

Practical Assignment

Jyrki Savolainen

Shortly

- Records from Sports Tracking software Endomondo
 - Terminated in 2020
 - Data of a one person 2017-2020 with 3456 events stored as .json
 - Download .zip from Moodle
- (Pre-)process, analyze and predict
 - Present analysis in a Jupyter-Notebook (.ipynb)

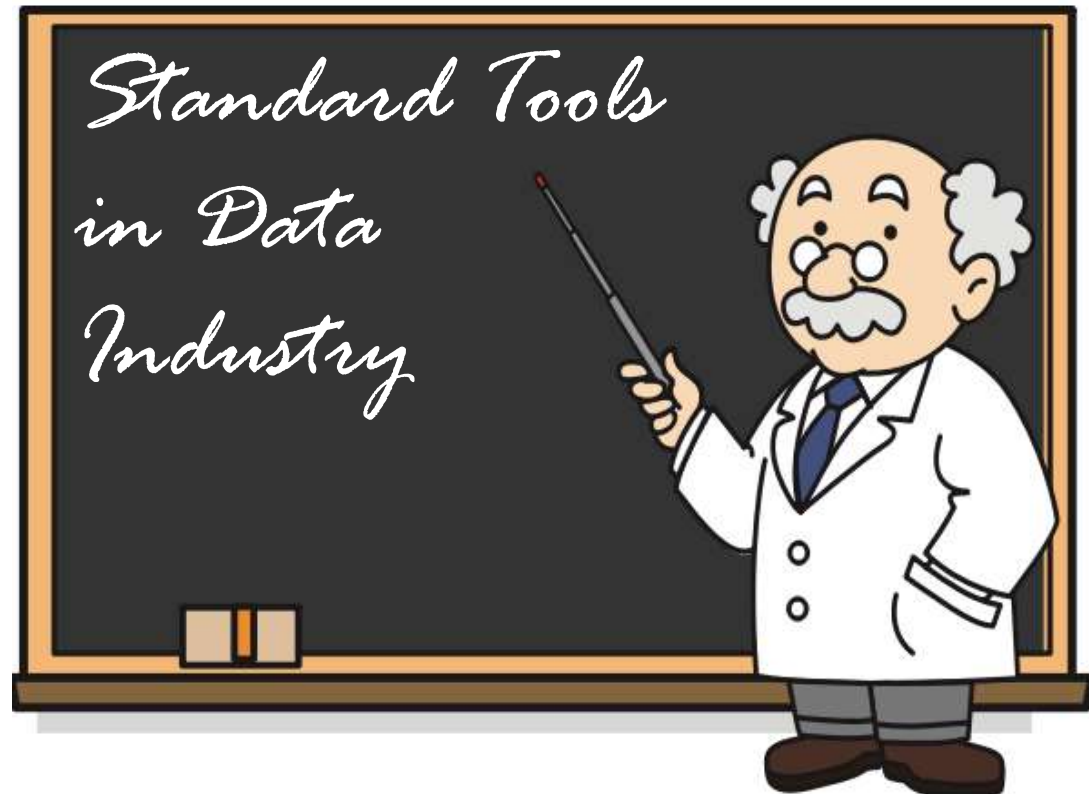


Learning goals

- Building a machine learning (ML) pipeline from proprietary raw data to a predictive model

Not available
in Kaggle, etc.

- Gain experience in:
 - .json files
 - Python with
 - Pandas, numpy, os...
 - Jupyter Notebooks



What is .json?

■ JavaScript Object Notation

- *open standard file format and data interchange format that uses human-readable text to store and transmit data objects consisting of attribute–value pairs and arrays (or other serializable values). It is a common data format with a diverse range of functionality in data interchange including communication of web applications with servers.*
(Wikipedia)

```
1 [{"sport": "WALKING",
2  {"source": "TRACK_MOBILE"},
3  {"created_date": "2017-01-01 08:54:23.0"},
4  {"start_time": "2017-01-01 08:53:04.0"},
5  {"end_time": "2017-01-01 09:27:49.0"},
6  {"duration_s": 2084},
7  {"distance_km": 2.15},
8  {"calories_kcal": 171.651},
9  {"altitude_min_m": 145.5},
10 {"altitude_max_m": 198},
11 {"speed_avg_kmh": 3.714011516314779},
12 {"speed_max_kmh": 6.3},
13 {"ascend_m": 78},
14 {"descend_m": 77},
15 {"points": [
16   [
17    {"location": [[
18     {"latitude": 64.231747},
19     {"longitude": 27.729461}
20    ]}],
21    {"distance_km": 0},
22    {"timestamp": "Sun Jan 01 08:53:04 UTC 2017"}
23   ],
24   [
25    {"location": [[
26     {"latitude": 64.231747},
27     {"longitude": 27.729461}
28    ]}],
29    {"altitude": 158},
30    {"distance_km": 0},
31    {"speed_kmh": 0},
32    {"timestamp": "Sun Jan 01 08:53:20 UTC 2017"}
33   ],
34   [
35    {"location": [[
36     {"latitude": 64.231667},
37     {"longitude": 27.729608}
38    ]}],
39    {"altitude": 147},
40    {"distance_km": 0.01},
41    {"speed_kmh": 3.6},
42    {"timestamp": "Sun Jan 01 08:54:18 UTC 2017"}
43   ],
44   [
45    {"location": [[
46     ...
```

Figure. json-file displayed in Notepad++

Why is .json?

HOW TO GET
OUR HANDS IN
THIS DATA?

Application
("data silo")
coded with C++ /
Java / Fortran /
Matlab / BASIC /
Pascal / ...

SIMPLE CASE

→ .CSV
(Tabular data; "Kaggle"-examples)

	Var1	Var 2	...	Var n
1	12	43		34
...				
n

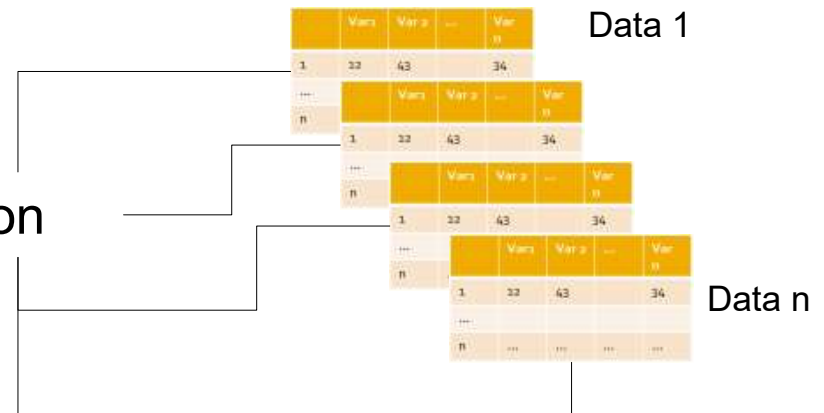
REAL CASE



→ .json
(Multidimensional data object)

```
myStruct =  
  
struct with fields:  
  
data1: [2×20 double]  
data2: {[3] [3] [4]}
```

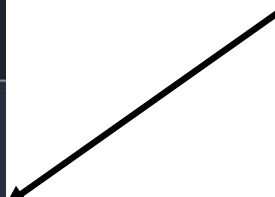
(e.g. Matlab-structure)



Getting started...

```
1  # -*- coding: utf-8 -*-
2  """
3  @author: h17163
4  """
5
6  #%% Import libraries
7  import pandas as pd
8  import os
9
10 #%% Settings
11 # .json-folder
12 folder = 'C:\\Users\\h17163\\Endomondo\\Workouts\\'
13 # List of files
14 fileList = os.listdir(folder)
15
16 #%% Read data
17 # First entry
18 ex0 = pd.read_json(folder+fileList[0], typ='series')
```

Pandas
provides an
easy way to
read json-files



Indexing data...

```
In [54]: ex0
Out[54]:
0          {'sport': 'CYCLING_SPORT'}
1          {'source': 'TRACK_MOBILE'}
2      {'created_date': '2016-04-15 11:46:51.0'}
3          {'start_time': '2016-04-15 11:45:44.0'}
4          {'end_time': '2016-04-15 13:36:44.0'}
5          {'duration_s': 4784}
6          {'distance_km': 19.54}
7          {'calories_kcal': 737.992}
8          {'altitude_min_m': 120.5}
9          {'altitude_max_m': 185}
10         {'speed_avg_kmh': 14.704013377926419}
11         {'speed_max_kmh': 29.7}
12         {'ascend_m': 178}
13         {'descend_m': 161}
14     {'points': [[{'location': [{'latitude': 64.23...
dtype: object


In [55]: type(ex0[0])
Out[55]: dict

In [56]: ex0[0]
Out[56]: {'sport': 'CYCLING_SPORT'}

In [57]: ex0[0]['sport']
Out[57]: 'CYCLING_SPORT'

In [67]: type(ex0)
Out[67]: pandas.core.series.Series
```

- The data is stored as a *pandas Series* containing a 'dictionary' on each row



Data of person's movement with location information as a function of time etc.

Script for reading all jsons (available in Moodle)...

```
def read_file_to_df(filename):
    data = pd.read_json(filename)
    value = []
    key = []
    for j in list(range(0, data)):
        if list(data[j].keys()):
            key.append(list(data[j].keys())[0])
            value.append(list(data[j].values())[0])
            dictionary = dict(zip(key, value))

    if list(data[j].keys())[0]:
        try:
            start = list(list(data[j].values())[0].keys())[0]
            dictionary['start_time'] = list(data[j].values())[0][start]
            dictionary['start_location'] = list(data[j].values())[0][start_location]
            dictionary['end_location'] = list(data[j].values())[0][end_location]
            dictionary['end_time'] = list(data[j].values())[0][end_time]
        except:
            print('No detailed data for this file')

    df = pd.DataFrame(dictionary)

    return df
```

```
### Read all files in a loop
```

```
# Create Empty DataFrame
df_res = pd.DataFrame()
```

```
# Read files to a common dataframe
for filename in file_list:
    print('\n'+filename)
    df_process = read_file_to_df(folder + '/' + filename)
    df_res = pd.concat([df_res, df_process], 0)
```

df_res - DataFrame

Index	sport	source	created_date	start_time	end_time	duration_s	distance_km	calories_kcal	altitude_min	altitude_max
0	WALKING	TRACK_MOBILE	2017-01-01 08:54:23.0	2017-01-01 08:53:04.0	2017-01-01 09:27:49.0	2084	2.15	171.651	145.5	198
0	WEIGHT_TRAINING	INPUT_MANUAL_MOBILE	2017-01-01 15:02:04.0	2017-01-01 14:01:00.0	2017-01-01 14:41:00.0	2400	0	393.333	nan	nan
0	WALKING	TRACK_MOBILE	2017-01-01 17:47:03.0	2017-01-01 17:46:00.0	2017-01-01 18:12:07.0	1566	1.69	132.168	126.5	174
0	WALKING	TRACK_MOBILE	2017-01-02 08:57:23.0	2017-01-02 08:55:52.0	2017-01-02 09:26:06.0	1812	2.07	157.828	81	201
0	RUNNING	TRACK_MOBILE	2017-01-02 16:20:51.0	2017-01-02 16:13:34.0	2017-01-02 16:54:52.0	2444	5.87	591.404	97.5	159
0	WALKING	TRACK_MOBILE	2017-01-03 09:25:22.0	2017-01-03 09:19:16.0	2017-01-03 09:53:05.0	1963	2.15	167.024	138	198
0	SKIING_CROSS_COUNTRY	INPUT_MANUAL_MOBILE	2017-01-03 17:41:24.0	2017-01-03 11:40:00.0	2017-01-03 12:30:00.0	3000	7.4	787	nan	nan
0	WALKING	TRACK_MOBILE	2017-01-03 17:54:10.0	2017-01-03 17:52:29.0	2017-01-03 18:24:36.0	1927	2.46	178.907	127	169
0	WALKING	TRACK_MOBILE	2017-01-04 09:16:33.0	2017-01-04 08:52:12.0	2017-01-04 09:18:36.0	1583	1.75	135.384	143.5	192
0	WALKING	TRACK_MOBILE	2017-01-04 09:52:22.0	2017-01-04 09:50:10.0	2017-01-04 10:18:29.0	1695	1.1	109.328	0	168
0	WALKING	TRACK_MOBILE	2017-01-04 18:04:33.0	2017-01-04 15:47:22.0	2017-01-04 16:01:53.0	869	1.12	81.1347	117.5	170
0	WALKING	TRACK_MOBILE	2017-01-04 18:05:32.0	2017-01-04 18:04:30.0	2017-01-04 18:20:42.0	854	0.99	75.0009	136.5	156
0	WEIGHT_TRAINING	INPUT_MANUAL_MOBILE	2017-01-04 18:05:48.0	2017-01-04 17:05:00.0	2017-01-04 18:00:00.0	3300	0	540.833	nan	nan
0	WALKING	TRACK_MOBILE	2017-01-05 08:42:51.0	2017-01-05 08:41:33.0	2017-01-05 09:08:28.0	1614	1.95	145.124	149	197
0	WALKING	TRACK_MOBILE	2017-01-05 18:06:57.0	2017-01-05 17:46:41.0	2017-01-05 18:09:10.0	1349	1.66	122.587	124.5	163
0	WALKING	TRACK_MOBILE	2017-01-06 09:19:39.0	2017-01-06 08:54:30.0	2017-01-06 09:19:28.0	1496	1.79	133.768	143.5	198
0	WALKING	TRACK_MOBILE	2017-01-06 15:04:55.0	2017-01-06 15:03:19.0	2017-01-06 15:26:39.0	1398	1.83	131.732	126	180
0	WALKING	TRACK_MOBILE	2017-01-07 09:34:13.0	2017-01-07 09:32:44.0	2017-01-07 10:04:02.0	1878	2.15	163.774	150	195
0	SWIMMING	INPUT_MANUAL_MOBILE	2017-01-07 15:23:19.0	2017-01-07 14:22:00.0	2017-01-07 14:52:00.0	1800	1	355.088	nan	nan
0	WALKING	TRACK_MOBILE	2017-01-07 17:43:47.0	2017-01-07 17:41:20.0	2017-01-07 18:14:41.0	2000	2.57	186.402	140.5	196

Format Resize Background color Column min/max Save and Close Close

Data on the movement

```
In [77]: mov_ex0 = ex0[14]
```

```
In [78]: type(mov_ex0)
```

```
Out[78]: dict
```

```
In [79]: mov_ex0 = ex0[14]['points']
```

Dictionary containing only one entry

→ Can be converted to a list

mov_ex0 - Dictionary (1 element)

Key	Type	Size	Value
points	list	1340	[[{'location': [...], 'altitude': 152, 'distance_km': 0, 'speed_kmh': ...}, {'location': [...], 'altitude': 158.5, 'distance_km': 0.01, 'speed_kmh': ...}, {'location': [...], 'altitude': 154, 'distance_km': 0.04, 'speed_kmh': ...}, {'location': [...], 'altitude': 154.5, 'distance_km': 0.05, 'speed_kmh': ...}, {'location': [...], 'altitude': 155, 'distance_km': 0.07, 'speed_kmh': ...}, {'location': [...], 'altitude': 157.5, 'distance_km': 0.08, 'speed_kmh': ...}, {'location': [...], 'altitude': 158, 'distance_km': 0.09, 'speed_kmh': ...}, {'location': [...], 'altitude': 156.5, 'distance_km': 0.1, 'speed_kmh': ...}, {'location': [...], 'altitude': 155, 'distance_km': 0.11, 'speed_kmh': ...}, {'location': [...], 'altitude': 151, 'distance_km': 0.12, 'speed_kmh': ...}]]

1-level dictionary...

mov_ex0 - List (1340 elements)

Ind	Type	Size	Value
0	list	3	[{'location': [...], 'distance_km': 0, 'timestamp': 'Fri Apr 15 11:45 ...}
1	list	5	[{'location': [...], 'altitude': 152, 'distance_km': 0, 'speed_kmh': ...}
2	list	5	[{'location': [...], 'altitude': 158.5, 'distance_km': 0.01, 'speed_kmh': ...}
3	list	5	[{'location': [...], 'altitude': 154, 'distance_km': 0.04, 'speed_kmh': ...}
4	list	5	[{'location': [...], 'altitude': 154.5, 'distance_km': 0.05, 'speed_kmh': ...}
5	list	5	[{'location': [...], 'altitude': 155, 'distance_km': 0.07, 'speed_kmh': ...}
6	list	5	[{'location': [...], 'altitude': 157.5, 'distance_km': 0.08, 'speed_kmh': ...}
7	list	5	[{'location': [...], 'altitude': 158, 'distance_km': 0.09, 'speed_kmh': ...}
8	list	5	[{'location': [...], 'altitude': 156.5, 'distance_km': 0.1, 'speed_kmh': ...}
9	list	5	[{'location': [...], 'altitude': 155, 'distance_km': 0.11, 'speed_kmh': ...}
10	list	5	[{'location': [...], 'altitude': 151, 'distance_km': 0.12, 'speed_kmh': ...}
11	list	5	[{'location': [...], 'altitude': 151, 'distance_km': 0.12, 'speed_kmh': ...}

List...

Dictionary...

```
mov_ex0[14]['points'][0][0]['location']  
[{'latitude': 64.231747}, {'longitude': 27.729461}]
```

Example visualizations of ONE ENTRY

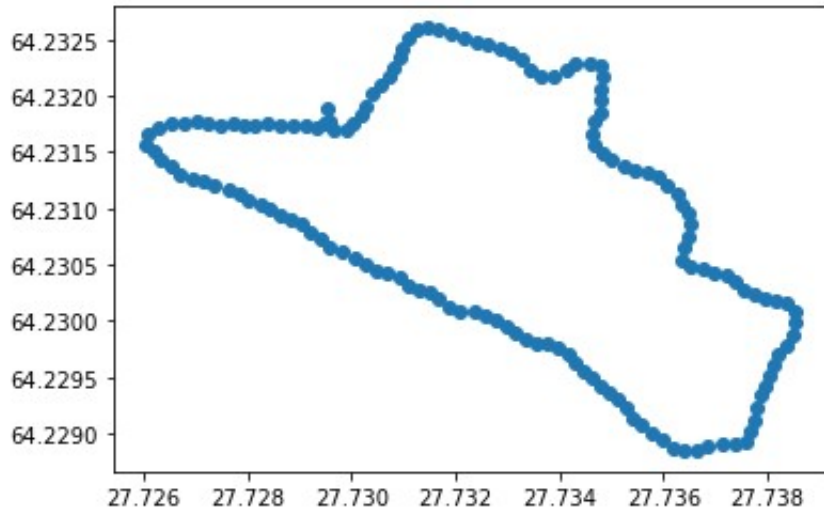


Figure. Longitude and latitude information as a scatter plot

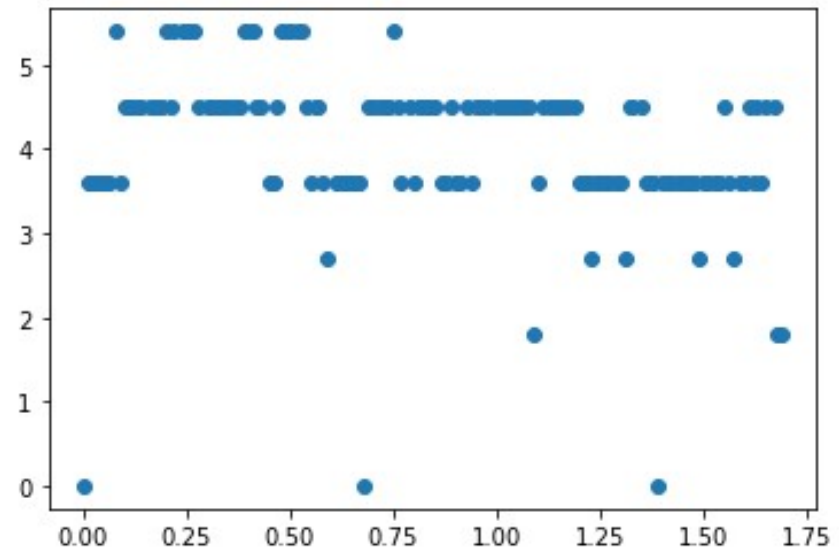


Figure. Distance and speed as a scatter plot

```
len(df_res.loc[df_res['source']=='TRACK_MOBILE'])  
3107  
  
len(df_res.loc[df_res['source']=='INPUT_MANUAL_MOBILE'])  
349
```

~**3107** exercises with location info
(e.g. walking, skiing, cycling,...)

~**349** exercises recorded manually after
the exercise (e.g. gym, swimming)

Index	Type	Size	Value
0	float64	1	0.
1	float64	1	3.
2	float64	1	4.
3	float64	1	4.
4	float64	1	4.
5	float64	1	5.
6	float64	1	5.
7	float64	1	5.
8	float64	1	5.
9	float64	1	4.
10	float64	1	4.
11	float64	1	6.
12	float64	1	4.
13	float64	1	3.
14	float64	1	3.
15	float64	1	3.
16	float64	1	3.
17	float64	1	2.
18	float64	1	3.
19	float64	1	3.
20	float64	1	3.
21	float64	1	2.
22	float64	1	2.
23	float64	1	4.5

```
In [23]: df_res['speed_arr']=np.nan
```

```
In [24]: df_res
```

```
Out[24]:
```

	sport	source	...	hydration_1	speed_arr
0	WALKING	TRACK_MOBILE	...	NaN	NaN
1	WEIGHT_TRAINING	INPUT_MANUAL_MOBILE	...	NaN	NaN
2	WALKING	TRACK_MOBILE	...	NaN	NaN
3	WALKING	TRACK_MOBILE	...	NaN	NaN

```
In [68]: df_res['speed_arr'][0] = speeds_ex0
```

```
In [69]: df_res
```

```
Out[69]:
```

	sport	...	speed_arr
0	WALKING	...	[0.0, 3.6, 4.5, 4.5, 4.5, 5.4, 5.4, 5.4, 5.4, ...]
1	WEIGHT_TRAINING	...	NaN
2	WALKING	...	NaN
3	WALKING	...	NaN
4	RUNNING	...	NaN
...
3451	WALKING	...	NaN
3452	WALKING	...	NaN
3453	WALKING	...	NaN
3454	WALKING	...	NaN
3455	SKIING_CROSS_COUNTRY	...	NaN

```
[3456 rows x 20 columns]
```

What to do

- **MAIN TASK:** Create a ML-model that forecasts the user's next exercise type, time and, possibly, duration as well

- **SUB-TASK:** Before the ML-model fitting, classify / label / cluster / categorize the data!

Start here
=
**FEATURE
ENGINEERING**

Exploratory Analysis 1/3

Geografic Location

The map is based on Longitude and Latitude from that data. Color shows details about Sport. We can observe from the map that most of the users are from Finland and a very few from Italy and UK. Another thing which we can observe is that the sports data from Italy and UK is only for Walking and all other sports are only recorded in Finland.

Geographic Location

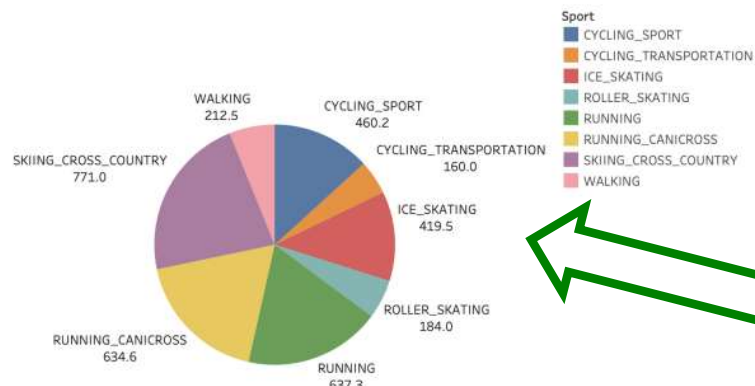


The location for different types of exercises?

Avg. Calories Burned (Kcal) w.r.t Sport

This pie chart shows the how much Calories (Kcal) on average are burned in each sport. Color shows details about Sport. The marks are labeled by Sport and average of Calories Kcal. We can observe from the chart that cross country skiing and both kinds of running are the type of sports that burn the most calories on average and they contribute to more than half of the division on the chart being the most physical sports.

Avg. Calories Burned (Kcal) w.r.t Sport



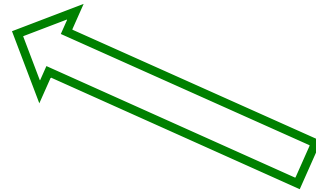
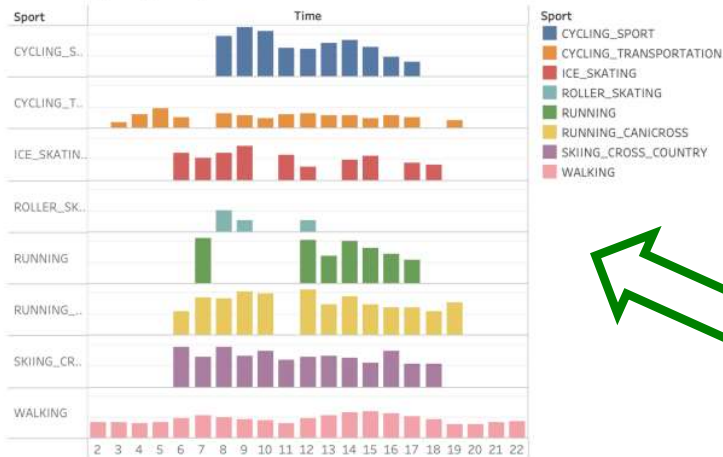
Heaviness of the exercise might be related to the resting period and the next type exercise

Exploratory Analysis 2/3

Time vs Sport (24 hr)

This bar plot shows us Average of duration for each time hour broken down by Sport. We can observe that walking is the only sport that has activity round the clock. Cycling, Ice skating, running and cross country skiing all have similar pattern of activity from 6 AM to 6 PM approximately. And roller skating is the sport that is done for the shortest time and only before noon.

Time vs Sport (24 hr)



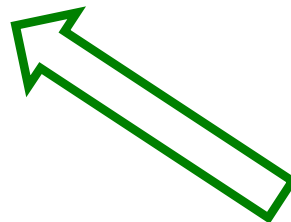
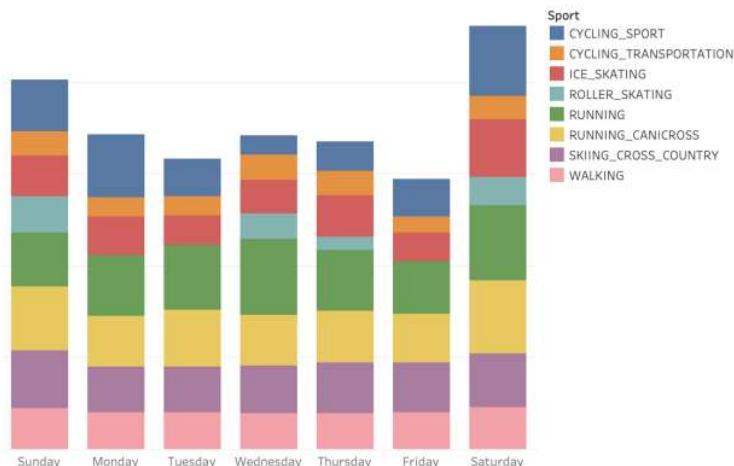
Regular morning / evening walks (with dogs) with similar type of path? Possible to identify these?

‘RUNNING_CANICROSS’ = ‘RUNNING’

WeekDay vs Sport

This bar plot shows the average duration for each weekday. Color shows details about Sport. We can see that the weekend (saturday and sunday) has the most activity with saturday being the most active day of the week and all other week days have on average similar division of sport activities with Friday being the least active among all comparatively.

WeekDay vs Sport



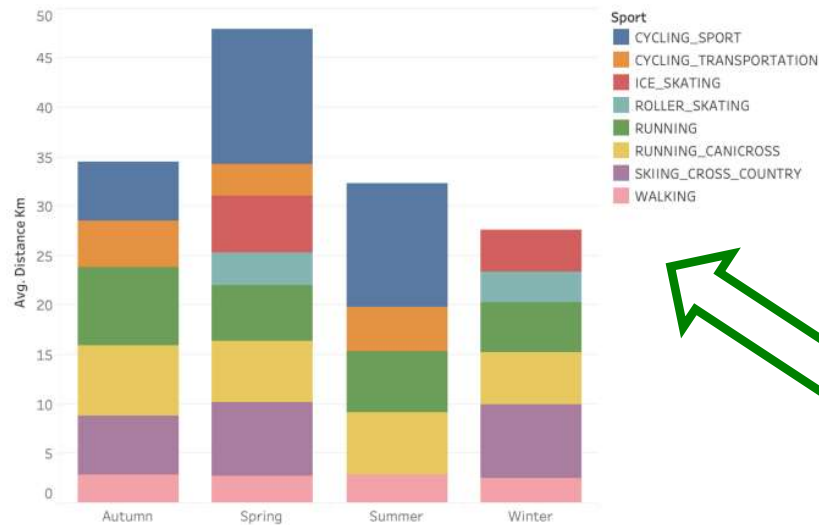
How the exercises are distributed throughout the week? Are some exercises more probable at some days?

Exploratory Analysis 3/3

Seasons w.r.t sports

This plot shows the activity for each season. Color shows details about Sport. We can see from this bar plot that the most active sports season according to this data is Spring and the least active as can be expected is Winter. Autumn and Summer are both similar in activities. And Summer suprisingly is less active than both Autumn and Spring

Seasons w.r.t sports



Seasonality: what options are available?

Example ML-model

Extracted and/or Engineered
features in the data
= x-variables

Predicted activity
based on features
= y-variable



Row	Last activity type	Last activity timing	Month	Location	...	Predicted next activity	(Predicted time)	(Predicted Length)
1	'Dog walk'	'Morning'	1	'Home'	...	'Swimming'	'Evening'	<1h
2	'Cycling'	'Skiing'	1	'
...								

HINT: calculating similarity between arrays

```
In [95]: ex1 = np.array([1, 5, 6, 9, 11, 20])
```

Values of exercise 1
(~row from a dataframe)

```
In [96]: ex2 = np.array([4, 5, 6, 9, 14, 21])
```

Values of exercise 2

```
In [97]: from scipy import spatial
```

```
In [98]: spatial.distance.euclidean(ex1, ex2)
```

```
Out[98]: 4.358898943540674
```

EXAMPLE distances 1 vs 2:
euclidean and cosine (these
are very similar)

```
In [99]: spatial.distance.cosine(ex1, ex2)
```

```
Out[99]: 0.009019993622814249
```

```
In [100]: ex3 = np.array([10, 14, 20, 25, 55, 20])
```

Values of exercise 3

```
In [101]: spatial.distance.euclidean(ex1, ex3)
```

```
Out[101]: 50.49752469181039
```

Exercise 3 differs from ex1
(and, therefore also, ex2)

```
In [102]: spatial.distance.cosine(ex1, ex3)
```

```
Out[102]: 0.1944581077691797
```

Results and reporting

- Provide your analysis and model as a Jupyter Notebook with:
 - code, images, explanation, etc.
 - Analysis on forecastin model's applicability and 'goodness'
- Grading (roughly; adequate documentation assumed):
 - Pre-processing: ~60%
 - Plan/draft of model: ~20%
 - Implementation / insights: ~20%

Using Treasure Data with Python and Pandas

Treasure Data has a [python client](#), which means pandas/python users can connect directly from their iPython Notebooks.

All you need is a Treasure Data account, which you can get from [here](#)

```
In [2]: import tdclient
import pandas as pd
import numpy as np
%matplotlib inline
```

Getting Treasure Data's apikey

You need to get your Treasure Data API key. There are two ways to fetch your API keys after you sign up for Treasure Data.

1. **From web console:** Please access [this URL](#). At the right most column, you can retrieve the API key. You want to use the Normal, not Write-Only API keys to run queries.
2. **From CLI:** If you are the td command user, running the following command exposes your API key.

```
td apikey:show
```

```
In [3]: apikey = 'Your API key here' # Setting your API key
```

```
In [4]: client = tdclient.Client(apikey) # instantiating the client
```

Running a query against the sample dataset

As you can see below, running queries is easy. Just use the query method, which accepts three arguments.

1. The first argument is the name of the database
2. The second argument is the query string (Make sure you use single quotes if you are using the Presto engine!)
3. The optional keyword arguments. I am using type='presto' here to use Presto and not Hive.

Figure. Jupyter Notebook-format