FINAL REPORT OF INTERNSHIP PROGRAM APRIL 2021

ON

"Analysis of Fitness Data"

MEDTOUREASY

27th April 2021

by: Madhavesh vishwakarma

ACKNOWLEDGEMENT

The internship opportunity that I had with MedTourEasy was a great change for learning and understanding the intricacies of the subject of Data Visualizations in Data Analytics; 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 internship project and made it a great learning curve for me. I would also like to thank the team of MedTourEasy and my colleagues who made the working environment productive and very conducive. 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 internship at their esteemed organization. Also, I express my thanks to the team for making me understand the details of the Data Analytics profile and training me in the same so that I can carry out the project properly and with maximum client satisfaction and also for spearing his valuable time in spite of his busy schedule.

TABLE OF CONTENTS

ABSTRACT

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 analyse 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 analyse their data like we will in this notebook.

After logging your run, the first step is to export the data from Runkeeper. Then import the data and start exploring to find potential problems. After that, create data cleaning strategies to fix the issues. Finally, analyse 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.

This project aims at analysing the data i exported from the Runkeeper App.

1. INTRODUCTION

1.1 About the Company

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.

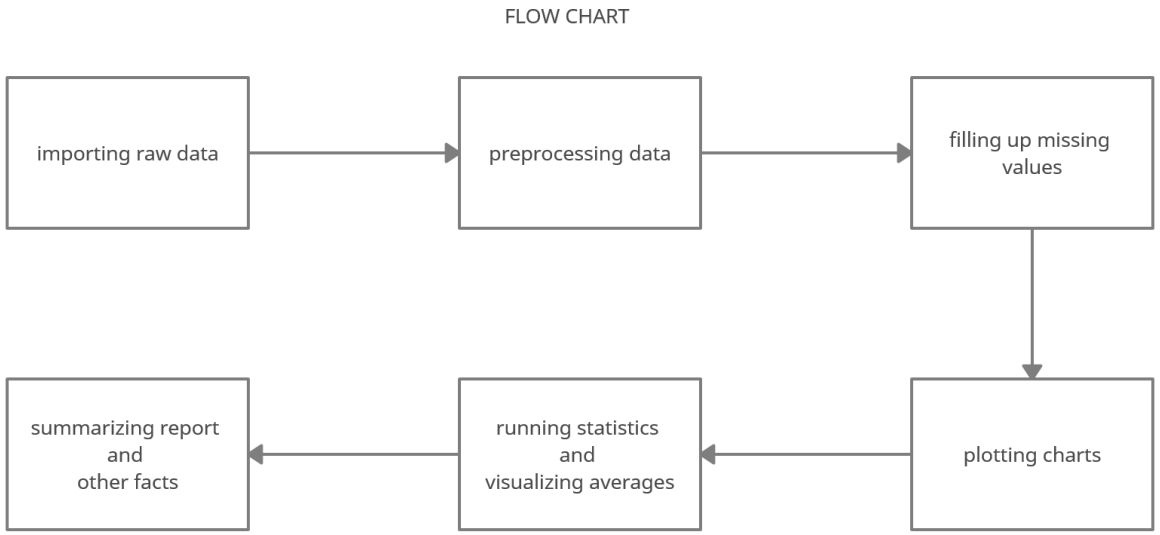
1.2 About the Project

this project aims at analysing fitness data from runkeeper app that tracks data of the user regarding their fitness activities. Throughout the notebook I have analysed my summary, my average progress throughout the years, as well as my training intensity.

2. METHODOLOGY

2.1 FLOW CHART

The project followed the flow shown in the figure below



2.2 CODING LANGUAGE

This project was done using Python language. Python is an interpreted, high-level and general-purpose programming language. Python's design philosophy emphasizes code readability with its notable use of significant indentation. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects.
Python is dynamically-typed and garbage-collected. It supports multiple programming paradigms, including structured (particularly, procedural), object-oriented and functional programming. Python is often described as a "batteries included" language due to its comprehensive standard library. [1]

2.3 PYTHON LIBRARIES USED

2.3.1 PANDAS

pandas is a Python package that provides fast, flexible, and expressive data structures designed to make working with structured (tabular, multidimensional, potentially heterogeneous) and time series data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real world data analysis in Python. Additionally, it has the broader goal of becoming the most powerful and flexible open source data analysis / manipulation tool available in any language. It is already well on its way toward this goal. [2]

2.3.2 MATPLOTLIB

Matplotlib is an amazing visualization library in Python for 2D plots of arrays. Matplotlib is a multi-platform data visualization library built on NumPy arrays and designed to work with the broader SciPy stack. It was introduced by John Hunter in the year 2002.
One of the greatest benefits of visualization is that it allows us visual access to huge amounts of data in easily digestible visuals. Matplotlib consists of several plots like line, bar, scatter, histogram etc. [3]

2.3.3 WARNINGS

Warnings are provided to warn the developer of situations that aren’t necessarily exceptions. Usually, a warning occurs when there is some obsolete of certain programming elements, such as keyword, function or class, etc. A warning in a program is distinct from an error. Python program terminates immediately if an error occurs. Conversely, a warning is not critical. It shows

some message, but the program runs. The `warn()` function defined in the '`warning`' module is used to show warning messages. The warning module is actually a subclass of Exception which is a built-in class in Python. [4]

2.3.4 STATSMODELS

statsmodels is a Python module that provides classes and functions for the estimation of many different statistical models, as well as for conducting statistical tests, and statistical data exploration. An extensive list of result statistics are available for each estimator. The results are tested against existing statistical packages to ensure that they are correct. The package is released under the open source Modified BSD (3-clause) license. The online documentation is hosted at statsmodels.org. [5]

2.3.5 NUMPY

NumPy, which stands for Numerical Python, is a library consisting of multidimensional array objects and a collection of routines for processing those arrays. Using NumPy, mathematical and logical operations on arrays can be performed. This tutorial explains the basics of NumPy such as its architecture and environment. It also discusses the various array functions, types of indexing, etc. An introduction to Matplotlib is also provided. All this is explained with the help of examples for better understanding. [6]

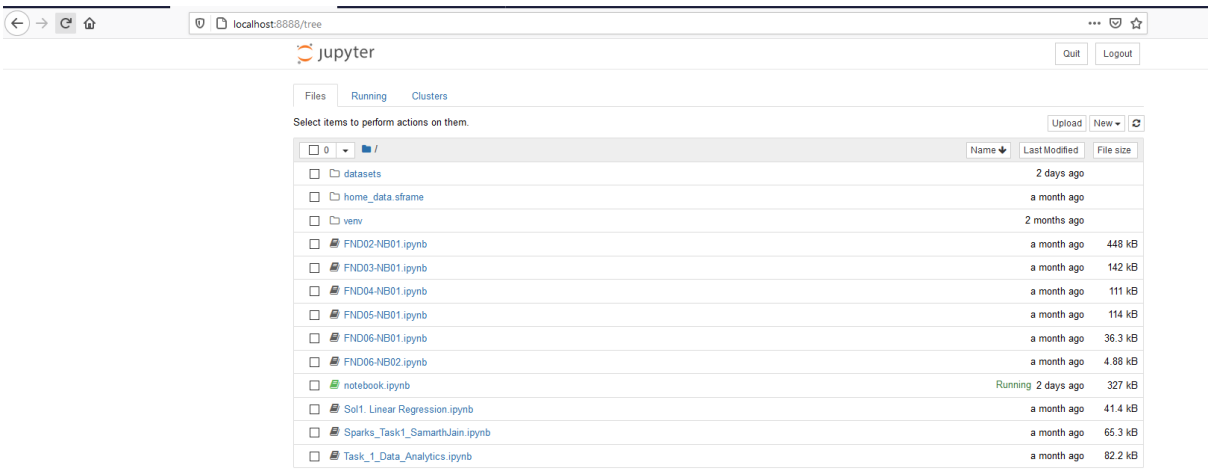
2.3.6 SCIKIT-LEARN

Scikit-learn (Sklearn) is the most useful and robust library for machine learning in Python. It provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction via a consistence interface in Python. This library,

which is largely written in Python, is built upon NumPy, SciPy and Matplotlib. [7]

2.4 PLATFORM

This project was done on a jupyter notebook. The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text. Uses include: data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more. [8]



jupyter notebook opened in a web browser

3. IMPLEMENTATION

3.1 IMPORTING THE DATABASE

the first step is to import the database containing the fitness data. It is stored is a CSV format and hence pandas can easily convert the file into a readable table.
The code to achieve this task is shown below

```
# 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 = 'Date')

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

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

In this code, i have imported pandas library to help work with the dataset. I have stored the dataset into the variable df_activities. I have also printed the table structure information using the ‘info’ method to get an idea of the data.

The results are as follows

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 F
2015-02-09 18:40:18	b76fc1f0-9f7d-49d0-a99f-d7305ba859c2	Running	NaN	9.94	55:22	5:34	10.77	723.2084	61	NaN	NaN	NaN	2015-02-09-184018.g
2013-07-30 18:49:22	e62b5ac5-e900-4c30-9f93-5f6a11da04f8	Running	NaN	12.47	1:05:43	5:16	11.39	893.0000	67	NaN	NaN	NaN	2013-07-30-184922.g
2014-08-28 18:10:00	9642232e-00a5-4e7d-8a92-a8766a7b5b01	Running	NaN	12.82	1:05:45	5:08	11.70	920.0000	70	NaN	NaN	NaN	2014-08-28-181000.g

```
<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):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   Activity Id             508 non-null    object
 1   Type                    508 non-null    object
 2   Route Name              1 non-null      object
 3   Distance (km)           508 non-null    float64
 4   Duration                508 non-null    object
 5   Average Pace            508 non-null    object
 6   Average Speed (km/h)    508 non-null    float64
 7   Calories Burned         508 non-null    float64
 8   Climb (m)               508 non-null    int64
 9   Average Heart Rate (bpm)  294 non-null  float64
 10  Friend's Tagged         0 non-null      float64
 11  Notes                   231 non-null    object
 12  GPX File                504 non-null    object
dtypes: float64(5), int64(1), object(7)
memory usage: 55.6+ KB
None
```

this table shows a sample of 3 rows from the dataset, as well as the database structure..

3.2 data preprocessing

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. In the case of the `Notes` column, it is an optional field that was sometimes left blank. Also, the `Route Name` column was only used once, and the `Friend's Tagged` column was never used.

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.

The code to do it is :-

```
# 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.drop(cols_to_drop, axis=1, inplace = True)
print(df_activities.head())
# Count types of training activities
print(df_activities.value_counts('Type'))
# 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())
```

cols_to_drop variable contains all the extra columns which are not needed. The proceeding lines of code drop the extra columns and display the counts of different types of activity entries.

We have also replaced the ‘other’ types to ‘unicycling’

Finally, we have printed the total number of missing entries in the table

the result is :-

```
                     Type  Distance (km) Duration Average Pace  \
Date
2018-11-11 14:05:12  Running          10.44    58:40         5:37
2018-11-09 15:02:35  Running          12.84  1:14:12         5:47
2018-11-04 16:05:00  Running          13.01  1:15:16         5:47
2018-11-01 14:03:58  Running          12.98  1:14:25         5:44
2018-10-27 17:01:36  Running          13.02  1:12:50         5:36

                    Average Speed (km/h)  Climb (m)  Average Heart Rate (bpm)
Date
2018-11-11 14:05:12                10.68        130                     159.0
2018-11-09 15:02:35                10.39        168                     159.0
2018-11-04 16:05:00                10.37        171                     155.0
2018-11-01 14:03:58                10.47        169                     158.0
2018-10-27 17:01:36                10.73        170                     154.0
Type
Running    459
Cycling     29
Walking     18
Other        2
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
```

this table shows that there are 214 missing values present in the average heart rate column of the table

3.3 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 Data Frames by activity type (`Type`) and calculate each activity's mean heart rate, then fill in the missing values with those means.

The following code will do the job:-

```
# 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
print(df_run.isnull().sum())
```

we have defined 2 variables for running and cycling which store the average amount of time spent on both the activities.

Next, we created 3 different sub databases containing entries of types running, walking and cycling respectively.

Next, we have filled the average values in all 3 databases for that particular type.

Results are:-

```
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
```

the table shows that there are no more missing values, because we have filled up all the missing values with their respective column averages.

3.4 plotting the data

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.

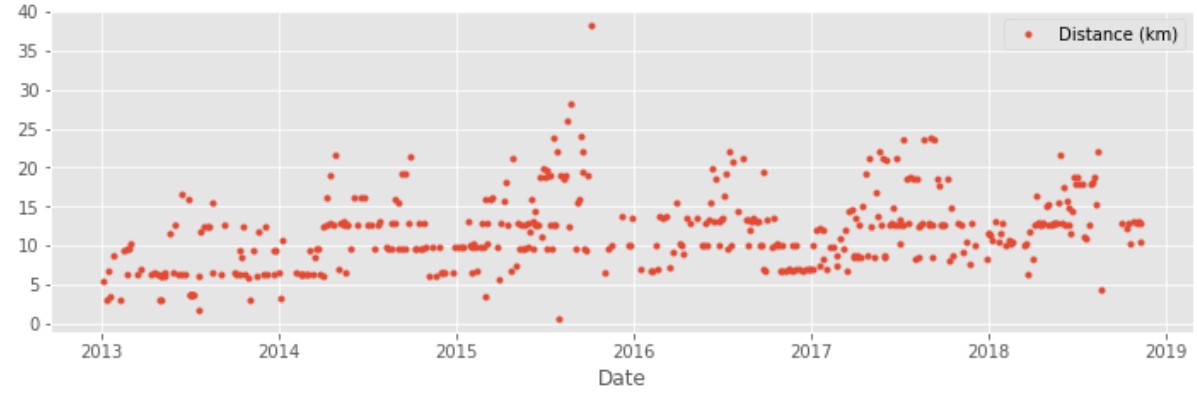
An excellent first visualization is a figure with four subplots, one for each running metric (each numerical column). Each subplot will have a different y-axis, which is explained in each legend. The x-axis, `Date`, is shared among all subplots.

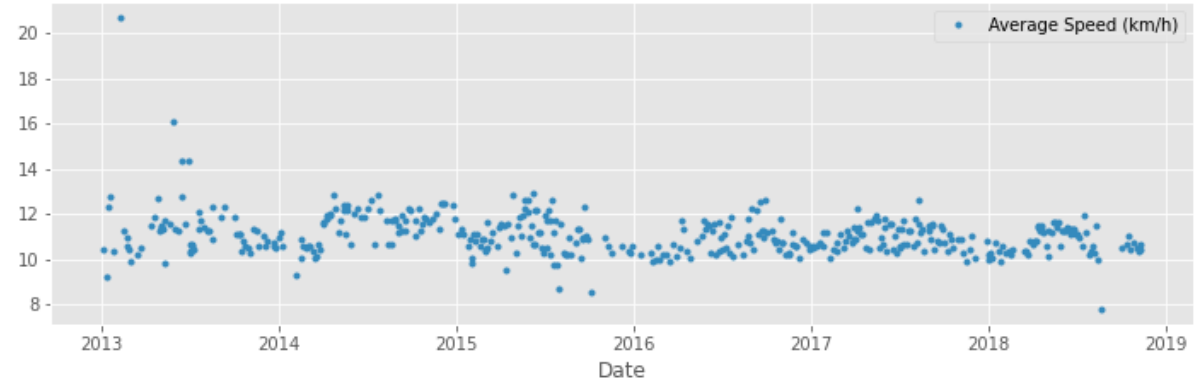
The code to create some plots is:-

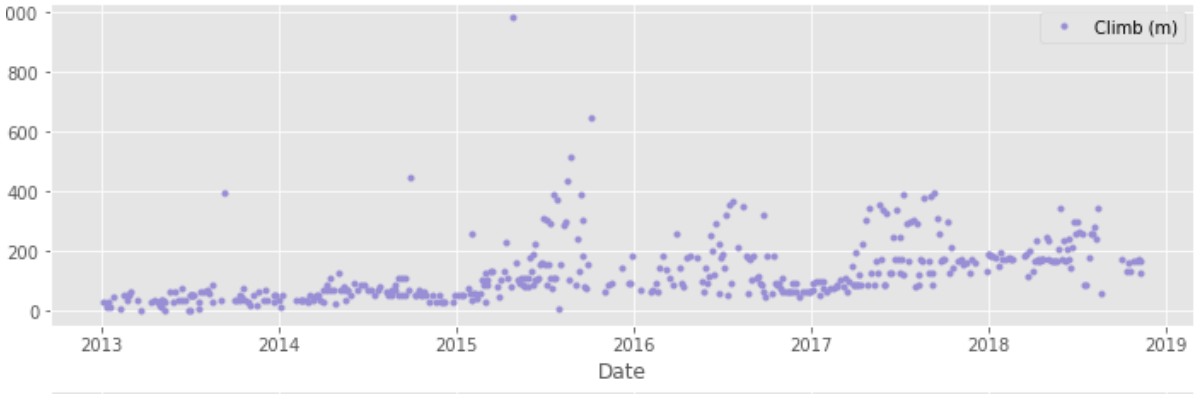
```
%matplotlib inline

# 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.loc['20190101':'20130101']
# 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()
```

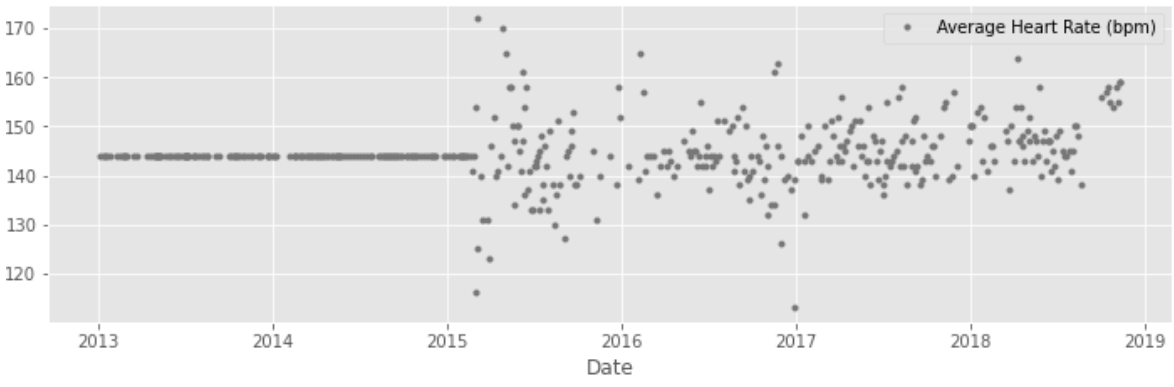
i have imported matplotlib.pyplot to create plots. I then set the plot style to ggplot. Then i have created a subset of the database from year 2013 to 2018 and plotted it using matplotlib.

The resulting plots are:-









3.5 statistics for running

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

If you're with a group of runners 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 dataset. looking 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.

The code for this is:-

```
# Prepare running data for the last 4 years
runs_subset_2015_2018 = df_run.loc['20190101':'20150101']

# 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)
```

we have created a subset of the dataset using the 'loc' method.
Next we have calculated the annual statistics and weekly statistics using the 'resample' method.
Finally we have printed out the results.

The output is;-

```
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
```

this resulting table shows the weekly averages of different statistics

3.6 VISUALISATION WITH AVERAGES

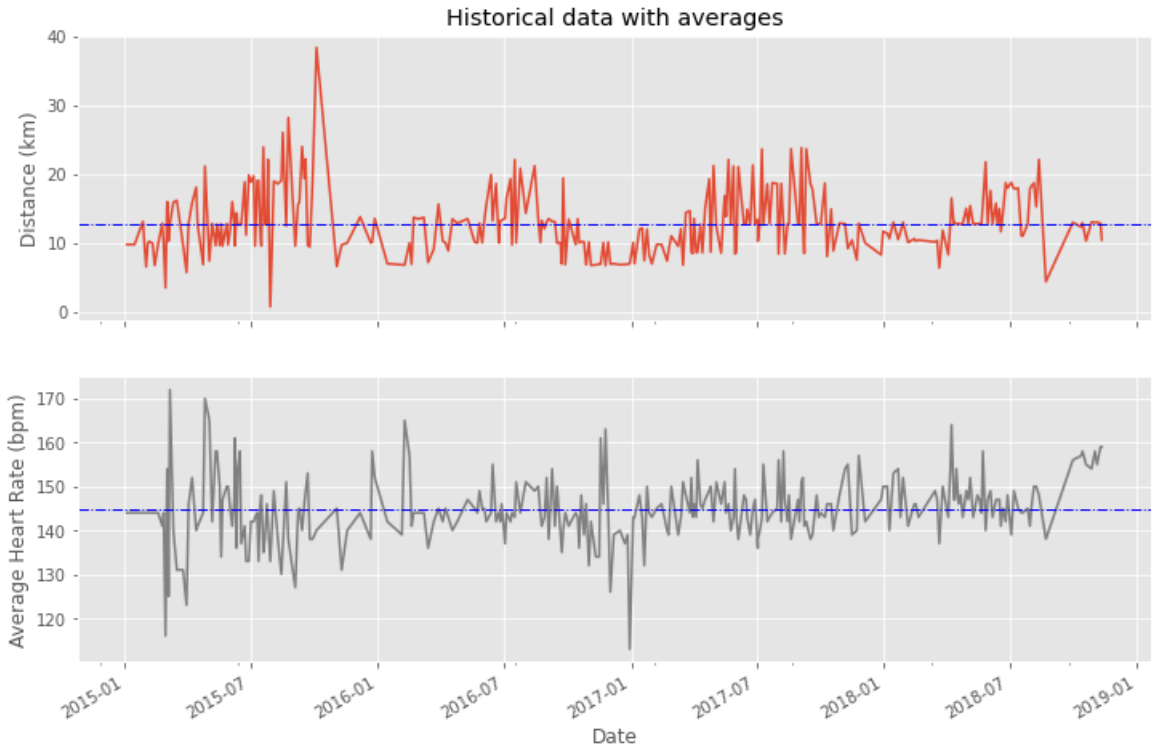
Let's plot the long term averages of the distance run and the heart rate with their raw data to visually compare the averages to each training session. Again, we'll use the data from 2015 through 2018.

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

The code is as follows:-

```
# 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()
```

the resulting plots are:-



3.7 TRACKING THE PROGRESS

Let's dive a little deeper into the data to answer a tricky question: is their any progress in terms of the running skills?

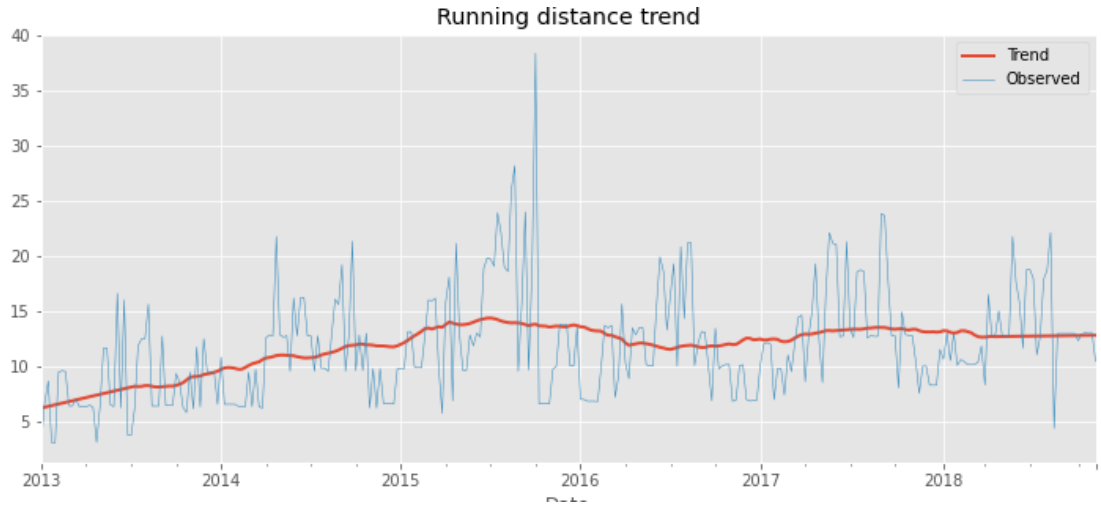
To answer this question, we'll decompose the weekly distance run and visually compare it to the raw data. A red trend line will represent the weekly distance run.

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

The code is:-

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

# Prepare data
df_run_dist_wkly = df_run.loc['20190101':'20130101']['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()
```

the resulting plot is :-



the plot shows a very slight and gradual increase in the average distance covered per day.

3.8 TRAINING INTENSITY ANALYSIS

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.

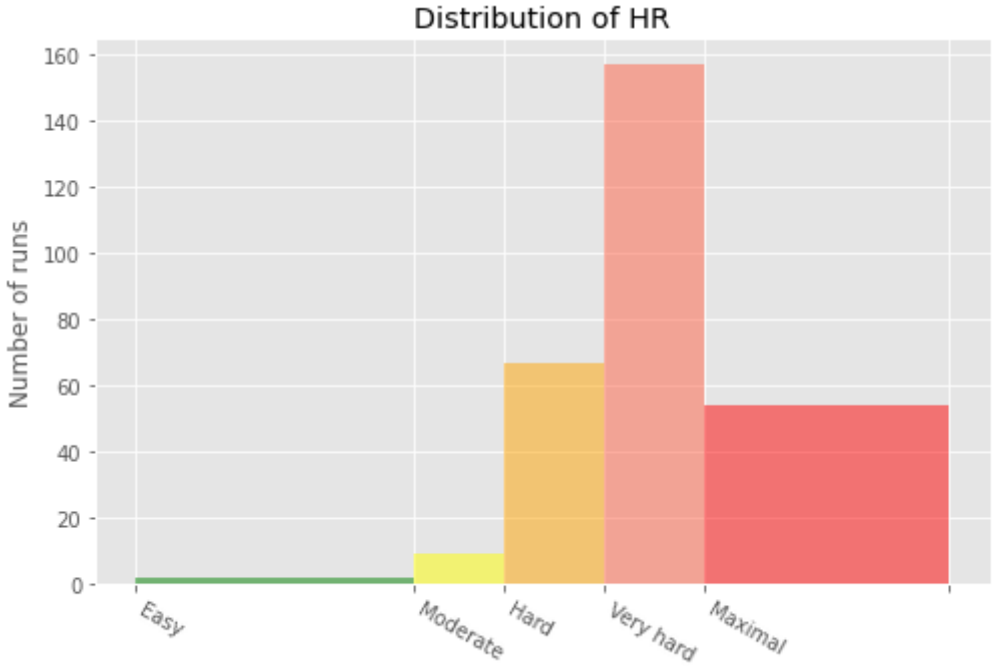
```
# Prepare data
hr_zones = [100,125, 133, 142, 151,173]
zone_names = ['Easy', 'Moderate', 'Hard', 'Very hard', 'Maximal','']
zone_colors = ['green', 'yellow', 'orange', 'tomato', 'red']
df_run_hr_all = df_run['2019':'2015-03']['Average Heart Rate (bpm)']

# 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.set_xticks(hr_zones)
ax.set_xticklabels(zone_names, rotation=-30, ha='left')

# Show plot
plt.show()
```

First we prepared the lists with different attributes to test.

Then we plotted a histogram with custom bins and patches. Then we set the necessary labels and ticks.

Finally we printed the plot.

The resulting histogram is:-



as we can see, on an average, the heart rate was very high according to the dataset.

3.9 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
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:')
df_summary.stack()
```

First we append the 3 datasets we created earlier for running, cycling and walking.

Then we calculate the total distance covered in each type of activity.

Finally we summarize different statistics and print them.

The results are:-

```
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

3.10 FUN FACTS!

To wrap up, let's pick some fun facts out of the summary tables. These data 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.

to calculate the required shoes, we use this code:-

```
# 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_run))
```

and the results are:-

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

hence we will need 33 pairs of shoes!

4. CONCLUSION AND FUTURE SCOPE

This project aimed at analysing and working with datasets and summarizing important results. We can further work on the data to create more visuals and analyse other metrics based on the requirement. Currently this project is on its final stage with the basic metrics analysed and visualised. The project notebook has been submitted for review.

5. REFERENCES

1. https://en.wikipedia.org/wiki/Python_(programming_language)

2. https://pypi.org/project/pandas/

3. https://www.geeksforgeeks.org/python-introduction-matplotlib/

4. https://www.geeksforgeeks.org/warnings-in-python/

5. https://www.statsmodels.org/stable/index.html

6. https://www.tutorialspoint.com/numpy/index.htm

7. https://www.tutorialspoint.com/scikit_learn/index.htm

8. https://jupyter.org/