1. Obtain and review raw data

One day, my old running friend and I were chatting about our running styles, training habits, and achievements, when I suddenly realized that I could take an in-depth analytical look at my training. I have been using a popular GPS fitness tracker called Runkeeper (https://runkeeper.com/) for years and decided it was time to analyze my running data to see how I was doing.

Since 2012, I've been using the Runkeeper app, and it's great. One key feature: its excellent data export. Anyone who has a smartphone can download the app and analyze their data like we will in this notebook.

After logging your run, the first step is to export the data from Runkeeper (which I've done already). Then import the data and start exploring to find potential problems. After that, create data cleaning strategies to fix the issues. Finally, analyze and visualize the clean time-series data.

I exported seven years worth of my training data, from 2012 through 2018. The data is a CSV file where each row is a single training activity. Let's load and inspect it.

In [1]:
```
# Import pandas
import pandas as pd
# Define file containing dataset
runkeeper_file = 'datasets/cardioActivities.csv'

# Create DataFrame with parse_dates and index_col parameters
df_activities = pd.read_csv(runkeeper_file, parse_dates=True, index_col='Dat
e')
# First look at exported data: select sample of 3 random rows
display(df_activities.sample(3))

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

Date	Activity Id	Type	Route Name	Distance (km)	Duration	Average Pace	Average Speed (km/h)	Calories Burned	Climb (m)
2012-08-24 12:59:42	018f66a7-da5e-4985-a8fe-725a33317c87	Walking	NaN	1.48	17:56	12:09	4.94	942192.0	12
2013-08-08 18:30:36	00f40213-297d-44f0-8311-a6254ac6fa4f	Running	NaN	12.47	1:06:33	5:20	11.24	891.0	61
2012-10-23 18:43:56	1387c891-e974-4c9f-9e52-3d211fb12a68	Running	NaN	5.20	29:26	5:40	10.59	371.0	27

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 508 entries, 2018-11-11 14:05:12 to 2012-08-22 18:53:54
Data columns (total 13 columns):
Activity Id                 508 non-null object
Type                        508 non-null object
Route Name                  1 non-null object
Distance (km)               508 non-null float64
Duration                    508 non-null object
Average Pace                508 non-null object
Average Speed (km/h)        508 non-null float64
Calories Burned             508 non-null float64
Climb (m)                   508 non-null int64
Average Heart Rate (bpm)    294 non-null float64
Friend's Tagged             0 non-null float64
Notes                       231 non-null object
GPX File                    504 non-null object
dtypes: float64(5), int64(1), object(7)
memory usage: 55.6+ KB
```

In [235]:
```
%%nose
# %%nose needs to be included at the beginning of every @tests cell

# One or more tests of the student's code
# The @solution should pass the tests
# The purpose of the tests is to try to catch common errors and
# to give the student a hint on how to resolve these errors

def test_pandas_loaded():
    assert 'pd' in globals(), \
    'Did you import the pandas library aliased as pd?'

def test_activities_correctly_loaded():
    correct_activities = pd.read_csv(runkeeper_file,  parse_dates=True, index_
col='Date')
    assert correct_activities.equals(df_activities), \
    "The variable df_activities should contain data read from runkeeper_file w
ith dates parsed and the index column set to date."

def strip_comment_lines(cell_input):
    """Returns cell input string with comment lines removed."""
    return '\n'.join(line for line in cell_input.splitlines() if not line.star
tswith('#'))

last_input = strip_comment_lines(In[-2])

def test_sample_command():
    assert ('df_activities.sample(n=3)' in last_input) or ('df_activities.samp
le(3)' in last_input), \
        "We expected the sample() method with the n parameter set to 3 in your
input."

def test_info_command():
    assert 'df_activities.info()' in last_input, \
        "We expected the info() method to be called on df_activities in your i
nput."
```

Out[235]:
```
4/4 tests passed
```

2. Data preprocessing

Lucky for us, the column names Runkeeper provides are informative, and we don't need to rename any columns.

But, we do notice missing values using the `info()` method. What are the reasons for these missing values? It depends. Some heart rate information is missing because I didn't always use a cardio sensor. In the case of the `Notes` column, it is an optional field that I sometimes left blank. Also, I 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.

In [2]:
```
# Define list of columns to be deleted
cols_to_drop = ['Friend\'s Tagged','Route Name','GPX File','Activity Id','Calo
ries Burned', 'Notes']

# Delete unnecessary columns
df_activities.drop(cols_to_drop,axis=1,inplace=True)

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

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

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

```
Running    459
Cycling     29
Walking     18
Other        2
Name: Type, dtype: int64
```

Out[2]:
```
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
```

In [237]:
```
%%nose
# %%nose needs to be included at the beginning of every @tests cell

# One or more tests of the student's code
# The @solution should pass the tests
# The purpose of the tests is to try to catch common errors and
# to give the student a hint on how to resolve these errors

last_output = _

cols_to_drop_correct = ['Friend\'s Tagged','Route Name','GPX File','Activity I
d','Calories Burned', 'Notes']

def test_columns_deleted():
    for cld in cols_to_drop_correct:
        assert cld not in list(df_activities.columns), \
        "Did you drop unnecessary columns as described in the instructions?"

def strip_comment_lines(cell_input):
    """Returns cell input string with comment lines removed."""
    return '\n'.join(line for line in cell_input.splitlines() if not line.star
tswith('#'))

last_input = strip_comment_lines(In[-2])

def test_values_count():
    assert "df_activities" in last_input, \
        "We expected the df_activities variable in your input."
    assert 'value_counts' in last_input, \
        "We expected the value_counts() method in your input."

def test_replace():
    assert df_activities[df_activities['Type'] == 'Unicycling']['Type'].count
() == 2, \
        "The count of activities of type 'Unicycling' is incorrect."

def test_missing_output():
    try:
        assert "214" in last_output.to_string()
    except AttributeError:
        assert False, \
        "Please use isnull().sum() as the last line of code in the cell to cou
nt missing values."
    except AssertionError:
        assert False, \
        "The output of the cell is not what we expected. You should see 214 mi
ssing values."
```

Out[237]:
```
4/4 tests passed
```

3. Dealing with missing values

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

In [3]:
```
# 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 Hear
t 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
df_run.isnull().sum()
```

Out[3]:
```
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
```

In [239]:
```
%%nose
# %%nose needs to be included at the beginning of every @tests cell

# One or more tests of the student's code
# The @solution should pass the tests
# The purpose of the tests is to try to catch common errors and
# to give the student a hint on how to resolve these errors


last_output = _

def strip_comment_lines(cell_input):
    """Returns cell input string with comment lines removed."""
    return '\n'.join(line for line in cell_input.splitlines() if not line.star
tswith('#'))

last_input = strip_comment_lines(In[-2])

def test_avg_hr_cycle():
    assert avg_hr_cycle == 124.4, \
    "The value of avg_hr_rate is not what we expected. Use code similar to avg
_hr_run with 'Type' == 'Cycling'."

def test_df_cycle_correct():
    assert len(df_cycle) == 29, \
    "The variable df_cycle should contain data filtered from df_activities wit
h 'Type' == 'Cycling'."

def test_df_cycle_fillna():
    assert df_cycle['Average Heart Rate (bpm)'].isnull().sum() == 0, \
    "There are missing values in df_cycle 'Average Heart Rate (bpm)' column."

def test_isnull_output():
    try:
        assert "Average Heart Rate (bpm)    0" in last_output.to_string()
    except AttributeError:
        assert False, "'Average Heart Rate (bpm)    0' should be the output of
the cell. Please do not use the print() or display() functions; ensure the isn
ull().sum() code is the last line of code in the cell."
    except AssertionError:
        assert False, \
        "The output of the cell is not what we expected. You should see 'Avera
ge Heart Rate (bpm)    0'."
```

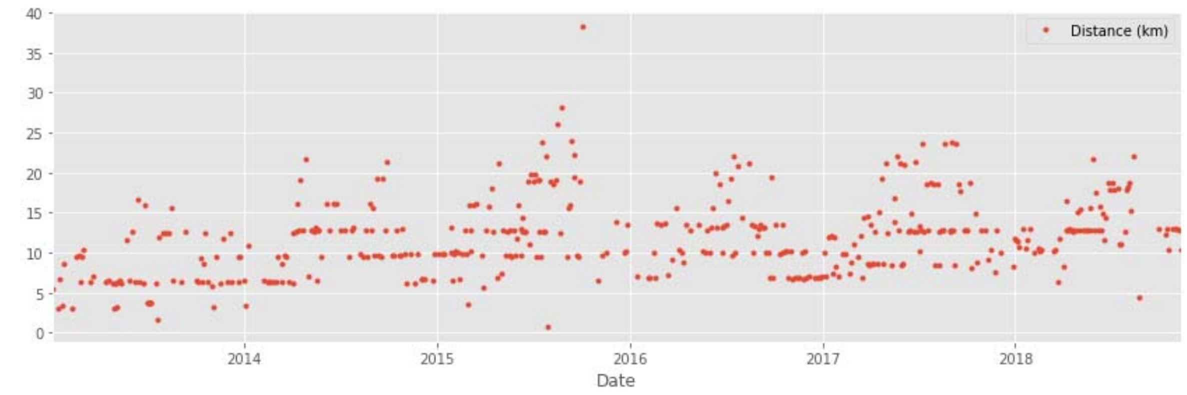
Out[239]:
```
4/4 tests passed
```

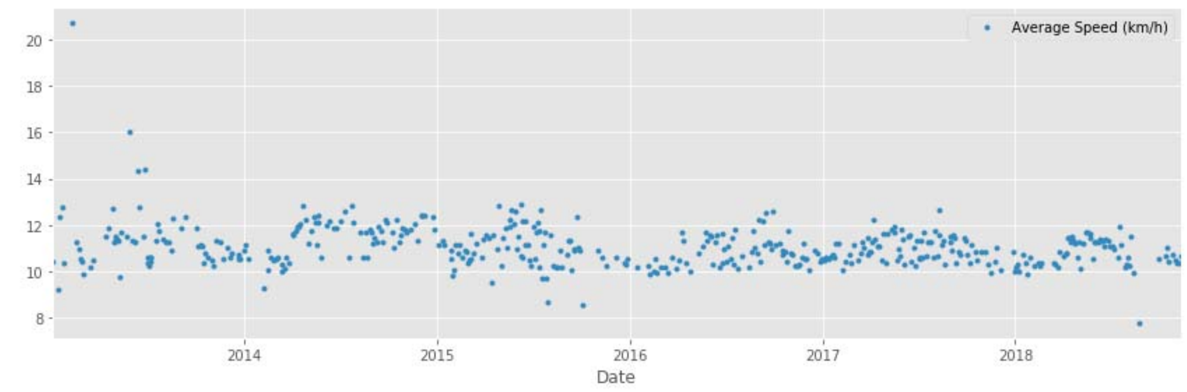
4. Plot running data

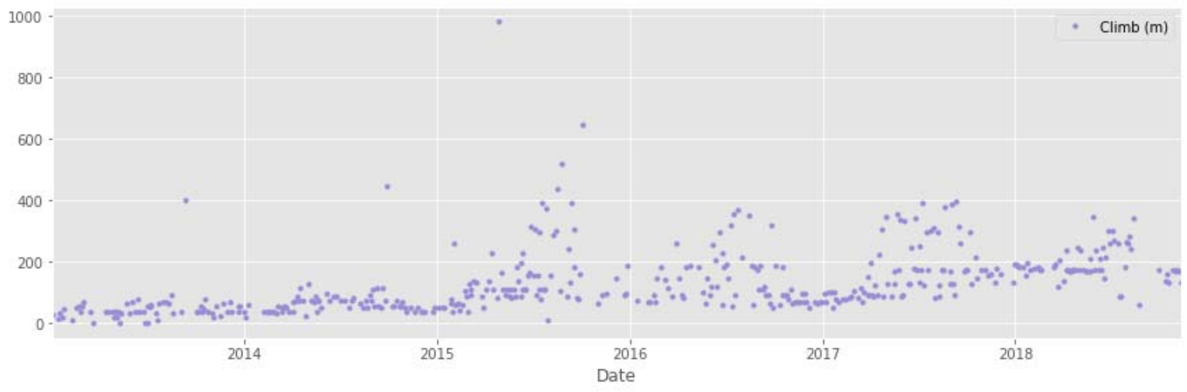
Now we can create our first plot! As we found earlier, most of the activities in my data were running (459 of them to be exact). There are only 29, 18, and two instances for cycling, walking, and unicycling, respectively. So for now, let's focus on plotting the different running metrics.

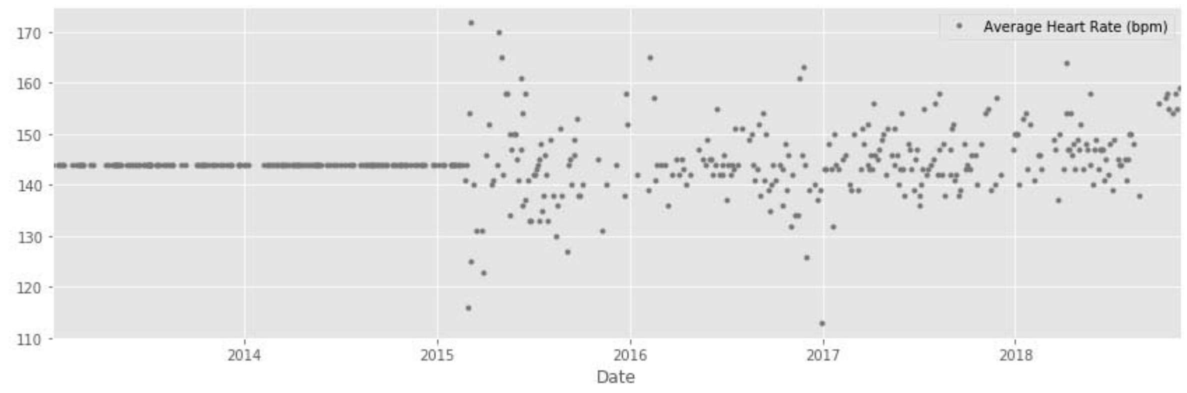
An excellent first visualization is a figure with four subplots, one for each running metric (each numerical column). Each subplot will have a different y-axis, which is explained in each legend. The x-axis, `Date` , is shared among all subplots.

In [4]:
```
%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_lay
out, so results might be incorrect.')
)

# Prepare data subsetting period from 2013 till 2018
runs_subset_2013_2018 = df_run['2018':'2013']

# 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()
```


Distance (km)
Date
2014
2015
2016
2017
2018


Average Speed (km/h)
Date


Climb (m)
Date


Average Heart Rate (bpm)
Date

In [241]:
```
%%nose
# %%nose needs to be included at the beginning of every @tests cell

# One or more tests of the student's code
# The @solution should pass the tests
# The purpose of the tests is to try to catch common errors and
# to give the student a hint on how to resolve these errors

def strip_comment_lines(cell_input):
    """Returns cell input string with comment lines removed."""
    return '\n'.join(line for line in cell_input.splitlines() if not line.star
tswith('#'))

last_input = strip_comment_lines(In[-2])

# def test_set_style():
#     assert "plt.style.use('ggplot')" in last_input, \
#         "We expected the plt.style.use('ggplot') in your input."

def test_subset_correct():
    correct_subset = df_run['2018':'2013']
    assert correct_subset.equals(runs_subset_2013_2018), \
    "The data in runs_subset_2013_2018 is not what we expected."

def test_subplots_param():
    assert "subplots=True" in last_input, \
        "We expected the subplots=True in your input WITHOUT whitespaces aroun
d '=' sign."

def test_plt_show():
    assert "plt.show()" in last_input, \
        "We expected plt.show() in your input."
```

Out[241]:
```
3/3 tests passed
```

5. Running statistics

No doubt, running helps people stay mentally and physically healthy and productive at any age. And it is great fun! 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 my 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.

In [5]:
```
# Prepare running data for the last 4 years
runs_subset_2015_2018 = df_run['2018':'2015']

# 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('W').c
ount().mean()
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: 1.5
```

In [243]:
```
%%nose
# %%nose needs to be included at the beginning of every @tests cell

# One or more tests of the student's code
# The @solution should pass the tests
# The purpose of the tests is to try to catch common errors and
# to give the student a hint on how to resolve these errors

last_output = _

def strip_comment_lines(cell_input):
    """Returns cell input string with comment lines removed."""
    return '\n'.join(line for line in cell_input.splitlines() if not line.star
tswith('#'))

last_input = strip_comment_lines(In[-2])

def test_subset_correct5():
    correct_subset = df_run['2018':'2015']
    assert correct_subset.equals(runs_subset_2015_2018), \
    "The data in runs_subset_2015_2018 is not what we expected."

def test_annual_stats():
    assert "resample('A').mean()" in last_input, \
    "Did you use resample('A').mean() to count annual averages for each year?"

def test_weekly_average():
    assert "resample('W').mean().mean()" in last_input, \
    "Did you use resample('W').mean().mean() to count average weekly statistic
s?"

def test_weekly_count_syntax():
    assert "runs_subset_2015_2018['Distance (km)']" in last_input, \
    "Did you filter column 'Distance (km)' from data subset using single quote
s?"

def test_weekly_count():
    assert weekly_counts_average == 1.5, \
    "We expected 1.5 trainings per week on average."
```

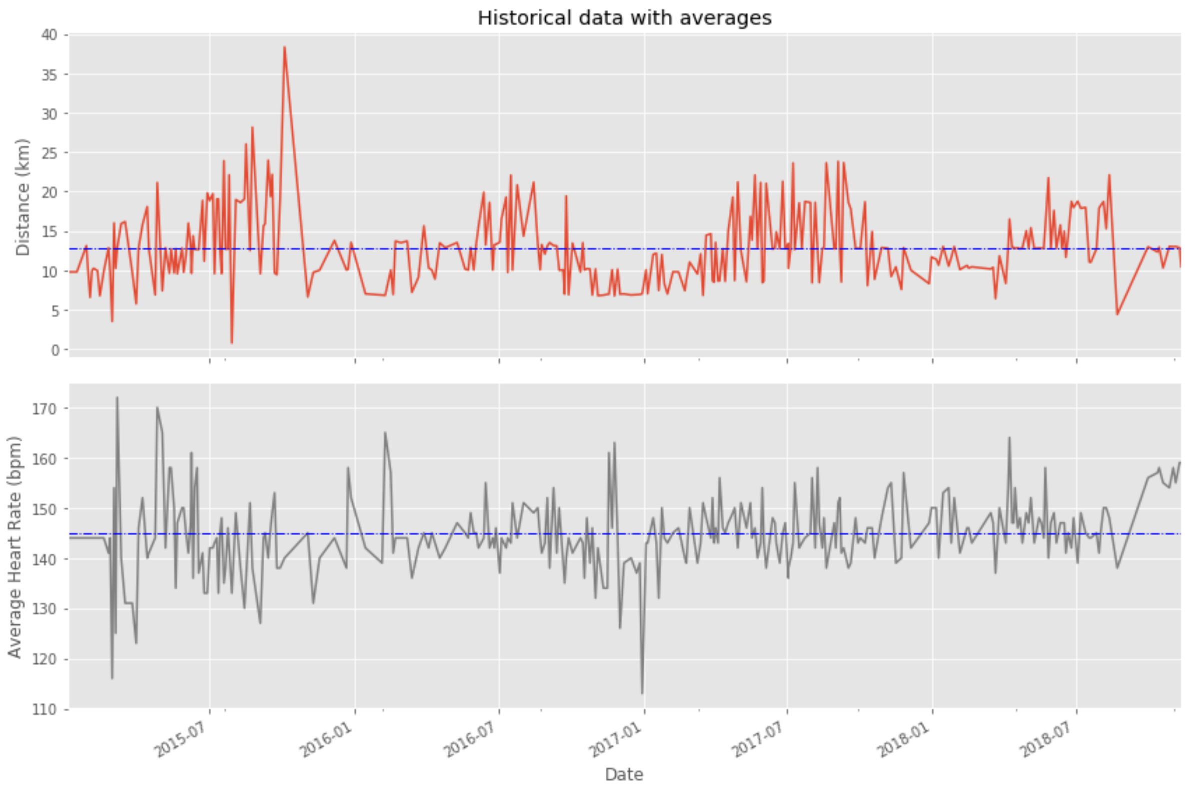
Out[243]:
```
5/5 tests passed
```

6. Visualization with averages

Let's plot the long term averages of my distance run and my 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.

In [6]:
```
# 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_distance.plot(ax=ax1)
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()
```



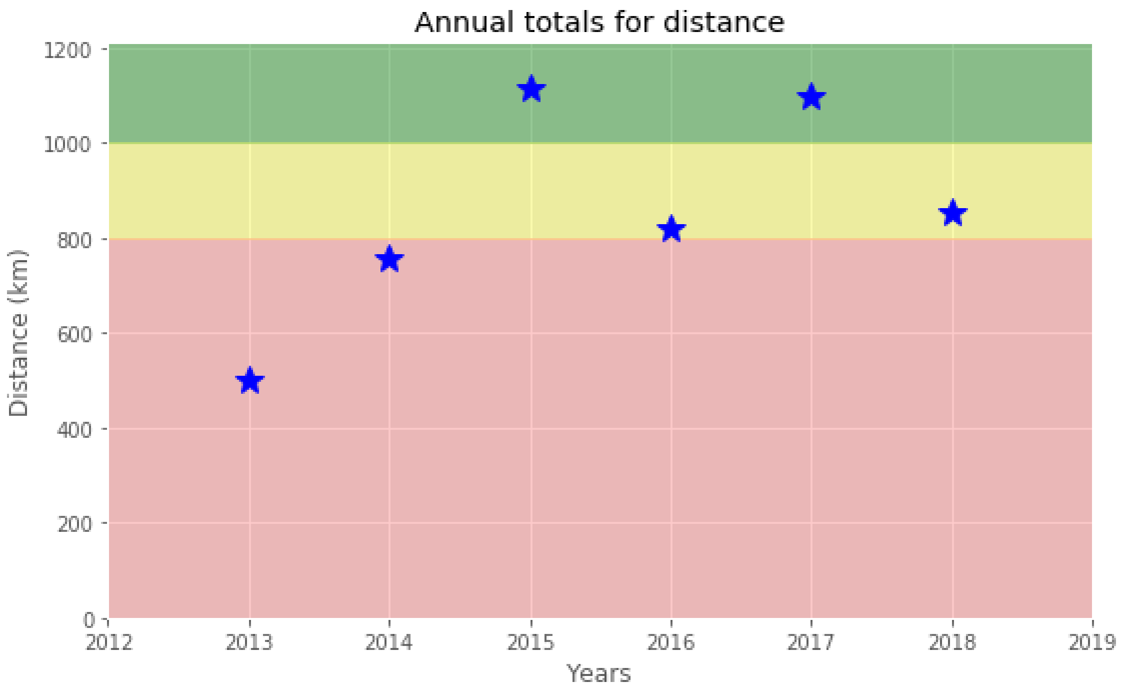
In [245]:
```
%%nose
# %%nose needs to be included at the beginning of every @tests cell

# One or more tests of the student's code
# The @solution should pass the tests
# The purpose of the tests is to try to catch common errors and
# to give the student a hint on how to resolve these errors

def test_data_correct6():
    cor_runs_distance = runs_subset_2015_2018['Distance (km)']
    cor_runs_hr = runs_subset_2015_2018['Average Heart Rate (bpm)']
    assert (cor_runs_distance.equals(runs_distance) & cor_runs_hr.equals(runs_
hr)), \
    "Did you use the correct column names to select distance and heart rate?"

def test_plot_exist6():
    assert (ax2.get_geometry()[0] == 2 and fig.get_figwidth() == 12), \
    "The size of plot and the number of subplots are not as expected."

def test_first_subplot6():
    assert all(ax1.lines[0].get_ydata() == runs_distance.sort_index()), \
    "Did you use runs_distance.plot(ax=ax1) to plot the first plot?"

def test_horline6():
    assert ax2.lines[1].get_ydata()[0] == runs_hr.mean(), \
    "The value of average heart rate is incorrect on the plot."
```

Out[245]:
```
4/4 tests passed
```

7. Did I reach my goals?

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

In [7]:
```
# Prepare data
df_run_dist_annual = df_run['2018':'2013']['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='bl
ue')
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()
```



In [247]:
```
%%nose
# %%nose needs to be included at the beginning of every @tests cell

# One or more tests of the student's code
# The @solution should pass the tests
# The purpose of the tests is to try to catch common errors and
# to give the student a hint on how to resolve these errors

def strip_comment_lines(cell_input):
    """Returns cell input string with comment lines removed."""
    return '\n'.join(line for line in cell_input.splitlines() if not line.star
tswith('#'))

last_input = strip_comment_lines(In[-2])

def test_data_correct7():
    cor_run_dist_annual = df_run['2018':'2013']['Distance (km)'].resample
('A').sum()
    assert cor_run_dist_annual.equals(df_run_dist_annual), \
    "The data in df_run_dist_annual is not what we expected."

def test_figsize7():
    assert (fig.get_figwidth() == 8.0 and fig.get_figheight() == 5.0), \
    "The figure size is incorrect. Did you set figsize=(8, 5) in plt.figure
()?"

def test_axspan7():
    assert len(ax.patches) == 3, \
    "Did you use ax.axhspan(0, 800, ...)?"

def test_show7():
    assert 'plt.show()' in last_input , \
    "We expected plt.show() in your input."
```

Out[247]:
```
4/4 tests passed
```

8. Am I progressing?

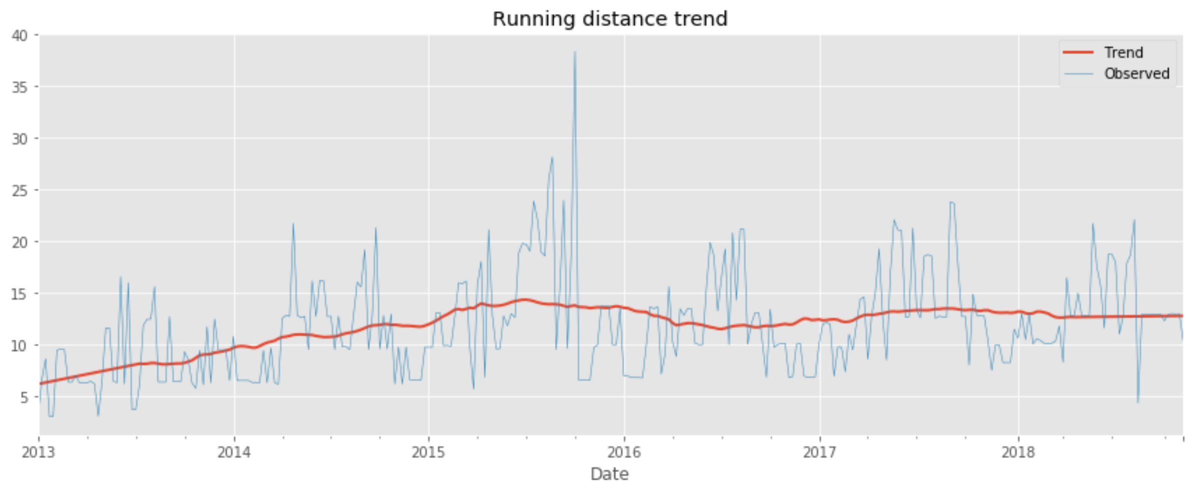
Let's dive a little deeper into the data to answer a tricky question: am I progressing in terms of my running skills?

To answer this question, we'll decompose my 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.

In [8]:
```
# Import required library
import statsmodels.api as sm

# Prepare data
df_run_dist_wkly = df_run['2018':'2013']['Distance (km)'].resample('W').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()
```

```
/usr/local/lib/python3.6/dist-packages/matplotlib/figure.py:2369: UserWarnin
g: This figure includes Axes that are not compatible with tight_layout, so re
sults might be incorrect.
  warnings.warn("This figure includes Axes that are not compatible "
```



In [249]:
```
%%nose
'''
def strip_comment_lines(cell_input):
    """Returns cell input string with comment lines removed."""
    return '\n'.join(line for line in cell_input.splitlines() if not line.star
tswith('#'))

last_input = strip_comment_lines(In[-2])
'''

def test_sm_imported():
    assert 'sm' in globals(), \
    "Did you import statsmodels.api aliased as sm?"

def test_data_correct8():
    cor_run_dist_wkly = df_run['2018':'2013']['Distance (km)'].resample('W').b
fill()
    assert cor_run_dist_wkly.equals(df_run_dist_wkly), \
    "The data in df_run_dist_wkly is not what we expected."

def test_figsize8():
    assert (fig.get_figwidth() == 12.0 and fig.get_figheight() == 5.0), \
    "The figure size is incorrect. Did you set figsize=(12, 5) in plt.figure
()?"
```

Out[249]:
```
3/3 tests passed
```

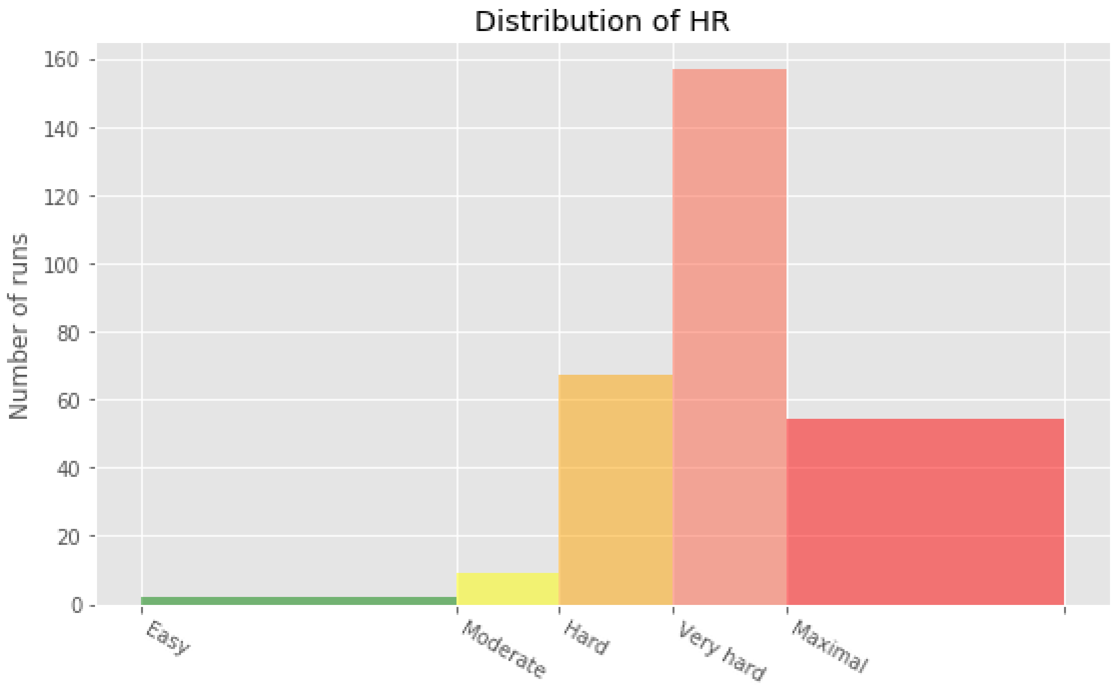
9. Training intensity

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 my heart rate data by training intensity. It will be a visual presentation for the number of activities from predefined training zones.

In [9]:
```python
# 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':'2015-03']['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()
```

```
/usr/local/lib/python3.6/dist-packages/matplotlib/figure.py:2369: UserWarnin
g: This figure includes Axes that are not compatible with tight_layout, so re
sults might be incorrect.
  warnings.warn("This figure includes Axes that are not compatible "
```



In [251]:
```
%%nose

def strip_comment_lines(cell_input):
    """Returns cell input string with comment lines removed."""
    return '\n'.join(line for line in cell_input.splitlines() if not line.star
tswith('#'))

last_input = strip_comment_lines(In[-2])

def test_data_correct9():
    corr_data = df_run['2018':'2015-03']['Average Heart Rate (bpm)']
    assert corr_data.equals(df_run_hr_all), \
    "The data in df_run_hr_all is not what we expected."

def test_subplots_used9():
    assert 'plt.subplots' in last_input, \
    "We expected plt.subplots in your input."

def test_figsize9():
    assert (fig.get_figwidth() == 8.0 and fig.get_figheight() == 5.0), \
    "The figure size is incorrect. Did you set figsize=(8, 5) in plt.subplots
()?"

def test_xticklabels9():
    lbls = [lb.get_text() for lb in ax.get_xticklabels()]
    del lbls[-1]
    assert lbls == zone_names, \
    "The x-axis labels are incorrect. Did you use ax.set_xticklabels(labels=zo
ne_names, ...) ?"

def test_show9():
    assert 'plt.show()' in last_input , \
    "We expected plt.show() in your input."
```

Out[251]:
```
5/5 tests passed
```

10. Detailed summary report

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

In [10]:
```
# Concatenating three DataFrames
df_run_walk_cycle = df_run.append(df_walk).append(df_cycle).sort_index(ascendi
ng=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')[dist_climb_cols].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].de
scribe()

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

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

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

Type	Distance (km)	Climb (m)
Cycling	680.58	6976
Running	5224.50	57278
Walking	33.45	349

```
Summary statistics for different training types:
```

Out[10]:

		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

In [253]:
```
%%nose
# %%nose needs to be included at the beginning of every @tests cell

# One or more tests of the student's code
# The @solution should pass the tests
# The purpose of the tests is to try to catch common errors and
# to give the student a hint on how to resolve these errors

def strip_comment_lines(cell_input):
    """Returns cell input string with comment lines removed."""
    return '\n'.join(line for line in cell_input.splitlines() if not line.star
tswith('#'))

last_input = strip_comment_lines(In[-2])
last_output = _

def test_data_correct10():
    corr_data = df_run.append(df_walk).append(df_cycle).sort_index(ascending=F
alse)
    assert corr_data.equals(df_run_walk_cycle), \
    "The data in df_run_walk_cycle is not what we expected. Did you apply sort
_index(ascending=False) ?"

def test_display_totals10():
    assert 'display(df_totals)' in last_input, \
    "Did you use display(df_totals) to show the calculations?"

def test_totals10():
    corr_tot = df_totals['Distance (km)']['Running'].round(1)
    assert corr_tot == 5224.5, \
    "We expected different values for totals. You should see 5224.5 km as tota
l distance."

def test_stack_used10():
    assert 'stack()' in last_input, \
    "Did you use stack() to create a compact view of the results?"
```

Out[253]:
```
4/4 tests passed
```

11. Fun facts

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

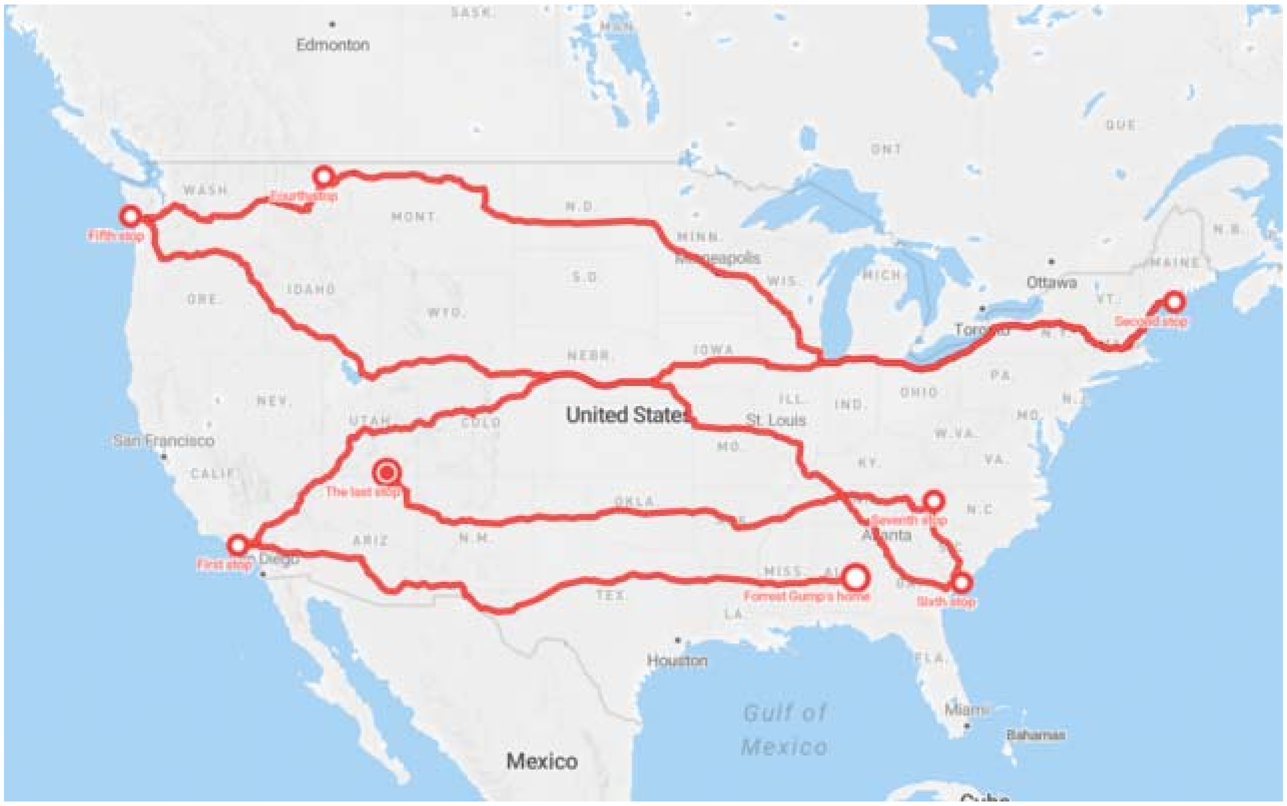
These data (my running history) represent 6 years, 2 months and 21 days. And I remember how many running shoes I went through–7.

```
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
- Number of running shoes gone through: 7 pairs
```

The story of Forrest Gump is well known–the man, who for no particular reason decided to go for a "little run." His epic run duration was 3 years, 2 months and 14 days (1169 days). In the picture you can see Forrest’s route of 24,700 km.

```
FORREST RUN FACTS
- Average distance: 21.13 km
- Total number of km run: 24,700 km
- Total runs: 1169
- Number of running shoes gone through: ...
```

Assuming Forest and I go through running shoes at the same rate, figure out how many pairs of shoes Forrest needed for his run.



In [11]:
```python
# Count average shoes per lifetime (as km per pair) using our fun facts
average_shoes_lifetime = 5224/7

# Count number of shoes for Forrest's run distance
shoes_for_forrest_run = 24700 // average_shoes_lifetime

print('Forrest Gump would need {} pairs of shoes!'.format(shoes_for_forrest_ru
n))
```

```
Forrest Gump would need 33.0 pairs of shoes!
```