Data Wrangling in Python

Erasmus Q-Intelligence B.V.

Data Science and Business Analytics
Programming

Introduction

Introduction

Data Wrangling is needed since

$\longrightarrow$ Data is often in a messy format

$\longrightarrow$ Data is often not readily usable for analysis and plotting

Data Wrangling often takes a considerable amount of time!

Content

References to Book

- Chapter 3
- Chapter 9

Online (https://r4ds.had.co.nz/)

- Chapter 5
- Chapter 12

Software requirements

Software requirements

```
import pandas as pd
```

$\longrightarrow$ Pandas and numpy contain the functions we primarily use in this lecture

Data set

Flight information of airports in New York in 2013

```
flights = pd.read_csv("flights.csv")
flights
```

	Unnamed: 0	year	month	day	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_time
0	1	2013	1	1	517.0	515	2.0	830.0	819
1	2	2013	1	1	533.0	529	4.0	850.0	830
2	3	2013	1	1	542.0	540	2.0	923.0	850
3	4	2013	1	1	544.0	545	-1.0	1004.0	1022
4	5	2013	1	1	554.0	600	-6.0	812.0	837

Data transformation with pandas

Five basic functions

The 5 most common functions in the Pandas-package:

⟶ `query()`: filter observations (rows) based on content

⟶ `sort_values()`: arrange observations (rows)

⟶ `filter()` and `loc()`: select variables (columns) based on names

⟶ `assign()`: mutate or add new variables (columns)

⟶ `agg()`: aggregate functions over subsets of the data

Combined with `groupby()`, these functions can also be evaluated on groups in the data

⟶ For example, find the average `dep_delay` for each day in the data set (data should be grouped by `day` and `month`)

query()

Example: *Find all flights on March 13th*

```
flights.query("month == 3 & day == 13")
```

	Unnamed: 0	year	month	day	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_time
147375	147376	2013	3	13	103.0	2355	68.0	457.0	340
147376	147377	2013	3	13	458.0	500	-2.0	648.0	648
147377	147378	2013	3	13	515.0	515	0.0	805.0	810
147378	147379	2013	3	13	525.0	530	-5.0	821.0	827
147379	147380	2013	3	13	541.0	545	-4.0	920.0	923

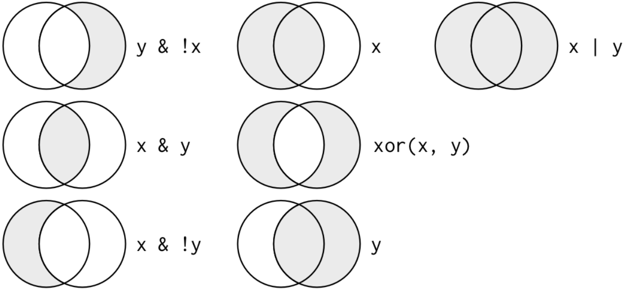
query() (2)

On the previous slide, we used the *Boolean*-notation &

⟶ &: 'and' - both condition should be TRUE to be selected

⟶ |: 'or' - any condition should be TRUE to be selected

⟶ !: 'not' - condition should be FALSE to be selected



`query()` (3)

Can also be used to check if a value is within a range

Example: *Find all flights in the 4th quarter*

```
flights_q4 = flights.query("month >= 10 & month <= 12")
```

In a first data analysis, you often want to find all observations with missings:

```
flights_no_dep_time = flights.query("dep_time.isnull()")
```

Exercises

Download and open the *Data_Wrangling_Exercises.pdf* file from Canvas, and do Exercises 1.1

sort_values()

Example: *Arrange* `flights` *by* `dep_delay`

```
flights_arranged = flights.sort_values('dep_delay')
```

	Unnamed: 0	year	month	day	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_time
89673	89674	2013	12	7	2040.0	2123	-43.0	40.0	2352
113633	113634	2013	2	3	2022.0	2055	-33.0	2240.0	2338
64501	64502	2013	11	10	1408.0	1440	-32.0	1549.0	1559
9619	9620	2013	1	11	1900.0	1930	-30.0	2233.0	2243
24915	24916	2013	1	29	1703.0	1730	-27.0	1947.0	1957

`sort_values()` (2)

If you want to arrange in descending order, use `ascending=0`

Example: *Arrange* `flights` *by* `dep_delay`, *in descending order*

```
flights_arranged = flights.sort_values('dep_delay',
                                        ascending=0)
```

$\longrightarrow$ Missing values (`NaN`) are always put last

Exercises

Open the *Data_Wrangling_Exercises.pdf* file from Canvas, and do Exercises 1.2

filter()

Example: *Select columns* `carrier`, `origin`, `dest`, `dep_delay`, `arr_delay` *from* `flights`

```
flights_selection = flights.filter(items=['carrier',
          'origin', 'dest', 'dep_delay', 'arr_delay'])
```

	carrier	origin	dest	dep_delay	arr_delay
0	UA	EWR	IAH	2.0	11.0
1	UA	LGA	IAH	4.0	20.0
2	AA	JFK	MIA	2.0	33.0
3	B6	JFK	BQN	-1.0	-18.0
4	DL	LGA	ATL	-6.0	-25.0
...	...	...	...	...	...

`filter()` (2)

You can select a range of columns by using `loc()` and ‘`:`‘

```
flights_selection = flights.loc[:, 'carrier':'dest']
```

Or specify which you *don't* want to select with ‘`!=`‘

```
flights_selection = flights.loc[:,
            flights.columns != 'year']
```

Columns can reordered by setting the order as follows

```
flights_reordered = flights[['flight', 'tailnum',
                            'carrier', 'origin', 'dest']]
```

Exercises

Open the *Data_Wrangling_Exercises.pdf* file from Canvas, and do Exercises 1.3

assign()

Example: *Create a column giving the gain in delay, and a column giving the gain per minute*

```
flights_selection = flights.loc[:, 'year':'air_time']

flights_gain = flights_selection.assign(gain =
    flights_selection['dep_delay']-
      flights_selection['arr_delay'])

flights_gain = flights_gain.assign(gain_per_minute =
    flights_gain['gain']/flights_selection['air_time'])
```

	year	month	day	dep_time	sched_dep_time	dep_delay	arr_time	\
0	2013	1	1	517.0	515	2.0	830.0	
1	2013	1	1	533.0	529	4.0	850.0	
2	2013	1	1	542.0	540	2.0	923.0	
3	2013	1	1	544.0	545	-1.0	1004.0	
4	2013	1	1	554.0	600	-6.0	812.0	

assign() (2)

⟶ the second assign can use the first assign

⟶ any function can be used, as long as the function results in a vector with the same length as the original data.frame

```
flights_gain = flights_gain.assign(gain_per_hour =
    flights_gain['gain_per_minute']*60)

flights_gain = flights_gain.assign(cul_gain =
    flights_gain['gain_per_hour'].expanding().mean())
```

Exercises

Open the *Data_Wrangling_Exercises.pdf* file from Canvas, and do Exercises 1.4

mean()

Example: *Get the mean of* `dep_delay`

```
flights_summary = flights['dep_delay'].mean()
```

```
12.639070257304708
```

Not so interesting. Better: Get the mean of `dep_delay` by day

groupby()

Example: *Get the mean of* `dep_delay` *by day*

```
flights_grouped = flights.groupby(['month', 'day'])
flights_summary = flights_grouped.dep_delay.mean()
```

```
month  day
1      1      11.548926
       2      13.858824
       3      10.987832
       4       8.951595
       5       5.732218
Name: dep_delay, dtype: float64
```

`agg()` **and** `groupby()`

Helpful functions for describing variables (by group using `groupby()`):

⟶ `mean()`, `median()`, `std()`: average, median or standard deviation of observations

⟶ `quantile()`: quantile of distribution

⟶ `count()`, `isna().sum()`: number of observations, and number of missings

agg() and groupby() (2)

```
flights_summary = flights_grouped.agg(
    delay = ('dep_delay', 'mean'),
    first_delay = ('dep_delay', 'first'),
    nr_flights = ('dep_delay', len))
```

		delay	first_delay	nr_flights
month	**day**			
1	**1**	11.548926	2.0	842
	2	13.858824	43.0	943
	3	10.987832	33.0	914
	4	8.951595	26.0	915
	5	5.732218	15.0	720
...	**...**	...	...	...

Exercises

Open the *Data_Wrangling_Exercises.pdf* file from Canvas, and do Exercises 1.5

`groupby()`, `query()` and `assign()`

Various functions can also be used together with `groupby`

Example: *Find, for each day, the 9 most delayed flights*

```
flights_most_delay = flights.sort_values(
    'arr_delay',ascending=False)
    .groupby(['month','days']).head(9)
```

`groupby()`, `query()` and `assign()`

`assign()` and `query()` can also be used together with `groupby`

Example: *Calculate, by destination, how much a flight adds to the total delay*

```
flights_delays = flights.query("arr_delay > 0")
flights_delays = flights_delays.assign(
    prop_delay = flights_delays['arr_delay']/
    sum(flights_delays['arr_delay']))
flights_delays.groupby('dest')
```

`sum(flights_delays[’arr_delay’])` is the sum over all positive delays, for that destination

(Intermezzo) Pipes

Pipes

Having to save every step to a `data.frame` in your environment is not very efficient

$\longrightarrow$ 'In between'-results are not of interest

Therefore, use pipes (adding functions after another) instead

```
flights_summary = (flights
            .groupby(['month', 'day'])
            .agg(delay = ('dep_delay', 'mean'),
                 first_delay = ('dep_delay', 'first'),
                 nr_flights = ('dep_delay', len))
            )
```

Pipes (2)

You do not have to repeat the data you work with and do not have to save every 'In between'-result!

⟶ Especially useful if many transformations (`query()`, `loc()`, `assign()`, `agg()`) have to be done

Example: *What does this code do?*

```
flights_gains = (flights
       .query("month == 3 & day == 13")
       .filter(['carrier', 'origin', 'dest',
                'dep_delay', 'arr_delay'])
       .assign(gain = flights.dep_delay -
                      flights.arr_delay)
       .groupby('dest')
       .agg(mean_gain = ('gain', 'mean'),
            max_gain = ('gain', 'max'),
            nr_flights = ('gain', len))
```

Pipes (3)

	mean_gain	max_gain	nr_flights
dest			
ALB	8.500000	11.0	2
ATL	3.060000	21.0	50
AUS	14.818182	31.0	11
BDL	13.000000	22.0	2
BHM	-1.000000	-1.0	1
...	...	...	...

Exercises

Open the *Data_Wrangling_Exercises.pdf* file from Canvas, and do Exercises 1.6 and 1.7

Data transformation (part 2)

Efficient coding

Applying the same function to a set of columns of the data frame

```
flights_across = flights_grouped.agg(
        {'dep_delay':'mean',
         'arr_delay':'mean',
         'air_time':'mean'})
```

Having to copy the same code (i.e. the 'mean' function) is not very efficient.

Better to first retrieve the desired columns from the groupby object and then call the mean function only once.

Efficient coding (2)

Example: *Find the average for* `dep_delay` , `arr_delay` *and* `air_time`

```
flights_across = flights_grouped[['dep_delay',
        'arr_delay', 'air_time']].agg('mean')
```

Efficient coding (3)

Several functions can be used for extracting columns

$\longrightarrow$ `str.startswith()`: ... to all columns that start with...

$\longrightarrow$ `str.isalpha()`: ... to all alphabetic columns

Efficient coding (4)

Example *Find the number of distinct values for all alphabetic columns in* `flights`

```
flights_across = flights[flights.columns
          [flights.dtypes == object]].nunique()
```

```
carrier          16
tailnum        4043
origin            3
dest            105
time_hour      6936
dtype: int64
```

Efficient coding (5)

Several function inputs can be used together, e.g. mean and median

Example *Find the mean and median, by day, for* `dep_delay` , `arr_delay` *and* `air_time`

```
flights_across = (flights
               .groupby(['month', 'day'])
               [['dep_delay', 'arr_delay', 'air_time']]
               .agg(['mean', 'median'])
               )
```

Exercises

Open the *Data_Wrangling_Exercises.pdf* file from Canvas, and do Exercise 1.8

Joining `dataframes`

Relational data

Often, a dataset from a database consists of multiple separate data

frames

⟶ To combine these tables, we use `joins` and `merges` `from` `panda`

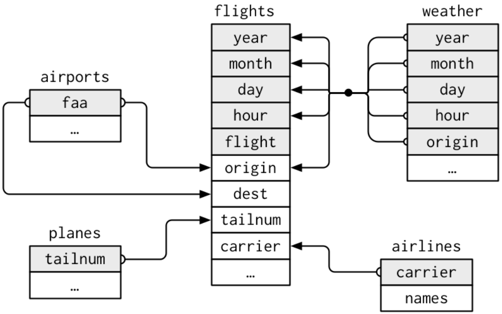
In the `nycflights13`-dataset, we have several `data.frames`

⟶ flights
⟶ airlines
⟶ airports
⟶ planes
⟶ weather

Relational data (2)

The relation between the data frames works with a *key*

$\longrightarrow$ A variable (column) which has unique elements and connects two `dataframes`

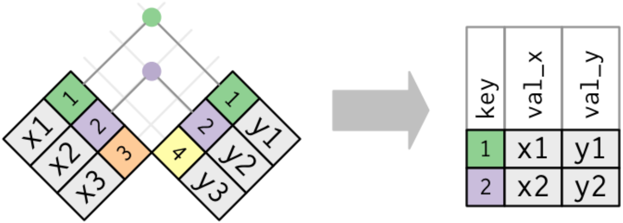


Example: *With* `tailnum` , `planes` *and* `flights` *can be joined.*

Inner-join

Add columns of `dataframe` together based on the `key`

With `inner_join`, keep only rows of which the `key` is in both `dataframes`

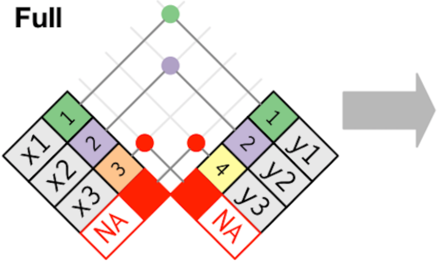


Full-join

Add columns of `dataframe` together based on the `key`

With `full_join`, keep all rows from both `dataframes` and join by `key`

⟶ If the value of `key` is not in the other `dataframe`, `NA` is added (Not Available)

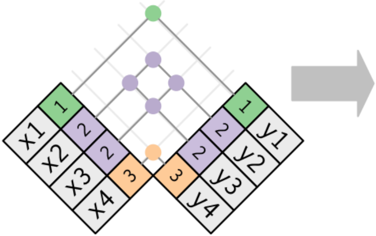


key	val_x	val_y
1	x1	y1
2	x2	y2
3	x3	NA
4	NA	y3

Left-join

Add columns of `dataframe` together based on the `key`

With `left_join`, keep all rows from the first `dataframe` and match observations with same `key` from the second `dataframe`



key	val_x	val_y
1	x1	y1
2	x2	y2
2	x2	y3
2	x3	y2
2	x3	y3
3	x4	y4

Example

Example: *Join* `flights` *and* `planes` *with* `key` `tailnum`, keeping all observations from `flights`

```
flights_select = flights.filter(['year', 'month',
       'day', 'tailnum', 'flight', 'carrier', 'origin',
       'dest', 'dep_time', 'arr_time'])
```

Example (2)

Example: *Join* `flights` *and* `planes` *with* `key tailnum`, keeping all observations from `flights`

```
print(flights_select.head(5))
```

	year	month	day	tailnum	flight	carrier	origin	dest	dep_time	arr_time
0	2013	1	1	N14228	1545	UA	EWR	IAH	517.0	830.0
1	2013	1	1	N24211	1714	UA	LGA	IAH	533.0	850.0
2	2013	1	1	N619AA	1141	AA	JFK	MIA	542.0	923.0
3	2013	1	1	N804JB	725	B6	JFK	BQN	544.0	1004.0
4	2013	1	1	N668DN	461	DL	LGA	ATL	554.0	812.0

Example (3)

Example: *Join* `flights` *and* `planes` *with* `key` `tailnum`, keeping all observations from `flights`

```
planes = pd.read_csv("planes.csv")
print(planes.head(5))
```

```
   Unnamed: 0 tailnum    year                    type      manufacturer  \
0           1  N10156  2004.0  Fixed wing multi engine           EMBRAER
1           2  N102UW  1998.0  Fixed wing multi engine  AIRBUS INDUSTRIE
2           3  N103US  1999.0  Fixed wing multi engine  AIRBUS INDUSTRIE
3           4  N104UW  1999.0  Fixed wing multi engine  AIRBUS INDUSTRIE
4           5  N10575  2002.0  Fixed wing multi engine           EMBRAER

       model  engines  seats  speed     engine
0  EMB-145XR        2     55    NaN  Turbo-fan
1   A320-214        2    182    NaN  Turbo-fan
2   A320-214        2    182    NaN  Turbo-fan
3   A320-214        2    182    NaN  Turbo-fan
4  EMB-145LR        2     55    NaN  Turbo-fan
```

Example (4)

Example: *Join* `flights` *and* `planes` *with* `key tailnum`, keeping all observations from `flights`

```
flights_planes = flights_select.merge(planes,
                    on='tailnum', how='left')
```

```
print(flights_planes.head())
```

	year_x	month	day	tailnum	flight	carrier	origin	dest	dep_time
0	2013	1	1	N14228	1545	UA	EWR	IAH	517.0
1	2013	1	1	N24211	1714	UA	LGA	IAH	533.0
2	2013	1	1	N619AA	1141	AA	JFK	MIA	542.0
3	2013	1	1	N804JB	725	B6	JFK	BQN	544.0
4	2013	1	1	N668DN	461	DL	LGA	ATL	554.0

Exercises

Open the *Data_Wrangling_Exercises.pdf* file from Canvas, and do Exercise 1.9

Data transformation

Tidy data

Data is tidy if

- Every variable has its own
- columnry observations has its own
- row Every value has its own cell

Tidy data

	Unnamed: 0	country	year	cases	population
0	1	Afghanistan	1999	745	19987071
1	2	Afghanistan	2000	2666	20595360
2	3	Brazil	1999	37737	172006362
3	4	Brazil	2000	80488	174504898
4	5	China	1999	212258	1272915272
5	6	China	2000	213766	1280428583

Data is tidy, since all variables and observations have their own columns and rows

Tidy data

	Unnamed: 0	country	year	type	count
0	1	Afghanistan	1999	cases	745
1	2	Afghanistan	1999	population	19987071
2	3	Afghanistan	2000	cases	2666
3	4	Afghanistan	2000	population	20595360
4	5	Brazil	1999	cases	37737
5	6	Brazil	1999	population	172006362
6	7	Brazil	2000	cases	80488
7	8	Brazil	2000	population	174504898
8	9	China	1999	cases	212258
9	10	China	1999	population	1272915272
10	11	China	2000	cases	213766
11	12	China	2000	population	1280428583

Data is untidy, since variables `cases` and `population` are together in one column

Tidy data

Data is tidy if

- Every variable has its own column
- Every observations has its own row
- Every value has its own cell

Tidy data is

- consistently saved
- easier to manipulate

Often, you will need untidy data to present your data in plots or tables

Gather data spread over columns

Often, a variable is spread over more than one column:

	Unnamed: 0	country	1999	2000
0	1	Afghanistan	745	2666
1	2	Brazil	37737	80488
2	3	China	212258	213766

Columns ‘`2019`‘ and ‘`2020`‘ are actually values of the variable `year`

Gather data spread over columns

Use the function `pd.melt` to transform the dataframe to a long format with two non-identifier columns: 'variable' and 'value'.

```
table_gathered = pd.melt(table4a, id_vars =['country'],
                value_vars =['1999', '2000'],
                var_name ='year', value_name ='cases')
```

Function arguments:

- `data` - `dataframe` to be used
- `id_vars` column(s) to use as identifier variables
- `value_vars` - column(s) to unpivot. If not specified, uses all columns that are not set as id_vars.
- `var_name` - name to use for the 'variable' column.
- `value_name` - name to use for the 'value' column.

Gather data spread over columns

	country	year	cases
0	Afghanistan	1999	745
1	Brazil	1999	37737
2	China	1999	212258
3	Afghanistan	2000	2666
4	Brazil	2000	80488
5	China	2000	213766

Spread data gathered in column

Often, more than one variable is gathered in one column:

	Unnamed: 0	country	year	type	count
0	1	Afghanistan	1999	cases	745
1	2	Afghanistan	1999	population	19987071
2	3	Afghanistan	2000	cases	2666
3	4	Afghanistan	2000	population	20595360
4	5	Brazil	1999	cases	37737
5	6	Brazil	1999	population	172006362
6	7	Brazil	2000	cases	80488
7	8	Brazil	2000	population	174504898
8	9	China	1999	cases	212258
9	10	China	1999	population	1272915272
10	11	China	2000	cases	213766
11	12	China	2000	population	1280428583

`cases` and `population` are actually separate variables, gathered in column `type`

Spread data gathered in column

Use the function `pivot`:

```
table_spread = pd.pivot(table2,
    index = ['country', 'year'],
    columns='type', values='count')
```

Function arguments:

- `data` - `dataframe` to be used
- `index` - column(s) to use to make new frame's index
- `columns` - column(s) with the new spread out column names
- `values` - column(s) with the values for the new spread out columns

Spread data gathered in column

```
type                  cases  population
country     year
Afghanistan 1999        745    19987071
            2000       2666    20595360
Brazil      1999      37737   172006362
            2000      80488   174504898
China       1999     212258  1272915272
            2000     213766  1280428583
```

Exercises

Open the *Data_Wrangling_Exercises.pdf* file from Canvas, and do Exercises 1.10 and 1.11

Conclusion

Conclusion

- $\longrightarrow$ Data is often in a messy format
- $\longrightarrow$ Data is often not readily usable for analysis and plotting
- $\longrightarrow$ `pandas` has handy functions to wrangle your data to make it usable
  - $\longrightarrow$ `filter`, `loc` and `query` to get subsets of the data
  - $\longrightarrow$ `assign`, `sort_values`, `agg` and `groupby` to manipulate your data

Data Wrangling often takes a considerable amount of time!