pandas

# Pandas

Week 7

# What is Pandas?

- Pandas is a Python library used for working with data sets.
- It has functions for analyzing, cleaning, exploring, and manipulating data.
- The name "Pandas" has a reference to both "Panel Data", and "Python Data Analysis" and was created by Wes McKinney in 2008.
- Pandas is built on top of NumPy

# By now, you should already know

- Python
- NumPy
- OOP

# Why Use Pandas?

- Pandas allows us to analyze big data and make conclusions based on statistical theories.
- Pandas can clean messy data sets, and make them readable and relevant.
- Relevant data is very important in data science.

# What Can Pandas Do?

- Pandas gives you answers about the data. Like:
  - Is there a correlation between two or more columns?
  - What is average value?
  - Max value?
  - Min value?
- Pandas are also able to delete rows that are not relevant, or contains wrong values, like empty or NULL values. This is called cleaning the data.

# Installation of Pandas

- If you have Python and PIP already installed on a system, then installation of Pandas is very easy.

- Install it using this command:

- C:\Users\Your Name>pip install pandas

- If this command fails, then use a python distribution that already has Pandas installed like, Anaconda, Spyder etc.

# Import Pandas

- Once Pandas is installed, import it in your applications by adding the import keyword:

```
import pandas
```

# Sample Pandas code

```
import pandas

mydataset = {
  'cars': ["BMW", "Volvo", "Ford"],
  'passings': [3, 7, 2]
}

myvar = pandas.DataFrame(mydataset)

print(myvar)
```

# Pandas as pd

- Pandas is usually imported under the pd alias.

alias: In Python alias are an alternate name for referring to the same thing.

import pandas as pd

Now the Pandas package can be referred to as pd instead of pandas.

# Pandas code with alias

```
import pandas as pd

mydataset = {
  'cars': ["BMW", "Volvo", "Ford"],
  'passings': [3, 7, 2]
}

myvar = pd.DataFrame(mydataset)

print(myvar)
```

# Checking Pandas Version

- The version string is stored under __version__ attribute.

**REMEMBER: Checking the version is important to prevent deprecated code or inconsistencies.**

```
import pandas as pd

print(pd.__version__)
```

# Pandas Series

# Pandas Series

- A one-dimensional (vector) labeled array capable of holding data of any type (integer, string, float, python objects, etc.).
- The axis labels are collectively called index.
- Nothing but a column in an excel sheet.
- Labels need not be unique but must be a hashable type (int, float, str, tuple, and NoneType).
- Unhashable types are dict, list, and set.
- Hashable means changeable or mutable. Unhashable means unchangeable or immutable.

# What is a Series?

- A Pandas Series is like a column in a table.

- It is a one-dimensional array holding data of any type.

Example:

```
import pandas as pd
a = [1, 7, 2]
myvar = pd.Series(a)
print(myvar)
```

Output:

```
0    1
1    7
2    2
dtype: int64
```

# Labels

- If nothing else is specified, the values are labeled with their index number. First value has index 0, second value has index 1 etc.

- This label can be used to access a specified value.

- Example:

Return the first value of the Series:

```
print(myvar[0]) #myvar is the same as the one in the previous slide.
```

Output: 1

# Create Labels

- With the index argument, you can name your own labels.

Example:

```
import pandas as pd
a = [1, 7, 2]
myvar = pd.Series(a, index = ["x", "y", "z"])
print(myvar)
```

Output:

```
x    1
y    7
z    2
dtype: int64
```

# Label Reference

- When you have created labels, you can access an item by referring to the label.

Example:

```
print(myvar["y"]) #Returns the value of "y“
```

Output: 7

# Key/Value Objects as Series

- You can also use a key/value object, like a dictionary, when creating a Series.

- Example: Create a simple Pandas Series from a dictionary:

```
import pandas as pd
calories = {"day1": 420, "day2": 380, "day3": 390}
myvar = pd.Series(calories)
print(myvar)
```

Output:

```
day1    420
day2    380
day3    390
dtype: int64
```

**Note: The keys of the dictionary become the labels.**

# Item Selection in a Series

- To select only some of the items in the dictionary, use the index argument and specify only the items you want to include in the Series.

Example:

Create a Series using only data from "day1" and "day2":

```
import pandas as pd
calories = {"day1": 420, "day2": 380, "day3": 390}
myvar = pd.Series(calories, index = ["day1", "day2"])
print(myvar)
```

Output:

```
day1    420
day2    380
dtype: int64
```

# Exercise 1

# E1-Q1

- Insert the correct Pandas method to create a Series.

pd.______(mylist)

# E1-Q2

- Insert the correct syntax to return the first value of a Pandas Series called "myseries"?

myseries___

# E1-Q3

- Insert the correct syntax to add the labels "x", "y", and "z" to a Pandas Series.

pd.Series(mylist, ____________________)

# Exercise 1 Accomplished!

# Data Frames

# What is a DataFrame?

- A Pandas DataFrame is a 2-dimensional data structure, like a 2-dimensional array, or a table with rows and columns.

Example:

```
import pandas as pd
data = {
  "calories": [420, 380, 390],
  "duration": [50, 40, 45]
}
#load data into a DataFrame object:
df = pd.DataFrame(data)
print(df)
```

Output:

```
    calories  duration
 0     420       50
 1     380       40
 2     390       45
```

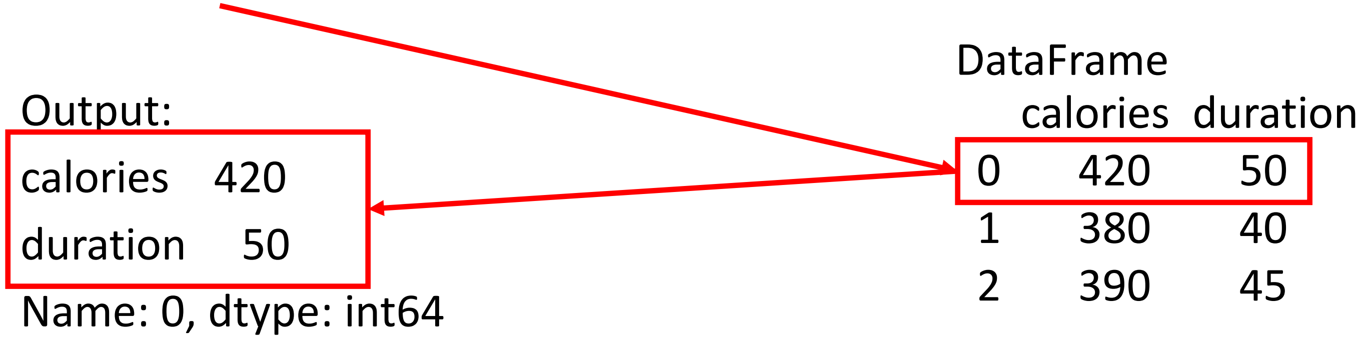
# Locate Row

- As you can see from the result above, the DataFrame is like a table with rows and columns.
- Pandas use the **loc** attribute to return one or more specified row(s)

Example:

*#refer to the row index*:

```
print(df.loc[0])
```



Output:

```
calories    420
duration     50
Name: 0, dtype: int64
```

DataFrame

```
     calories  duration
0       420        50
1       380        40
2       390        45
```

**Note: This example returns a Pandas Series.**

# Locate Row

Example:

```
#Return row 0 and 1:
#use a list of indexes:
print(df.loc[[0, 1]])
```

Output:

```
   calories  duration
0       420        50
1       380        40
```

**Note: When using [], the result is a Pandas DataFrame.**

# Named Indexes

- With the **index** argument, you can name your own indexes.

Example:

```
#Add a list of names to give each row a name:
import pandas as pd
data = {
  "calories": [420, 380, 390],
  "duration": [50, 40, 45]
}
df = pd.DataFrame(data, index = ["day1", "day2", "day3"])
print(df)
```

Output:

```
       calories  duration
 day1       420        50
 day2       380        40
 day3       390        45
```

# Locate Named Indexes

- Use the "**named index** in the loc attribute to return the specified row(s).

Example:

```
#Return "day2":
#refer to the named index:
print(df.loc["day2"])
```

Output:

```
calories    380
duration     40
Name: 0, dtype: int64
```

DataFrame

| | calories | duration |
|---|---|---|
| **day1** | 420 | 50 |
| **day2** | **380** | **40** |
| **day3** | 390 | 45 |

# Load Files Into a DataFrame

- If your data sets are stored in a file, Pandas can load them into a DataFrame using the **read_csv()** function within the Pandas class.

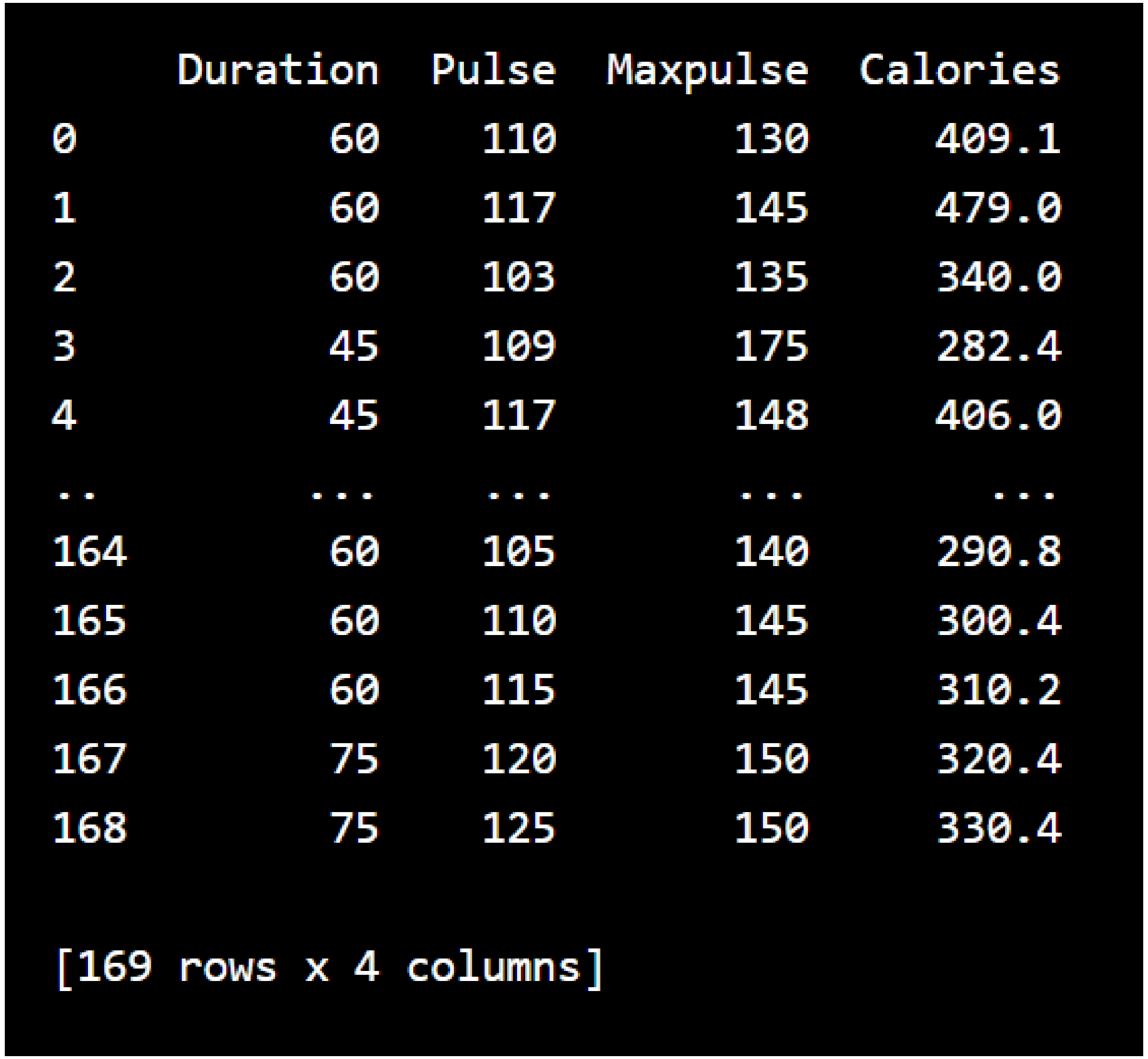
Example:

#Load a comma separated file (CSV file) into a DataFrame:

import pandas as pd

df = pd.**read_csv(**'data.csv’**)**

print(df)



```
     Duration  Pulse  Maxpulse  Calories
0          60    110       130     409.1
1          60    117       145     479.0
2          60    103       135     340.0
3          45    109       175     282.4
4          45    117       148     406.0
..        ...    ...       ...       ...
164        60    105       140     290.8
165        60    110       145     300.4
166        60    115       145     310.2
167        75    120       150     320.4
168        75    125       150     330.4

[169 rows x 4 columns]
```

# Excersie 2

# E2-Q1

- Insert the correct Pandas method to create a DataFrame.

pd.__________(data)

# E2-Q2

- Insert the correct syntax to return the first row of a DataFrame.

df.________

# E2-Q3

- Insert the correct syntax for loading CSV files into a DataFrame.

df.________(data)

# Exercise 2 Accomplished!

# Pandas Read CSV

# Read CSV Files

- A simple way to store big data sets is to use CSV files (comma separated files).

- CSV files contains plain text and is a well know format that can be read by everyone including Pandas.

- In our examples we will be using a CSV file called 'data.csv’.

- To try this out, download the data.csv → [https://www.w3schools.com/python/pandas/data.csv](https://www.w3schools.com/python/pandas/data.csv)

# Reading the Downloaded CSV File

Example:

#Load the CSV into a DataFrame:

import pandas as pd

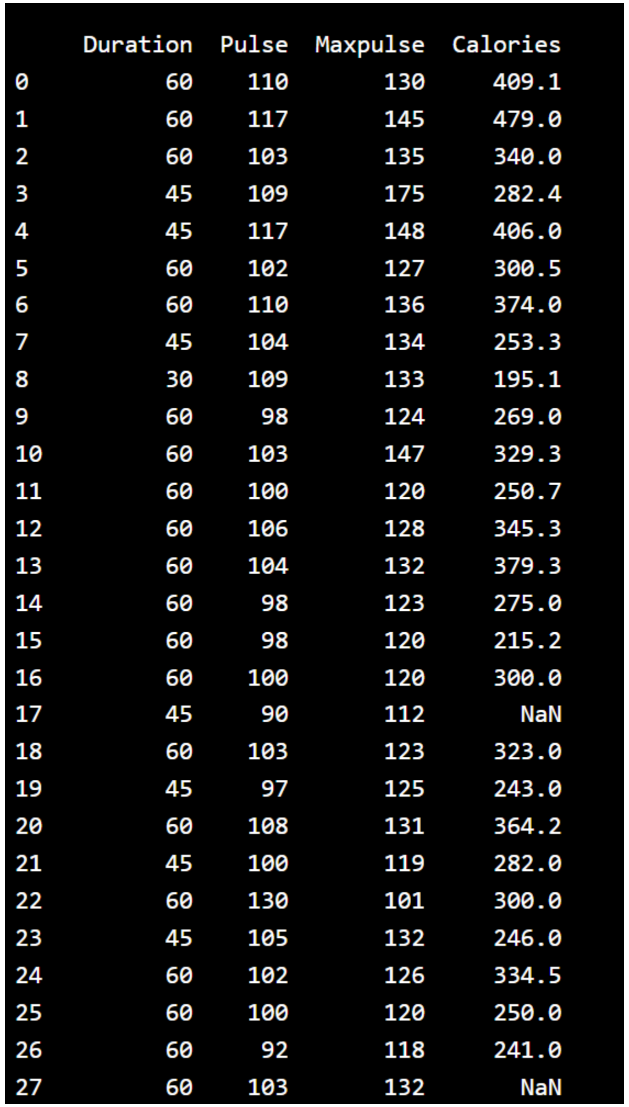
df = pd.read_csv('data.csv')

print(df.**to_string()**)

NOTE: The **to_string()** canprint the **ENTIRE** DataFrame. It’s a function available for the Pandas data frame.

**The succeeding parts in the image are not shown in this slide but they are all printed out.**

**In some cases, If you have a large DataFrame with many rows, Pandas will only return the first 5 rows, and the last 5 rows:**



```
    Duration  Pulse  Maxpulse  Calories
0         60    110       130     409.1
1         60    117       145     479.0
2         60    103       135     340.0
3         45    109       175     282.4
4         45    117       148     406.0
5         60    102       127     300.5
6         60    110       136     374.0
7         45    104       134     253.3
8         30    109       133     195.1
9         60     98       124     269.0
10        60    103       147     329.3
11        60    100       120     250.7
12        60    106       128     345.3
13        60    104       132     379.3
14        60     98       123     275.0
15        60     98       120     215.2
16        60    100       120     300.0
17        45     90       112       NaN
18        60    103       123     323.0
19        45     97       125     243.0
20        60    108       131     364.2
21        45    100       119     282.0
22        60    130       101     300.0
23        45    105       132     246.0
24        60    102       126     334.5
25        60    100       120     250.0
26        60     92       118     241.0
27        60    103       132       NaN
```

# max_rows

- The number of rows returned is defined in Pandas option settings.
- You can check your system's maximum rows with the pd.options.display.max_rows statement.

Example:

*#Check the number of maximum returned rows:*

import pandas as pd

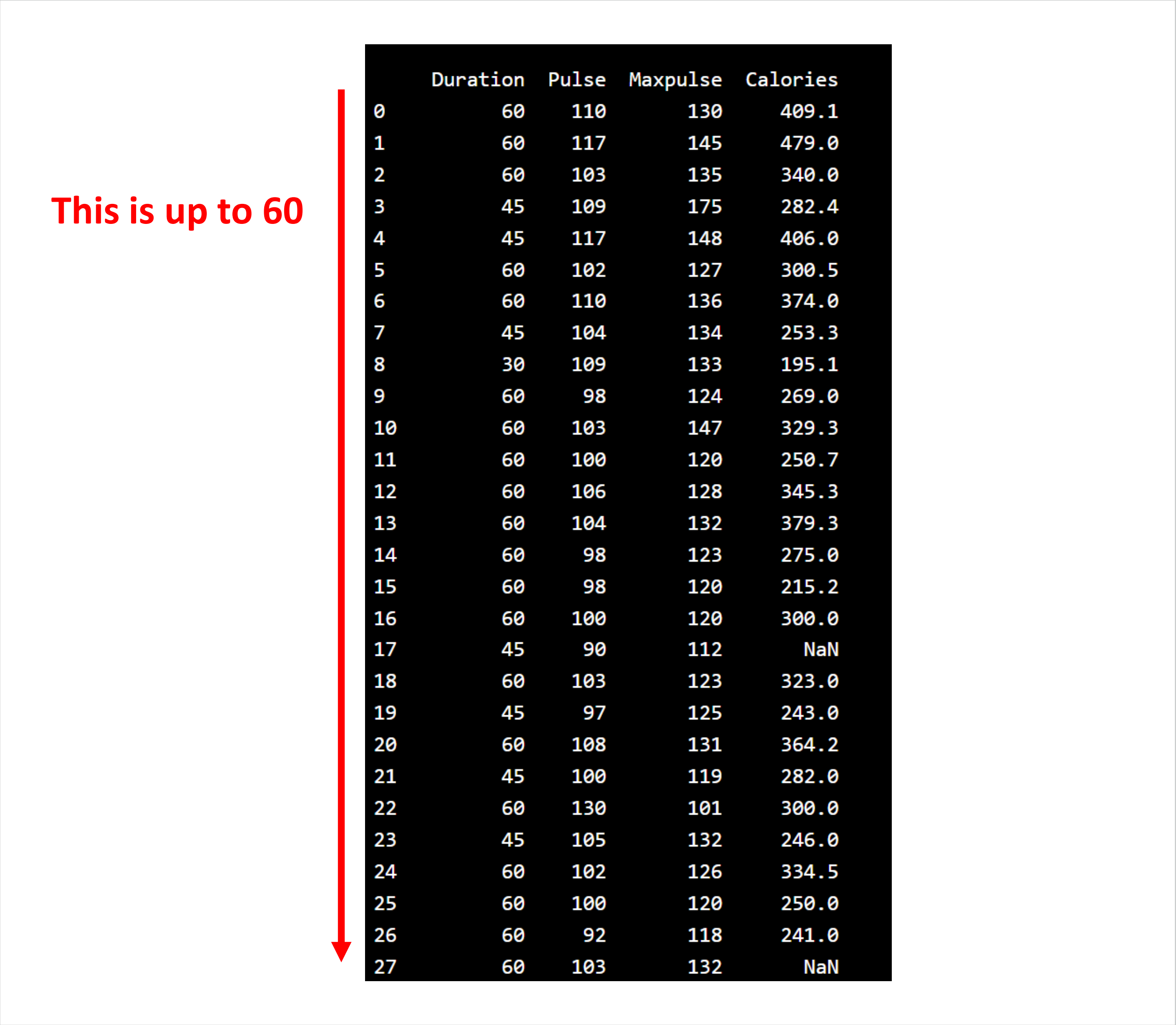
print(pd.options.display.max_rows)

***#The output is 60, which is based on the previous "data.csv" file.***

Output: 60

**HOWEVER! 60 can be changed, though it means that the DataFrame contains more than 60 rows, the print(df) statement will return only the headers and the first and last 5 rows.**

**You can change the maximum rows number with the same statement.**



**This is up to 60**

```
    Duration  Pulse  Maxpulse  Calories
0         60    110       130     409.1
1         60    117       145     479.0
2         60    103       135     340.0
3         45    109       175     282.4
4         45    117       148     406.0
5         60    102       127     300.5
6         60    110       136     374.0
7         45    104       134     253.3
8         30    109       133     195.1
9         60     98       124     269.0
10        60    103       147     329.3
11        60    100       120     250.7
12        60    106       128     345.3
13        60    104       132     379.3
14        60     98       123     275.0
15        60     98       120     215.2
16        60    100       120     300.0
17        45     90       112       NaN
18        60    103       123     323.0
19        45     97       125     243.0
20        60    108       131     364.2
21        45    100       119     282.0
22        60    130       101     300.0
23        45    105       132     246.0
24        60    102       126     334.5
25        60    100       120     250.0
26        60     92       118     241.0
27        60    103       132       NaN
```

# Increase the maximum number of rows to display the entire DataFrame

Example:

import pandas as pd

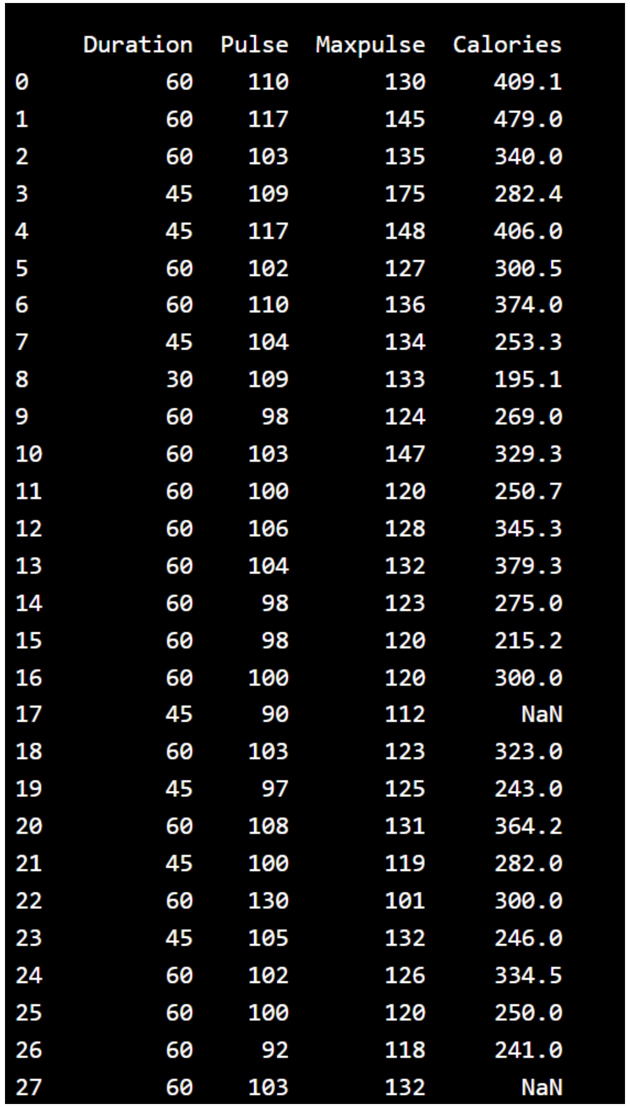
pd.options.display.max_rows = 9999
***#This changes the output settings on how many rows will be displayed upon printing. 9999 is just an arbitrary number, if there are >9999, the succeeding will not be displayed.***

df = pd.read_csv('data.csv')

print(df)

**This is what your *df* contains.**

**Prints up to 9999**
**Rows are from 0-162**



```
    Duration  Pulse  Maxpulse  Calories
0         60    110       130     409.1
1         60    117       145     479.0
2         60    103       135     340.0
3         45    109       175     282.4
4         45    117       148     406.0
5         60    102       127     300.5
6         60    110       136     374.0
7         45    104       134     253.3
8         30    109       133     195.1
9         60     98       124     269.0
10        60    103       147     329.3
11        60    100       120     250.7
12        60    106       128     345.3
13        60    104       132     379.3
14        60     98       123     275.0
15        60     98       120     215.2
16        60    100       120     300.0
17        45     90       112       NaN
18        60    103       123     323.0
19        45     97       125     243.0
20        60    108       131     364.2
21        45    100       119     282.0
22        60    130       101     300.0
23        45    105       132     246.0
24        60    102       126     334.5
25        60    100       120     250.0
26        60     92       118     241.0
27        60    103       132       NaN
```

# Pandas Read JSON

# Read JSON

- Big data sets are often stored or extracted as JSON.
- JSON is plain text, but has the format of an object, and is well known in the world of programming, including Pandas.
- In our examples we will be using a JSON file called 'data.json’.
- For a sample JSON file get the data.json → https://www.w3schools.com/python/pandas/data.js
- Save it in your device by right-clicking and use the ‘save as’, then save with a file name and format data.js

**NOTE: MAKE SURE YOU CAN LOCATE IT!**

# Read JSON Example

#Load the JSON file into a DataFrame using the read_json() function:

import pandas as pd

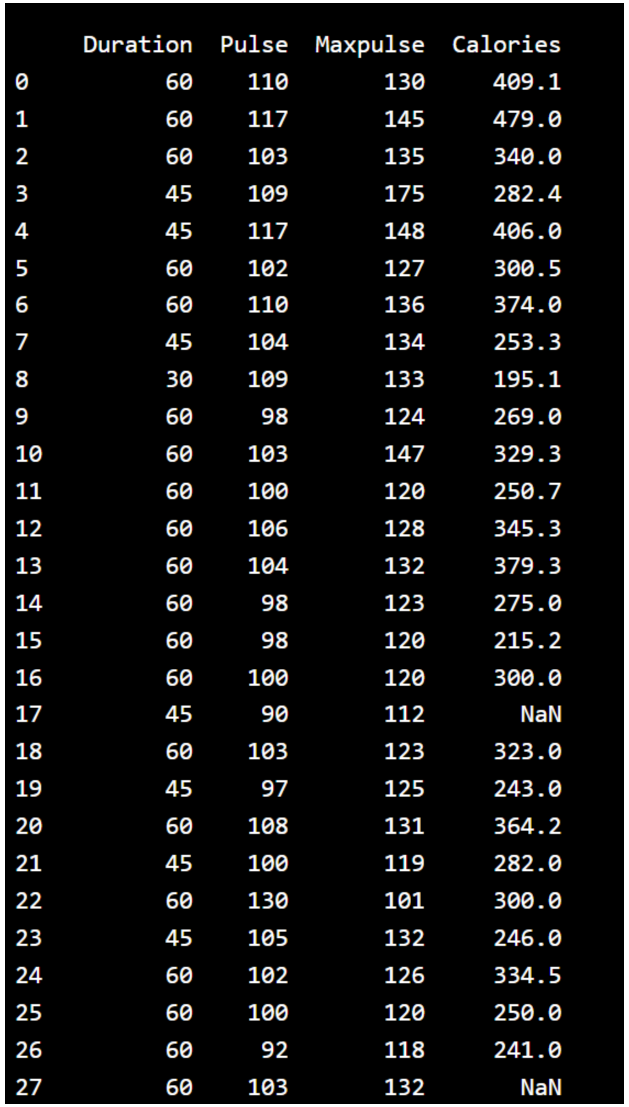
df = pd.read_json('data.json')

print(df.to_string())

#Remember, the to_string() function in the DataFrame class can print your entire dataset depending on the settings of your **options.display.max_rows**

**This is what your *df* contains the same thing. Except, it reads from a JSON .JS file not a .CSV file**

**Prints up to the max_rows setting**

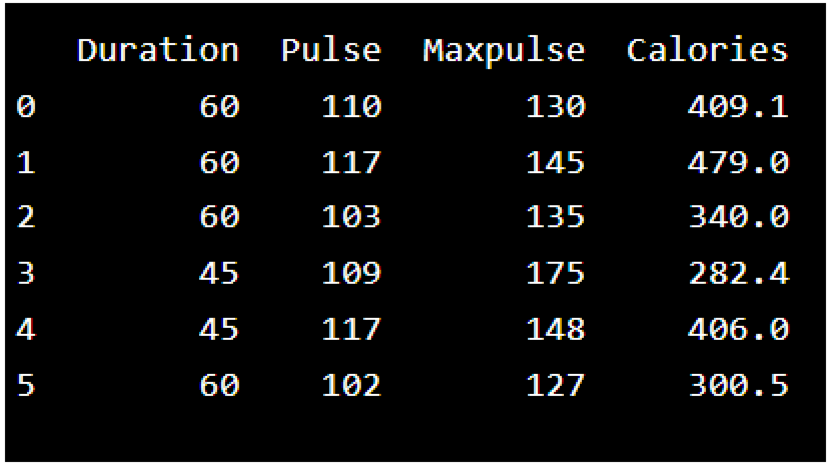


```
    Duration  Pulse  Maxpulse  Calories
0         60    110       130     409.1
1         60    117       145     479.0
2         60    103       135     340.0
3         45    109       175     282.4
4         45    117       148     406.0
5         60    102       127     300.5
6         60    110       136     374.0
7         45    104       134     253.3
8         30    109       133     195.1
9         60     98       124     269.0
10        60    103       147     329.3
11        60    100       120     250.7
12        60    106       128     345.3
13        60    104       132     379.3
14        60     98       123     275.0
15        60     98       120     215.2
16        60    100       120     300.0
17        45     90       112       NaN
18        60    103       123     323.0
19        45     97       125     243.0
20        60    108       131     364.2
21        45    100       119     282.0
22        60    130       101     300.0
23        45    105       132     246.0
24        60    102       126     334.5
25        60    100       120     250.0
26        60     92       118     241.0
27        60    103       132       NaN
```

# Dictionary as JSON

- JSON = Python Dictionary
- JSON objects have the same format as Python dictionaries.
- Video explains what JSON is. It serves as a review.

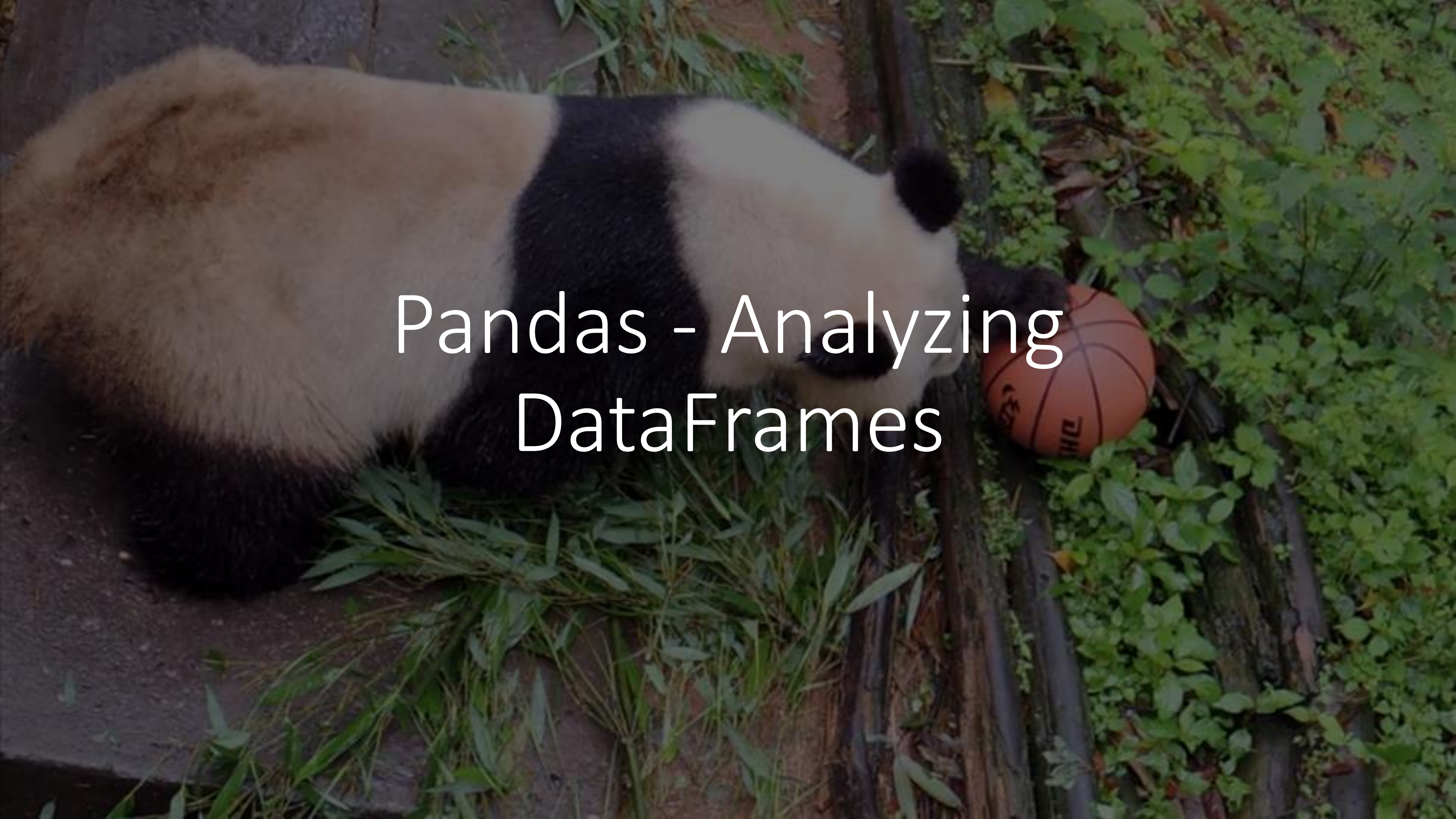
# Dictionary as JSON



```
   Duration  Pulse  Maxpulse  Calories
0        60    110       130     409.1
1        60    117       145     479.0
2        60    103       135     340.0
3        45    109       175     282.4
4        45    117       148     406.0
5        60    102       127     300.5
```

- If your JSON code is not in a file, but in a Python Dictionary, you can load it into a DataFrame directly:

```
import pandas as pd
data = { "Duration":{"0":60,
                      "1":60,
                      "2":60,
                      "3":45,
                      "4":45,
                      "5":60},
        "Pulse":{"0":110,
                 "1":117,
                 "2":103,
                 "3":109,
                 "4":117,
                  "5":102},
       "Maxpulse":{"0":130,
                   "1":145,
                    "2":135,
                    "3":175,
                    "4":148,
                     "5":127},
       "Calories":{"0":409,
                   "1":479,
                   "2":340,
                   "3":282,
                    "4":406,
                    "5":300}
}
df = pd.DataFrame(data)
print(df)
```



# Pandas - Analyzing DataFrames

# Viewing the Data from 0 index

- One of the most used method for getting a quick overview of the DataFrame, is the **head()** method.

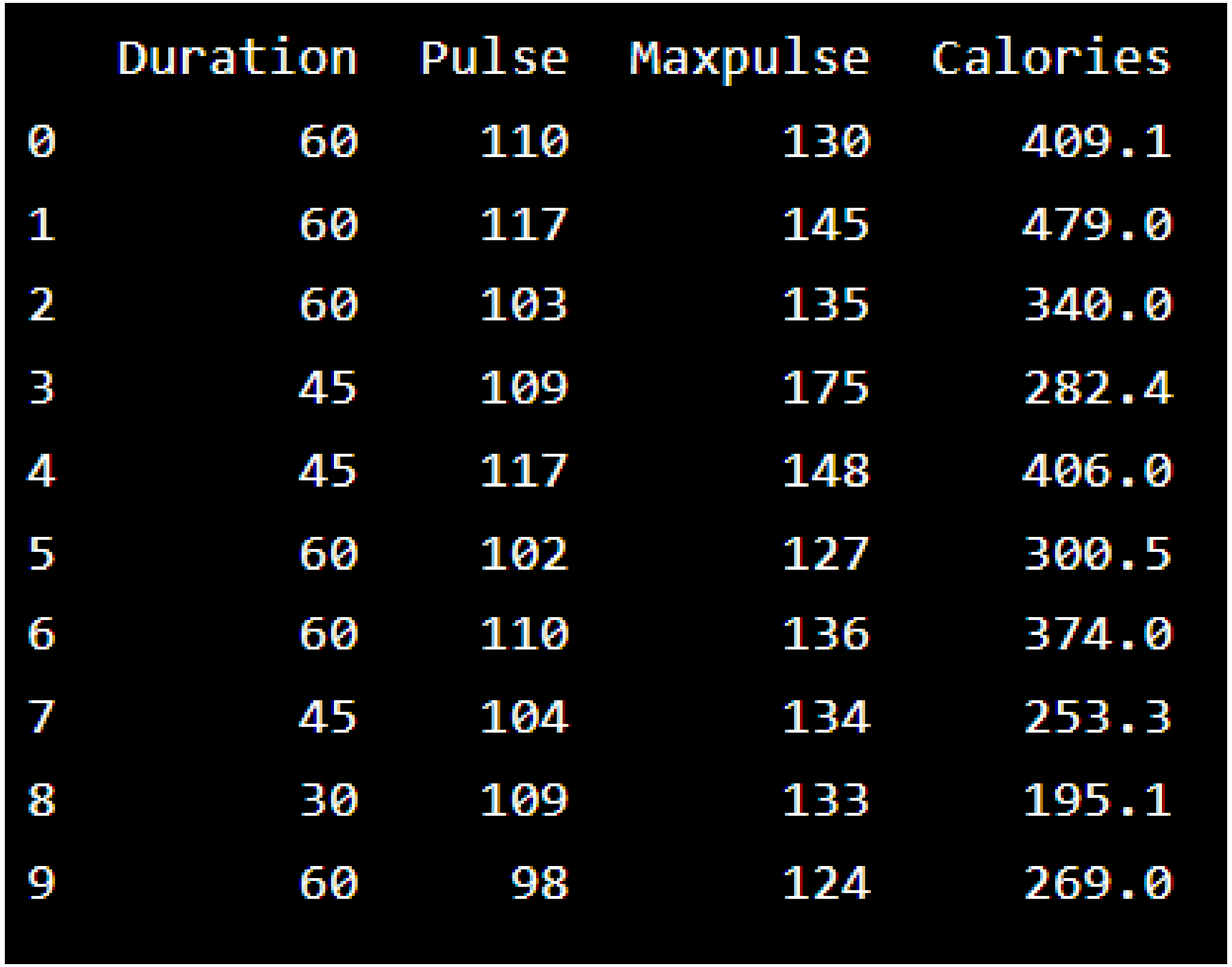
- The **head()** method returns the headers and a specified number of rows, starting from the top.

# Viewing the Data from 0

Example:

#Get a quick overview by printing the first **10** rows of the DataFrame using the **head()** function:

```
import pandas as pd
df = pd.read_csv('data.csv')

print(df.head(10)) #shows only the first 10 indexes starting from 0-9. Leaving the argument value will just print the first 5.
```



```
    Duration  Pulse  Maxpulse  Calories
0         60    110       130     409.1
1         60    117       145     479.0
2         60    103       135     340.0
3         45    109       175     282.4
4         45    117       148     406.0
5         60    102       127     300.5
6         60    110       136     374.0
7         45    104       134     253.3
8         30    109       133     195.1
9         60     98       124     269.0
```

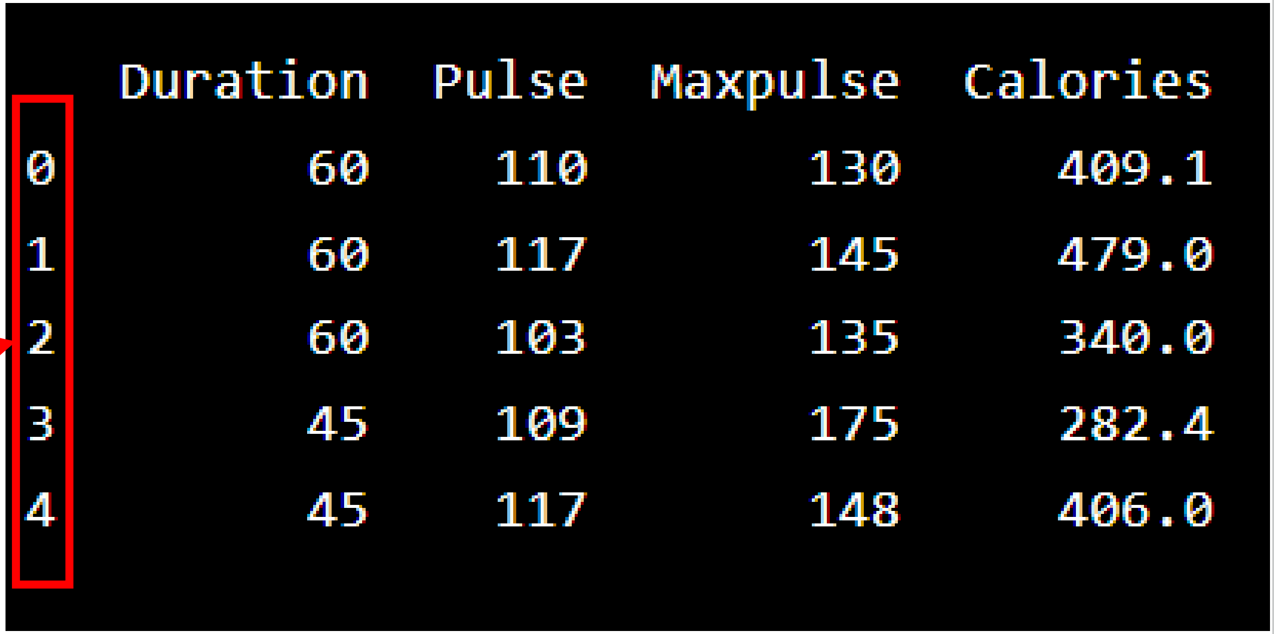
# Viewing the Data from 0

print(df.head(10)) **#shows only the first 10 indexes starting from 0-9. However, leaving the argument value will just print the first 5. Therefore, the output becomes 0-4.**

print(df.head())

*#No value set. The default is =5.*

**Please be reminded that we are still using the same 'data.csv' file from the previous slides.**



```
   Duration  Pulse  Maxpulse  Calories
0        60    110       130     409.1
1        60    117       145     479.0
2        60    103       135     340.0
3        45    109       175     282.4
4        45    117       148     406.0
```

# Viewing the Data from the last index

- There is also a **tail()** method for viewing the last rows of the DataFrame.

- The **tail()** method returns the headers and a specified number of rows, starting from the bottom.

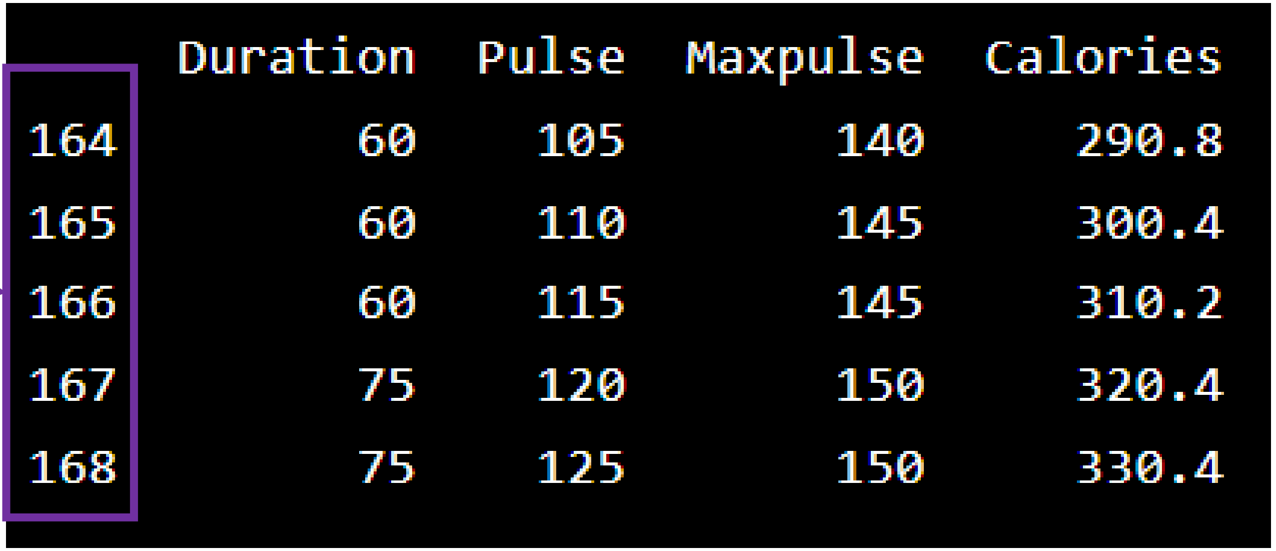
# Viewing the Data from the last index

Example:

#Print the **last 5 rows** of the DataFrame:

print(df.**tail()**)

***#NOTE: We are still using the same 'data.csv'. Again, if the default value is not specified, argument value becomes =5.***

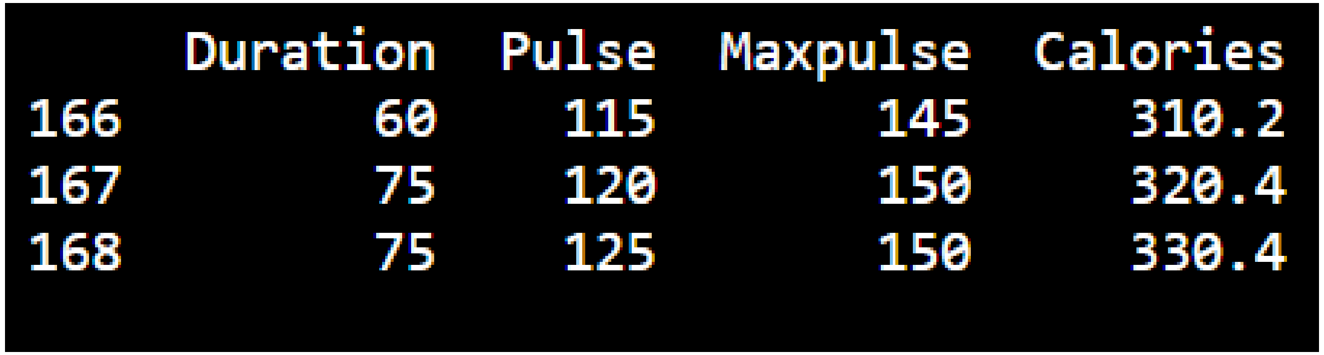


```
     Duration  Pulse  Maxpulse  Calories
164        60    105       140     290.8
165        60    110       145     300.4
166        60    115       145     310.2
167        75    120       150     320.4
168        75    125       150     330.4
```

# Viewing the Data from the last index

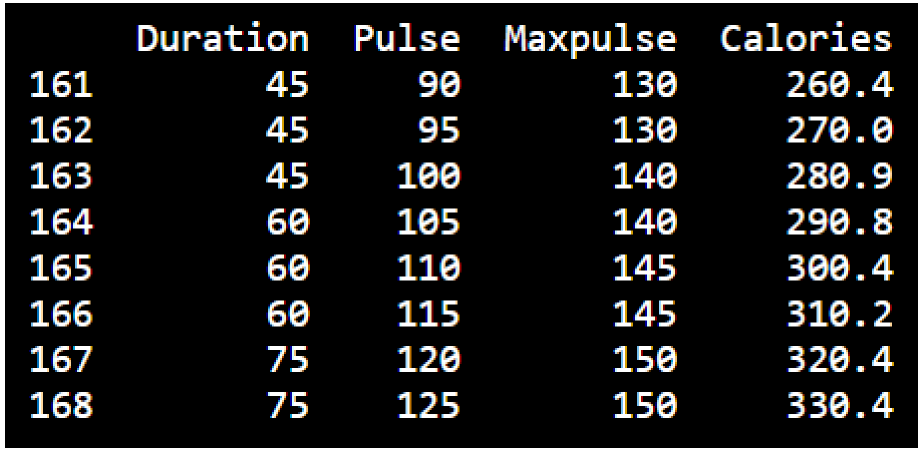
Examples:

print(df.tail(3))



```
     Duration  Pulse  Maxpulse  Calories
166        60    115       145     310.2
167        75    120       150     320.4
168        75    125       150     330.4
```

print(df.tail(8))



```
     Duration  Pulse  Maxpulse  Calories
161        45     90       130     260.4
162        45     95       130     270.0
163        45    100       140     280.9
164        60    105       140     290.8
165        60    110       145     300.4
166        60    115       145     310.2
167        75    120       150     320.4
168        75    125       150     330.4
```

# Info About the Data

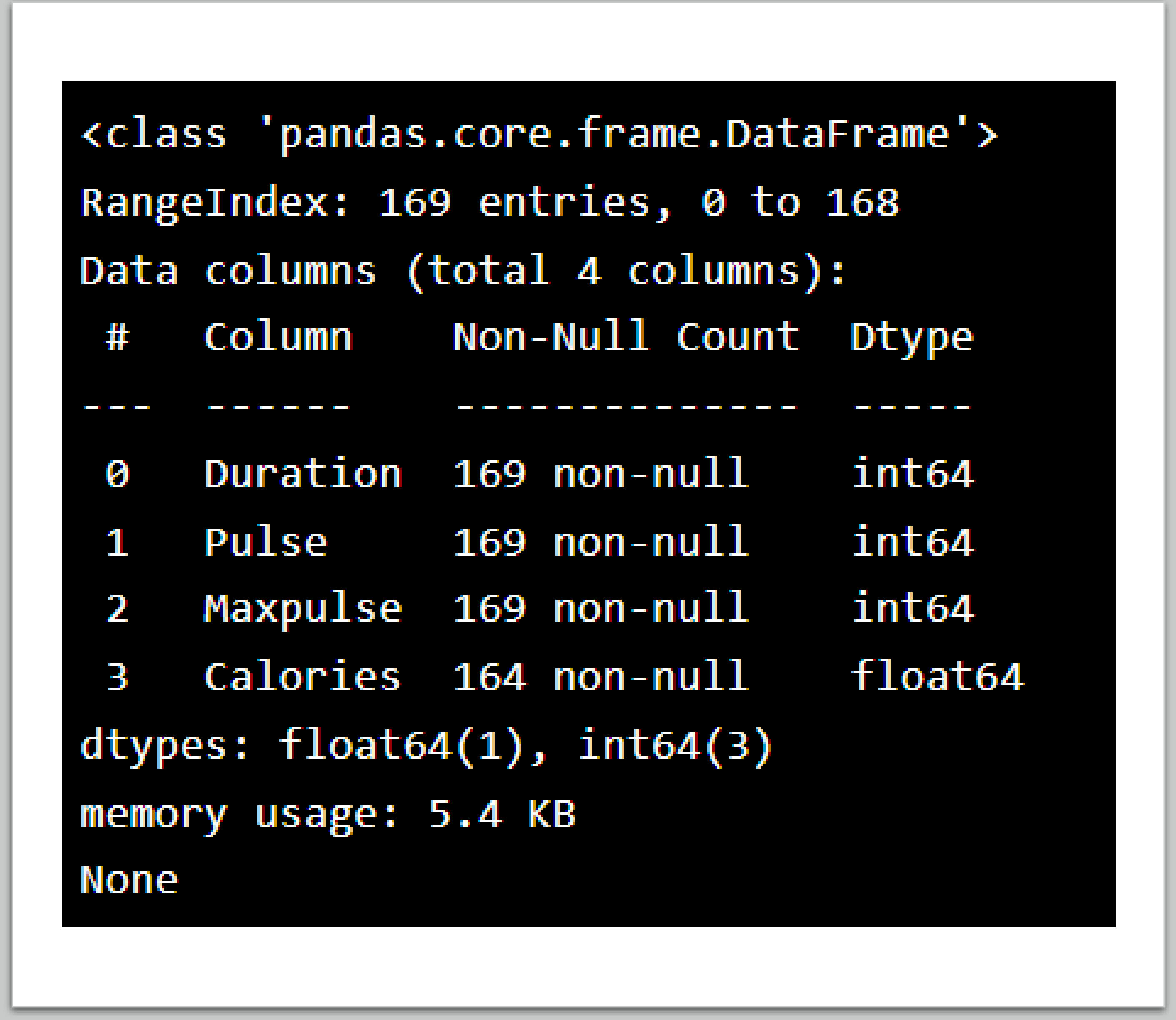
- The DataFrames object has a method called **info()**, that gives you more information about the data set.

# Info About the Data

Example:

Print information about the data:

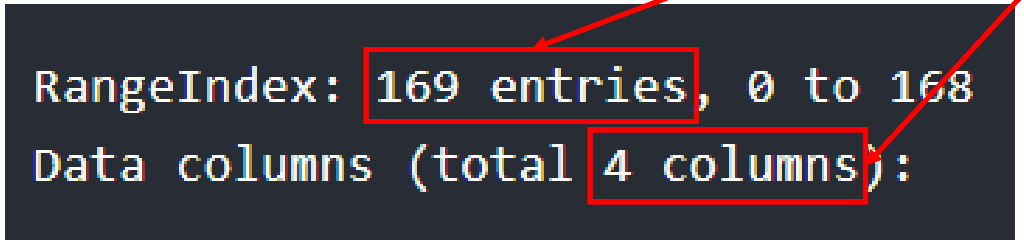
print(df.**info()**)



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 169 entries, 0 to 168
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Duration  169 non-null    int64
 1   Pulse     169 non-null    int64
 2   Maxpulse  169 non-null    int64
 3   Calories  164 non-null    float64
dtypes: float64(1), int64(3)
memory usage: 5.4 KB
None
```

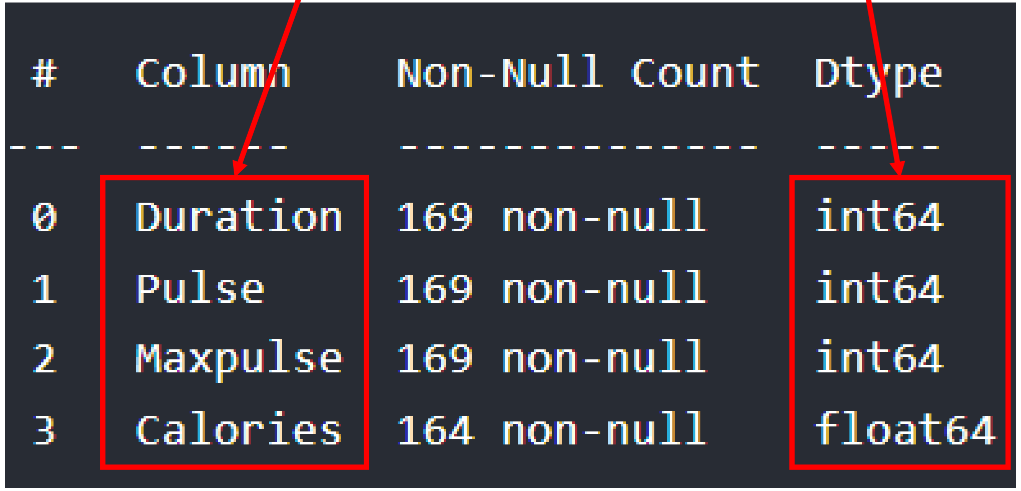
# Info Results Explanation

- The result tells us there are **169 rows** and **4 columns**



```
RangeIndex: 169 entries, 0 to 168
Data columns (total 4 columns):
```

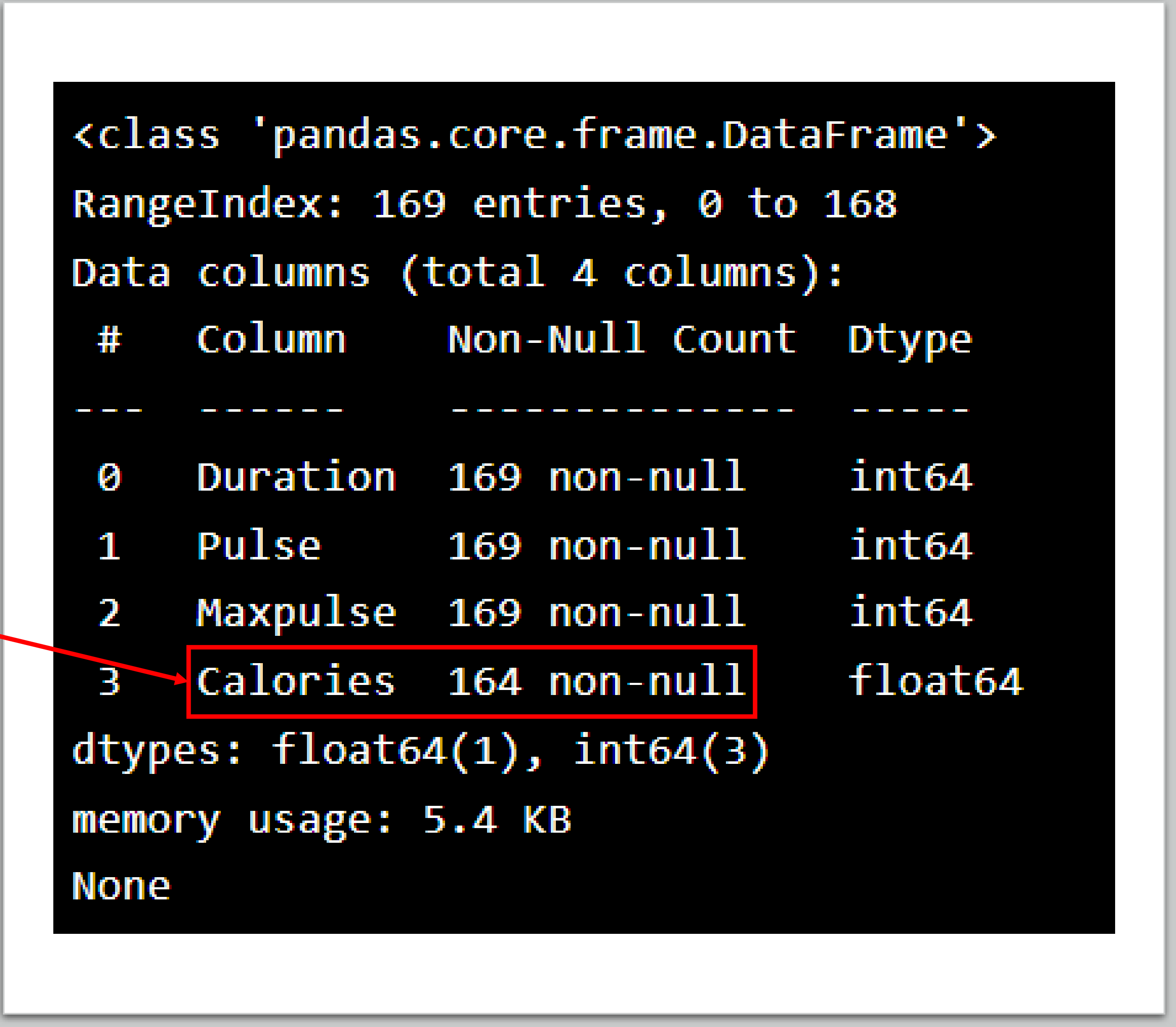
The **name of each column**, with the **data type**:



```
#   Column    Non-Null Count  Dtype
---  ------    --------------  -----
0   Duration  169 non-null    int64
1   Pulse     169 non-null    int64
2   Maxpulse  169 non-null    int64
3   Calories  164 non-null    float64
```

# Null Values

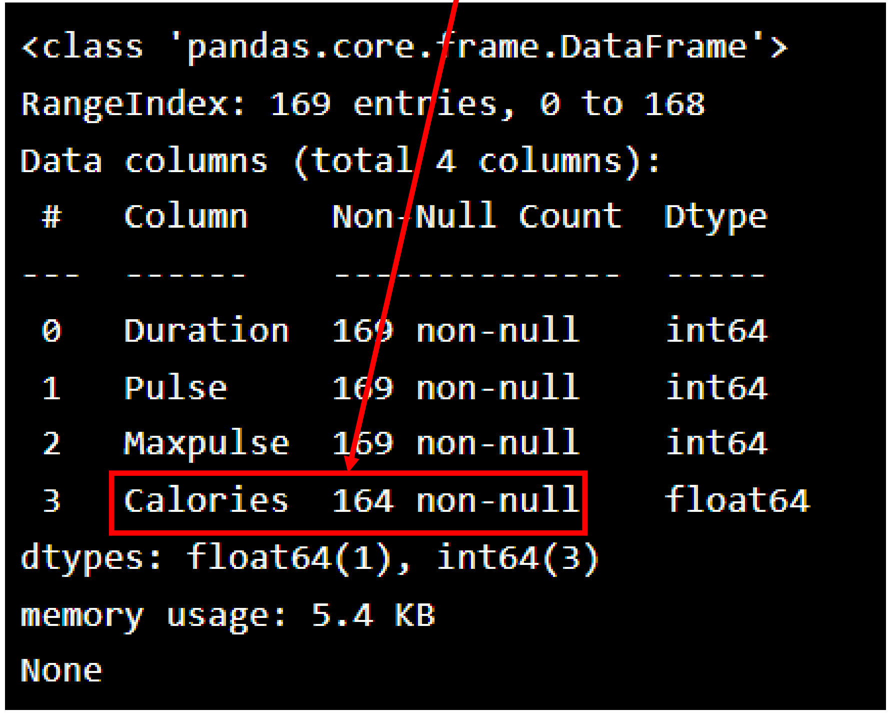
- The **info()** method also tells us how many Non-Null values there are present in each column, and in our data set it seems like **there are 164 of 169 Non-Null values in the "Calories" column.**
- **Empty values, or Null values, can be bad when analyzing data, and you should consider removing rows with empty values. This is a step towards what is called cleaning data, and you will learn more about that in the next chapters.**



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 169 entries, 0 to 168
Data columns (total 4 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Duration   169 non-null    int64
 1   Pulse      169 non-null    int64
 2   Maxpulse   169 non-null    int64
 3   Calories   164 non-null    float64
dtypes: float64(1), int64(3)
memory usage: 5.4 KB
None
```

# Null Values

- **5 rows in “Calories” has no value at all.**



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 169 entries, 0 to 168
Data columns (total 4 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Duration   169 non-null    int64
 1   Pulse      169 non-null    int64
 2   Maxpulse   169 non-null    int64
 3   Calories   164 non-null    float64
dtypes: float64(1), int64(3)
memory usage: 5.4 KB
None
```

Non-zero value

0

null

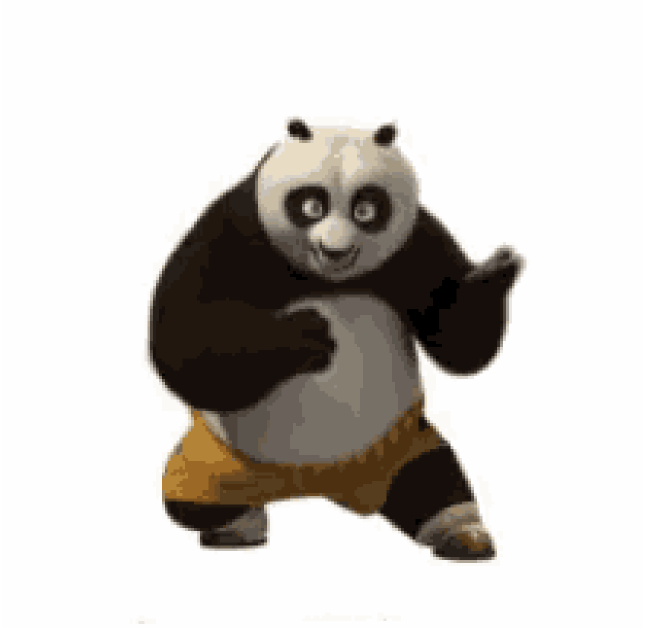
undefined

# Exercise 3

# E3-Q1

- Insert the correct syntax for loading JSON files into a DataFrame.

df.__________(data)



# E3-Q2

- Insert the correct syntax for returning the headers and the first 32 rows of a DataFrame.

df.________

# E3-Q3

- When using the head() method, how many rows are returned if you do not specify the number?

_ rows

# E3-Q4

- If head() method returns the first rows, what method returns the last rows?

df.____

# E3-Q5

- Insert the correct syntax to return the entire DataFrame.

df.___________

# Exercise 3 Accomplished!

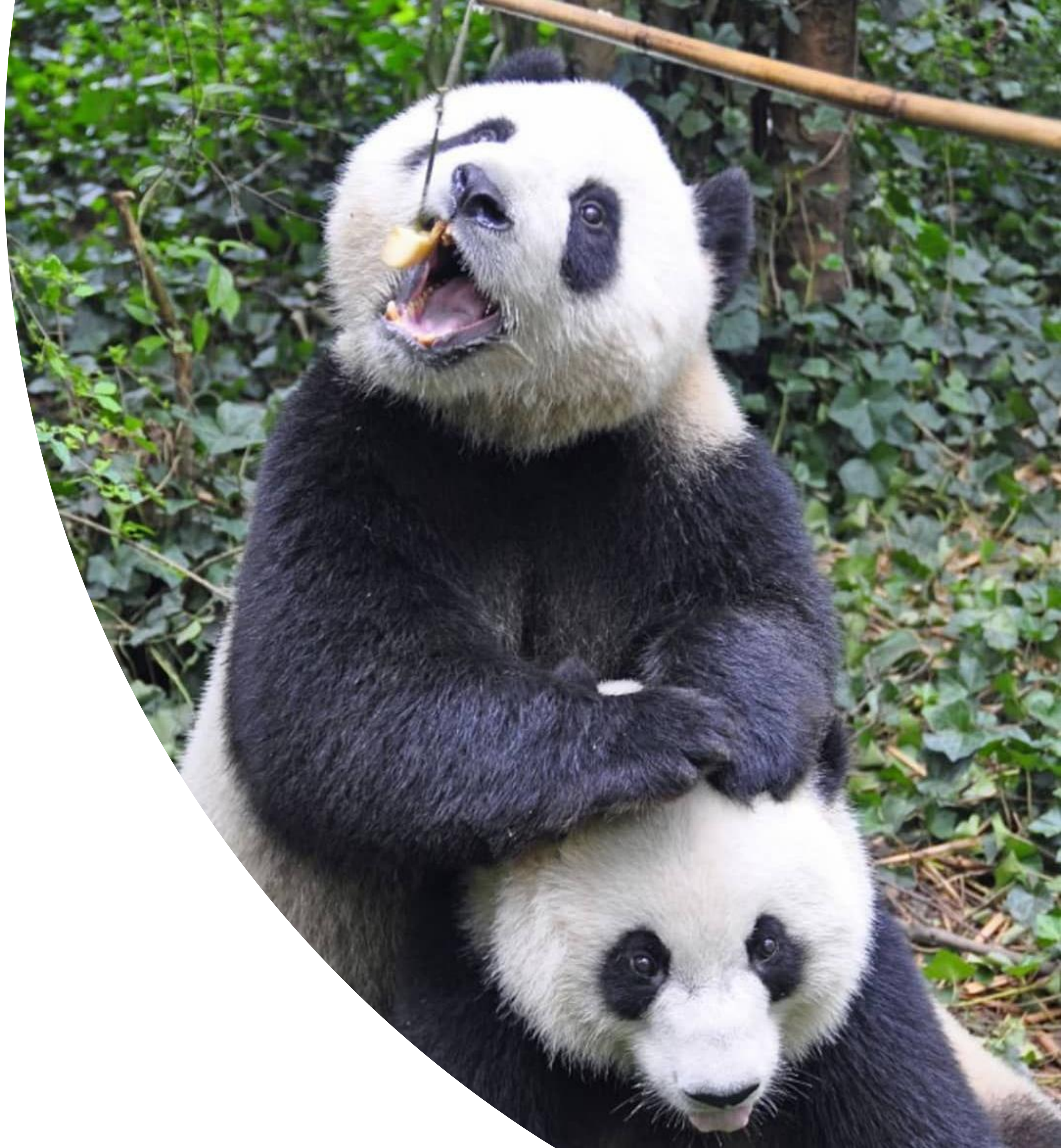
# Pandas - Cleaning Data

# Data Cleaning

- Data cleaning means fixing bad data in your data set.

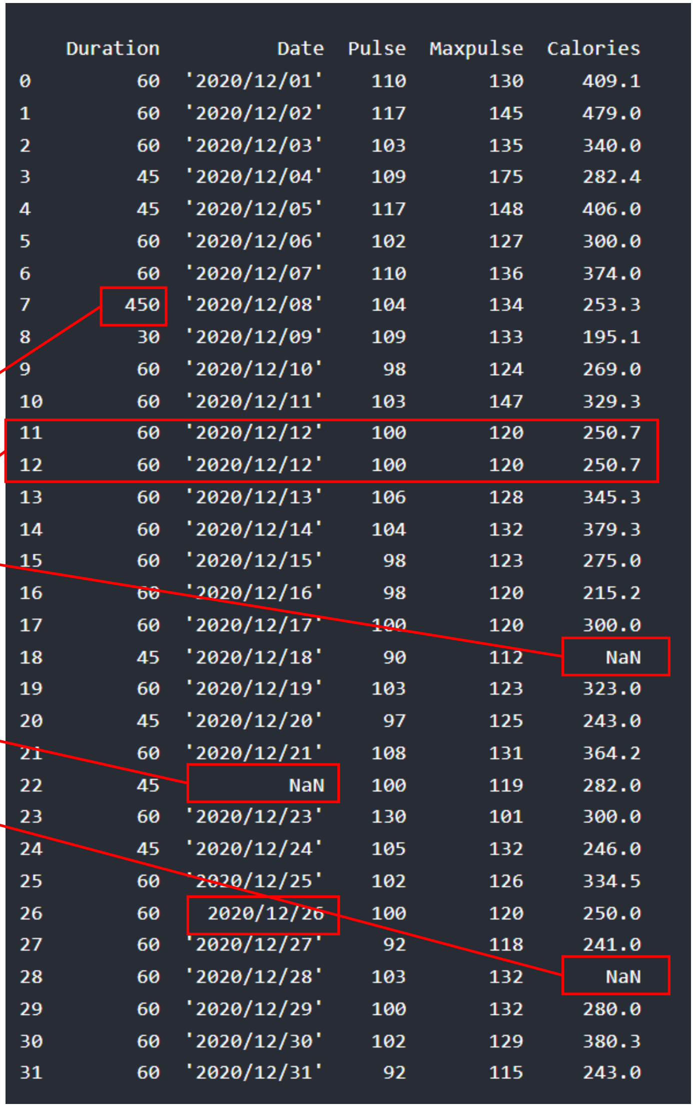
**Bad data could be:**

- Empty cells
- Data in wrong format
- Wrong data
- Duplicates



# Dataset for this lecture

- The data set contains some empty cells ("Date" in row 22, and "Calories" in row 18 and 28).
- The data set contains wrong format ("Date" in row 26).
- The data set contains wrong data ("Duration" in row 7).
- The data set contains duplicates (row 11 and 12).
- **NAN** means, **MISSING VALUE!**



```
    Duration          Date  Pulse  Maxpulse  Calories
0         60  '2020/12/01'    110       130     409.1
1         60  '2020/12/02'    117       145     479.0
2         60  '2020/12/03'    103       135     340.0
3         45  '2020/12/04'    109       175     282.4
4         45  '2020/12/05'    117       148     406.0
5         60  '2020/12/06'    102       127     300.0
6         60  '2020/12/07'    110       136     374.0
7        450  '2020/12/08'    104       134     253.3
8         30  '2020/12/09'    109       133     195.1
9         60  '2020/12/10'     98       124     269.0
10        60  '2020/12/11'    103       147     329.3
11        60  '2020/12/12'    100       120     250.7
12        60  '2020/12/12'    100       120     250.7
13        60  '2020/12/13'    106       128     345.3
14        60  '2020/12/14'    104       132     379.3
15        60  '2020/12/15'     98       123     275.0
16        60  '2020/12/16'     98       120     215.2
17        60  '2020/12/17'    100       120     300.0
18        45  '2020/12/18'     90       112       NaN
19        60  '2020/12/19'    103       123     323.0
20        45  '2020/12/20'     97       125     243.0
21        60  '2020/12/21'    108       131     364.2
22        45           NaN    100       119     282.0
23        60  '2020/12/23'    130       101     300.0
24        45  '2020/12/24'    105       132     246.0
25        60  '2020/12/25'    102       126     334.5
26        60    2020/12/26    100       120     250.0
27        60  '2020/12/27'     92       118     241.0
28        60  '2020/12/28'    103       132       NaN
29        60  '2020/12/29'    100       132     280.0
30        60  '2020/12/30'    102       129     380.3
31        60  '2020/12/31'     92       115     243.0
```

# Links to access the dataset

- Download the dataset first → https://www.w3schools.com/python/pandas/dirtydata.csv

- You can also just open it if you want to (Optional) → https://www.w3schools.com/python/pandas/dirtydata.csv.txt

# Pandas - Cleaning Empty Cells

# Empty Cells

- Empty cells can potentially give you a wrong result when you analyze data.
- Keep all your cells checked!

# Checking Empty Cells

- While making a Data Frame from a csv file, many blank columns are imported as null value into the Data Frame which later creates problems while operating that data frame.
- Pandas **isnull()** and **notnull()** methods are used to check and manage **NULL** values in a data frame.

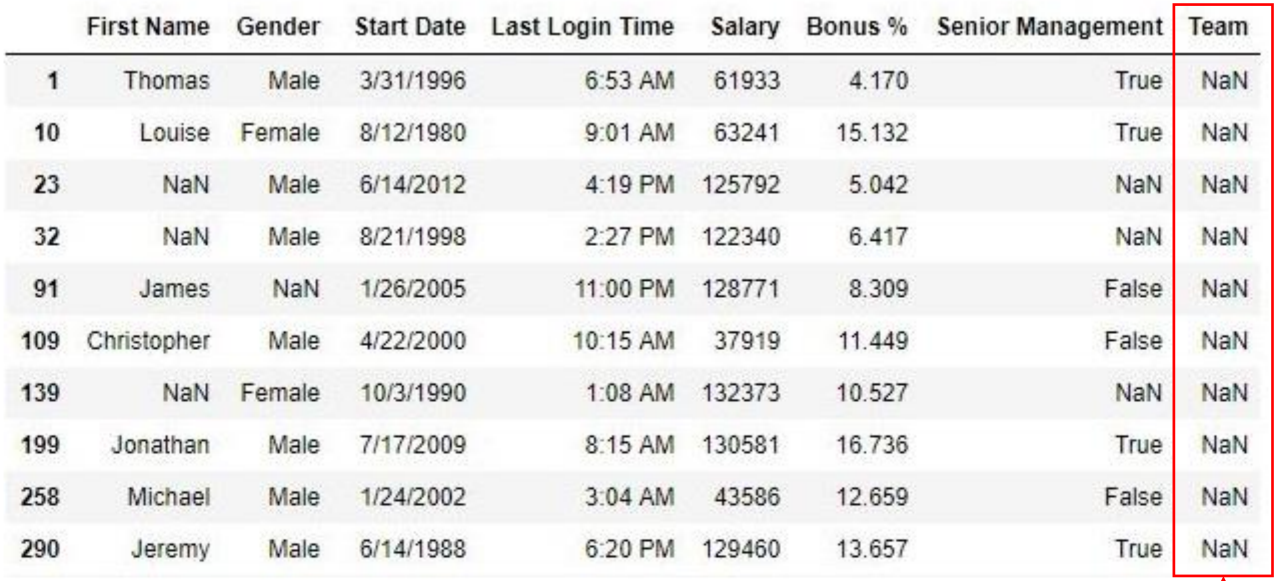
# Dataframe.isnull()

```
# importing pandas package
import pandas as pd

# making data frame from csv file
data = pd.read_csv("employees.csv")

# creating bool series True for NaN values
bool_series = pd.isnull(data["Team"])

# filtering data
# displayind data only with team = NaN
data[bool_series]
```

# Dataframe.isnull() Output

- As shown in output image, only the rows having Team=NULL are displayed.

```
# importing pandas package
import pandas as pd

# making data frame from csv file
data = pd.read_csv("employees.csv")

# creating bool series True for NaN values
bool_series = pd.isnull(data["Team"])

# filtering data
# displayind data only with team = NaN
data[bool_series]
```



| | First Name | Gender | Start Date | Last Login Time | Salary | Bonus % | Senior Management | Team |
|---|---|---|---|---|---|---|---|---|
| **1** | Thomas | Male | 3/31/1996 | 6:53 AM | 61933 | 4.170 | True | NaN |
| **10** | Louise | Female | 8/12/1980 | 9:01 AM | 63241 | 15.132 | True | NaN |
| **23** | NaN | Male | 6/14/2012 | 4:19 PM | 125792 | 5.042 | NaN | NaN |
| **32** | NaN | Male | 8/21/1998 | 2:27 PM | 122340 | 6.417 | NaN | NaN |
| **91** | James | NaN | 1/26/2005 | 11:00 PM | 128771 | 8.309 | False | NaN |
| **109** | Christopher | Male | 4/22/2000 | 10:15 AM | 37919 | 11.449 | False | NaN |
| **139** | NaN | Female | 10/3/1990 | 1:08 AM | 132373 | 10.527 | NaN | NaN |
| **199** | Jonathan | Male | 7/17/2009 | 8:15 AM | 130581 | 16.736 | True | NaN |
| **258** | Michael | Male | 1/24/2002 | 3:04 AM | 43586 | 12.659 | False | NaN |
| **290** | Jeremy | Male | 6/14/1988 | 6:20 PM | 129460 | 13.657 | True | NaN |

# Dataframe.notnull()

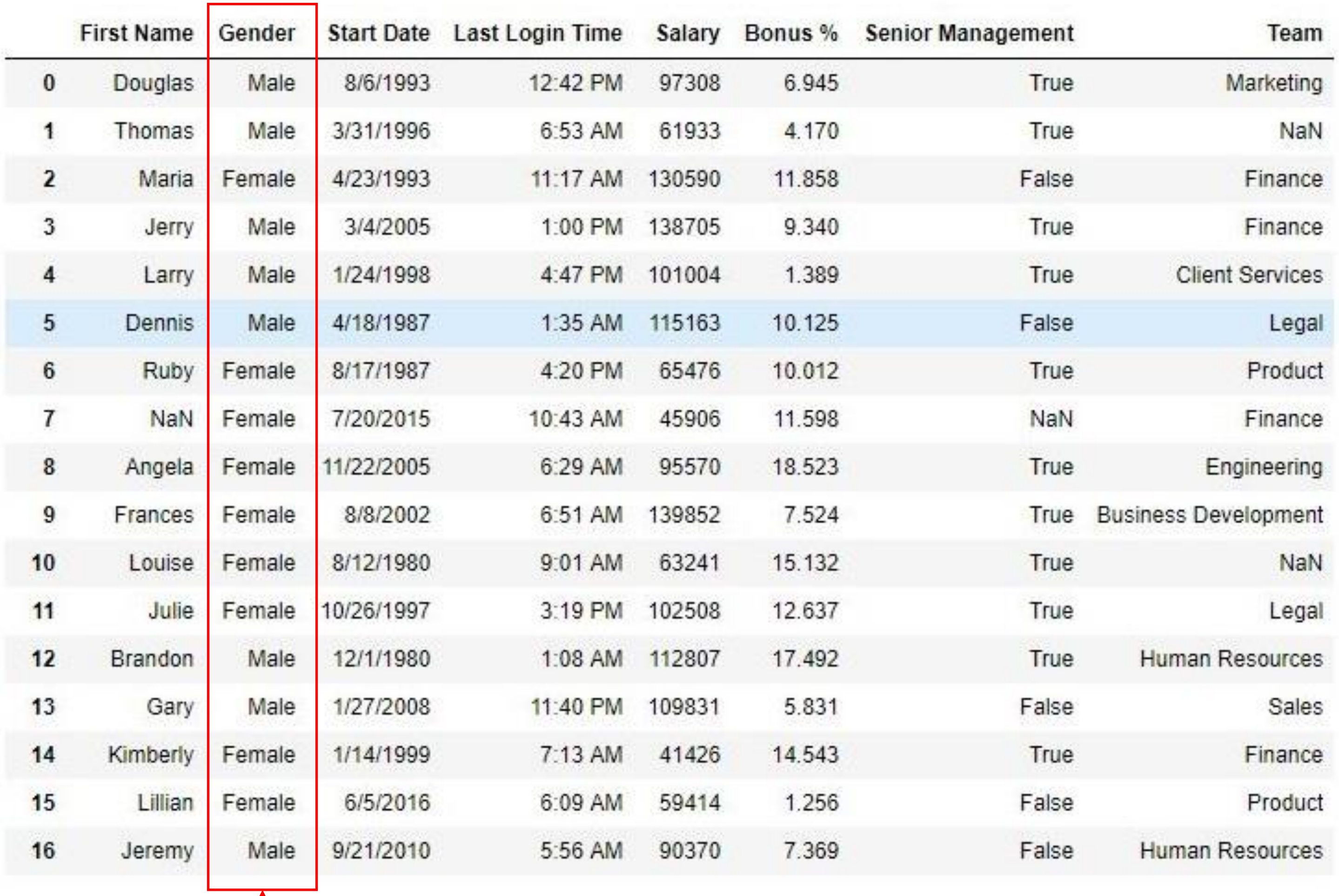
- In the following example, Gender column is checked for NULL values and a Boolean series is returned by the **notnull()** method which stores True for ever NON-NULL value and False for a null value.

```
# importing pandas package
import pandas as pd

# making data frame from csv file
data = pd.read_csv("employees.csv")

# creating bool series False for NaN values
bool_series = pd.notnull(data["Gender"])

# displayed data only with team = NaN
data[bool_series]
```

# Dataframe.notnull() Output

- As shown in output image, only the rows having some value in Gender are displayed.

```
import pandas as pd

data = pd.read_csv("employees.csv")

# creating bool series False for NaN values
bool_series = pd.notnull(data["Gender"])

# displayed data only with team = NaN
data[bool_series]
```



| | First Name | Gender | Start Date | Last Login Time | Salary | Bonus % | Senior Management | Team |
|---|---|---|---|---|---|---|---|---|
| 0 | Douglas | Male | 8/6/1993 | 12:42 PM | 97308 | 6.945 | True | Marketing |
| 1 | Thomas | Male | 3/31/1996 | 6:53 AM | 61933 | 4.170 | True | NaN |
| 2 | Maria | Female | 4/23/1993 | 11:17 AM | 130590 | 11.858 | False | Finance |
| 3 | Jerry | Male | 3/4/2005 | 1:00 PM | 138705 | 9.340 | True | Finance |
| 4 | Larry | Male | 1/24/1998 | 4:47 PM | 101004 | 1.389 | True | Client Services |
| 5 | Dennis | Male | 4/18/1987 | 1:35 AM | 115163 | 10.125 | False | Legal |
| 6 | Ruby | Female | 8/17/1987 | 4:20 PM | 65476 | 10.012 | True | Product |
| 7 | NaN | Female | 7/20/2015 | 10:43 AM | 45906 | 11.598 | NaN | Finance |
| 8 | Angela | Female | 11/22/2005 | 6:29 AM | 95570 | 18.523 | True | Engineering |
| 9 | Frances | Female | 8/8/2002 | 6:51 AM | 139852 | 7.524 | True | Business Development |
| 10 | Louise | Female | 8/12/1980 | 9:01 AM | 63241 | 15.132 | True | NaN |
| 11 | Julie | Female | 10/26/1997 | 3:19 PM | 102508 | 12.637 | True | Legal |
| 12 | Brandon | Male | 12/1/1980 | 1:08 AM | 112807 | 17.492 | True | Human Resources |
| 13 | Gary | Male | 1/27/2008 | 11:40 PM | 109831 | 5.831 | False | Sales |
| 14 | Kimberly | Female | 1/14/1999 | 7:13 AM | 41426 | 14.543 | True | Finance |
| 15 | Lillian | Female | 6/5/2016 | 6:09 AM | 59414 | 1.256 | False | Product |
| 16 | Jeremy | Male | 9/21/2010 | 5:56 AM | 90370 | 7.369 | False | Human Resources |

# Remove Rows

- One way to deal with empty cells is to remove rows that contain empty cells.

- This is usually OK, since data sets can be very big, and removing a few rows will not have a big impact on the result.

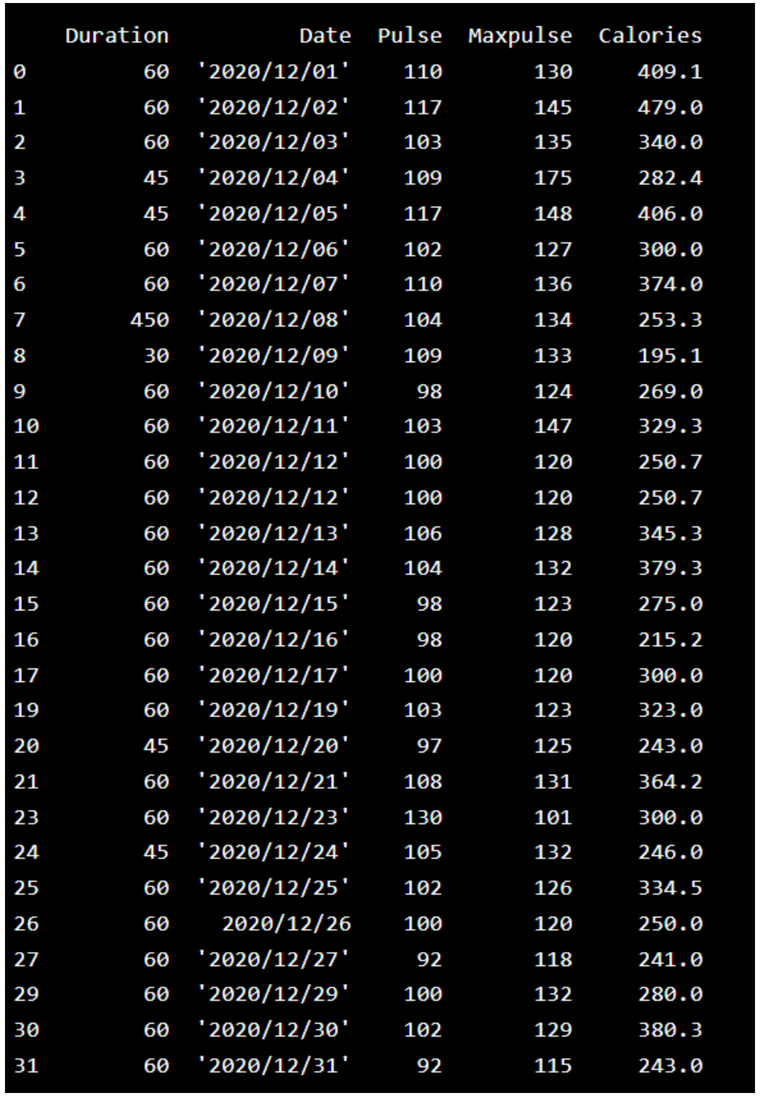
# Remove Rows

- Return a new Data Frame with no empty cells:

```
import pandas as pd #imports pandas with alias
pd

df = pd.read_csv('data.csv’) #reads the csv
file

new_df = df.dropna() #drops all rows with NAN

print(new_df.to_string()) #print outs the
entire dataset
```

# Output

- After using the dropna() function for the dataset ‘df.**dropna()**’, all rows with **NAN** are dropped.
- 18, 22, and 28 are dropped because of the **dropna()** function.



```
    Duration          Date  Pulse  Maxpulse  Calories
0         60  '2020/12/01'    110       130     409.1
1         60  '2020/12/02'    117       145     479.0
2         60  '2020/12/03'    103       135     340.0
3         45  '2020/12/04'    109       175     282.4
4         45  '2020/12/05'    117       148     406.0
5         60  '2020/12/06'    102       127     300.0
6         60  '2020/12/07'    110       136     374.0
7        450  '2020/12/08'    104       134     253.3
8         30  '2020/12/09'    109       133     195.1
9         60  '2020/12/10'     98       124     269.0
10        60  '2020/12/11'    103       147     329.3
11        60  '2020/12/12'    100       120     250.7
12        60  '2020/12/12'    100       120     250.7
13        60  '2020/12/13'    106       128     345.3
14        60  '2020/12/14'    104       132     379.3
15        60  '2020/12/15'     98       123     275.0
16        60  '2020/12/16'     98       120     215.2
17        60  '2020/12/17'    100       120     300.0
19        60  '2020/12/19'    103       123     323.0
20        45  '2020/12/20'     97       125     243.0
21        60  '2020/12/21'    108       131     364.2
23        60  '2020/12/23'    130       101     300.0
24        45  '2020/12/24'    105       132     246.0
25        60  '2020/12/25'    102       126     334.5
26        60    2020/12/26    100       120     250.0
27        60  '2020/12/27'     92       118     241.0
29        60  '2020/12/29'    100       132     280.0
30        60  '2020/12/30'    102       129     380.3
31        60  '2020/12/31'     92       115     243.0
```

# Remove Rows

- Note: By default, the **dropna()** method returns a new DataFrame, and will not change the original.
- If you want to change the original DataFrame, use the inplace = True argument:

```
import pandas as pd

df = pd.read_csv('data.csv')

df.dropna(inplace = True) #inplace argument = True  will NOT return a new DataFrame, but it will remove all rows containing NULL values from the original DataFrame.

print(df.to_string())
```

# Replace Empty Values

- Another way of dealing with empty cells is to insert a new value instead.

- This way you do not have to delete entire rows just because of some empty cells.

- The **fillna()** method allows us to replace empty cells with a value:

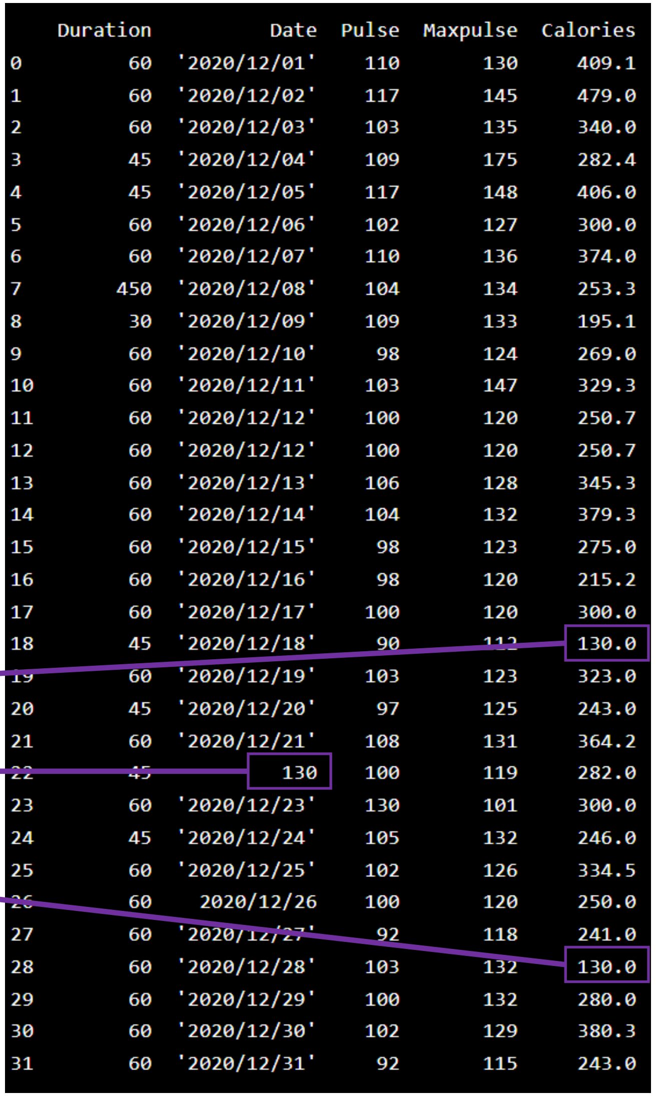
# Replace Empty Values

- Replace NULL values with the number 130:

```
import pandas as pd

df = pd.read_csv('data.csv')

df.fillna(130, inplace = True) #all values in the dataframe ‘df’ will have its values replaced with ‘130.’
```

# Replace Empty Values Output

- Because of the **fillna()** function, all rows with an NAN value had been replaced with the value given, which is **130**.
- Empty cells got the value **130** (in row **18**, **22** and **28**).



```
    Duration          Date  Pulse  Maxpulse  Calories
0         60  '2020/12/01'    110       130     409.1
1         60  '2020/12/02'    117       145     479.0
2         60  '2020/12/03'    103       135     340.0
3         45  '2020/12/04'    109       175     282.4
4         45  '2020/12/05'    117       148     406.0
5         60  '2020/12/06'    102       127     300.0
6         60  '2020/12/07'    110       136     374.0
7        450  '2020/12/08'    104       134     253.3
8         30  '2020/12/09'    109       133     195.1
9         60  '2020/12/10'     98       124     269.0
10        60  '2020/12/11'    103       147     329.3
11        60  '2020/12/12'    100       120     250.7
12        60  '2020/12/12'    100       120     250.7
13        60  '2020/12/13'    106       128     345.3
14        60  '2020/12/14'    104       132     379.3
15        60  '2020/12/15'     98       123     275.0
16        60  '2020/12/16'     98       120     215.2
17        60  '2020/12/17'    100       120     300.0
18        45  '2020/12/18'     90       112     130.0
19        60  '2020/12/19'    103       123     323.0
20        45  '2020/12/20'     97       125     243.0
21        60  '2020/12/21'    108       131     364.2
22        45           130    100       119     282.0
23        60  '2020/12/23'    130       101     300.0
24        45  '2020/12/24'    105       132     246.0
25        60  '2020/12/25'    102       126     334.5
26        60    2020/12/26    100       120     250.0
27        60  '2020/12/27'     92       118     241.0
28        60  '2020/12/28'    103       132     130.0
29        60  '2020/12/29'    100       132     280.0
30        60  '2020/12/30'    102       129     380.3
31        60  '2020/12/31'     92       115     243.0
```

# Replace Only For Specified Columns

- The example above replaces all empty cells in the whole Data Frame.

- To only replace empty values for one column, specify the column name for the DataFrame:

# Replace Only For Specified Columns

- Replace NULL values in the "Calories" columns with the number 130:

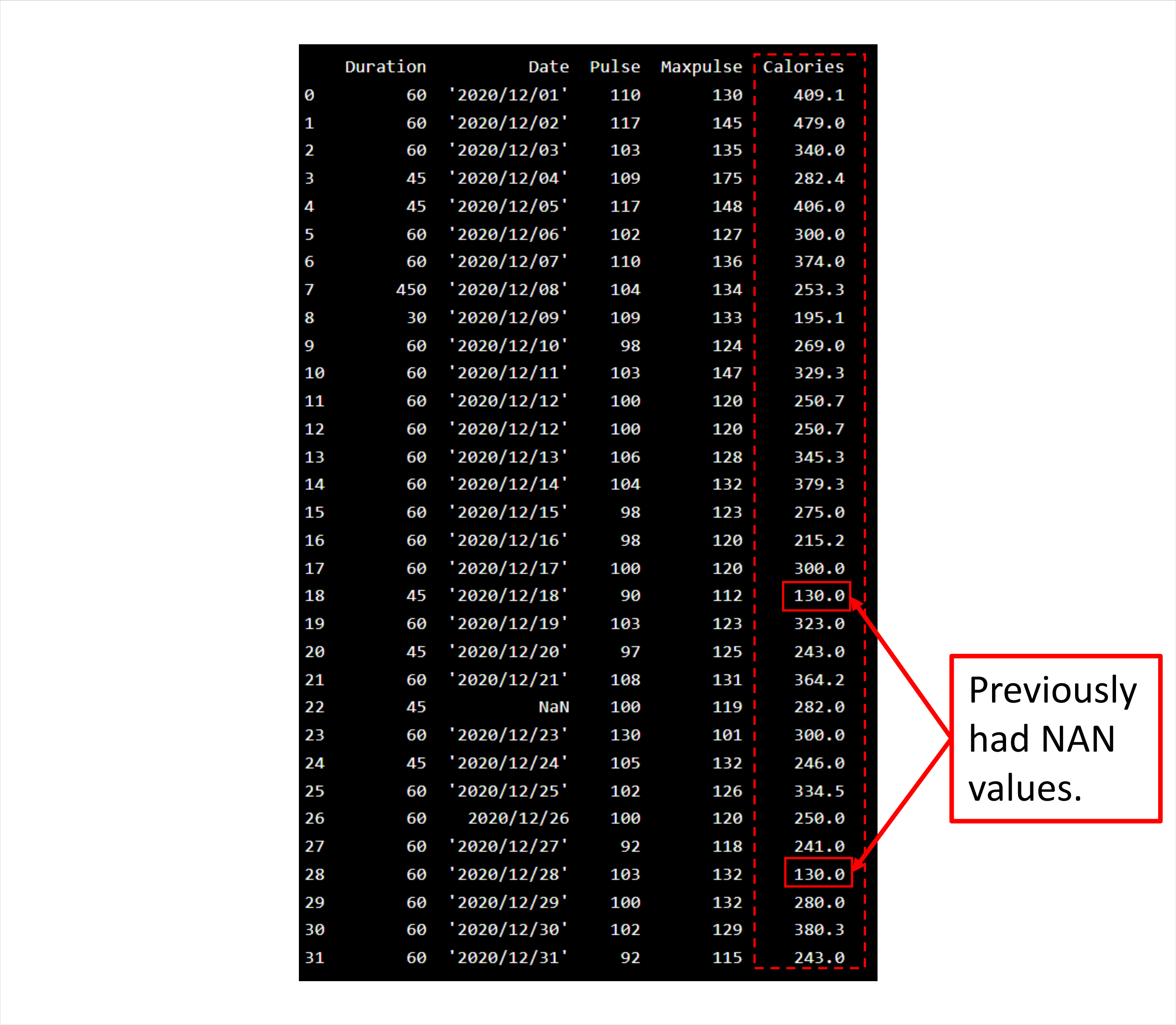
```
import pandas as pd

df = pd.read_csv('data.csv')

df["Calories"].fillna(130, inplace = True)
```

# Replace Only For Specified Columns Output

- Using the **fillna()** function with the **df['Calories']** allows a specified replacement.
- With the example:

**df['Calories'].fillna(**130, inplace=True**)**

**Only the empty row fields in the 'Calories' column got replaced with 130.**



```
    Duration          Date  Pulse  Maxpulse  Calories
0         60  '2020/12/01'    110       130     409.1
1         60  '2020/12/02'    117       145     479.0
2         60  '2020/12/03'    103       135     340.0
3         45  '2020/12/04'    109       175     282.4
4         45  '2020/12/05'    117       148     406.0
5         60  '2020/12/06'    102       127     300.0
6         60  '2020/12/07'    110       136     374.0
7        450  '2020/12/08'    104       134     253.3
8         30  '2020/12/09'    109       133     195.1
9         60  '2020/12/10'     98       124     269.0
10        60  '2020/12/11'    103       147     329.3
11        60  '2020/12/12'    100       120     250.7
12        60  '2020/12/12'    100       120     250.7
13        60  '2020/12/13'    106       128     345.3
14        60  '2020/12/14'    104       132     379.3
15        60  '2020/12/15'     98       123     275.0
16        60  '2020/12/16'     98       120     215.2
17        60  '2020/12/17'    100       120     300.0
18        45  '2020/12/18'     90       112     130.0
19        60  '2020/12/19'    103       123     323.0
20        45  '2020/12/20'     97       125     243.0
21        60  '2020/12/21'    108       131     364.2
22        45           NaN    100       119     282.0
23        60  '2020/12/23'    130       101     300.0
24        45  '2020/12/24'    105       132     246.0
25        60  '2020/12/25'    102       126     334.5
26        60    2020/12/26    100       120     250.0
27        60  '2020/12/27'     92       118     241.0
28        60  '2020/12/28'    103       132     130.0
29        60  '2020/12/29'    100       132     280.0
30        60  '2020/12/30'    102       129     380.3
31        60  '2020/12/31'     92       115     243.0
```

Previously had NAN values.

# Replace Using Mean, Median, or Mode

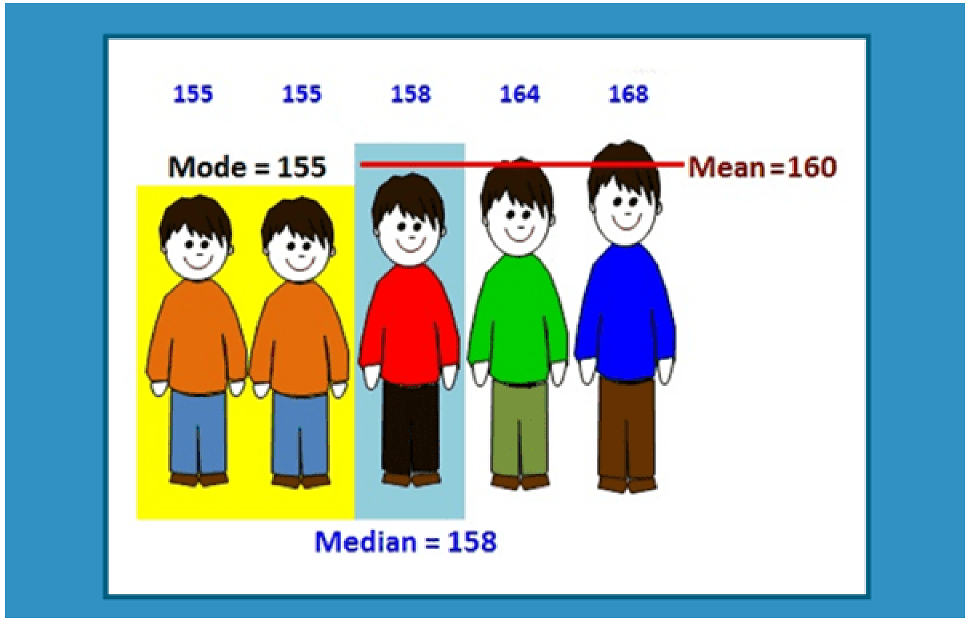
- A common way to replace empty cells, is to calculate the mean, median or mode value of the column.

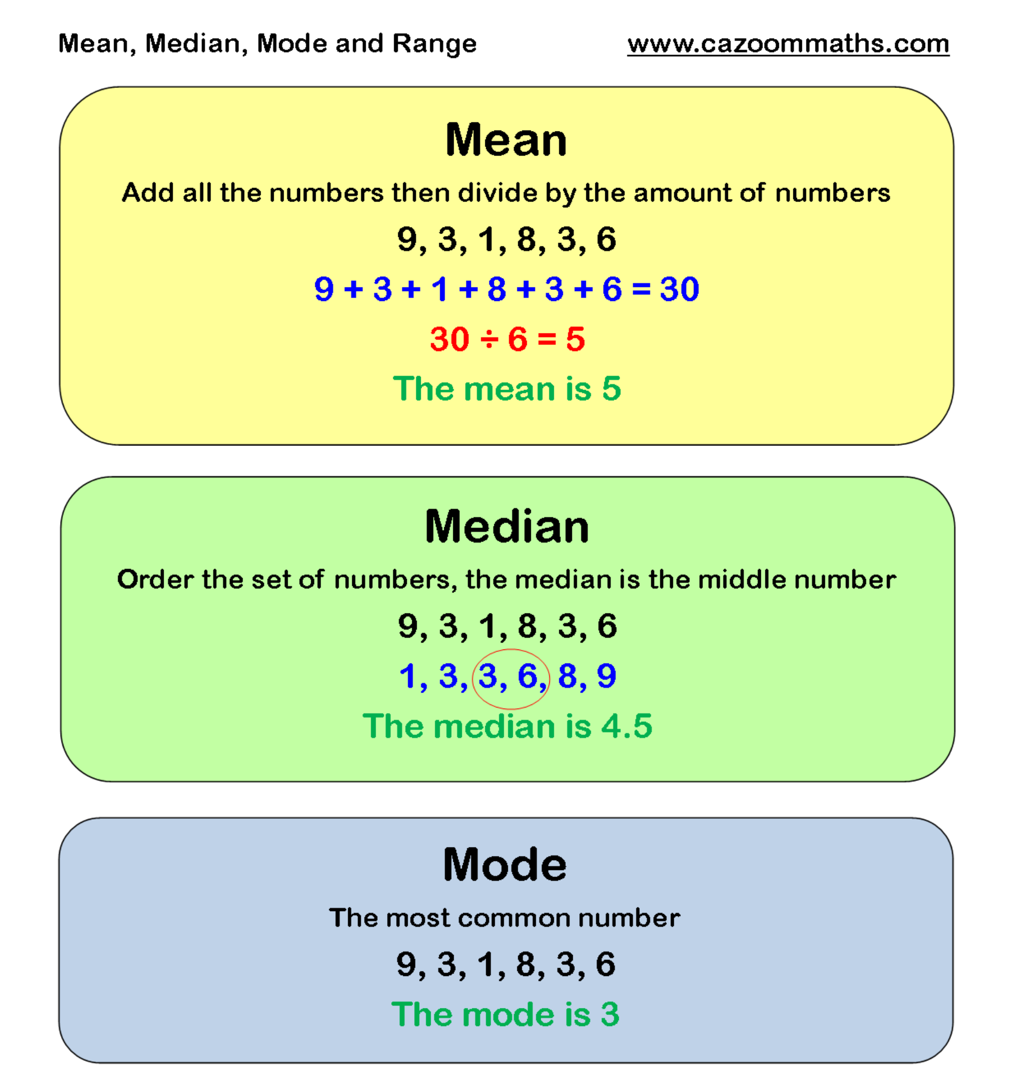
- Pandas uses the **mean() median()** and **mode()** methods to calculate the respective values for a specified column:

# Mean, Median, Mode REVIEW!

- The **mean** (average) of a data set is found by adding all numbers in the data set and then dividing by the number of values in the set.
- The **median** is the middle value when a data set is ordered from least to greatest.
- The **mode** is the number that occurs most often in a data set.
- A **Mean**, **Median**, and **Mode** video (3-minute video):

# Mean, Median, and Mode Cheat Sheet





Mean, Median, Mode and Range www.cazoommaths.com

## Mean

Add all the numbers then divide by the amount of numbers

9, 3, 1, 8, 3, 6

9 + 3 + 1 + 8 + 3 + 6 = 30

30 ÷ 6 = 5

The mean is 5

## Median

Order the set of numbers, the median is the middle number

9, 3, 1, 8, 3, 6

1, 3, 3, 6, 8, 9

The median is 4.5

## Mode

The most common number

9, 3, 1, 8, 3, 6

The mode is 3

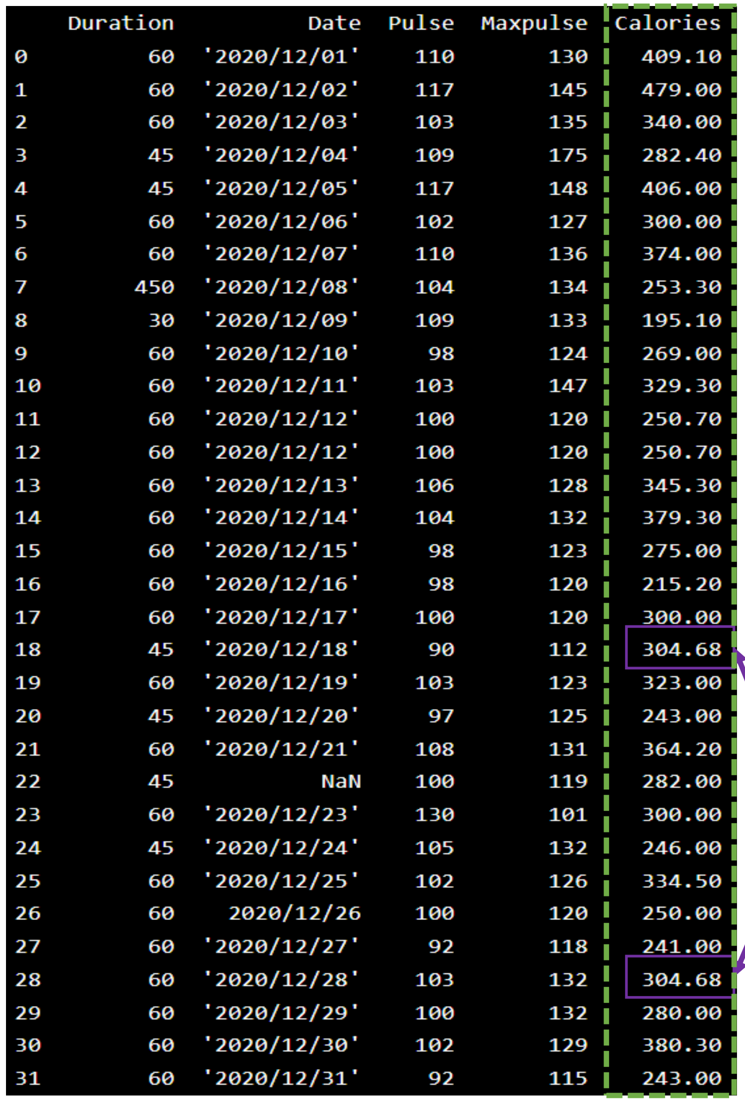
# Replace Using Mean

- Calculate the MEAN, and replace any empty values with it:

```
import pandas as pd

df = pd.read_csv('data.csv')

x = df["Calories"].mean() #captures the ‘Calories column in the dataframe ‘df’ and calculates the mean values of it. The mean values are then store in variable x.

df["Calories"].fillna(x, inplace = True) #The fillna() function replaces the empty values with the calculated mean from the ‘Calories’ column.
```

# Replace Using Mean Output

- With the **mean()** function, it calculated the mean of the entire **'Calories'** column, which is **304.68**.
- Having the calculated mean place in the **x** variable, using the **x**=**df["Calories"]**.fillna(**x**, inplace = True) rows **18** and **28**, the empty values from **"Calories"** was replaced with the mean: **304.68**
- **REMEMBER:**
  **Mean = the average value (the sum of all values divided by number of values).**



```
    Duration          Date  Pulse  Maxpulse  Calories
0         60  '2020/12/01'    110       130    409.10
1         60  '2020/12/02'    117       145    479.00
2         60  '2020/12/03'    103       135    340.00
3         45  '2020/12/04'    109       175    282.40
4         45  '2020/12/05'    117       148    406.00
5         60  '2020/12/06'    102       127    300.00
6         60  '2020/12/07'    110       136    374.00
7        450  '2020/12/08'    104       134    253.30
8         30  '2020/12/09'    109       133    195.10
9         60  '2020/12/10'     98       124    269.00
10        60  '2020/12/11'    103       147    329.30
11        60  '2020/12/12'    100       120    250.70
12        60  '2020/12/12'    100       120    250.70
13        60  '2020/12/13'    106       128    345.30
14        60  '2020/12/14'    104       132    379.30
15        60  '2020/12/15'     98       123    275.00
16        60  '2020/12/16'     98       120    215.20
17        60  '2020/12/17'    100       120    300.00
18        45  '2020/12/18'     90       112    304.68
19        60  '2020/12/19'    103       123    323.00
20        45  '2020/12/20'     97       125    243.00
21        60  '2020/12/21'    108       131    364.20
22        45           NaN    100       119    282.00
23        60  '2020/12/23'    130       101    300.00
24        45  '2020/12/24'    105       132    246.00
25        60  '2020/12/25'    102       126    334.50
26        60    2020/12/26    100       120    250.00
27        60  '2020/12/27'     92       118    241.00
28        60  '2020/12/28'    103       132    304.68
29        60  '2020/12/29'    100       132    280.00
30        60  '2020/12/30'    102       129    380.30
31        60  '2020/12/31'     92       115    243.00
```

Previously had NAN values.

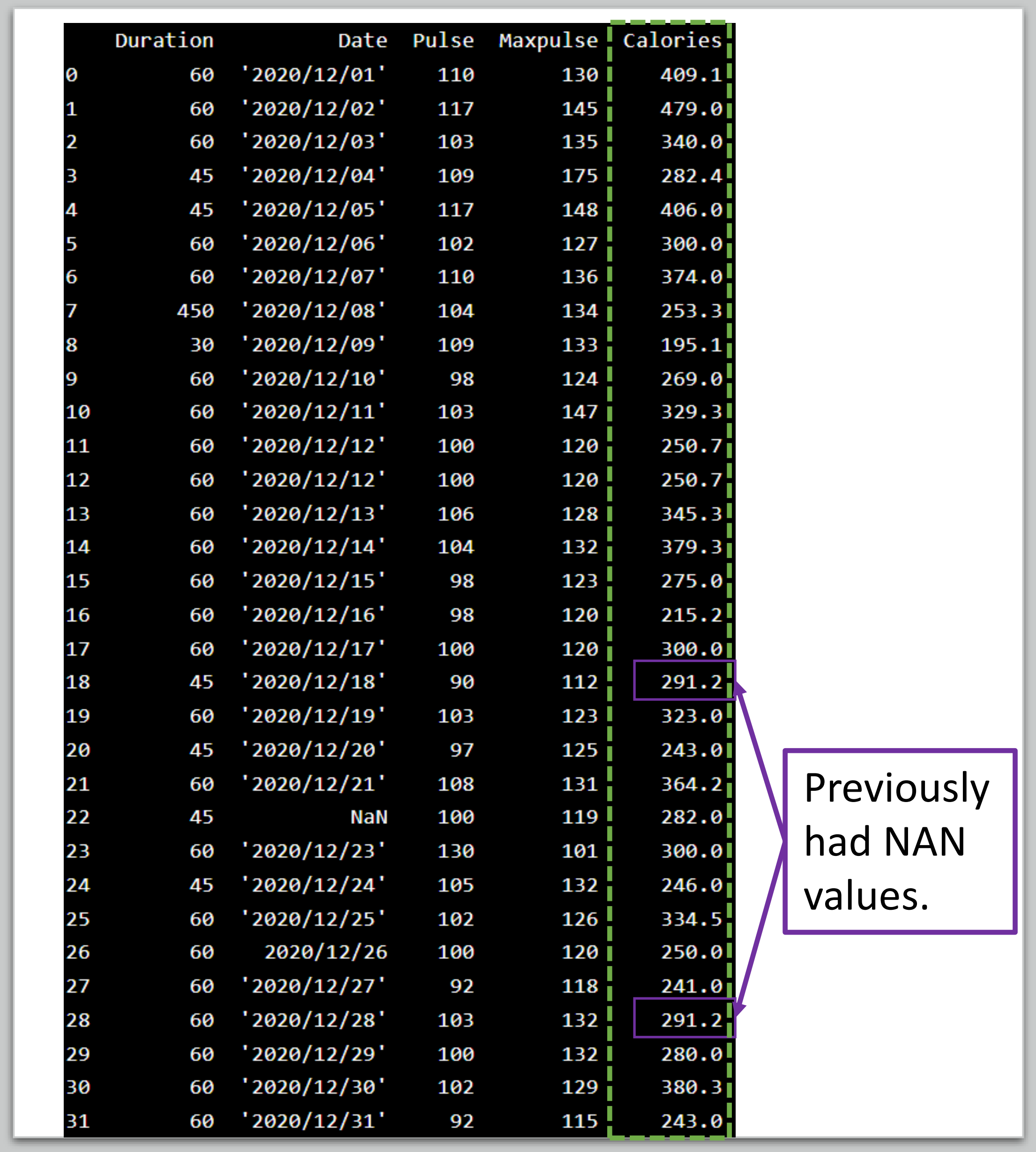
# Replace Using Median

- Calculate the MEDIAN, and replace any empty values with it:

```
import pandas as pd

df = pd.read_csv('data.csv')

x = df["Calories"].median() #the following does the same as the previous mean example, but this time it calculates and assigns the median value to the x variable.

df["Calories"].fillna(x, inplace = True)
```

# Replace Using Median Output

- With the **median()** function, it calculated the mean of the entire **'Calories'** column, which is **291.2**.
- Having the calculated mean place in the **x** variable, using the **x**=**df["Calories"]**.fillna(**x**, inplace = True) rows **18** and **28**, the empty values from **"Calories"** was replaced with the median: **291.2**
- **REMEMBER:**
  **Median = the value in the middle, after you have sorted all values ascending.**



```
    Duration          Date  Pulse  Maxpulse  Calories
0         60  '2020/12/01'    110       130     409.1
1         60  '2020/12/02'    117       145     479.0
2         60  '2020/12/03'    103       135     340.0
3         45  '2020/12/04'    109       175     282.4
4         45  '2020/12/05'    117       148     406.0
5         60  '2020/12/06'    102       127     300.0
6         60  '2020/12/07'    110       136     374.0
7        450  '2020/12/08'    104       134     253.3
8         30  '2020/12/09'    109       133     195.1
9         60  '2020/12/10'     98       124     269.0
10        60  '2020/12/11'    103       147     329.3
11        60  '2020/12/12'    100       120     250.7
12        60  '2020/12/12'    100       120     250.7
13        60  '2020/12/13'    106       128     345.3
14        60  '2020/12/14'    104       132     379.3
15        60  '2020/12/15'     98       123     275.0
16        60  '2020/12/16'     98       120     215.2
17        60  '2020/12/17'    100       120     300.0
18        45  '2020/12/18'     90       112     291.2
19        60  '2020/12/19'    103       123     323.0
20        45  '2020/12/20'     97       125     243.0
21        60  '2020/12/21'    108       131     364.2
22        45           NaN    100       119     282.0
23        60  '2020/12/23'    130       101     300.0
24        45  '2020/12/24'    105       132     246.0
25        60  '2020/12/25'    102       126     334.5
26        60    2020/12/26    100       120     250.0
27        60  '2020/12/27'     92       118     241.0
28        60  '2020/12/28'    103       132     291.2
29        60  '2020/12/29'    100       132     280.0
30        60  '2020/12/30'    102       129     380.3
31        60  '2020/12/31'     92       115     243.0
```

Previously had NAN values.

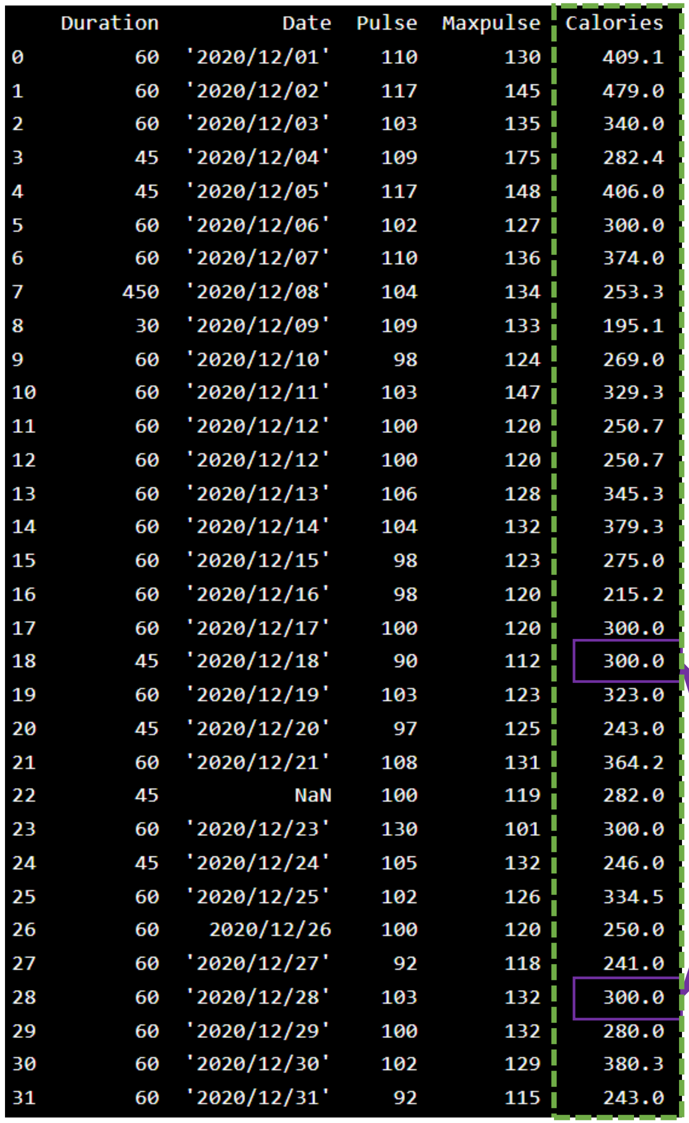
# Replace Using Mode

- Calculate the MODE, and replace any empty values with it:

```
import pandas as pd

df = pd.read_csv('data.csv')

x = df["Calories"].mode() #the following does
the same as the previous mean example, but this
time it calculates and assigns the mode value
to the x variable.

df["Calories"].fillna(x, inplace = True)
```

# Replace Using Mode Output

- With the **mode()** function, it calculated the mean of the entire **‘Calories’** column, which is **300.0**.
- Having the calculated mean place in the **x** variable, using the **x**=**df["Calories"]**.fillna(**x**, inplace = True) rows **18** and **28**, the empty values from **"Calories"** was replaced with the median: **300.0**
- **REMEMBER:**
  **Mode = the value that appears most frequently.**



```
    Duration          Date  Pulse  Maxpulse  Calories
0         60  '2020/12/01'    110       130     409.1
1         60  '2020/12/02'    117       145     479.0
2         60  '2020/12/03'    103       135     340.0
3         45  '2020/12/04'    109       175     282.4
4         45  '2020/12/05'    117       148     406.0
5         60  '2020/12/06'    102       127     300.0
6         60  '2020/12/07'    110       136     374.0
7        450  '2020/12/08'    104       134     253.3
8         30  '2020/12/09'    109       133     195.1
9         60  '2020/12/10'     98       124     269.0
10        60  '2020/12/11'    103       147     329.3
11        60  '2020/12/12'    100       120     250.7
12        60  '2020/12/12'    100       120     250.7
13        60  '2020/12/13'    106       128     345.3
14        60  '2020/12/14'    104       132     379.3
15        60  '2020/12/15'     98       123     275.0
16        60  '2020/12/16'     98       120     215.2
17        60  '2020/12/17'    100       120     300.0
18        45  '2020/12/18'     90       112     300.0
19        60  '2020/12/19'    103       123     323.0
20        45  '2020/12/20'     97       125     243.0
21        60  '2020/12/21'    108       131     364.2
22        45           NaN    100       119     282.0
23        60  '2020/12/23'    130       101     300.0
24        45  '2020/12/24'    105       132     246.0
25        60  '2020/12/25'    102       126     334.5
26        60    2020/12/26    100       120     250.0
27        60  '2020/12/27'     92       118     241.0
28        60  '2020/12/28'    103       132     300.0
29        60  '2020/12/29'    100       132     280.0
30        60  '2020/12/30'    102       129     380.3
31        60  '2020/12/31'     92       115     243.0
```

Previously had NAN values.

# When to use Mean, Median or Mode?

# Pandas - Cleaning Data of Wrong Format

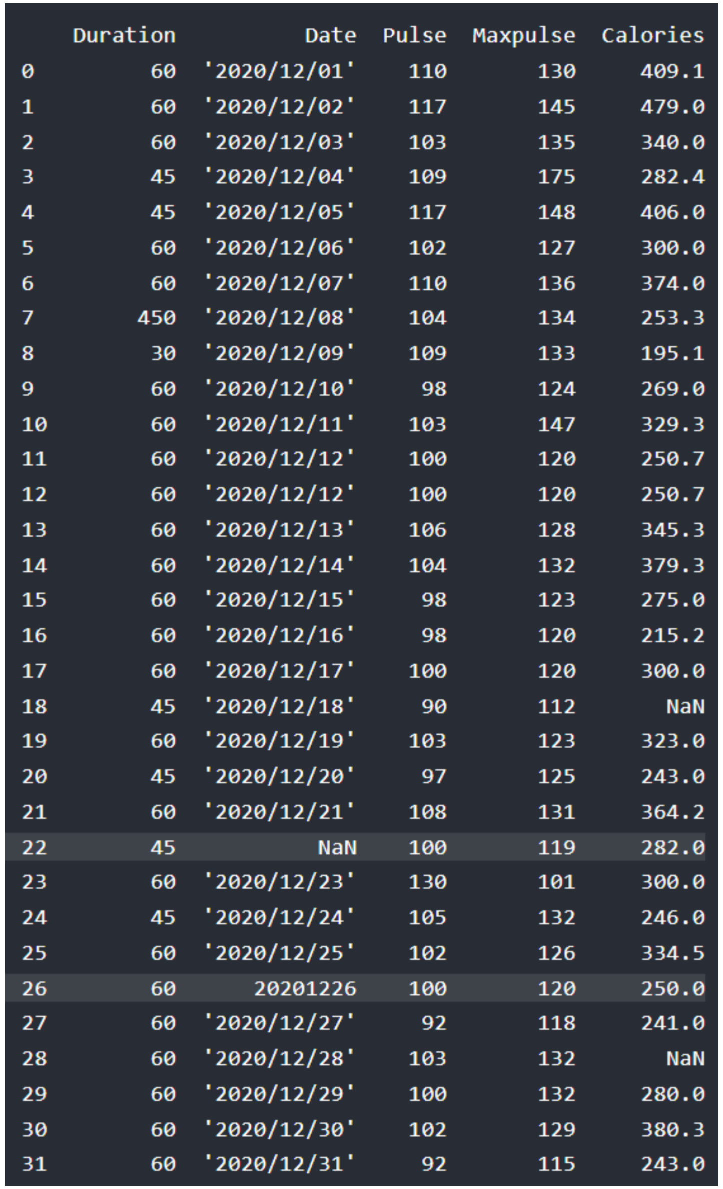
# Data of Wrong Format

- Cells with data of wrong format can make it difficult, or even impossible, to analyze data.

- To fix it, you have two options: remove the rows, or convert all cells in the columns into the same format.

# Convert Date Into a Correct Format

- In our Data Frame, we have two cells with the wrong format. Check out row 22 and 26, the 'Date' column should be a string that represents a date:
- Let's try to convert all cells in the 'Date' column into dates.
- Pandas has a **to_datetime()** method for this:



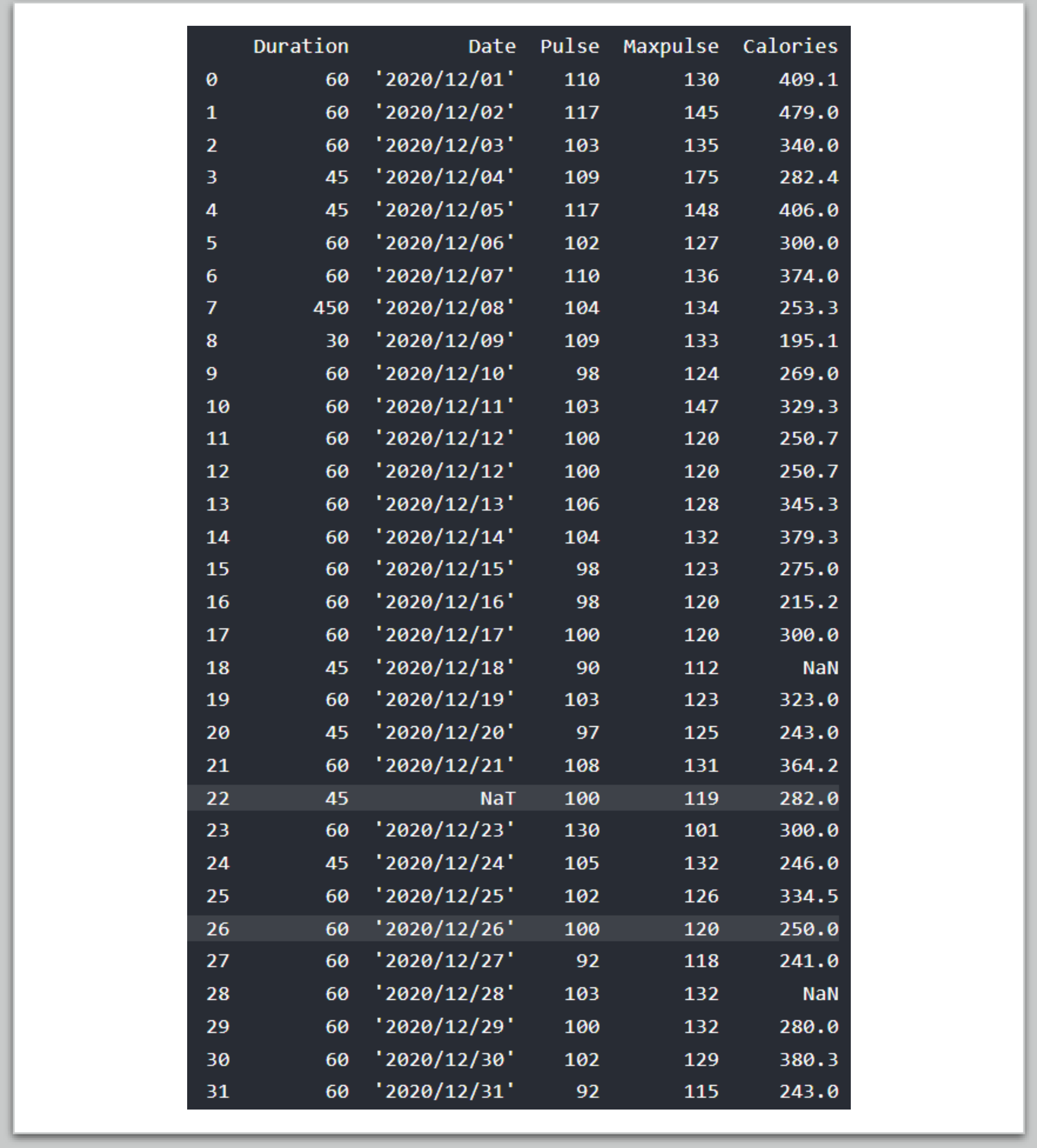
```
    Duration          Date  Pulse  Maxpulse  Calories
0         60  '2020/12/01'    110       130     409.1
1         60  '2020/12/02'    117       145     479.0
2         60  '2020/12/03'    103       135     340.0
3         45  '2020/12/04'    109       175     282.4
4         45  '2020/12/05'    117       148     406.0
5         60  '2020/12/06'    102       127     300.0
6         60  '2020/12/07'    110       136     374.0
7        450  '2020/12/08'    104       134     253.3
8         30  '2020/12/09'    109       133     195.1
9         60  '2020/12/10'     98       124     269.0
10        60  '2020/12/11'    103       147     329.3
11        60  '2020/12/12'    100       120     250.7
12        60  '2020/12/12'    100       120     250.7
13        60  '2020/12/13'    106       128     345.3
14        60  '2020/12/14'    104       132     379.3
15        60  '2020/12/15'     98       123     275.0
16        60  '2020/12/16'     98       120     215.2
17        60  '2020/12/17'    100       120     300.0
18        45  '2020/12/18'     90       112       NaN
19        60  '2020/12/19'    103       123     323.0
20        45  '2020/12/20'     97       125     243.0
21        60  '2020/12/21'    108       131     364.2
22        45           NaN    100       119     282.0
23        60  '2020/12/23'    130       101     300.0
24        45  '2020/12/24'    105       132     246.0
25        60  '2020/12/25'    102       126     334.5
26        60      20201226    100       120     250.0
27        60  '2020/12/27'     92       118     241.0
28        60  '2020/12/28'    103       132       NaN
29        60  '2020/12/29'    100       132     280.0
30        60  '2020/12/30'    102       129     380.3
31        60  '2020/12/31'     92       115     243.0
```

# Convert Date Into a Correct Format

```python
import pandas as pd

df = pd.read_csv('data.csv')

df['Date'] = pd.to_datetime(df['Date'])

print(df.to_string())
```

# Convert Date Into a Correct Format

- As you can see from the result, the date in row 26 was fixed, but the empty date in row 22 got a **NaT (Not a Time)** value, in other words an empty value.
- One way to deal with empty values is simply **removing the entire row**.



```
    Duration          Date  Pulse  Maxpulse  Calories
0         60  '2020/12/01'    110       130     409.1
1         60  '2020/12/02'    117       145     479.0
2         60  '2020/12/03'    103       135     340.0
3         45  '2020/12/04'    109       175     282.4
4         45  '2020/12/05'    117       148     406.0
5         60  '2020/12/06'    102       127     300.0
6         60  '2020/12/07'    110       136     374.0
7        450  '2020/12/08'    104       134     253.3
8         30  '2020/12/09'    109       133     195.1
9         60  '2020/12/10'     98       124     269.0
10        60  '2020/12/11'    103       147     329.3
11        60  '2020/12/12'    100       120     250.7
12        60  '2020/12/12'    100       120     250.7
13        60  '2020/12/13'    106       128     345.3
14        60  '2020/12/14'    104       132     379.3
15        60  '2020/12/15'     98       123     275.0
16        60  '2020/12/16'     98       120     215.2
17        60  '2020/12/17'    100       120     300.0
18        45  '2020/12/18'     90       112       NaN
19        60  '2020/12/19'    103       123     323.0
20        45  '2020/12/20'     97       125     243.0
21        60  '2020/12/21'    108       131     364.2
22        45           NaT    100       119     282.0
23        60  '2020/12/23'    130       101     300.0
24        45  '2020/12/24'    105       132     246.0
25        60  '2020/12/25'    102       126     334.5
26        60  '2020/12/26'    100       120     250.0
27        60  '2020/12/27'     92       118     241.0
28        60  '2020/12/28'    103       132       NaN
29        60  '2020/12/29'    100       132     280.0
30        60  '2020/12/30'    102       129     380.3
31        60  '2020/12/31'     92       115     243.0
```

# Removing Rows

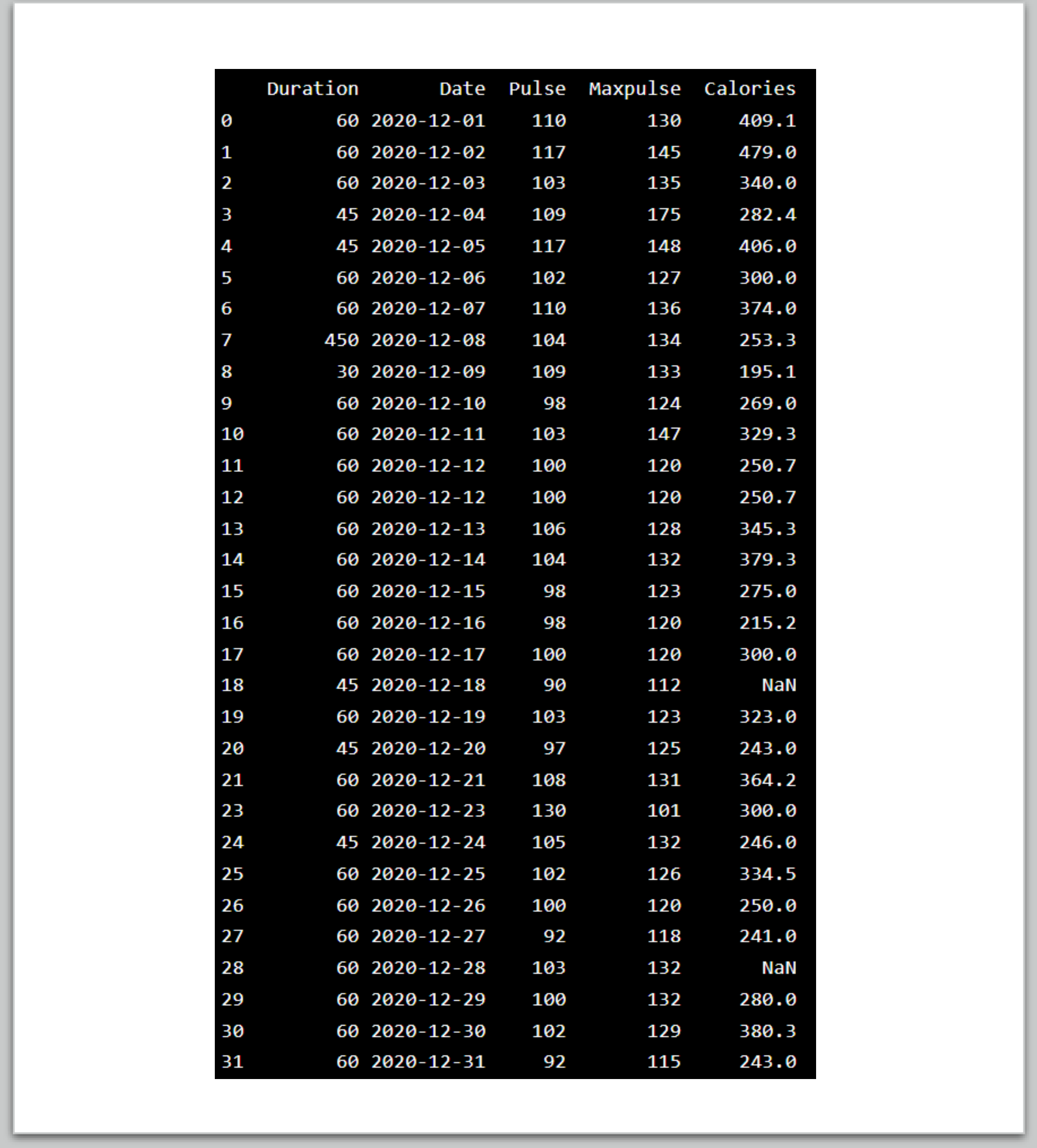
- The result from the converting in the example above gave us a **NaT** value, which can be handled as a NULL value, and we can remove the row by using the **dropna()** method.

```
df.dropna(subset=['Date'], inplace = True)
```

# Removing NAT in a specific Row

- Even though the incorrect formats were fixed by the df['Date'] = pd.to_datetime(df['Date’]) it left a NAT value.
- With the **dropna()** function, it removed the NAT value in the ‘Date’ column based on the given code below.

```
import pandas as pd

df = pd.read_csv('data.csv')

df['Date'] = pd.to_datetime(df['Date'])

df.dropna(subset=['Date'], inplace = True)

print(df.to_string())
```

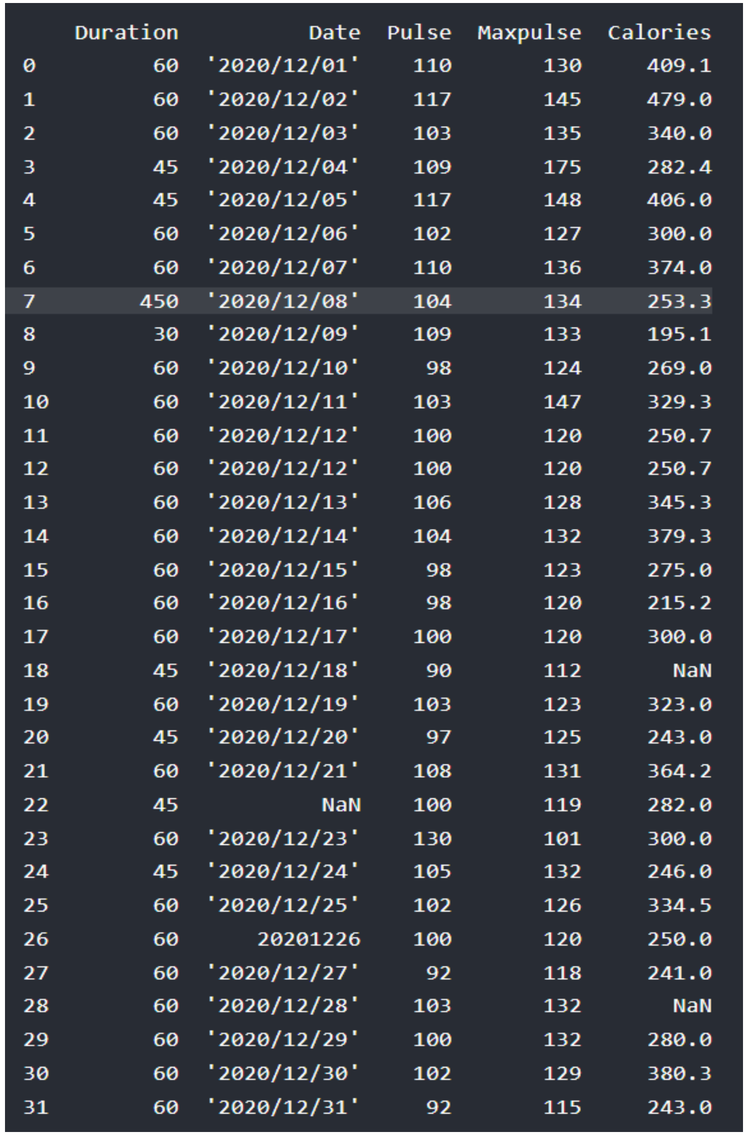


```
    Duration        Date  Pulse  Maxpulse  Calories
0         60  2020-12-01    110       130     409.1
1         60  2020-12-02    117       145     479.0
2         60  2020-12-03    103       135     340.0
3         45  2020-12-04    109       175     282.4
4         45  2020-12-05    117       148     406.0
5         60  2020-12-06    102       127     300.0
6         60  2020-12-07    110       136     374.0
7        450  2020-12-08    104       134     253.3
8         30  2020-12-09    109       133     195.1
9         60  2020-12-10     98       124     269.0
10        60  2020-12-11    103       147     329.3
11        60  2020-12-12    100       120     250.7
12        60  2020-12-12    100       120     250.7
13        60  2020-12-13    106       128     345.3
14        60  2020-12-14    104       132     379.3
15        60  2020-12-15     98       123     275.0
16        60  2020-12-16     98       120     215.2
17        60  2020-12-17    100       120     300.0
18        45  2020-12-18     90       112       NaN
19        60  2020-12-19    103       123     323.0
20        45  2020-12-20     97       125     243.0
21        60  2020-12-21    108       131     364.2
23        60  2020-12-23    130       101     300.0
24        45  2020-12-24    105       132     246.0
25        60  2020-12-25    102       126     334.5
26        60  2020-12-26    100       120     250.0
27        60  2020-12-27     92       118     241.0
28        60  2020-12-28    103       132       NaN
29        60  2020-12-29    100       132     280.0
30        60  2020-12-30    102       129     380.3
31        60  2020-12-31     92       115     243.0
```

# Pandas - Fixing Wrong Data

# Wrong Data

- "Wrong data" does not have to be "empty cells" or "wrong format", it can just be wrong, like if someone registered "199" instead of "1.99".
- Sometimes you can spot wrong data by looking at the data set, because you have an expectation of what it should be.
- If you review our data set, you can see that in row 7, the duration is 450, but for all the other rows the duration is between 30 and 60.
- It doesn't have to be wrong but taking in consideration that this is the data set of someone's workout sessions, we conclude with the fact that this person did not work out in 450 minutes.
- How can we fix wrong values, like the one for "Duration" in row 7?



```
    Duration          Date  Pulse  Maxpulse  Calories
0         60  '2020/12/01'    110       130     409.1
1         60  '2020/12/02'    117       145     479.0
2         60  '2020/12/03'    103       135     340.0
3         45  '2020/12/04'    109       175     282.4
4         45  '2020/12/05'    117       148     406.0
5         60  '2020/12/06'    102       127     300.0
6         60  '2020/12/07'    110       136     374.0
7        450  '2020/12/08'    104       134     253.3
8         30  '2020/12/09'    109       133     195.1
9         60  '2020/12/10'     98       124     269.0
10        60  '2020/12/11'    103       147     329.3
11        60  '2020/12/12'    100       120     250.7
12        60  '2020/12/12'    100       120     250.7
13        60  '2020/12/13'    106       128     345.3
14        60  '2020/12/14'    104       132     379.3
15        60  '2020/12/15'     98       123     275.0
16        60  '2020/12/16'     98       120     215.2
17        60  '2020/12/17'    100       120     300.0
18        45  '2020/12/18'     90       112       NaN
19        60  '2020/12/19'    103       123     323.0
20        45  '2020/12/20'     97       125     243.0
21        60  '2020/12/21'    108       131     364.2
22        45           NaN    100       119     282.0
23        60  '2020/12/23'    130       101     300.0
24        45  '2020/12/24'    105       132     246.0
25        60  '2020/12/25'    102       126     334.5
26        60      20201226    100       120     250.0
27        60  '2020/12/27'     92       118     241.0
28        60  '2020/12/28'    103       132       NaN
29        60  '2020/12/29'    100       132     280.0
30        60  '2020/12/30'    102       129     380.3
31        60  '2020/12/31'     92       115     243.0
```

# Replacing Values

- One way to fix wrong values is to replace them with something else.
- In our example, it is most likely a typo, and the value should be "45" instead of "450", and we could just insert "45" in row 7:
- Set "Duration" = 45 in row 7:

```
df.loc[7, 'Duration'] = 45
```

# Replacing Values

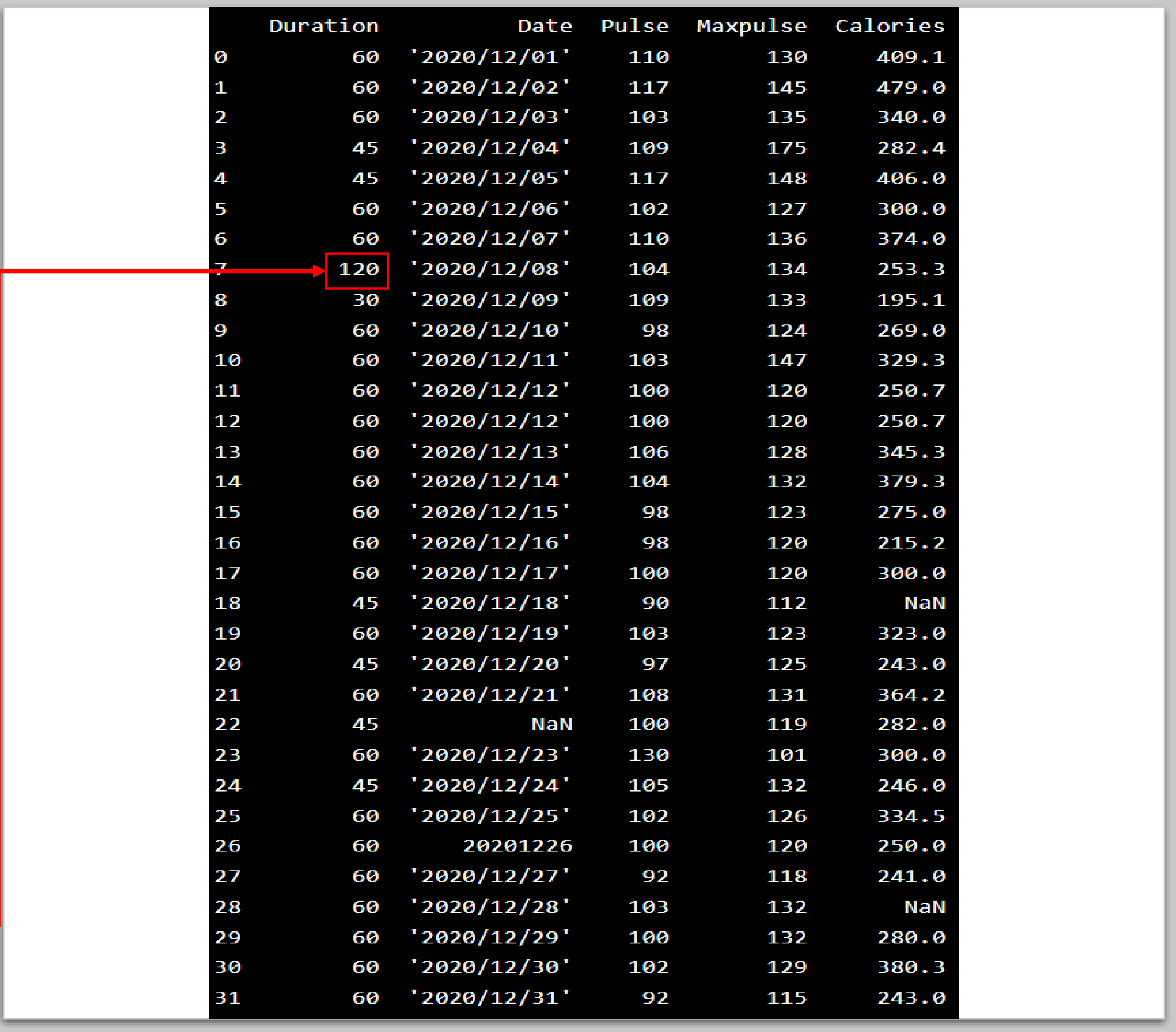
- For small data sets you might be able to replace the wrong data one by one, but not for big data sets.
- To replace wrong data for larger data sets you can create some rules, e.g., set some boundaries for legal values and replace any values that are outside of the boundaries.
- Loop through all values in the "Duration" column.
- If the value is higher than 120, set it to 120:

```
for x in df.index:
  if df.loc[x, "Duration"] > 120:
    df.loc[x, "Duration"] = 120
```

# Replacing Values Output

- To replace specific values quickly, you can refer to the code below.
- The code below shows that if the value in the ‘Duration’ column is >120 it will become 120. This also limits the value to 120.
- The code uses a for loop statement with if conditions to prevent manual lookups.

```
import pandas as pd


df = pd.read_csv('data.csv’)


for x in df.index:
   if df.loc[x, "Duration"] > 120:
      df.loc[x, "Duration"] = 120


print(df.to_string())
```



```
    Duration          Date  Pulse  Maxpulse  Calories
0         60  '2020/12/01'    110       130     409.1
1         60  '2020/12/02'    117       145     479.0
2         60  '2020/12/03'    103       135     340.0
3         45  '2020/12/04'    109       175     282.4
4         45  '2020/12/05'    117       148     406.0
5         60  '2020/12/06'    102       127     300.0
6         60  '2020/12/07'    110       136     374.0
7        120  '2020/12/08'    104       134     253.3
8         30  '2020/12/09'    109       133     195.1
9         60  '2020/12/10'     98       124     269.0
10        60  '2020/12/11'    103       147     329.3
11        60  '2020/12/12'    100       120     250.7
12        60  '2020/12/12'    100       120     250.7
13        60  '2020/12/13'    106       128     345.3
14        60  '2020/12/14'    104       132     379.3
15        60  '2020/12/15'     98       123     275.0
16        60  '2020/12/16'     98       120     215.2
17        60  '2020/12/17'    100       120     300.0
18        45  '2020/12/18'     90       112       NaN
19        60  '2020/12/19'    103       123     323.0
20        45  '2020/12/20'     97       125     243.0
21        60  '2020/12/21'    108       131     364.2
22        45           NaN    100       119     282.0
23        60  '2020/12/23'    130       101     300.0
24        45  '2020/12/24'    105       132     246.0
25        60  '2020/12/25'    102       126     334.5
26        60      20201226    100       120     250.0
27        60  '2020/12/27'     92       118     241.0
28        60  '2020/12/28'    103       132       NaN
29        60  '2020/12/29'    100       132     280.0
30        60  '2020/12/30'    102       129     380.3
31        60  '2020/12/31'     92       115     243.0
```

# Removing Rows with Wrong Data

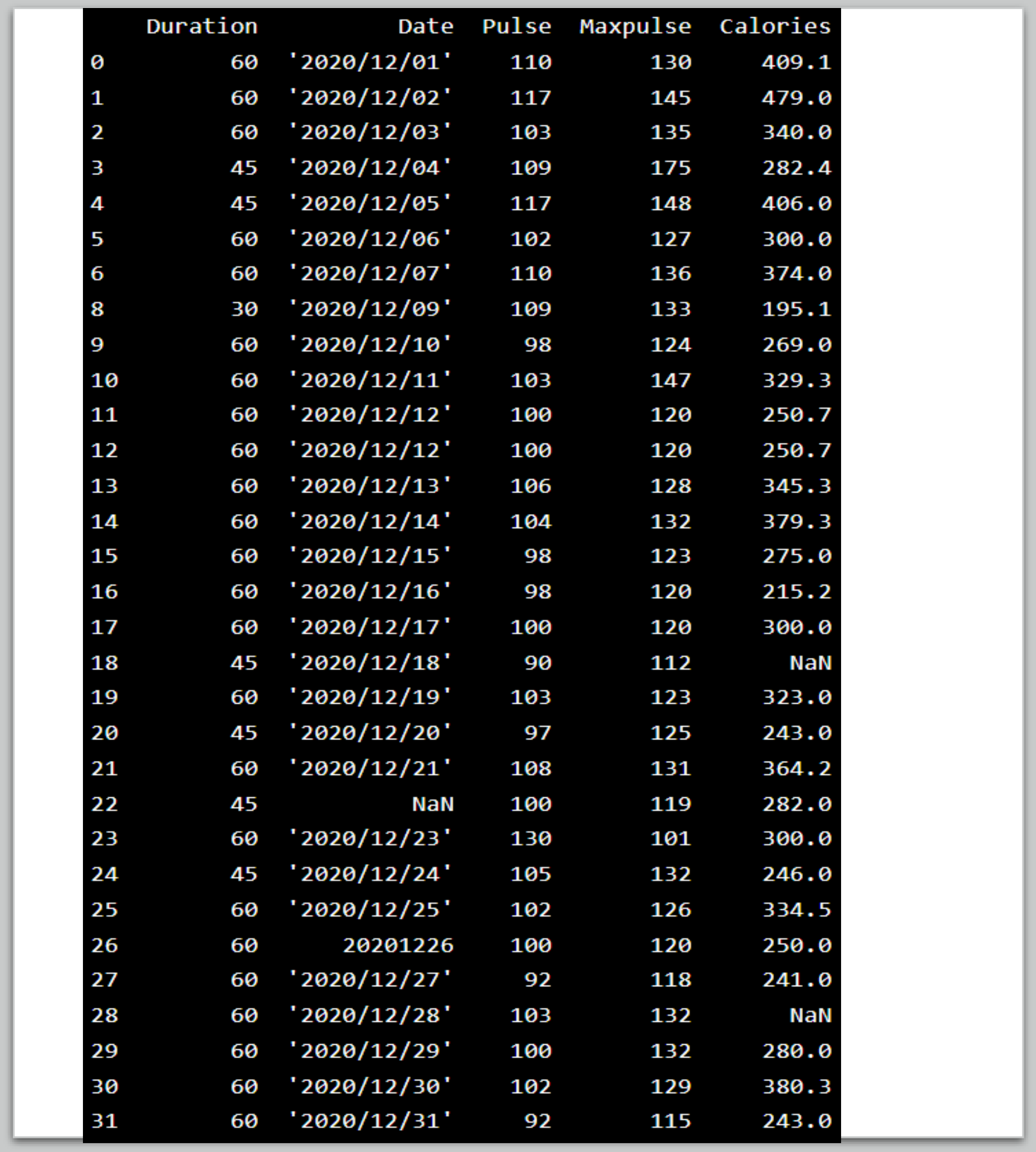
- Another way of handling wrong data is to remove the rows that contains wrong data.
- This way you do not have to find out what to replace them with, and there is a good chance you do not need them to do your analyses.
- Delete rows where "Duration" is higher than 120:

```
for x in df.index:
  if df.loc[x, "Duration"] > 120:
    df.drop(x, inplace = True)
```

# Removing Rows with Wrong Data Output

- To remove incorrect values quickly, you can refer to the code below.
- The code below shows that if the value in the ‘Duration’ column is incorrect or has a value >120 it will be removed.
- *Its LOGICALLY incorrect because the dataset might be considering a max of 120 only for the duration column, this is a case-to-case basis.*
- The code uses a for loop statement with if conditions to prevent manual lookups.

```
import pandas as pd


df = pd.read_csv('data.csv')


for x in df.index:
    if df.loc[x, "Duration"] > 120:
        df.drop(x, inplace = True)


#remember to include the 'inplace = True' argument to make the changes in the original DataFrame object instead of returning a copy


print(df.to_string())
```

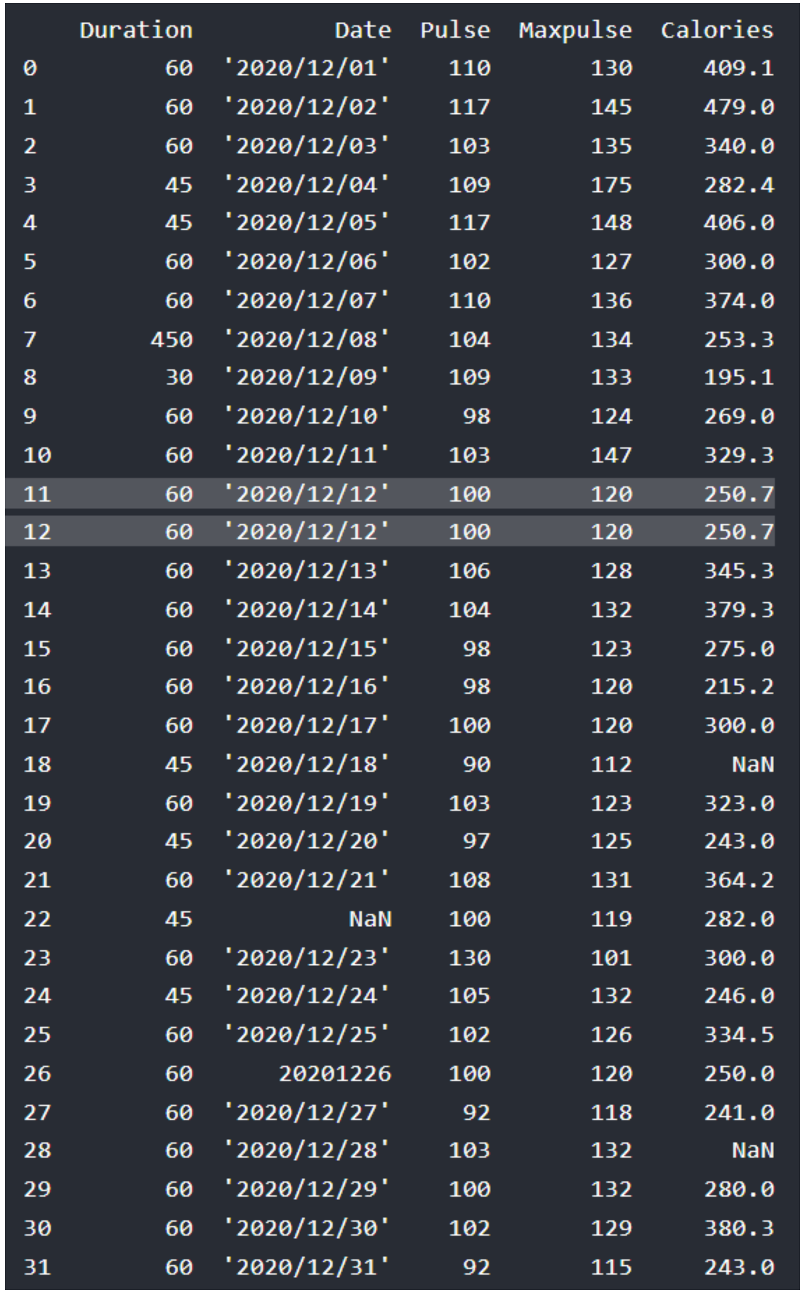


```
    Duration          Date  Pulse  Maxpulse  Calories
0         60  '2020/12/01'    110       130     409.1
1         60  '2020/12/02'    117       145     479.0
2         60  '2020/12/03'    103       135     340.0
3         45  '2020/12/04'    109       175     282.4
4         45  '2020/12/05'    117       148     406.0
5         60  '2020/12/06'    102       127     300.0
6         60  '2020/12/07'    110       136     374.0
8         30  '2020/12/09'    109       133     195.1
9         60  '2020/12/10'     98       124     269.0
10        60  '2020/12/11'    103       147     329.3
11        60  '2020/12/12'    100       120     250.7
12        60  '2020/12/12'    100       120     250.7
13        60  '2020/12/13'    106       128     345.3
14        60  '2020/12/14'    104       132     379.3
15        60  '2020/12/15'     98       123     275.0
16        60  '2020/12/16'     98       120     215.2
17        60  '2020/12/17'    100       120     300.0
18        45  '2020/12/18'     90       112       NaN
19        60  '2020/12/19'    103       123     323.0
20        45  '2020/12/20'     97       125     243.0
21        60  '2020/12/21'    108       131     364.2
22        45           NaN    100       119     282.0
23        60  '2020/12/23'    130       101     300.0
24        45  '2020/12/24'    105       132     246.0
25        60  '2020/12/25'    102       126     334.5
26        60      20201226    100       120     250.0
27        60  '2020/12/27'     92       118     241.0
28        60  '2020/12/28'    103       132       NaN
29        60  '2020/12/29'    100       132     280.0
30        60  '2020/12/30'    102       129     380.3
31        60  '2020/12/31'     92       115     243.0
```

# Pandas - Removing Duplicates

# Discovering Duplicates

- Duplicate rows are rows that have been registered more than one time.

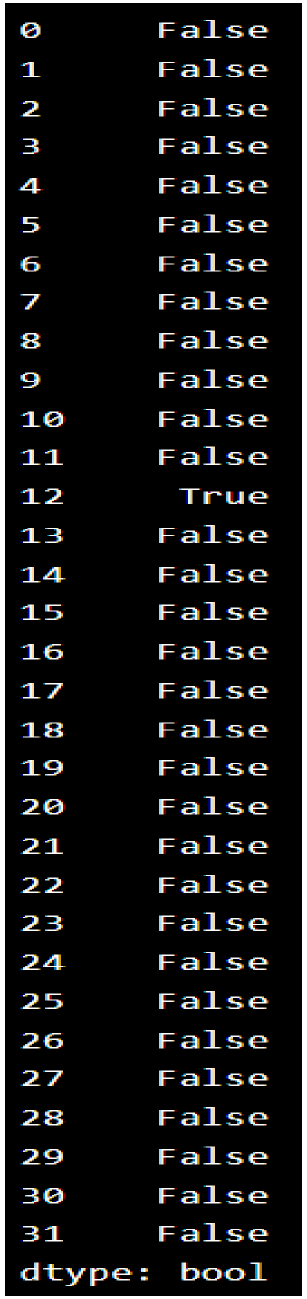


```
    Duration          Date  Pulse  Maxpulse  Calories
0         60  '2020/12/01'    110       130     409.1
1         60  '2020/12/02'    117       145     479.0
2         60  '2020/12/03'    103       135     340.0
3         45  '2020/12/04'    109       175     282.4
4         45  '2020/12/05'    117       148     406.0
5         60  '2020/12/06'    102       127     300.0
6         60  '2020/12/07'    110       136     374.0
7        450  '2020/12/08'    104       134     253.3
8         30  '2020/12/09'    109       133     195.1
9         60  '2020/12/10'     98       124     269.0
10        60  '2020/12/11'    103       147     329.3
11        60  '2020/12/12'    100       120     250.7
12        60  '2020/12/12'    100       120     250.7
13        60  '2020/12/13'    106       128     345.3
14        60  '2020/12/14'    104       132     379.3
15        60  '2020/12/15'     98       123     275.0
16        60  '2020/12/16'     98       120     215.2
17        60  '2020/12/17'    100       120     300.0
18        45  '2020/12/18'     90       112       NaN
19        60  '2020/12/19'    103       123     323.0
20        45  '2020/12/20'     97       125     243.0
21        60  '2020/12/21'    108       131     364.2
22        45           NaN    100       119     282.0
23        60  '2020/12/23'    130       101     300.0
24        45  '2020/12/24'    105       132     246.0
25        60  '2020/12/25'    102       126     334.5
26        60      20201226    100       120     250.0
27        60  '2020/12/27'     92       118     241.0
28        60  '2020/12/28'    103       132       NaN
29        60  '2020/12/29'    100       132     280.0
30        60  '2020/12/30'    102       129     380.3
31        60  '2020/12/31'     92       115     243.0
```

# Discovering Duplicates

- By looking at our test data set, we can assume that row 11 and 12 are duplicates.
- To discover duplicates, we can use the **duplicated()** method.
- The **duplicated()** method returns a Boolean values for each row:
- Returns True for every row that is a duplicate, othwerwise False:

```
print(df.duplicated())
```

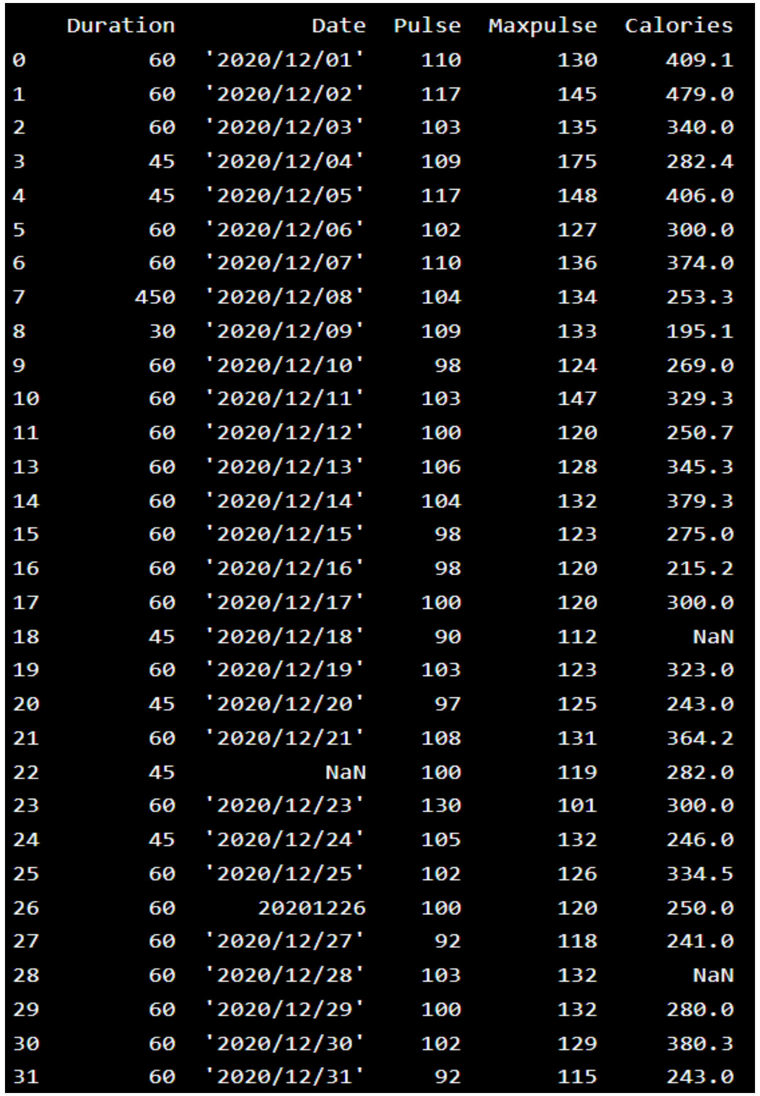


```
0       False
1       False
2       False
3       False
4       False
5       False
6       False
7       False
8       False
9       False
10      False
11      False
12       True
13      False
14      False
15      False
16      False
17      False
18      False
19      False
20      False
21      False
22      False
23      False
24      False
25      False
26      False
27      False
28      False
29      False
30      False
31      False
dtype: bool
```

# Removing Duplicates

- To remove duplicates, use the **drop_duplicates()** method.
- Row 12 is dropped for being a duplicate of Row 11
- ***REMINDER: The (inplace = True) will make sure that the method does NOT return a new DataFrame, but it will remove all duplicates from the original DataFrame.***

```
df.drop_duplicates(inplace = True)
```



```
    Duration          Date  Pulse  Maxpulse  Calories
0         60  '2020/12/01'    110       130     409.1
1         60  '2020/12/02'    117       145     479.0
2         60  '2020/12/03'    103       135     340.0
3         45  '2020/12/04'    109       175     282.4
4         45  '2020/12/05'    117       148     406.0
5         60  '2020/12/06'    102       127     300.0
6         60  '2020/12/07'    110       136     374.0
7        450  '2020/12/08'    104       134     253.3
8         30  '2020/12/09'    109       133     195.1
9         60  '2020/12/10'     98       124     269.0
10        60  '2020/12/11'    103       147     329.3
11        60  '2020/12/12'    100       120     250.7
13        60  '2020/12/13'    106       128     345.3
14        60  '2020/12/14'    104       132     379.3
15        60  '2020/12/15'     98       123     275.0
16        60  '2020/12/16'     98       120     215.2
17        60  '2020/12/17'    100       120     300.0
18        45  '2020/12/18'     90       112       NaN
19        60  '2020/12/19'    103       123     323.0
20        45  '2020/12/20'     97       125     243.0
21        60  '2020/12/21'    108       131     364.2
22        45           NaN    100       119     282.0
23        60  '2020/12/23'    130       101     300.0
24        45  '2020/12/24'    105       132     246.0
25        60  '2020/12/25'    102       126     334.5
26        60      20201226    100       120     250.0
27        60  '2020/12/27'     92       118     241.0
28        60  '2020/12/28'    103       132       NaN
29        60  '2020/12/29'    100       132     280.0
30        60  '2020/12/30'    102       129     380.3
31        60  '2020/12/31'     92       115     243.0
```

# EXERCISE 4

# E4-Q1

- Insert the correct syntax for removing rows with empty cells.

df.______()

# E4-Q2

- Insert the correct syntax for replacing empty cells with the value "130".

```
df.______(130)
```

# E4-Q3

- Insert the correct argument to make sure that the changes are done for the original DataFrame instead of returning a new one.

dropna(______________)

# E4-Q4

- Insert the correct syntax for removing duplicates in a DataFrame.

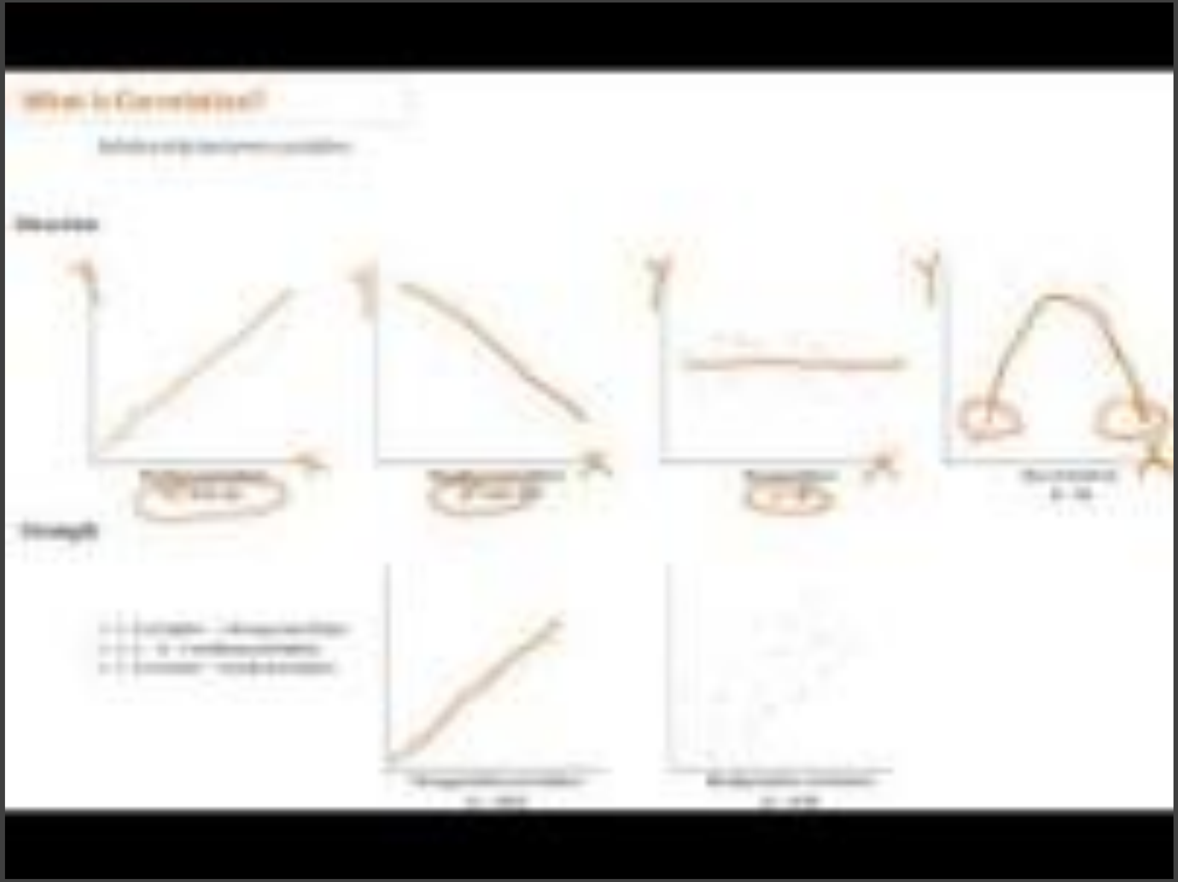
df.________________()

# Pandas - Data Correlations

# Finding Relationships

- A great aspect of the Pandas module is the **corr()** method.
- The **corr()** method calculates the relationship between each column in your data set.
- The following will use a CSV file named ‘data.csv’
- Download it from here → https://www.w3schools.com/python/pandas/data.csv

# Correlations



What is Correlation?

Direction

Strength

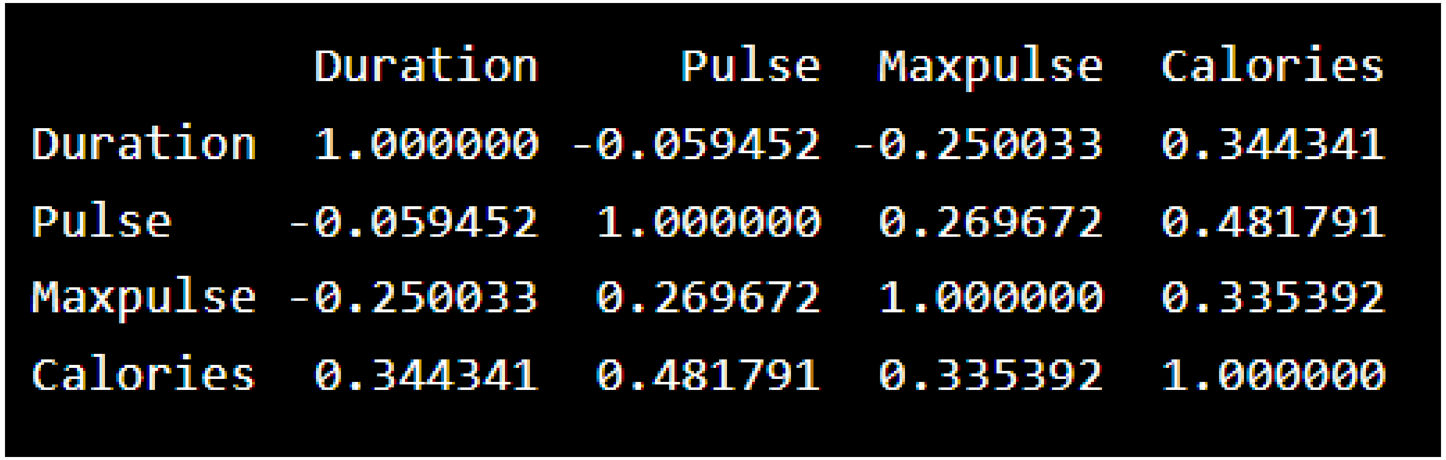
- A statistical measure that expresses the extent to which two variables are linearly related (meaning they change together at a constant rate).

# Finding Relationships

- Show the relationship between the columns:

*Note: The **corr()** method ignores "not numeric" columns.*

```
df.corr()
```

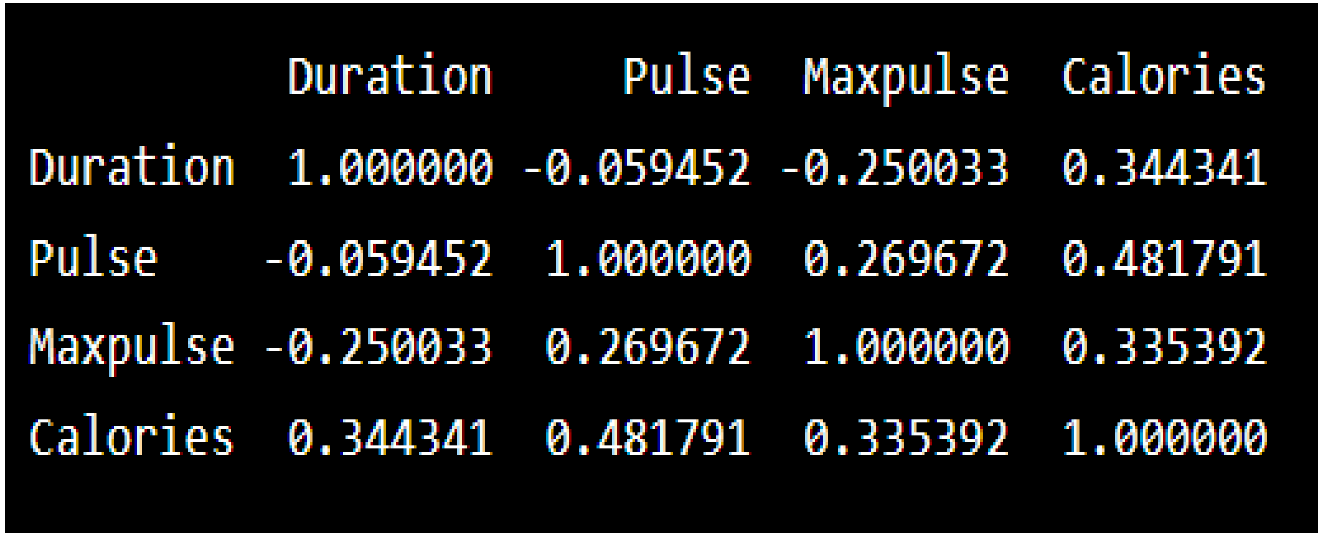


```
          Duration     Pulse  Maxpulse  Calories
Duration  1.000000 -0.059452 -0.250033  0.344341
Pulse    -0.059452  1.000000  0.269672  0.481791
Maxpulse -0.250033  0.269672  1.000000  0.335392
Calories  0.344341  0.481791  0.335392  1.000000
```

# Result Explained

- The Result of the **corr()** method is a table with a lot of numbers that represents how well the relationship is between two columns.
- The number varies from -1 to 1.
- 1 means that there is a 1 to 1 relationship (a perfect correlation), and for this data set, each time a value went up in the first column, the other one went up as well.
- 0.9 is also a good relationship, and if you increase one value, the other will probably increase as well.
- **-0.9 would be just as good relationship as 0.9**, but if you increase one value, the other will probably go down.
- **0.2** means **NOT a good relationship**, meaning that if one value goes up does not mean that the other will.

***NOTE: A good correlation depends on the use, but I think it is safe to say you must have at least 0.6 (or -0.6) to call it a good correlation. In some, they consider at least a 0.7.***



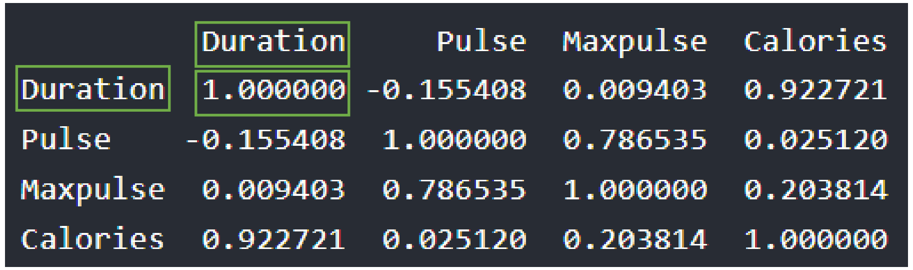
```
          Duration     Pulse  Maxpulse  Calories
Duration  1.000000 -0.059452 -0.250033  0.344341
Pulse    -0.059452  1.000000  0.269672  0.481791
Maxpulse -0.250033  0.269672  1.000000  0.335392
Calories  0.344341  0.481791  0.335392  1.000000
```

# Types of Correlations

**Perfect Correlation:**

- We can see that "Duration" and "Duration" got the number 1.000000, which makes sense, each column always has a perfect relationship with itself.

***NOTE: ALL WILL HAVE A 1.0 CORELLATION WITH ITSELF!***

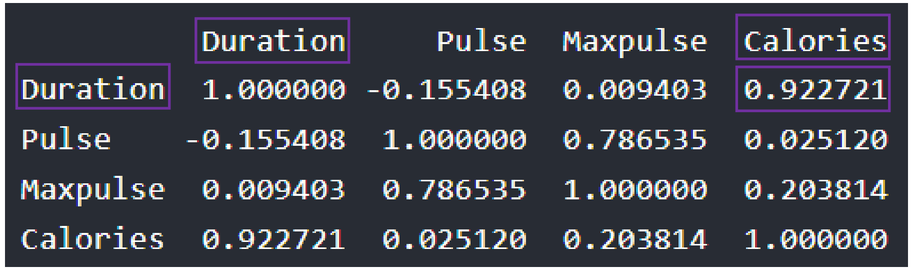


```
          Duration     Pulse  Maxpulse  Calories
Duration  1.000000 -0.155408  0.009403  0.922721
Pulse    -0.155408  1.000000  0.786535  0.025120
Maxpulse  0.009403  0.786535  1.000000  0.203814
Calories  0.922721  0.025120  0.203814  1.000000
```

**Good Correlation:**

- "Duration" and "Calories" got a 0.922721 correlation, which is a very good correlation

**ANALYSIS:** We can manually predict that the longer you work out directly, the more calories you burn, and the other way around: **if you burned a lot of calories, you probably had a long work out.**

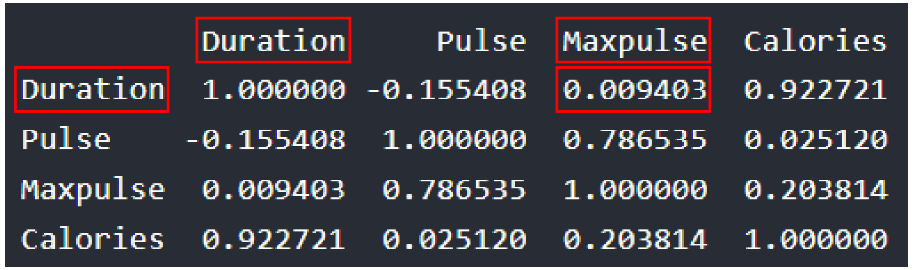


```
          Duration     Pulse  Maxpulse  Calories
Duration  1.000000 -0.155408  0.009403  0.922721
Pulse    -0.155408  1.000000  0.786535  0.025120
Maxpulse  0.009403  0.786535  1.000000  0.203814
Calories  0.922721  0.025120  0.203814  1.000000
```

**Bad Correlation:**

- "Duration" and "Maxpulse" got a 0.009403 correlation, which is a very bad correlation.

**ANALYSIS:** We can not manually and directly predict the max pulse by just looking at the duration of the work out, and vice versa.



```
          Duration     Pulse  Maxpulse  Calories
Duration  1.000000 -0.155408  0.009403  0.922721
Pulse    -0.155408  1.000000  0.786535  0.025120
Maxpulse  0.009403  0.786535  1.000000  0.203814
Calories  0.922721  0.025120  0.203814  1.000000
```

# EXERCISE 5

# E5-Q1

- Insert a correct syntax for finding relationships between columns in a DataFrame.

df.______

# E5-Q2

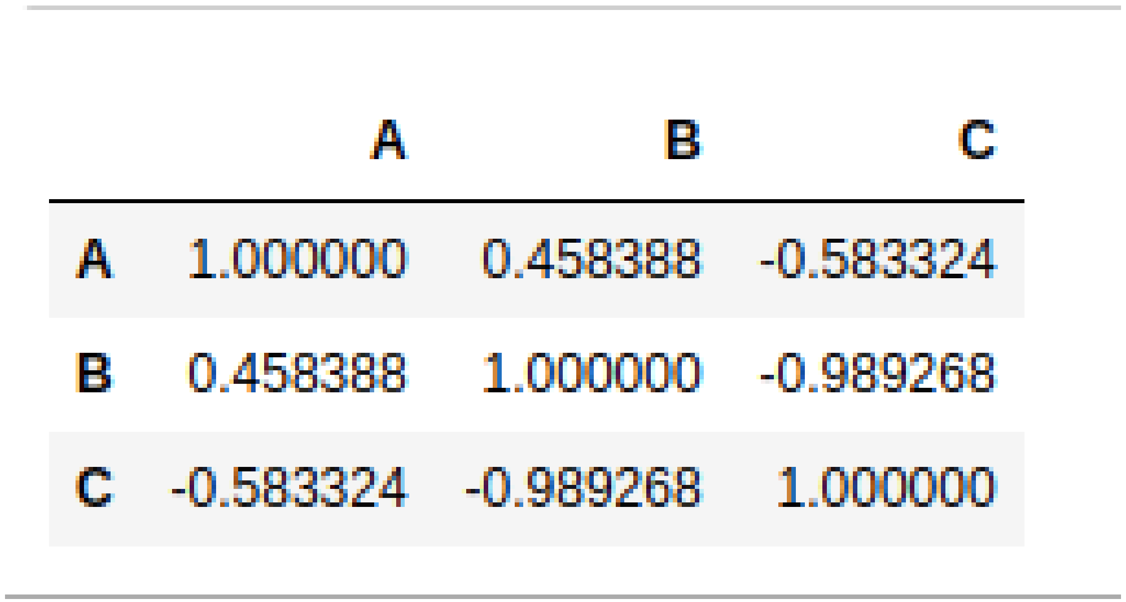
- True or false: A correlation of 0.9 is considered a good correlation.

# E5-Q3

- True or false: A correlation of -0.9 is considered a good correlation.

# E5-Q4

- Provide an analysis for the given correlation matrix.



|   | A | B | C |
|---|---|---|---|
| **A** | 1.000000 | 0.458388 | -0.583324 |
| **B** | 0.458388 | 1.000000 | -0.989268 |
| **C** | -0.583324 | -0.989268 | 1.000000 |

# Pandas 101 Tutorial Video

Complete Python Pandas Data Science Tutorial! (Reading CSV/Excel files, Sorting, Filtering, Groupby)

https://www.youtube.com/watch?v=vmEHCJofslg

# Sample Notebooks

**GOOD STARTER**

https://www.kaggle.com/code/kralmachine/pandas-tutorial-for-beginners

**MORE SAMPLES WITH DETAILS**

https://www.kaggle.com/code/sohier/tutorial-accessing-data-with-pandas

**ADVANCE YOUR KNOWLEDGE**

https://www.kaggle.com/code/rohan1506/pandas-tips-tricks-tutorial