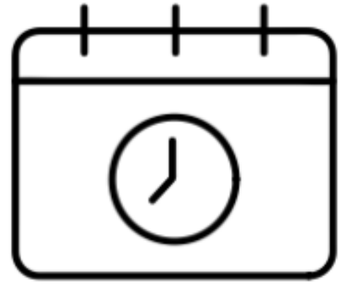
In this chapter

Chapter 3 - Advanced data problems

Data range constraints

Out of range movie ratings



Subscription dates in the future

Uniformity

Column	Unit
Temperature	`32°C` **is also** `89.6°F`
Weight	`70 Kg` **is also** `11 st.`
Date	`26-11-2019` **is also** `26, November, 2019`
Money	`100$` **is also** `10763.90¥`

An example

```
temperatures = pd.read_csv('temperature.csv')
temperatures.head()
```

```
        Date  Temperature
0  03.03.19         14.0
1  04.03.19         15.0
2  05.03.19         18.0
3  06.03.19         16.0
4  07.03.19         62.6
```

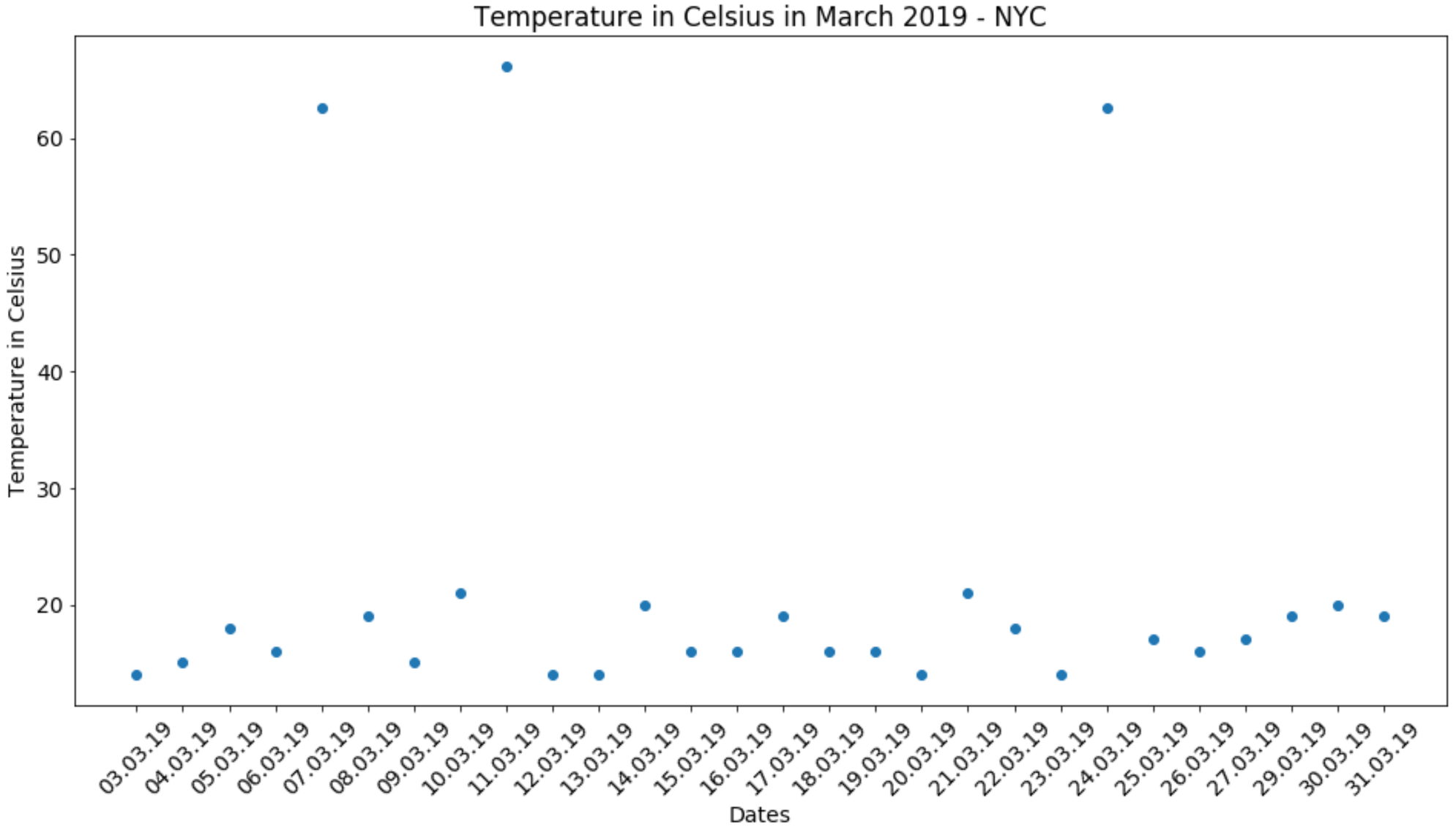
An example

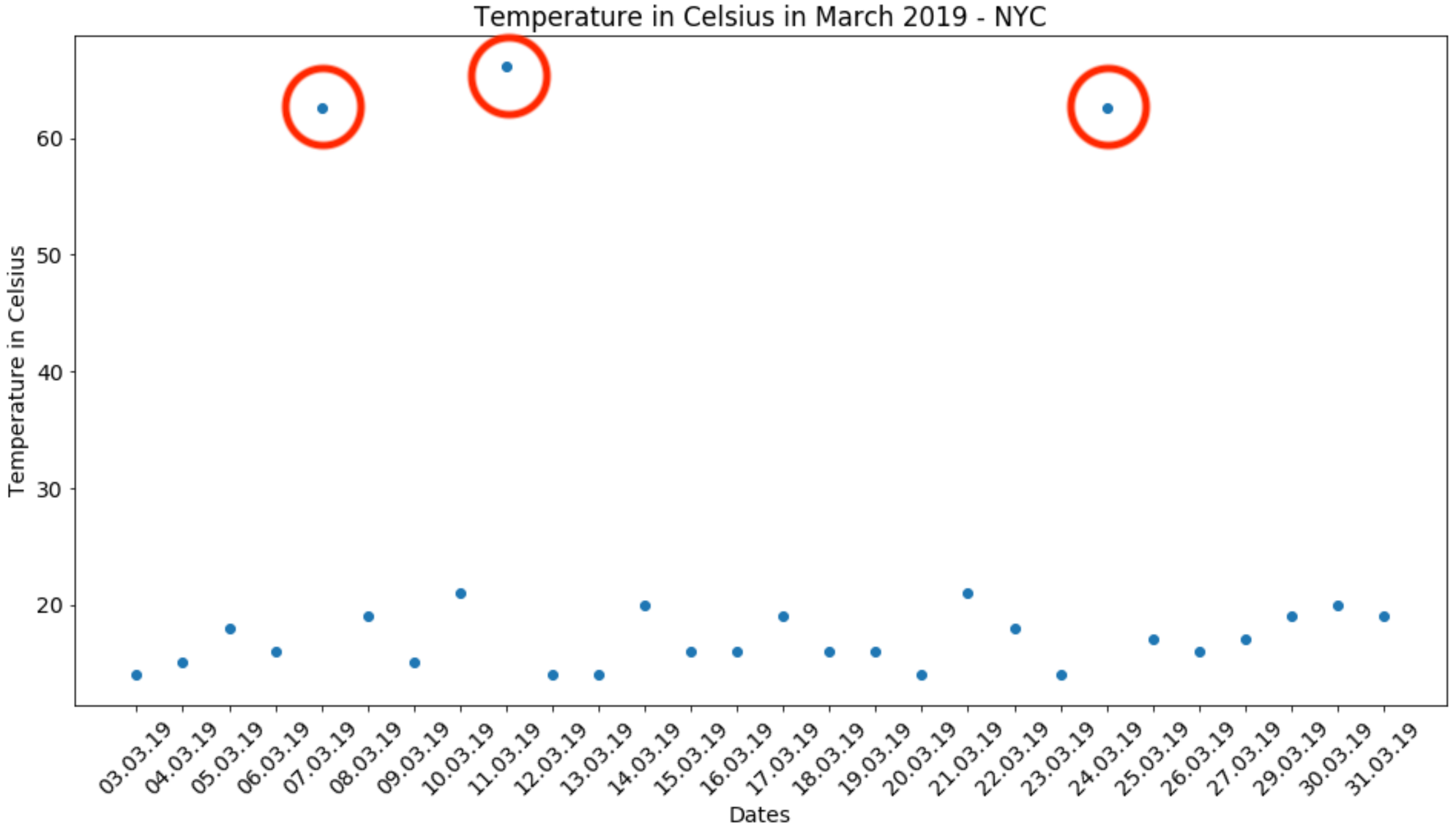
```
temperatures = pd.read_csv('temperature.csv')
temperatures.head()
```

```
        Date  Temperature
0  03.03.19         14.0
1  04.03.19         15.0
2  05.03.19         18.0
3  06.03.19         16.0
4  07.03.19         62.6   <--
```

An example

```
# Import matplotlib
import matplotlib.pyplot as plt
# Create scatter plot
plt.scatter(x = 'Date', y = 'Temperature', data = temperatures)
# Create title, xlabel and ylabel
plt.title('Temperature in Celsius March 2019 - NYC')
plt.xlabel('Dates')
plt.ylabel('Temperature in Celsius')
# Show plot
plt.show()
```


Temperature in Celsius in March 2019 - NYC
Temperature in Celsius
60
50
40
30
20
03.03.19
04.03.19
05.03.19
06.03.19
07.03.19
08.03.19
09.03.19
10.03.19
11.03.19
12.03.19
13.03.19
14.03.19
15.03.19
16.03.19
17.03.19
18.03.19
19.03.19
20.03.19
21.03.19
22.03.19
23.03.19
24.03.19
25.03.19
26.03.19
27.03.19
29.03.19
30.03.19
31.03.19
Dates


Temperature in Celsius in March 2019 - NYC
Temperature in Celsius
60
50
40
30
20
03.03.19
04.03.19
05.03.19
06.03.19
07.03.19
08.03.19
09.03.19
10.03.19
11.03.19
12.03.19
13.03.19
14.03.19
15.03.19
16.03.19
17.03.19
18.03.19
19.03.19
20.03.19
21.03.19
22.03.19
23.03.19
24.03.19
25.03.19
26.03.19
27.03.19
29.03.19
30.03.19
31.03.19
Dates

Treating temperature data

$$C = (F - 32) \times \frac{5}{9}$$

```
temp_fah = temperatures.loc[temperatures['Temperature'] > 40, 'Temperature']
temp_cels = (temp_fah - 32) * (5/9)
temperatures.loc[temperatures['Temperature'] > 40, 'Temperature'] = temp_cels
```

```
# Assert conversion is correct
assert temperatures['Temperature'].max() < 40
```

Treating date data

```
birthdays.head()
```

```
          Birthday First name Last name
0         27/27/19      Rowan     Nunez
1         03-29-19      Brynn      Yang
2  March 3rd, 2019     Sophia    Reilly
3         24-03-19     Deacon    Prince
4         06-03-19   Griffith      Neal
```

Treating date data

```
birthdays.head()
```

```
           Birthday First name Last name
0          27/27/19       Rowan     Nunez    ??
1          03-29-19       Brynn      Yang    MM-DD-YY
2  March 3rd, 2019      Sophia    Reilly    Month Day, YYYY
3          24-03-19      Deacon    Prince
4          06-03-19    Griffith      Neal
```

Datetime formatting

`datetime` is useful for representing dates

Date	`datetime` format
25-12-2019	`%d-%m-%Y`
December 25th 2019	`%c`
12-25-2019	`%m-%d-%Y`
...	...

`pandas.to_datetime()`

- Can recognize most formats automatically
- Sometimes fails with erroneous or unrecognizable formats

Treating date data

```
# Converts to datetime - but won't work!
birthdays['Birthday'] = pd.to_datetime(birthdays['Birthday'])
```

```
ValueError: month must be in 1..12
```

```
# Will work!
birthdays['Birthday'] = pd.to_datetime(birthdays['Birthday'],
                                       # Attempt to infer format of each date
                                       infer_datetime_format=True,
                                       # Return NA for rows where conversion failed
                                       errors = 'coerce')
```

Treating date data

```
birthdays.head()
```

```
    Birthday First name Last name
0        NaT       Rowan     Nunez
1 2019-03-29       Brynn      Yang
2 2019-03-03      Sophia    Reilly
3 2019-03-24      Deacon    Prince
4 2019-06-03    Griffith      Neal
```

Treating date data

```
birthdays['Birthday'] = birthdays['Birthday'].dt.strftime("%d-%m-%Y")
birthdays.head()
```

```
     Birthday First name Last name
0         NaT       Rowan     Nunez
1  29-03-2019       Brynn      Yang
2  03-03-2019      Sophia    Reilly
3  24-03-2019      Deacon    Prince
4  03-06-2019    Griffith      Neal
```

Treating ambiguous date data

Is `2019-03-08` ***in August or March?***

- Convert to `NA` and treat accordingly
- Infer format by understanding data source
- Infer format by understanding previous and subsequent data in DataFrame

Let's practice!

CLEANING DATA IN PYTHON

Motivation

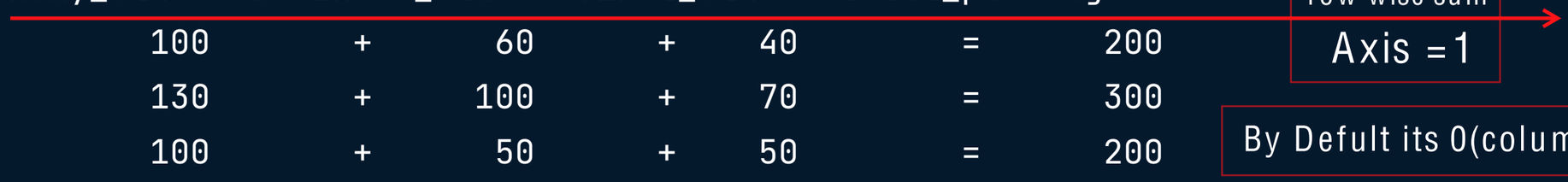
```
import pandas as pd

flights = pd.read_csv('flights.csv')
flights.head()
```

	flight_number	economy_class	business_class	first_class	total_passengers
0	DL140	100	60	40	200
1	BA248	130	100	70	300
2	MEA124	100	50	50	200
3	AFR939	140	70	90	300
4	TKA101	130	100	20	250

Cross field validation

*The use of **multiple** fields in a dataset to sanity check data integrity*

	flight_number	economy_class		business_class		first_class		total_passengers
0	DL140	100	+	60	+	40	=	200
1	BA248	130	+	100	+	70	=	300
2	MEA124	100	+	50	+	50	=	200
3	AFR939	140	+	70	+	90	=	300
4	TKA101	130	+	100	+	20	=	250




```
sum_classes = flights[['economy_class', 'business_class', 'first_class']].sum(axis = 1)
passenger_equ = sum_classes == flights['total_passengers']
# Find and filter out rows with inconsistent passenger totals
inconsistent_pass = flights[~passenger_equ]
consistent_pass = flights[passenger_equ]
```

Cross field validation

```
users.head()
```

```
   user_id  Age   Birthday
0    32985   22 1998-03-02
1    94387   27 1993-12-04
2    34236   42 1978-11-24
3    12551   31 1989-01-03
4    55212   18 2002-07-02
```

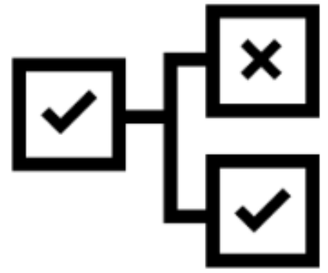
Cross field validation

```
import pandas as pd
import datetime as dt

# Convert to datetime and get today's date
users['Birthday'] = pd.to_datetime(users['Birthday'])
today = dt.date.today()
# For each row in the Birthday column, calculate year difference
age_manual = today.year - users['Birthday'].dt.year
# Find instances where ages match
age_equ = age_manual == users['Age']
# Find and filter out rows with inconsistent age
inconsistent_age = users[~age_equ]
consistent_age = users[age_equ]
```

What to do when we catch inconsistencies?

Dropping Data

Set to missing and impute



Apply rules from domain knowledge

Let's practice!

CLEANING DATA IN PYTHON

What is missing data?

Occurs when no data value is stored for a variable in an observation

Can be represented as `NA`, `nan`, `0`, `.` ...

Technical error

Human error

Airquality example

```
import pandas as pd
airquality = pd.read_csv('airquality.csv')
print(airquality)
```

```
            Date  Temperature  CO2
987   20/04/2004         16.8  0.0
2119  07/06/2004         18.7  0.8
2451  20/06/2004        -40.0  NaN
1984  01/06/2004         19.6  1.8
8299  19/02/2005         11.2  1.2
...       ...             ...  ...
```

Airquality example

```
import pandas as pd
airquality = pd.read_csv('airquality.csv')
print(airquality)
```

```
            Date  Temperature  CO2
987   20/04/2004         16.8  0.0
2119  07/06/2004         18.7  0.8
2451  20/06/2004        -40.0  NaN   <--
1984  01/06/2004         19.6  1.8
8299  19/02/2005         11.2  1.2
...       ...             ...  ...
```

Airquality example

```
# Return missing values
airquality.isna()
```

```
       Date  Temperature    CO2
987   False        False  False
2119  False        False  False
2451  False        False   True
1984  False        False  False
8299  False        False  False
```

Airquality example

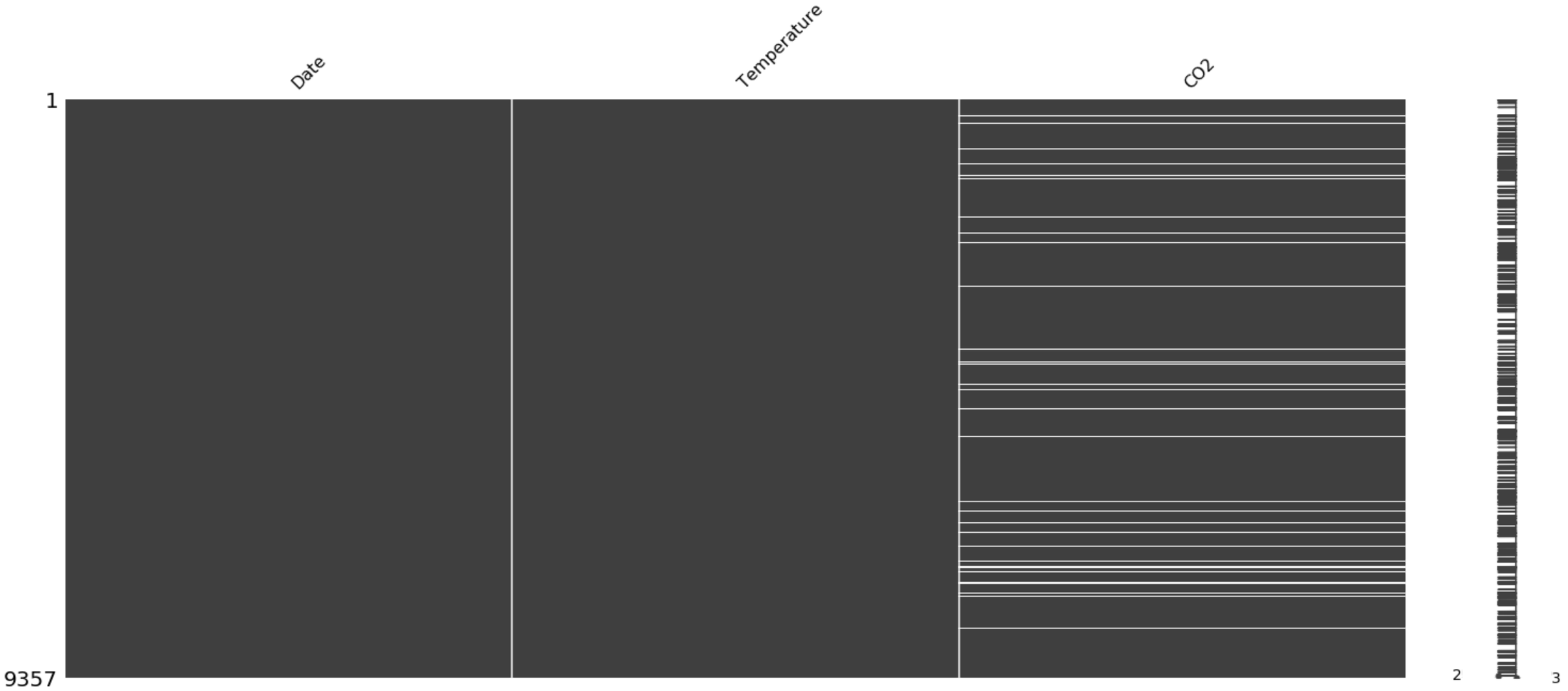
```
# Get summary of missingness
airquality.isna().sum()
```


```
Date              0
Temperature       0
CO2             366
dtype: int64
```

Missingno

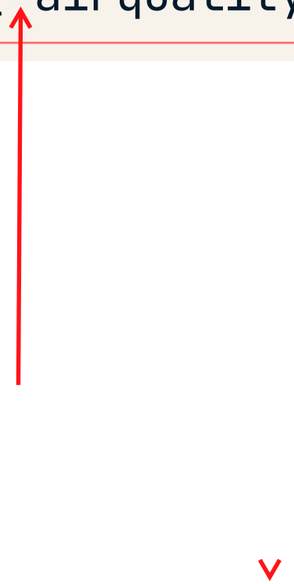
Useful package for visualizing and understanding missing data

```
import missingno as msno
import matplotlib.pyplot as plt
# Visualize missingness
msno.matrix(airquality)
plt.show()
```


Date
Temperature
CO2
1
9357
2
3

Airquality example

```
# Isolate missing and complete values aside
missing = airquality[airquality['CO2'].isna()]
complete = airquality[~airquality['CO2'].isna()]
```



Airquality example

```
# Describe complete DataFramee
complete.describe()
```

```
       Temperature          CO2
count  8991.000000  8991.000000
mean     18.317829     1.739584
std       8.832116     1.537580
min      -1.900000     0.000000
...            ...          ...
max      44.600000    11.900000
```

```
# Describe missing DataFramee
missing.describe()
```

```
       Temperature  CO2
count   366.000000  0.0
mean    -39.655738  NaN
std       5.988716  NaN
min     -49.000000  NaN
...            ...  ...
max     -30.000000  NaN
```

Airquality example

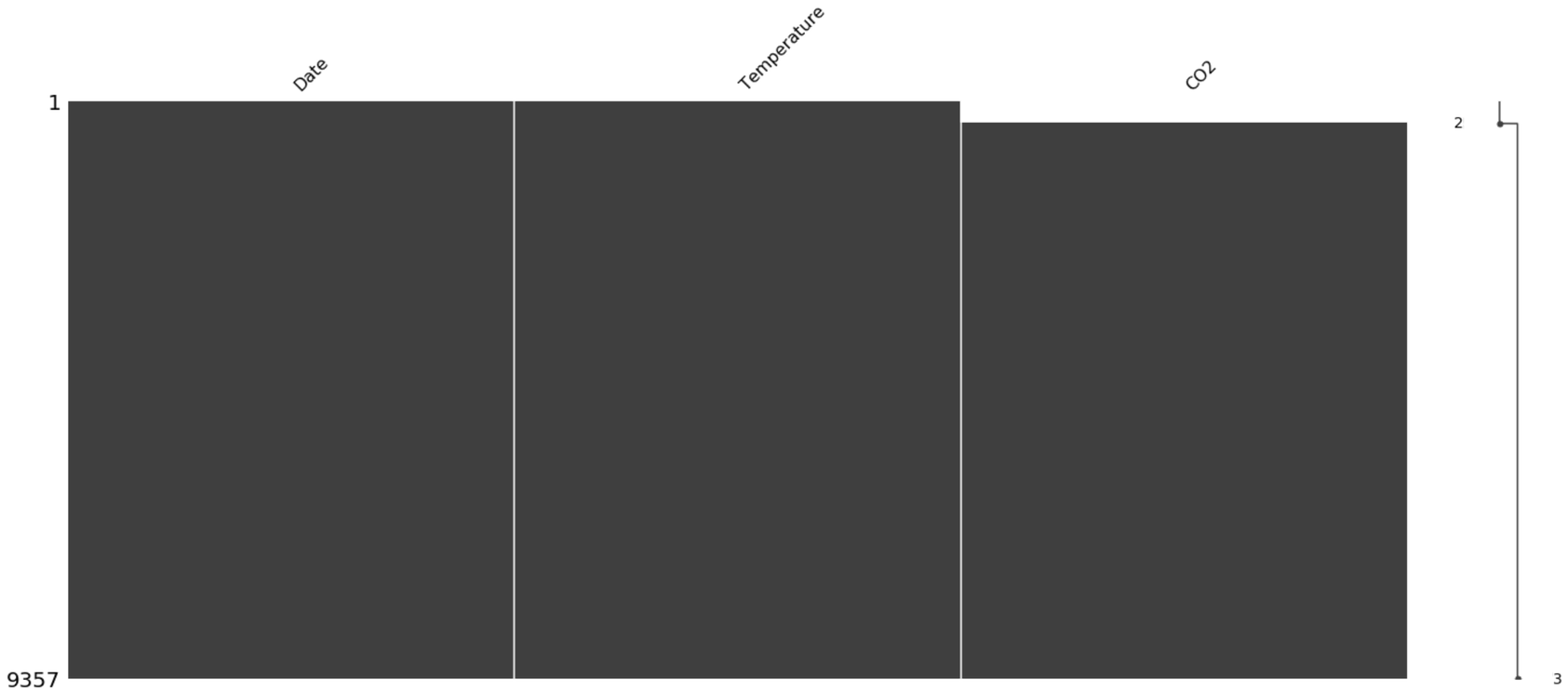
```
# Describe complete DataFramee
complete.describe()
```

```
       Temperature          CO2
count  8991.000000  8991.000000
mean     18.317829     1.739584
std       8.832116     1.537580
min      -1.900000     0.000000
...            ...          ...
max      44.600000    11.900000
```

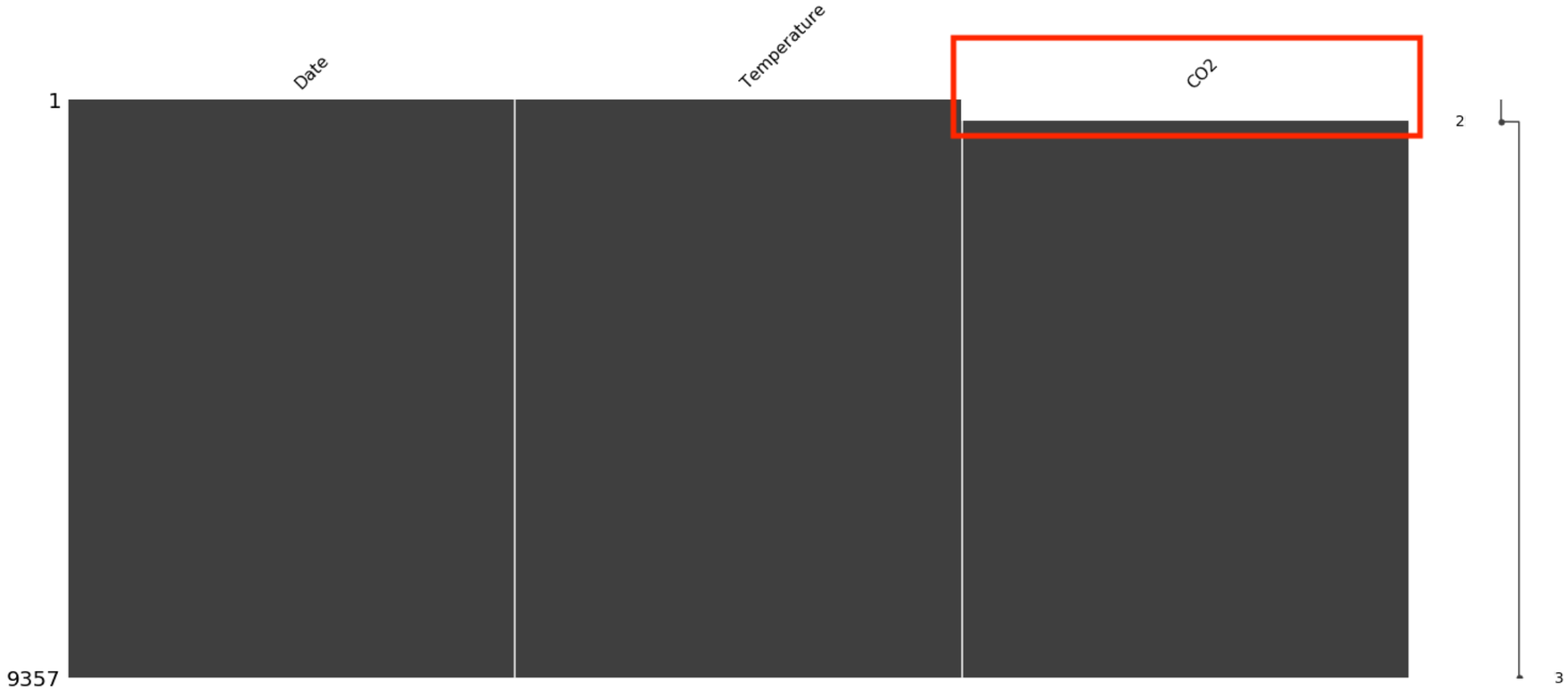
```
# Describe missing DataFramee
missing.describe()
```

```
       Temperature  CO2
count   366.000000  0.0
mean    -39.655738  NaN   <--
std       5.988716  NaN
min     -49.000000  NaN   <--
...            ...  ...
max     -30.000000  NaN   <--
```

```
sorted_airquality = airquality.sort_values(by = 'Temperature')
msno.matrix(sorted_airquality)
plt.show()
```



```
sorted_airquality = airquality.sort_values(by = 'Temperature')
msno.matrix(sorted_airquality)
plt.show()
```



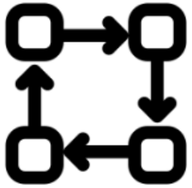
Missingness types

Missing Completely at Random

(MCAR)

Missing at Random

(MAR)



Missing Not at Random

(MNAR)

Missingness types

Missing Completely at Random

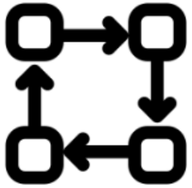
(MCAR)

No systematic relationship between missing data and other values

Data entry errors when inputting data

Missing at Random

(MAR)



Missing Not at Random

(MNAR)

Missingness types

Missing Completely at Random

(MCAR)

No systematic relationship between missing data and other values

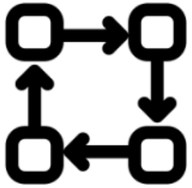
Data entry errors when inputting data

Missing at Random

(MAR)

Systematic relationship between missing data and other observed values

Missing ozone data for high temperatures



Missing Not at Random

(MNAR)

Missingness types

Missing Completely at Random

(MCAR)

No systematic relationship between missing data and other values

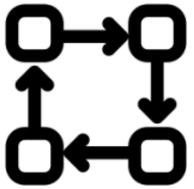
Data entry errors when inputting data

Missing at Random

(MAR)

Systematic relationship between missing data and other observed values

Missing ozone data for high temperatures



Missing Not at Random

(MNAR)

Systematic relationship between missing data and unobserved values

Missing temperature values for high temperatures

How to deal with missing data?

Simple approaches:

1. Drop missing data
2. Impute with statistical measures *(mean, median, mode..)*

More complex approaches:

1. Imputing using an algorithmic approach
2. Impute with machine learning models

Dealing with missing data

```
airquality.head()
```

```
         Date  Temperature  CO2
0  05/03/2005          8.5  2.5
1  23/08/2004         21.8  0.0
2  18/02/2005          6.3  1.0
3  08/02/2005        -31.0  NaN
4  13/03/2005         19.9  0.1
```

Dropping missing values

```
# Drop missing values
airquality_dropped = airquality.dropna(subset = ['CO2'])
airquality_dropped.head()
```

```
         Date  Temperature  CO2
0  05/03/2005          8.5  2.5
1  23/08/2004         21.8  0.0
2  18/02/2005          6.3  1.0
4  13/03/2005         19.9  0.1
5  02/04/2005         17.0  0.8
```

Replacing with statistical measures

```
co2_mean = airquality['CO2'].mean()
airquality_imputed = airquality.fillna({'CO2': co2_mean})
airquality_imputed.head()
```

dict

```
         Date  Temperature       CO2
0  05/03/2005          8.5  2.500000
1  23/08/2004         21.8  0.000000
2  18/02/2005          6.3  1.000000
3  08/02/2005        -31.0  1.739584
4  13/03/2005         19.9  0.100000
```

Let's practice!

CLEANING DATA IN PYTHON