# Content

- Sorting function - `sort()` , `argsort()`
- **Use Case: Fitness Data analysis**
  - Loading data set and EDA using numpy
  - `np.argmax()`

## Use Case: Fitbit

## Imagine you are a Data Scientist at Fitbit

You've been given a user data to analyse and find some insights which can be shown on the smart watch.

#### But why would we want to analyse the user data for desiging the watch?

These insights from the user data can help business make customer oriented decision for the product design.

#### Lets first look at the data we have gathered

Link: [https://drive.google.com/file/d/1Uxwd4H-tfM64giRS1VExMpQXKtBBtuP0/view?usp=sharing (https://drive.google.com/file/d/1Uxwd4H-tfM64giRS1VExMpQXKtBBtuP0/view?usp=sharing)](https://drive.google.com/file/d/1Uxwd4H-tfM64giRS1VExMpQXKtBBtuP0/view?usp=sharing)

fit.txt

| #date | step_count | mood | calories_burned | hours_of_sleep | active |
|---|---|---|---|---|---|
| 06-10-2017 | 5464 | Neutral | 181 | 5 | Inactive |
| 07-10-2017 | 6041 | Sad | 197 | 8 | Inactive |
| 08-10-2017 | 25 | Sad | 0 | 5 | Inactive |
| 09-10-2017 | 5461 | Sad | 174 | 4 | Inactive |
| 10-10-2017 | 6915 | Neutral | 223 | 5 | Active |
| 11-10-2017 | 4545 | Sad | 149 | 6 | Inactive |
| 12-10-2017 | 4340 | Sad | 140 | 6 | Inactive |
| 13-10-2017 | 1230 | Sad | 38 | 7 | Inactive |
| 14-10-2017 | 61 | Sad | 1 | 5 | Inactive |
| 15-10-2017 | 1258 | Sad | 40 | 6 | Inactive |
| 16-10-2017 | 3148 | Sad | 101 | 8 | Inactive |
| 17-10-2017 | 4687 | Sad | 152 | 5 | Inactive |
| 18-10-2017 | 4732 | Happy | 150 | 6 | Active |
| 19-10-2017 | 3519 | Sad | 113 | 7 | Inactive |
| 20-10-2017 | 1580 | Sad | 49 | 5 | Inactive |
| 21-10-2017 | 2822 | Sad | 86 | 6 | Inactive |
| 22-10-2017 | 181 | Sad | 6 | 8 | Inactive |
| 23-10-2017 | 3158 | Neutral | 99 | 5 | Inactive |
| 24-10-2017 | 4383 | Neutral | 143 | 4 | Inactive |
| 25-10-2017 | 3881 | Neutral | 125 | 5 | Inactive |
| 26-10-2017 | 4037 | Neutral | 129 | 6 | Inactive |

**Notice that there are some user features in the data**

There are provided as various columns in the data.

**Every row is called a record or data point**

**What are all the features provided to us?**

- Date
- Step Count
- Mood (Categorical)
- Calories Burned
- Hours of sleep
- Feeling Active (Categorical)

**Using NumPy, we will explore this data to look for some interesting insights - Exploratory Data Analysis.**

**EDA is all about asking the right questions**

**What kind of questions can we answer using this data?**

- How many records and features are there in the dataset?
- What is the **average step count**?
- On which day the **step count was highest/lowest?**

**Can we find some deeper insights?**

We can probably see how daily activity affects sleep and moood.

We will try finding

- How daily activity affects mood?

In [ ]:
```
import numpy as np
```

# Universal Functions (`ufunc`) on 2D & Axis

## Sorting Arrays

- We can also sort the elements of an array along a given specified axis
- Default axis is the last axis of the array.

**`np.sort()`**

In [ ]:
```
a = np.array([2,30,41,7,17,52])
a
```

Out[81]:
```
array([ 2, 30, 41,  7, 17, 52])
```

In [ ]:
```
np.sort(a)
```

Out[82]:
```
array([ 2,  7, 17, 30, 41, 52])
```

In [ ]:
```
a
```

Out[83]:
```
array([ 2, 30, 41,  7, 17, 52])
```

Let's work with 2D array

In [ ]:
```
a = np.arange(9,0,-1).reshape(3,3)
a
```

Out[84]:
```
array([[9, 8, 7],
       [6, 5, 4],
       [3, 2, 1]])
```

**Question: What will be the result when we sort using axis = 0 ?**

In [ ]:
```
np.sort(a, axis = 0)
```

Out[85]:
```
array([[3, 2, 1],
       [6, 5, 4],
       [9, 8, 7]])
```

Recall that when axis =0

- change will happen along vertical axis.

Hence, it will sort out row wise.

In [ ]:
```
a
```

Out[86]:
```
array([[9, 8, 7],
       [6, 5, 4],
       [3, 2, 1]])
```

- Original array is still the same. It hasn't changed

**`np.argsort()`**

- Returns the **indices** that would sort an array.

- Performs an indirect sort along the given axis.
- It returns **an array of indices of the same shape as a that index data along the given axis in sorted order**.

In [ ]:
```
a = np.array([2,30,41,7,17,52])
a
```

Out[89]:
```
array([ 2, 30, 41,  7, 17, 52])
```

In [ ]:
```
np.argsort(a)
```

Out[90]:
```
array([0, 3, 4, 1, 2, 5])
```

**As you can see:**

- The orginal indices of elements are in same order as the orginal elements would be in sorted order

# Use Case: Fitness data analysis

## Let's first download the dataset

In [ ]:
```
!gdown 1vk1Pu0djiYcrdc85yUXZ_Rqq2oZNcohd

```

```
Downloading...
From: https://drive.google.com/uc?id=1vk1Pu0djiYcrdc85yUXZ_Rqq2oZNcohd (https://drive.google.com/uc?id=1vk1Pu0djiYcrdc85yUXZ_Rqq2oZNcohd)
To: /content/fit.txt
100% 3.43k/3.43k [00:00<00:00, 4.10MB/s]
```

## Let's load the data we saw earlier. For this we will use `.loadtxt()` function

In [ ]:
```
data = np.loadtxt('fit.txt', dtype='str')
```

We provide file name along with the dtype of data we want to load in

In [ ]:
```
1 data[:5]
```

Out[93]:
```
array([['06-10-2017', '5464', 'Neutral', '181', '5', 'Inactive'],
       ['07-10-2017', '6041', 'Sad', '197', '8', 'Inactive'],
       ['08-10-2017', '25', 'Sad', '0', '5', 'Inactive'],
       ['09-10-2017', '5461', 'Sad', '174', '4', 'Inactive'],
       ['10-10-2017', '6915', 'Neutral', '223', '5', 'Active']],
      dtype='<U10')
```

What's the shape of the data?

In [ ]:
```
1 data.shape
```

Out[94]:
```
(96, 6)
```

There are 96 records and each record has 6 features. These features are:

- Date
- Step count
- Mood
- Calories Burned
- Hours of sleep
- activity status

**Notice that above array is a homogenous containing all the data as strings**

In order to work with strings, categorical data and numerical data, we will have save every feature seperately

**How will we extract features in seperate variables?**

We can get some idea on how data is saved.

Lets see whats the first element of `data`

In [ ]:
```
1 data[0]
```

Out[95]:
```
array(['06-10-2017', '5464', 'Neutral', '181', '5', 'Inactive'],
      dtype='<U10')
```

Hm, this extracts a row not a column

Think about it.

**Whats the way to change columns to rows and rows to columns?**

Transpose

In [ ]:
```
1 data.T[0]
```

Out[96]:
```
array(['06-10-2017', '07-10-2017', '08-10-2017', '09-10-2017',
       '10-10-2017', '11-10-2017', '12-10-2017', '13-10-2017',
       '14-10-2017', '15-10-2017', '16-10-2017', '17-10-2017',
       '18-10-2017', '19-10-2017', '20-10-2017', '21-10-2017',
       '22-10-2017', '23-10-2017', '24-10-2017', '25-10-2017',
       '26-10-2017', '27-10-2017', '28-10-2017', '29-10-2017',
       '30-10-2017', '31-10-2017', '01-11-2017', '02-11-2017',
       '03-11-2017', '04-11-2017', '05-11-2017', '06-11-2017',
       '07-11-2017', '08-11-2017', '09-11-2017', '10-11-2017',
       '11-11-2017', '12-11-2017', '13-11-2017', '14-11-2017',
       '15-11-2017', '16-11-2017', '17-11-2017', '18-11-2017',
       '19-11-2017', '20-11-2017', '21-11-2017', '22-11-2017',
       '23-11-2017', '24-11-2017', '25-11-2017', '26-11-2017',
       '27-11-2017', '28-11-2017', '29-11-2017', '30-11-2017',
       '01-12-2017', '02-12-2017', '03-12-2017', '04-12-2017',
       '05-12-2017', '06-12-2017', '07-12-2017', '08-12-2017',
       '09-12-2017', '10-12-2017', '11-12-2017', '12-12-2017',
       '13-12-2017', '14-12-2017', '15-12-2017', '16-12-2017',
       '17-12-2017', '18-12-2017', '19-12-2017', '20-12-2017',
       '21-12-2017', '22-12-2017', '23-12-2017', '24-12-2017',
       '25-12-2017', '26-12-2017', '27-12-2017', '28-12-2017',
       '29-12-2017', '30-12-2017', '31-12-2017', '01-01-2018',
       '02-01-2018', '03-01-2018', '04-01-2018', '05-01-2018',
       '06-01-2018', '07-01-2018', '08-01-2018', '09-01-2018'],
      dtype='<U10')
```

Great, we could extract first column

**Lets extract all the columns and save them in seperate variables**

In [ ]:
```
1 date, step_count, mood, calories_burned, hours_of_sleep, activity_status
```

In [ ]:
```
1 step_count
```

Out[98]:
```
array(['5464', '6041', '25', '5461', '6915', '4545', '4340', '1230', '61',
       '1258', '3148', '4687', '4732', '3519', '1580', '2822', '181',
       '3158', '4383', '3881', '4037', '202', '292', '330', '2209',
       '4550', '4435', '4779', '1831', '2255', '539', '5464', '6041',
       '4068', '4683', '4033', '6314', '614', '3149', '4005', '4880',
       '4136', '705', '570', '269', '4275', '5999', '4421', '6930',
       '5195', '546', '493', '995', '1163', '6676', '3608', '774', '1421',
       '4064', '2725', '5934', '1867', '3721', '2374', '2909', '1648',
       '799', '7102', '3941', '7422', '437', '1231', '1696', '4921',
       '221', '6500', '3575', '4061', '651', '753', '518', '5537', '4108',
       '5376', '3066', '177', '36', '299', '1447', '2599', '702', '133',
       '153', '500', '2127', '2203'], dtype='<U10')
```

In [ ]:
```
1 step_count.dtype
```

Out[99]:
```
dtype('<U10')
```

Notice the data type of step_count and other variables. It's a string type where **U** means Unicode String. and 10 means 10 bytes.

**Why? Because Numpy type-casted all the data to strings.**

**Let's convert the data types of these variables**

**Step Count**

In [ ]:
```
1 step_count = np.array(step_count, dtype = 'int')
2 step_count.dtype
```

Out[100]:
```
dtype('int64')
```

In [ ]:
```
1 step_count
```

Out[101]:
```
array([5464, 6041,   25, 5461, 6915, 4545, 4340, 1230,   61, 1258, 3148,
       4687, 4732, 3519, 1580, 2822,  181, 3158, 4383, 3881, 4037,  202,
        292,  330, 2209, 4550, 4435, 4779, 1831, 2255,  539, 5464, 6041,
       4068, 4683, 4033, 6314,  614, 3149, 4005, 4880, 4136,  705,  570,
        269, 4275, 5999, 4421, 6930, 5195,  546,  493,  995, 1163, 6676,
       3608,  774, 1421, 4064, 2725, 5934, 1867, 3721, 2374, 2909, 1648,
        799, 7102, 3941, 7422,  437, 1231, 1696, 4921,  221, 6500, 3575,
       4061,  651,  753,  518, 5537, 4108, 5376, 3066,  177,   36,  299,
       1447, 2599,  702,  133,  153,  500, 2127, 2203])
```

**Calories Burned**

In [ ]:
```
1 calories_burned = np.array(calories_burned, dtype = 'int')
2 calories_burned.dtype
```

Out[102]:
```
dtype('int64')
```

**Hours of Sleep**

In [ ]:
```
1 hours_of_sleep = np.array(hours_of_sleep, dtype = 'int')
2 hours_of_sleep.dtype
```

Out[103]:
```
dtype('int64')
```

**Mood**

`Mood` is a categorical data type. As a name says, categorical data type has two or more categories in it.

In [ ]:
```
1 mood
```

Out[104]:
```
array(['Neutral', 'Sad', 'Sad', 'Sad', 'Neutral', 'Sad', 'Sad', 'Sad',
       'Sad', 'Sad', 'Sad', 'Sad', 'Happy', 'Sad', 'Sad', 'Sad', 'Sad',
       'Neutral', 'Neutral', 'Neutral', 'Neutral', 'Neutral', 'Neutral',
       'Happy', 'Neutral', 'Happy', 'Happy', 'Happy', 'Happy', 'Happy',
       'Happy', 'Happy', 'Neutral', 'Happy', 'Happy', 'Happy', 'Happy',
       'Happy', 'Happy', 'Happy', 'Happy', 'Happy', 'Happy', 'Neutral',
       'Happy', 'Happy', 'Happy', 'Happy', 'Happy', 'Happy', 'Happy',
       'Happy', 'Happy', 'Neutral', 'Sad', 'Happy', 'Happy', 'Happy',
       'Happy', 'Happy', 'Happy', 'Happy', 'Sad', 'Neutral', 'Neutral',
       'Sad', 'Sad', 'Neutral', 'Neutral', 'Happy', 'Neutral', 'Neutral',
       'Sad', 'Neutral', 'Sad', 'Neutral', 'Neutral', 'Sad', 'Sad', 'Sad',
       'Sad', 'Happy', 'Neutral', 'Happy', 'Neutral', 'Sad', 'Sad', 'Sad',
       'Neutral', 'Neutral', 'Sad', 'Sad', 'Happy', 'Neutral', 'Neutral',
       'Happy'], dtype='<U10')
```

In [ ]:
```
1 np.unique(mood)
```

Out[105]:
```
array(['Happy', 'Neutral', 'Sad'], dtype='<U10')
```

**Activity Status**

In [ ]:
```
1 activity_status
```

Out[106]:
```
array(['Inactive', 'Inactive', 'Inactive', 'Inactive', 'Active',
       'Inactive', 'Inactive', 'Inactive', 'Inactive', 'Inactive',
       'Inactive', 'Inactive', 'Active', 'Inactive', 'Inactive',
       'Inactive', 'Inactive', 'Inactive', 'Inactive', 'Inactive',
       'Inactive', 'Inactive', 'Inactive', 'Inactive', 'Inactive',
       'Active', 'Inactive', 'Inactive', 'Inactive', 'Inactive', 'Active',
       'Inactive', 'Inactive', 'Inactive', 'Inactive', 'Inactive',
       'Active', 'Active', 'Active', 'Active', 'Active', 'Active',
       'Active', 'Active', 'Active', 'Inactive', 'Inactive', 'Inactive',
       'Inactive', 'Inactive', 'Inactive', 'Active', 'Active', 'Active',
       'Active', 'Active', 'Active', 'Active', 'Active', 'Active',
       'Active', 'Active', 'Active', 'Inactive', 'Active', 'Active',
       'Inactive', 'Active', 'Active', 'Active', 'Active', 'Active',
       'Inactive', 'Active', 'Active', 'Active', 'Active', 'Inactive',
       'Inactive', 'Inactive', 'Inactive', 'Active', 'Active', 'Active',
       'Active', 'Inactive', 'Inactive', 'Inactive', 'Inactive',
       'Inactive', 'Inactive', 'Inactive', 'Inactive', 'Active',
       'Inactive', 'Active'], dtype='<U10')
```

## Let's try to get some insights from the data.

**What's the average step count?**

How can we calculate average? => `.mean()`

In [ ]:
```
1 step_count.mean()
```

Out[107]: 2935.9375

User moves an average of 2900 steps a day.

**On which day the step count was highest?**

How will be find it?

First we find the index of maximum step count and use that index to get the date.

How'll we find the index? =>

Numpy provides a function `np.argmax()` which returns the index of maximum value element.

Similarly, we have a function `np.argmin()` which returns the index of minimum element.

In [ ]:
```
1 step_count.argmax()
2
```

Out[108]: 69

Here 69 is the index of maximum step count element.

In [ ]:
```
1 date[step_count.argmax()]
```

Out[109]: '14-12-2017'

Let's check the calorie burnt on the day

In [ ]:
```
1 calories_burned[step_count.argmax()]
```

Out[110]: 243

Not bad! 243 calories. Let's try to get the number of steps on that day as well

In [ ]:
```
1 step_count.max()
```

Out[111]: 7422

7k steps!! Sports mode on!

**Let's try to compare step counts on bad mood days and good mood days**

Average step count on Sad mood days

In [ ]:
```
1 np.mean(step_count[mood == 'Sad'])
```

Out[112]:
```
2103.0689655172414
```

In [ ]:
```
1 np.sort(step_count[mood == 'Sad'])
```

Out[113]:
```
array([  25,   36,   61,  133,  177,  181,  221,  299,  518,  651,  702,
        753,  799, 1230, 1258, 1580, 1648, 1696, 2822, 3148, 3519, 3721,
       4061, 4340, 4545, 4687, 5461, 6041, 6676])
```

In [ ]:
```
1 np.std(step_count[mood == 'Sad'])
```

Out[114]:
```
2021.2355035376254
```

Average step count on happy days

In [ ]:
```
1 np.mean(step_count[mood == 'Happy'])
```

Out[115]:
```
3392.725
```

In [ ]:
```
1 np.sort(step_count[mood == 'Happy'])
```

Out[116]:
```
array([ 153,  269,  330,  493,  539,  546,  614,  705,  774,  995, 1421,
       1831, 1867, 2203, 2255, 2725, 3149, 3608, 4005, 4033, 4064, 4068,
       4136, 4275, 4421, 4435, 4550, 4683, 4732, 4779, 4880, 5195, 5376,
       5464, 5537, 5934, 5999, 6314, 6930, 7422])
```

Average step count on sad days - 2103.

Average step count on happy days - 3392

There may be relation between mood and step count

**Let's try to check inverse. Mood when step count was greater/lesser**

Mood when step count > 4000

In [ ]:
```
1 np.unique(mood[step_count > 4000], return_counts = True)
```

Out[117]:
```
(array(['Happy', 'Neutral', 'Sad'], dtype='<U10'), array([22,  9,  7]))
```

Out of 38 days when step count was more than 4000, user was feeling happy on 22 days.

Mood when step count <= 2000

In [ ]:
```
1 np.unique(mood[step_count < 2000], return_counts = True)
```

Out[118]:
```
(array(['Happy', 'Neutral', 'Sad'], dtype='<U10'), array([13,  8, 18]))
```

Out of 39 days, when step count was less than 2000, user was feeling sad on 18 days.

**There may be a correlation between Mood and step count**