The University of Sheffield Research IT

ADVANCED PYTHON PROGRAMMING

Data Science

http://rcg.group.shef.ac.uk/courses/python

Course Outline:

- Capture Data
- Manage and Clean Data
- Data Analysis
- Report data

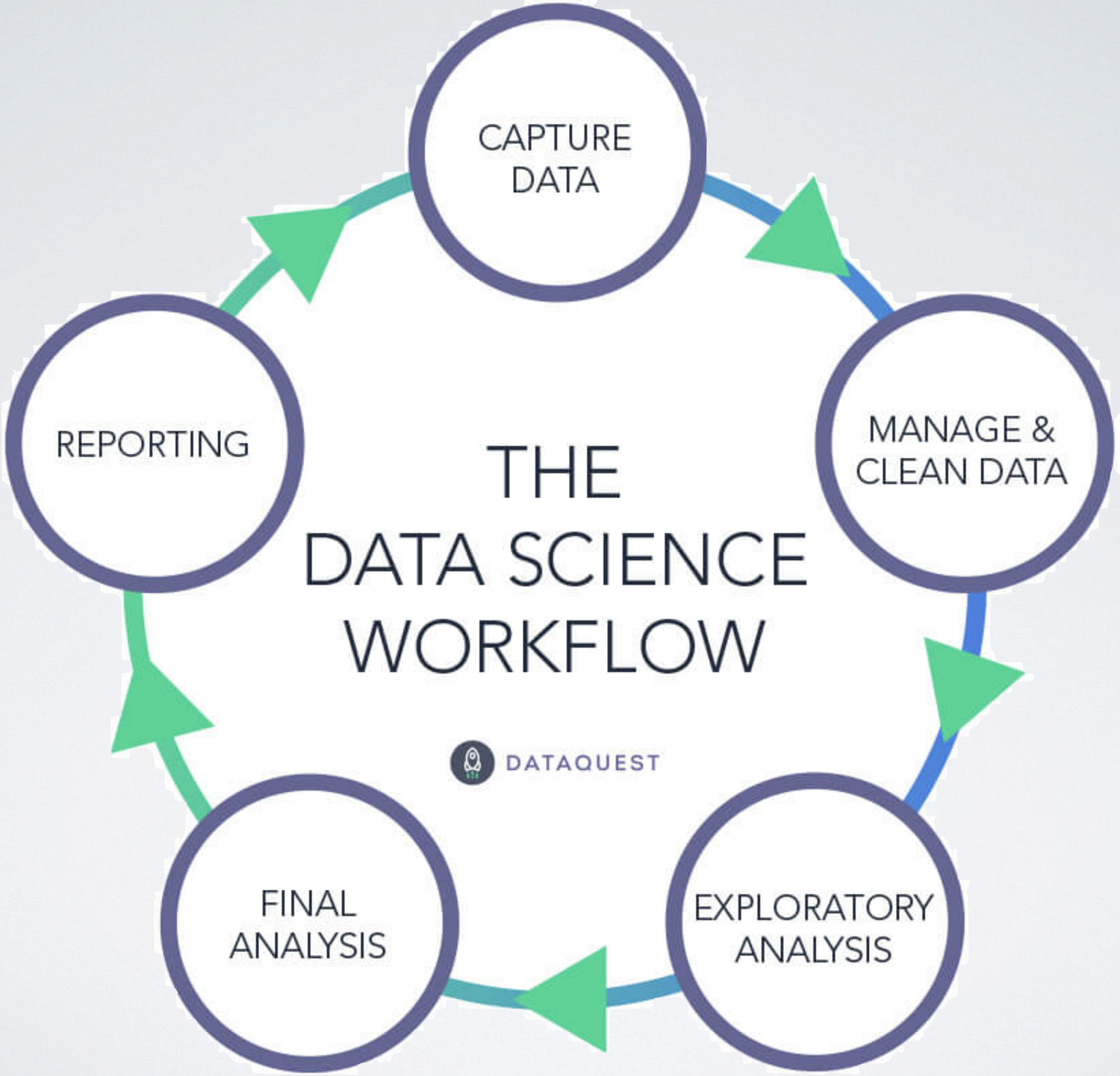
Requirements:

- CI6010a
- CI6010b
- CI6011a

Anaconda Python should be installed on your desktop, please start Spyder.

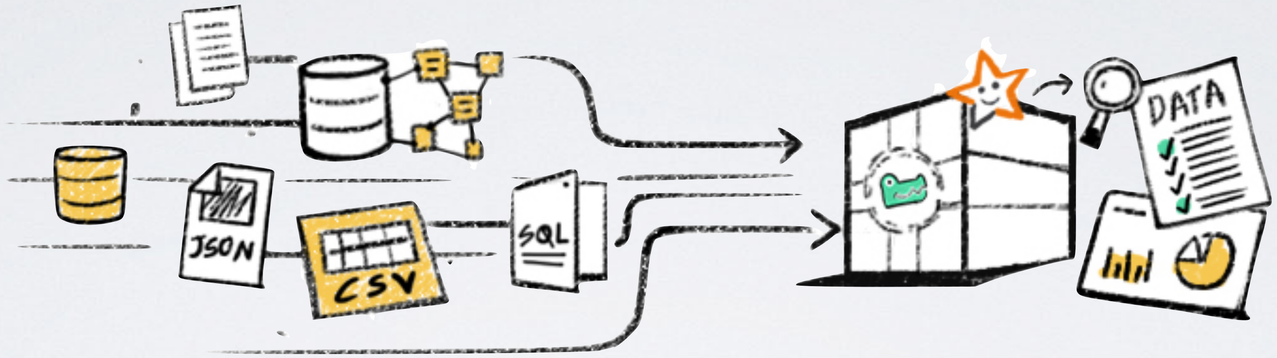
If it is not installed, please go to the Software Centre and install it.

WHAT IS DATA SCIENCE?


CAPTURE DATA
MANAGE & CLEAN DATA
EXPLORATORY ANALYSIS
FINAL ANALYSIS
REPORTING
THE DATA SCIENCE WORKFLOW
DATAQUEST

CAPTURE DATA

Data Sources:



Scraping it from a **website** from **figures**, etc.

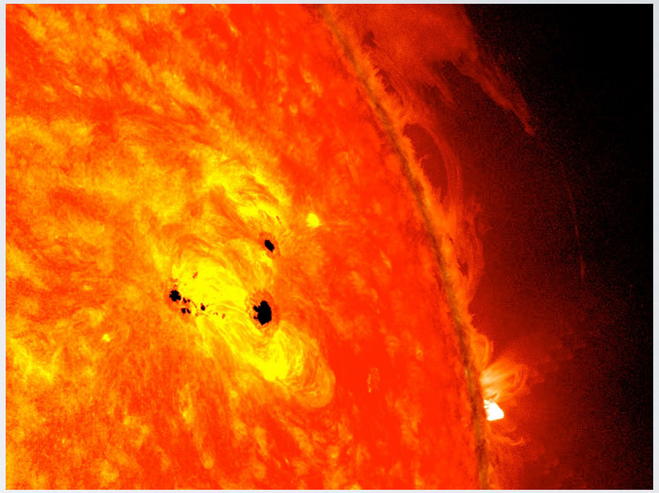
Pulling the data from a **database**.

Data
for study.

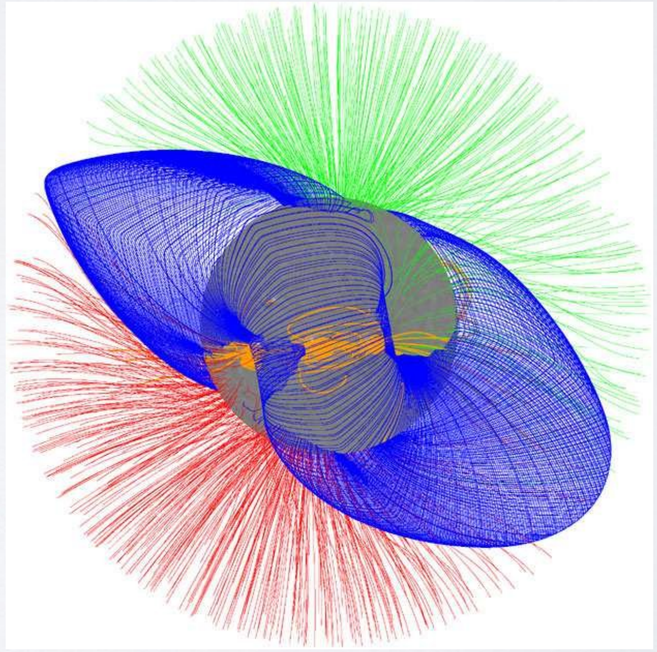
Accessing an **API**, etc.

Data types:

- **Observational**: Captured in real-time, cannot be reproduced.
- **Experimental**: Data from lab equipment and under controlled conditions.
- **Simulation**: Data generated from test models studying actual or theoretical systems
- **Compiled**: The results of data analysis, or aggregated from multiple sources.
- **Canonical**: Fixed or organic collection datasets, usually peer-reviewed, and often published and curated.



Observational data



Simulation data

Reading Data in Python

Unstructured:

- Data without inherent structure.

Quasi-Structured:

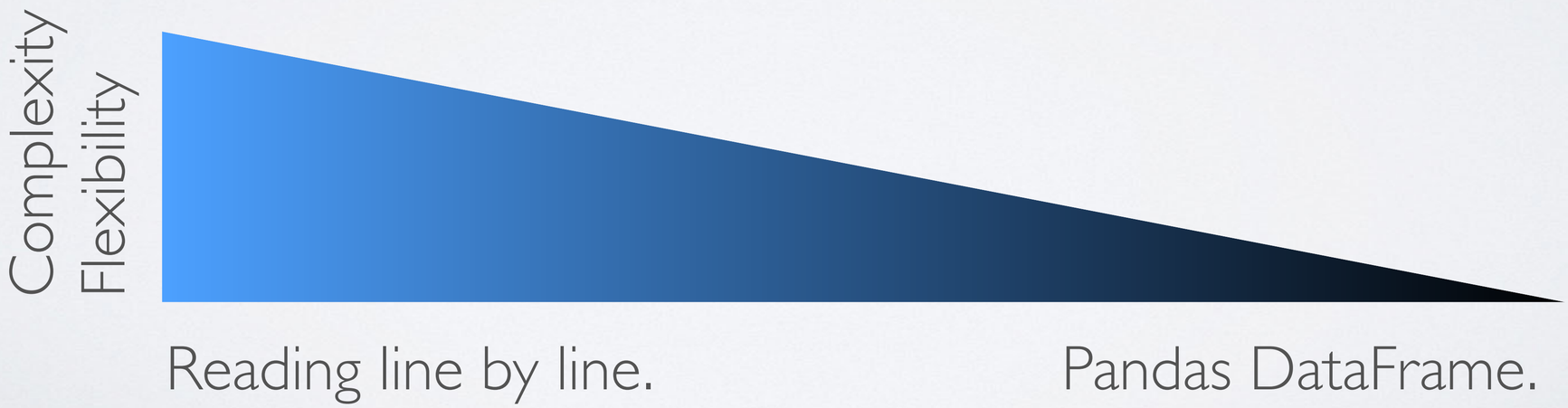
- Textual data with erratic format that can be formatted with effort.

Semi-Structured:

- Textual data with apparent pattern (including errors)

Structured:

- Defined data model (errors less likely).



PANDAS DATAFRAME

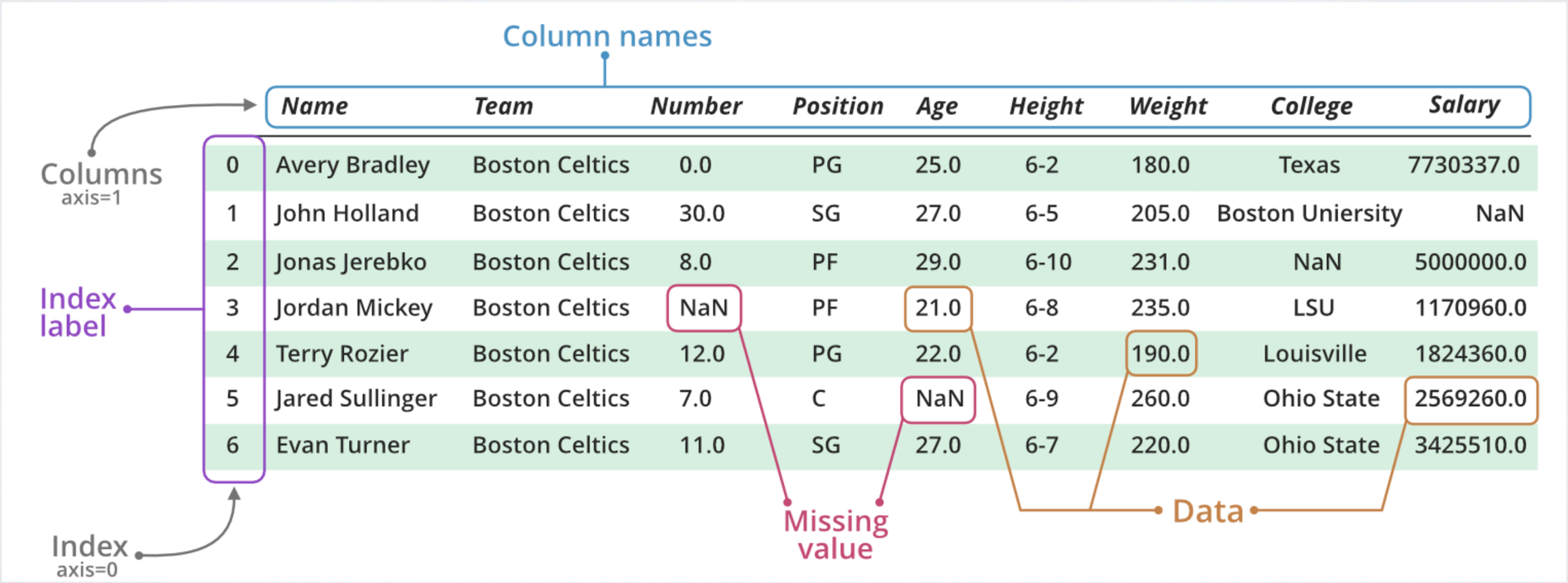
The Pandas DataFrame is a **multi-dimensional** size-mutable, potentially heterogeneous **tabular data structure** with labeled axes (rows and columns).

Advantages:

It can present data in a way that is **suitable for data analysis.**
The package contains multiple methods for convenient **data filtering**.
Pandas has a variety of utilities to perform **Input/Output operations** in a seamless manner.

Constructing a DataFrame

```
import pandas as pd

df1 = pd.read_excel('sample.xlsx') # Excel file
df2 = pd.read_csv('sample.csv')    # Comma Separated file
df3 = pd.read_table('sample.txt', sep= ' ')  # Text file
```



	Name	Team	Number	Position	Age	Height	Weight	College	Salary
0	Avery Bradley	Boston Celtics	0.0	PG	25.0	6-2	180.0	Texas	7730337.0
1	John Holland	Boston Celtics	30.0	SG	27.0	6-5	205.0	Boston Uniersity	NaN
2	Jonas Jerebko	Boston Celtics	8.0	PF	29.0	6-10	231.0	NaN	5000000.0
3	Jordan Mickey	Boston Celtics	NaN	PF	21.0	6-8	235.0	LSU	1170960.0
4	Terry Rozier	Boston Celtics	12.0	PG	22.0	6-2	190.0	Louisville	1824360.0
5	Jared Sullinger	Boston Celtics	7.0	C	NaN	6-9	260.0	Ohio State	2569260.0
6	Evan Turner	Boston Celtics	11.0	SG	27.0	6-7	220.0	Ohio State	3425510.0

Constructing a DataFrame Manually

```
df = pd.DataFrame(data=d, index=i, columns=c)
```

- **Parameter data**: ndarray, iterable, dictionary or DataFrame.
- **Parameter index**: array. RangeIndex by default (0, 1, 2, 3, …, n).
- **Parameter columns**: array. RangeIndex by default (0, 1, 2, 3, …, n) or the keys of a dictionary if the data input is a dictionary.

0	1	2	3
1	*data*	*data*	*data*
2	*data*	*data*	*data*
3	*data*	*data*	*data*
4	*data*	*data*	*data*

```
import pandas as pd

d_1 = [1,2,3]
d_2 = {‘header_1': [1, 2], ‘header_2': [3, 4]}

df_1 = pd.DataFrame(data=d_1) # Constructing DataFrame from a list
df_2 = pd.DataFrame(data=d_2) # Constructing DataFrame from a dict

print(df_1)
print(df_2)
```

```
   0                          # Header
0  1                          # First row
1  2                          # Second row
2  3                          # …

   header_1  header_2                    # Header
0         1         3                    # First row
1         2         4                    # Second row
```

Create a Pandas DataFrame based on the file ‘global_temp.txt’. Print out the database.

```
import pandas as pd

df = pd.read_table('global_temp.txt', sep= ' ')

print(df)
```

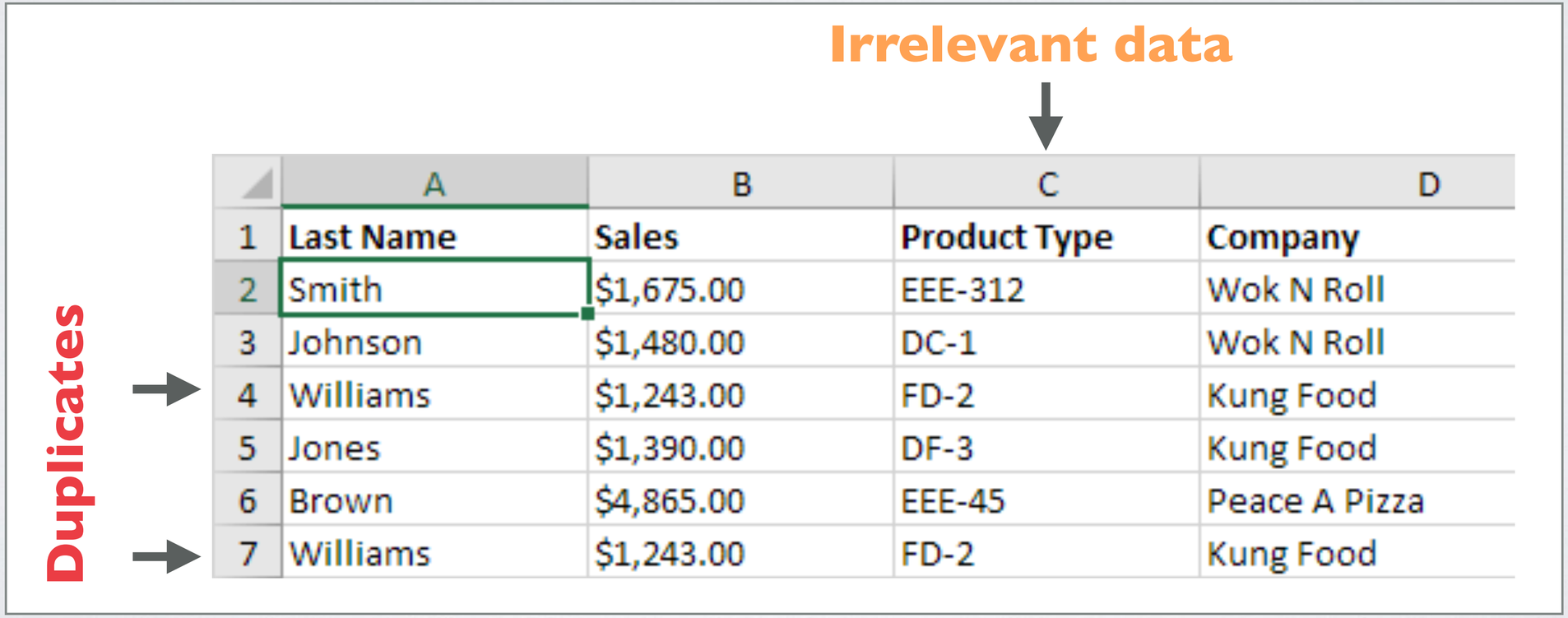
MANAGE DATA

- Unwanted Observations
- Remove Outliers
- Fix Structural Errors
- Handle Missing Data
- Filtering and Sorting Data

The **least enjoyable** part of data science. **Spending the most time** doing it.

Unwanted observations

- **Duplicates**: Frequently arise during collection, such as combining different datasets.
- **Irrelevant data**: They don't actually fit the specific problem.



	A	B	C	D
1	**Last Name**	**Sales**	**Product Type**	**Company**
2	Smith	$1,675.00	EEE-312	Wok N Roll
3	Johnson	$1,480.00	DC-1	Wok N Roll
4	Williams	$1,243.00	FD-2	Kung Food
5	Jones	$1,390.00	DF-3	Kung Food
6	Brown	$4,865.00	EEE-45	Peace A Pizza
7	Williams	$1,243.00	FD-2	Kung Food

- Removing identical rows

```
df = df.drop_duplicates(subset='Last Name', keep='first')
```

  - Parameter **subset**: It takes a column or list of column label. After passing columns, it will consider them only for duplicates.

  - Parameter **keep**: It could be 'first', 'last' or 'False' (it consider all of the same values as duplicates).

	A	B	C	D
1	**Last Name**	**Sales**	**Product Type**	**Company**
2	Smith	$1,675.00	EEE-312	Wok N Roll
3	Johnson	$1,480.00	DC-1	Wok N Roll
4	Williams	$1,064.00	EE-2	Peace A Pizza
5	Jones	$1,390.00	DF-3	Kung Food
6	Brown	$4,865.00	EEE-45	Peace A Pizza
7	Williams	$1,243.00	FD-2	Kung Food

- Dropping irrelevant columns

```
df = df.drop(["Sales"], axis = 1)
```

- Dropping irrelevant rows

```
df = df.drop(["Johnson", "Smith"])
```

- Dropping rows containing NaN

```
df = df.dropna()
```

	A	B	C	D
1	**Last Name**	**Sales**	**Product Type**	**Company**
2	Smith	$1,675.00	EEE-312	Wok N Roll
3	Johnson	$1,480.00	DC-1	Wok N Roll
4	Williams	$1,064.00	EE-2	Peace A Pizza
5	Jones	$1,390.00	DF-3	Kung Food
6	Brown	$4,865.00	EEE-45	Peace A Pizza
7	Williams	$1,243.00	FD-2	Kung Food

Handle Missing Data:

- Dropping observations:
  - Replace the entry with value "NaN".

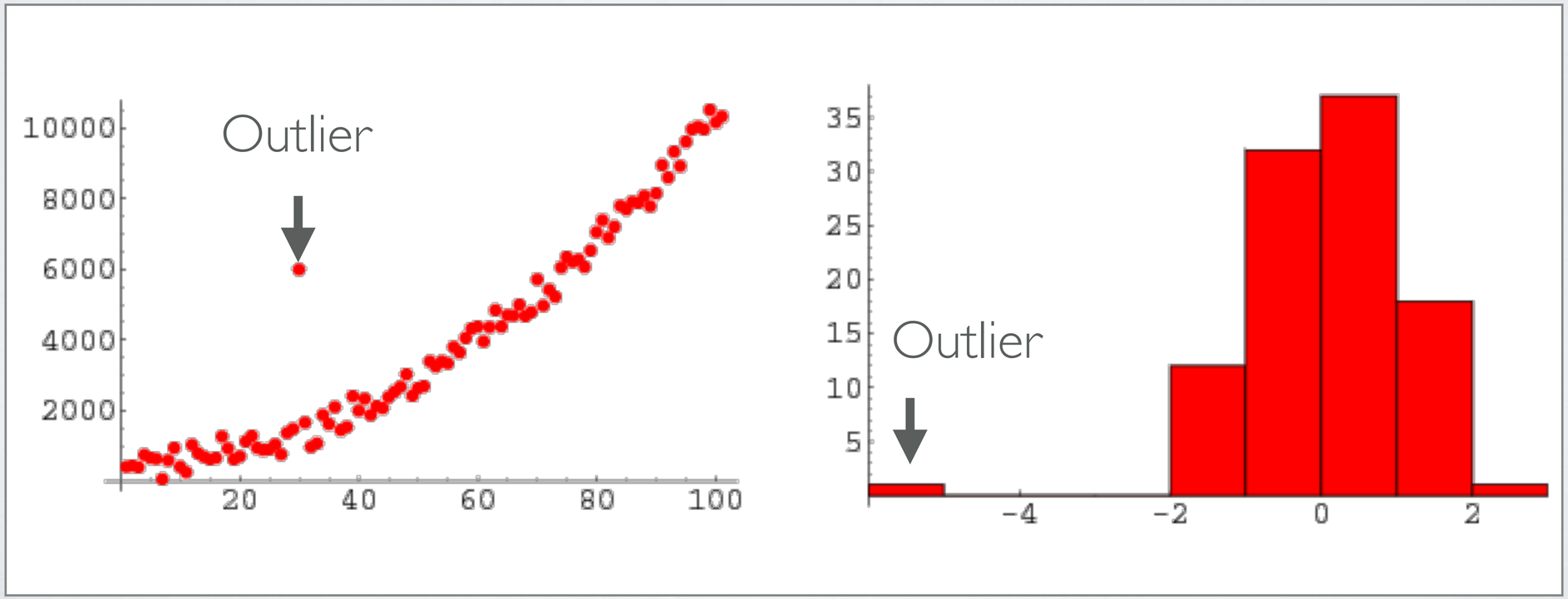
- By using the method .replace():

```
df = df.replace(1, 31)        # Replace 1 with 31
df = df.replace(1, np.nan)    # Replace 1 with "np.nan"
```

- Remove the whole row where information is missing.
  - Warning! Missing data may be informative itself.
- Input missing values:
  - The gap will be filled with artificial data (mean, median, std), having similar properties then real observation. The added value will not be scientifically valid, no matter how sophisticated your filling method is.

Unwanted Outliers:

- An observation that lies outside the overall pattern of a distribution.
- Common causes: human, measurement, experimental errors.
- Outliers are innocent until proven guilty.



Finding outliers with method .describe()

- The core statistics about a particular column can be studied by the **describe()** method. The method returns the following:

  A. For **numeric** columns: the value count, mean, standard deviation, minimum, maximum, and 25th, 50th, and 75th quantiles for the data in a column.

  B. For **string** columns: the number of unique entries, the most frequently occurring value ('top'), and the number of times the top value occurs ('freq')

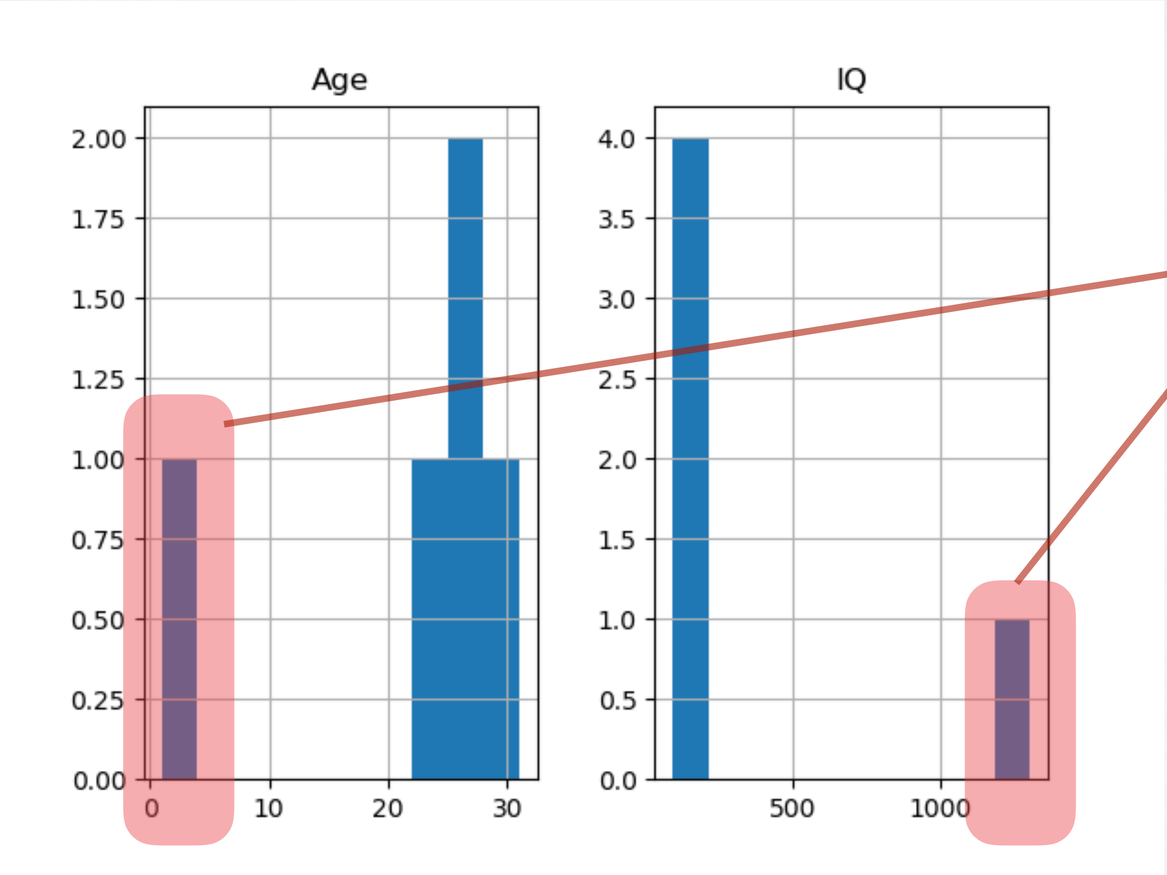
```
import pandas as pd

d = {“Name”: [”Alisa”,”Bobby”,”Cat”,”Madonna”,”Rocky”],
     “Age”:  [1,27,25,24,31],
     “IQ”:   [100, 120, 95, 1300, 101]}

df = pd.DataFrame(d)
print(df.describe())
```

- Investigate the output and look for potential outliers.

```
              Age           IQ
count    5.000000     5.000000
mean    21.600000   343.200000       # Suspicious
std     11.823705   534.952054       # Suspicious
min      1.000000    95.000000       # Outlier: Too young
25%     24.000000   100.000000
50%     25.000000   101.000000
75%     27.000000   120.000000
max     31.000000  1300.000000       # Outlier: Too smart
```

Finding Outliers with Histograms

```
df.hist(['Age','IQ])
plt.show() #It may be necessary after importing matplotlib
```



Unexpected behaviour, i.e., far from general population, nonsense value, wrong distribution shape, etc …

Removing Outliers from the data

Remove the outlier by dropping the row, replacing its value one by one or introducing a threshold.

- Dropping column or row can be done by the method .drop() as discussed before.

- Replace the outlier with another value

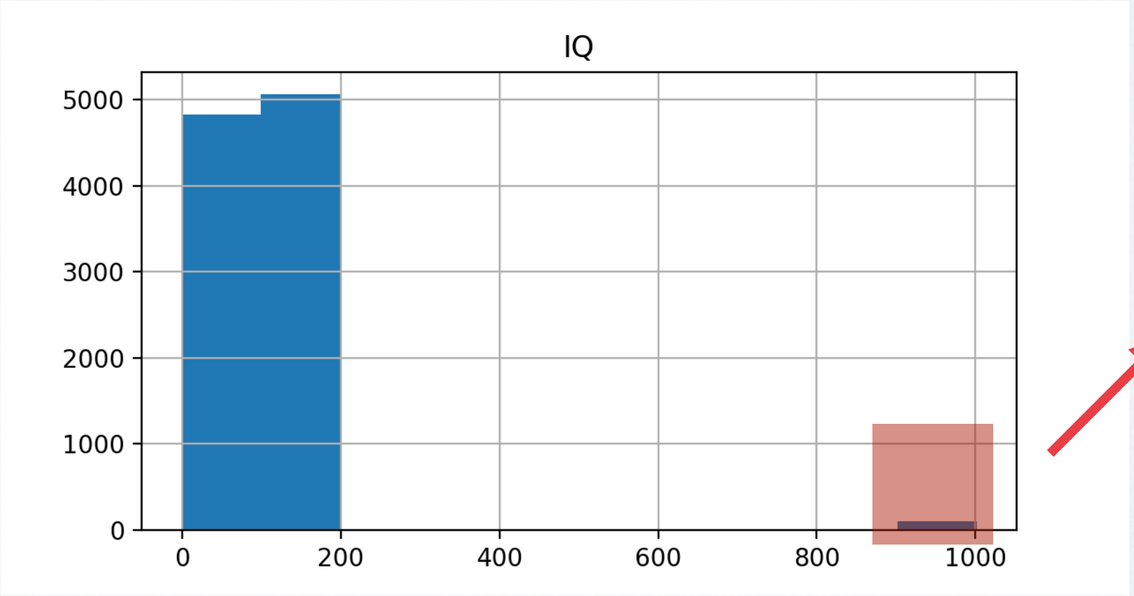
```
df = df.replace(1, 31)        # Replace 1 with 31
df = df.replace(1, np.nan)    # Replace 1 with “np.nan”
```

- Introducing a threshold and remove the outlier:

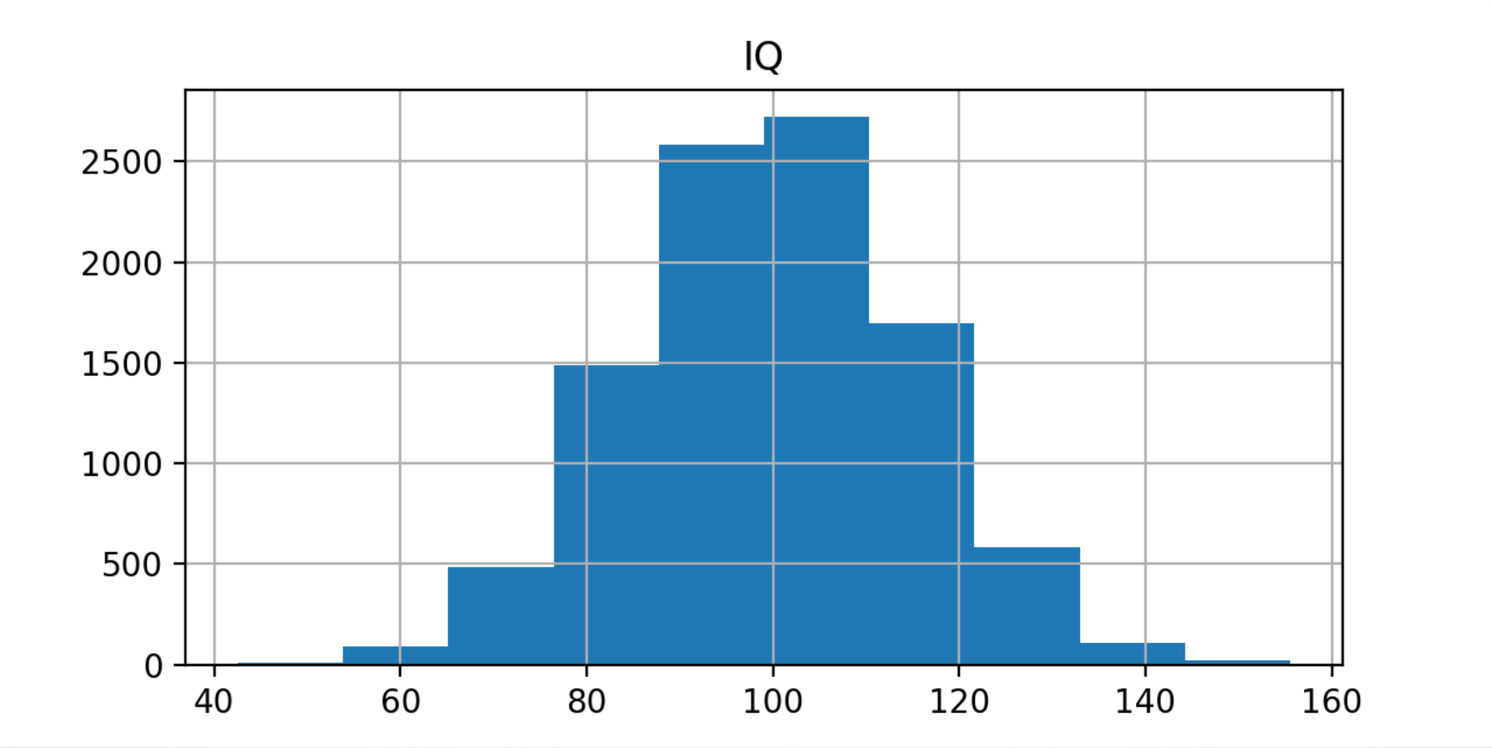
```
df = df.mask(df <= 1, 10) # Replace every element<=1 with 10
```

- Read the database, named “iq_scores.csv”.
- Drop the insignificant rows: UID and LOCATION_ID
- Drop the duplicated lines.
- Errors are marked by the number -1. Remove them.
- Investigate the histogram of the variable IQ. Search for unexpected behaviour and remove the outliers is there are any.
- Plot the histogram IQ without any outliers or errors.

```
import pandas as pd

# Read the database
df = pd.read_csv("iq_scores.csv")

#Drop duplicates
df = df.drop_duplicates(subset='UID', keep='first')

# Drop irrelevant columns
df = df.drop(['UID','LOCATION_ID'], axis=1)

# Investigate the data
df.hist('IQ')
```



```
import numpy as np

# Remove known errors/missing data
df = df.replace(-1, np.nan).dropna()

# Remove the outlier
df.mask(df['IQ'] > 900)

# Investigate the data
df.hist('IQ')
```



Filtering Data

- Data segmentation: Limits of computation, e.g. insufficient memory or CPU performance.
- Filtering by data attributes, e.g. separate the data by time.
- Use the method **.iloc()**.

Sorting Data

- Sorting by some dimension alphabetically or numerically, e.g. sorting by time or date.
- Ascending or Descending.
- Use the method **.sort_values()**.

Filtering Data by Using iloc()

- Select one element of the DataFrame

```
df.iloc[row, col])
df.iloc[1,2] #Out: 8
```

	A	B	C
0	1	4	7
1	2	5	8
2	3	6	9

- Slicing through dimensions:

```
df.iloc[row1: row2, col1: col2])
df.iloc[0: 2, 2: 3])

Output:
   C
0  7
1  8
```

- Select a column of the DataFrame

```
print(df.iloc[:,1])     # Output: 4, 5, 6
```

- Select a row of the DataFrame

```
print(df.iloc[2,:])     # Output: 3, 6, 9
```

- First 2 rows:

```
print(df.iloc[0:2,:])
```

- Remove the last row

```
print(df.iloc[:2,:])
```

- And so on…

CLEAN DATA

- **Normalisation** typically means rescales the values into a range of [0,1].
- In most cases, when you normalise data you eliminate the units of measurement for data, enabling you to more easily compare data from different places.

$$x = [1,43,65,23,4,57,87,45,45,23]$$

$$x_{new} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

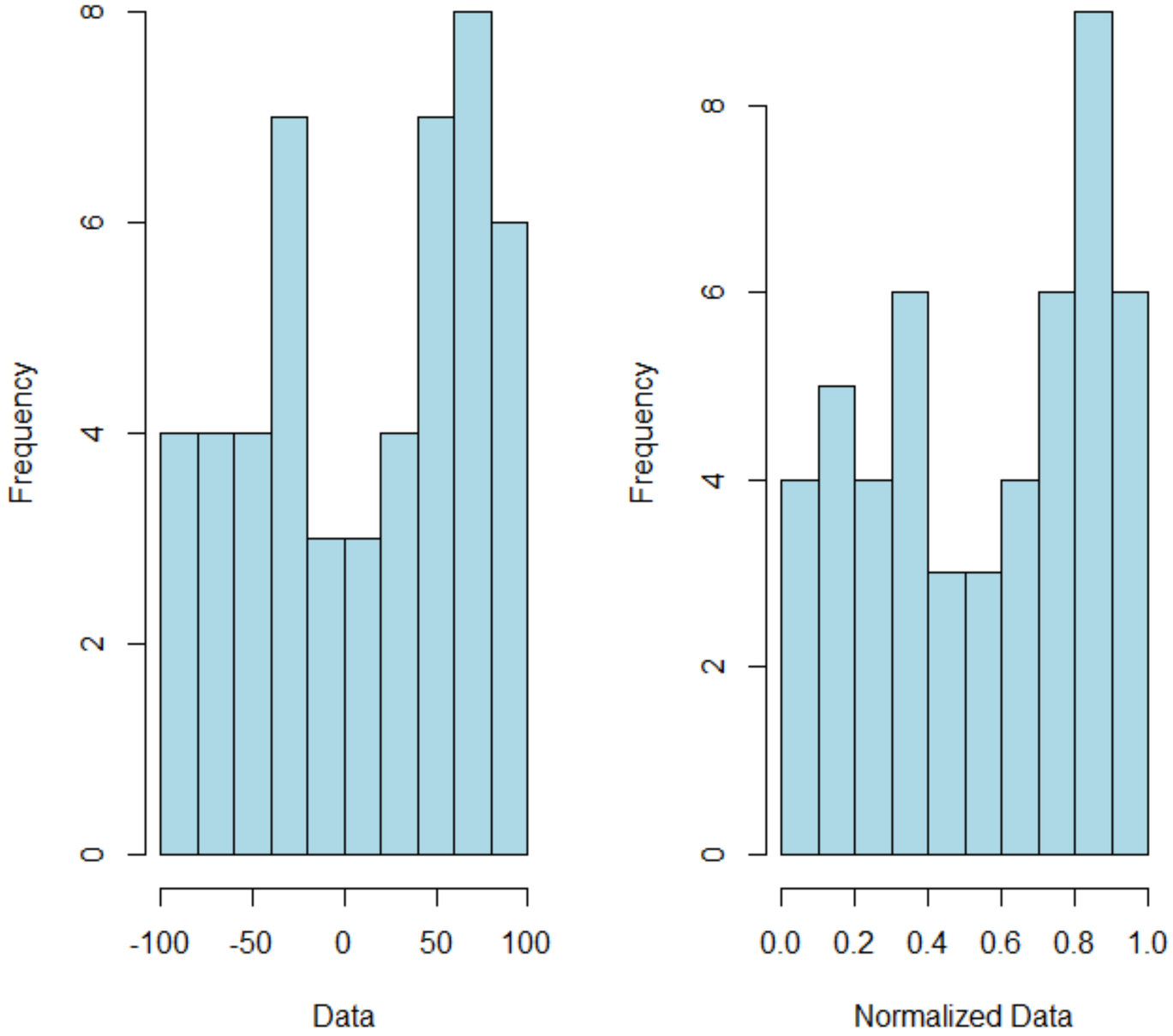
$$x_{min} = 1$$
$$x_{max} = 87$$

$$x_{new} = [0,0.48,0.74,0.25,0.03,0.65,1,0.51,0.51,0.25]$$

Normalising a **Numpy array** or Normalising a column of **Pandas DataFrame** (normalise column named “score” in Dataframe “df”):

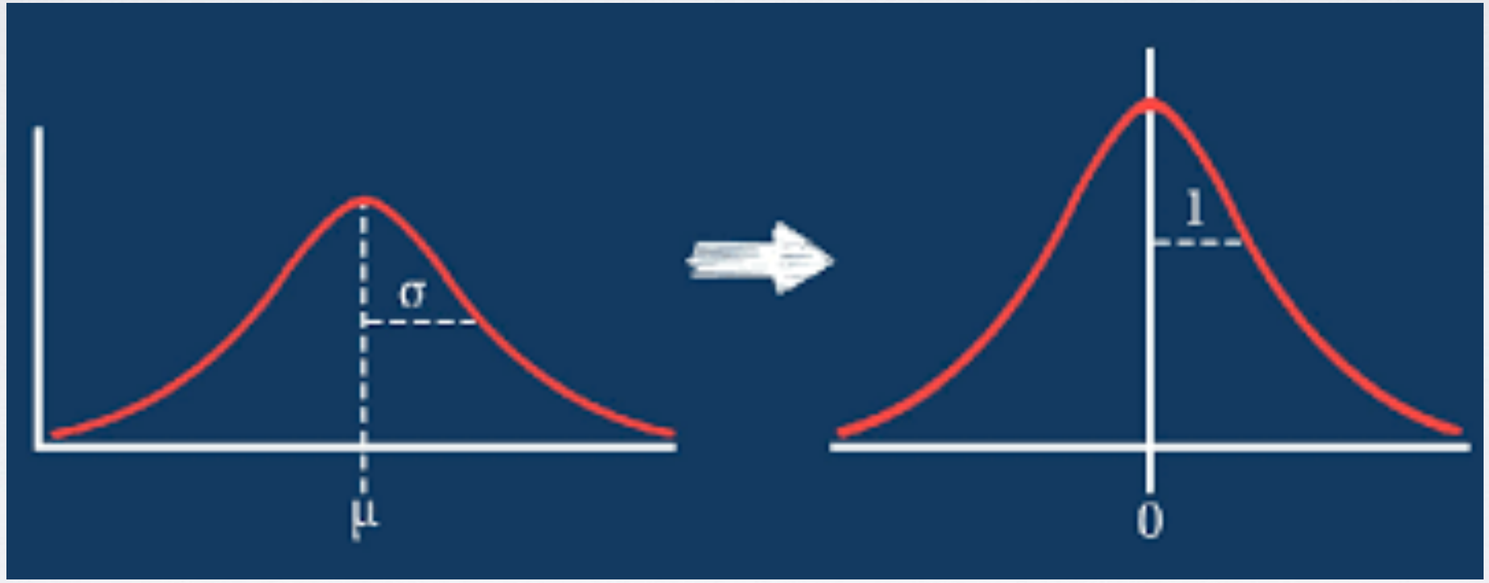
```
import numpy as np
import pandas as pd

raw_data= [1,43,65,23,4,57,87,45,45,23])

x = np.array(raw_data)
x_new = (x - x.min()) / (x.max() - x.min())

df = pd.DataFrame({‘score’: raw_data})
df[‘score’] = (df[‘score’] - df[‘score’].min()) /
              (df[‘score’].max() - df[‘score’].min())
```

Data normalisation example



- **Data Standardisation:**
  Standardisation typically means rescales data to have a mean of 0 and a standard deviation of 1 (unit variance).

$$x_{new} = \frac{x - \mu}{\sigma}$$



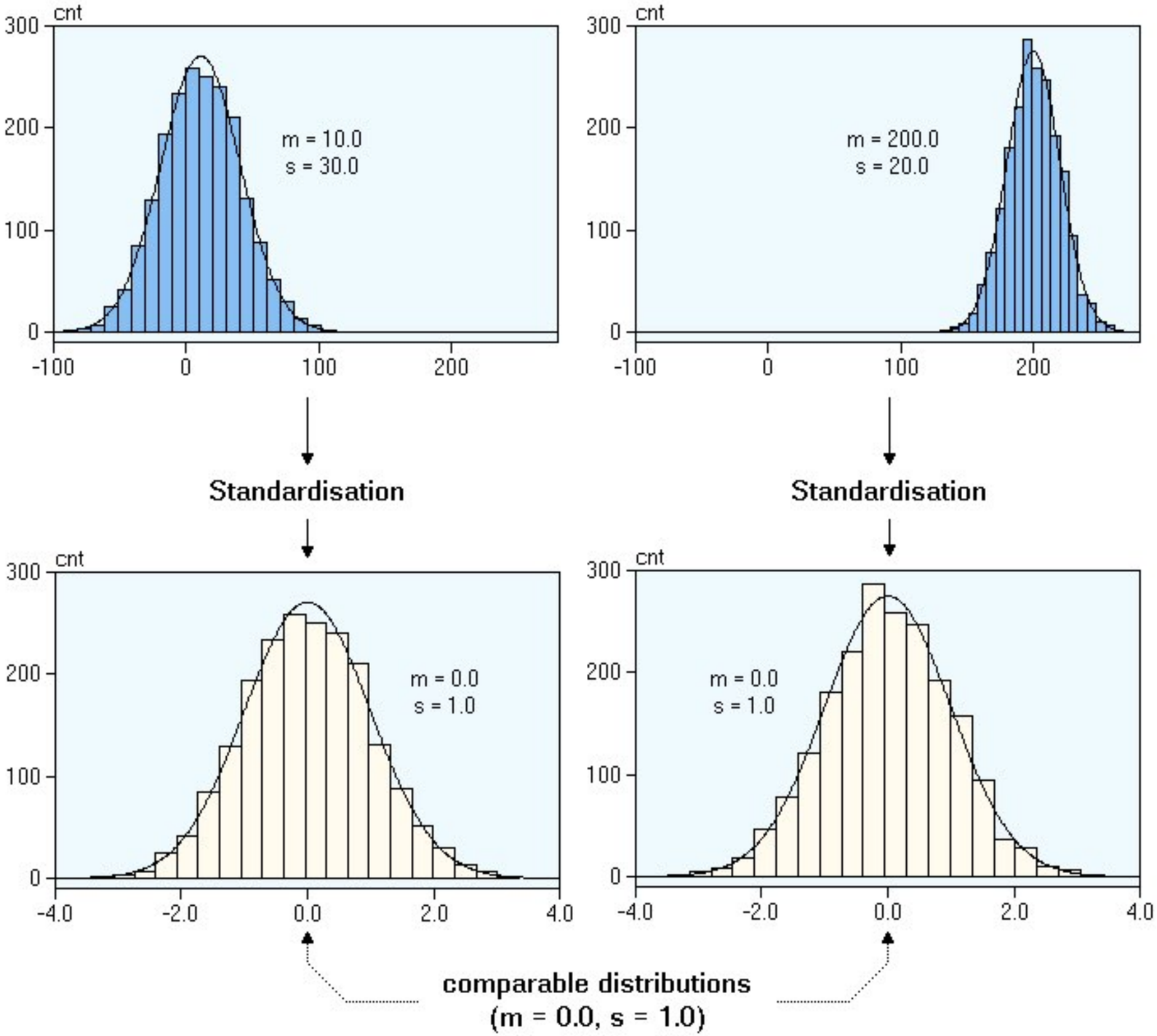
$$x = [1,43,65,23,4,57,87,45,45,23] \qquad \mu = 39.3 \qquad x_{max} = 87$$

$$\downarrow$$

$$x_{new} = [-1.49,0.14,1.00,-0.63,-1.37,0.69,1.86,0.22,0.22,-0.63]$$

Standardising a **Numpy array** or a column of **Pandas DataFrame** (normalise column named “sc” in Dataframe “df”):

```
import numpy as np
import pandas as pd

raw_data= [1,43,65,23,4,57,87,45,45,23])

x = np.array(raw_data)
x_new = (x - x.mean()) / x.std()

df = pd.DataFrame({‘sc’: raw_data})
df[‘sc’]=(df[‘sc’]-df[‘sc’].mean())/
df[‘sc’].std()
```


cnt
300
200
100
0
m = 10.0
s = 30.0
-100
0
100
200
cnt
300
200
100
0
m = 200.0
s = 20.0
-100
0
100
200
Standardisation
Standardisation
cnt
300
200
100
0
m = 0.0
s = 1.0
-4.0
-2.0
0.0
2.0
4.0
cnt
300
200
100
0
m = 0.0
s = 1.0
-4.0
-2.0
0.0
2.0
4.0
comparable distributions
(m = 0.0, s = 1.0)

EXPLORATORY DATA ANALYSIS

Aim:

An approach to understanding the entire dataset.

Objectives:

1) Detection of mistakes. 2) Checking assumptions. 3) Detecting relationships between variables. 4) Start to play with the data!

Tools:

EDA typically relies heavily on visualising the data to assess patterns and identify data characteristics that the analyst would not otherwise know to look for.

Example database: Airline safety

Aim: Should Travelers Avoid Flying Airlines That Have Had Crashes in the Past?

Objectives: We are going to explore the airline safety database between 1985-2014.

Tools: Univariate and multivariate data visualisation and simple statistical tools.

Example database: Airline safety

The data is stored in **csv format** and it appears to be **structured** (no missing data, no structural error).

The data contains the following information:

- **airline**: The new of the airline company.
- **avail_seat**: Passenger capacity. Available seat per km per week.
- **incidents_85_99**: Incidents between 1985 and 1999.
- **fatal_accidents_85_99**: Fatal accidents between 1985 and 1999.
- **fatalities_85_99**: Fatalities be between 1985 and 1999.
- **incidents_00_14**: Incidents between 2000 and 2014.
- **fatal_accidents_00_14**: Fatal accidents between 2000 and 2014.
- **fatalities_00_14**: Fatalities be between 2000 and 2014.

```
df = pd.read_csv('airline-safety_csv.csv')
```

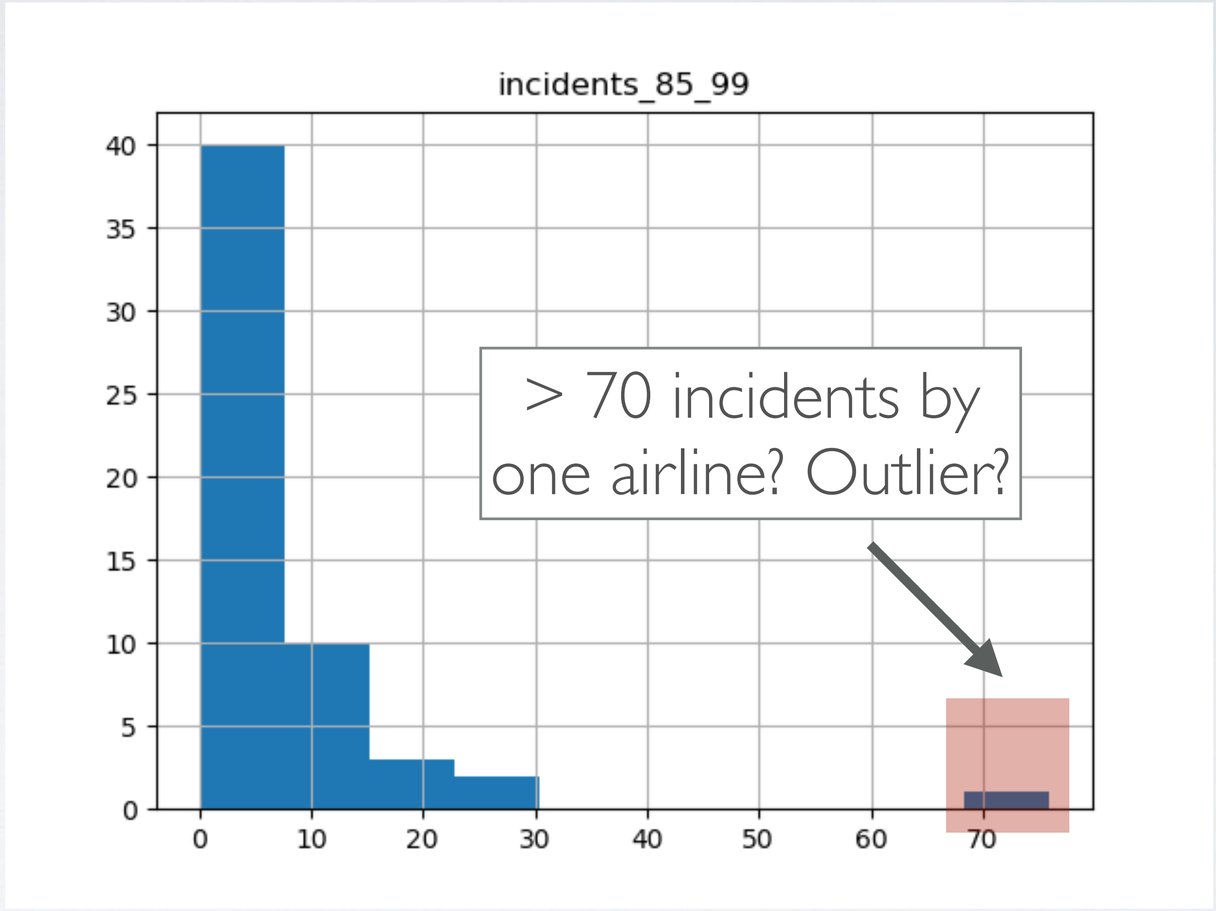
airline	avail_seat_km	incidents_85_99	fatal_accidents_85_99	…
aeroflot*	1197672318	76	14	…
aerolineas arg.	385803648	6	0	…
aeromexico*	596871813	3	1	…
…	…	…	…	…

- Standardisation: Inconveniently big numbers
- What is the meaning of these numbers?

airline	avail_seat_km	incidents_85_99	fatal_accidents_85_99	…
aeroflot*	-0.12	76	14	…
aerolineas arg.	-0.68	6	0	…
aeromexico*	-0.53	3	1	…
…	…	…	…	…

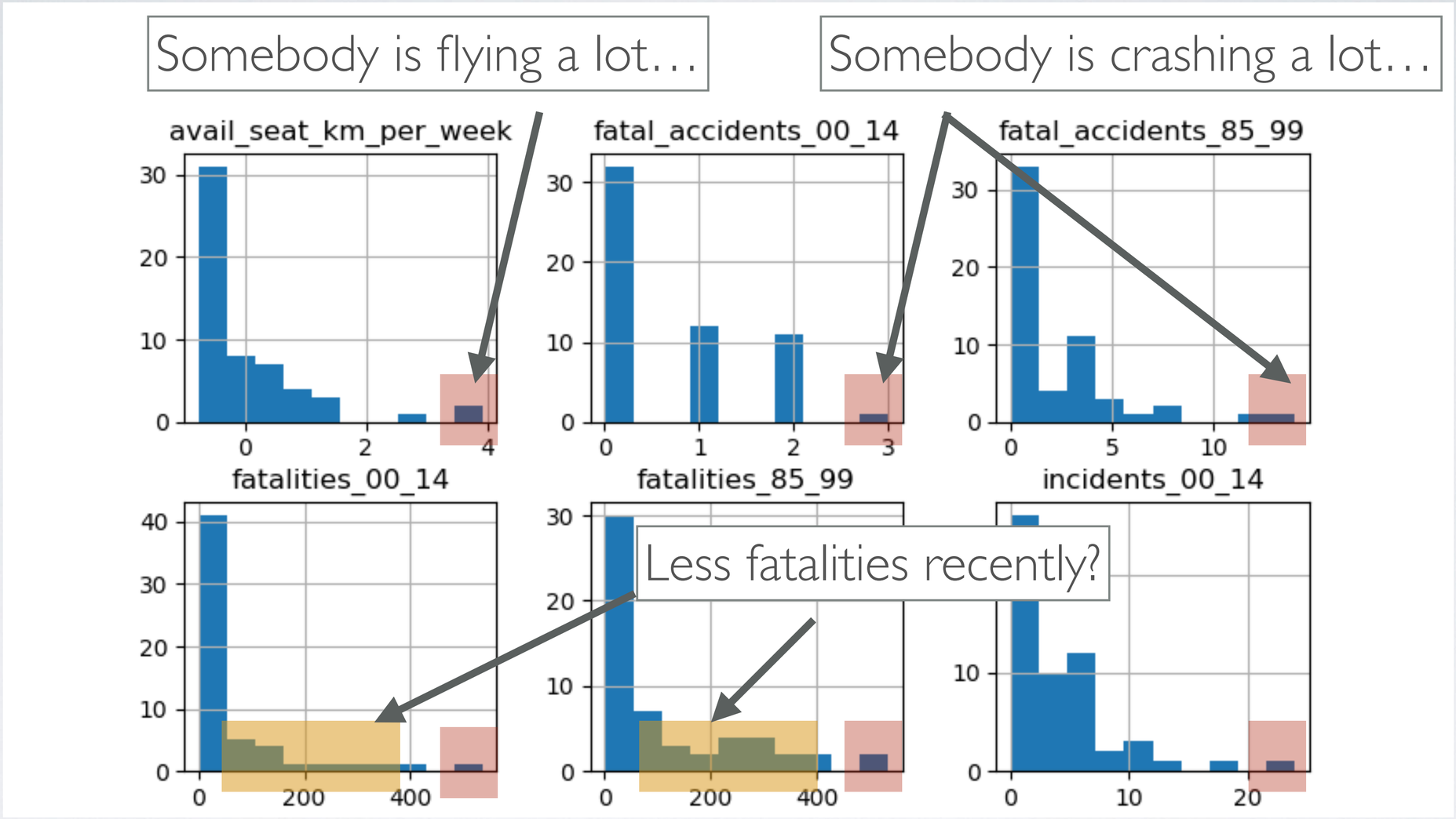
Univariate visualisation

- For each field in the raw dataset.
- Is it the expected distribution?
- Are there any outliers?




```
df.hist('incidents_85_99')
```

Univariate visualisation




```
df.hist()
```

The investigation starts

Somebody is flying a lot…

Connection?

Somebody is crashing a lot…

Is my data reliable?

> 70 incidents by one airline? Outlier?

Is my data reliable?

Is it safer to fly today than before?

So on…

Insights

Questions

Download and load the airline safety database. Standardise the column “avail_seat” and find the airline who had more than 70 incidents between 1985 and 1999.

```
import pandas as pd

# Read the database
df = pd.read_csv('airline-safety_csv.csv')

# Filter out the airlines if incidents < 70
dfnan = df.mask(df["incidents_85_99"] < 70)

# Drop the irrelevant rows
df_filtered = dfnan.dropna()

# Print the results
print(df_filtered)
```

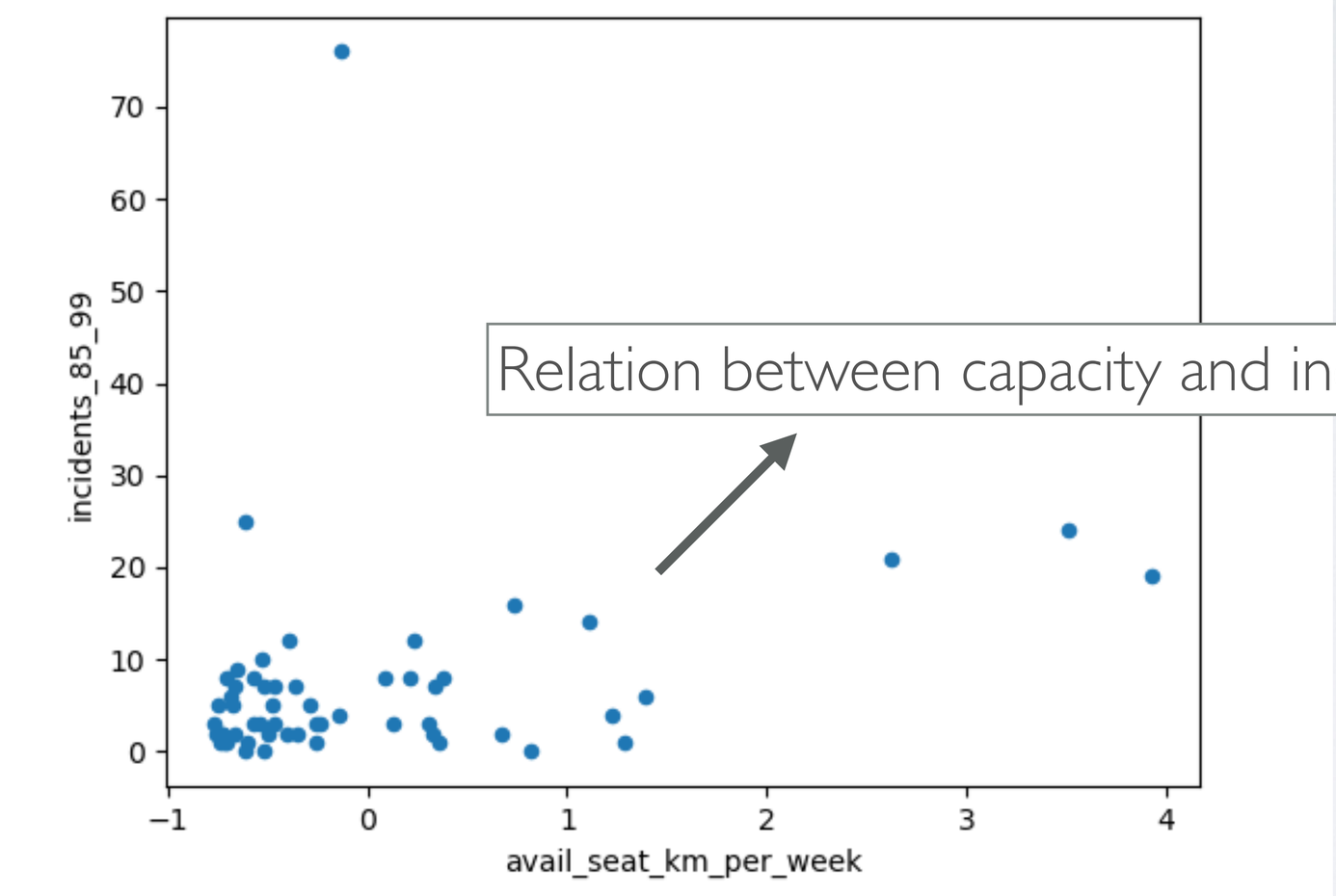
	airline	avail_seat_km_per_week	incidents_85_99	fatal_accidents_85_99
1	aeroflot*	-0.127583	76.0	14.0

Flying less than the average.

High number of incidents.

Multivariate visualisations

- Is there any relationship between the investigated data subsets?
- Is the relationship significant statistically or interesting scientifically?




```
df.plot.scatter('avail_seat_km_per_week', 'incidents_85_99')
```

- Use **corr()** function to find the correlation among the columns in the dataframe using 'Pearson' method.
- **Correlations are never lower than -1.** A correlation of -1 indicates that the data points in a scatter plot lie exactly on a straight descending line.
- A **correlation of 0** means that two variables don't have any linear relation whatsoever. However, some non linear relation may exist between the two variables.
- **Correlation coefficients are never higher than 1.** A correlation coefficient of 1 means that two variables are *perfectly* positively linearly related.

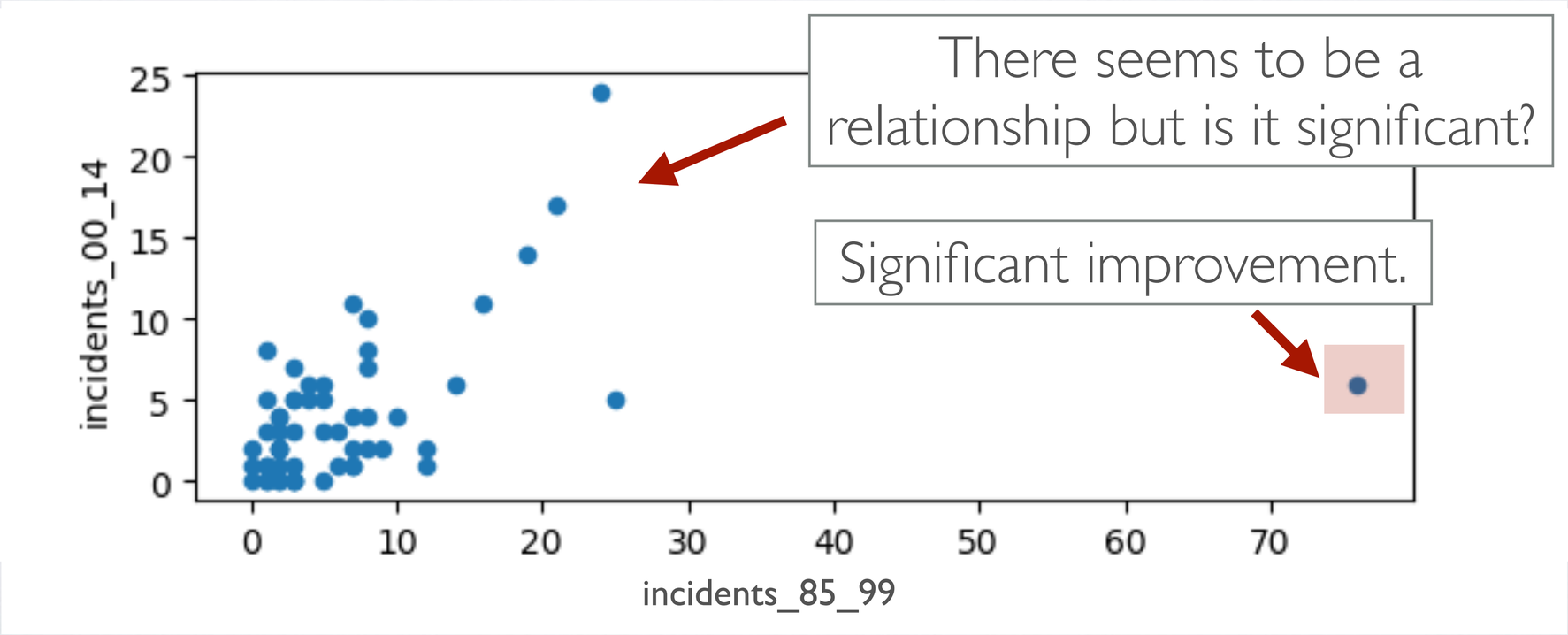
	avail_seat_km_per_week	incidents_85_99
avail_seat_km_per_week	1.000000	0.279538
incidents_85_99	0.279538	1.000000
fatal_accidents_85_99	0.468300	0.856991
incidents_00_14	0.725917	0.403009

High correlation coefficient and interesting.

High correlation coefficient but not scientifically interesting.

Investigate the relationship between the variables “incidents_85_99” and “incidents_85_99”. Use scatter plot the visualise the results.

```
import pandas as pd

# Read the database
df = pd.read_csv(‘airline-safety_csv.csv’)

df.plot.scatter('incidents_85_99',
'incidents_00_14')
```



DATA ANALYSIS

Turn insight and ideas into scientifically valid results.

Use the most promising finding.

↓

Perform in-depth analysis.

↓

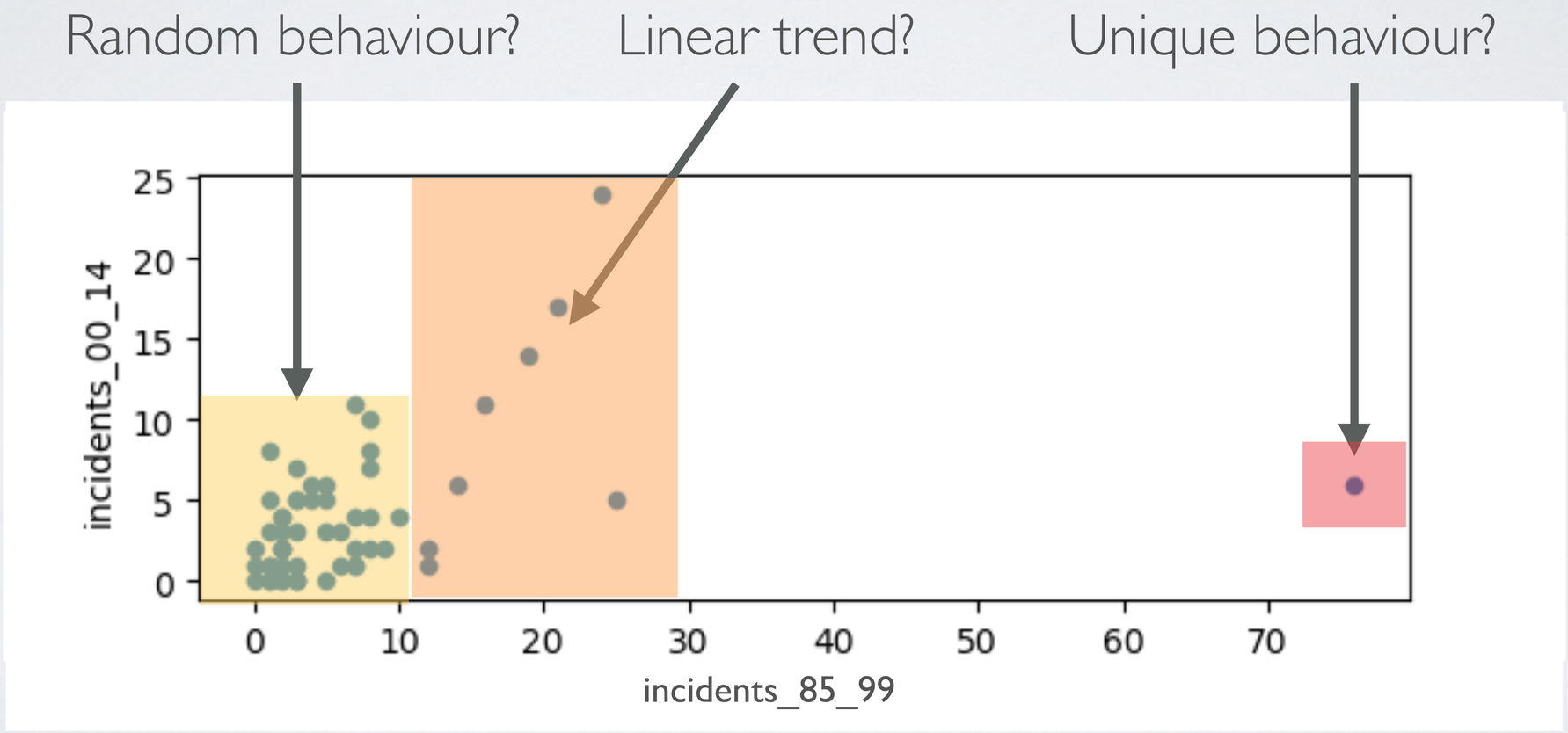
Check your results.

↓

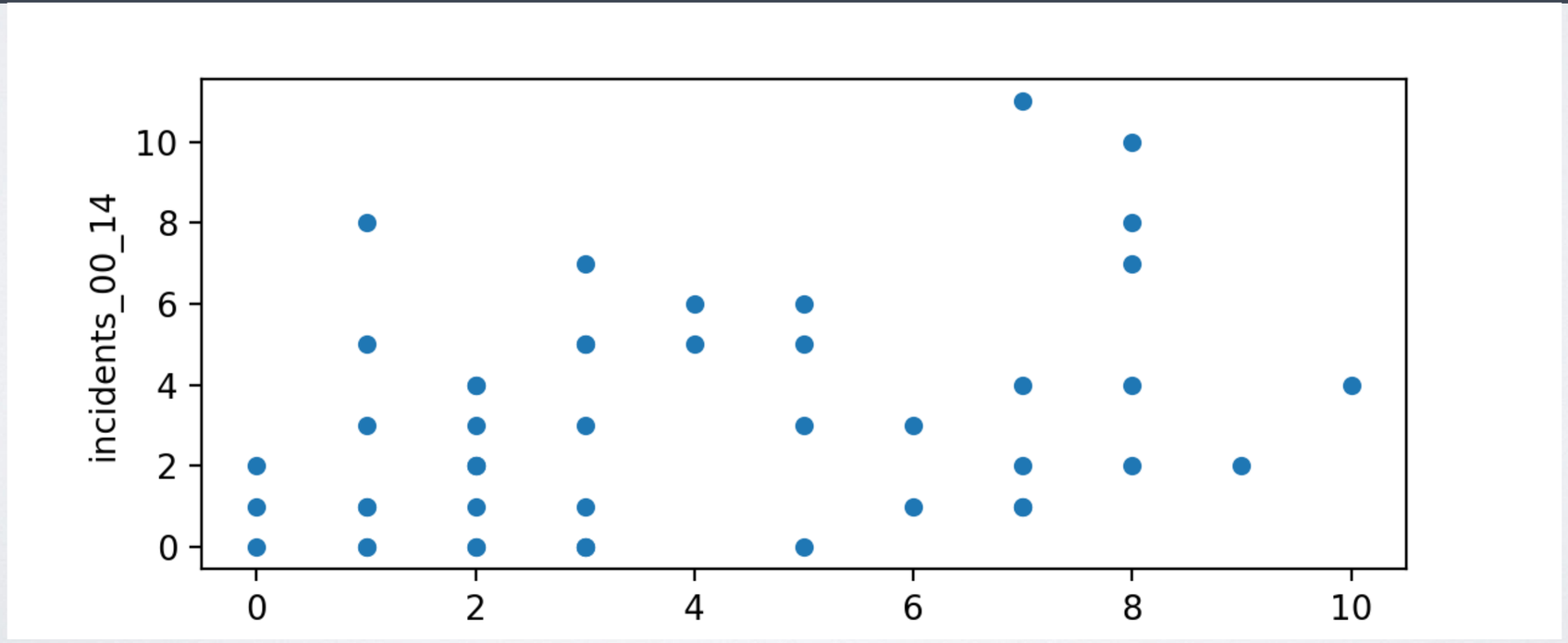
Prove your results.

Continue to investigate the details!

Different behaviours seem to be mixed in this statistics. If possible try to separate the data. Separate manually or by using an algorithm



```
# Filter incidents < 10
df_l = df.mask(df["incidents_85_99"] > 10).dropna()

# Output: air canada, air india, air new zealand …
print(df_l['airline'])

# Plot the results
df_l.plot.scatter('incidents_85_99', 'incidents_00_14')

#Check the correlation: Output: 0.36
df_l['incidents_85_99'].corr(df_l['incidents_00_14'])
```



```
df_m = df.mask((df["incidents_85_99"] < 10) |
               (df["incidents_85_99"] > 70)).dropna()

# Output: air france, china airlines, delta …
print(df_m[‘airline’])

# Plot the results
df_m.plot.scatter(‘incidents_85_99’, ‘incidents_00_14')

#Check the correlation: Output: 0.68
df_m[‘incidents_85_99'].corr(df_m['incidents_00_14'])
```