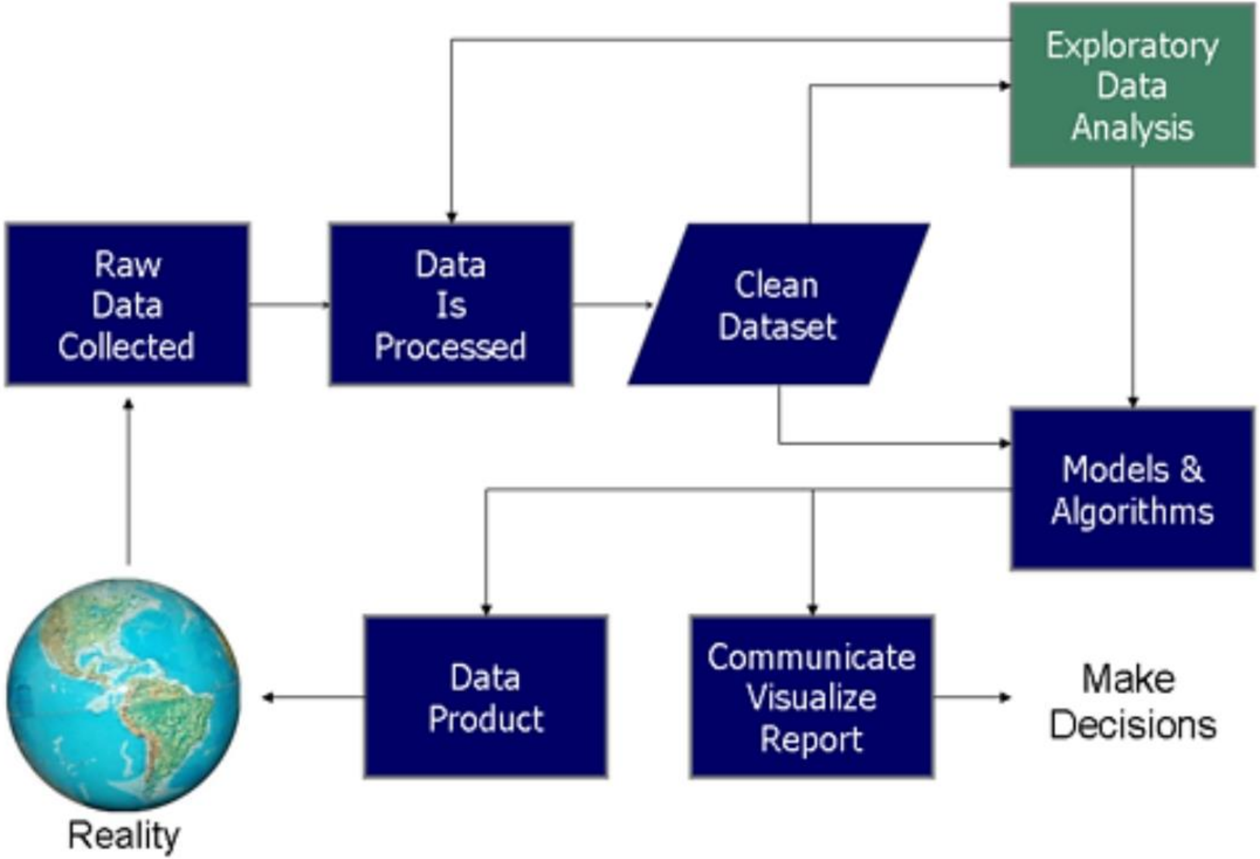
Data Analytics Course - Lesson 03

Ths. Vu Duy Khuong

Agenda

- ❑I. Introduction to Cleaning and Preparing Dataset
- ❑II. Cleaning and Preparing Dataset - Part 1

I. Introduction to Data Cleaning



I. Introduction to Data Cleaning

1. What is Data Cleaning?

- *Data cleaning* is the process of changing or eliminating garbage, incorrect, duplicate, corrupted, or incomplete data in a dataset.

- There's no such absolute way to describe the precise steps in the data cleaning process because the processes may vary from dataset to dataset.

- Data cleaning plays an important part in developing reliable answers and within the analytical process and is observed to be a basic feature of the info science basics.

I. Introduction to Data Cleaning

2. Why data cleaning is essential?

- Data cleaning is the most important task that should be done as a data science professional.

- Having wrong or bad quality data can be detrimental to processes and analysis.

- Having clean data will ultimately increase overall productivity and permit the very best quality information in your decision-making.

2. Why data cleaning is essential?

- **Error-Free Data:** When multiple sources of data are combined there may be chances of so much error. Through Data Cleaning, errors can be removed from data.

- **Data Quality:** The quality of the data is the degree to which it follows the rules of particular requirements.

- **Accurate and Efficient:** Ensuring the data is close to the correct values. We know that most of the data in a dataset are valid, and we should focus on establishing its accuracy.

- **Complete Data:** Completeness is the degree to which we should know all the required values. Completeness is a little more challenging to achieve than accuracy or quality.

- **Maintains Data Consistency:** To ensure the data is consistent within the same dataset or across multiple datasets, we can measure consistency by comparing two similar systems.

3. Data cleaning Cycle

- It is the method of analyzing, distinguishing, and correcting untidy, raw data.

- Data cleaning involves filling in missing values, distinguish and fix errors present in the dataset.

- Whereas the techniques used for data cleaning might vary in step with different types of datasets, the following are standard steps to map out data cleaning:

II. Handle Missing Data

1. What is missing data?

- Missing Data is the phenomenon of missing some values in the data set. Those missing positions can be represented by a zero, a negative number, a space, or a special character.

- This phenomenon can be caused by errors in data collection, or data corruption (corruption) during storage and exchange.

- Most ML algorithms cannot work with Missing Data, or if they do, the results are unreliable. Therefore, we need to eliminate this problem before performing the data modeling steps.

II. Handle Missing Data

- Import pandas

```
>>> import pandas as pd
```

- Import Dataset

```
>>> data = pd.read_csv(/content/Iris.csv)
>>> data.head()
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

2. Missing Data Detection

There are 4 ways to find the null values if present in the dataset:

- Using `isnull()` function: `data.isnull()`

This function provides the ***boolean value*** for the complete dataset to know if any null value is present or not.

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	False	False	False	False	False	False
1	False	False	False	False	False	False
2	False	False	False	False	False	False
3	False	False	False	False	False	False
4	False	False	False	False	False	False
...	...	...	...	...	...	...
145	False	False	False	False	False	False
146	False	False	False	False	False	False
147	False	False	False	False	False	False
148	False	False	False	False	False	False
149	False	False	False	False	False	False

150 rows × 6 columns

II. Handle Missing Data

2. Missing Data Detection

There are 4 ways to find the null values if present in the dataset:

- Using `isna()` function: `data.isna()`

This is the same as the `isnull()` function. It provides the same output.

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	False	False	False	False	False	False
1	False	False	False	False	False	False
2	False	False	False	False	False	False
3	False	False	False	False	False	False
4	False	False	False	False	False	False
...	...	...	...	...	...	...
145	False	False	False	False	False	False
146	False	False	False	False	False	False
147	False	False	False	False	False	False
148	False	False	False	False	False	False
149	False	False	False	False	False	False

150 rows × 6 columns

II. Handle Missing Data

2. Missing Data Detection

There are 4 ways to find the null values if present in the dataset:

- Using `isna().any()` function: `data.isna().any()`

This function also gives a ***boolean value*** if any null value is present or not, but it gives results column-wise, not in tabular format.

```
Id               False
SepalLengthCm    False
SepalWidthCm     False
PetalLengthCm    False
PetalWidthCm     False
Species          False
dtype: bool
```

II. Handle Missing Data

2. Missing Data Detection

There are 4 ways to find the null values if present in the dataset:

- Using `isna(). sum()` function: `data.isna().sum()`

This function gives the ***sum of the null values*** preset in the dataset column-wise.

```
Id               0
SepalLengthCm    0
SepalWidthCm     0
PetalLengthCm    0
PetalWidthCm     0
Species          0
dtype: int64
```

3. Handle Missing Data

- Delete rows/columns containing Missing Data

Usually, if the Missing Data rate is less than 5% of the total, we should ***delete*** them.

```
# example of removing rows that contain missing values
from numpy import nan
from pandas import read_csv
# load the dataset
dataset = read_csv('pima-indians-diabetes.csv' , header=None)
# summarize the shape of the raw data
print(dataset.shape)
# replace ' 0 ' values with ' nan '
dataset[[1,2,3,4,5]] = dataset[[1,2,3,4,5]].replace(0, nan)
# drop rows with missing values
dataset.dropna(inplace=True)
# summarize the shape of the data with missing rows removed
print(dataset.shape)
```

```
(768, 9)
(392, 9)
```

3. Handle Missing Data

- Statistic Imputation

This is a method that uses statistical values to replace Missing Data. Its advantage is simple, fast calculation. Some alternatives of Missing Data with statistical value that can be used here are:

- Replace Missing Data in a column with the average value of the column containing that Missing Data.
- Replace Missing Data in a column with the median value of the column containing that Missing Data.
- Replace Missing Data in a column with the most frequently occurring value (mode) in the column containing that Missing Data.
- Replace Missing Data in a column with another constant value.

3. Handle Missing Data

- Statistic Imputation

Check missing values:

```
# summarize the horse colic dataset
from pandas import read_csv
# load dataset
dataframe = read_csv('horse-colic.csv' , header=None, na_values= '?')
# summarize the first few rows
print(dataframe.head())
# summarize the number of rows with missing values for each column
for i in range(dataframe.shape[1]):
    # count number of rows with missing values
    n_miss = dataframe[[i]].isnull().sum()
    perc = n_miss / dataframe.shape[0] * 100
    print('> %d, Missing: %d (%.1f%%)' % (i, n_miss, perc))
```

```
> 0, Missing: 1 (0.3%)
> 1, Missing: 0 (0.0%)
> 2, Missing: 0 (0.0%)
> 3, Missing: 60 (20.0%)
> 4, Missing: 24 (8.0%)
> 5, Missing: 58 (19.3%)
> 6, Missing: 56 (18.7%)
> 7, Missing: 69 (23.0%)
> 8, Missing: 47 (15.7%)
> 9, Missing: 32 (10.7%)
> 10, Missing: 55 (18.3%)
> 11, Missing: 44 (14.7%)
> 12, Missing: 56 (18.7%)
```

3. Handle Missing Data

- Statistic Imputation

- First, declare an Instance of the ***SimpleImputer*** class, pass in the type of statistics you want to use: ***mean, median, mode, etc.***

```
# define imputer
imputer = SimpleImputer(strategy= 'mean')
```

- Next, use the imputer just declared fit on the dataset to calculate the average of each column.

```
...
# fit on the dataset
imputer.fit(X)
```

- Finally, the imputer is applied to the entire data set to create a new instance of the data set in which all Missing Data is replaced by the mean of its containing column.

```
...
# fit on the dataset
imputer.fit(X)
```

3. Handle Missing Data

- Statistic Imputation

Complete code:

```
# statistical imputation transform for the horse colic dataset
from numpy import isnan
from pandas import read_csv
from sklearn.impute import SimpleImputer

# load dataset
dataframe = read_csv('horse-colic.csv' , header=None, na_values= '?' )
# split into input and output elements. #23 is label column
data = dataframe.values
ix = [i for i in range(data.shape[1]) if i != 23]
X, y = data[:, ix], data[:, 23]
# summarize total missing
print( 'Missing: %d' % sum(isnan(X).flatten()))
# define imputer
imputer = SimpleImputer(strategy= 'mean' )
# fit on the dataset
imputer.fit(X)
# transform the dataset
Xtrans = imputer.transform(X)
# summarize total missing
print( 'Missing: %d' % sum(isnan(Xtrans).flatten()))
```

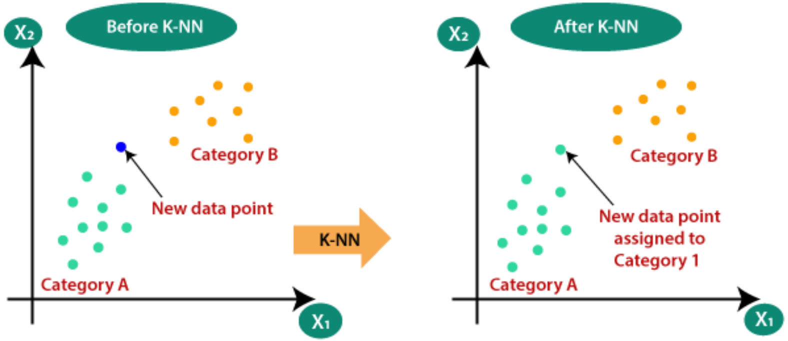
```
Missing: 1605
Missing: 0
```

3. Handle Missing Data

K-Nearest Neighbor (kNN) Imputation

- ***kNN*** is a simple supervised-learning algorithm that does not learn anything from the training data, all computations are performed when it needs to predict the outcome of the new data.

- The configuration for ***kNN*** usually includes the selection of 2 values: ***type of metric*** that measures the distance between the data samples (***Euclidean***, ***Cosine***, ...) and the ***number of samples (k)*** adjacent to the sample to determine the ***value/class.***



3. Handle Missing Data

K-Nearest Neighbor (kNN) Imputation

- Declare an Instance of ***KNNImputer***

```
...
# define imputer
imputer = KNNImputer(n_neighbors=5, weights= 'distance' , metric= 'nan_euclidean')
```

- Calculate value for Missing Data on dataset

```
...
# fit on the dataset
imputer.fit(X)
```

- Create Transform Data

```
...
# transform the dataset
Xtrans = imputer.transform(X)
```

3. Handle Missing Data

K-Nearest Neighbor (kNN) Imputation

Complete code:

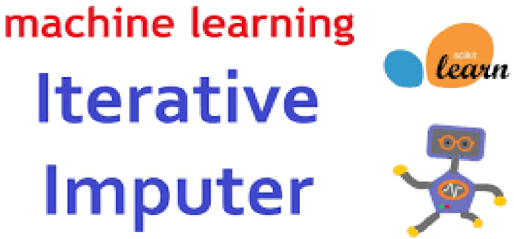
```
# knn imputation transform for the horse colic dataset
from numpy import isnan
from pandas import read_csv
from sklearn.impute import KNNImputer
# load dataset
dataframe = read_csv('horse-colic.csv' , header=None, na_values= '?')
# split into input and output elements
data = dataframe.values
ix = [i for i in range(data.shape[1]) if i != 23]
X, y = data[:, ix], data[:, 23]
# summarize total missing
print('Missing: %d' % sum(isnan(X).flatten()))
# define imputer
imputer = KNNImputer()
# fit on the dataset
imputer.fit(X)
# transform the dataset
Xtrans = imputer.transform(X)
# summarize total missing
print('Missing: %d' % sum(isnan(Xtrans).flatten()))
```

```
Missing: 1605
Missing: 0
```

3. Handle Missing Data

- Iterative Imputation

- Iterative Imputation is the process in which each feature is modeled as a function of other features.
- Each feature is defined sequentially, in turn, allowing previously defined features to be used as part of the model's input in predicting subsequent features.
- This process is repeated many times, allowing estimates to always improve.



3. Handle Missing Data

- Iterative Imputation

- Declare an Instance of Iterative Imputer

```
...
# define imputer
imputer = IterativeImputer(estimator=BayesianRidge(), imputation_order= 'ascending')
```

- Estimating the value for Missing Data on the dataset

```
...
# fit on the dataset
imputer.fit(X)
```

- Create Transform Data

```
...
# transform the dataset
Xtrans = imputer.transform(X)
```

3. Handle Missing Data

- Iterative Imputation

Complete code:

```
# iterative imputation transform for the horse colic dataset
from numpy import isnan
from pandas import read_csv
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
# load dataset
dataframe = read_csv('horse-colic.csv', header=None, na_values='?')
# split into input and output elements
data = dataframe.values
ix = [i for i in range(data.shape[1]) if i != 23]
X, y = data[:, ix], data[:, 23]
# summarize total missing
print('Missing: %d' % sum(isnan(X).flatten()))
# define imputer
imputer = IterativeImputer()
# fit on the dataset
imputer.fit(X)
# transform the dataset
Xtrans = imputer.transform(X)
# summarize total missing
print('Missing: %d' % sum(isnan(Xtrans).flatten()))
```

```
Missing: 1605
Missing: 0
```

III. Clean Data of Wrong Format

1. 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.

III. Clean Data of Wrong Format

2. Convert into correct format

	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

```python
import pandas as pd

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

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

III. Clean Data of Wrong Format

2. Convert into correct format

	A	B	C
2020-12-02	0	0	0
2020-12-03	1	1	1
2020-12-04	2	2	Sahil
2020-12-05	3	3	3
2020-12-06	4	4	Robin
2020-12-07	5	5	5
2020-12-08	6	6	6
2020-12-09	7	7	7
2020-12-10	8	8	DSL
2020-12-11	9	9	9

```
pd.to_numeric(df["A"])
```

```
2020-12-02    0
2020-12-03    1
2020-12-04    2
2020-12-05    3
2020-12-06    4
2020-12-07    5
2020-12-08    6
2020-12-09    7
2020-12-10    8
2020-12-11    9
Freq: D, Name: A, dtype: int32
```

```
pd.to_numeric(df["C"],errors="coerce")
```

```
2020-12-02    0.0
2020-12-03    1.0
2020-12-04    NaN
2020-12-05    3.0
2020-12-06    NaN
2020-12-07    5.0
2020-12-08    6.0
2020-12-09    7.0
2020-12-10    NaN
2020-12-11    9.0
Freq: D, Name: C, dtype: float64
```

IV. Fixing Wrong Data

1. 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.

	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

2. Replace Values

One way to fix wrong values is to replace them with something else.

	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

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

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

3. Remove Rows

- 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.

	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

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

V. Removing Duplicates

1. Discovering Duplicates

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

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

- The *duplicated()* method returns a ***Boolean*** values for each 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

V. Removing Duplicates

1. Discovering Duplicates

```python
import pandas as pd

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

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
```

V. Removing Duplicates

2. Removing Duplicates

To remove duplicates, use the `drop_duplicates()` method.

```
import pandas as pd

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

df.drop_duplicates(inplace = True)

print(df.to_string())

#Notice that row 12 has been removed from the result
```

```
    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
```

V. Removing Duplicates

2. Removing Duplicates

To remove duplicates, use the `drop_duplicates()` method.

```
import pandas as pd

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

df.drop_duplicates(inplace = True)

print(df.to_string())

#Notice that row 12 has been removed from the result
```

	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

VI. Reference

Book:

Python Data Science Handbook, chapter 3

Q & A