode does not support 'encoding' and 'newline'.
    861     if ioargs.encoding and "b" not in ioargs.mode:
    862         # Encoding
--> 863         handle = open(
    864             handle,
    865             ioargs.mode,
    866             encoding=ioargs.encoding,
    867             errors=errors,
    868             newline="",
    869         )
    870     else:
    871         # Binary mode
    872         handle = open(handle, ioargs.mode)

FileNotFoundError: [Errno 2] No such file or directory: '/content/cardioActivities.csv'
```

```
df_activities.describe()
```

	Distance (km)	Average Speed (km/h)	Calories Burned	Climb (m)	Average Heart Rate (bpm)	Friend's Tagged
count	508.000000	508.000000	5.080000e+02	508.00000	294.000000	0.0
mean	11.757835	11.341654	1.878197e+04	128.00000	143.530612	NaN
std	6.209219	2.510516	2.186930e+05	108.52604	10.583848	NaN
min	0.760000	1.040000	4.000000e+01	0.00000	77.000000	NaN
25%	7.015000	10.470000	4.917500e+02	53.00000	140.000000	NaN
50%	11.460000	11.030000	7.280884e+02	92.00000	144.000000	NaN
75%	13.642500	11.642500	9.212500e+02	172.25000	149.000000	NaN
max	49.180000	24.330000	4.072685e+06	982.00000	172.000000	NaN

2. Data preprocessing

Missing values using the `info()` method are spotted. What are the reasons for these missing values? Some heart rate information is missing because the subject might have not always used a cardio sensor. In the case of the `Notes` column, it is an optional field that he sometimes might have left blank. Also, he only used the `Route Name` column once, and never used the `Friend's Tagged` column.

We'll fill in missing values in the heart rate column to avoid misleading results later, but right now, our first data preprocessing steps will be to:

- Remove columns not useful for our analysis.
- Replace the "Other" activity type to "Unicycling" because that was always the "Other" activity.
- Count missing values.

```
# Define list of columns to be deleted
cols_to_drop = ['Friend\'s Tagged',
                'Route Name',
                'GPX File',
                'Activity Id',
                'Calories Burned',
                'Notes']

# Delete unnecessary columns
df_activities = df_activities.drop(columns=cols_to_drop)

# Count types of training activities
display(df_activities['Type'].value_counts())

# Rename 'Other' type to 'Unicycling'
df_activities['Type'] = df_activities['Type'].str.replace('Other', 'Unicycling')

# # Count missing values for each column
print(df_activities.isnull().sum())
```

```
Running    459
Cycling     29
Walking     18
Other        2
Name: Type, dtype: int64
Type                          0
Distance (km)                 0
Duration                      0
Average Pace                  0
Average Speed (km/h)          0
Climb (m)                     0
Average Heart Rate (bpm)    214
dtype: int64
```

3. Dealing with missing values

As we can see from the last output, there are 214 missing entries for the average heart rate.

We can't go back in time to get those data, but we can fill in the missing values with an average value. This process is called *mean imputation*. When imputing the mean to fill in missing data, we need to consider that the average heart rate varies for different activities (e.g., walking vs. running). We'll filter the DataFrames by activity type (`Type`) and calculate each activity's mean heart rate, then fill in the missing values with those means.

```
# Calculate sample means for heart rate for each training activity type
avg_hr_run = df_activities[df_activities['Type'] == 'Running']['Average Heart Rate (bpm)'].mean()
avg_hr_cycle = df_activities[df_activities['Type'] == 'Cycling']['Average Heart Rate (bpm)'].mean()

# Split whole DataFrame into several, specific for different activities
df_run = df_activities[df_activities['Type'] == 'Running'].copy()
df_walk = df_activities[df_activities['Type'] == 'Walking'].copy()
df_cycle = df_activities[df_activities['Type'] == 'Cycling'].copy()

# Filling missing values with counted means
df_walk['Average Heart Rate (bpm)'].fillna(110, inplace=True)
df_run['Average Heart Rate (bpm)'].fillna(int(avg_hr_run), inplace=True)
df_cycle['Average Heart Rate (bpm)'].fillna(int(avg_hr_cycle), inplace=True)

# Count missing values for each column in running data
print(df_run.isnull().sum())
```

```
Type              0
Distance (km)     0
Duration          0
Average Pace      0
```

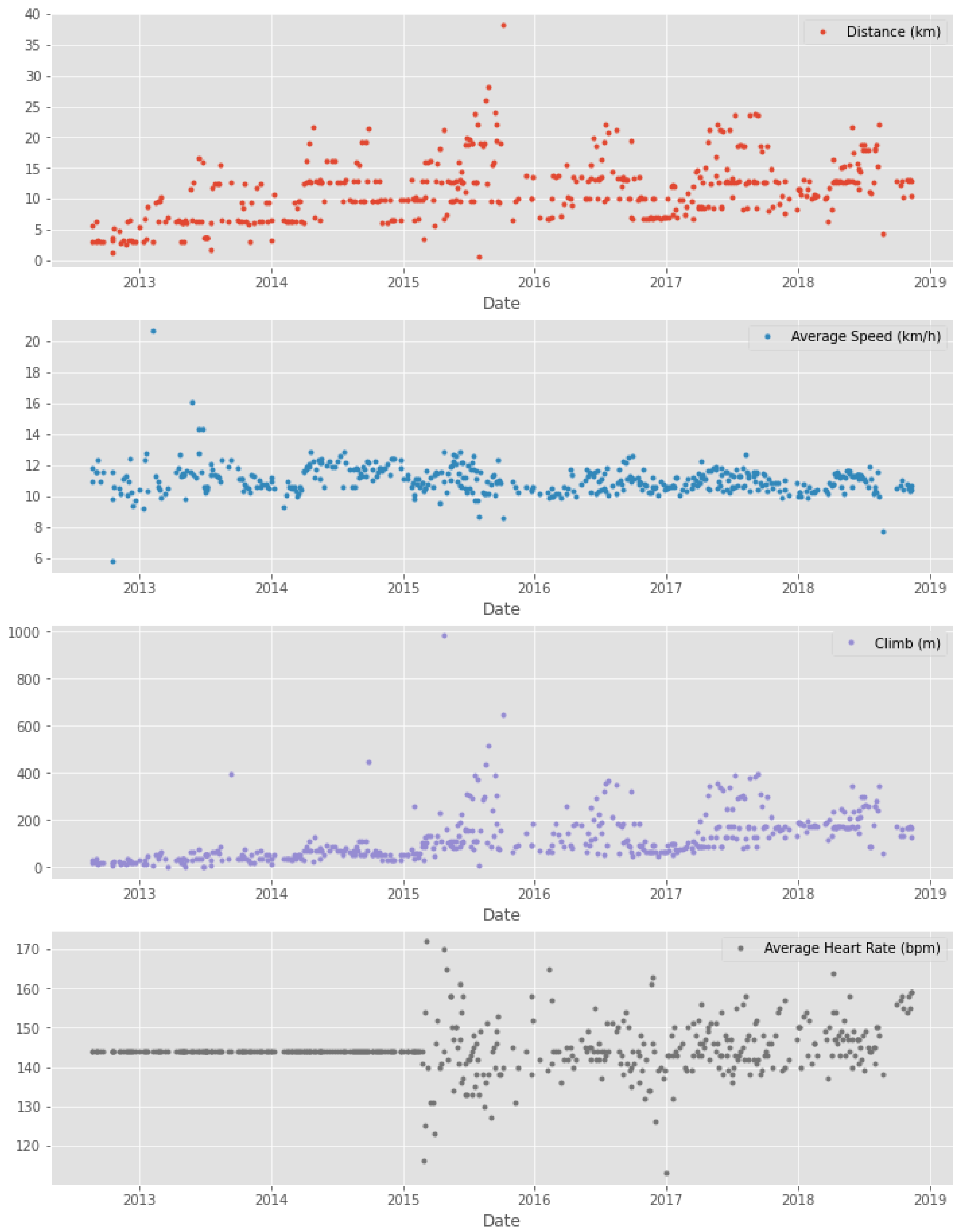
```
Average Speed (km/h)      0
Climb (m)                 0
Average Heart Rate (bpm)  0
dtype: int64
```

4. Plot running data

Now we can create our first plot! As we found earlier, most of the activities in the data were running (459 of them to be exact). There are only 29, 18, and two instances for cycling, walking, and unicycling, respectively. So for now, let's focus on plotting the different running metrics.

An excellent first visualization is a figure with four subplots, one for each running metric (each numerical column). Each subplot will have a different y-axis, which is explained in each legend. The x-axis, `Date`, is shared among all subplots.

```
%matplotlib inline

# Import matplotlib, set style and ignore warning
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
plt.style.use('ggplot')
warnings.filterwarnings(
    action='ignore', module='matplotlib.figure', category=UserWarning,
    message=('This figure includes Axes that are not compatible with tight_layout, so results might be incorrect.')
)

# Prepare data subsetting period from 2013 till 2018
runs_subset_2013_2018 = df_run

# Create, plot and customize in one step
runs_subset_2013_2018.plot(subplots=True,
                           sharex=False,
                           figsize=(12,16),
                           linestyle='none',
                           marker='o',
                           markersize=3,
                          )

# Show plot
plt.show()
```



5. Running statistics

Running helps people stay mentally and physically healthy and productive at any age. When runners talk to each other about their hobby, we not only discuss our results, but we also discuss different training strategies.

You'll know you're with a group of runners if you commonly hear questions like:

- What is your average distance?
- How fast do you run?
- Do you measure your heart rate?
- How often do you train?

Let's find the answers to these questions in the data. If you look back at plots in Task 4, you can see the answer to, *Do you measure your heart rate?* Before 2015: no. To look at the averages, let's only use the data from 2015 through 2018.

In pandas, the `resample()` method is similar to the `groupby()` method - with `resample()` you group by a specific time span. We'll use `resample()` to group the time series data by a sampling period and apply several methods to each sampling period. In our case, we'll resample annually and weekly.

```
# Prepare running data for the last 4 years
runs_subset_2015_2018 = df_run.iloc[:303]

# Calculate annual statistics
print('How my average run looks in last 4 years:')
display(runs_subset_2015_2018.resample('A').mean())

# Calculate weekly statistics
print('Weekly averages of last 4 years:')
display(runs_subset_2015_2018.resample('W').mean().mean())

# Mean weekly counts
weekly_counts_average = runs_subset_2015_2018['Distance (km)'].resample('A').mean().count()
print('How many trainings per week I had on average:', weekly_counts_average)
```

```
How my average run looks in last 4 years:
```

	Distance (km)	Average Speed (km/h)	Climb (m)	Average Heart Rate (bpm)
Date				
2015-12-31	13.602805	10.998902	160.170732	143.353659
2016-12-31	11.411667	10.837778	133.194444	143.388889
2017-12-31	12.935176	10.959059	169.376471	145.247059
2018-12-31	13.339063	10.777969	191.218750	148.125000

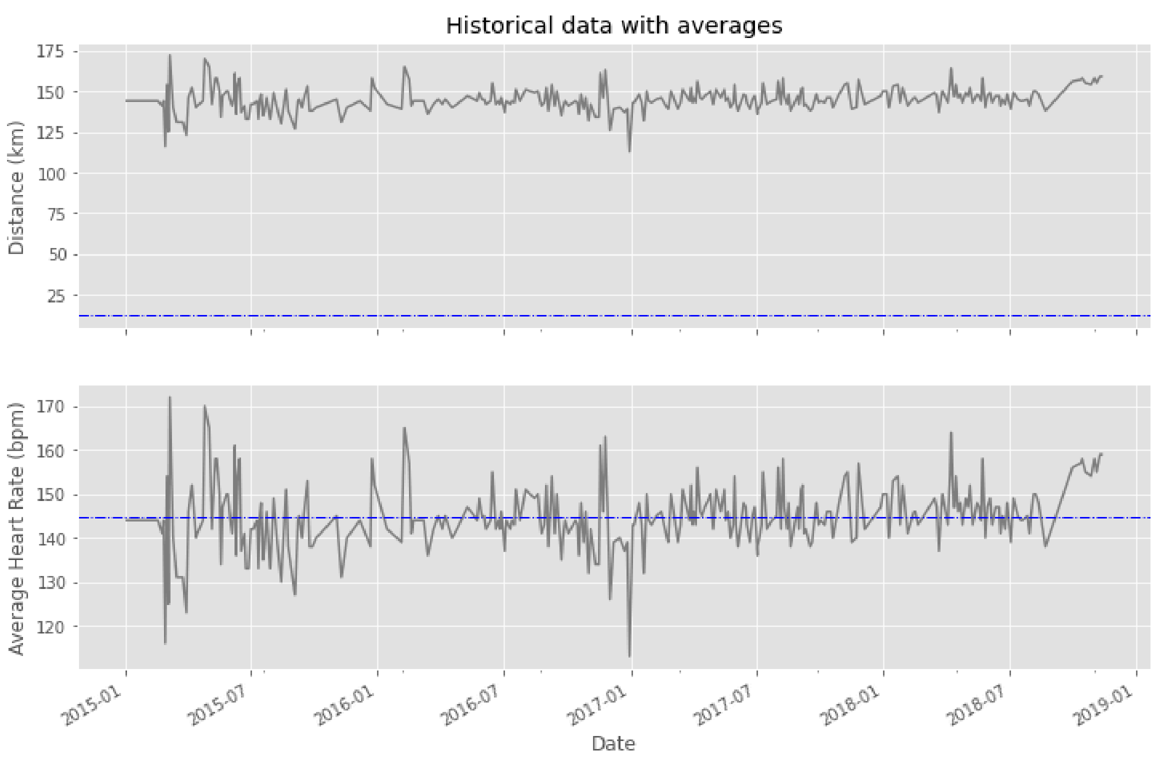
```
Weekly averages of last 4 years:
Distance (km)               12.518176
Average Speed (km/h)        10.835473
Climb (m)                  158.325444
Average Heart Rate (bpm)   144.801775
dtype: float64
How many trainings per week I had on average: 4
```

6. Visualization with averages

Let's plot the long term averages of distance run and heart rate with their raw data to visually compare the averages to each training session. Again, we'll use the data from 2015 through 2018.

In this task, we will use `matplotlib` functionality for plot creation and customization.

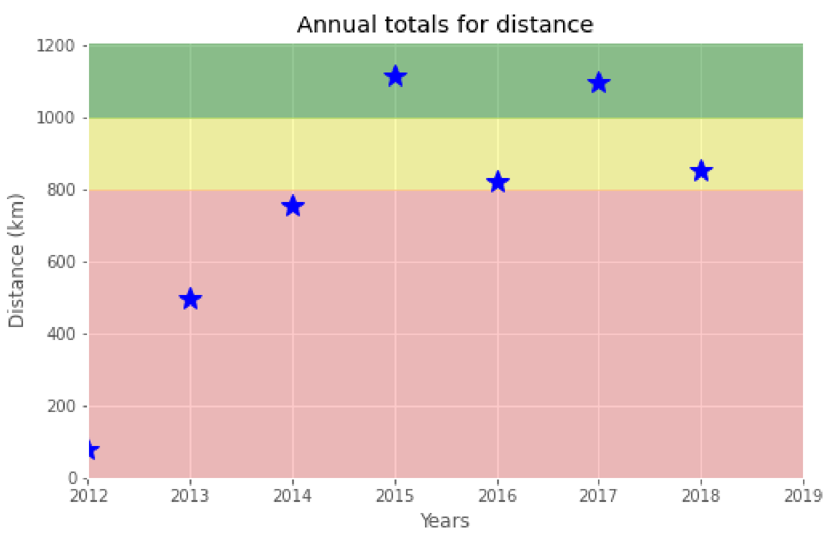
```
# Prepare data
runs_subset_2015_2018 = df_run['2018':'2015']
runs_distance = runs_subset_2015_2018['Distance (km)']
runs_hr = runs_subset_2015_2018['Average Heart Rate (bpm)']

# Create plot
fig, (ax1, ax2) = plt.subplots(2, sharex=True, figsize=(12,8))

# Plot and customize first subplot
runs_hr.plot(ax=ax1, color='gray')
ax1.set(ylabel='Distance (km)', title='Historical data with averages')
ax1.axhline(runs_distance.mean(), color='blue', linewidth=1, linestyle='-.')

# Plot and customize second subplot
runs_hr.plot(ax=ax2, color='gray')
ax2.set(xlabel='Date', ylabel='Average Heart Rate (bpm)')
ax2.axhline(runs_hr.mean(), color='blue', linewidth=1, linestyle='-.')

# Show plot
plt.show()
```



7. Did the subject reach his goals?

To motivate himself to run regularly, the subject set a target goal of running 1000 km per year. Let's visualize his annual running distance (km) from 2013 through 2018 to see if he reached his goal each year. Only stars in the green region indicate success.

```
# Prepare data
df_run_dist_annual = df_run['Distance (km)'].resample('A').sum()

# Create plot
fig = plt.figure(figsize=(8,5))

# Plot and customize
ax = df_run_dist_annual.plot(marker='*', markersize=14, linewidth=0, color='blue')
ax.set(ylim=[0, 1210],
       xlim=['2012','2019'],
       ylabel='Distance (km)',
       xlabel='Years',
       title='Annual totals for distance')

ax.axhspan(1000, 1210, color='green', alpha=0.4)
ax.axhspan(800, 1000, color='yellow', alpha=0.3)
ax.axhspan(0, 800, color='red', alpha=0.2)

# Show plot
plt.show()
```



8. Progress tracking

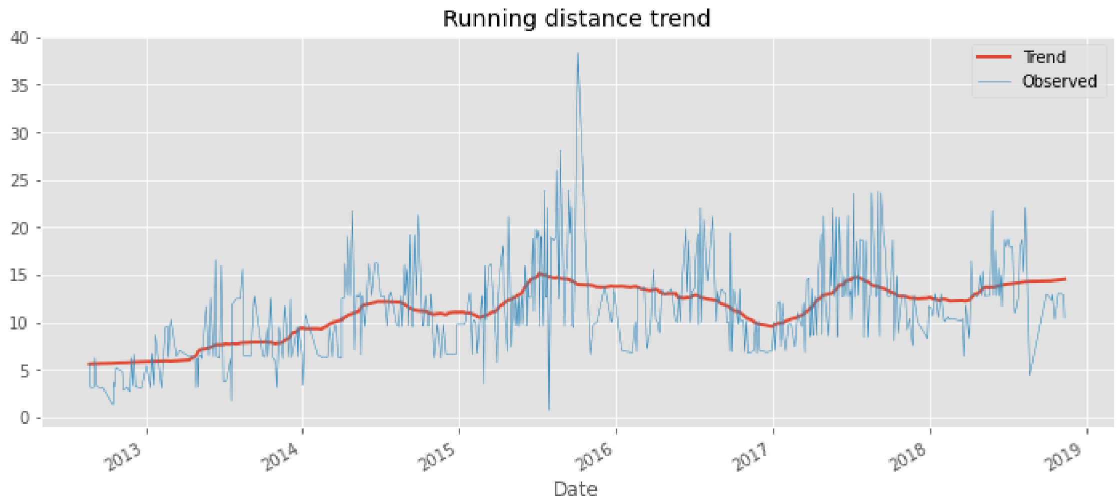
Let's dive a little deeper into the data to answer a tricky question: Is the subject progressing in terms of his running skills?

To answer this question, we'll decompose his weekly distance run and visually compare it to the raw data. A red trend line will represent the weekly distance run.

We are going to use `statsmodels` library to decompose the weekly trend.

```
# Import required library
import statsmodels.api as sm

# Prepare data
df_run_dist_wkly = df_run['Distance (km)'].bfill()
decomposed = sm.tsa.seasonal_decompose(df_run_dist_wkly, extrapolate_trend=1, freq=52)

# Create plot
fig = plt.figure(figsize=(12,5))

# Plot and customize
ax = decomposed.trend.plot(label='Trend', linewidth=2)
ax = decomposed.observed.plot(label='Observed', linewidth=0.5)
```

```
ax.legend()
ax.set_title('Running distance trend')

# Show plot
plt.show()
```

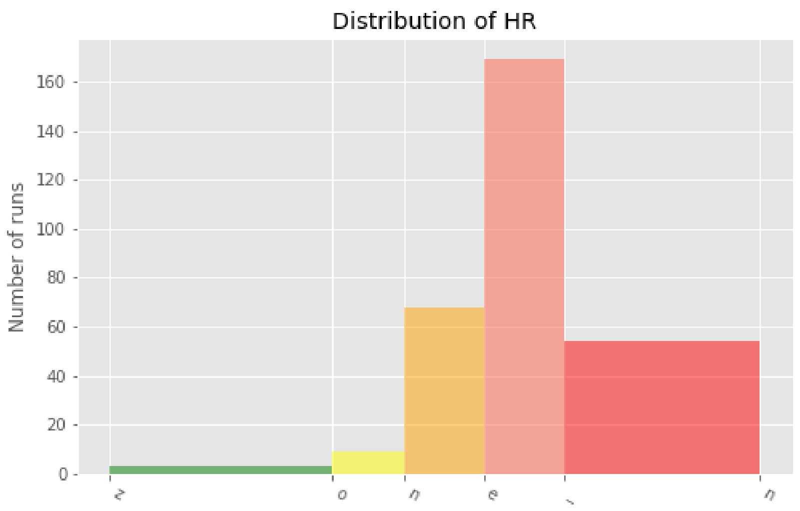


9. Training intensity

Heart rate is a popular metric used to measure training intensity. Depending on age and fitness level, heart rates are grouped into different zones that people can target depending on training goals. A target heart rate during moderate-intensity activities is about 50-70% of maximum heart rate, while during vigorous physical activity it's about 70-85% of maximum.

We'll create a distribution plot of the heart rate data by training intensity. It will be a visual presentation for the number of activities from predefined training zones.

```
# Prepare data
hr_zones = [100, 125, 133, 142, 151, 173]
zone_names = ['Easy', 'Moderate', 'Hard', 'Very hard', 'Maximal']
zone_colors = ['green', 'yellow', 'orange', 'tomato', 'red']
df_run_hr_all = df_run['2018':'2015']['Average Heart Rate (bpm)']

# Create plot
fig, ax = plt.subplots(figsize=(8,5))

# Plot and customize
n, bins, patches = ax.hist(df_run_hr_all, bins=hr_zones, alpha=0.5)
for i in range(0, len(patches)):
    patches[i].set_facecolor(zone_colors[i])

ax.set(title='Distribution of HR', ylabel='Number of runs')
ax.xaxis.set(ticks=hr_zones)
ax.set_xticklabels(labels='zone_names', rotation=-30, ha='left')

# Show plot
plt.show()
```



10. Detailed summary report

With all this data cleaning, analysis, and visualization, let's create detailed summary tables of the training.

To do this, we'll create two tables. The first table will be a summary of the distance (km) and climb (m) variables for each training activity. The second table will list the summary statistics for the average speed (km/hr), climb (m), and distance (km) variables for each training activity.

```
# Concatenating three DataFrames
frames = [df_walk, df_cycle]
df_run_walk_cycle = df_run.append(frames, sort=False)

dist_climb_cols, speed_col = ['Distance (km)', 'Climb (m)'], ['Average Speed (km/h)']

# Calculating total distance and climb in each type of activities
df_totals = df_run_walk_cycle.groupby('Type').sum()

print('Totals for different training types:')
display(df_totals)

# Calculating summary statistics for each type of activities
df_summary = df_run_walk_cycle.groupby('Type')[dist_climb_cols + speed_col].describe()
```

```
# Combine totals with summary
for i in dist_climb_cols:
    df_summary[i, 'total'] = df_totals[i]

print('Summary statistics for different training types:')
print(df_summary.stack())
```

```
Totals for different training types:
```

Type	Distance (km)	Average Speed (km/h)	Climb (m)	Average Heart Rate (bpm)
Cycling	680.58	554.63	6976	3602.0
Running	5224.50	5074.84	57278	66369.0
Walking	33.45	99.89	349	1980.0

```
Summary statistics for different training types:
               Average Speed (km/h)     Climb (m)  Distance (km)
Type
Cycling 25%               16.980000    139.000000      15.530000
        50%               19.500000    199.000000      20.300000
        75%               21.490000    318.000000      29.400000
        count             29.000000     29.000000      29.000000
        max               24.330000    553.000000      49.180000
        mean              19.125172    240.551724      23.468276
        min               11.380000     58.000000      11.410000
        std                3.257100    128.960289       9.451040
        total                   NaN   6976.000000     680.580000
Running 25%               10.495000     54.000000       7.415000
        50%               10.980000     91.000000      10.810000
        75%               11.520000    171.000000      13.190000
        count            459.000000    459.000000     459.000000
        max               20.720000    982.000000      38.320000
        mean              11.056296    124.788671      11.382353
        min                5.770000      0.000000       0.760000
        std                0.953273    103.382177       4.937853
        total                   NaN  57278.000000    5224.500000
Walking 25%                5.555000      7.000000       1.385000
        50%                5.970000     10.000000       1.485000
        75%                6.512500     15.500000       1.787500
        count             18.000000     18.000000      18.000000
        max                6.910000    112.000000       4.290000
        mean               5.549444     19.388889       1.858333
        min                1.040000      5.000000       1.220000
        std                1.459309     27.110100       0.880055
        total                   NaN    349.000000      33.450000
```

11. Fun facts

To wrap up, let’s pick some fun facts out of the summary tables and solve the last exercise.

This data (running history) represent 6 years, 2 months and 21 days.

```
FUN FACTS
- Average distance: 11.38 km
- Longest distance: 38.32 km
- Highest climb: 982 m
- Total climb: 57,278 m
- Total number of km run: 5,224 km
- Total runs: 459
```