Teaching the teacher: Python
Day 2 – Morning: Pandas

This morning

Python modules

What are Pandas

Basic operations

Data wrangling (now + afternoon)

Recap last week

- What are the different data types in Python?
- What is the difference between 5 and ”5”?
- What is the difference between a list and a dictionary?
- Do I have to write my own functions?

Functions

Name of the function - arbitrary
- no number at the beginning
- no spaces
- not used by built-in functions

Arguments that you pass along:
- this is what addone will use
- as many as you want
- arbitrary naming
- *can be skipped*

Creating (defining a function)

Indicates that the indented block with the definition of what to do follows

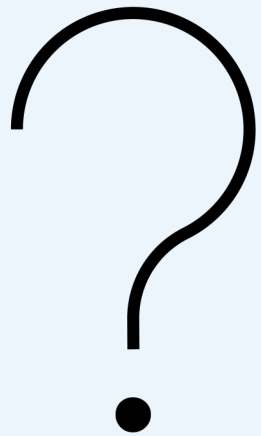
```
def addone(number):
    new_number = number + 1
    return new_number
```

Define what the function should do (notice indent!)

```
result = addone(8)
```

Here, we **call** (= run, execute) the addone function

Questions about last week



Python modules

Modules, packages, libraries

"Code library" that you can import

```
import pandas as pd
```

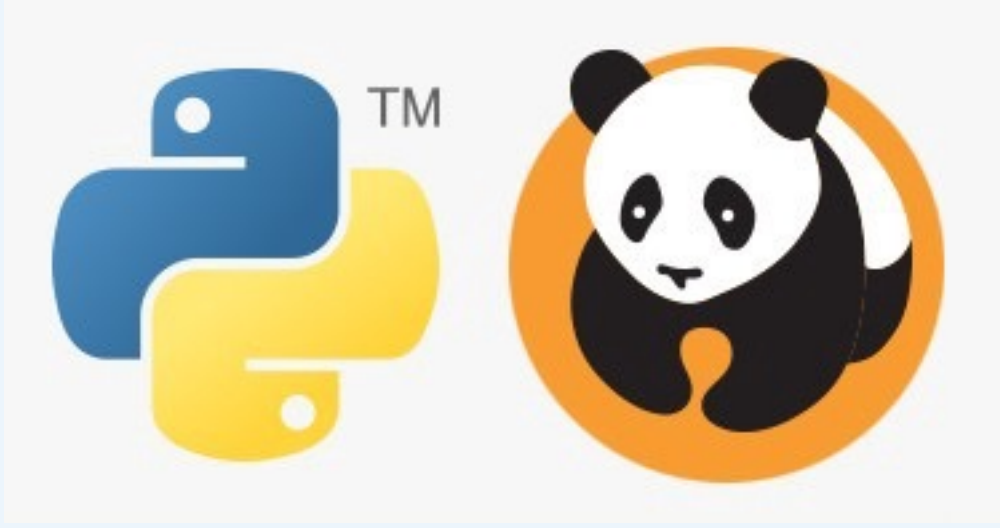
Several built-in modules

https://docs.python.org/3/library/

Pip install one's written by others

```
pip install …
```

Pandas



Pandas

"pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language."

Pd.DataFrames:

- Objects stroing tabular data in rows and columns
- Colums and rows have names
- Many built-in methods for data wrangling and basic analyses

- *SPSS, Excel, R…*

Pandas

- pandas is a fast, powerful, flexible and easy to use open-source data analysis and manipulation package, built on top of Python
- Pandas is generally imported as pd via
  import pandas as pd

Pandas

- Pandas can be used for
  - Creating dataframes
  - Reading and writing data (xlsx, csv, sav, json, pkls, etc.)
  - Filtering, selecting and renaming dataframe data
  - Merging dataframes
  - In brief: pandas is an excellent tool for **data wrangling**

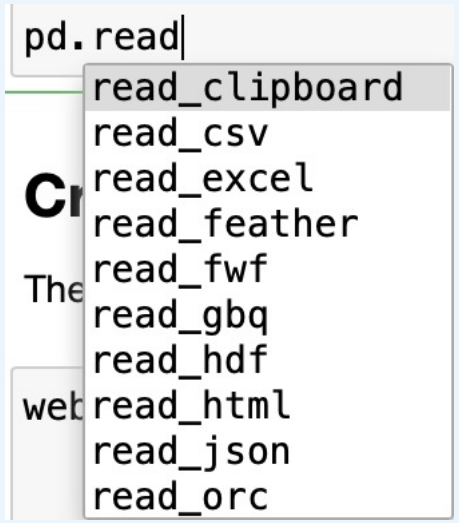
How do you make a panda?

Make a dataframe from a dict or list

```
df = pd.DataFrame(list-of-lists,
dict etc.)
```

Read from an existing file

```
df = pd.read
```



Reading dataframes from c(omma)s(eparated)v(alue)s

- Use `pd.read_csv("dataset", sep="…", encoding="…")`
- Separators – if not specified, the dataset is likely to look funny!
  - `, ; \ /`
- Encodings – numerical (binary and machine readable) representations of characters
  - **utf-8**, **ascii**, non-ascii, unicode…, lots of other old stuf…

Writing csv is also possible

- Use `pd.write_csv("dataset", sep="…", encoding="…")`
- Separators – if not specified, the dataset is likely to look funny!
  - `, ; \ /`
- Encodings – numerical (binary and machine readable) representations of characters
  - **utf-8**, **ascii**, non-ascii, unicode…, lots of other old stuf…

```
videos = pd.read_csv('videolist_search500_2020_01_25-12_34_16.tab')
```

```
---------------------------------------------------------------------------
ParserError                               Traceback (most recent call last)
<ipython-input-3-70a34e0eabd1> in <module>
----> 1 videos = pd.read_csv('videolist_search500_2020_01_25-12_34_16.tab')

/anaconda3/lib/python3.7/site-packages/pandas/io/parsers.py in parser_f(filepath_or_buffer, sep, delimiter, header, names, index_col, usecols, squeeze, prefix, mangle_dupe_cols, dtype, engine, converters, true_values, false_values, skipinitialspace, skiprows, skipfooter, nrows, na_values, keep_default_na, na_filter, verbose, skip_blank_lines, parse_dates, infer_datetime_format, keep_date_col, date_parser, dayfirst, iterator, chunksize, compression, thousands, decimal, lineterminator, quotechar, quoting, doublequote, escapechar, comment, encoding, dialect, tupleize_cols, error_bad_lines, warn_bad_lines, delim_whitespace, low_memory, memory_map, float_precision)
    700                     skip_blank_lines=skip_blank_lines)
    701
--> 702         return _read(filepath_or_buffer, kwds)
    703
    704     parser_f.__name__ = name

/anaconda3/lib/python3.7/site-packages/pandas/io/parsers.py in _read(filepath_or_buffer, kwds)
    433
    434     try:
--> 435         data = parser.read(nrows)
    436     finally:
    437         parser.close()
```

```
videos = pd.read_csv('videolist_search500_2020_01_25-12_34_16.tab', sep='\t')
videos
```

	position	channelId	channelTitle	videoId	publishedAt	publishedAtSQL	videoTitle	videoDescription	videoCategoryId	videoCategoryLabel	...
0	1	UCIALMKvObZNtJ6AmdCLP7Lg	Bloomberg Markets and Finance	sGHq_EwXDn8	2020-01-24T04:15:28.000Z	2020-01-24 04:15:28	Australia's Policies Going in Wrong Direction ...	Jan.23 -- Michael Mann, distinguished professo...	25	News & Politics	...
1	2	UCb1Ti1WKPauPpXkYKVHNpsw	LBC	PRtn1W2RAVU	2020-01-23T10:32:38.000Z	2020-01-23 10:32:38	Nigel Farage compares President Trump and Prin...	This is Nigel Farage's reaction to President T...	25	News & Politics	...
2	3	UC-SJ6nODDmufqBzPBwCvYvQ	CBS This Morning	2CQvBGSiDvw	2019-12-23T13:38:55.000Z	2019-12-23 13:38:55	Climate change in the 2020s: What impacts to e...	In our series The 2020's, we're exploring the ...	25	News & Politics	...
3	4	UCcyq283he07B7_KUX07mmtA	Business Insider	Cbwv1jg4gZU	2020-01-22T22:28:34.000Z	2020-01-22 22:28:34	Solution To Climate Change Is To Make It Profi...	Environmental problems rose to the top of the ...	25	News & Politics	...

In what situations would you choose for approaches from last week (for-loop) to read or write tabular data and in which would you use pandas?
Pros? Cons?

When to use Panda dataframes

Pandas	Other formats
• Tabular data • Easy to have a look • Data wrangling, descriptives… • R/SPSS/Stata user-friendly	• Non-tabular data • No clear cases (rows) and variables (columns) • Large size of data • Long operations (avoid crashing)

Pandas – basic operations

- dataframe.columns
- dataframe.isna().sum()
- dataframe.head()
- dataframe.describe().transpose()
- dataframe.groupby(“x”)

Data wrangling 101

1) Upon loading a dataset, use **.head()** to examine the data
2) Print all columns names using **.columns** to see all the columns in the dataset
3) Use **.isna().sum()** to see which columns contain missing values (NaN)
4) Check if all column types are as expected with **.dtypes**
    1) **Refresher: What are the various types in Python and what can each be used for?**

Make a game plan based on

1) Which columns do you need?
2) Which values do you expect?
3) How will you deal with missing values if any?
4) How will you deal with unexpected column types?

Make procedures like this part of your standard practice to save yourself a lot of trouble later down the line!

Operations on columns

"Recoding columns"

Built-in functions of Pandas

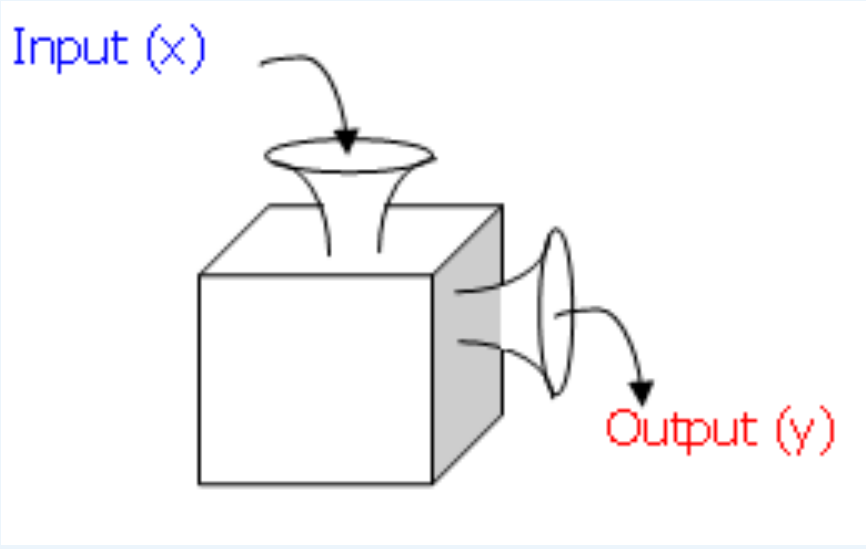
```
df['new_column'] = pd.to_numeric(df['old_column'])
```

Applying external packages & own functions

```
df['new_date'] = df['old_date'].apply(pd.to_datetime)
```

Applying fuctions

```
df['new_column'] = df['source_column'].apply(function)
```



Subsetting and slicing

list: `list[0:5]` – 0,1,2,3,4

Dict: `mydict[“mijnkey”]` – value associated with a key

Dataframe:

- `df[[‘col2’, ‘col2’]]` – two columns from a dataframe
- `df[df[‘col1’] == ‘value’]` – only rows that score 1 on col1
- `df[df[‘col2’]>0]` – only rows that score more than 0 on col 2

Filtering/subsetting and selecting

1) Filtering/subsetting – selecting parts of your dataframe
   1) Only some columns
   2) Only some rows
   3) Only some values

```
my_df = my_df[[“column_one”, “column_two”, “column_three”]]
```

Filtering/subsetting and selecting

1) Filtering/subsetting – selecting parts of your dataframe
   1) Only some columns
   2) Only some rows
   3) Only some values

```
my_df = my_df[50:100]
```

Filtering/subsetting and selecting

1) Filtering/subsetting – selecting parts of your dataframe
   1) Only some columns
   2) Only some rows/values
   3) Only some values

```
my_df = my_df[(my_df["colname"] > 209) & (my_df["colname"] < 700)]
```

Operations on a specific column

```
my_df[‘column_one’] = my_df[‘column_one’].astype(str)
```

Pandas to combine strings

```
my_df[‘column_one’] = my_df[‘column_one’] + my_df[‘column_two’]
```

Pandas for math

```
my_df[‘column_one’] = my_df[‘numbers’] + my_df[‘other_numbers!’]
```

Subsetting with iloc

Choose column or row

`iloc[]` – number of the column/row

`.loc[]` – name of the column

Teaching the teacher: Python
Day 2 – Afternoon: Stats+visualisations

This afternoon

Data wrangling – continued

Basic statistics in Python
- descriptives, correlations (more on Friday)

Data visualisations
- univariate
- bivariate

Data wrangling

Concat and merging

Two situations:

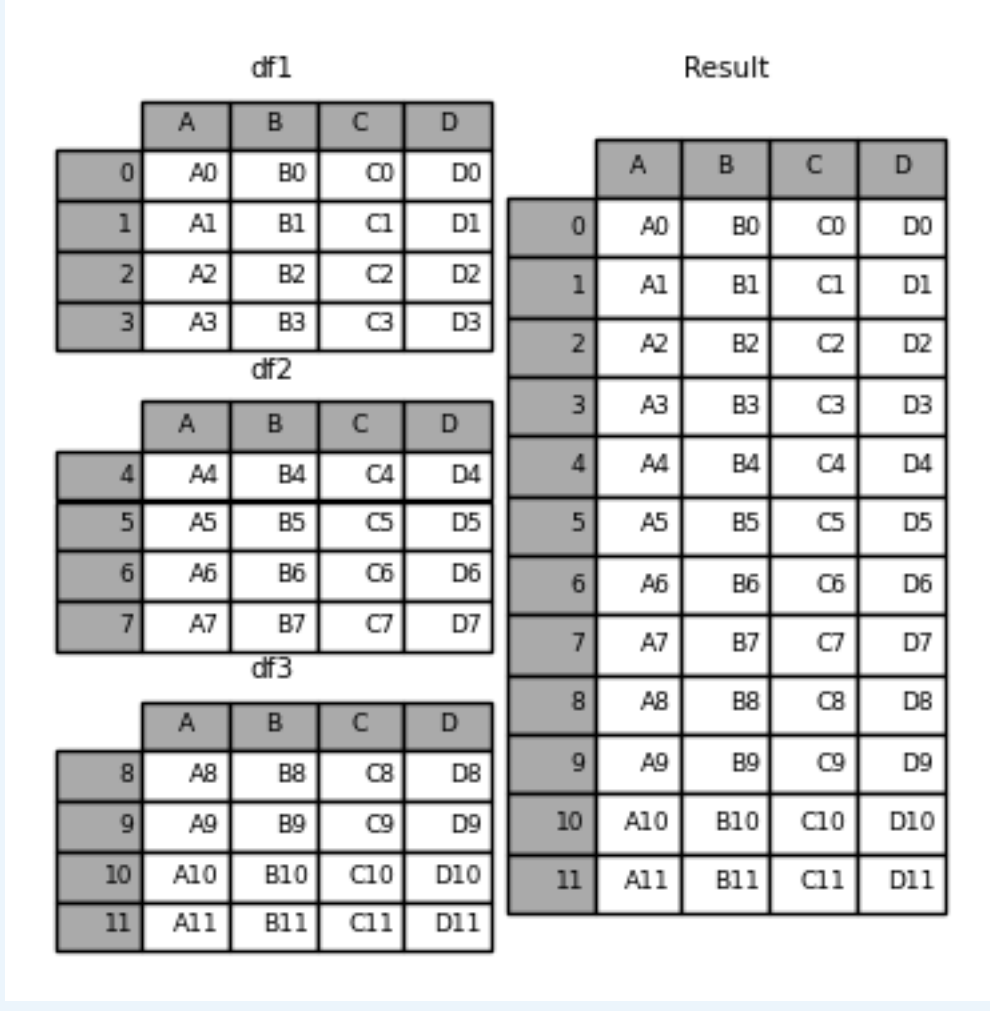
1. Two datasets that you can merge together on a unique identifier
   → merge
1. Two datasets that have the same variables/columns; you can “add cases”
   → concat

Add cases

Concat

```
frames = [df1, df2, df3]

result = pd.concat(frames)
```



	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3

	A	B	C	D
4	A4	B4	C4	D4
5	A5	B5	C5	D5
6	A6	B6	C6	D6
7	A7	B7	C7	D7

	A	B	C	D
8	A8	B8	C8	D8
9	A9	B9	C9	D9
10	A10	B10	C10	D10
11	A11	B11	C11	D11

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3
4	A4	B4	C4	D4
5	A5	B5	C5	D5
6	A6	B6	C6	D6
7	A7	B7	C7	D7
8	A8	B8	C8	D8
9	A9	B9	C9	D9
10	A10	B10	C10	D10
11	A11	B11	C11	D11

```
df_bezoekers
```

	bezoeker	source	tijd	paginas
0	476	social	360	3
1	467	organic	36	2
2	234	organic	12	1
3	626	search	98	2
4	964	social	2	1
5	125	social	68	3
6	784	search	43	2
7	346	search	87	3
8	567	organic	276	3
9	345	social	45	2
10	246	social	8	1
11	865	search	78	2
12	135	search	2	1
13	357	search	35	3
14	126	search	43	2
15	765	social	77	3

```
df_info
```

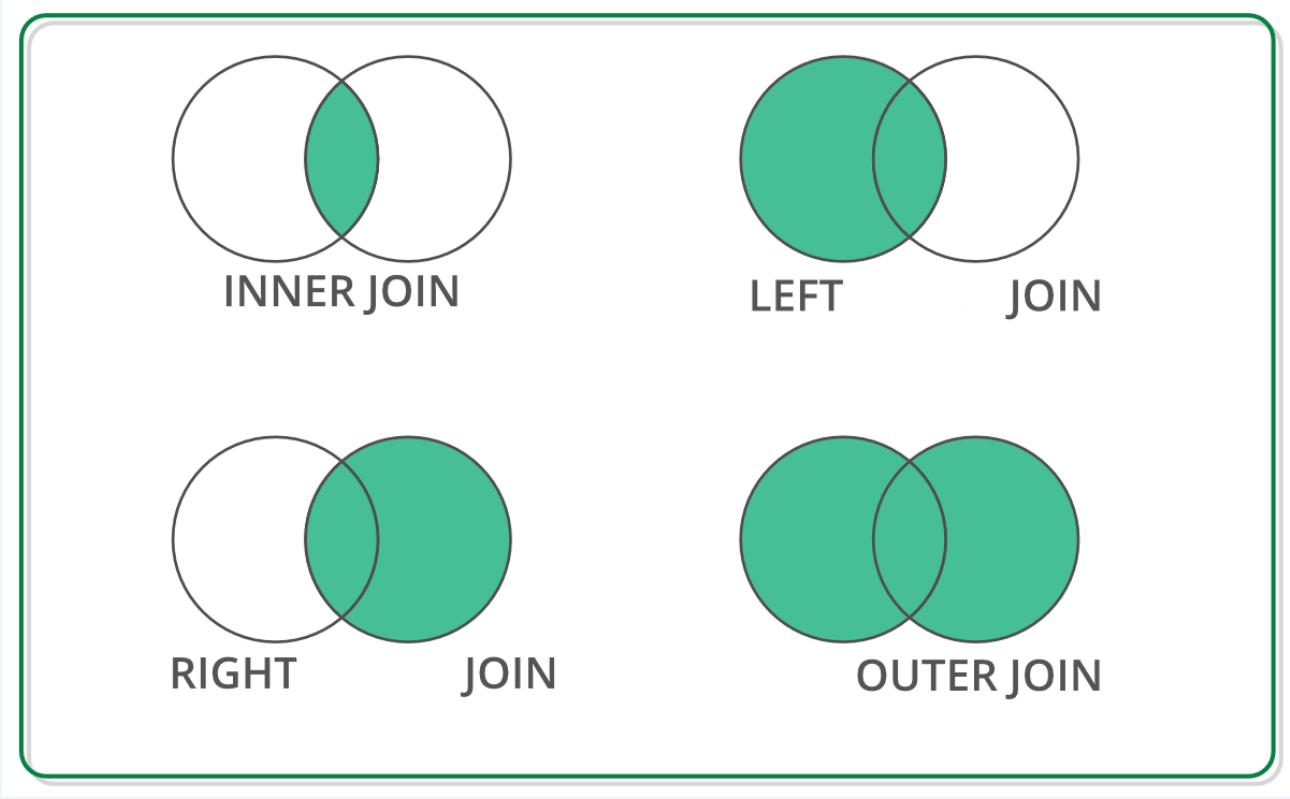
	ID	Geslacht	Leeftijd	Postcode	Subscribed
0	476.0	man	23.0	1019.0	True
1	467.0	man	56.0	3842.0	False
2	234.0	man	32.0	7539.0	True
3	626.0	vrouw	56.0	8163.0	False
4	964.0	vrouw	32.0	7815.0	True
...	...	...	...	...	...
99	NaN	NaN	NaN	NaN	NaN
100	NaN	NaN	NaN	NaN	NaN
101	NaN	NaN	NaN	NaN	NaN
102	NaN	NaN	NaN	NaN	NaN
103	NaN	NaN	NaN	NaN	NaN

Merge

Merge

→ Add columns

```
df3 = df1.merge(df2, on="ID", how=?)
```



Things to keep in mind when merging

- There must be *at least one* shared column
- The shared column might be named differently across the two datasets
  - Rename in one dataset or specify column names when merging/joining
- Which observations do you want the final dataset to contain?
- Will your final dataset include NaN values and is that a problem?

Aggregation

Change ”unit of analysis”

Used after “groupby”

Takes functions as an argument

```
df.groupby('month').agg(np.mean, sum)
```

Flexible, to use with plotting etc.

Basic stats

Stats in Python?

- All standard methods (similar to SPSS, stata)
- More advanced methods (time series, SEM)
- Some models – R is better

Useful packages for stats

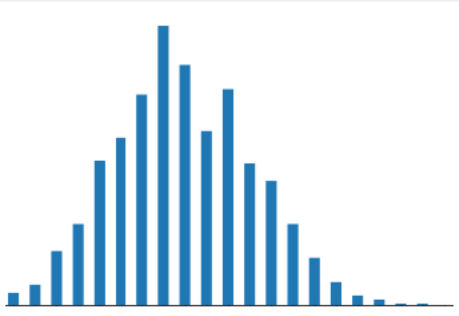
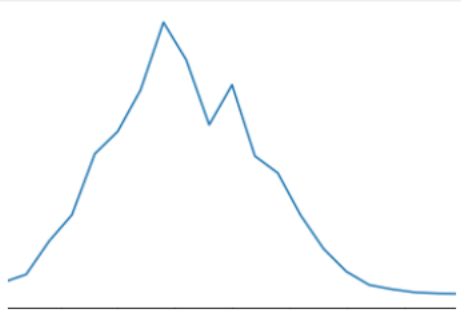
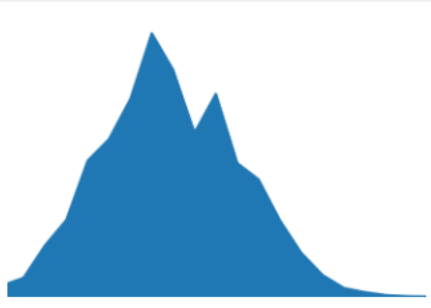
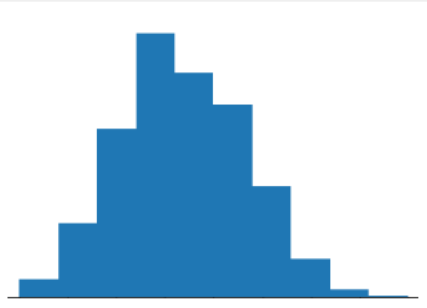
Numpy	many useful functions, e.g., mean, SD, correlations
Scipy	
Statsmodels	Statistical models, e.g., regressions, time series
Matplotlib	Plots
Seaborn	Nice plots plots
Plotly	Interactive plots

Useful funcations in Pandas

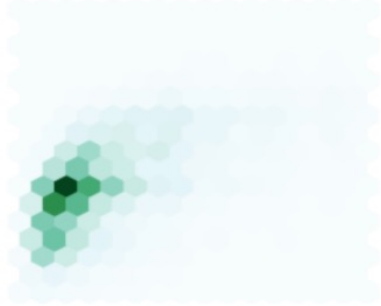
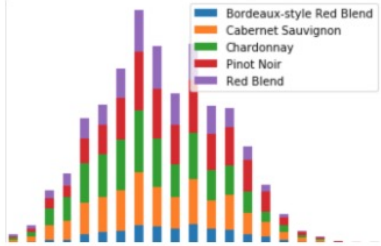
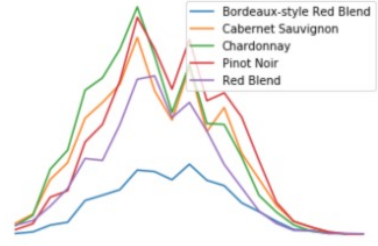
Function	Description
count()	Number of non-null observations
sum()	Sum of values
mean()	Mean of Values
median()	Median of Values
mode()	Mode of values
std()	Standard Deviation of the Values
min()	Minimum Value
max()	Maximum Value
abs()	Absolute Value
prod()	Product of Values
cumsum()	Cumulative Sum
cumprod()	Cumulative Product

Visualisations

Univariate

			
Bar Chat	**Line Chart**	**Area Chart**	**Histogram**
df.plot.bar()	df.plot.line()	df.plot.area()	df.plot.hist()
Good for nominal and small ordinal categorical data.	Good for ordinal categorical and interval data.	Good for ordinal categorical and interval data.	Good for interval data.

Bivariate

			
Scatter Plot	**Hex Plot**	**Stacked Bar Chart**	**Bivariate Line Chart**
df.plot.scatter()	df.plot.hexbin()	df.plot.bar(stacked=True)	df.plot.line()
Good for interval and some nominal categorical data.	Good for interval and some nominal categorical data.	Good for nominal and ordinal categorical data.	Good for ordinal categorical and interval data.

Teaching fun

Pick one of the three topics and using examples from the exercises, explain it to "students"

- Applying function to "recode" a column of a dataframe
  - *Tip*: can you use what you learned last week?
- Merging two dataframes and different merge types
  - *Tip*: think about visualizing different types of merges
- Visualizing with seaborn
  - *Tip*: think about explaining the "logic" of seaborn