Python for data analysis and visualization (part 2)

Manuela Salvucci, RCSI

manuelasalvucci@rcsi.ie

2019-12-04

Agenda

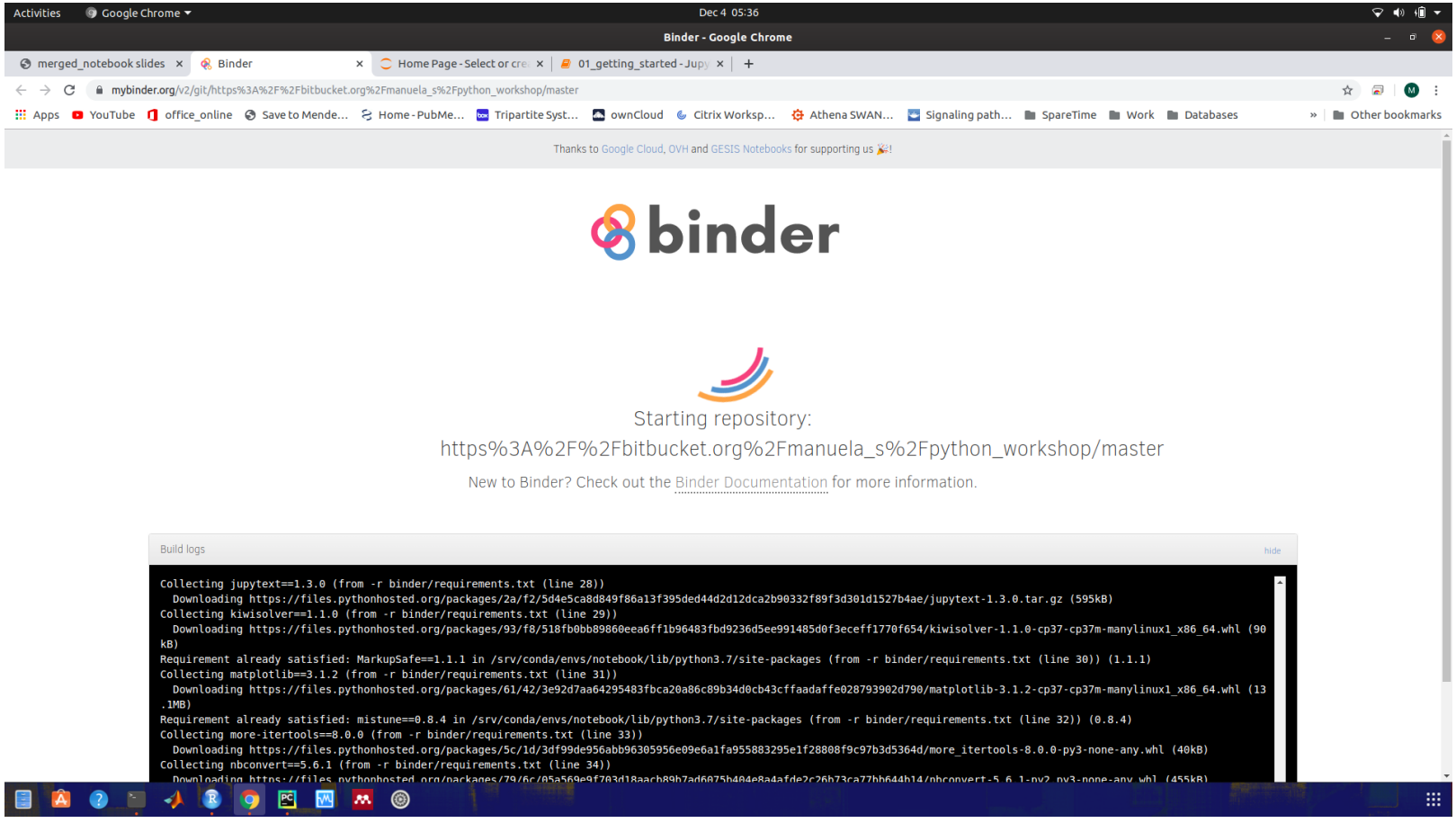
1. Getting Started
2. Jupyter Notebook
3. Pandas imports
4. Pandas dataframes
5. Pandas data cleaning and preprocessing
6. Pandas descriptive statistics
7. Data plotting
8. Full example

Mix of theory and hands-on practise

Please go to http://tiny.cc/4c33gz (http://tiny.cc/4c33gz)

Agenda

Please go to [http://tiny.cc/4c33gz (http://tiny.cc/4c33gz)](http://tiny.cc/4c33gz)



Getting started - section 1

Expectations: "Introduction to Python programming" or similar basic python experience.

Installing software

- **python** - https://www.python.org/downloads/ (https://www.python.org/downloads/)

- **anaconda**: includes python and all the data-science libraries used in this workshop (numpy, pandas, matplotlib, seaborn) - https://www.anaconda.com/distribution (https://www.anaconda.com/distribution)

- **pycharm**: IDE for python development - https://www.jetbrains.com/pycharm/download (https://www.jetbrains.com/pycharm/download)

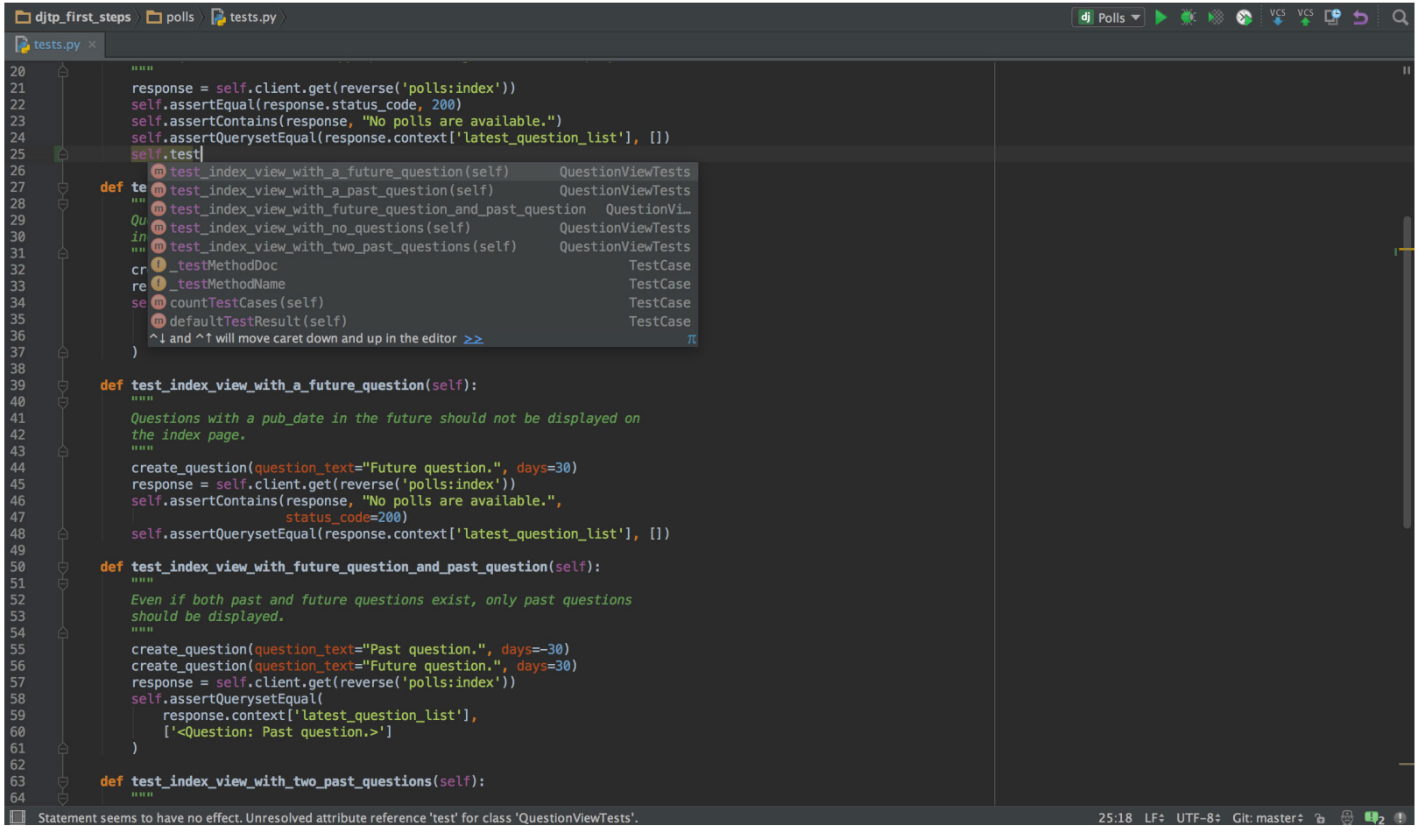
Pycharm

- Integrated Development Environment for python
- Simplifies writing and running python code
- Available for Windows, Mac and Linux
- Available in Professional, Community and Educational edition. Community (free) editon has everything you need.

Pycharm

	PyCharm Professional Edition	PyCharm Community Edition
Intelligent Python editor	✓	✓
Graphical debugger and test runner	✓	✓
Navigation and Refactorings	✓	✓
Code inspections	✓	✓
VCS support	✓	✓
Scientific tools	✓	
Web development	✓	
Python web frameworks	✓	
Python Profiler	✓	
Remote development capabilities	✓	
Database & SQL support	✓	

Pycharm


djtp_first_steps
polls
tests.py
Polls
tests.py
"""
response = self.client.get(reverse('polls:index'))
self.assertEqual(response.status_code, 200)
self.assertContains(response, "No polls are available.")
self.assertQuerysetEqual(response.context['latest_question_list'], [])
self.test
test_index_view_with_a_future_question(self) QuestionViewTests
test_index_view_with_a_past_question(self) QuestionViewTests
test_index_view_with_future_question_and_past_question QuestionVi…
test_index_view_with_no_questions(self) QuestionViewTests
test_index_view_with_two_past_questions(self) QuestionViewTests
_testMethodDoc TestCase
_testMethodName TestCase
countTestCases(self) TestCase
defaultTestResult(self) TestCase
^↓ and ^↑ will move caret down and up in the editor >>
)
def test_index_view_with_a_future_question(self):
"""
Questions with a pub_date in the future should not be displayed on
the index page.
"""
create_question(question_text="Future question.", days=30)
response = self.client.get(reverse('polls:index'))
self.assertContains(response, "No polls are available.",
status_code=200)
self.assertQuerysetEqual(response.context['latest_question_list'], [])
def test_index_view_with_future_question_and_past_question(self):
"""
Even if both past and future questions exist, only past questions
should be displayed.
"""
create_question(question_text="Past question.", days=-30)
create_question(question_text="Future question.", days=30)
response = self.client.get(reverse('polls:index'))
self.assertQuerysetEqual(
response.context['latest_question_list'],
['<Question: Past question.>']
)
def test_index_view_with_two_past_questions(self):
"""
Statement seems to have no effect. Unresolved attribute reference 'test' for class 'QuestionViewTests'.
25:18 LF UTF-8 Git: master

But not today...

Will use online version of jupyter notebook, so no local installation required.

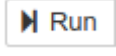
Jupyter notebook foundamental - Section 2

What's a jupyter notebook?

- "jupyter" + "notebook"
    - "jupyter": "acronym" from Julia + python + R (though many more languages are now supported)
    - "notebook": collection of text, code, plots for an analysis (literate programming)
- Jupyter project: https://jupyter.org/index.html (https://jupyter.org/index.html)
- Gallery of interesting jupyter notebooks: https://github.com/jupyter/jupyter/wiki/A-gallery-of-interesting-Jupyter-Notebooks (https://github.com/jupyter/jupyter/wiki/A-gallery-of-interesting-Jupyter-Notebooks)

Anatomy of a jupyter notebook

- Notebook consists of a series of "cells"
- Each cell can have text (markdown) or code (in our case, python).
- A cell can be run by selecting it and pressing



  button at the top (or pressing ctr-enter)
- When the cell is run, the output is shown underneath:

```
In [1]: 2+2
Out[1]: 4
```

- Simple expressions

```
In [1]: 2+2
```

```
Out[1]: 4
```

- Comments

```
In [2]: # Lines that start with a # are comments.
```

- Store data in variables

In [3]:
```
a = 2 + 2
b = 1 + 1
a + b
```

Out[3]: 6

- Cells can reference variables from prevoius cells (make sure to run the previous cells first!):

In [4]:
```
a - b
```

Out[4]: 2

Where can I find this material?

- Presentation and code materials at https://bitbucket.org/manuela_s/python_workshop/ (https://bitbucket.org/manuela_s/python_workshop/)
- Binder version at http://tiny.cc/4c33gz (http://tiny.cc/4c33gz)
- Any question, comment, come chat to me or email me at manuelasalvucci@rcsi.ie

Practise

1. Go to http://tiny.cc/4c33gz (http://tiny.cc/4c33gz)
2. Click on "02_jupyter_notebook_practise.py"
3. Play around

Continue in 5 min.

Pandas import - section 3

Pandas library

pandas is a Python package providing fast, flexible, and expressive data structures *designed to make working with structured (tabular, multidimensional, potentially heterogeneous) and time series data both easy and intuitive**. It aims to be the fundamental high-level building block for doing practical, real world data analysis in Python. Additionally, it has the broader goal of becoming the most powerful and flexible open source data analysis / manipulation tool available in any language. It is already well on its way toward this goal. (From https://pypi.org/project/pandas/ (https://pypi.org/project/pandas/))*

"the name is derived from the term “panel data”, an econometrics term for multidimensional structured data sets." (Wikipedia)

Pandas library

pandas is well suited for many different kinds of data:

- *Tabular data with heterogeneously-typed columns, as in an SQL table or Excel spreadsheet*
- *Ordered and unordered (not necessarily fixed-frequency) time series data.*
- *Arbitrary matrix data (homogeneously typed or heterogeneous) with row and column labels*

Any other form of observational / statistical data sets. The data actually need not be labeled at all to be placed into a
- *pandas data structure*

The two primary data structures of pandas, ***Series (1-dimensional) and DataFrame (2-dimensional)****, handle the vast majority of typical use cases in finance, statistics, social science, and many areas of engineering.* ***For R users, DataFrame provides everything that R's data.frame provides and much more****. From https://pypi.org/project/pandas/*
- *(https://pypi.org/project/pandas/)*

Example of data import with pandas

In [1]:
```
import os
import pandas

input_filename = os.path.join('data', 'countries.xls')
countries = pandas.read_excel(input_filename, index_col='Country')
countries
```

Out[1]:

Country	Population	Area	Capital
Ireland	4784000	84421	Dublin
Italy	60590000	301338	Rome
Germany	82790000	357386	Berlin

Step by step

Importing libraries

In [2]:
```
import os
import pandas
```

- Built-in libraries (like "os") is already installed and can just be imported. List of built in modules: https://docs.python.org/3/py-modindex.html (https://docs.python.org/3/py-modindex.html)
- External libraries (like pandas) need to be installed first. Can be installed from the python package index (https://pypi.org/ (https://pypi.org/)) with the pip command. If you installed python with anaconda (https://www.anaconda.com/ (https://www.anaconda.com/)), pandas and other data-science packages are aleady installed.
- Makes library available in the "pandas" namespace. Functions from the pandas library can be called as pandas.(function_name), for example `pandas.read_excel()`
- Alternative import forms:
    - `from pandas import read_excel` and `read_excel()`
    - `from pandas import *` and `read_excel()`
    - `import pandas as pd` and `pd.read_excel()`

Importing data

read_excel

```
pandas.read_excel(io, sheet_name=0, header=0, names=None,
index_col=None, usecols=None, squeeze=False, dtype=None,
engine=None, converters=None, true_values=None,
false_values=None, skiprows=None, nrows=None, na_values=None,
keep_default_na=True, verbose=False, parse_dates=False,
date_parser=None, thousands=None, comment=None, skip_footer=0,
skipfooter=0, convert_float=True, mangle_dupe_cols=True,
**kwds)
```

- full documentation at https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_excel.html (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_excel.html)

Filename (aka io)

In [3]:
```
input_filename = os.path.join('data', 'countries.xls')
input_filename
```

Out[3]:
```
'data/countries.xls'
```

Basic:

- `countries.xls` - file in the current directory

Sub-directories:

- `data/countries.xls` - file in the `data` sub-directory (linux/mac)
- `data\countries.xls` - file in the `data` sub-directory (windows)
- `os.path.join('data', 'countries.xls')` - file in the `data` sub-directory (any os)

Can also import "directly" from a URL:

Setting index/indices

Basic:

- `index_col=0` or `index_col='Country'`

Can set multiple indices:

- for example `index_col=[0,3]` or `index_col=['Country', 'Capital']`

No necessary to set an index, **however** setting index/indices allows for:

- more performant and simpler look-ups, for example `countries.loc['Ireland']`
- easier joins between dataframes (will use index/indices by default if what to join on is not specified)
- separation between metadata (index/indices) and data (regular columns) which makes data manipulation easier (`countries.stack()`, `countries.describe'`, ...)

Rule of thumbs: set metadata as index

Other pandas imports

- `read_csv`: read from flat file (use separator ',' (default), ';' and '\t' to read different text files)
- `read_sql`: import data from SQL database
- `read_spps`: import from SPPS file
- `read_stata`: import from stata
- `read_pickle`: read from python object serialization files.
- ...many more

See full documentation at https://pandas.pydata.org/pandas-docs/stable/reference/io.html (https://pandas.pydata.org/pandas-docs/stable/reference/io.html)

Import from MATLAB and R

- No native support from pandas
- Can be imported with 3rd party packages:
    - MATLAB .mat files: https://scipy-cookbook.readthedocs.io/items/Reading_mat_files.html (https://scipy-cookbook.readthedocs.io/items/Reading_mat_files.html)
    - R .rds or .Rdata: https://github.com/ofajardo/pyreadr (https://github.com/ofajardo/pyreadr)

Building a dataframe directly in pandas

Make a dictionary with the data to include in the dataframe

In [4]:
```
data = {
    'Country': ['Ireland', 'Italy', 'Germany'],
    'Population': [4784000, 60590000, 82790000],
    'Area': [84421, 301338, 357386],
    'Capital': ['Dublin', 'Rome', 'Berlin'],
}
```

Add the data into the dataframe

In [5]:
```
countries = pandas.DataFrame(data)
countries
```

Out[5]:

	Country	Population	Area	Capital
0	Ireland	4784000	84421	Dublin
1	Italy	60590000	301338	Rome
2	Germany	82790000	357386	Berlin

Building a dataframe directly in pandas

Set indices (optional). Note `inplace=True` (alternatively `countries = countries.set_index(['Country', 'Capital'])`)

In [6]:
```
countries.set_index(['Country', 'Capital'], inplace=True)
countries
```

Out[6]:

		Population	Area
Country	**Capital**		
Ireland	**Dublin**	4784000	84421
Italy	**Rome**	60590000	301338
Germany	**Berlin**	82790000	357386

Pandas export

- `countries.to_excel(filename, ...)`
- `countries.to_csv(filename, ...)`
- `countries.to_sql(filename, ...)`
- `countries.to_stata(filename, ...)`
- `countries.to_pickle(filename, ...)`

Alternatives to pandas for heavy datasets

- **Almost** drop-in pandas replacement:
    - **dask**: https://dask.org/ (https://dask.org/)
    - **datatable**: https://github.com/h2oai/datatable (https://github.com/h2oai/datatable)

Practise

1. Go to http://tiny.cc/4c33gz (http://tiny.cc/4c33gz)
2. Click on "03_pandas_import_practise.py"
3. Play around

Continue in 10 min.

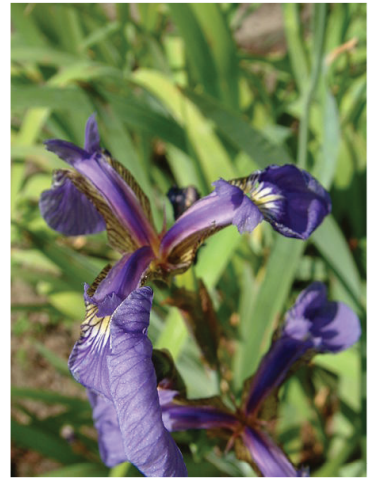
Pandas dataframe foundamentals - section 4

The iris dataset

- The iris dataset (https://en.wikipedia.org/wiki/Iris_flower_data_set (https://en.wikipedia.org/wiki/Iris_flower_data_set)) was first described by Fisher in 1936
- It includes measurements for 4 flower characteristics (length and width for petals and sepals) for 150 flowers from 3 different iris species

Iris versicolor

Iris setosa



Iris virginica

The iris dataset

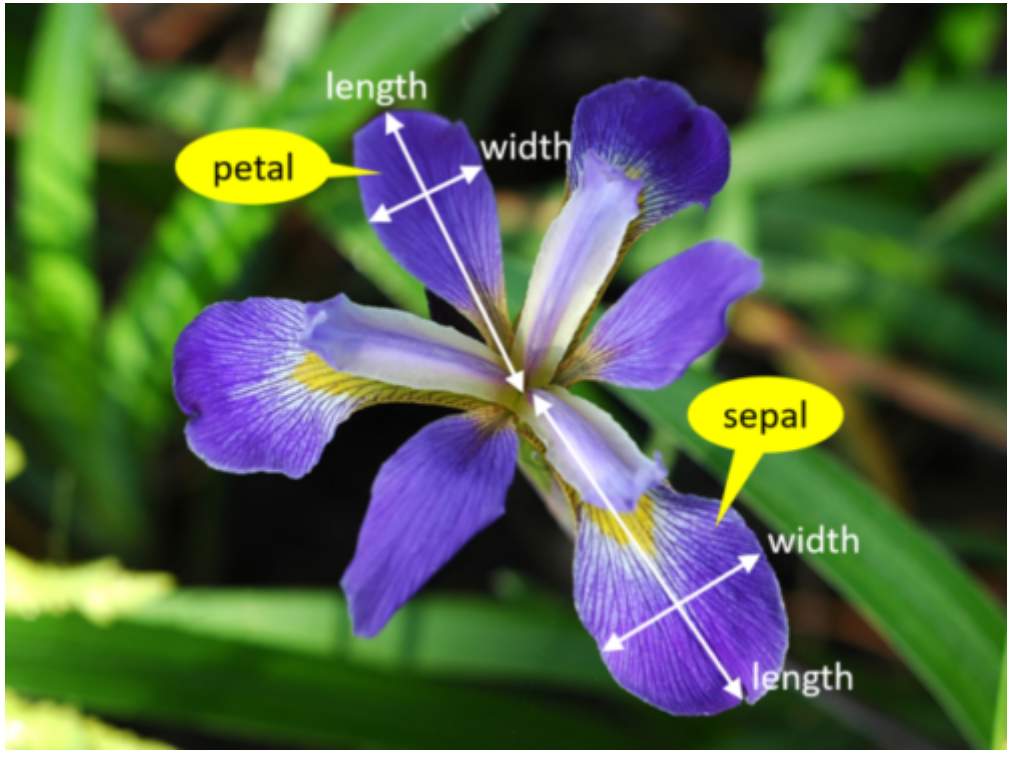


Image from https://www.integratedots.com/determine-number-of-iris-species-with-k-means/ (https://www.integratedots.com/determine-number-of-iris-species-with-k-means/)

Let's explore the iris dataset

To load the data, we do:

In [1]:
```
import os
import pandas
```

In [2]:
```
iris = pandas.read_csv(os.path.join('data', 'iris.csv'))
```

Let's explore the iris dataset

The dataframe has columns with column headers

Out[3]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5	3.6	1.4	0.2	setosa

Use `iris.columns` to get the column names (returns a pandas Index object)

In [4]:
```
iris.columns
```

Out[4]:
```
Index(['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)',
       'petal width (cm)', 'species'],
      dtype='object')
```

Let's explore the iris dataset

Each column represents 1 variable (i.e. 1 type of attribute/characteristic/feature)

Out[5]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5	3.6	1.4	0.2	setosa

Select a single column by doing `iris.[column_name]` or `iris.column_name` (if no special characters). returns a pandas series object.

In [6]:
```
iris['sepal length (cm)']
```

Out[6]:
```
0      5.1
1      4.9
2      4.7
3      4.6
4      5.0
      ...
145    6.7
146    6.3
147    6.5
148    6.2
149    5.9
Name: sepal length (cm), Length: 150, dtype: float64
```

Let's explore the iris dataset

Out[7]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5	3.6	1.4	0.2	setosa

Use `iris.index` to get the index of the table. The index is like a special column that can be used for easy lookups. One of the data-columns can be set to be the index, or by default a simple range-index is used.

In [8]:
```
iris.index
```

Out[8]:
```
RangeIndex(start=0, stop=150, step=1)
```

Let's explore the iris dataset

Each row represents 1 observation (i.e. 1 sample/flower)

Out[9]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5	3.6	1.4	0.2	setosa

We can select rows based on the row-number (`iris.iloc[row_number]`) or the index (`iris.loc[index_value]`). In this case the index value is the same as the row_number (zero-indexed). Returns a pandas series object.

In [10]:
```
iris.iloc[0]
```

Out[10]:
```
sepal length (cm)       5.1
sepal width (cm)        3.5
petal length (cm)       1.4
petal width (cm)        0.2
species                setosa
Name: 0, dtype: object
```

Let's explore the iris dataset

What is the size of the dataset? rows x columns

In [11]:
```
iris.shape
```

Out[11]:
```
(150, 5)
```

Let's explore the iris dataset

Use `iris.head()` to show first few rows (default n=5)

In [12]:
```
iris.head()
```

Out[12]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

In [13]:
```
iris.head(n=3)
```

Out[13]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa

Let's explore the iris dataset

Use `iris.tail()` to show last few rows (default n=5)

In [14]:
```
iris.tail()
```

Out[14]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

In [15]:
```
iris.tail(n=3)
```

Out[15]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

Let's explore the iris dataset

Use `iris.sample()` to show a random subset of rows (default n=1)

In [16]:
```
iris.sample(n=5)
```

Out[16]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
123	6.3	2.7	4.9	1.8	virginica
22	4.6	3.6	1.0	0.2	setosa
84	5.4	3.0	4.5	1.5	versicolor
93	5.0	2.3	3.3	1.0	versicolor
91	6.1	3.0	4.6	1.4	versicolor

Specify `axis='columns'` to show a random subset of columns (deafult n=1)

In [17]:
```
iris.sample(n=2).sample(n=3, axis='columns')
```

Out[17]:

	sepal length (cm)	sepal width (cm)	petal length (cm)
82	5.8	2.7	3.9
59	5.2	2.7	3.9

Let's explore the iris dataset

`iris.dtypes` to show data type of each column

In [18]:
```
iris.dtypes
```

Out[18]:
```
sepal length (cm)    float64
sepal width (cm)     float64
petal length (cm)    float64
petal width (cm)     float64
species               object
dtype: object
```

Let's explore the iris dataset

Use `iris.info()` to get an overview of the dataframe

In [19]:
```
iris.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
sepal length (cm)    150 non-null float64
sepal width (cm)     150 non-null float64
petal length (cm)    150 non-null float64
petal width (cm)     150 non-null float64
species              150 non-null object
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

Practise

1. Go to http://tiny.cc/4c33gz (http://tiny.cc/4c33gz)
2. Click on "04_pandas_dataframe_practise.py"
3. Play around

Continue in 15 min.

Data cleaning and preprocessing with pandas - section 5

Data cleaning and preprocessing with pandas

To load the data, we do:

In [1]:
```
import os
import numpy
import pandas
```

In [2]:
```
iris = pandas.read_csv(os.path.join('data', 'iris.csv'))
```

Data cleaning and preprocessing

The iris dataset is a "clean" dataset, but real-world datasets never are...

Common data cleaning includes:

- recode missing data
- harmonize entries (spelling, etc.)
- dropping columns (redundant, not informative, etc.)
- drop rows (duplicates, missing data, etc.)
- map original values to values required for downstream analysis (binarize, convert units, etc.)
- synthetizing features (features engineering)
- normalization/scaling features

Let's make the iris dataset a bit messy...

In [3]:
```
# Change the spelling of the species on some of the rows
to_replace = iris.species.sample(frac=0.2, random_state=2)
iris.loc[to_replace.index, 'species'] = to_replace.str.upper()

# Replace some of the values with NaN (missing value)
iris.where(numpy.random.RandomState(seed=0).random(iris.shape) > 0.05, numpy.nan
, inplace=True)
```

In [4]:
```
iris.head()
```

Out[4]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	SETOSA
3	4.6	NaN	1.5	0.2	SETOSA
4	5.0	3.6	1.4	0.2	setosa

Missing data

Use `iris.isna()` to visualise missing data (`True` if missing)

In [5]:
```
iris.isna().head()
```

Out[5]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
0	False	False	False	False	False
1	False	False	False	False	False
2	False	False	False	False	False
3	False	True	False	False	False
4	False	False	False	False	False

Use `iris.isna().sum()` to count missing data along each column

In [6]:
```
iris.isna().sum()
```

Out[6]:
```
sepal length (cm)    6
sepal width (cm)     7
petal length (cm)    9
petal width (cm)     6
species              9
dtype: int64
```

Missing data

Use `iris.isna().sum(axis=1)` to count missing data along each row

In [7]:
```
iris.isna().sum(axis=1).head()
```

Out[7]:
```
0    0
1    0
2    0
3    1
4    0
dtype: int64
```

Use `iris.isna().all().sum(axis=1)` to count rows with missing data across all columns

In [8]:
```
iris.isna().all(axis=1).sum()
```

Out[8]:
```
0
```

Missing data

Use `iris.dropna()` to drop rows with missing data (by default a row is dropped if contains 'any' missing data)

In [9]:
```
iris.dropna().shape
```

Out[9]:
```
(116, 5)
```

Use `iris.dropna(how='all')` to drop rows with missing data in *all* columns

In [10]:
```
iris.dropna(how='all').shape
```

Out[10]:
```
(150, 5)
```

Use `iris.dropna(subset=[column names])` to drop rows with missing data in specific columns (list of column names)

In [11]:
```
iris.dropna(subset=['sepal length (cm)', 'sepal width (cm)']).shape
```

Out[11]:
```
(137, 5)
```

Filling missing data

Important note: whether or not to fill missing data and how depends on the dataset and the specific research question

Replace missing data with constant value (for example 0, for illustration only)

In [12]:
```
iris.fillna(0).head()
```

Out[12]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	SETOSA
3	4.6	0.0	1.5	0.2	SETOSA
4	5.0	3.6	1.4	0.2	setosa

Filling missing data

Replace missing data with constant value per column (for example median, for illustration only)

In [13]:
```
iris.fillna(iris.median()).head()
```

Out[13]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	SETOSA
3	4.6	3.0	1.5	0.2	SETOSA
4	5.0	3.6	1.4	0.2	setosa

Filling missing data

Replace missing data by linear interpolation (for illustration only)

In [14]:
```
iris.interpolate(method='linear').head()
```

Out[14]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	SETOSA
3	4.6	3.4	1.5	0.2	SETOSA
4	5.0	3.6	1.4	0.2	setosa

Harmonize entries

See existing spellings

In [15]:
```
iris.species.value_counts()
```

Out[15]:
```
virginica     41
versicolor    39
setosa        34
SETOSA        11
VERSICOLOR     8
VIRGINICA      8
Name: species, dtype: int64
```

Harmonize entries

Use `.map()` with a dictionary that maps current to desidered values

In [16]:
```
iris.species.map({
    'virginica': 'Virginica',
    'versicolor': 'Versicolor',
    'setosa': 'Setosa',
    'VIRGINICA': 'Virginica',
    'VERSICOLOR': 'Versicolor',
    'SETOSA': 'Setosa',
}).value_counts()
```

Out[16]:
```
Virginica     49
Versicolor    47
Setosa        45
Name: species, dtype: int64
```

Harmonize entries

Use `.replace()` with a dictionary that maps current to desidered values. Unlisted values remain unchanged

In [17]:
```
iris.species.replace({
    'VIRGINICA': 'virginica',
    'VERSICOLOR': 'versicolor',
    'SETOSA': 'setosa',
}).value_counts()
```

Out[17]:
```
virginica     49
versicolor    47
setosa        45
Name: species, dtype: int64
```

Harmonize entries

Use `.replace()` with a dictionary that maps current to desidered values using regular expression (https://en.wikipedia.org/wiki/Regular_expression (https://en.wikipedia.org/wiki/Regular_expression))

In [18]:
```
iris.species.replace({
    'VIRGINICA|virginica': 'Virginica', # A or B
    '(?i:VERSICOLOR)': 'Versicolor', # Case insensitive groups
    '[Ss].*': 'Setosa', # String starting with S or S
}, regex=True).value_counts()
```

Out[18]:
```
Virginica    49
VerSetosa    47
Setosa       45
Name: species, dtype: int64
```

Harmonize entries

Use your own function for custom logic

In [19]:
```
def harmonize_species(old_name):
    if old_name == 'SETOSA':
        return 'setosa'
    elif old_name == 'VIRGINICA':
        return 'virginica'
    elif old_name == 'VERSICOLOR':
        return 'versicolor'
    else:
        return old_name
iris.species.apply(harmonize_species).value_counts()
```

Out[19]:
```
virginica     49
versicolor    47
setosa        45
Name: species, dtype: int64
```

Harmonize entries

Use pandas built-in string methods for common manipulations (https://pandas.pydata.org/pandas-docs/stable/user_guide/text.html#method-summary (https://pandas.pydata.org/pandas-docs/stable/user_guide/text.html#method-summary))

In [20]:
```
iris.species.str.title().value_counts()
```

Out[20]:
```
Virginica     49
Versicolor    47
Setosa        45
Name: species, dtype: int64
```

Removing data

Use `iris.drop(index=index_to_drop)` to drop rows with specific index/indices

In [21]:
```
iris.drop(index=range(0,5)).head()
```

Out[21]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
5	5.4	3.9	1.7	0.4	SETOSA
6	4.6	3.4	1.4	0.3	NaN
7	5.0	3.4	1.5	0.2	setosa
8	4.4	2.9	1.4	0.2	setosa
9	4.9	3.1	1.5	0.1	setosa

Removing data

Use `iris.drop(columns=columns_to_drop)` to drop columns with specific column(s)

In [22]:
```
iris.drop(columns=['sepal length (cm)', 'sepal width (cm)']).head()
```

Out[22]:

	petal length (cm)	petal width (cm)	species
0	1.4	0.2	setosa
1	1.4	0.2	setosa
2	1.3	0.2	SETOSA
3	1.5	0.2	SETOSA
4	1.4	0.2	setosa

Removing data

Use `.drop_duplicated()` to drop rows with duplicated values

In [23]:
```
iris.drop_duplicates().shape
```

Out[23]:
```
(150, 5)
```

Discretize numeric data

Use `pandas.qcut()` to discretize in quantiles

In [24]:
```
pandas.qcut(iris['sepal length (cm)'], q=3).value_counts(sort=False)
```

Out[24]:
```
(4.2989999999999995, 5.4]    51
(5.4, 6.3]                   53
(6.3, 7.9]                   40
Name: sepal length (cm), dtype: int64
```

Discretize numeric data

Use `pandas.cut()` to discretize in n equally wide bins

In [25]:
```
pandas.cut(iris['sepal length (cm)'], bins=3).value_counts(sort=False)
```

Out[25]:
```
(4.296, 5.5]    58
(5.5, 6.7]      67
(6.7, 7.9]      19
Name: sepal length (cm), dtype: int64
```

Discretize numeric data

Use `pandas.cut()` to discretize with a custom bin edges (for example, [0, mean, Inf))

In [26]:
```
pandas.cut(iris['sepal length (cm)'], bins=[0, iris['sepal length (cm)'].mean(),
 numpy.Inf]).value_counts(sort=False)
```

Out[26]:
```
(0.0, 5.833]    77
(5.833, inf]    67
Name: sepal length (cm), dtype: int64
```

Unit conversion

Use standard arithmetic operations to scale values in columns (for example, convert measurements to meters)

In [27]:
```
iris['sepal length (cm)'].head() / 100
```

Out[27]:
```
0    0.051
1    0.049
2    0.047
3    0.046
4    0.050
Name: sepal length (cm), dtype: float64
```

Synthetizing new columns by combining existing columns

Use standard arithmetic operations to construct a new column (for example, compute ratio length to width for sepal)

In [28]:
```
(iris['sepal length (cm)'] / iris['sepal width (cm)']).head()
```

Out[28]:
```
0    1.457143
1    1.633333
2    1.468750
3         NaN
4    1.388889
dtype: float64
```

Putting all together

Drop rows with missing values and save result into a new dataframe called `iris_clean`

In [29]:
```
iris_clean = iris.dropna().copy()
```

Harmonize species spelling and save results back in the species column

In [30]:
```
iris_clean['species'] = iris_clean['species'].str.title()
```

Synthetize new column with ratio between length and width for sepals and petals

In [31]:
```
iris_clean['sepal_length_over_width_ratio'] = (iris_clean['sepal length (cm)'] /
 iris_clean['sepal width (cm)'])
iris_clean['petal_length_over_width_ratio'] = (iris_clean['petal length (cm)'] /
 iris_clean['petal width (cm)'])
```

Drop redundant columns

In [32]:
```
iris_clean = iris_clean.drop(columns=iris.columns[0:4])
```

Resulting dataframe

In [33]:
```
iris_clean.head()
```

Out[33]:

	species	sepal_length_over_width_ratio	petal_length_over_width_ratio
0	Setosa	1.457143	7.00
1	Setosa	1.633333	7.00
2	Setosa	1.468750	6.50
4	Setosa	1.388889	7.00
5	Setosa	1.384615	4.25

Practise

1. Go to http://tiny.cc/4c33gz (http://tiny.cc/4c33gz)
2. Click on "05_pandas_data_cleaning_and_preprocessing_practise.py"
3. Play around

Continue in 15 min.

Descriptive statistics with pandas - section 6

Descriptive statistics for the iris dataset

Import libraries and dataset

In [1]:
```
import os
import pandas
import numpy

iris = pandas.read_csv(os.path.join('data', 'iris.csv'))
iris.head()
```

Out[1]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

Descriptive statistics for the iris dataset

Use `iris.mean()` to get mean across all rows for each numeric column

In [2]:
```
iris.mean()
```

Out[2]:
```
sepal length (cm)    5.843333
sepal width (cm)     3.057333
petal length (cm)    3.758000
petal width (cm)     1.199333
dtype: float64
```

Use `iris.mean(axis=1)` to get mean across all numeric columns for each row

In [3]:
```
iris.mean(axis=1)
```

Out[3]:
```
0      2.550
1      2.375
2      2.350
3      2.350
4      2.550
       ...
145    4.300
146    3.925
147    4.175
148    4.325
149    3.950
Length: 150, dtype: float64
```

Descriptive statistics for the iris dataset

Use `iris.median()` to get median across all rows for each numeric column

In [4]:
```
iris.median()
```

Out[4]:
```
sepal length (cm)    5.80
sepal width (cm)     3.00
petal length (cm)    4.35
petal width (cm)     1.30
dtype: float64
```

Descriptive statistics for the iris dataset

Similar common aggregation functions are available:

- `.sum()`
- `.quantile()`
- `.count()`
- `.min()`
- `.max()`
- `.std()`
- `.abs()`
- `.corr()`
- ....

Check documentation at https://pandas.pydata.org/pandas-docs/stable/reference/frame.html#computations-descriptive-stats (https://pandas.pydata.org/pandas-docs/stable/reference/frame.html#computations-descriptive-stats)

Can be applied to full dataframe or to selected column(s)

In [5]:
```
iris['sepal length (cm)'].quantile(0.5)
```

Out[5]:
```
5.8
```

In [6]:
```
iris[['sepal length (cm)', 'sepal width (cm)']].quantile(0.5)
```

Out[6]:
```
sepal length (cm)    5.8
sepal width (cm)     3.0
Name: 0.5, dtype: float64
```

Descriptive statistics for the iris dataset

In [7]:
```
# Compute count
iris['sepal length (cm)'].count()
```

Out[7]:
```
150
```

In [8]:
```
# Compute min
iris['sepal length (cm)'].min()
```

Out[8]:
```
4.3
```

In [9]:
```
# Compute max
iris['sepal length (cm)'].max()
```

Out[9]:
```
7.9
```

In [10]:
```
# Compute standard deviation
iris['sepal length (cm)'].std()
```

Out[10]:
```
0.828066127977863
```

Descriptive statistics for the iris dataset

In [11]:
```
# Compute abs
iris['sepal length (cm)'].abs()
```

Out[11]:
```
0      5.1
1      4.9
2      4.7
3      4.6
4      5.0
      ...
145    6.7
146    6.3
147    6.5
148    6.2
149    5.9
Name: sepal length (cm), Length: 150, dtype: float64
```

Descriptive statistics for the iris dataset

In [12]:
```
# Compute correlation (by deafult uses `method='pearson'`)
iris.corr()
```

Out[12]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
sepal length (cm)	1.000000	-0.117570	0.871754	0.817941
sepal width (cm)	-0.117570	1.000000	-0.428440	-0.366126
petal length (cm)	0.871754	-0.428440	1.000000	0.962865
petal width (cm)	0.817941	-0.366126	0.962865	1.000000

In [13]:
```
# Compute Spearman correlation
iris.corr(method='spearman')
```

Out[13]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
sepal length (cm)	1.000000	-0.166778	0.881898	0.834289
sepal width (cm)	-0.166778	1.000000	-0.309635	-0.289032
petal length (cm)	0.881898	-0.309635	1.000000	0.937667
petal width (cm)	0.834289	-0.289032	0.937667	1.000000

Descriptive statistics for the iris dataset

Other useful functions:

```
- `.nsmallest`
- `.nlargest`
- `.idxmin()`
- `.idxmax()`
```

- `.nlargest()` and `.nsmallest()` are applied to a specific column
- `.idxmax()` and `.idxmin()` can be applied to either a full dataframe or to selected column(s)

Descriptive statistics for the iris dataset

Use `iris.nsmallest()` to identify rows with highest measurement for a column

In [14]:
```
iris.nsmallest(n=3, columns='petal length (cm)')
```

Out[14]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
22	4.6	3.6	1.0	0.2	setosa
13	4.3	3.0	1.1	0.1	setosa
14	5.8	4.0	1.2	0.2	setosa

Verify results

In [15]:
```
iris['petal length (cm)'].min()
```

Out[15]:
```
1.0
```

In [16]:
```
iris['petal length (cm)'].idxmin()
```

Out[16]:
```
22
```

In [17]:
```
iris['petal length (cm)'].sort_values().head(3)
```

Out[17]:
```
22    1.0
13    1.1
14    1.2
Name: petal length (cm), dtype: float64
```

Descriptive statistics for the iris dataset

Use `iris.nlargest()` to identify rows with highest measurement for a column

In [18]:
```
iris.nlargest(n=3, columns='petal length (cm)')
```

Out[18]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
118	7.7	2.6	6.9	2.3	virginica
117	7.7	3.8	6.7	2.2	virginica
122	7.7	2.8	6.7	2.0	virginica

Verify results

In [19]:
```
iris['petal length (cm)'].max()
```

Out[19]:
```
6.9
```

In [20]:
```
iris['petal length (cm)'].idxmax()
```

Out[20]:
```
118
```

In [21]:
```
iris['petal length (cm)'].sort_values(ascending=False).head(3)
```

Out[21]:
```
118    6.9
122    6.7
117    6.7
Name: petal length (cm), dtype: float64
```

Descriptive statistics for the iris dataset

Use `iris.apply()` to apply a custom function. Note: subset to numerical columns

In [22]:
```
iris.iloc[:, 0:4].head()
```

Out[22]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

In [23]:
```
iris.iloc[:, 0:4].apply(numpy.mean)
```

Out[23]:
```
sepal length (cm)    5.843333
sepal width (cm)     3.057333
petal length (cm)    3.758000
petal width (cm)     1.199333
dtype: float64
```

Descriptive statistics for the iris dataset

Use `iris.aggregate()` to apply one or more custom functions

In [24]:
```
iris.iloc[:, 0:4].aggregate(lambda x: numpy.mean(x))
```

Out[24]:
```
sepal length (cm)    5.843333
sepal width (cm)     3.057333
petal length (cm)    3.758000
petal width (cm)     1.199333
dtype: float64
```

In [25]:
```
iris.iloc[:, 0:4].aggregate(lambda x: [numpy.mean(x), numpy.std(x)])
```

Out[25]:
```
sepal length (cm)     [5.843333333333334, 0.8253012917851409]
sepal width (cm)     [3.057333333333337, 0.4344109677354946]
petal length (cm)     [3.7580000000000005, 1.759404065775303]
petal width (cm)     [1.1993333333333336, 0.7596926279021594]
dtype: object
```

Descriptive statistics for the iris dataset

Use `iris.aggregate()` to apply one or more custom functions

In [26]:
```
iris.iloc[:, 0:4].aggregate(['sum', 'mean', numpy.median])
```

Out[26]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
sum	876.500000	458.600000	563.700	179.900000
mean	5.843333	3.057333	3.758	1.199333
median	5.800000	3.000000	4.350	1.300000

Descriptive statistics for the iris dataset

Use `iris.aggregate()` to compute range (max - min)

In [27]:
```
iris.iloc[:, 0:4].aggregate(lambda x: [numpy.max(x) - numpy.min(x)])
```

Out[27]:
```
sepal length (cm)    [3.6000000000000005]
sepal width (cm)     [2.4000000000000004]
petal length (cm)                   [5.9]
petal width (cm)                    [2.4]
dtype: object
```

Use `iris.aggregate()` to compute interquantile range (75th percentile - 25th percentile)

In [28]:
```
iris.iloc[:, 0:4].aggregate(lambda x: [numpy.quantile(x, 0.75) - numpy.quantile
(x, 0.25)])
```

Out[28]:
```
sepal length (cm)    [1.3000000000000007]
sepal width (cm)                   [0.5]
petal length (cm)    [3.4999999999999996]
petal width (cm)                   [1.5]
dtype: object
```

Descriptive statistics for the iris dataset

In [29]:
```
iris.iloc[:, 0:4].aggregate(lambda x: numpy.quantile(x, [0.025, 0.25, 0.5, 0.75
, 0.975]))
```

Out[29]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	4.4725	2.2725	1.2725	0.1
1	5.1000	2.8000	1.6000	0.3
2	5.8000	3.0000	4.3500	1.3
3	6.4000	3.3000	5.1000	1.8
4	7.7000	3.9275	6.4550	2.4

Descriptive statistics for the iris dataset

Use `iris.describe()` to get summary statistics for the measurements

In [30]:
```
iris.describe()
```

Out[30]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333
std	0.828066	0.435866	1.765298	0.762238
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

Descriptive statistics for the iris dataset

Use `iris.describe(include='all')` to get summary statistics for all the columns

In [31]:
```
iris.describe(include='all')
```

Out[31]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
count	150.000000	150.000000	150.000000	150.000000	150
unique	NaN	NaN	NaN	NaN	3
top	NaN	NaN	NaN	NaN	virginica
freq	NaN	NaN	NaN	NaN	50
mean	5.843333	3.057333	3.758000	1.199333	NaN
std	0.828066	0.435866	1.765298	0.762238	NaN
min	4.300000	2.000000	1.000000	0.100000	NaN
25%	5.100000	2.800000	1.600000	0.300000	NaN
50%	5.800000	3.000000	4.350000	1.300000	NaN
75%	6.400000	3.300000	5.100000	1.800000	NaN
max	7.900000	4.400000	6.900000	2.500000	NaN

Descriptive statistics for the iris dataset

Use `iris.value_counts()` to get a breakdown of each species in the dataset

In [32]:
```
iris.species.value_counts()
```

Out[32]:
```
virginica     50
setosa        50
versicolor    50
Name: species, dtype: int64
```

Show results as fraction

In [33]:
```
iris.species.value_counts(normalize=True)
```

Out[33]:
```
virginica     0.333333
setosa        0.333333
versicolor    0.333333
Name: species, dtype: float64
```

Descriptive statistics for the iris dataset by flower species

In [34]:
```
iris.groupby('species').describe().style.apply(lambda x: ['background: lightblu
e' if x.name == '50%' else '' for i in x], axis=1)
```

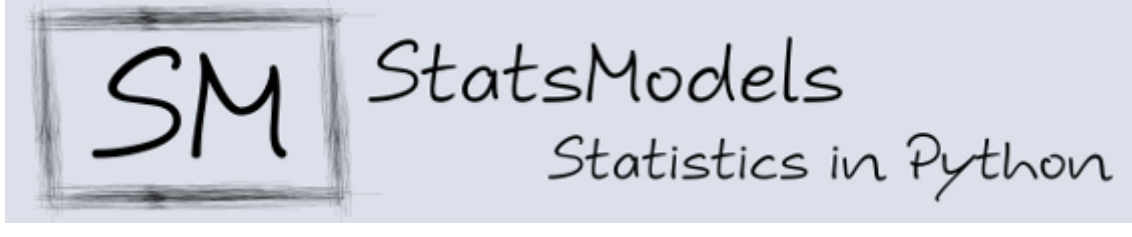
Out[34]:

	sepal length (cm)								sepal width (cm)								petal leng
	count	mean	std	min	25%	50%	75%	max	count	mean	std	min	25%	50%	75%	max	count	n
species																		
setosa	50	5.006	0.35249	4.3	4.8	5	5.2	5.8	50	3.428	0.379064	2.3	3.2	3.4	3.675	4.4	50	1
versicolor	50	5.936	0.516171	4.9	5.6	5.9	6.3	7	50	2.77	0.313798	2	2.525	2.8	3	3.4	50	4
virginica	50	6.588	0.63588	4.9	6.225	6.5	6.9	7.9	50	2.974	0.322497	2.2	2.8	3	3.175	3.8	50	5

Stats beyond pandas built-in functions

There are a few main general pourpose libraries for stats in python:

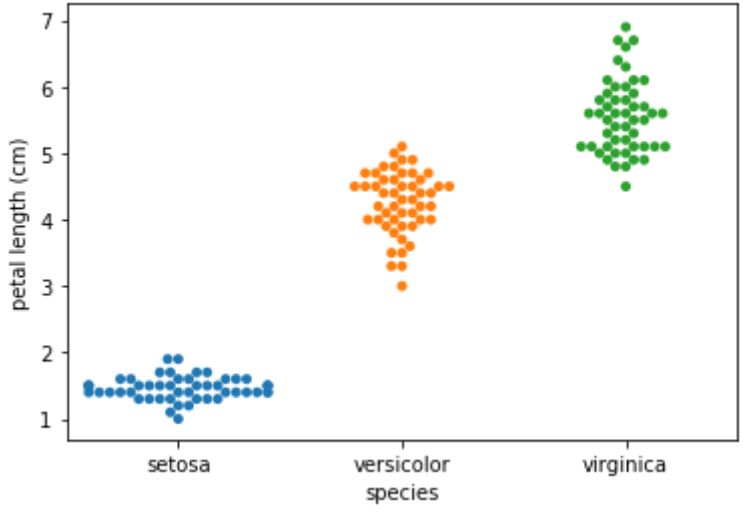
```
- **statsmodels**: https://www.statsmodels.org/stable/index.html
- **scipy**: https://www.scipy.org/
-...
```



Simple example using statsmodels

In [35]:
```
import statsmodels.formula.api
import seaborn


seaborn.swarmplot(x='species', y='petal length (cm)', data=iris);
```



Simple example using statsmodels

In [36]:
```
statsmodels.formula.api.ols('Q("petal length (cm)") ~ species', data=iris).fit()
.summary()
```

Out[36]:

OLS Regression Results

Dep. Variable:	Q("petal length (cm)")	**R-squared:**	0.941
Model:	OLS	**Adj. R-squared:**	0.941
Method:	Least Squares	**F-statistic:**	1180.
Date:	Wed, 04 Dec 2019	**Prob (F-statistic):**	2.86e-91
Time:	11:34:21	**Log-Likelihood:**	-84.847
No. Observations:	150	**AIC:**	175.7
Df Residuals:	147	**BIC:**	184.7
Df Model:	2		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	1.4620	0.061	24.023	0.000	1.342	1.582
species[T.versicolor]	2.7980	0.086	32.510	0.000	2.628	2.968
species[T.virginica]	4.0900	0.086	47.521	0.000	3.920	4.260

Omnibus:	4.394	**Durbin-Watson:**	1.997
Prob(Omnibus):	0.111	**Jarque-Bera (JB):**	5.366
Skew:	0.122	**Prob(JB):**	0.0683
Kurtosis:	3.894	**Cond. No.**	3.73

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

Using R packages from python with rpy2

Some statistical functions may be available in R, but not in python. For this purpose you can call R packages from python using rpy2 (https://rpy2.bitbucket.io/ (https://rpy2.bitbucket.io/)).

In [37]:
```
# Import rpy2 modules
import rpy2.robjects
import rpy2.robjects.packages
import rpy2.robjects.pandas2ri

# Save the R 'stats' package as a python variable
stats = rpy2.robjects.packages.importr('stats')

# Convert pandas dataframes to R objects
with rpy2.robjects.conversion.localconverter(
    rpy2.robjects.default_converter + rpy2.robjects.pandas2ri.converter):
    # Run model in R
    model = stats.lm('petal_length ~ species',
                     data=iris.rename(columns={'petal length (cm)': 'petal_lengt
h'}))
    summary = stats.summary_lm(model)
```

Using R packages from python with rpy2

In [38]:
```
# Extract the coefficients
coef = stats.coef(summary)

# Convert coefficients to python dataframe
with rpy2.robjects.conversion.localconverter(
    rpy2.robjects.default_converter + rpy2.robjects.pandas2ri.converter):
    coef = pandas.DataFrame(rpy2.robjects.conversion.rpy2py(coef), index=coef.ro
wnames, columns=coef.colnames)

coef
```

Out[38]:

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	1.462	0.060858	24.022945	9.303052e-53
speciesversicolor	2.798	0.086067	32.509597	5.254587e-69
speciesvirginica	4.090	0.086067	47.521176	4.106139e-91

Practise

1. Go to http://tiny.cc/4c33gz (http://tiny.cc/4c33gz)
2. Click on "05_pandas_data_cleaning_and_preprocessing_practise.py"
3. Play around

Continue in 10 min.

Data plotting - section 07

Data plotting

- Exploratory data analysis (EDA) is a crucial part of any research project
    - quality control (missing data, outlier detection, ...)
    - descriptive statistics
    - visualization
    - insights to guide downstream analysis

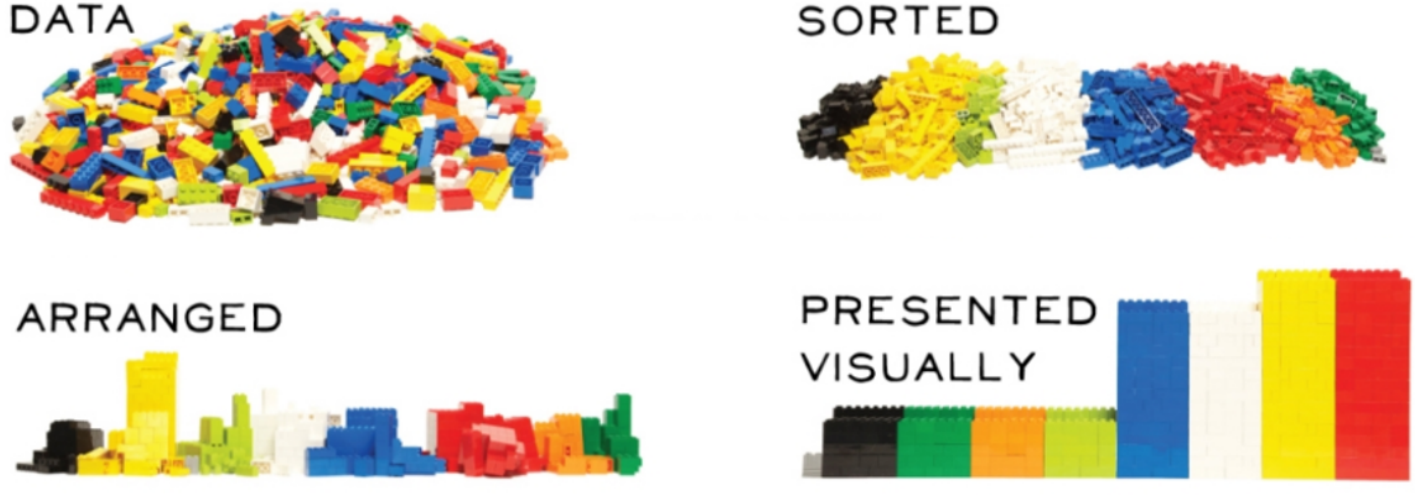


Image from http://www.codeheroku.com/post.html?name=Introduction%20to%20Exploratory%20Data%20Analysis%20(EDA) (http://www.codeheroku.com/post.html?name=Introduction%20to%20Exploratory%20Data%20Analysis%20(EDA))

Beyond built-in plotting with pandas

(A few) of python's plotting libraries:

- **matplotlib**: https://matplotlib.org/ (https://matplotlib.org/)
- **seaborn**: https://seaborn.pydata.org/ (https://seaborn.pydata.org/)
- plotnine (ggplot-like): https://plotnine.readthedocs.io/en/stable/ (https://plotnine.readthedocs.io/en/stable/)
- plotly: https://plot.ly/ (https://plot.ly/)
- bokeh: https://docs.bokeh.org/en/latest/index.html (https://docs.bokeh.org/en/latest/index.html)
- ...

Matplotlib

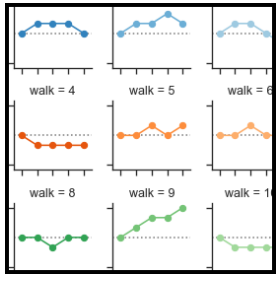
> ***Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms.***
>
> ***Matplotlib tries to make easy things easy and hard things possible.*** *You can generate plots, histograms, power spectra, bar charts, errorcharts, scatterplots, etc., with just a few lines of code. For examples, see the sample plots and thumbnail gallery.*
>
> *For the power user, you have full control of line styles, font properties, axes properties, etc, via an object oriented interface or via a set of functions familiar to MATLAB users.*

From https://matplotlib.org/ (https://matplotlib.org/)

Seaborn

Seaborn is a Python visualization library based on matplotlib. It provides a high-level interface for drawing attractive statistical graphics. (https://seaborn.pydata.org/index.html (https://seaborn.pydata.org/index.html))

- Examples gallery: https://seaborn.pydata.org/examples/index.html (https://seaborn.pydata.org/examples/index.html)



(https://seaborn.pydata.org/examples/many_facets.html)

Let's get plotting the iris dataset

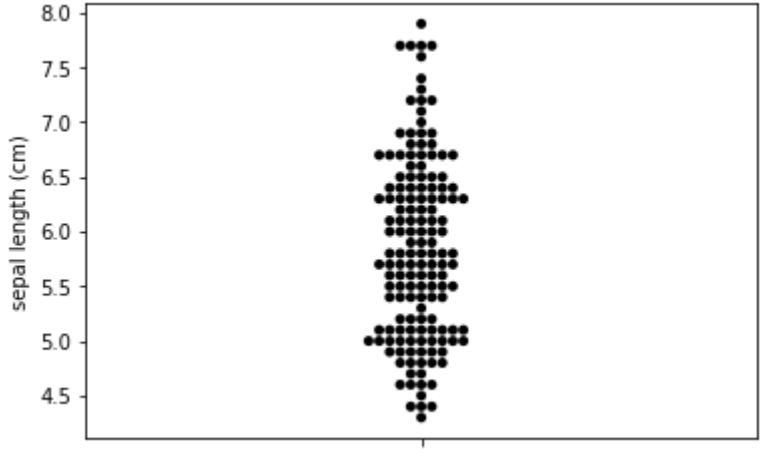
In [1]:
```
import os
import numpy
import pandas

import matplotlib.pyplot
import seaborn

iris = pandas.read_csv(os.path.join('data', 'iris.csv'))
```

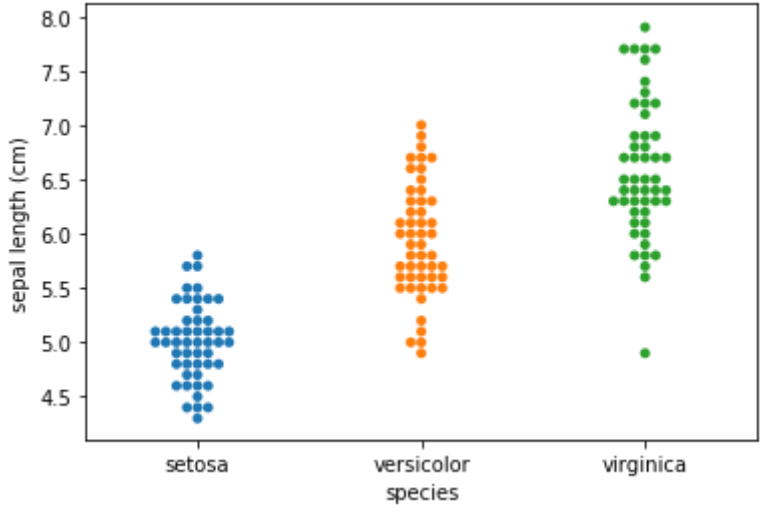
Univariate analysis - swarm-plot

In [2]:
```
seaborn.swarmplot(y='sepal length (cm)', color='k', data=iris);
```



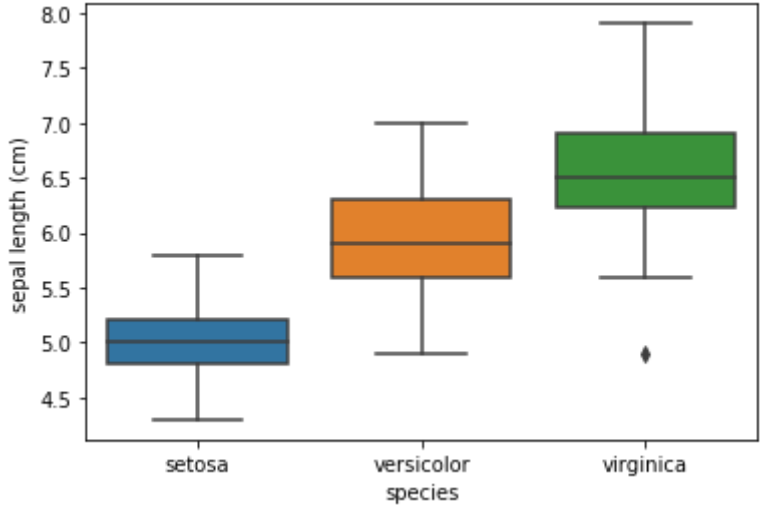
Grouped swarm-plot

In [3]:
```
seaborn.swarmplot(x='species', y='sepal length (cm)', data=iris.reset_index());
```



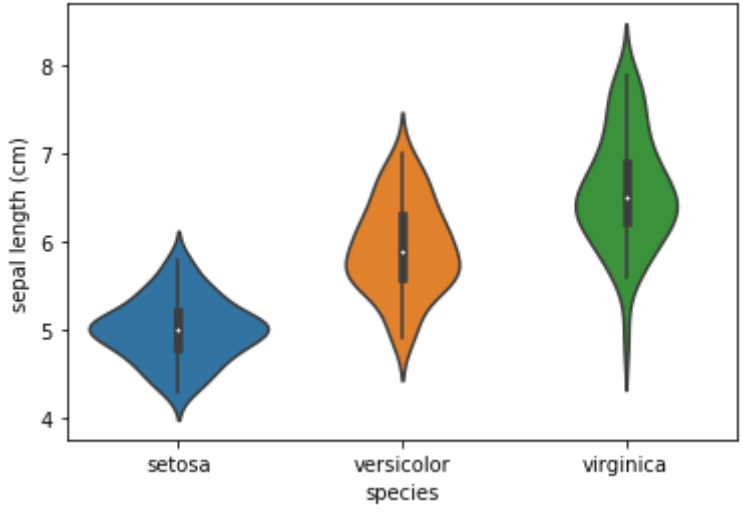
Grouped box-plot

In [4]:
```
seaborn.boxplot(x='species', y='sepal length (cm)', data=iris.reset_index());
```



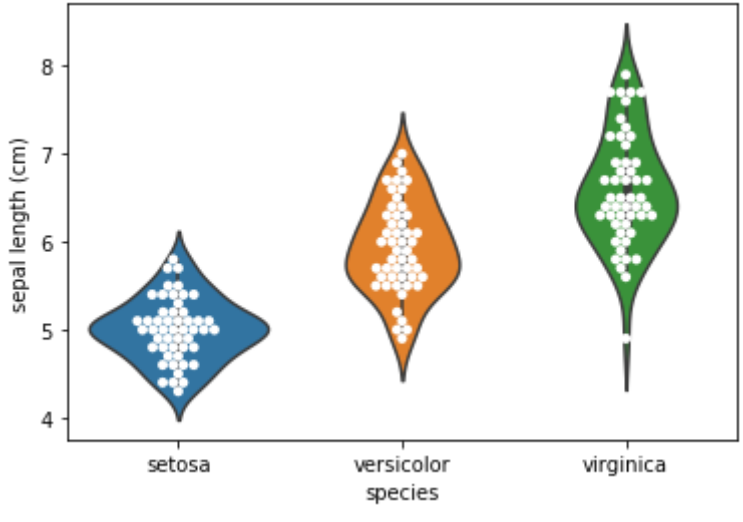
Grouped violin-plot

In [5]:
```
seaborn.violinplot(x='species', y='sepal length (cm)', data=iris.reset_index());
```



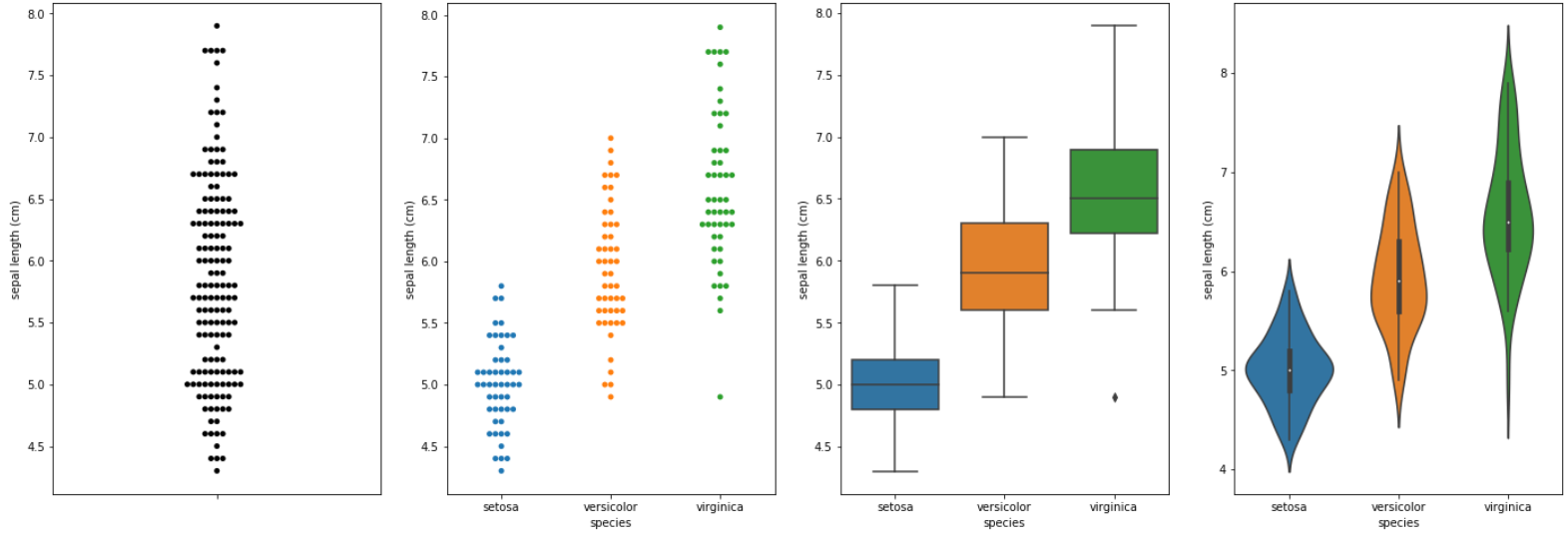
Combining multiple univariate plots

In [6]:
```
seaborn.violinplot(x='species', y='sepal length (cm)', data=iris.reset_index());
seaborn.swarmplot(x='species', y='sepal length (cm)', color='w', data=iris.reset
_index());
```



Compose a figure with multiple sub-plots

In [7]:
```
fig, axes = matplotlib.pyplot.subplots(nrows=1, ncols=4, figsize=(24, 8))
# Dot plot with no grouping variable
seaborn.swarmplot(y='sepal length (cm)', color='k', data=iris, ax=axes[0])
# Dot plot grouped by species
seaborn.swarmplot(x='species', y='sepal length (cm)', dodge=True, data=iris.rese
t_index(), ax=axes[1])
# Boxplot grouped by species
seaborn.boxplot(x='species', y='sepal length (cm)', dodge=True, data=iris.reset_
index(), ax=axes[2])
# Violin plot grouped by species
seaborn.violinplot(x='species', y='sepal length (cm)', dodge=True, data=iris.res
et_index(), ax=axes[3]);
```



Compose a figure with a grid of *identical* sub-plots (FacetGrid)

seaborn.catplot (https://seaborn.pydata.org/generated/seaborn.catplot.html)

> *Figure-level interface for drawing categorical plots onto a FacetGrid.*

seaborn.relplot (https://seaborn.pydata.org/generated/seaborn.relplot.html)

> *Figure-level interface for drawing relational plots onto a FacetGrid.*

> *x, y, hue : names of variables in data Inputs for plotting* ***long-form data****. See examples for interpretation.*

Reshape a dataframe from a wide-format to a long-format

Wide-format:

In [8]:
```
iris.loc[[6, 92, 140]]
```

Out[8]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
6	4.6	3.4	1.4	0.3	setosa
92	5.8	2.6	4.0	1.2	versicolor
140	6.7	3.1	5.6	2.4	virginica

Reshape a dataframe from a wide-format to a long-format

Let's stack the table and add a grouping variable to convert to a tall format

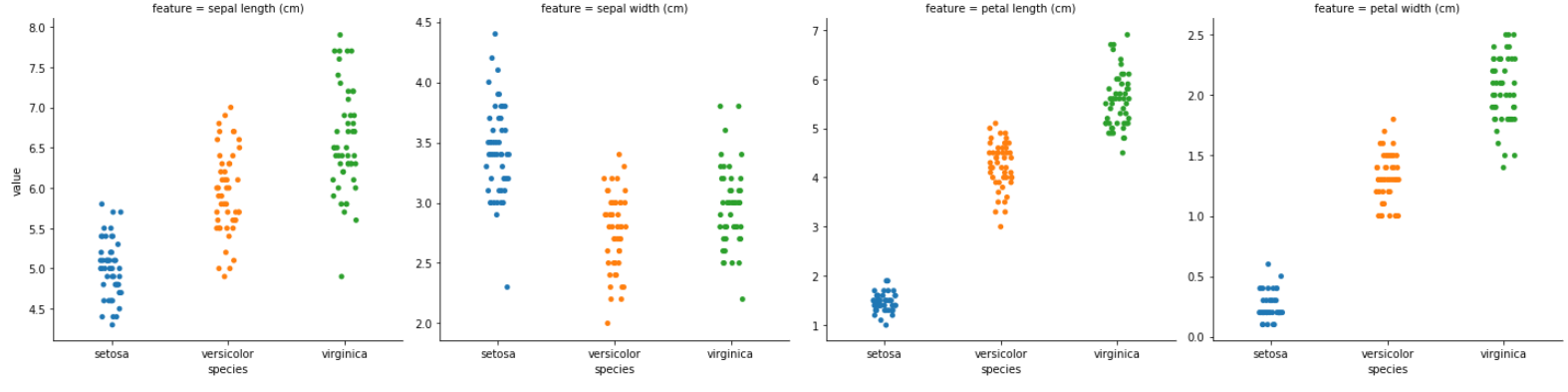
In [9]:
```
iris_long = iris.set_index('species', append=True).stack().to_frame('value')
iris_long.index.names = ['id', 'species', 'feature']
iris_long.loc[[6, 92, 140]]
```

Out[9]:

			value
id	species	feature	
6	setosa	sepal length (cm)	4.6
		sepal width (cm)	3.4
		petal length (cm)	1.4
		petal width (cm)	0.3
92	versicolor	sepal length (cm)	5.8
		sepal width (cm)	2.6
		petal length (cm)	4.0
		petal width (cm)	1.2
140	virginica	sepal length (cm)	6.7
		sepal width (cm)	3.1
		petal length (cm)	5.6
		petal width (cm)	2.4

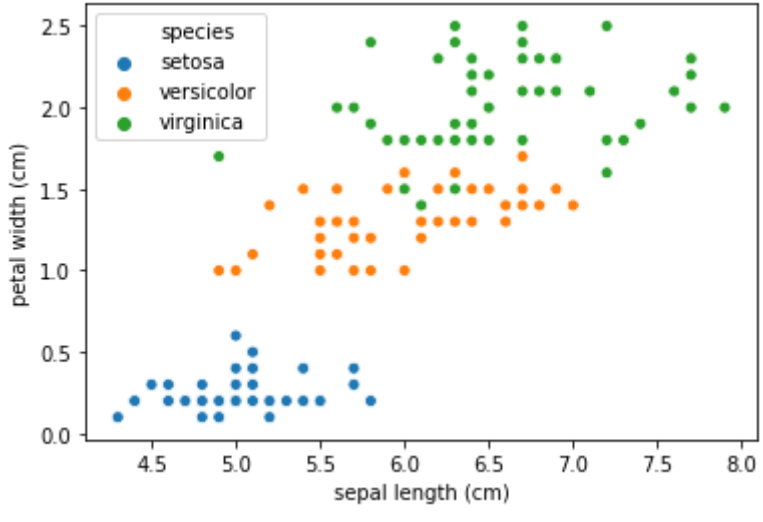
Draw a grid of sub-plots where each categorical plot includes a single feature

In [10]:
```
seaborn.catplot(x='species', y='value', col='feature', sharey=False, data=iris_
long.reset_index());
```



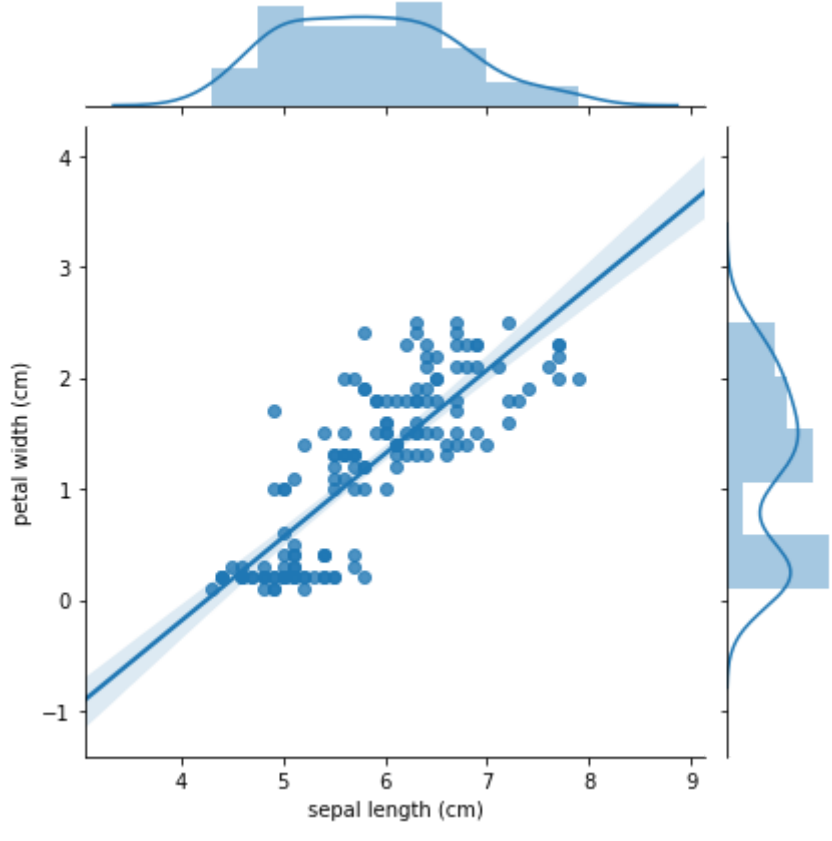
Bivariate analysis - scatter-plot

In [11]:
```
seaborn.scatterplot(x='sepal length (cm)', y='petal width (cm)', hue='species',
data=iris);
```



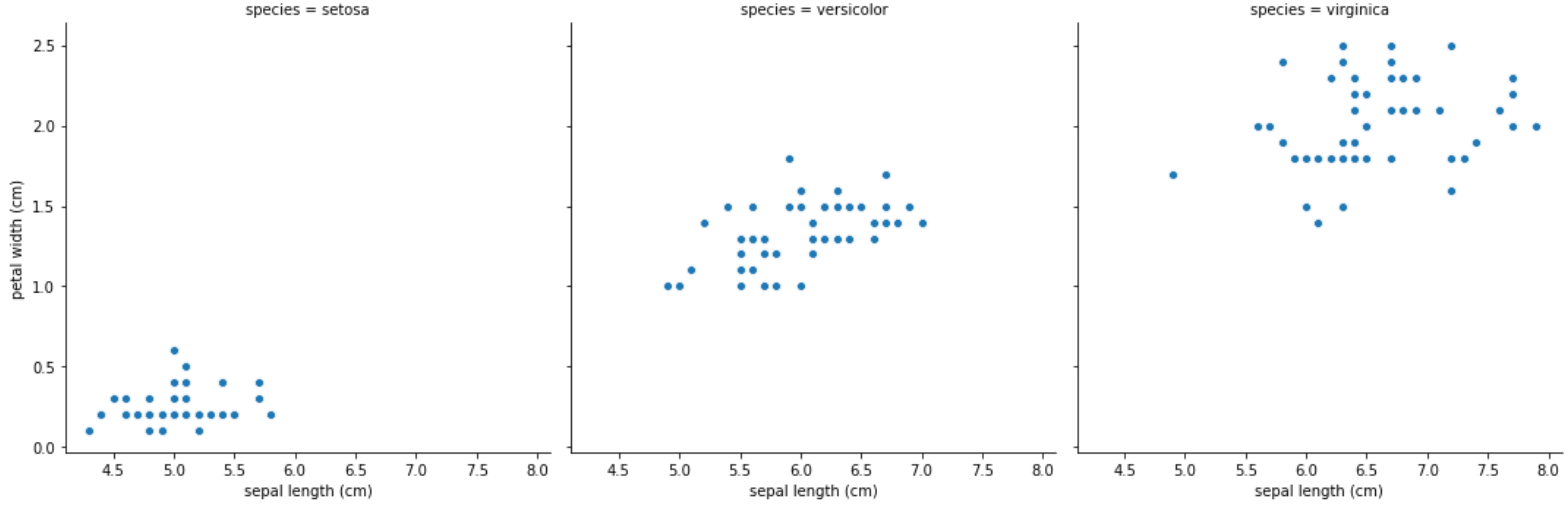
Bivariate analysis - scatterplot + histograms (joint-plot)

In [12]:
```
seaborn.jointplot(x='sepal length (cm)', y='petal width (cm)', kind='reg', data=
iris);
```



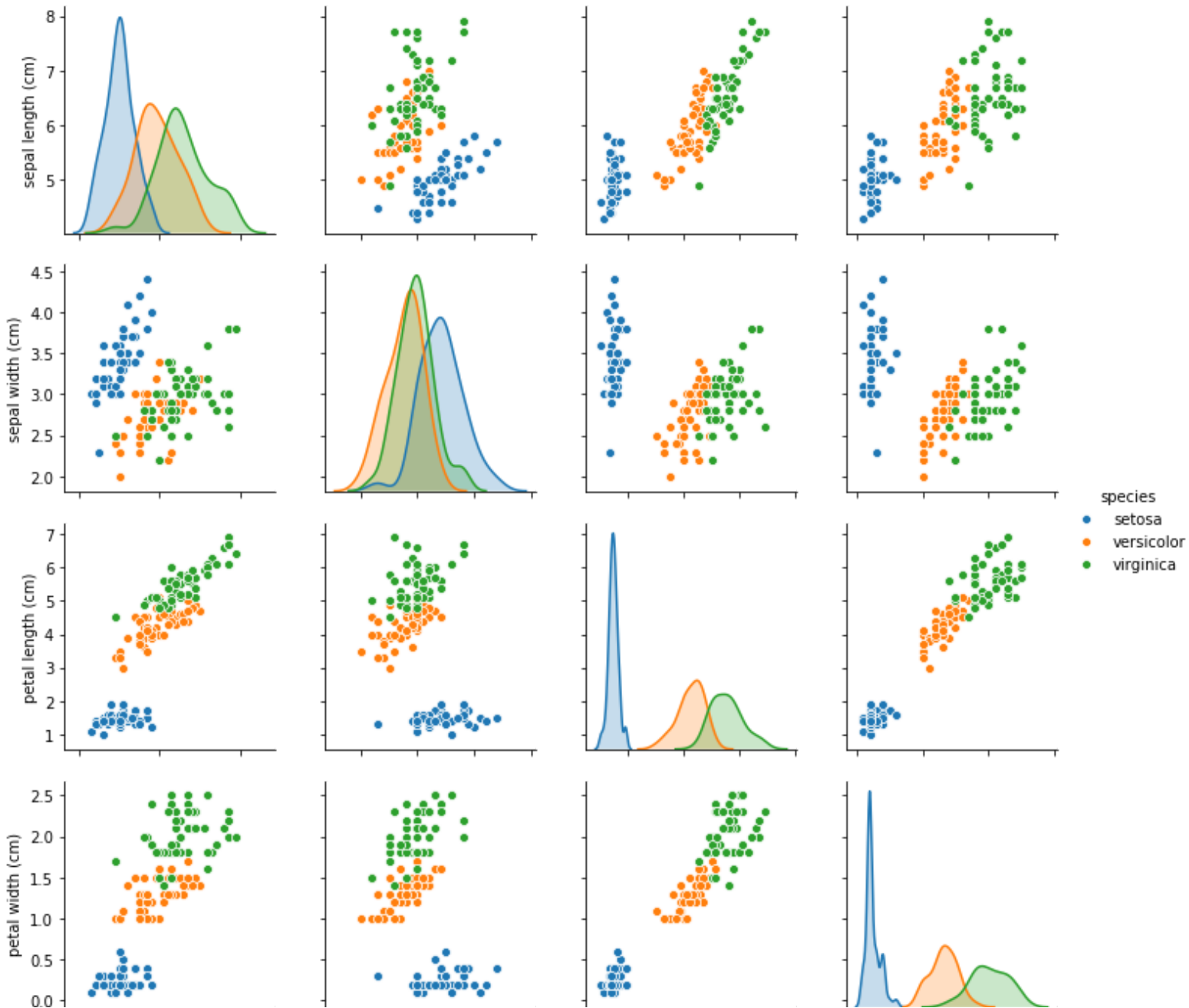
Draw a grid of sub-plots where each relational plot includes a single feature

In [13]:
```
seaborn.relplot(x='sepal length (cm)', y='petal width (cm)', col='species', data
=iris.reset_index());
```

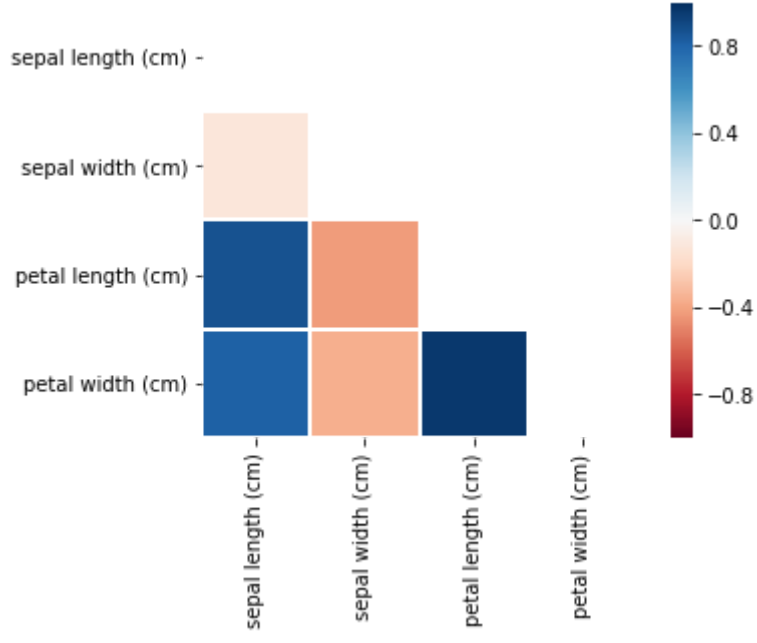


Visualize relationship and distribution between feature pairs

In [14]:
```
seaborn.pairplot(iris, hue='species');
```



Plot heatmap to visualize correlation among features



Practise

1. Go to http://tiny.cc/4c33gz (http://tiny.cc/4c33gz)
2. Click on "07_data_plotting_practise.py"
3. Play around

Continue in 10 min.

Full Example - Dogs and (dog bites) in New York - section 8

Dogs in New York

Data from New York Open Data (https://opendata.cityofnewyork.us/ (https://opendata.cityofnewyork.us/))

- Dogs licensing dataset: https://data.cityofnewyork.us/Health/NYC-Dog-Licensing-Dataset/nu7n-tubp (https://data.cityofnewyork.us/Health/NYC-Dog-Licensing-Dataset/nu7n-tubp)
- Dog bites dataset: https://data.cityofnewyork.us/Health/DOHMH-Dog-Bite-Data/rsgh-akpg (https://data.cityofnewyork.us/Health/DOHMH-Dog-Bite-Data/rsgh-akpg)

Dogs in New York - data import

Import libraries and dataset

In [1]:
```
import os
import pandas
import matplotlib.pyplot

dogs = pandas.read_csv(os.path.join('data', 'NYC_Dog_Licensing_Dataset.csv'),
                       index_col=0,
                       na_values=['Unknown', 'UNKNOWN', 'NAME NOT PROVIDED'])
```

In [2]:
```
dogs.head()
```

Out[2]:

RowNumber	AnimalName	AnimalGender	AnimalBirthMonth	BreedName	Borough	ZipCode	LicenseIssuedDate	LicenseExpiredDate
1	PAIGE	F	2014	American Pit Bull Mix / Pit Bull Mix	NaN	10035	09/12/2014	09/12/2017
2	YOGI	M	2010	Boxer	NaN	10465	09/12/2014	10/02/2017
3	ALI	M	2014	Basenji	NaN	10013	09/12/2014	09/12/2019
4	QUEEN	F	2013	Akita Crossbreed	NaN	10013	09/12/2014	09/12/2017
5	LOLA	F	2009	Maltese	NaN	10028	09/12/2014	10/09/2017

Dogs in New York - data cleaning

In [3]:
```
dogs.rename(columns={
    'BreedName': 'Breed',
    'AnimalGender': 'Gender'},
    inplace=True)
```

Dogs in New York - data aggregation

Count number of dogs per breed and gender

In [4]:
```
dogs_by_breed_and_gender = dogs.groupby(['Breed', 'Gender']).size()
dogs_by_breed_and_gender.head()
```

Out[4]:
```
Breed                     Gender
Affenpinscher             F         43
                          M         48
Afghan Hound              F         26
                          M         33
Afghan Hound Crossbreed   F          6
dtype: int64
```

Dogs in New York - data reshaping

Use `unstack` to reshape the table from tall to wide and put female- and male- counts in separate columns.

In [5]:
```
dogs_by_breed = dogs_by_breed_and_gender.unstack('Gender')
dogs_by_breed.head()
```

Out[5]:

Gender	F	M
Breed		
Affenpinscher	43.0	48.0
Afghan Hound	26.0	33.0
Afghan Hound Crossbreed	6.0	1.0
Airedale Terrier	74.0	81.0
Akita	124.0	234.0

Dogs in New York - data processing

Let's add another column for total number of dogs per breed

In [6]:
```
dogs_by_breed['Total'] = dogs_by_breed.sum(axis=1)
dogs_by_breed.head()
```

Out[6]:

Gender	F	M	Total
Breed			
Affenpinscher	43.0	48.0	91.0
Afghan Hound	26.0	33.0	59.0
Afghan Hound Crossbreed	6.0	1.0	7.0
Airedale Terrier	74.0	81.0	155.0
Akita	124.0	234.0	358.0

Dogs in New York - data processing

Let's focus on the most common breeds (top 50)

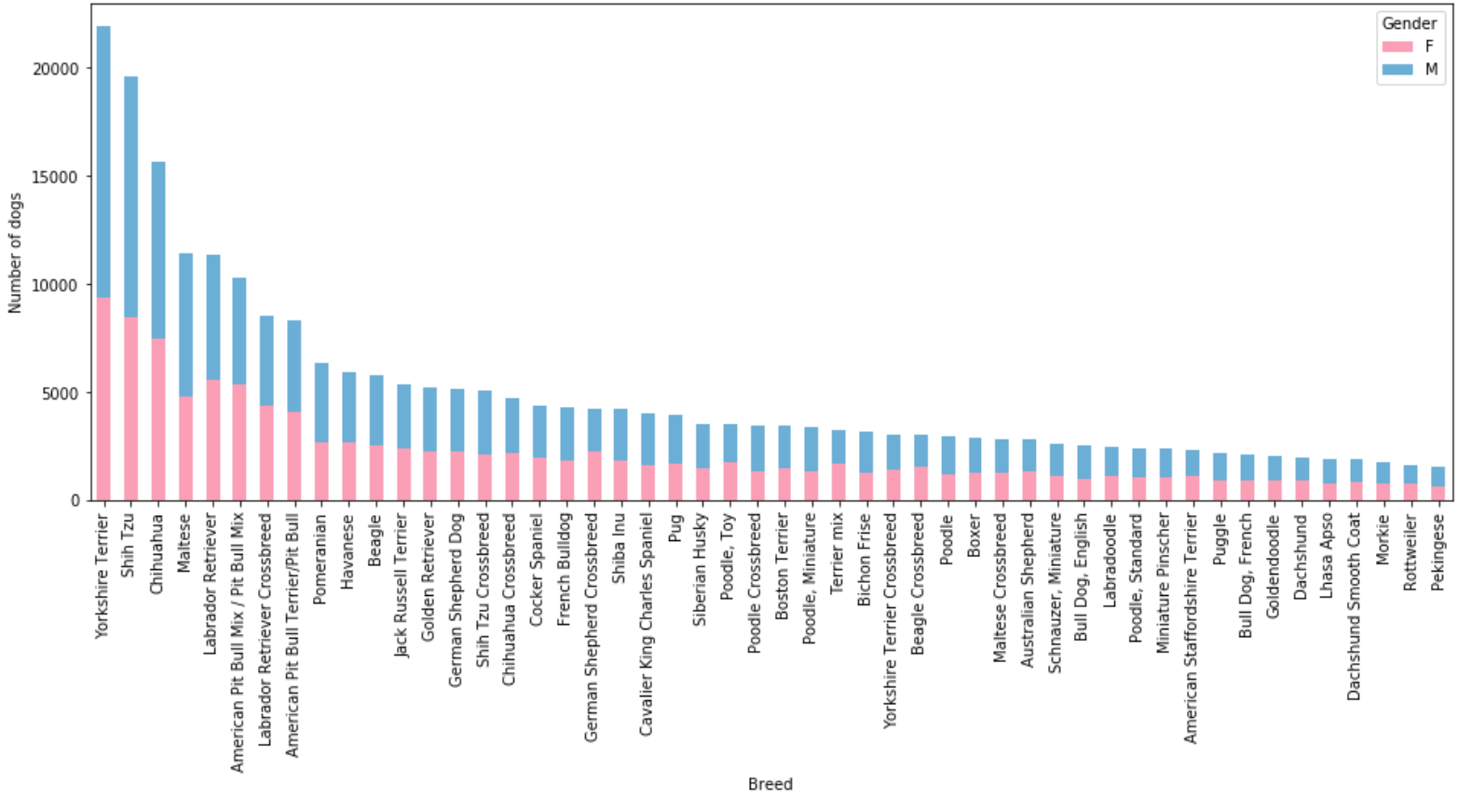
In [7]:
```
dogs_by_breed = dogs_by_breed.sort_values(by='Total', ascending=False).head(50)
dogs_by_breed.head()
```

Out[7]:

Gender	F	M	Total
Breed			
Yorkshire Terrier	9331.0	12588.0	21919.0
Shih Tzu	8414.0	11214.0	19628.0
Chihuahua	7483.0	8164.0	15647.0
Maltese	4802.0	6586.0	11388.0
Labrador Retriever	5548.0	5779.0	11327.0

Dogs in New York - data plotting

In [8]:
```
dogs_by_breed[['F', 'M']].plot.bar(stacked=True, color=['#fa9fb5', '#6baed6'],
figsize=(16,6));
matplotlib.pyplot.gca().set_ylabel('Number of dogs');
```



Dog bites in New York

Joining with another dataset

In [9]:
```
dog_bites = pandas.read_csv(os.path.join('data', 'DOHMH_Dog_Bite_Data.csv'),
                            index_col=0,
                            na_values=['U'])
dog_bites.head()
```

Out[9]:

UniqueID	DateOfBite	Species	Breed	Age	Gender	SpayNeuter	Borough	ZipCode
1	January 02 2015	DOG	Poodle, Standard	3	M	True	Brooklyn	11238
2	January 02 2015	DOG	HUSKY	NaN	NaN	False	Brooklyn	11249
3	January 02 2015	DOG	NaN	NaN	NaN	False	Brooklyn	NaN
4	January 01 2015	DOG	American Pit Bull Terrier/Pit Bull	6	M	False	Brooklyn	11221
5	January 03 2015	DOG	American Pit Bull Terrier/Pit Bull	1	M	False	Brooklyn	11207

Dog bites in New York - data aggregation

Let's group by breed and gender like the previous data-set

In [10]:
```
dog_bites_by_breed_and_gender = dog_bites.groupby(['Breed', 'Gender']).size()
dog_bites_by_breed_and_gender.head()
```

Out[10]:
```
Breed                       Gender
/SHIH TZU MIX               M         1
2 DOGS: TERR X & DOBERMAN   F         1
AFRICAN BOERBOEL            M         1
AIREDALE TERRIER            M         1
AKITA/CHOW CHOW             M         1
dtype: int64
```

Combine dogs info with dig bites info in New York - data join

Now we can join the two data-sets together, to get both number of dogs and number of bites per breed and gender

In [11]:
```
bites_per_dog  = dogs_by_breed_and_gender.to_frame('dogs').join(
    dog_bites_by_breed_and_gender.to_frame('bites'), how='outer').\
    sort_values(by='dogs', ascending=False)

bites_per_dog.head()
```

Out[11]:

		dogs	bites
Breed	**Gender**		
Yorkshire Terrier	**M**	12588.0	110.0
Shih Tzu	**M**	11214.0	203.0
Yorkshire Terrier	**F**	9331.0	44.0
Shih Tzu	**F**	8414.0	66.0
Chihuahua	**M**	8164.0	144.0

Dogs in New York - data cleaning

For some breeds we may not have information about number of dogs or bites.

```
- If there are no known recorded bites for a breed, we replace it with 0.
- There may also be cases where we have a recorded bite for a breed where we
 have no known dog count. This is probably due to breed names not being spell
ed correctly. We drop these rows.
```

In [12]:
```
bites_per_dog[bites_per_dog.isna().any(axis=1)].sample(5)
```

Out[12]:

		dogs	bites
Breed	**Gender**		
BOXER-X	**M**	NaN	1.0
FELKY TERRIER	**M**	NaN	1.0
DOXXIN	**F**	NaN	1.0
SHARPEI/GERMAN SHEP X	**M**	NaN	1.0
Black Russian Terrier	**M**	13.0	NaN

Dogs in New York - data cleaning

In [13]:
```
bites_per_dog.dropna(subset=['dogs'], inplace=True)
bites_per_dog.fillna(0, inplace=True)
bites_per_dog.head()
```

Out[13]:

		dogs	bites
Breed	Gender		
Yorkshire Terrier	M	12588.0	110.0
Shih Tzu	M	11214.0	203.0
Yorkshire Terrier	F	9331.0	44.0
Shih Tzu	F	8414.0	66.0
Chihuahua	M	8164.0	144.0

Bites per dog breed in New York

Now we can caulculate the number of bites per dog for each breed/gender.

In [14]:
```
bites_per_dog['bites_per_dog'] = bites_per_dog.bites / bites_per_dog.dogs
bites_per_dog.head()
```

Out[14]:

		dogs	bites	bites_per_dog
Breed	Gender			
Yorkshire Terrier	M	12588.0	110.0	0.008738
Shih Tzu	M	11214.0	203.0	0.018102
Yorkshire Terrier	F	9331.0	44.0	0.004715
Shih Tzu	F	8414.0	66.0	0.007844
Chihuahua	M	8164.0	144.0	0.017638

Bites per dog breed in New York

Lets subset to breeds with many (>5000) data-points

In [15]:
```
bites_per_dog = bites_per_dog[bites_per_dog.dogs > 5000]
bites_per_dog.head()
```

Out[15]:

		dogs	bites	bites_per_dog
Breed	**Gender**			
Yorkshire Terrier	**M**	12588.0	110.0	0.008738
Shih Tzu	**M**	11214.0	203.0	0.018102
Yorkshire Terrier	**F**	9331.0	44.0	0.004715
Shih Tzu	**F**	8414.0	66.0	0.007844
Chihuahua	**M**	8164.0	144.0	0.017638

Bites per dog breed in New York - safest and most dangerous dog breeds are revelead

In [16]:
```
ax = bites_per_dog.bites_per_dog.sort_values().plot.bar(figsize=(16,3.5));
ax.set_ylabel('Bites per dog');
ax.set_xticklabels(ax.get_xticklabels(), rotation=20);
```