Final Report of Traineeship Program 2024

On

“Analyzing Fitness Data”

MEDTOUREASY

27th March 2024

Aly Saleh

https://github.com/AlyHSaleh/MTE-Fitness-Data-Analysis

ACKNOWLDEGMENTS

The traineeship opportunity that I had with MedTourEasy was a great change for learning and understanding the intricacies of the subject of Data Visualizations in Data Science; and also, for personal as well as professional development. I am very obliged for having a chance to interact with so many professionals who guided me throughout the traineeship project and made it a great learning curve for me.

Firstly, I express my deepest gratitude and special thanks to the Training & Development Team of MedTourEasy who gave me an opportunity to carry out my traineeship at their esteemed organization. Also, I express my thanks to the team for making me understand the details of the Data Science profile and training me in the same so that I can carry out the project properly and with maximum client satisfactionand also for spearing his valuable time in spite of his busy schedule.

I would also like to thank the team of MedTourEasy and my colleagues who made theworking environment productive and very conducive.

TABLE OF CONTENTS

About MedTourEasy

MedTourEasy, a global healthcare company, provides you the informational resources needed to evaluate your global options. It helps you find the right healthcare solution based on specific health needs, affordable care while meeting the quality standards that you expect to have in healthcare. MedTourEasy improves access to healthcare for people everywhere. It is an easy-to-use platform and service that helps patients to get medical second opinions and to schedule affordable, high quality medical treatment abroad.

Project Description

With the explosion in fitness tracker popularity, runners all of the world are collecting data with gadgets (smartphones, watches, etc.) to keep themselves motivated. They look for answers to questions like:

- How fast, long, and intense was my run today?
- Have I succeeded with my training goals?
- Am I progressing?
- What were my best achievements?
- How do I perform compared to others?

This data was exported from Runkeeper. The data is a CSV file where each row is a single training activity. In this project, you'll create import, clean, and analyze my data to answer the above questions. You can then apply the same strategy to your training data if you wish!

Project Tasks

- Obtain and review raw data
- Data preprocessing
- Dealing with missing values
- Plot running data
- Running statistics
- Visualization with averages
- Did I reach my goals?
- Am I progressing?
- Training intensity
- Detailed summary report
- Fun Facts

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 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.

Task 1: Instructions

Load `pandas` and the training activities data.

- Import `pandas` under the alias `pd`.
- Use the `read_csv()` function to load the dataset (`runkeeper_file`) into a variable called `df_activities`. Parse the dates with the `parse_dates` parameter and set the index to the `Date` column using the `index_col` parameter.
- Display 3 random rows from `df_activities` using the `sample()` method.
- Print a summary of `df_activities` using the `info()` method.

```python
# Import pandas
# ... YOUR CODE FOR TASK 1 ...
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=['Date'], index_col='Date')

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

# Print DataFrame summary
# ... YOUR CODE FOR TASK 1 ...
df_activities.info()
```

Output:

Date	Activity Id	Type	Route Name	Distance (km)	Duration	Average Pace	Average Speed (km/h)	Calories Burned	Climb (m)	Average Heart Rate (bpm)	Friend's Tagged	Notes	GPX File
2018-01-21 11:46:12	2a24b579-7365-49da-a4e4-67c13d097e39	Running	NaN	10.52	1:00:59	5:48	10.35	744.0	153	154.0	NaN	TomTom MySports Watch	2018-01-21-114612.gpx
2015-03-16 18:23:38	760c4a61-d435-47ca-a787-005202a803df	Running	NaN	16.15	1:26:37	5:22	11.19	1118.0	133	131.0	NaN	NaN	2015-03-16-182338.gpx
2014-06-04 18:07:39	0f94765c-b3ea-4400-9dbf-3f6aa534b94b	Running	NaN	16.17	1:20:57	5:00	11.98	1146.0	93	NaN	NaN	NaN	2014-06-04-180739.gpx

```
<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
```

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.

Task 2: Instructions

Implement the following data preprocessing tasks:

- Delete unnecessary columns from `df_activities` with the `drop()` method, setting the `columns` parameter to the `cols_to_drop` list.
- Calculate the activity type counts using the `value_counts()` method on the `Type` column.
- Rename the 'Other' values to 'Unicycling' in the `Type` column using `str.replace()`.
- Count the missing values in each column using `isnull().sum()`.

```python
# 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
# ... YOUR CODE FOR TASK 2 ...
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'].str.replace('Other', 'Unicycling', regex=False)

# Count missing values for each column
# ... YOUR CODE FOR TASK 2 ...
df_activities.isnull().sum(axis=0)
```

Output:

```
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 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.

Task 3: Instructions

Implement mean imputation for missing values.

- Calculate the sample mean for `Average Heart Rate (bpm)` for the 'Cycling' activity type. Assign the result to `avg_hr_cycle`.
- Filter the `df_activities` for the 'Cycling' activity type. Create a copy of the result using `copy()` and assign the copy to `df_cycle`.
- Fill in the missing values for `Average Heart Rate (bpm)` in `df_cycle` with `int(avg_hr_cycle)` using the `fillna()` method.
- Count the missing values for all columns in `df_run`.

```python
# 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)
# ... YOUR CODE FOR TASK 3 ...
df_cycle['Average Heart Rate (bpm)'].fillna(int(avg_hr_cycle), inplace=True)

# Count missing values for each column in running data
# ... YOUR CODE FOR TASK 3 ...
df_run.isnull().sum()
```

Output:

```
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 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.

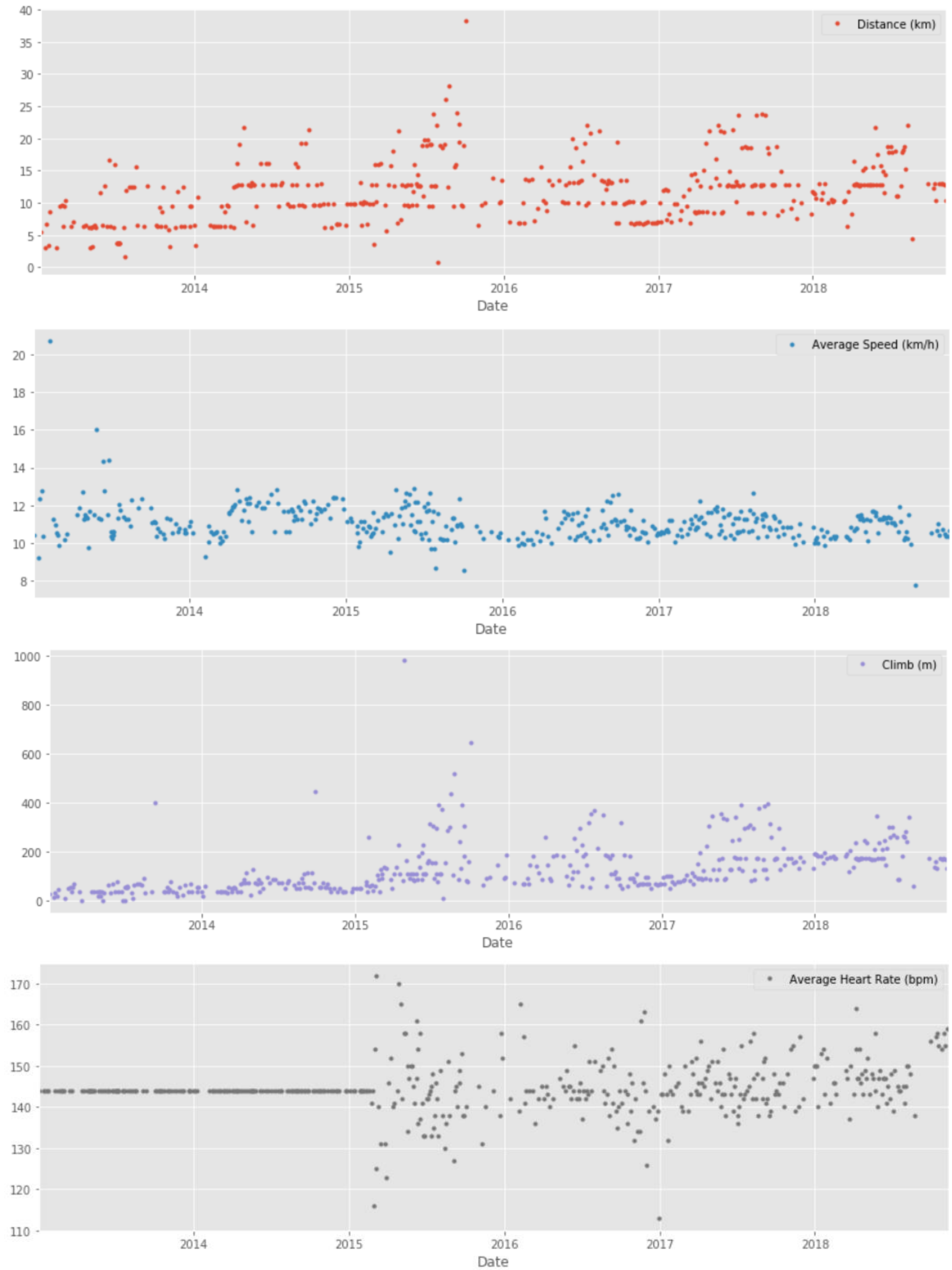
An excellent first visualization is a figure with four subplots, one for each running metric (each numerical column). Each subplot will have a different y-axis, which is explained in each legend. The x-axis, Date, is shared among all subplots.

Task 4: Instructions

Plot running data from 2013 through 2018.

- Subset `df_run` for data from 2013 through 2018. Take into account that observations in dataset stored in chronological order - most recent records first. Assign the result to `runs_subset_2013_2018`.
- In the plotting code, enable subplots by setting the `subplots` parameter to `True`. Don't use spaces around the `=` sign when used to indicate a keyword argument, as recommended in PEP 8 style guide for Python code.
- Show the plot using `plt.show()`.

```
%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['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
# ... YOUR CODE FOR TASK 4 ...
plt.show()
```


Distance (km)
Average Speed (km/h)
Climb (m)
Average Heart Rate (bpm)
Date
2014
2015
2016
2017
2018

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.

Task 5: Instructions

Calculate annual and weekly means for `Distance (km)`, `Average Speed (km/h)`, `Climb (m)` and `Average Heart Rate (bpm)`.

- Subset `df_run` for data from 2015 through 2018. Assign the result to `runs_subset_2015_2018`.
- Count the annual averages using `resample()` with 'A' alias, and the `mean()` method for `runs_subset_2015_2018`.
- Count the average weekly statistics using `resample()` with 'W' alias, and the `mean()` method twice.
- Filter from dataset column `Distance (km)` and count the average number of trainings per week using `resample()` with the `count()` and `mean()` methods. Assign the result to `weekly_counts_average`.

```python
# 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').count().mean()
print('How many trainings per week I had on average:', weekly_counts_average)
```

Output:

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

Date	Distance (km)	Average Speed (km/h)	Climb (m)	Average Heart Rate (bpm)
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
```

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.

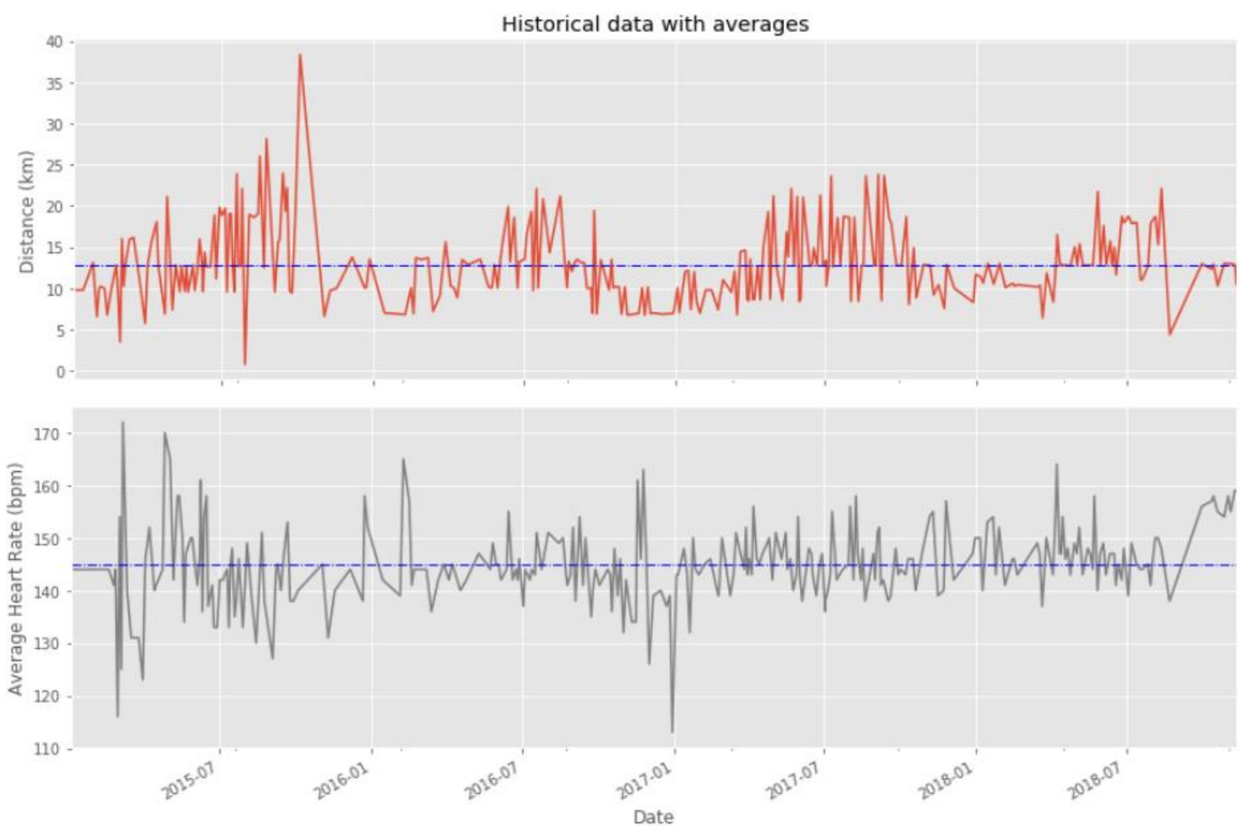
In this task, we will use matplotlib functionality for plot creation and customization.

Task 6: Instructions

Prepare data and create a plot.

- Select information for distance and then for heart rate from `runs_subset_2015_2018` and assign to `runs_distance` and `runs_hr`, respectively.
- Create two subplots with shared x-axis using the `plt.subplots()` method, setting the first positional parameter to 2, `sharex` to `True`, and `figsize` to `(12,8)`. Assign the output to `fig, (ax1, ax2)` variables.
- Plot distance on the first subplot, setting parameter `ax` to `ax1`.
- On the second subplot (`ax2`), add a horizontal line with `axhline()` for the average value of heart rate counted as `runs_hr.mean()`. Set `color` to `'blue'`, `linewidth` to `1`, and `linestyle` to `'-.'`.

```python
# 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
# ... YOUR CODE FOR TASK 6 ...
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)')
# ... YOUR CODE FOR TASK 6 ...
ax2.axhline(runs_hr.mean(), color='blue', linewidth=1, linestyle='-.')

# Show plot
plt.show()
```



7. Did I reach my goals?

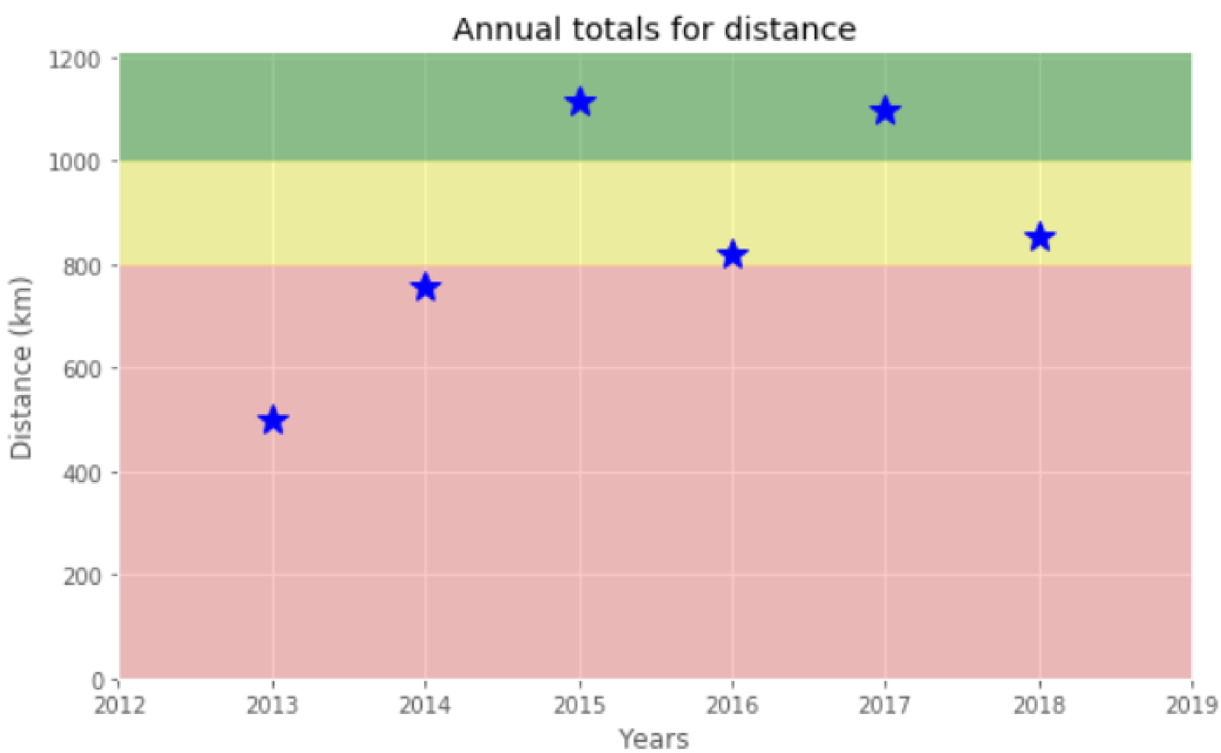
To motivate myself to run regularly, I set a target goal of running 1000 km per year. Let's visualize my annual running distance (km) from 2013 through 2018 to see if I reached my goal each year. Only stars in the green region indicate success.

Task 7: Instructions

Prepare data and create a plot.

- Subset `df_run` for data from 2013 through 2018 and select the `Distance (km)` column. Count annual totals with `resample()` and `sum()`. Assign the result to `df_run_dist_annual`.
- Create a plot with `plt.figure()`, setting `figsize` to define a plot of size 8.0 inches x 5.0 inches.
- Customize the plot with horizontal span from 0 to 800 km with `ax.axhspan()`. Set `color` to `'red'` and `alpha` to `0.2`.
- Show the plot with `plt.show()`.

```
# 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='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)
# ... YOUR CODE FOR TASK 7 ...
ax.axhspan(0, 800, color = 'red', alpha=0.2)

# Show plot
# ... YOUR CODE FOR TASK 7 ...
plt.show()
```



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.

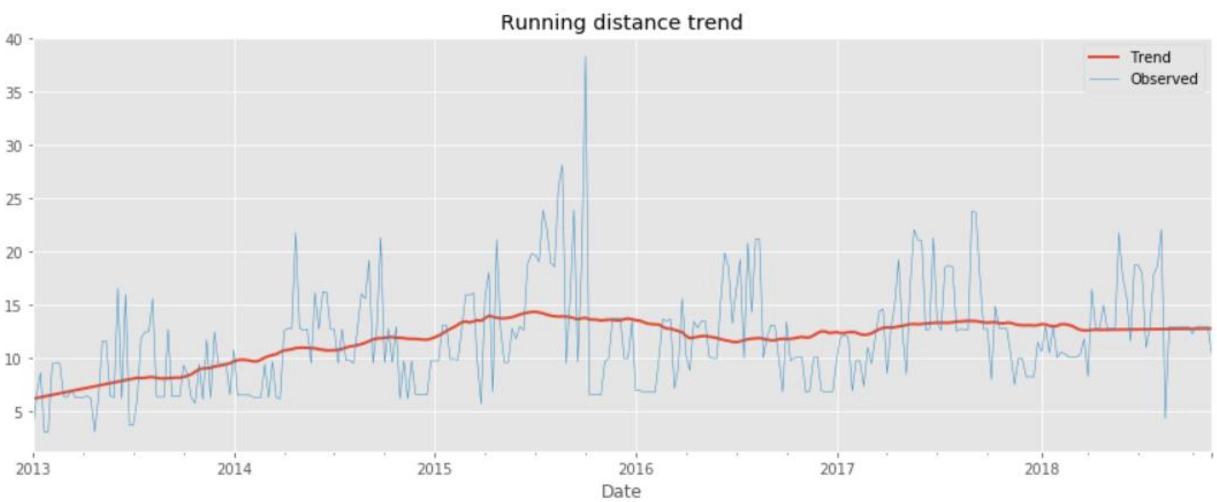
We are going to use statsmodels library to decompose the weekly trend.

Task 8: Instructions

Create a plot with observed distance of runs and decomposed trend.

- Import the `statsmodels.api` under the alias `sm`.
- Subset `df_run` from 2013 through 2018, select `Distance (km)` column, resample weekly, and fill NaN values with the `bfill()` method. Assign to `df_run_dist_wkly`.
- Create a plot with `plt.figure()`, defining plot size by setting `figsize` to `(12,5)`.

```python
# Import required library
# ... YOUR CODE FOR TASK 8 ...
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()
```



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.

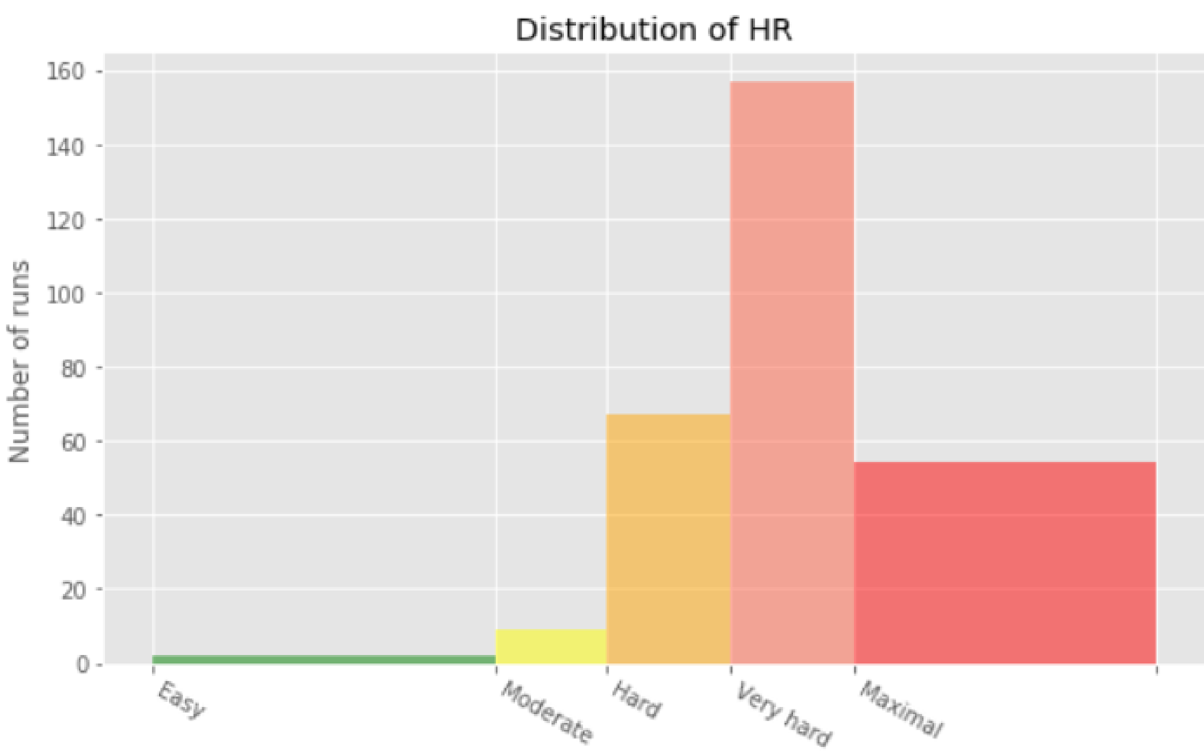
We'll create a distribution plot of my heart rate data by training intensity. It will be a visual presentation for the number of activities from predefined training zones.

Task 9: Instructions

Create a customized histogram for heart rate distribution.

- Subset `df_run` from March 2015 through 2018 then select the `Average Heart Rate (bpm)` column. Assign the result to `df_run_hr_all`.
- Create a plot with `plt.subplots()`, setting `figsize` to `(8,5)`. Assign the result to `fig, ax`.
- Create customized x-axis ticks with `ax.set_xticklabels()`. Set the parameters `labels` to `zone_names`, `rotation` to `-30`, and `ha` to `'left'`.
- Show the plot with `plt.show()`.

```
# 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.loc['20190101':'20150301', '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)
# ... YOUR CODE FOR TASK 9 ...
ax.set_xticklabels(labels=zone_names, rotation=-30, ha='left')

# Show plot
# ... YOUR CODE FOR TASK 9 ...
plt.show()
```



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.

Task 10: Instructions

Create a summary report.

- Concatenate the `df_run` DataFrame with `df_walk` and `df_cycle` using `append()`, then sort based on the index in descending order. Assign the result to `df_run_walk_cycle`.
- Group `df_run_walk_cycle` by activity type, then select the columns in `dist_climb_cols`. Sum the result using `sum()`. Assign the result to `df_totals`.
- Use the `stack()` method on `df_summary` to show a compact reshaped form of the full summary report.

```python
# Concatenating three DataFrames
df_run_walk_cycle = df_run.append([df_walk, df_cycle]).sort_index(ascending=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].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:')
# ... YOUR CODE FOR TASK 10 ...
df_summary.stack()
```

Output:

```
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:
```

Type		Average Speed (km/h)	Climb (m)	Distance (km)
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.

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: …

Task 11: Instructions

Use FUN FACTS data to answer some fun questions.

- Calculate the instructor's average shoes per lifetime. Use number of 'Total number of km run' from FUN FACTS and divide by the number of pairs of shoes gone through.
- Calculate an estimated number of shoes gone through for Forrest Gump's route. Use 'Total number of km run' from FORREST RUN FACTS, then divide (using floor division) by the result from the previous step.

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


```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 = int(24700 / average_shoes_lifetime)

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

```
Forrest Gump would need 33 pairs of shoes!
```

CONCLUSION AND FUTURE SCOPE

The Runkeeper Fitness Data Analysis project delivers personalized insights into individual fitness behaviors using data from the Runkeeper app. Through data cleaning, exploratory analysis, and machine learning, it offers tailored feedback for users and contributes to wider research on exercise habits.

Potential areas for expansion include integrating more data sources, improving predictive modeling, providing customized workout suggestions, adding social features, and developing a mobile app for instant feedback. These advancements would enhance user experience and advance research in exercise science and health promotion.