pandas

Pandas is a Python library.

Pandas is used to analyze data.

Pandas Introduction

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.

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

New Section

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

```
import pandas

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

myvar = pandas.DataFrame(mydataset)

print(myvar)
```

```
    cars  passings
0    BMW         3
1  Volvo         7
2   Ford         2
```

Pandas Series

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.

```
import pandas as pd

a = [1, 7, 2]

myvar = pd.Series(a)

print(myvar)
```

```
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.

```
print(myvar[0])
```

```
380
```

Create Labels

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

```
import pandas as pd

a = [1, 7, 2]

myvar = pd.Series(a, index = ["x", "y", "z"])

print(myvar)
print(myvar["y"])
```

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

Key/Value Objects as Series

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

```
import pandas as pd

calories = {"day1": 420, "day2": 380, "day3": 390}

myvar = pd.Series(calories)

print(myvar)
```

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

DataFrames

Data sets in Pandas are usually multi-dimensional tables, called DataFrames.

Series is like a column, a DataFrame is the whole table.

```
import pandas as pd
```

```
data = {
  "calories": [420, 380, 390],
  "duration": [50, 40, 45]
}

myvar = pd.DataFrame(data)

print(myvar)
```

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

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.

```
import pandas as pd

df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/data.c

print(df.to_string())
```

```
     108        90     90       120     300.3
     109       210    137       184    1860.4
     110        60    102       124     325.2
     111        45    107       124     275.0
     112        15    124       139     124.2
     113        45    100       120     225.3
     114        60    108       131     367.6
     115        60    108       151     351.7
     116        60    116       141     443.0
     117        60     97       122     277.4
     118        60    105       125       NaN
     119        60    103       124     332.7
     120        30    112       137     193.9
     121        45    100       120     100.7
     122        60    119       169     336.7
     123        60    107       127     344.9
     124        60    111       151     368.5
     125        60     98       122     271.0
     126        60     97       124     275.3
     127        60    109       127     382.0
     128        90     99       125     466.4
     129        60    114       151     384.0
     130        60    104       134     342.5
     131        60    107       138     357.5
     132        60    103       133     335.0
     133        60    106       132     327.5
```

```
134        60    103       136     339.0
135        20    136       156     189.0
136        45    117       143     317.7
137        45    115       137     318.0
138        45    113       138     308.0
139        20    141       162     222.4
140        60    108       135     390.0
141        60     97       127       NaN
142        45    100       120     250.4
143        45    122       149     335.4
144        60    136       170     470.2
145        45    106       126     270.8
146        60    107       136     400.0
147        60    112       146     361.9
148        30    103       127     185.0
149        60    110       150     409.4
150        60    106       134     343.0
151        60    109       129     353.2
152        60    109       138     374.0
153        30    150       167     275.8
154        60    105       128     328.0
155        60    111       151     368.5
156        60     97       131     270.4
157        60    100       120     270.4
158        60    114       150     382.8
159        30     80       120     240.9
160        30     85       120     250.4
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     300.0
166        60    115       145     310.2
167        75    120       150     320.4
```

i

```
import pandas as pd

df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/data.c
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     300.0
166        60    115       145     310.2
167        75    120       150     320.4
168        75    125       150     330.4

[169 rows x 4 columns]
```

Pandas - Analyzing DataFrames

Viewing the Data 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.

```
import pandas as pd

df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/data.c


print(df.head(10))
```

```
   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     300.0
6        60    110       136     374.0
7        45    104       134     253.3
8        30    109       133     195.1
9        60     98       124     269.0
```

if the number of rows is not specified, the head() method will return the top 5 rows.

```
import pandas as pd

df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/data.c


print(df.head())
```

```
   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
```

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.

Example Print the last 5 rows of the DataFrame:

```
print(df.tail())
```

```
     Duration  Pulse  Maxpulse  Calories
164        60    105       140     290.8
165        60    110       145     300.0
166        60    115       145     310.2
167        75    120       150     320.4
168        75    125       150     330.4
```

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

Example

Print information about the data:

```
print(df.info())
```

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

Empty Cells

Empty cells can potentially give you a wrong result when you analyze data.

Remove Rows

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

```
import pandas as pd

df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/data.c

new_df = df.dropna()
```

```
print(new_df.to_string())
```

```
4          45    117       148     406.0
5          60    102       127     300.0
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
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
28         60    100       132     280.0
29         60    102       129     380.3
30         60     92       115     243.0
31         45     90       112     180.1
32         60    101       124     299.0
33         60     93       113     223.0
34         60    107       136     361.0
35         60    114       140     415.0
36         60    102       127     300.0
37         60    100       120     300.0
38         60    100       120     300.0
39         45    104       129     266.0
40         45     90       112     180.1
41         60     98       126     286.0
42         60    100       122     329.4
43         60    111       138     400.0
44         60    111       131     397.0
45         60     99       119     273.0
46         60    109       153     387.6
47         45    111       136     300.0
48         45    108       129     298.0
49         60    111       139     397.6
50         60    107       136     380.2
51         80    123       146     643.1
52         60    106       130     263.0
53         60    118       151     486.0
54         30    136       175     238.0
55         60    121       146     450.7
56         60    118       121     413.0
57         45    115       144     305.0
58         20    153       172     226.4
59         45    123       152     321.0
60        210    108       160    1376.0
61        160    110       137    1034.4
62        160    109       135     853.0
63         45    118       141     341.0
64         20    110       130     131.4
```

```
65        180     90        130     800.4
```

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:

```
import pandas as pd

df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/data.c

df.fillna(130, inplace = True)
print(df.to_string())
```

Double-click (or enter) to edit

Replace Only For a 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:

```
import pandas as pd

df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/data.c

df['Calories'].fillna(130, inplace = True)
print(df.to_string())
```

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 = the average value (the sum of all values divided by number of values).

Median = the value in the middle, after you have sorted all values ascending.

Mode = the value that appears most frequently.

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

```
import pandas as pd

df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/data.c

x = df["Calories"].mean()

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

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

```
import pandas as pd

df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/data.c

x = df["Calories"].median()

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

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 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:

[ ] ↳ *4 cells hidden*

Pandas - Fixing Wrong Data

If you take a look at 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.

```
import pandas as pd
```

```
df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/dirtyd
```

```
print(df.to_string())
```

```
import pandas as pd

df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/dirtyd

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

print(df.to_string())
```

```
import pandas as pd

df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/dirtyd

for x in df.index:
  if df.loc[x, "Duration"] > 120:
    df.loc[x, "Duration"] = 45

print(df.to_string())
```

Pandas - Removing Duplicates

To discover duplicates, we can use the duplicated() method.

Returns True for every row that is a duplicate, othwerwise False:

```
import pandas as pd

df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/dirtyd
```

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

To remove duplicates, use the drop_duplicates() method.

```
import pandas as pd

df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/dirtyd

df.drop_duplicates(inplace = True)

print(df.to_string())
```

#Notice that row 12 has been removed from the result

Double-click (or enter) to edit

looking at Columns, rows and cell

```
import pandas as pd

df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/dirtyd

print(df.head())
```

```
   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
```

Get the first row

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

```
Duration             60
Date        '2020/12/01'
Pulse                110
Maxpulse             130
Calories           409.1
Name: 0, dtype: object
```

Get the 20 th row

```
print(df.loc[19,'Duration'])
```

```
60
```

```
print(df.loc[19])
```

```
Duration             60
Date        '2020/12/19'
Pulse                103
Maxpulse             123
Calories             323
Name: 19, dtype: object
```

get the last row

```
print(df.tail())
```

```
    Duration          Date  Pulse  Maxpulse  Calories
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
```

```
print(df.tail(n=1))
print(df.tail(n=3))
```

```
    Duration          Date  Pulse  Maxpulse  Calories
31        60  '2020/12/31'     92       115     243.0
    Duration          Date  Pulse  Maxpulse  Calories
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
```

loc is label-based, which means that you have to specify rows and columns based on their row and column labels.

iloc is integer position-based, so you have to specify rows and columns by their integer position values (0-based integer position).

```
print(df.iloc[1])
```

```
Duration             60
Date        '2020/12/02'
Pulse                117
Maxpulse             145
```

```
Calories            479
Name: 1, dtype: object
```

```
print(df.iloc[-1])
```

```
Duration             60
Date        '2020/12/31'
Pulse                92
Maxpulse            115
Calories            243
Name: 31, dtype: object
```

Subsetting Multiple rows

```
print(df.iloc[[0,9,19]])
```

```
    Duration          Date  Pulse  Maxpulse  Calories
0         60  '2020/12/01'    110       130     409.1
9         60  '2020/12/10'     98       124     269.0
19        60  '2020/12/19'    103       123     323.0
```

✓ 0s completed at 1:06 PM