Data science

Chapter 5 – Intro Pandas

2025-2026

Overview

1. Introduction
2. Data structures in pandas: Series and DataFrame
3. Extracting and assigning data
4. Reading/writing data (CSV, JSON and SQLite)
5. Basic DataFrame operations
6. Missing values
7. Conditional selections, sorting, grouping & counting
8. Cleaning a dataset (using an example)

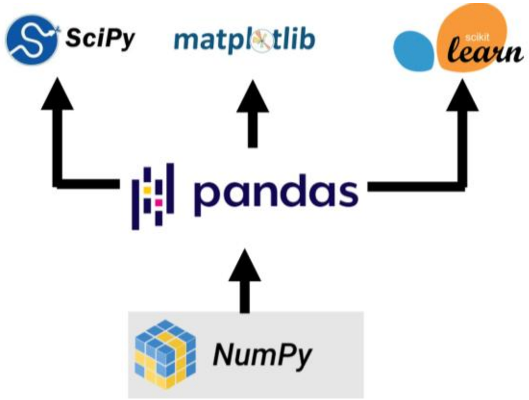
pandas

- The most popular Python library for data analysis
- Through pandas, you get acquainted with your data
- Pandas helps you
  - calculate statistics and answer questions about the data, like
    - What's the average, median, max, or min of each column?
    - Does column A correlate to column B?
    - What does the distribution of data in column C look like?
  - clean the data by doing things like removing missing values and filtering rows or columns
  - visualize the data with help from Matplotlib (plot bars, lines, histograms, bubbles, …)
  - store the cleaned, transformed data back into a CSV, other file or database

pandas is derived from the term “panel data” which are data sets that include observations over multiple time periods for the same individuals [Wikipedia]

pandas

- The pandas library is a central component of the data science toolkit, but it is used in conjunction with other libraries.
- Pandas is built on top of the **NumPy** package, meaning a lot of the structure of NumPy is used or replicated in pandas.
- Data in pandas is often used to feed statistical analysis in **SciPy**, plotting functions in **Matplotlib** and **SeaBorn**, and machine learning algorithms in **Scikit-learn**.



Install and import pandas

- Run the install command in a terminal window:

```
pip install pandas
```

- In Jupyter you must use a preceding ! to force the code cell to be executed in a terminal window:

```
!pip install pandas
```

- To import pandas we usually import it with a shorter name since it's used so much:

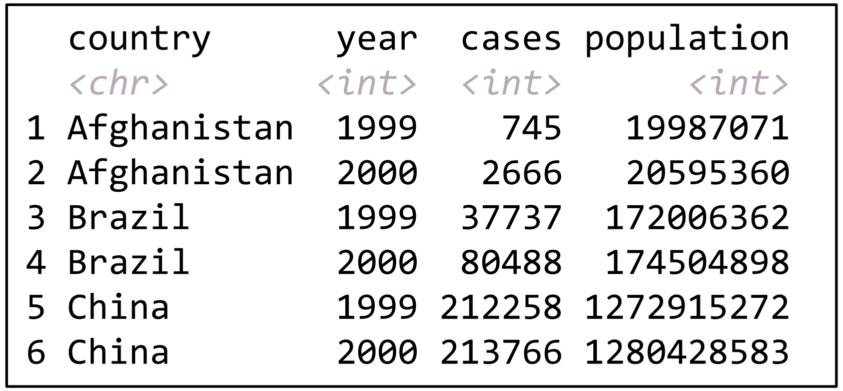
```
import pandas as pd
```

Data science

About dirty data

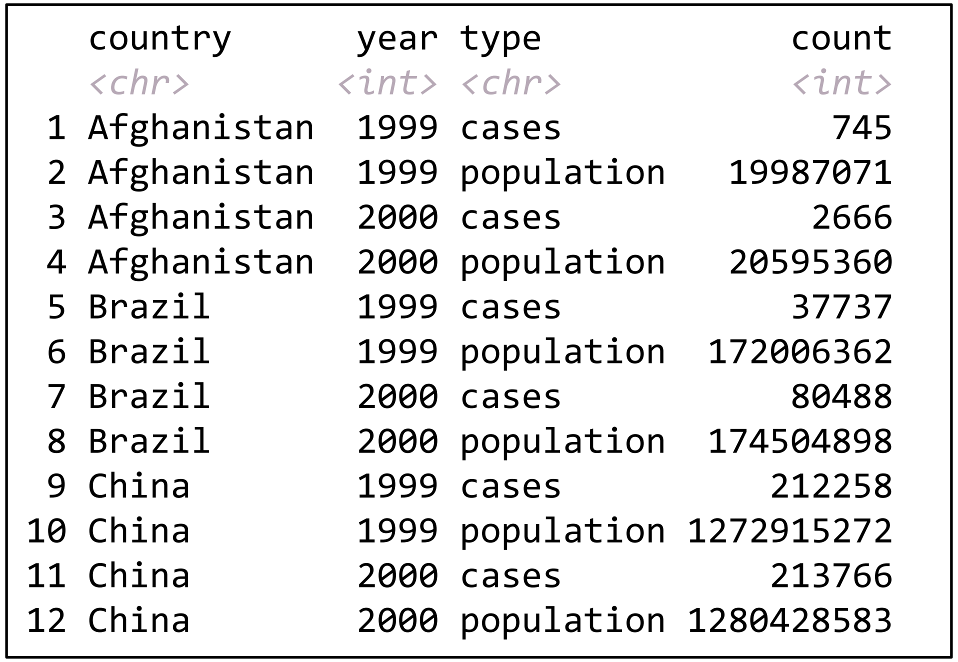
Same data, different formats

1



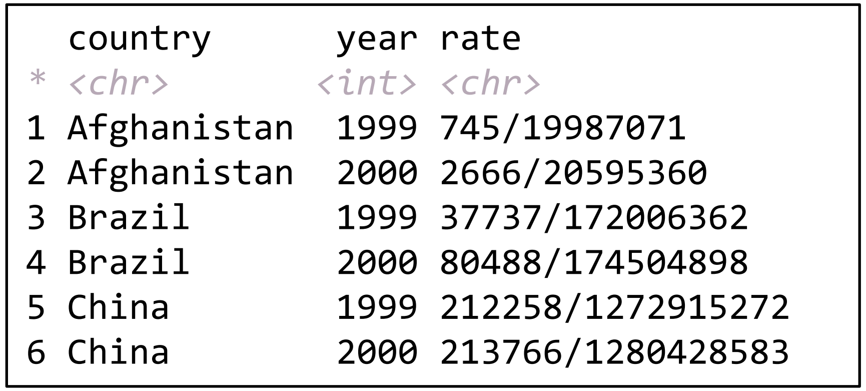
```
  country      year  cases population
  <chr>       <int>  <int>      <int>
1 Afghanistan  1999    745   19987071
2 Afghanistan  2000   2666   20595360
3 Brazil       1999  37737  172006362
4 Brazil       2000  80488  174504898
5 China        1999 212258 1272915272
6 China        2000 213766 1280428583
```

2



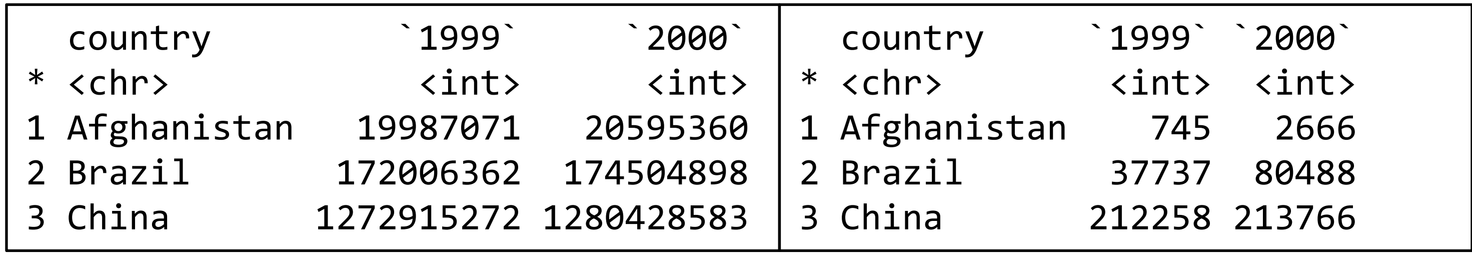
```
   country      year type            count
   <chr>       <int> <chr>           <int>
 1 Afghanistan  1999 cases             745
 2 Afghanistan  1999 population   19987071
 3 Afghanistan  2000 cases            2666
 4 Afghanistan  2000 population   20595360
 5 Brazil       1999 cases           37737
 6 Brazil       1999 population  172006362
 7 Brazil       2000 cases           80488
 8 Brazil       2000 population  174504898
 9 China        1999 cases          212258
10 China        1999 population 1272915272
11 China        2000 cases          213766
12 China        2000 population 1280428583
```

3



```
  country      year rate
* <chr>       <int> <chr>
1 Afghanistan  1999 745/19987071
2 Afghanistan  2000 2666/20595360
3 Brazil       1999 37737/172006362
4 Brazil       2000 80488/174504898
5 China        1999 212258/1272915272
6 China        2000 213766/1280428583
```

4

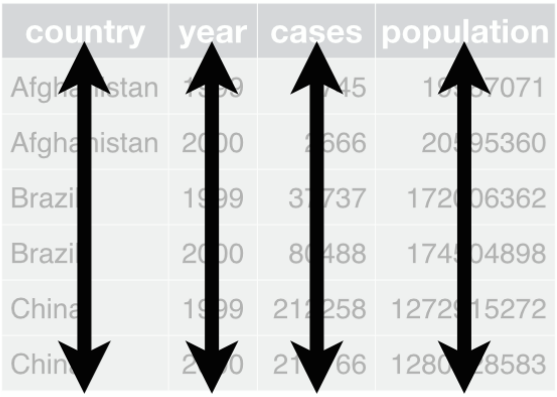


```
  country         `1999`     `2000`
* <chr>            <int>      <int>
1 Afghanistan   19987071   20595360
2 Brazil       172006362  174504898
3 China       1272915272 1280428583
```

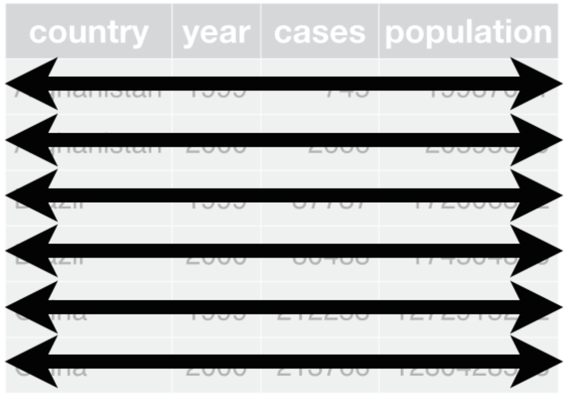
```
  country     `1999` `2000`
* <chr>        <int>  <int>
1 Afghanistan    745   2666
2 Brazil       37737  80488
3 China       212258 213766
```

3 rules for data files

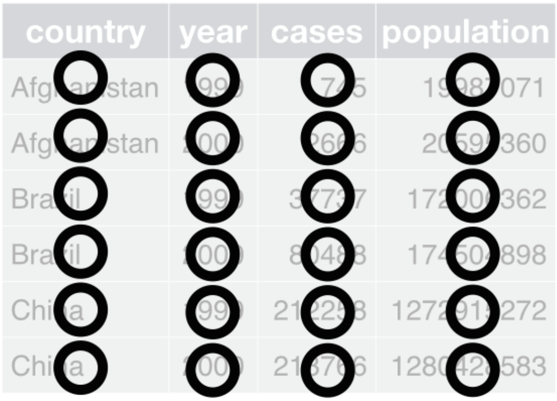
- Each variable must have its own column.
- Each observation must have its own row.
- Each value must have its own cell.



variables



observations



values

Does each variable have its own column?

1

```
  country      year  cases population
  <chr>       <int>  <int>      <int>
1 Afghanistan  1999    745   19987071
2 Afghanistan  2000   2666   20595360
3 Brazil       1999  37737  172006362
4 Brazil       2000  80488  174504898
5 China        1999 212258 1272915272
6 China        2000 213766 1280428583
```

2

```
   country      year type            count
   <chr>       <int> <chr>           <int>
 1 Afghanistan  1999 cases             745
 2 Afghanistan  1999 population   19987071
 3 Afghanistan  2000 cases            2666
 4 Afghanistan  2000 population   20595360
 5 Brazil       1999 cases           37737
 6 Brazil       1999 population  172006362
 7 Brazil       2000 cases           80488
 8 Brazil       2000 population  174504898
 9 China        1999 cases          212258
10 China        1999 population 1272915272
11 China        2000 cases          213766
12 China        2000 population 1280428583
```

3

```
  country      year rate
* <chr>       <int> <chr>
1 Afghanistan  1999 745/19987071
2 Afghanistan  2000 2666/20595360
3 Brazil       1999 37737/172006362
4 Brazil       2000 80488/174504898
5 China        1999 212258/1272915272
6 China        2000 213766/1280428583
```

4

```
  country          `1999`     `2000`
* <chr>             <int>      <int>
1 Afghanistan    19987071   20595360
2 Brazil        172006362  174504898
3 China        1272915272 1280428583
```

```
  country     `1999` `2000`
* <chr>        <int>  <int>
1 Afghanistan    745   2666
2 Brazil       37737  80488
3 China       212258 213766
```

Each variable must have its own column

```
  country      year  cases population
  <chr>       <int>  <int>      <int>
1 Afghanistan  1999    745   19987071
2 Afghanistan  2000   2666   20595360
3 Brazil       1999  37737  172006362
4 Brazil       2000  80488  174504898
5 China        1999 212258 1272915272
6 China        2000 213766 1280428583
```

```
   country      year type            count
   <chr>       <int> <chr>           <int>
 1 Afghanistan  1999 cases             745
 2 Afghanistan  1999 population   19987071
 3 Afghanistan  2000 cases            2666
 4 Afghanistan  2000 population   20595360
 5 Brazil       1999 cases           37737
 6 Brazil       1999 population  172006362
 7 Brazil       2000 cases           80488
 8 Brazil       2000 population  174504898
 9 China        1999 cases          212258
10 China        1999 population 1272915272
11 China        2000 cases          213766
12 China        2000 population 1280428583
```

type and count are no true variables (= characteristics of the observation object)

```
  country      year rate
* <chr>       <int> <chr>
1 Afghanistan  1999 745/19987071
2 Afghanistan  2000 2666/20595360
3 Brazil       1999 37737/172006362
4 Brazil       2000 80488/174504898
5 China        1999 212258/1272915272
6 China        2000 213766/1280428583
```

One column contains two variables

1999 and 2000 are not variables, they are values!

```
  country          `1999`     `2000`
* <chr>             <int>      <int>
1 Afghanistan    19987071   20595360
2 Brazil        172006362  174504898
3 China        1272915272 1280428583
```

```
  country     `1999` `2000`
* <chr>        <int>  <int>
1 Afghanistan    745   2666
2 Brazil       37737  80488
3 China       212258 213766
```

Does each observation have its own row?

1

```
  country      year  cases population
  <chr>       <int>  <int>      <int>
1 Afghanistan  1999    745   19987071
2 Afghanistan  2000   2666   20595360
3 Brazil       1999  37737  172006362
4 Brazil       2000  80488  174504898
5 China        1999 212258 1272915272
6 China        2000 213766 1280428583
```

2

```
   country      year type            count
   <chr>       <int> <chr>           <int>
 1 Afghanistan  1999 cases             745
 2 Afghanistan  1999 population   19987071
 3 Afghanistan  2000 cases            2666
 4 Afghanistan  2000 population   20595360
 5 Brazil       1999 cases           37737
 6 Brazil       1999 population  172006362
 7 Brazil       2000 cases           80488
 8 Brazil       2000 population  174504898
 9 China        1999 cases          212258
10 China        1999 population 1272915272
11 China        2000 cases          213766
12 China        2000 population 1280428583
```

3

```
  country      year rate
* <chr>       <int> <chr>
1 Afghanistan  1999 745/19987071
2 Afghanistan  2000 2666/20595360
3 Brazil       1999 37737/172006362
4 Brazil       2000 80488/174504898
5 China        1999 212258/1272915272
6 China        2000 213766/1280428583
```

4

```
  country          `1999`     `2000`
* <chr>             <int>      <int>
1 Afghanistan    19987071   20595360
2 Brazil        172006362  174504898
3 China        1272915272 1280428583
```

```
  country     `1999` `2000`
* <chr>        <int>  <int>
1 Afghanistan    745   2666
2 Brazil       37737  80488
3 China       212258 213766
```

Each observation must have its own row

```
  country      year  cases population
  <chr>       <int>  <int>      <int>
1 Afghanistan  1999    745   19987071
2 Afghanistan  2000   2666   20595360
3 Brazil       1999  37737  172006362
4 Brazil       2000  80488  174504898
5 China        1999 212258 1272915272
6 China        2000 213766 1280428583
```

```
  country      year rate
* <chr>       <int> <chr>
1 Afghanistan  1999 745/19987071
2 Afghanistan  2000 2666/20595360
3 Brazil       1999 37737/172006362
4 Brazil       2000 80488/174504898
5 China        1999 212258/1272915272
6 China        2000 213766/1280428583
```

```
   country      year type            count
   <chr>       <int> <chr>           <int>
 1 Afghanistan  1999 cases             745
 2 Afghanistan  1999 population   19987071
 3 Afghanistan  2000 cases            2666
 4 Afghanistan  2000 population   20595360
 5 Brazil       1999 cases           37737
 6 Brazil       1999 population  172006362
 7 Brazil       2000 cases           80488
 8 Brazil       2000 population  174504898
 9 China        1999 cases          212258
10 China        1999 population 1272915272
11 China        2000 cases          213766
12 China        2000 population 1280428583
```

2 rows per observation

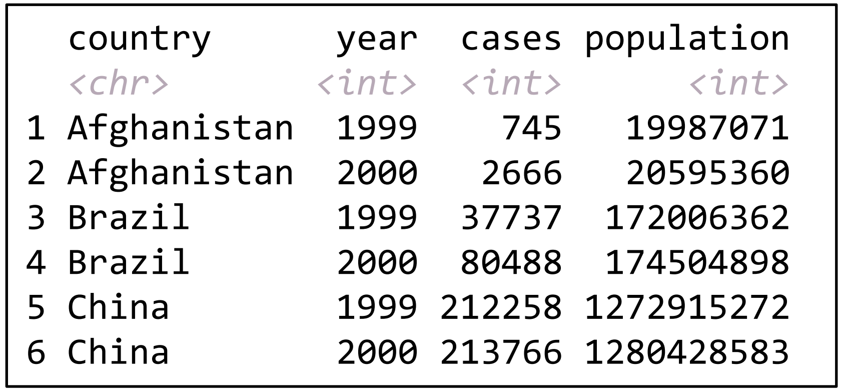
Observations spread across two tables

```
  country         `1999`     `2000`
* <chr>            <int>      <int>
1 Afghanistan   19987071   20595360
2 Brazil       172006362  174504898
3 China       1272915272 1280428583
```

```
  country     `1999` `2000`
* <chr>        <int>  <int>
1 Afghanistan    745   2666
2 Brazil       37737  80488
3 China       212258 213766
```

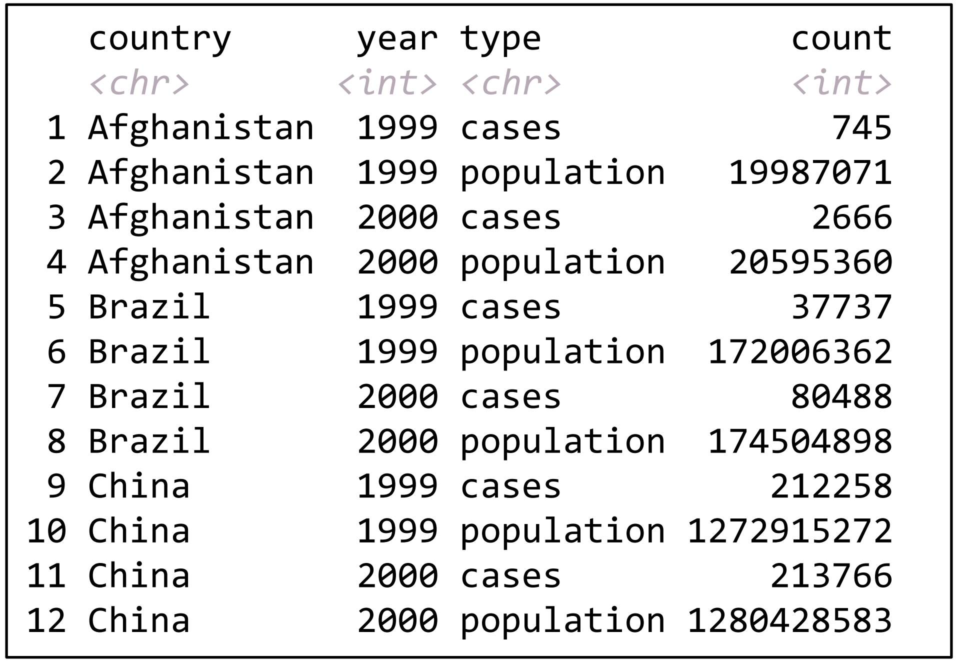
Does each value have its own cell?

1



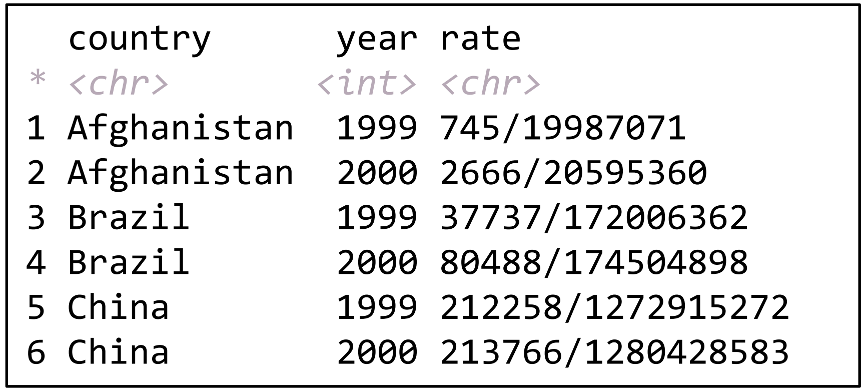
```
  country      year  cases population
  <chr>       <int>  <int>      <int>
1 Afghanistan  1999    745   19987071
2 Afghanistan  2000   2666   20595360
3 Brazil       1999  37737  172006362
4 Brazil       2000  80488  174504898
5 China        1999 212258 1272915272
6 China        2000 213766 1280428583
```

2



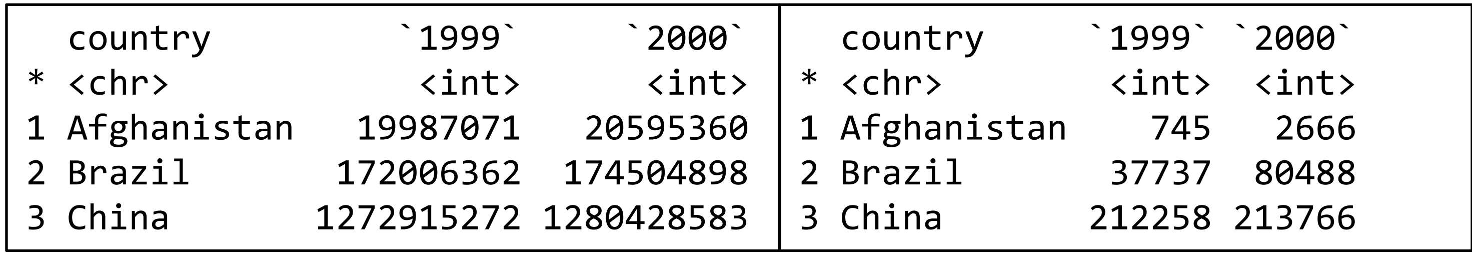
```
   country      year type            count
   <chr>       <int> <chr>           <int>
 1 Afghanistan  1999 cases             745
 2 Afghanistan  1999 population   19987071
 3 Afghanistan  2000 cases            2666
 4 Afghanistan  2000 population   20595360
 5 Brazil       1999 cases           37737
 6 Brazil       1999 population  172006362
 7 Brazil       2000 cases           80488
 8 Brazil       2000 population  174504898
 9 China        1999 cases          212258
10 China        1999 population 1272915272
11 China        2000 cases          213766
12 China        2000 population 1280428583
```

3



```
  country      year rate
* <chr>       <int> <chr>
1 Afghanistan  1999 745/19987071
2 Afghanistan  2000 2666/20595360
3 Brazil       1999 37737/172006362
4 Brazil       2000 80488/174504898
5 China        1999 212258/1272915272
6 China        2000 213766/1280428583
```

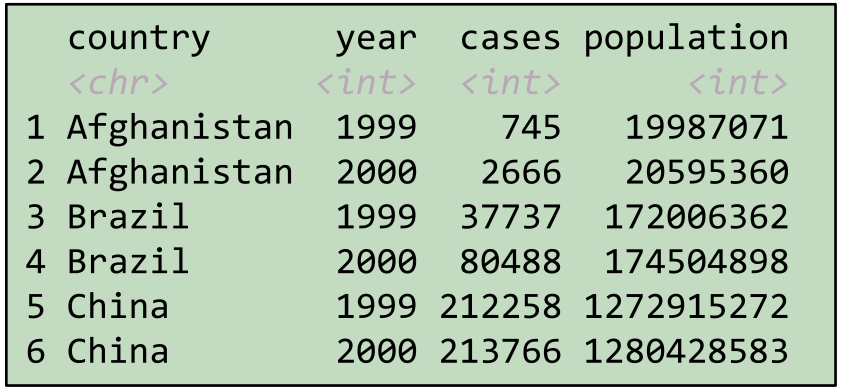
4



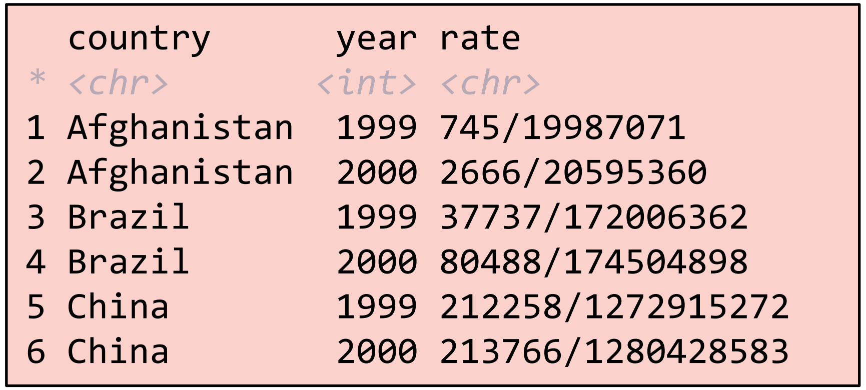
```
  country          `1999`     `2000`
* <chr>             <int>      <int>
1 Afghanistan    19987071   20595360
2 Brazil        172006362  174504898
3 China        1272915272 1280428583
```

```
  country     `1999` `2000`
* <chr>        <int>  <int>
1 Afghanistan    745   2666
2 Brazil       37737  80488
3 China       212258 213766
```

Each value must have its own cell

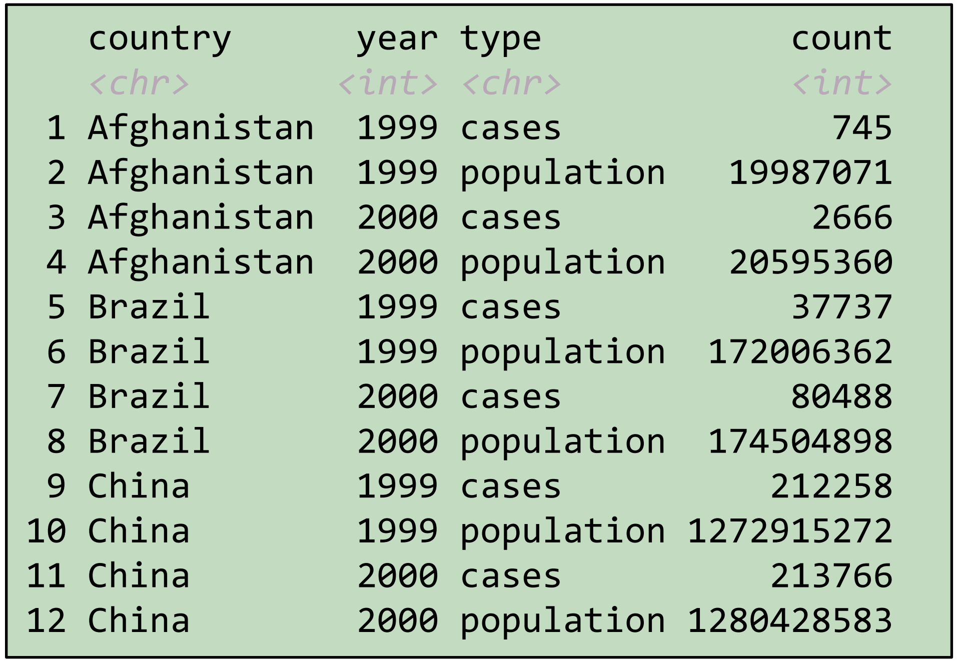


```
  country      year  cases population
  <chr>       <int>  <int>      <int>
1 Afghanistan  1999    745   19987071
2 Afghanistan  2000   2666   20595360
3 Brazil       1999  37737  172006362
4 Brazil       2000  80488  174504898
5 China        1999 212258 1272915272
6 China        2000 213766 1280428583
```

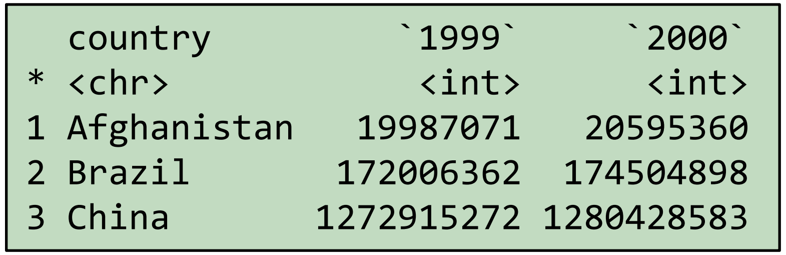


```
  country      year rate
* <chr>       <int> <chr>
1 Afghanistan  1999 745/19987071
2 Afghanistan  2000 2666/20595360
3 Brazil       1999 37737/172006362
4 Brazil       2000 80488/174504898
5 China        1999 212258/1272915272
6 China        2000 213766/1280428583
```

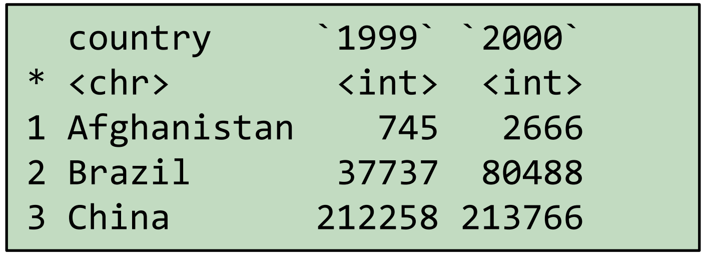
One cell contains two values



```
   country      year type            count
   <chr>       <int> <chr>           <int>
 1 Afghanistan  1999 cases             745
 2 Afghanistan  1999 population   19987071
 3 Afghanistan  2000 cases            2666
 4 Afghanistan  2000 population   20595360
 5 Brazil       1999 cases           37737
 6 Brazil       1999 population  172006362
 7 Brazil       2000 cases           80488
 8 Brazil       2000 population  174504898
 9 China        1999 cases          212258
10 China        1999 population 1272915272
11 China        2000 cases          213766
12 China        2000 population 1280428583
```



```
  country          `1999`     `2000`
* <chr>             <int>      <int>
1 Afghanistan    19987071   20595360
2 Brazil        172006362  174504898
3 China        1272915272 1280428583
```



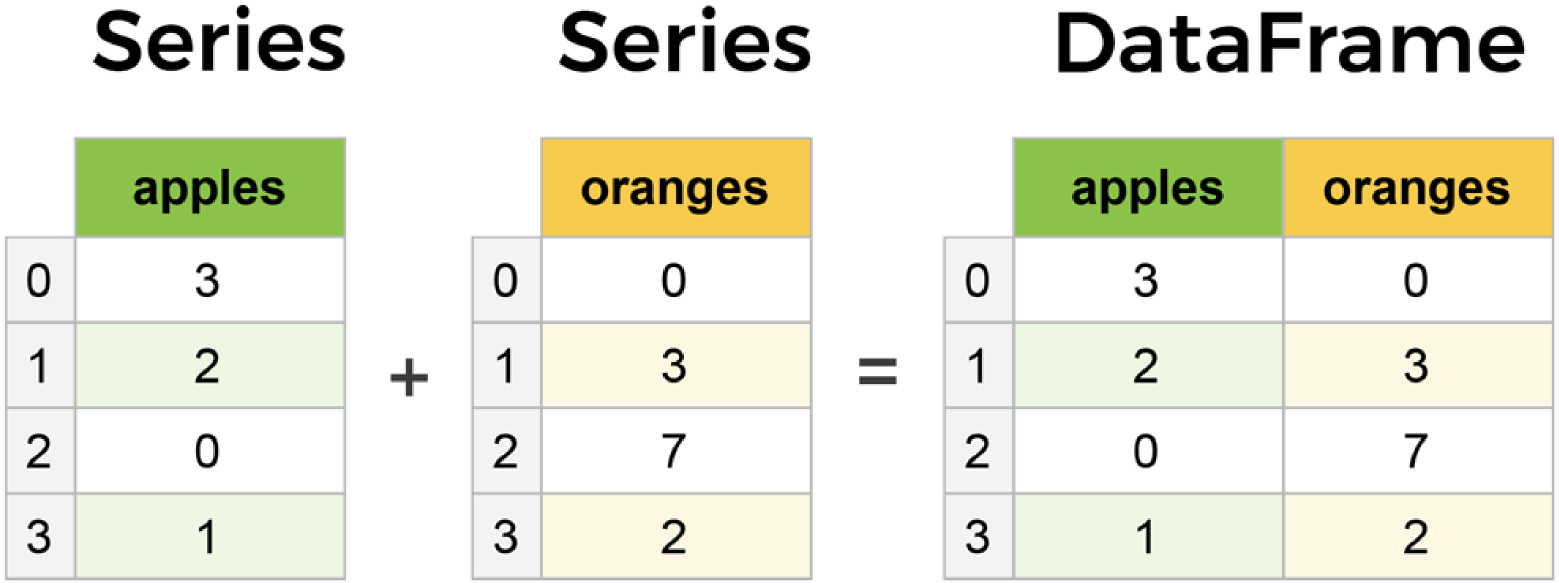
```
  country     `1999` `2000`
* <chr>        <int>  <int>
1 Afghanistan    745   2666
2 Brazil       37737  80488
3 China       212258 213766
```

Data science

Pandas – Data Structures

Core components of pandas: Series & DataFrame

- A Series is essentially an indexed column
- A DataFrame is a multi-dimensional table made up of a collection of Series



Series

	apples
0	3
1	2
2	0
3	1

+

Series

	oranges
0	0
1	3
2	7
3	2

=

DataFrame

	apples	oranges
0	3	0
1	2	3
2	0	7
3	1	2

Creating DataFrames from scratch

- There are many ways to create a DataFrame from scratch, but a great option is to use a simple dictionary.
- Let's say we have a fruit stand that sells apples and oranges. We want to have a column for each fruit and a row for each customer purchase. Organized as a dictionary this looks like:

```
data = {
'apples': [3, 2, 0, 1],
'oranges': [0, 3, 7, 2]
}
```

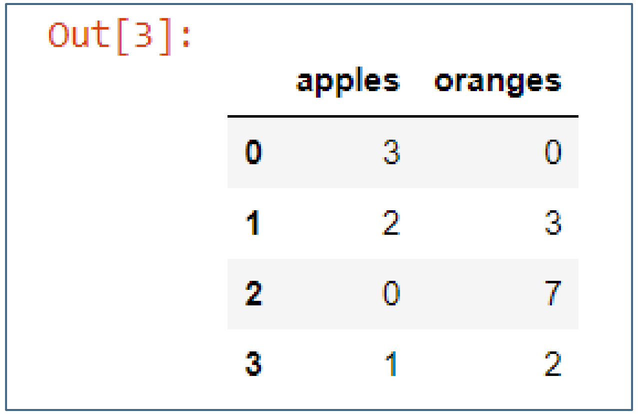
- The data dictionary counts 2 key-value pairs
  - Keys: apples + oranges   Values: lists with the corresponding purchases

Creating DataFrames from scratch

- Next the dictionary can be passed to the DataFrame constructor

```
purchases = pd.DataFrame(data)

purchases
```

- Each (key, value) pair in the dictionary now corresponds to a column in the resulting DataFrame. The **index** of this DataFrame is generated on creation as the numbers 0-3.



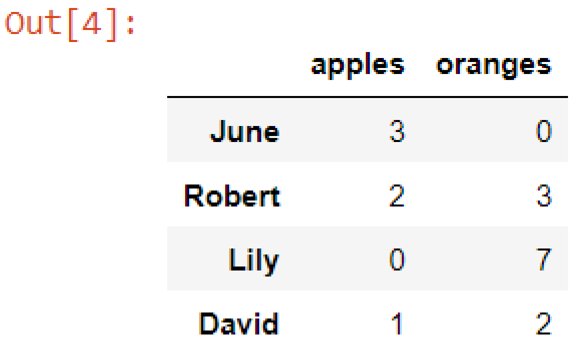
Out[3]:

	apples	oranges
0	3	0
1	2	3
2	0	7
3	1	2

Creating DataFrames from scratch

- Instead of the default numeric indexing, each row can get a custom index label: e.g. customer names

```
purchases = pd.DataFrame(data, index=['June', 'Robert', 'Lily', ‘David'])
purchases
```

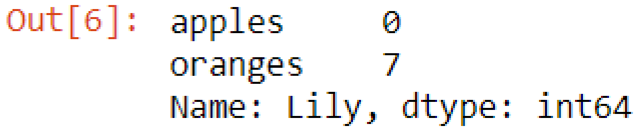


Out[4]:

	apples	oranges
June	3	0
Robert	2	3
Lily	0	7
David	1	2

- A datarow of a specific customer can then be located based on their name

```
purchases = purchases.loc[ 'Lily']
```



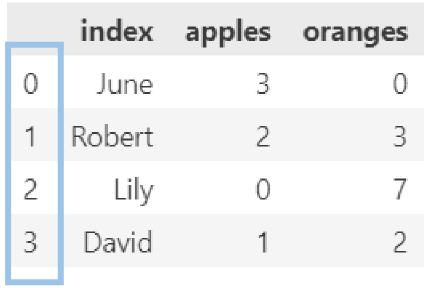
```
Out[6]: apples     0
        oranges    7
        Name: Lily, dtype: int64
```

To extract rows based on the index label, you use:
.loc[‘label’]

Reset index

- You can always go back from the labeled indices (e.g. index=['June', 'Robert', 'Lily', ‘David’]) to the default numerical ones (e.g. 0-3) by using *reset_index()*

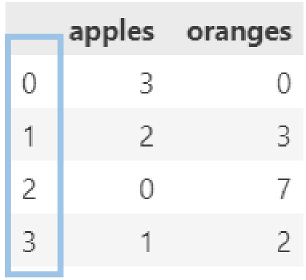
```
purchases =purchases.reset_index()
```



	index	apples	oranges
0	June	3	0
1	Robert	2	3
2	Lily	0	7
3	David	1	2

- Use the drop parameter to avoid the old index being added as a column:

```
purchases =purchases.reset_index(drop = True)
```



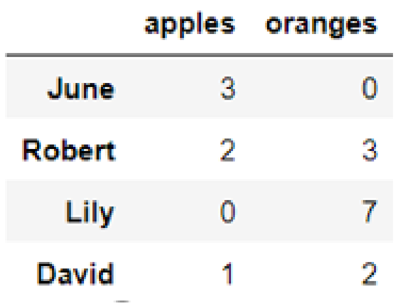
	apples	oranges
0	3	0
1	2	3
2	0	7
3	1	2

Data science

Pandas – Extracting and assigning data

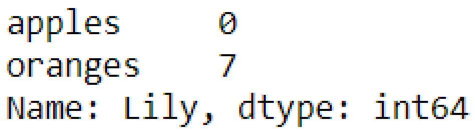
Extracting data

- A datarow can also be extracted based on index number with *iloc[index]*



	apples	oranges
June	3	0
Robert	2	3
Lily	0	7
David	1	2

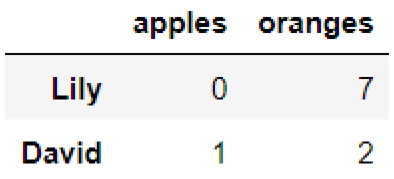
```
purchases = purchases.iloc[2]
```



```
apples     0
oranges    7
Name: Lily, dtype: int64
```

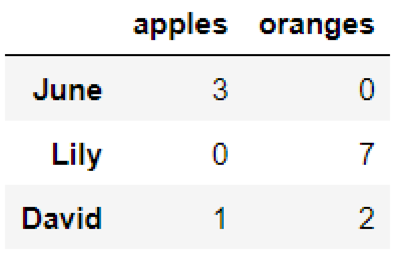
- By specifying a list of indexes multiple rows are retrieved

```
purchases = purchases.iloc[[2,3]]
```



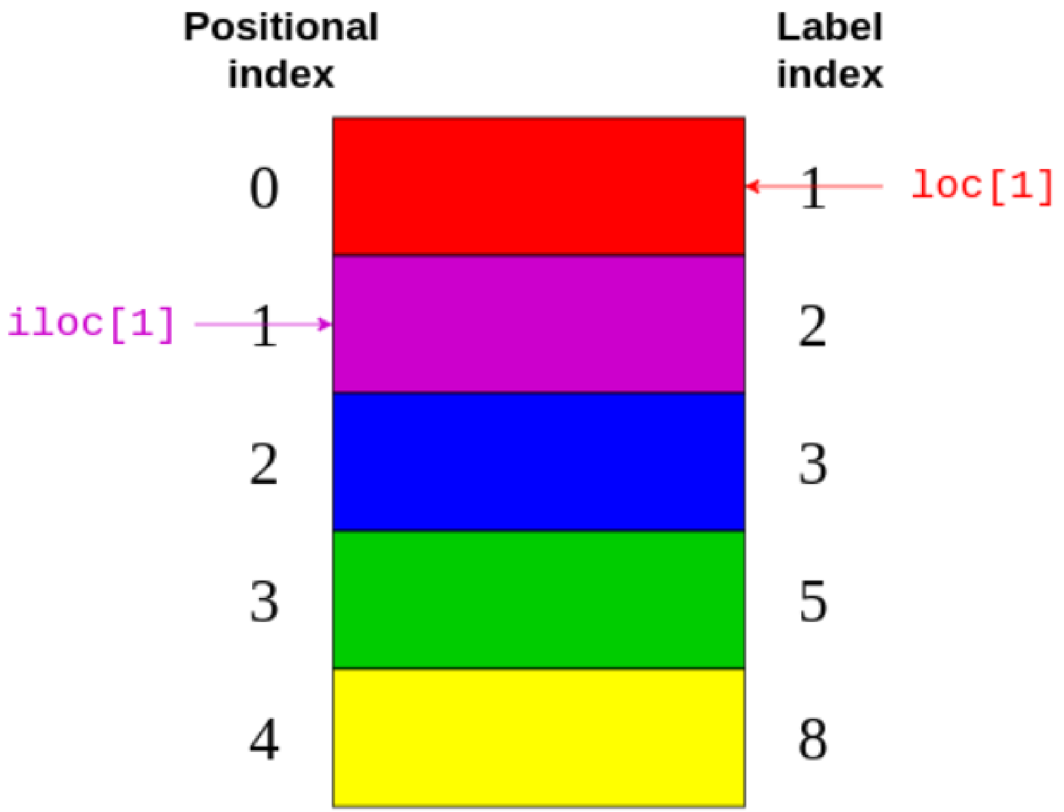
	apples	oranges
Lily	0	7
David	1	2

```
purchases = purchases.iloc[[0,2,3]]
```



	apples	oranges
June	3	0
Lily	0	7
David	1	2

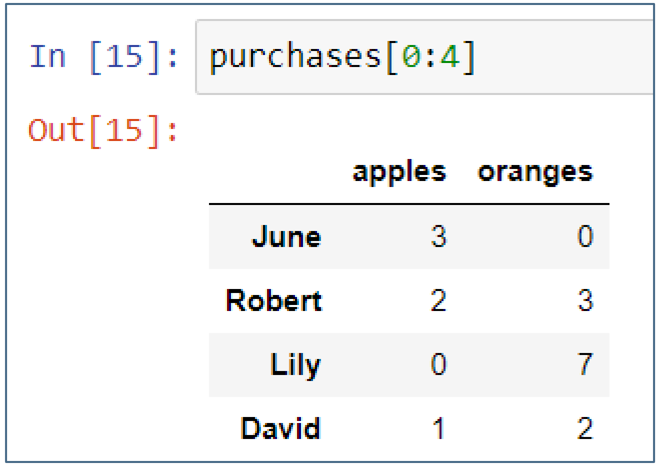
Extracting data



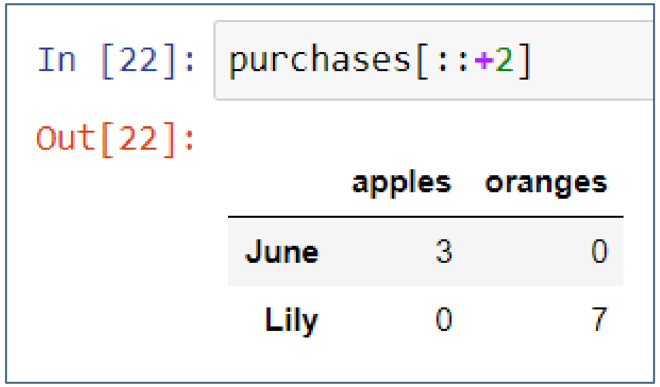
https://files.realpython.com/media/iloc_vs_loc_80_border20.d5280f475f4e.png

Extracting data

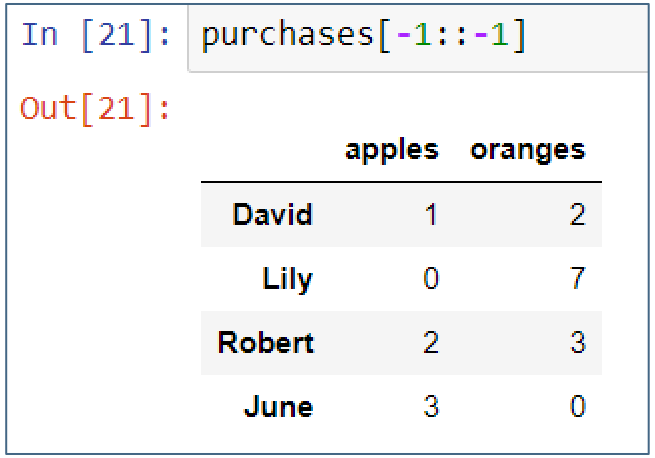
- Extracting rows from a DataFrame can also be done by **slicing**



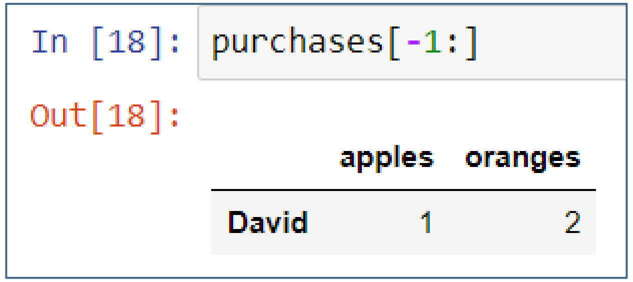
```
In [15]: purchases[0:4]
Out[15]:
         apples  oranges
June          3        0
Robert        2        3
Lily          0        7
David         1        2
```



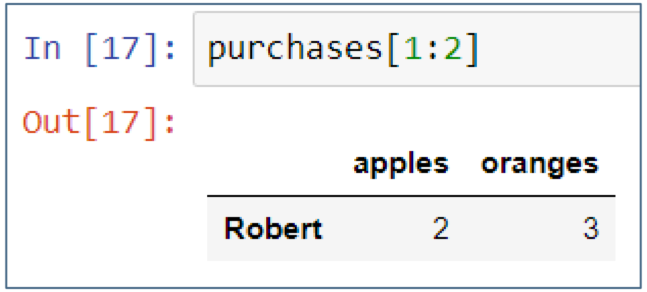
```
In [22]: purchases[::+2]
Out[22]:
         apples  oranges
June          3        0
Lily          0        7
```



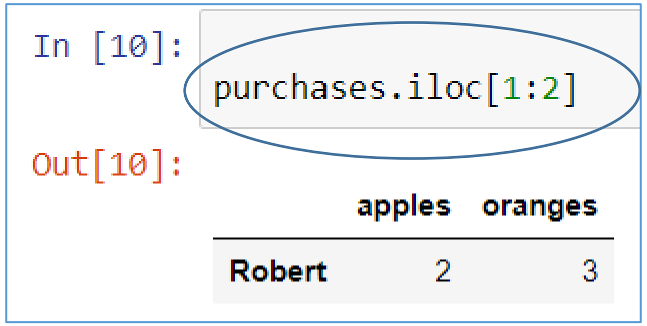
```
In [21]: purchases[-1::-1]
Out[21]:
         apples  oranges
David         1        2
Lily          0        7
Robert        2        3
June          3        0
```



```
In [18]: purchases[-1:]
Out[18]:
         apples  oranges
David         1        2
```



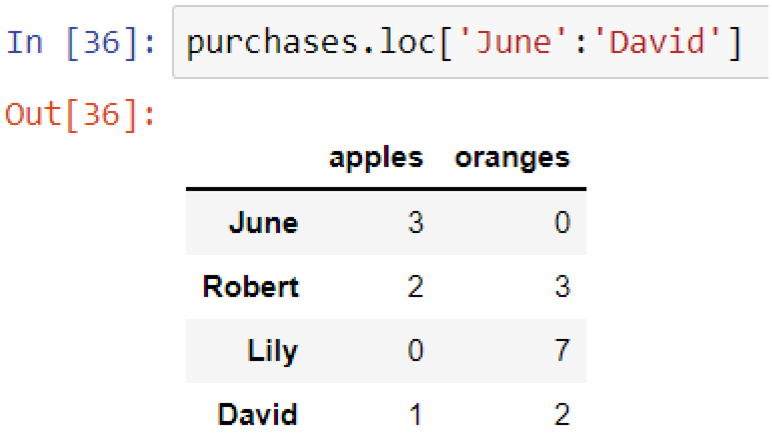
```
In [17]: purchases[1:2]
Out[17]:
         apples  oranges
Robert        2        3
```



```
In [10]: purchases.iloc[1:2]
Out[10]:
         apples  oranges
Robert        2        3
```

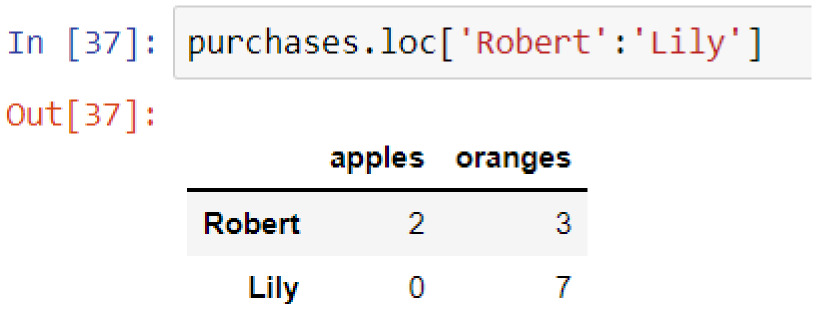
Extracting data

- Slicing works with index labels



```
In [36]: purchases.loc['June':'David']
Out[36]:
```

	apples	oranges
June	3	0
Robert	2	3
Lily	0	7
David	1	2



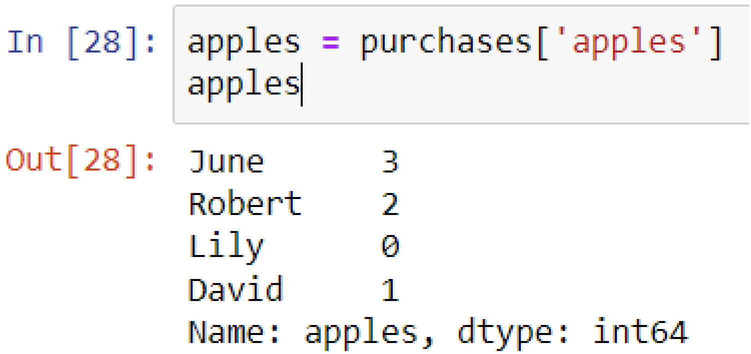
```
In [37]: purchases.loc['Robert':'Lily']
Out[37]:
```

	apples	oranges
Robert	2	3
Lily	0	7

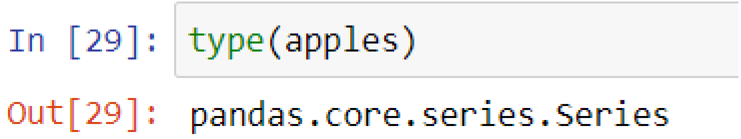
- Note that if using labels (loc) the end-element is included in the result, which is not true in case of normal index-based slicing (iloc)

Extracting data

- Extracting 1 column from a dataframe results in a Series



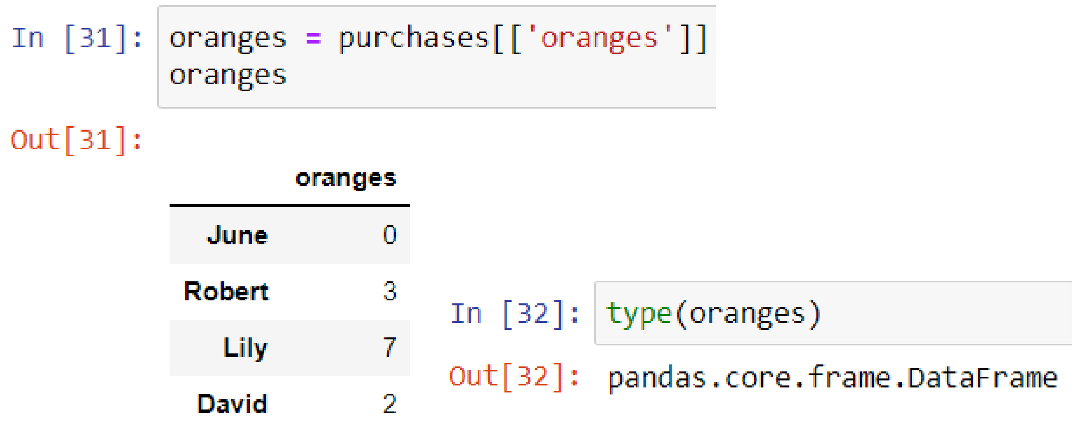
```
In [28]: apples = purchases['apples']
         apples

Out[28]: June      3
         Robert    2
         Lily      0
         David     1
         Name: apples, dtype: int64
```



```
In [29]: type(apples)

Out[29]: pandas.core.series.Series
```

Note that purchases.apples will also work

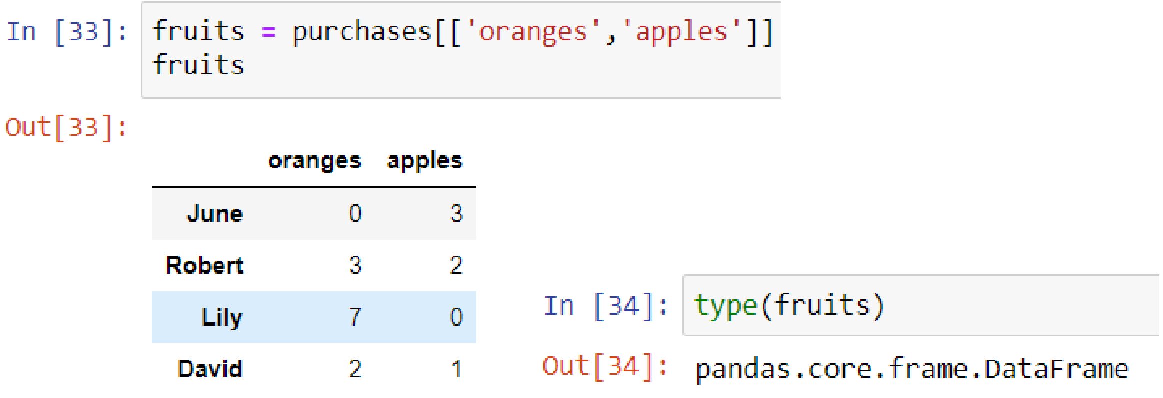
- To extract columns as a *DataFrame*, you need to pass a list[] of column names



```
In [31]: oranges = purchases[['oranges']]
         oranges

Out[31]:
```

	oranges
June	0
Robert	3
Lily	7
David	2

```
In [32]: type(oranges)

Out[32]: pandas.core.frame.DataFrame
```



```
In [33]: fruits = purchases[['oranges','apples']]
         fruits

Out[33]:
```

	oranges	apples
June	0	3
Robert	3	2
Lily	7	0
David	2	1

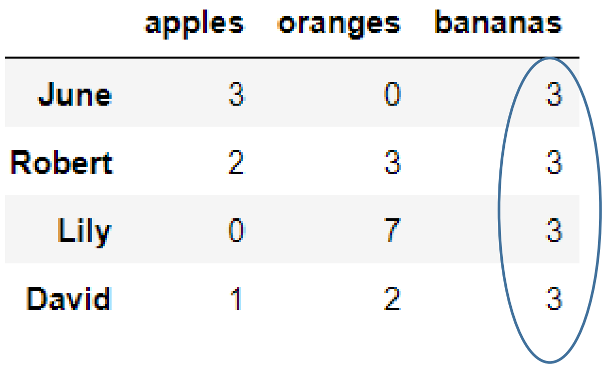
```
In [34]: type(fruits)

Out[34]: pandas.core.frame.DataFrame
```

Assigning data

Going the other way, assigning data to a DataFrame is easy.
You can insert/update a column with either:

- a constant value:

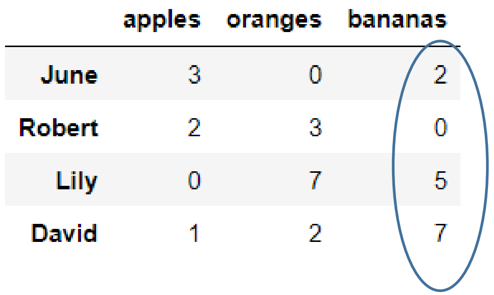
```
purchases['bananas']=3
```



	apples	oranges	bananas
June	3	0	3
Robert	2	3	3
Lily	0	7	3
David	1	2	3

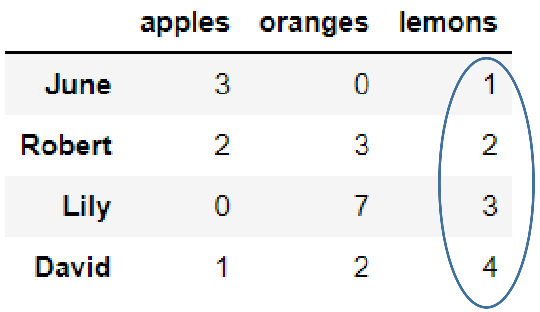
- or an iterable of values:

```
purchases['bananas’]=[2,0,5,7]
```



	apples	oranges	bananas
June	3	0	2
Robert	2	3	0
Lily	0	7	5
David	1	2	7

```
purchases['lemons'] = range(1,len(purchases)+1,+1)
```

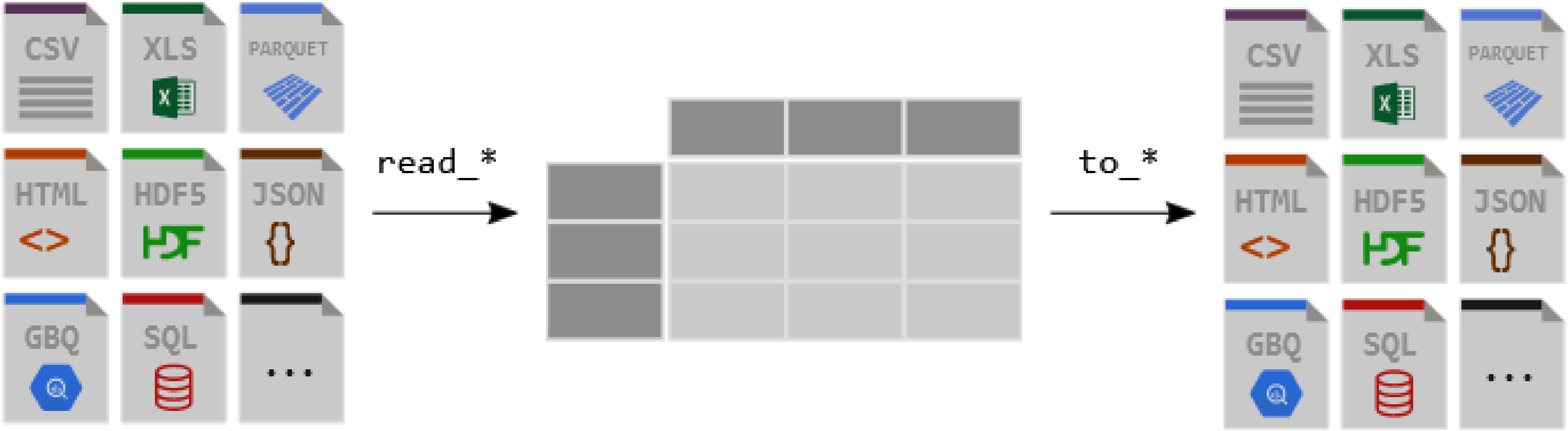


	apples	oranges	lemons
June	3	0	1
Robert	2	3	2
Lily	0	7	3
David	1	2	4

Data science

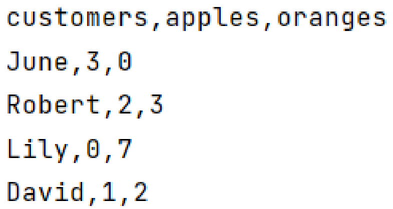
Pandas – Read and write data

Read and Write



https://pandas.pydata.org/docs/getting_started/index.html

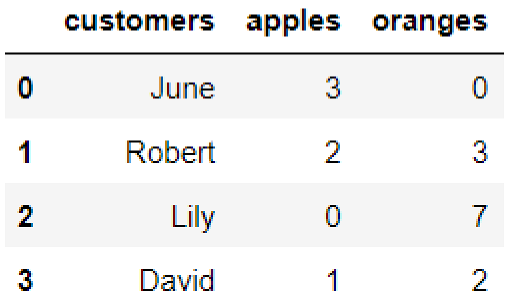
Reading data from CSVs



```
customers,apples,oranges
June,3,0
Robert,2,3
Lily,0,7
David,1,2
```

- With CSV files all you need is a single line to load in the data:

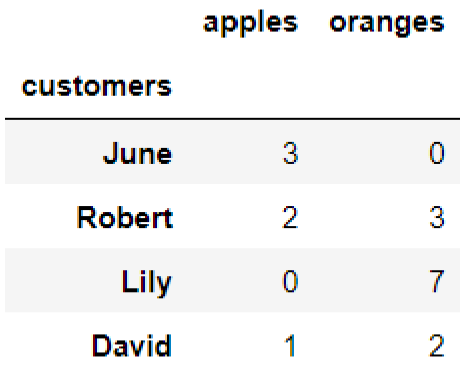
```
df = pd.read_csv('purchases.csv')
```



	customers	apples	oranges
0	June	3	0
1	Robert	2	3
2	Lily	0	7
3	David	1	2

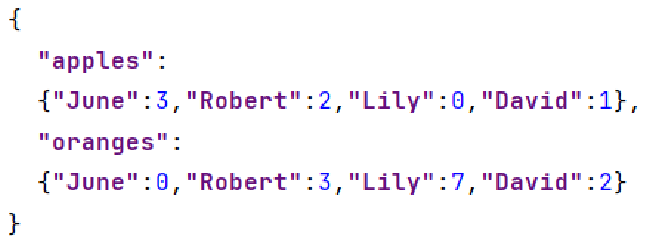
- CSVs don't have indexes like our DataFrames, so you need to designate the index_col when reading:

```
df = pd.read_csv('purchases.csv', index_col=0)
```



	apples	oranges
customers		
June	3	0
Robert	2	3
Lily	0	7
David	1	2

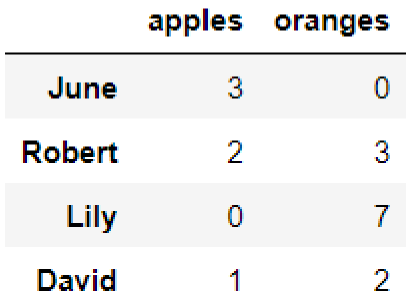
Reading data from JSON



```
{
  "apples":
  {"June":3,"Robert":2,"Lily":0,"David":1},
  "oranges":
  {"June":0,"Robert":3,"Lily":7,"David":2}
}
```

- If you have a JSON file pandas can read this just as easily.

```
df = pd.read_json('purchases.json')
```



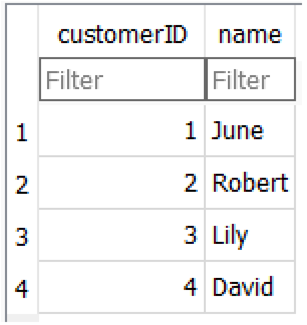
	apples	oranges
June	3	0
Robert	2	3
Lily	0	7
David	1	2

- Notice this time the index came correctly since JSON allows indexes to work through nesting.

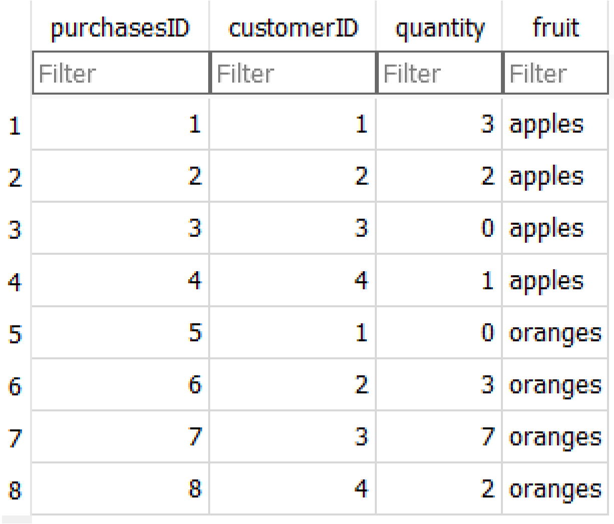
- JSON isn't a tabular format. When pandas cannot deduce the JSON structure, set the *orient* parameter. (https://www.roelpeters.be/pandas-read-json-orient)

Reading data from databases

- You can use any database, but we’ll be using SQLite
  - No installation required, the entire database is contained in a file that you can open using portable software
  - https://sqlitebrowser.org/
- We’re using 2 tables and yes, they are badly normalized



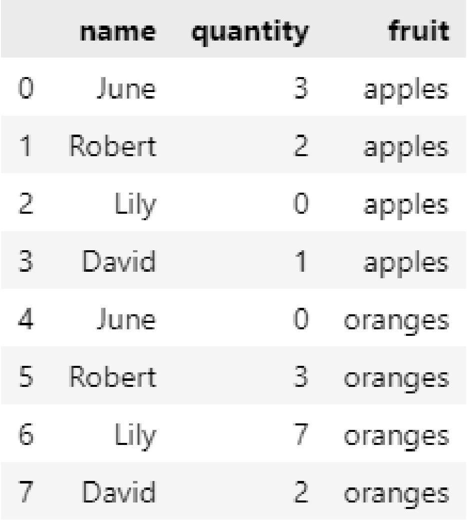
	customerID	name
	Filter	Filter
1	1	June
2	2	Robert
3	3	Lily
4	4	David



	purchasesID	customerID	quantity	fruit
	Filter	Filter	Filter	Filter
1	1	1	3	apples
2	2	2	2	apples
3	3	3	0	apples
4	4	4	1	apples
5	5	1	0	oranges
6	6	2	3	oranges
7	7	3	7	oranges
8	8	4	2	oranges

Reading data from databases

```python
import sqlite3

con = sqlite3.connect("purchases.db")
df = pd.read_sql_query("SELECT c.name, p.quantity, p.fruit
FROM customers c join purchases p on c.customerID =
p.customerID", con)
df
```



	name	quantity	fruit
0	June	3	apples
1	Robert	2	apples
2	Lily	0	apples
3	David	1	apples
4	June	0	oranges
5	Robert	3	oranges
6	Lily	7	oranges
7	David	2	oranges

- The resulting dataframe is different from what we got from a CSV-file
  - That’s because of how normalized data works
- We could fix this using the techniques that follow in this and the next chapter
  - But for now it’s a good enough proof of concept

Storing DataFrame in CSV, JSON or Excel

- After extensive work on cleaning data in a DataFrame, the result can be saved as a file of your choice.
- Similar to the ways we read in data, pandas provides intuitive commands to save it:

```
df.to_csv('new_purchases.csv')
df.to_json('new_purchases.json’)
df.to_excel('new_purchases.xlsx’)
```

Data science

Pandas – DataFrame basic operations

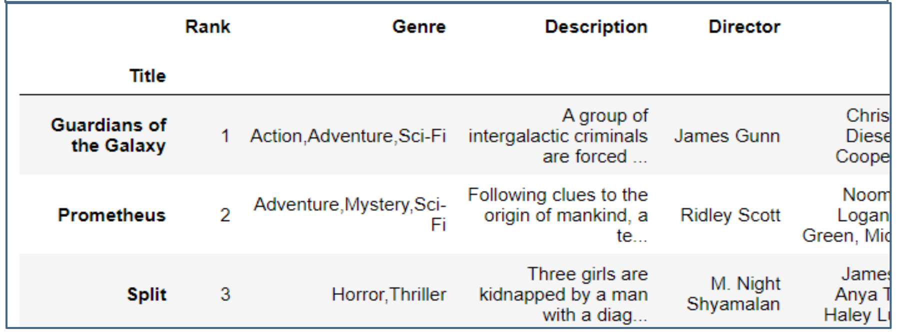
Basic DataFrame operations

- Load the IMDB movies dataset from csv using the movie titles as index.

```
df_movies = pd.read_csv("IMDB-Movie-Data.csv", index_col="Title")
```

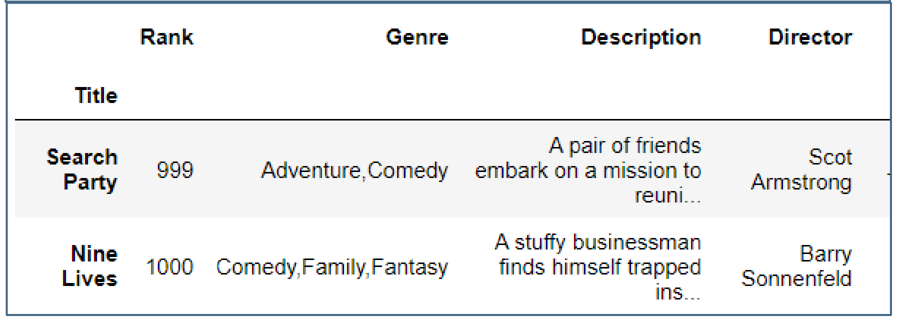
- To get an overview of the first or last records in the DataFrame, use the head() or tail() methods. Both accept a number as argument to indicate how many records should be retrieved.

```
df_movies.head(3)
```



	Rank	Genre	Description	Director	
Title					
Guardians of the Galaxy	1	Action,Adventure,Sci-Fi	A group of intergalactic criminals are forced ...	James Gunn	Chris Diese Coope
Prometheus	2	Adventure,Mystery,Sci-Fi	Following clues to the origin of mankind, a te...	Ridley Scott	Noom Logan Green, Mic
Split	3	Horror,Thriller	Three girls are kidnapped by a man with a diag...	M. Night Shyamalan	James Anya T Haley L

```
df_movies.tail(2)
```

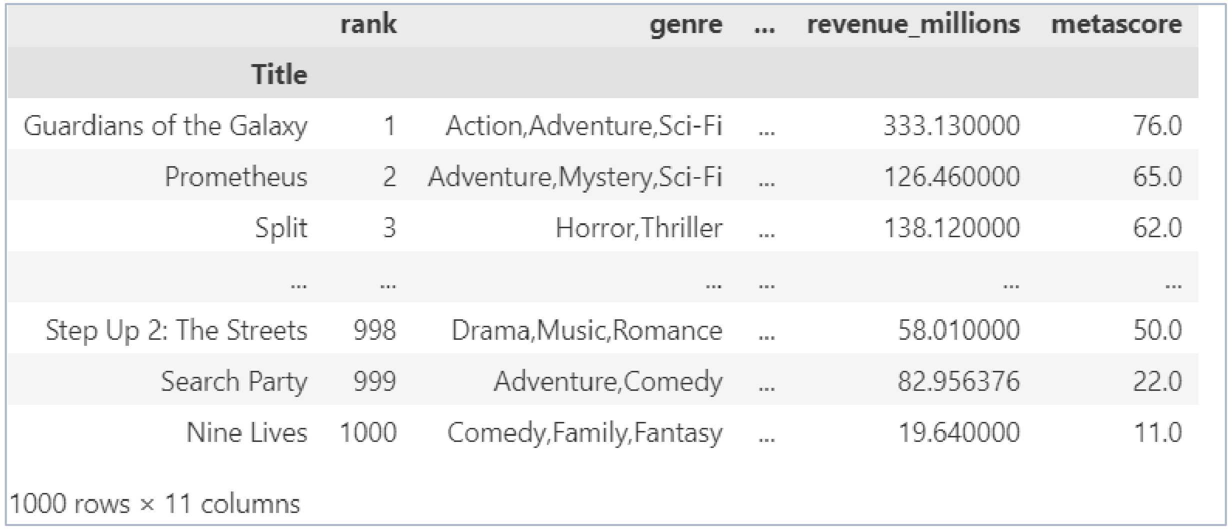


	Rank	Genre	Description	Director
Title				
Search Party	999	Adventure,Comedy	A pair of friends embark on a mission to reuni...	Scot Armstrong
Nine Lives	1000	Comedy,Family,Fantasy	A stuffy businessman finds himself trapped ins...	Barry Sonnenfeld

set_option("display.max_...")

- To customize the number of rows or columns to be displayed:

```
pd.set_option('display.max_columns', 4)
pd.set_option('display.max_rows', 6)
df_movies
```



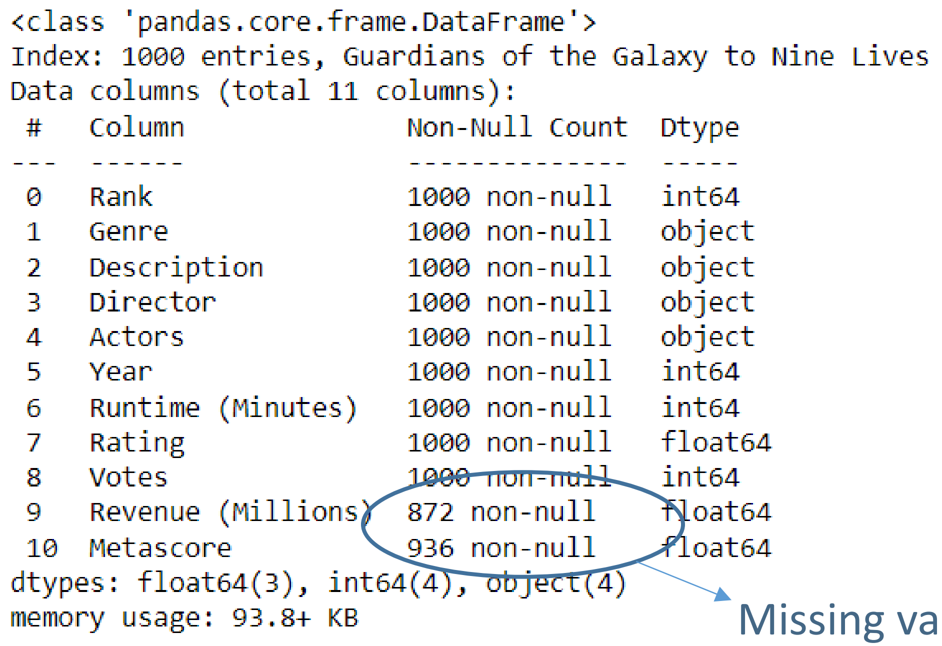
	rank	genre	...	revenue_millions	metascore
Title					
Guardians of the Galaxy	1	Action,Adventure,Sci-Fi	...	333.130000	76.0
Prometheus	2	Adventure,Mystery,Sci-Fi	...	126.460000	65.0
Split	3	Horror,Thriller	...	138.120000	62.0
...	...	...	...	...	...
Step Up 2: The Streets	998	Drama,Music,Romance	...	58.010000	50.0
Search Party	999	Adventure,Comedy	...	82.956376	22.0
Nine Lives	1000	Comedy,Family,Fantasy	...	19.640000	11.0

1000 rows × 11 columns

Getting info about the data

The method *.info()* provides the essential details about the dataset, such as the number of rows and columns, the number of non-null values, what type of data is in each column, and how much memory the DataFrame is using.

```
df_movies.info()
```

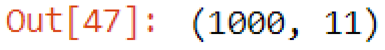


```
<class 'pandas.core.frame.DataFrame'>
Index: 1000 entries, Guardians of the Galaxy to Nine Lives
Data columns (total 11 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   Rank                1000 non-null   int64  
 1   Genre               1000 non-null   object 
 2   Description         1000 non-null   object 
 3   Director            1000 non-null   object 
 4   Actors              1000 non-null   object 
 5   Year                1000 non-null   int64  
 6   Runtime (Minutes)   1000 non-null   int64  
 7   Rating              1000 non-null   float64
 8   Votes               1000 non-null   int64  
 9   Revenue (Millions)  872 non-null    float64
 10  Metascore           936 non-null    float64
dtypes: float64(3), int64(4), object(4)
memory usage: 93.8+ KB
```

Missing values!

The dataframe counts 1000 rows and 11 columns
This information can also be retrieved with the property **shape**.

```
df_movies.shape
```



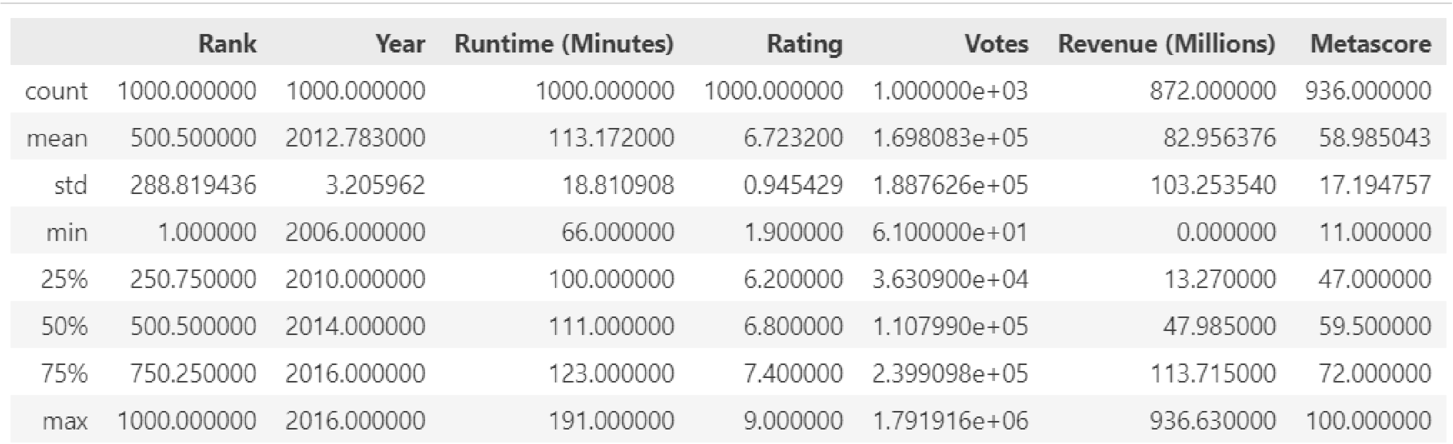
```
Out[47]:  (1000, 11)
```

Getting info about the data

- You can use the ***.describe()*** method to find basic statistical characteristics:

```
df_movies.describe()
```

  - Count: number of non-missing values
  - Mean
  - Standard deviation
  - Median (0,50)
  - Quartiles (0,25/0,75)
  - Range: min-max

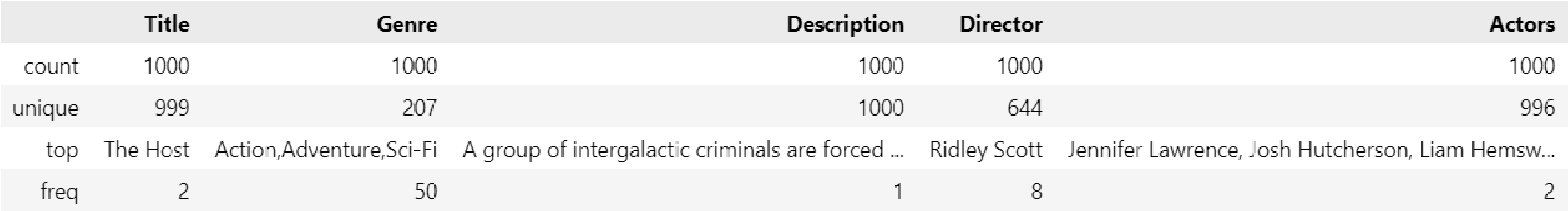


	Rank	Year	Runtime (Minutes)	Rating	Votes	Revenue (Millions)	Metascore
count	1000.000000	1000.000000	1000.000000	1000.000000	1.000000e+03	872.000000	936.000000
mean	500.500000	2012.783000	113.172000	6.723200	1.698083e+05	82.956376	58.985043
std	288.819436	3.205962	18.810908	0.945429	1.887626e+05	103.253540	17.194757
min	1.000000	2006.000000	66.000000	1.900000	6.100000e+01	0.000000	11.000000
25%	250.750000	2010.000000	100.000000	6.200000	3.630900e+04	13.270000	47.000000
50%	500.500000	2014.000000	111.000000	6.800000	1.107990e+05	47.985000	59.500000
75%	750.250000	2016.000000	123.000000	7.400000	2.399098e+05	113.715000	72.000000
max	1000.000000	2016.000000	191.000000	9.000000	1.791916e+06	936.630000	100.000000

Getting info about the data

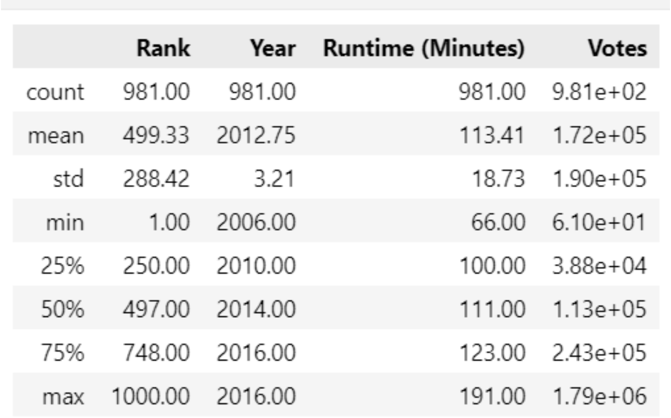
- When using parameter “include” you can select columns of a certain type

```
df_movies.describe(include=“object")
```



	Title	Genre	Description	Director	Actors
count	1000	1000	1000	1000	1000
unique	999	207	1000	644	996
top	The Host	Action,Adventure,Sci-Fi	A group of intergalactic criminals are forced ...	Ridley Scott	Jennifer Lawrence, Josh Hutcherson, Liam Hemsw...
freq	2	50	1	8	2

```
df_movies.describe(include="int")
```



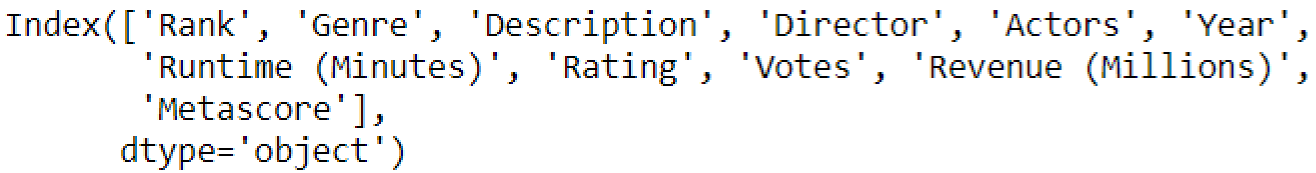
	Rank	Year	Runtime (Minutes)	Votes
count	981.00	981.00	981.00	9.81e+02
mean	499.33	2012.75	113.41	1.72e+05
std	288.42	3.21	18.73	1.90e+05
min	1.00	2006.00	66.00	6.10e+01
25%	250.00	2010.00	100.00	3.88e+04
50%	497.00	2014.00	111.00	1.13e+05
75%	748.00	2016.00	123.00	2.43e+05
max	1000.00	2016.00	191.00	1.79e+06

! Notice the characteristics change with the datatype

Column names cleanup

- Best practice for column names is to lowercase them, remove special characters, and replace spaces with underscores
- Here's how to get the column names of our dataset:

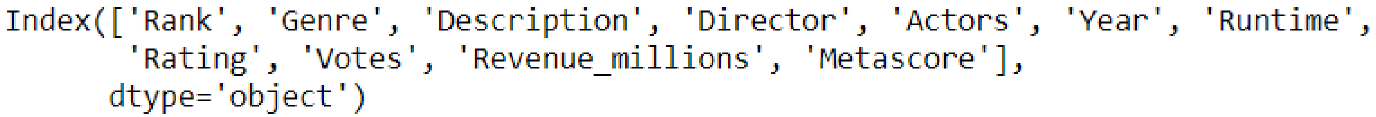
```
df_movies.columns
```



```
Index(['Rank', 'Genre', 'Description', 'Director', 'Actors', 'Year',
       'Runtime (Minutes)', 'Rating', 'Votes', 'Revenue (Millions)',
       'Metascore'],
      dtype='object')
```

- Use the ***.rename()*** method to rename certain or all columns via a dictionary.

```
df_movies.rename(columns={
'Runtime (Minutes)': 'Runtime',
'Revenue (Millions)': 'Revenue_millions'
}, inplace=True)
```



```
Index(['Rank', 'Genre', 'Description', 'Director', 'Actors', 'Year', 'Runtime',
       'Rating', 'Votes', 'Revenue_millions', 'Metascore'],
      dtype='object')
```

Inplace = true is an alternative way to code df_movies = df_movies.rename(….)

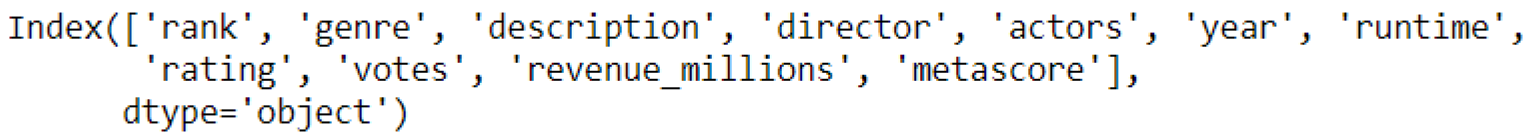
Column names cleanup

- You can also set the columns property to a list of appropriate names:

```
df_movies.columns = ['rank', 'genre', 'description', 'director', 'actors', 'year', 'runtime',
'rating', 'votes', 'revenue_millions', 'metascore']
```

- The above solution is a long way to lowercase each column. Instead of just renaming each column manually we can do a list comprehension:

```
df_movies.columns = [col.lower() for col in df_movies]
```



```
Index(['rank', 'genre', 'description', 'director', 'actors', 'year', 'runtime',
       'rating', 'votes', 'revenue_millions', 'metascore'],
      dtype='object')
```

Data science

Pandas – Missing values

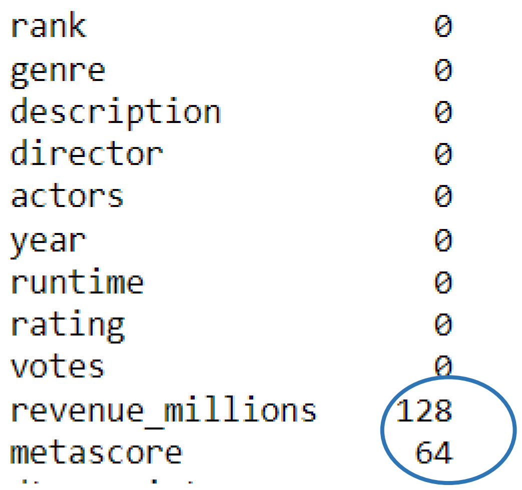
Missing values

- When exploring data, you’ll most likely encounter missing or null values, which are essentially placeholders for non-existent values.
- Most commonly you'll see Python's None or NumPy's np.nan
- There are two options in dealing with nulls:
  1. Get rid of rows or columns with nulls (**amputation**)
  2. Replace nulls with non-null values (**imputation**)
- Data scientists regularly face the dilemma of dropping or imputing null values. This decision requires profound knowledge of the data and its context. Overall, removing null data is only suggested if you have a small amount of missing data.

Missing values

- *.isnull()* returns for every data cell whether a value is present.
- To count the number of nulls in each column use an aggregate function for summing.

```
df_movies.isnull().sum()
```



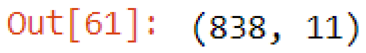
```
rank                  0
genre                 0
description           0
director              0
actors                0
year                  0
runtime               0
rating                0
votes                 0
revenue_millions    128
metascore            64
```

We can see now that our data has 128 missing values for revenue_millions and 64 missing values for metascore.

Missing values: remove rows or columns

- Remove rows with missing elements:

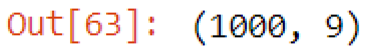
```
df_movies.dropna(inplace=True)
df_movies. shape
```



```
Out[61]: (838, 11)
```

- 162 rows are deleted with missing revenue_millions and/or metascore
- This obviously seems like a waste since there's perfectly good data in the other columns of those dropped rows
- Other than just dropping rows, you can also drop columns with null values by setting the parameter axis=1:

```
df_movies.dropna(axis = 1, inplace=True)
df_movies. shape
```



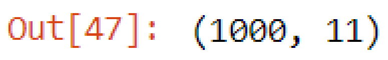
```
Out[63]: (1000, 9)
```

- Now the columns revenue_millions and metascore are gone

What's with this axis=1 parameter?

It's not immediately obvious where axis comes from and why is must be 1 to affect columns. To understand, just look at the .shape output:

```
df_movies.shape
```



```
Out[47]: (1000, 11)
```

This is a tuple that represents the shape of the DataFrame, i.e. 1000 rows and 11 columns.

Note that the rows are at index 0 of this tuple and columns are at index 1.

Therefore axis=1 affects columns. This comes from NumPy and is a great example of why learning NumPy is worth your time.

Missing values: imputation

- Imputation is a conventional technique used to keep valuable data that have null values.
- There may be instances where dropping every row with a null value removes too big a chunk from your dataset, so instead we can impute that null with another value, usually the **mean** or the **median** of that column.

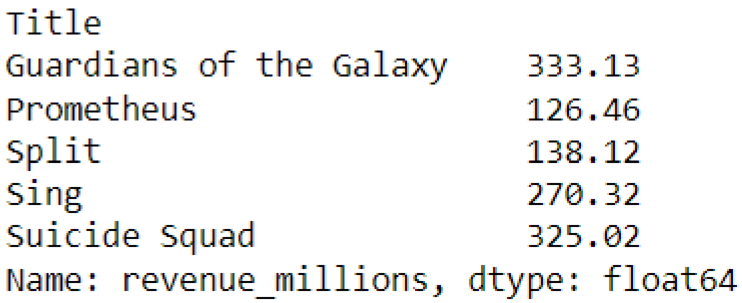
Missing values: imputation

- Let's look at imputing the missing values in the revenue_millions column. First extract that column into its own variable:

```
revenue = df_movies['revenue_millions']
```

- revenue now contains a Series:

```
revenue.head()
```

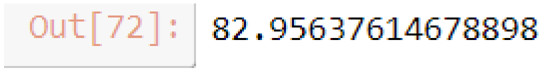


```
Title
Guardians of the Galaxy    333.13
Prometheus                 126.46
Split                      138.12
Sing                       270.32
Suicide Squad              325.02
Name: revenue_millions, dtype: float64
```

Missing values: imputation

- Impute the missing values of revenue using the mean.
- First calculate the mean value:

```
revenue_mean = revenue.mean()
revenue_mean
```



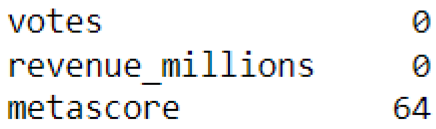
```
Out[72]: 82.95637614678898
```

- Next fill the nulls using ***.fillna()*** :

```
revenue.fillna(revenue_mean, inplace=True)
```

We have now replaced all nulls in revenue with the mean of the column.
Notice that by using inplace=True we have actually affected the original df_movies and not only the series.

```
df_movies.isnull().sum()
```



```
votes               0
revenue_millions    0
metascore          64
```

Data science

Pandas – Data manipulation techniques

Conditional Selection

Sorting

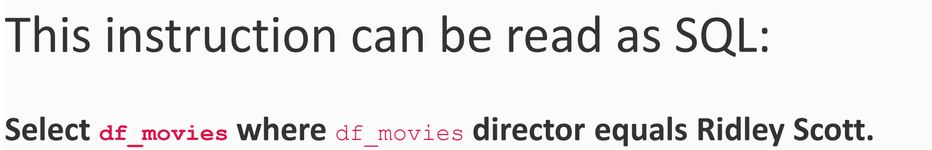
Grouping

Counting

Conditional selections

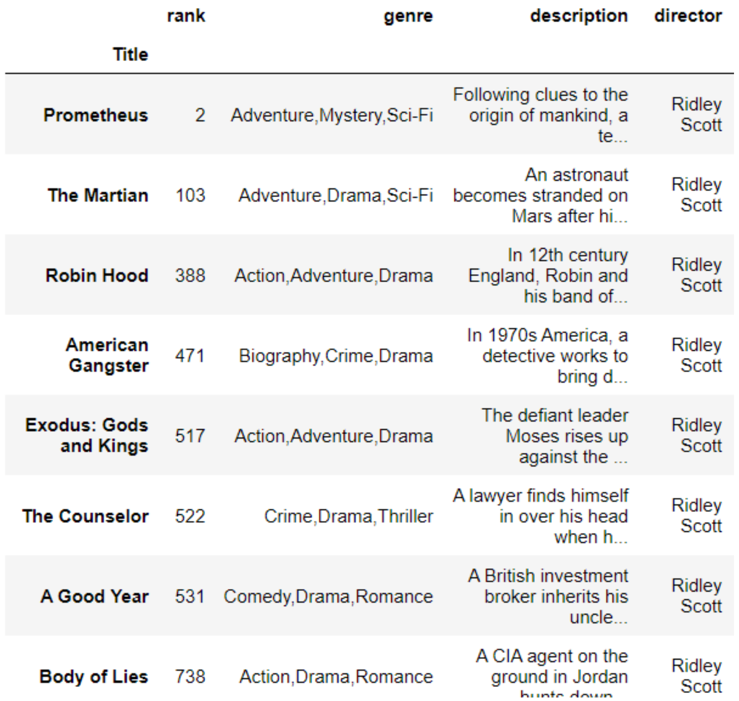
- We’ve gone over how to select columns and rows, but what if we want to make a conditional selection?
- For example, what if we want to filter our movies DataFrame to show only films directed by Ridley Scott?

```
df_movies[df_movies['director'] == 'Ridley Scott']
```



This instruction can be read as SQL:

Select `df_movies` where `df_movies` director equals Ridley Scott.

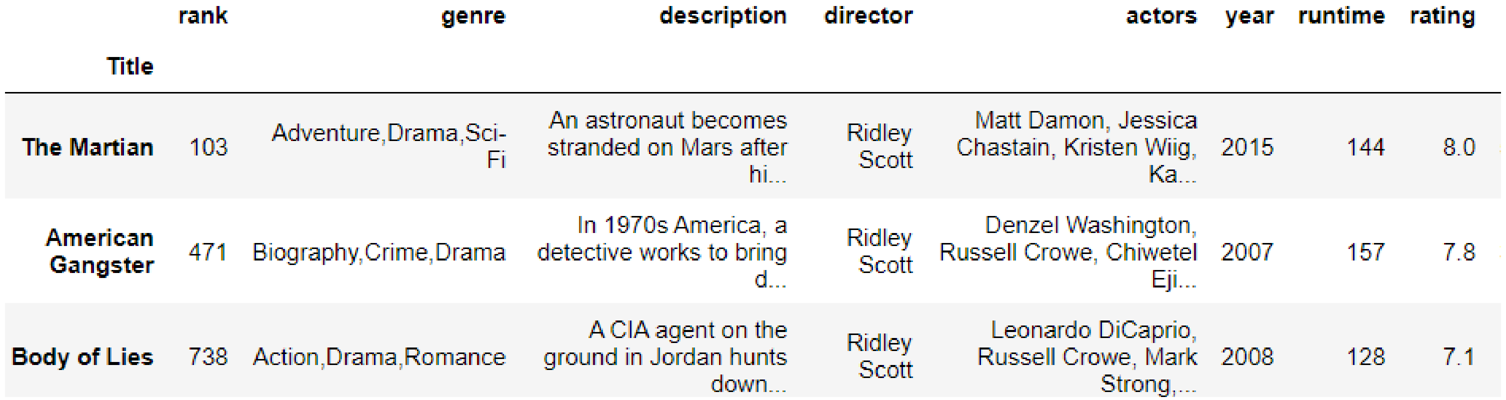


	rank	genre	description	director
Title				
Prometheus	2	Adventure,Mystery,Sci-Fi	Following clues to the origin of mankind, a te...	Ridley Scott
The Martian	103	Adventure,Drama,Sci-Fi	An astronaut becomes stranded on Mars after hi...	Ridley Scott
Robin Hood	388	Action,Adventure,Drama	In 12th century England, Robin and his band of...	Ridley Scott
American Gangster	471	Biography,Crime,Drama	In 1970s America, a detective works to bring d...	Ridley Scott
Exodus: Gods and Kings	517	Action,Adventure,Drama	The defiant leader Moses rises up against the ...	Ridley Scott
The Counselor	522	Crime,Drama,Thriller	A lawyer finds himself in over his head when h...	Ridley Scott
A Good Year	531	Comedy,Drama,Romance	A British investment broker inherits his uncle...	Ridley Scott
Body of Lies	738	Action,Drama,Romance	A CIA agent on the ground in Jordan hunts down	Ridley Scott

Conditional selections

- Conditions can be combined by using logical operators:
  - | for "or"
  - & for "and“
  - Each condition is written between ()
- E.g. movies directed by Ridley Scott with a rating above 7.0

```
df_movies[(df_movies['director'] == 'Ridley Scott') & (df_movies['rating'] > 7.0 ) ]
```

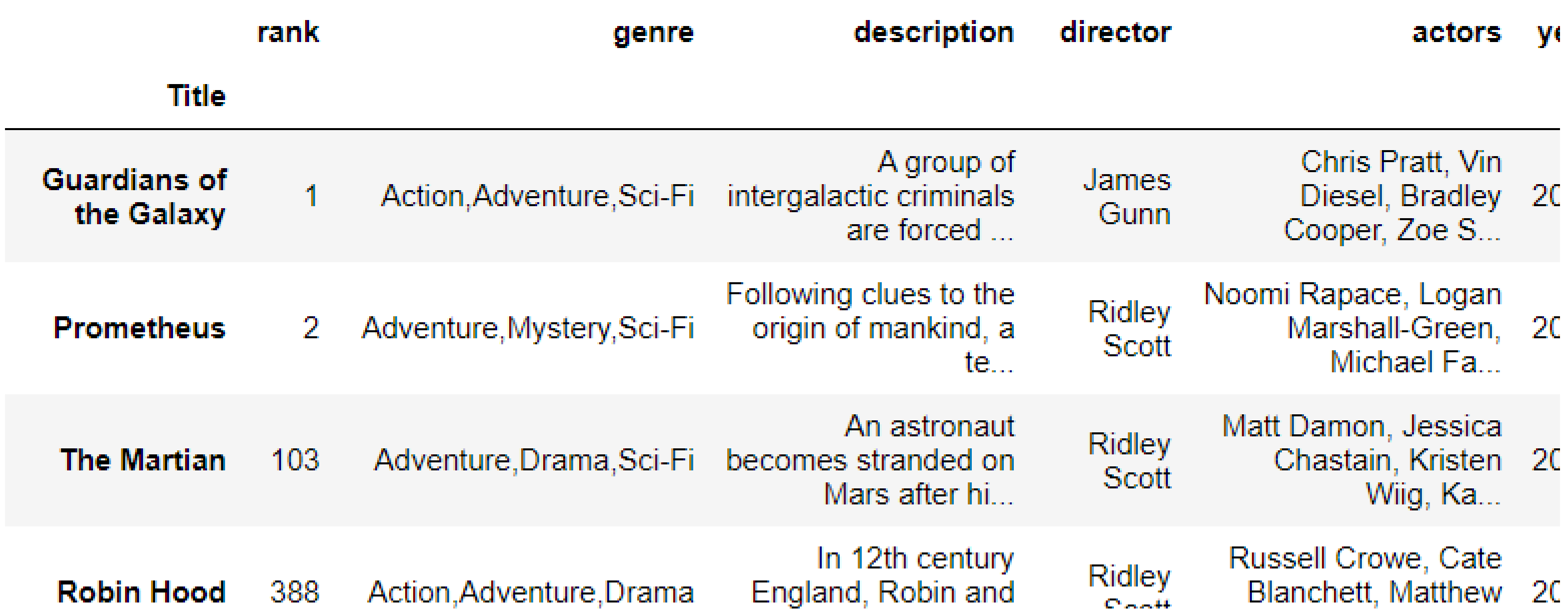


Title	rank	genre	description	director	actors	year	runtime	rating
The Martian	103	Adventure,Drama,Sci-Fi	An astronaut becomes stranded on Mars after hi...	Ridley Scott	Matt Damon, Jessica Chastain, Kristen Wiig, Ka...	2015	144	8.0
American Gangster	471	Biography,Crime,Drama	In 1970s America, a detective works to bring d...	Ridley Scott	Denzel Washington, Russell Crowe, Chiwetel Eji...	2007	157	7.8
Body of Lies	738	Action,Drama,Romance	A CIA agent on the ground in Jordan hunts down...	Ridley Scott	Leonardo DiCaprio, Russell Crowe, Mark Strong,...	2008	128	7.1

Conditional selections

- Frequently used in selection making is the ***isin()*** method (~ OR)
- E.g. movies directed by Ridley Scott OR James Gunn

```
df_movies[df_movies['director'].isin(['Ridley Scott','James Gunn'] )]
```

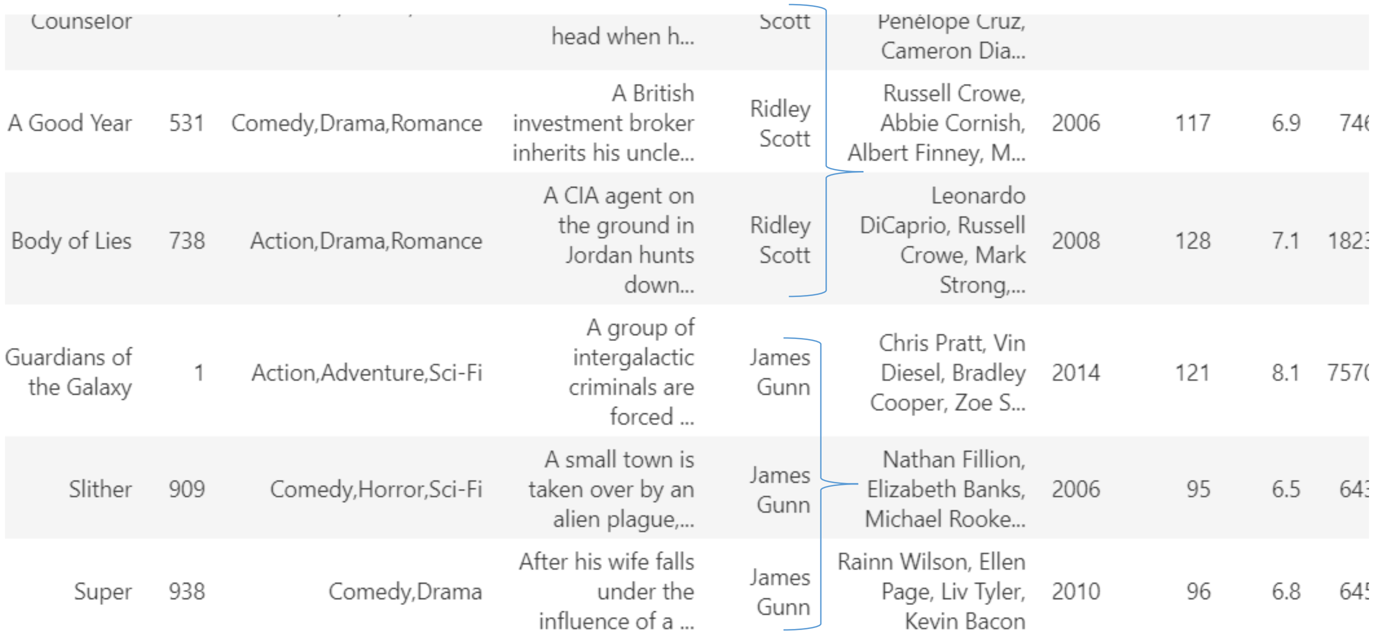


	rank	genre	description	director	actors	ye
Title						
Guardians of the Galaxy	1	Action,Adventure,Sci-Fi	A group of intergalactic criminals are forced ...	James Gunn	Chris Pratt, Vin Diesel, Bradley Cooper, Zoe S...	20
Prometheus	2	Adventure,Mystery,Sci-Fi	Following clues to the origin of mankind, a te...	Ridley Scott	Noomi Rapace, Logan Marshall-Green, Michael Fa...	20
The Martian	103	Adventure,Drama,Sci-Fi	An astronaut becomes stranded on Mars after hi...	Ridley Scott	Matt Damon, Jessica Chastain, Kristen Wiig, Ka...	20
Robin Hood	388	Action,Adventure,Drama	In 12th century England, Robin and	Ridley	Russell Crowe, Cate Blanchett, Matthew	20

Sorting

- In case you want to sort the movies by director, you can use the method ***sort_values()***

```
df_movies[df_movies['director'].isin(['Ridley Scott','James Gunn'] )].sort_values(by='director',ascending=False)
```

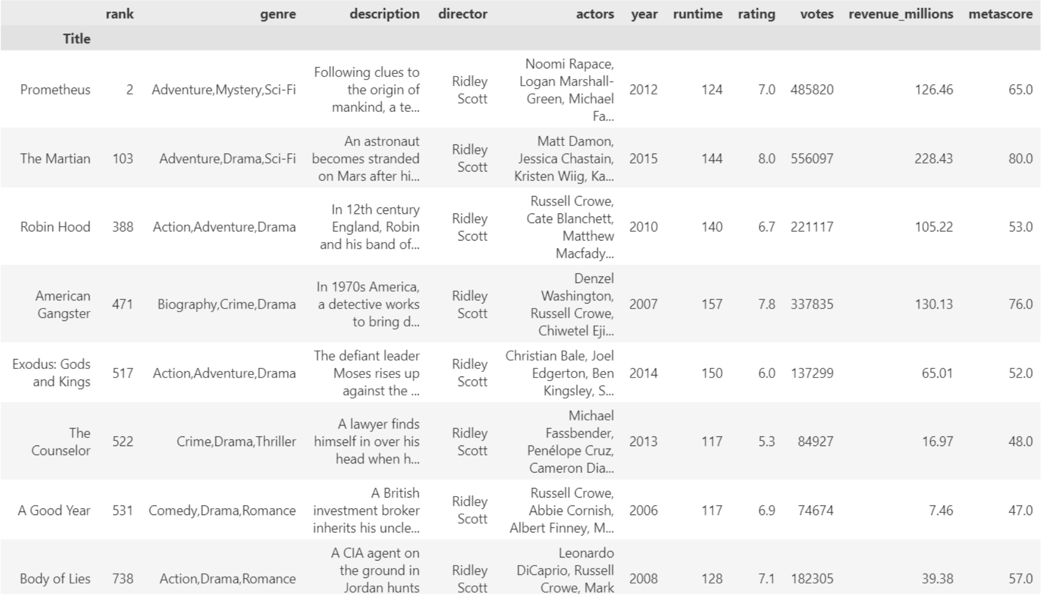


Counselor			head when h...	Scott	Penélope Cruz, Cameron Dia...				
A Good Year	531	Comedy,Drama,Romance	A British investment broker inherits his uncle...	Ridley Scott	Russell Crowe, Abbie Cornish, Albert Finney, M...	2006	117	6.9	74
Body of Lies	738	Action,Drama,Romance	A CIA agent on the ground in Jordan hunts down...	Ridley Scott	Leonardo DiCaprio, Russell Crowe, Mark Strong,...	2008	128	7.1	182
Guardians of the Galaxy	1	Action,Adventure,Sci-Fi	A group of intergalactic criminals are forced ...	James Gunn	Chris Pratt, Vin Diesel, Bradley Cooper, Zoe S...	2014	121	8.1	757
Slither	909	Comedy,Horror,Sci-Fi	A small town is taken over by an alien plague,...	James Gunn	Nathan Fillion, Elizabeth Banks, Michael Rooke...	2006	95	6.5	64
Super	938	Comedy,Drama	After his wife falls under the influence of a ...	James Gunn	Rainn Wilson, Ellen Page, Liv Tyler, Kevin Bacon	2010	96	6.8	64

Grouping

- The output “movies directed by Ridley Scot” can also be retrieved by using the methods ***groupby()*** and ***get_group()***

```
grouped = df_movies.groupby('director')
grouped.get_group('Ridley Scott')
```



	rank	genre	description	director	actors	year	runtime	rating	votes	revenue_millions	metascore
Title											
Prometheus	2	Adventure,Mystery,Sci-Fi	Following clues to the origin of mankind, a te...	Ridley Scott	Noomi Rapace, Logan Marshall-Green, Michael Fa...	2012	124	7.0	485820	126.46	65.0
The Martian	103	Adventure,Drama,Sci-Fi	An astronaut becomes stranded on Mars after hi...	Ridley Scott	Matt Damon, Jessica Chastain, Kristen Wiig, Ka...	2015	144	8.0	556097	228.43	80.0
Robin Hood	388	Action,Adventure,Drama	In 12th century England, Robin and his band of...	Ridley Scott	Russell Crowe, Cate Blanchett, Matthew Macfady...	2010	140	6.7	221117	105.22	53.0
American Gangster	471	Biography,Crime,Drama	In 1970s America, a detective works to bring d...	Ridley Scott	Denzel Washington, Russell Crowe, Chiwetel Eji...	2007	157	7.8	337835	130.13	76.0
Exodus: Gods and Kings	517	Action,Adventure,Drama	The defiant leader Moses rises up against the ...	Ridley Scott	Christian Bale, Joel Edgerton, Ben Kingsley, S...	2014	150	6.0	137299	65.01	52.0
The Counselor	522	Crime,Drama,Thriller	A lawyer finds himself in over his head when h...	Ridley Scott	Michael Fassbender, Penélope Cruz, Cameron Dia...	2013	117	5.3	84927	16.97	48.0
A Good Year	531	Comedy,Drama,Romance	A British investment broker inherits his uncle...	Ridley Scott	Russell Crowe, Abbie Cornish, Albert Finney, M...	2006	117	6.9	74674	7.46	47.0
Body of Lies	738	Action,Drama,Romance	A CIA agent on the ground in Jordan hunts	Ridley Scott	Leonardo DiCaprio, Russell Crowe, Mark	2008	128	7.1	182305	39.38	57.0

Counting

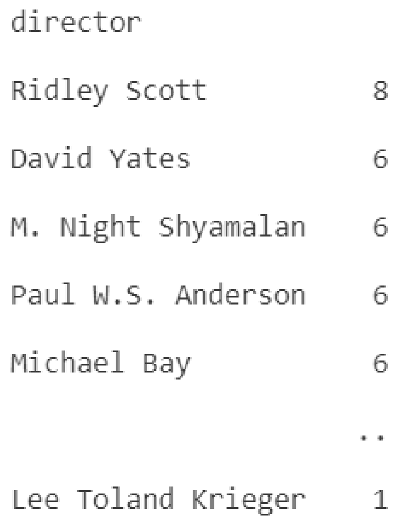
- To retrieve the number of movies directed by Ridley Scot, just use the function *len()*

```
grouped = df_movies.groupby('director')
len(grouped.get_group('Ridley Scott’))
```

```
8
```

- To retrieve the number of movies directed per director, use the method ***size()***

```
df_movies.groupby('director').size().sort_values(ascending=False)
```

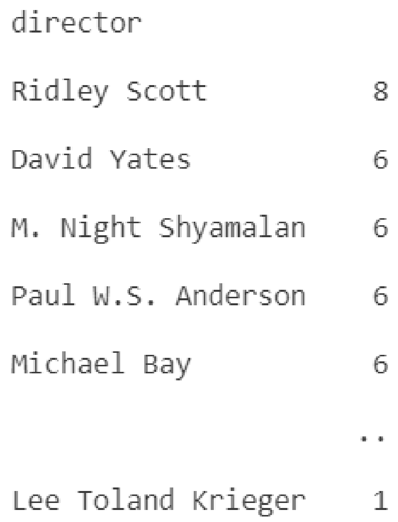


```
director
Ridley Scott          8
David Yates           6
M. Night Shyamalan    6
Paul W.S. Anderson    6
Michael Bay           6
                     ..
Lee Toland Krieger    1
```

Counting

- To count the number of movies directed by each director, you can immediately - without grouping - use the method ***value_counts()***

```
df_movies['director’].value_counts()
```



```
director
Ridley Scott          8
David Yates           6
M. Night Shyamalan    6
Paul W.S. Anderson    6
Michael Bay           6
                     ..
Lee Toland Krieger    1
```

Data science

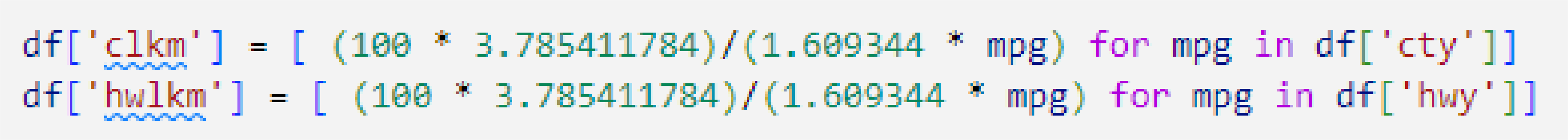
Pandas – Cleaning a dataset using an example

Cleaning a dataset

- Open the cars_cleaning notebook
  - Part 1 contains a detailed description of the opening and cleaning of some random dataset
  - We’ll go over the highlights in the following slides.

Create a calculated column

- The cars dataset has mileage in miles per gallon, not in liters per 100 km
- We can create a new column!



```
df['clkm'] = [ (100 * 3.785411784)/(1.609344 * mpg) for mpg in df['cty']]
df['hwlkm'] = [ (100 * 3.785411784)/(1.609344 * mpg) for mpg in df['hwy']]
```

- The formula is something you know (or lookup)
- Note how we used a list comprehension
  - By going over a column we created a list of the same number of rows as the dataframe
  - This ensures the data is stored nicely

Categorical versus numerical variables

- A **categorical** variable can have a finite number of values:
  - T-shirt sizes, places of embarkment for titanic, gender, …
- There are 2 types of categorical variables
  - Ordinal (ordered) “S/M/L/XL/XXL”
  - Nominal (unordered) “Male/Female”

- A **numerical** variable can have any quantitative value:
  - Height, weight, number of bikes you own, …
- There are 2 types of numerical variables
  - Discrete (finite number of possible values) : number of bikes
  - Continuous (infinite number of decimal values) : height, weight

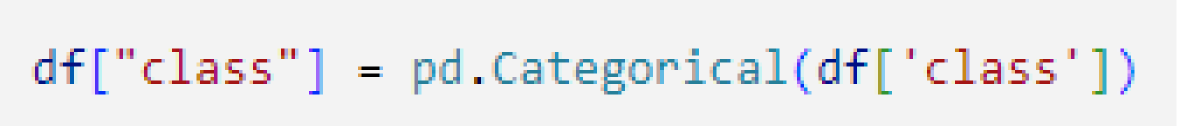
Categorical versus numerical variables

- Let’s say “age” is in the database

- Is a continuous numerical field if: You store the age as number with digits behind the comma (1,675; 3,54; 43,33453; …)
- Is a discrete numerical field if: You store it as an age (1, 3, 43, …)
- Is an ordered categorical field if: You can only have a limited number of ages, like kids in a kindergarten class (2, 3 or 4, but nothing else)
- Is never an unordered categorical field.

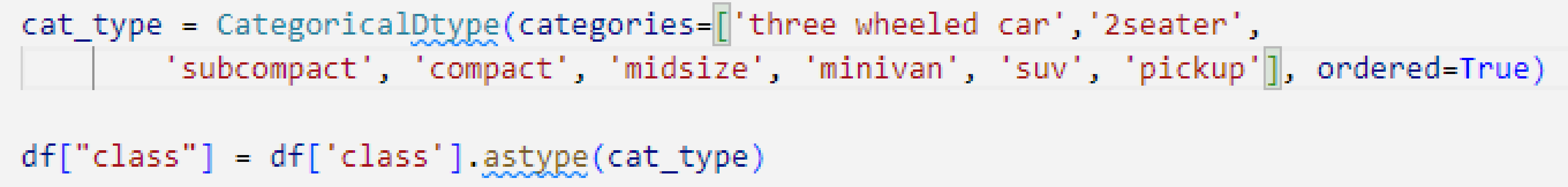
Categorical variables

- Why is this important?
- If you have categorical data and you tell pandas, pandas will take it into account when creating graphs
- How then?
    - Unordered:



```
df["class"] = pd.Categorical(df['class'])
```

    - Ordered:



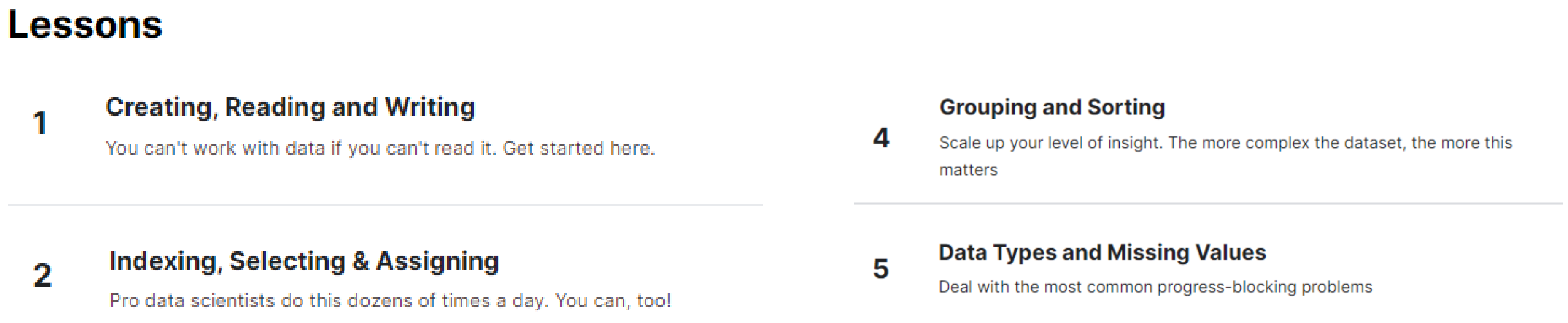
```
cat_type = CategoricalDtype(categories=['three wheeled car','2seater',
        'subcompact', 'compact', 'midsize', 'minivan', 'suv', 'pickup'], ordered=True)

df["class"] = df['class'].astype(cat_type)
```

Cars notebook

- Part 2 of the notebook is about selections and aggregations.
- Make sure you understand the basics well.

Exercises

- Kaggle course Pandas: https://www.kaggle.com/learn/pandas
- Make the exercises from lessons 1 - 2 - 4 – 5



Lessons

1 **Creating, Reading and Writing**
You can't work with data if you can't read it. Get started here.

2 **Indexing, Selecting & Assigning**
Pro data scientists do this dozens of times a day. You can, too!

4 **Grouping and Sorting**
Scale up your level of insight. The more complex the dataset, the more this matters

5 **Data Types and Missing Values**
Deal with the most common progress-blocking problems

- Notebook Flights

Resources

- https://www.learndatasci.com/tutorials/python-pandas-tutorial-complete-introduction-for-beginners/
- https://pandas.pydata.org/Pandas_Cheat_Sheet.pdf