📦

01 Build a package

Dimensions to cover when building a package

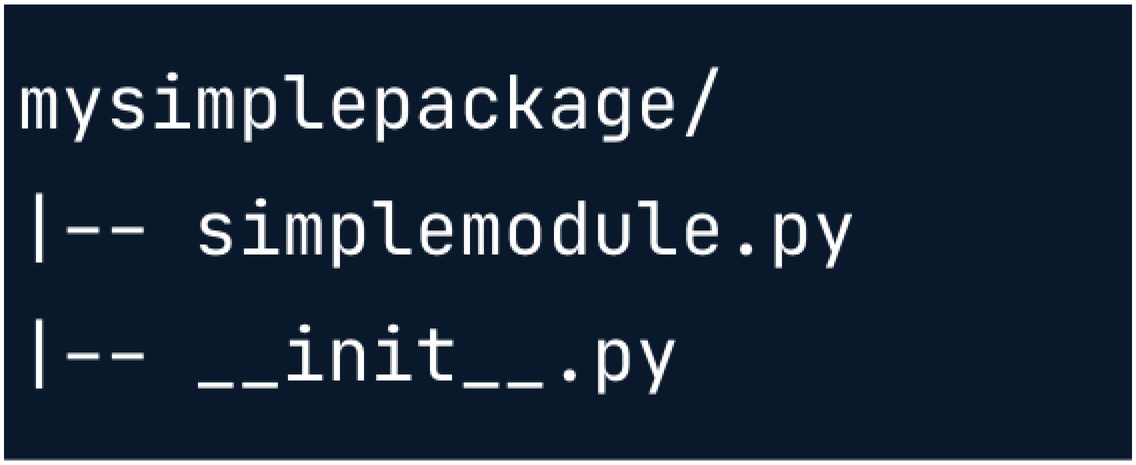
1. File Layout
2. Import Structure
3. Making your package installable
4. Adding licenses and readme
5. Style and unit test for a high quality package
6. Registering and publishing your package to PyPI

Parts of a package:

- Scripts
- Modules
- Packages (subpackages)

Scripts, modules, and packages

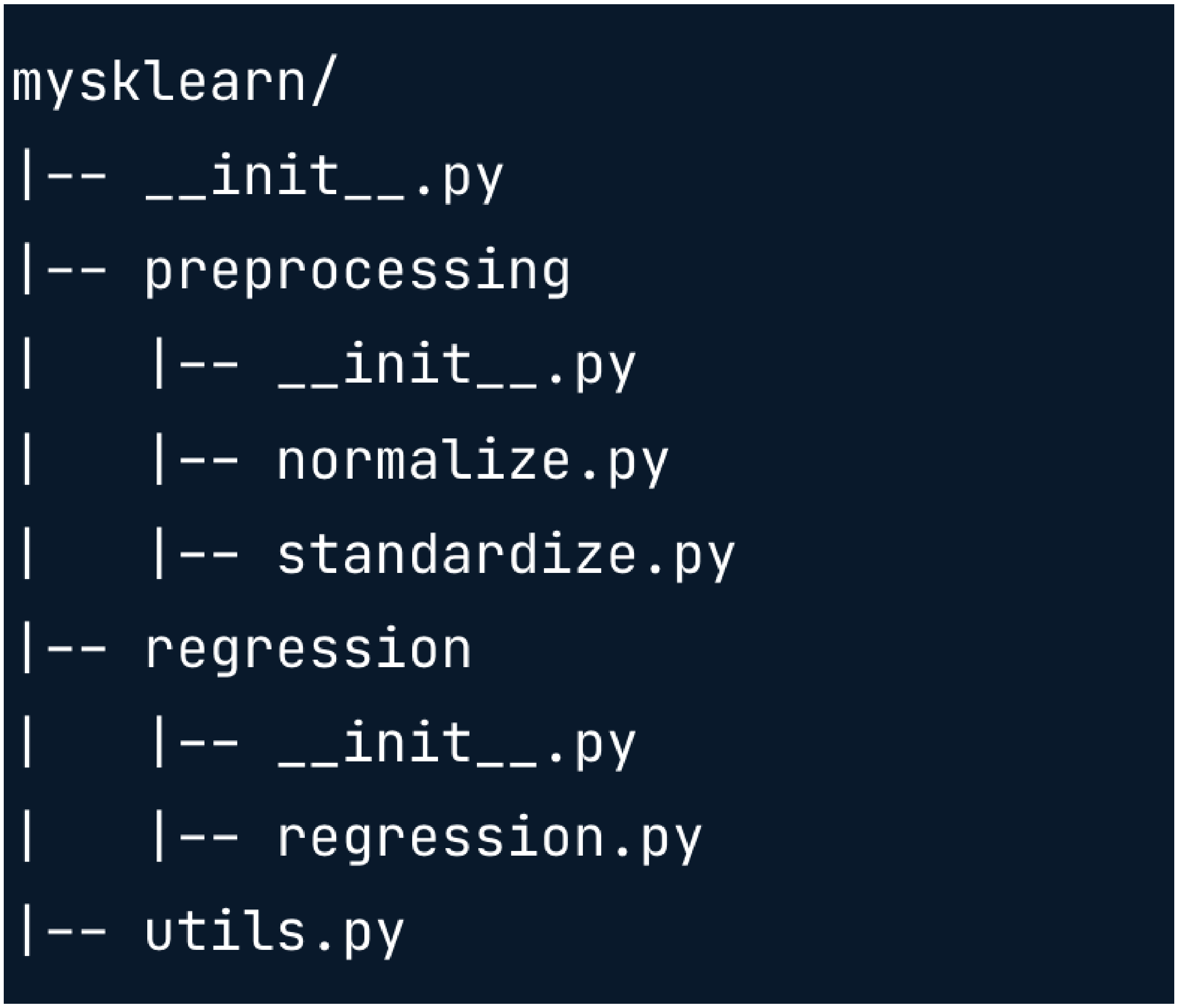
- Script - A Python file which is run like `python myscript.py` .
- Package - A directory full of Python code to be imported
  - e.g. `numpy` .
- Subpackage - A smaller package inside a package
  - e.g. `numpy.random` and `numpy.linalg` .
- Module - A Python file inside a package which stores the package code.
  - e.g. example coming in next 2 slide.
- Library - Either a package, or a collection of packages.
  - e.g., the Python standard library (`math` , `os` , `datetime` ,...)



```
mysimplepackage/
|-- simplemodule.py
|-- __init__.py
```

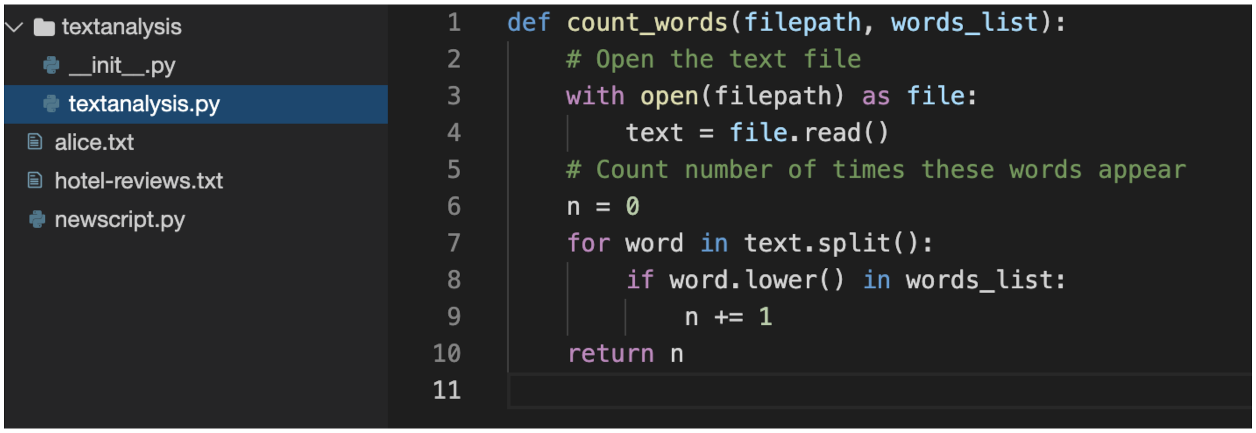
- This directory, called `mysimplepackage` , is a Python Package
- `simplemodule.py` contains all the package code
- `__init__.py` marks this directory as a Python package

Directory Tree for package with subpackages



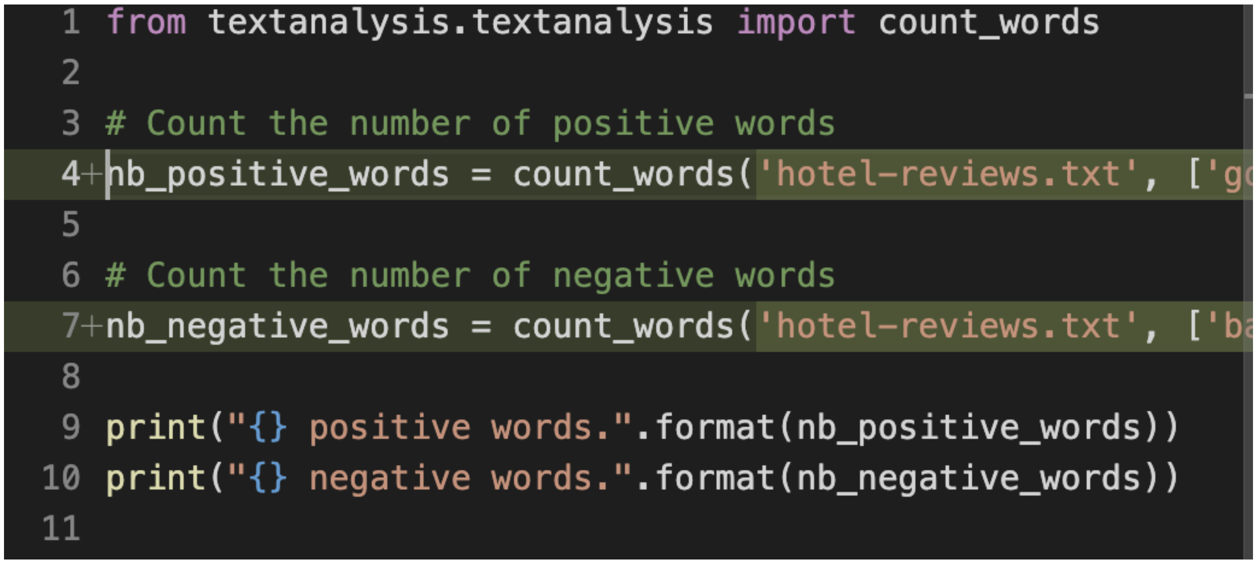
```
mysklearn/
|-- __init__.py
|-- preprocessing
|    |-- __init__.py
|    |-- normalize.py
|    |-- standardize.py
|-- regression
|    |-- __init__.py
|    |-- regression.py
|-- utils.py
```

Example of package very simple:



```
textanalysis
    __init__.py
    textanalysis.py
alice.txt
hotel-reviews.txt
newscript.py
```

```python
def count_words(filepath, words_list):
    # Open the text file
    with open(filepath) as file:
        text = file.read()
    # Count number of times these words appear
    n = 0
    for word in text.split():
        if word.lower() in words_list:
            n += 1
    return n
```



```python
from textanalysis.textanalysis import count_words

# Count the number of positive words
nb_positive_words = count_words('hotel-reviews.txt', ['go

# Count the number of negative words
nb_negative_words = count_words('hotel-reviews.txt', ['ba

print("{} positive words.".format(nb_positive_words))
print("{} negative words.".format(nb_negative_words))

```

Documentation

- Helps user use your code
- Document each
  - Functions
  - Class
  - Class method

Important that user can access documentation

```python
import numpy as np
help(np.sum)
```

```
...
sum(a, axis=None, dtype=None, out=None)
    Sum of array elements over a given axis.

    Parameters
    ----------
    a : array_like
        Elements to sum.
    axis : None or int or tuple of ints, optional
        Axis or axes along which a sum is performed.
        The default, axis=None, will sum all of the
        elements of the input array.
...
```

Function Documentation

```python
def count_words(filepath, words_list):
    """Count the total number of times these words appear.

    The count is performed on a text file at the given location.

    [explain what filepath and words_list are]

    [what is returned]
    """
```

Different documentation style, importance of consistency:

Google documentation style

```
"""Summary line.

Extended description of function.

Args:
    arg1 (int): Description of arg1
    arg2 (str): Description of arg2
```

NumPy style

```
    """Summary line.

    Extended description of function.

    Parameters
    ----------
    arg1 : int
        Description of arg1 ...
```

reStructured text style

```
    """Summary line.

    Extended description of function.

    :param arg1: Description of arg1
    :type arg1: int
    :param arg2: Description of arg2
    :type arg2: str
```

Epytext style

```
"""Summary line.

  Extended description of function.

  @type arg1: int
  @param arg1:  Description of arg1
  @type arg2: str
  @param arg2: Description of arg2
```

NumPy documentation style

Popular in scientific Python packages like

- `numpy`
- `scipy`
- `pandas`
- `sklearn`
- `matplotlib`
- `dask`
- etc.

NumPy documentation style

```python
import scipy
help(scipy.percentile)
```

```
percentile(a, q, axis=None, out=None, overwrite_input=False, interpolation='linear')
    Compute the q-th percentile of the data along the specified axis.

    Returns the q-th percentile(s) of the array elements.

    Parameters
    ----------
    a : array_like
        Input array or object that can be converted to an array.
```

Other types include - `int` , `float` , `bool` , `str` , `dict` , `numpy.array` , etc.

NumPy documentation style

```python
import scipy
help(scipy.percentile)
```

```
percentile(a, q, axis=None, out=None, overwrite_input=False, interpolation='linear')
    ...
    Parameters
    ----------
    ...
    axis : {int, tuple of int, None}
    ...
    interpolation : {'linear', 'lower', 'higher', 'midpoint', 'nearest'}
```

- List multiple types for parameter if appropriate
- List accepted values if only a few valid options

NumPy documentation style

```python
import scipy
help(scipy.percentile)
```

```
percentile(a, q, axis=None, out=None, overwrite_input=False, interpolation='linear')
    ...
    Returns
    -------
    percentile : scalar or ndarray
        If `q` is a single percentile and `axis=None`, then the result
        is a scalar. If multiple percentiles are given, first axis of
        the result corresponds to the percentiles...
    ...
```

▼ Other sections in the function documentation includes:

- Raises
- See Also
- Notes
- References
- Examples

Documentation templates and style translation

- piment can be used to generate docs strings
- Run from terminal
- Any documentation style from
    - Google
    - Numpydoc
    - reST
    - Javadoc
- Modify documentation from one style to the other

```
pyment -w -o numpydoc textanalysis.py
```

```python
def count_words(filepath, words_list):
    # Open the text file
    ...
    return n
```

- `-w` - overwrite file
- `-o numpydoc` - output in NumPy style

```
pyment -w -o numpydoc textanalysis.py
```

```python
def count_words(filepath, words_list):
    """

    Parameters
    ----------
    filepath :

    words_list :


    Returns
    -------
    type
    """
```

Package, subpackage and module documentation

`mysklearn/__init__.py`

```python
"""
Linear regression for Python
============================

mysklearn is a complete package for implmenting
linear regression in python.
```

`mysklearn/preprocessing/__init__.py`

```python
"""
A subpackage for standard preprocessing operations.
"""
```

`mysklearn/preprocessing/normalize.py`

```python
"""
A module for normalizing data.
"""
```

Importing subpackages into packages

`mysklearn/__init__.py`

Absolute import

```python
from mysklearn import preprocessing
```

- Used most - more explicit

Relative import

```python
from . import preprocessing
```

- Used sometimes - shorter and sometimes simpler

Directory tree for package with subpackages

```
mysklearn/
|-- __init__.py        <--
|-- preprocessing
|   |-- __init__.py
|   |-- normalize.py
|   |-- standardize.py
|-- regression
|   |-- __init__.py
|   |-- regression.py
|-- utils.py
```

Structuring Imports

Importing modules

`mysklearn/preprocessing/__init__.py`

Absolute import

```python
from mysklearn.preprocessing import normalize
```

Relative import

```python
from . import normalize
```

Directory tree for package with subpackages

```
mysklearn/
|-- __init__.py
|-- preprocessing
|   |-- __init__.py    <--
|   |-- normalize.py
|   |-- standardize.py
|-- regression
|   |-- __init__.py
|   |-- regression.py
|-- utils.py
```

Importing between sibling modules

In `normalize.py`

Absolute import

```
from mysklearn.preprocessing.funcs import (
    mymax, mymin
)
```

Relative import

```
from .funcs import mymax, mymin
```

Directory tree for package with subpackages

```
mysklearn/
|-- __init__.py
|-- preprocessing
|   |-- __init__.py
|   |-- normalize.py   <--
|   |-- funcs.py
|   |-- standardize.py
|-- regression
|   |-- __init__.py
|   |-- regression.py
|-- utils.py
```

Importing between modules far apart

A custom exception `MyException` is in `utils.py`

In `normalize.py`, `standardize.py` and `regression.py`

Absolute import

```
from mysklearn.utils import MyException
```

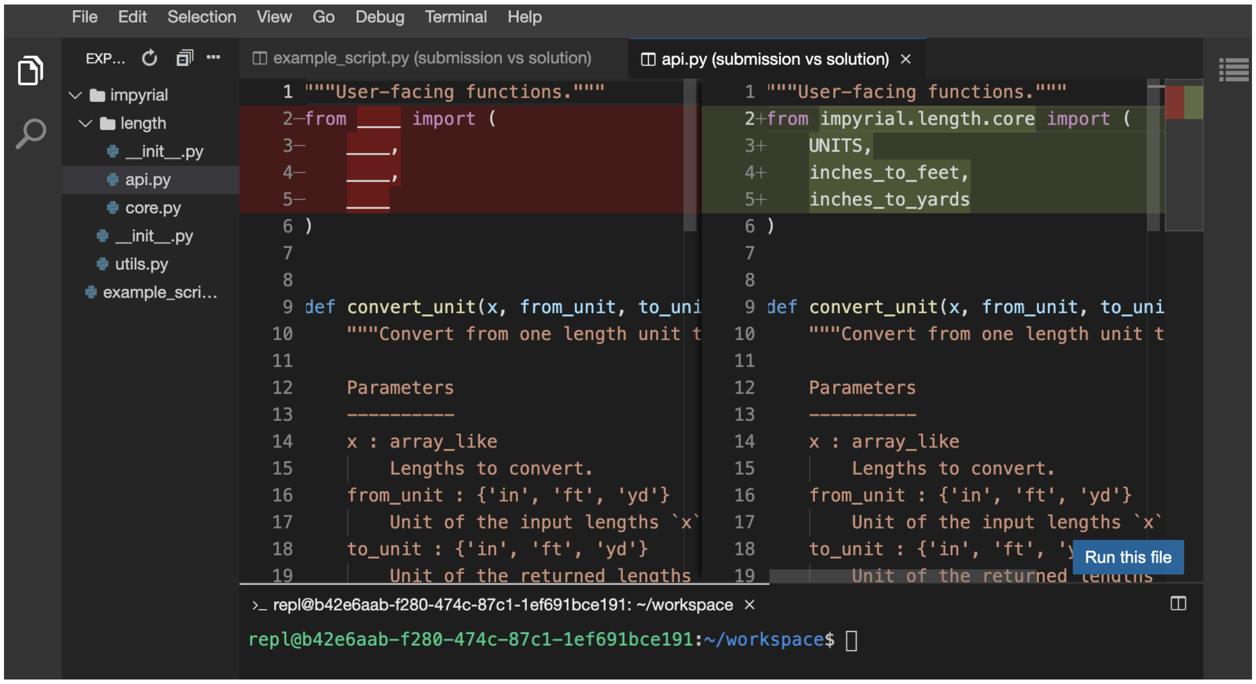
Relative import

```
from ..utils import MyException
```

Directory tree for package with subpackages

```
mysklearn/
|-- __init__.py
|-- preprocessing
|   |-- __init__.py
|   |-- normalize.py   <--
|   `-- standardize.py <--
|-- regression
|   |-- __init__.py
|   |-- regression.py  <--
`-- utils.py
```

Example


File Edit Selection View Go Debug Terminal Help
EXP...
impyrial
length
__init__.py
api.py
core.py
__init__.py
utils.py
example_scri...
example_script.py (submission vs solution)
api.py (submission vs solution)
Run this file

```
"""User-facing functions."""
from ____ import (
    ____,
    ____,
    ____
)


def convert_unit(x, from_unit, to_uni
    """Convert from one length unit t

    Parameters
    ----------
    x : array_like
        Lengths to convert.
    from_unit : {'in', 'ft', 'yd'}
        Unit of the input lengths `x`
    to_unit : {'in', 'ft', 'yd'}
        Unit of the returned lengths
```

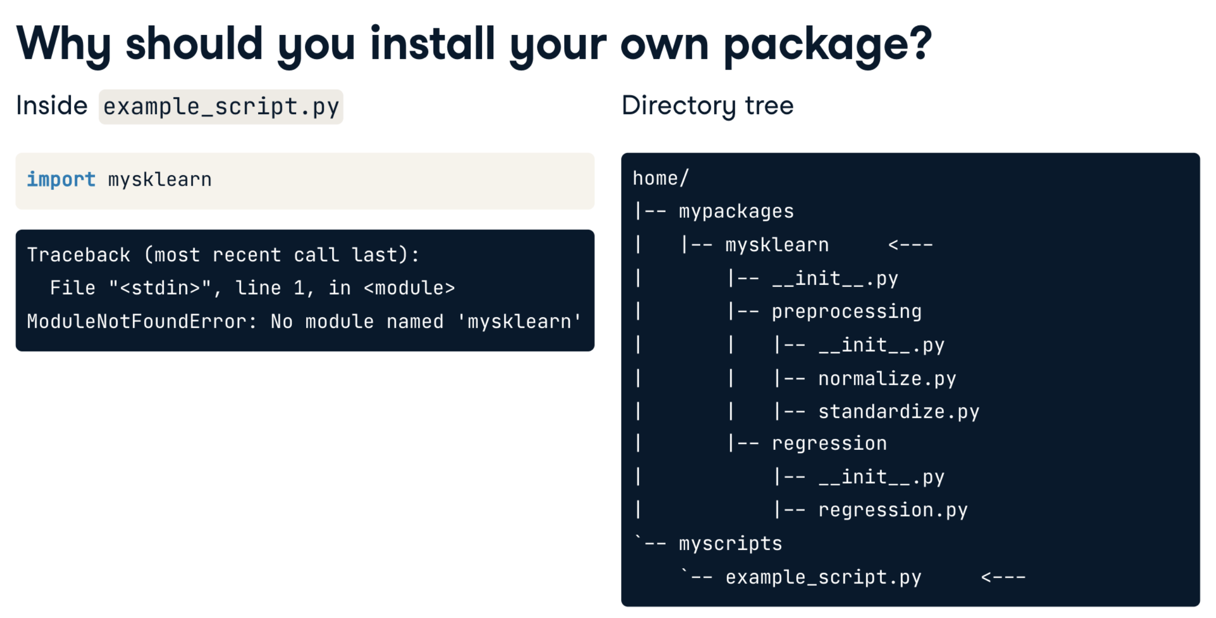
```
"""User-facing functions."""
from impyrial.length.core import (
    UNITS,
    inches_to_feet,
    inches_to_yards
)


def convert_unit(x, from_unit, to_uni
    """Convert from one length unit t

    Parameters
    ----------
    x : array_like
        Lengths to convert.
    from_unit : {'in', 'ft', 'yd'}
        Unit of the input lengths `x`
    to_unit : {'in', 'ft', 'y
        Unit of the returned lengths
```

```
repl@b42e6aab-f280-474c-87c1-1ef691bce191: ~/workspace
repl@b42e6aab-f280-474c-87c1-1ef691bce191:~/workspace$
```

Installing your own package



Why should you install your own package?

Inside `example_script.py`

```
import mysklearn
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ModuleNotFoundError: No module named 'mysklearn'
```

Directory tree

```
home/
|-- mypackages
|   |-- mysklearn      <---
|       |-- __init__.py
|       |-- preprocessing
|       |   |-- __init__.py
|       |   |-- normalize.py
|       |   |-- standardize.py
|       |-- regression
|           |-- __init__.py
|           |-- regression.py
`-- myscripts
    `-- example_script.py     <---
```

create a setup.py

- Is used to install the package
- Contains metadata on the package

Package directory structure

Directory tree for package with subpackages

```
mysklearn/      <-- outer directory
|-- mysklearn   <--- inner source code directory
|   |-- __init__.py
|   |-- preprocessing
|   |   |-- __init__.py
|   |   |-- normalize.py
|   |   |-- standardize.py
|   |-- regression
|   |   |-- __init__.py
|   |   |-- regression.py
|   |-- utils.py
|-- setup.py   <-- setup script in outer
```

Inside setup.py

```python
# Import required functions
from setuptools import setup

# Call setup function
setup(
    author="James Fulton",
    description="A complete package for linear regression.",
    name="mysklearn",
    version="0.1.0",
)
```

version number = (major number) . (minor number) . (patch number)

Editable installation

```
pip install -e .
```

- `.` = package in current directory
- `-e` = editable

Directory tree for package with subpackages

```
mysklearn/  <-- navigate to here
|-- mysklearn
|   |-- __init__.py
|   |-- preprocessing
|   |   |-- __init__.py
|   |   |-- normalize.py
|   |   |-- standardize.py
|   |-- regression
|   |   |-- __init__.py
|   |   |-- regression.py
|   |-- utils.py
|-- setup.py
```

---

pip install -e .

Dealing with dependencies

What are dependencies?

- Other packages you import inside your package
- Inside `mymodule.py` :

```python
# These imported packages are dependencies
import numpy as np
import pandas as pd
...
```

Adding dependencies to setup.py

```python
from setuptools import setup, find_packages

setup(
    ...
    install_requires=['pandas', 'scipy', 'matplotlib'],
)
```

Controlling dependency version

```python
from setuptools import setup, find_packages

setup(
    ...
    install_requires=[
        'pandas>=1.0',          # good
        'scipy==1.1',           # bad
        'matplotlib>=2.2.1,<3' # good
    ],
)
```

- Allow as many package versions as possible

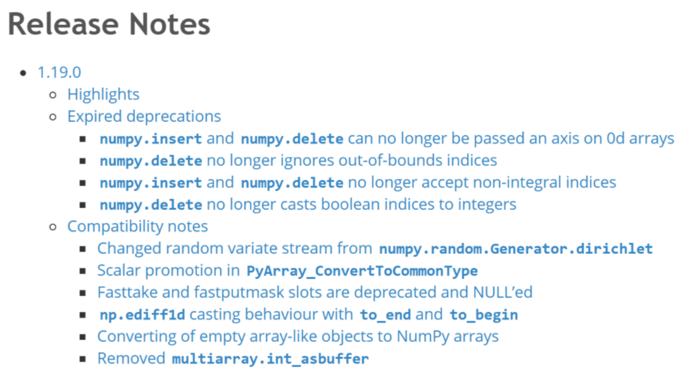
Python versions

```python
from setuptools import setup, find_packages

setup(
    ...
    python_requires='>=2.7, !=3.0.*, !=3.1.*',
)
```

Choosing dependency and package versions

- Check the package history or release notes
  - e.g. the **NumPy release notes**
- Test different versions



Release Notes

- 1.19.0
  - Highlights
  - Expired deprecations
    - `numpy.insert` and `numpy.delete` can no longer be passed an axis on 0d arrays
    - `numpy.delete` no longer ignores out-of-bounds indices
    - `numpy.insert` and `numpy.delete` no longer accept non-integral indices
    - `numpy.delete` no longer casts boolean indices to integers
  - Compatibility notes
    - Changed random variate stream from `numpy.random.Generator.dirichlet`
    - Scalar promotion in `PyArray_ConvertToCommonType`
    - Fasttake and fastputmask slots are deprecated and NULL'ed
    - `np.ediff1d` casting behaviour with `to_end` and `to_begin`
    - Converting of empty array-like objects to NumPy arrays
    - Removed `multiarray.int_asbuffer`

Making an environment for developers

```
pip freeze
```

```
alabaster==0.7.12
appdirs==1.4.4
argh==0.26.2
...
wrapt==1.11.2
yapf==0.29.0
zipp==3.1.0
```

Making an environment for developers

Save package requirements to a file

```
pip freeze > requirements.txt
```

Install requirements from file

```
pip install -r requirements.txt
```

```
mysklearn/
|-- mysklearn
|   |-- __init__.py
|   |-- preprocessing
|   |   |-- __init__.py
|   |   |-- normalize.py
|   |   |-- standardize.py
|   |-- regression
|   |   |-- __init__.py
|   |   |-- regression.py
|   |-- utils.py
|-- setup.py
|-- requirements.txt   <-- developer environment
```

What to include in a README

README sections

- Title
- Description and Features
- Installation
- Usage examples
- Contributing
- License

README format

Markdown (commonmark)

- Contained in `README.md` file
- Simpler
- Used in this course and in the wild

reStructuredText

- Contained in `README.rst` file
- More complex
- Also common in the wild

Distributions

- **Distribution package** - a bundled version of your package which is ready to install.
- **Source distribution** - a distribution package which is mostly your source code.
- **Wheel distribution** - a distribution package which has been processed to make it faster to install.

How to build distributions

`python setup.py sdist bdist_wheel`

```
mysklearn/
|-- mysklearn
|-- setup.py
|-- requirements.txt
|-- LICENSE
|-- README.md
```

Getting your package out there

Upload your distributions to PyPI

```
twine upload dist/*
```

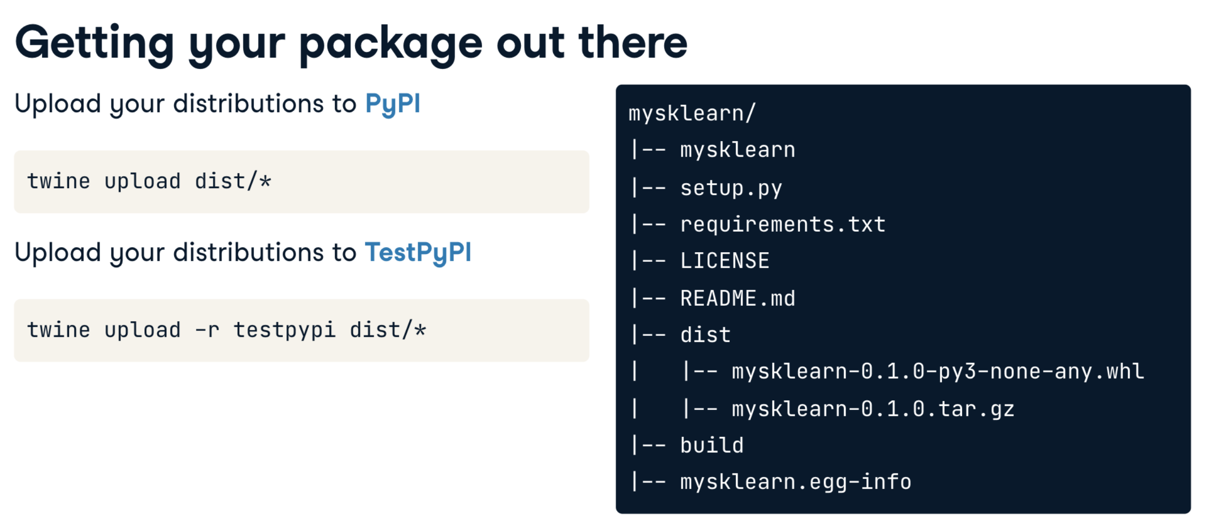
Upload your distributions to TestPyPI

```
twine upload -r testpypi dist/*
```

```
mysklearn/
|-- mysklearn
|-- setup.py
|-- requirements.txt
|-- LICENSE
|-- README.md
|-- dist
|   |-- mysklearn-0.1.0-py3-none-any.whl
|   |-- mysklearn-0.1.0.tar.gz
|-- build
|-- mysklearn.egg-info
```

```
python3 setup.py sdist bdist_wheel
```

Testing your package



Getting your package out there

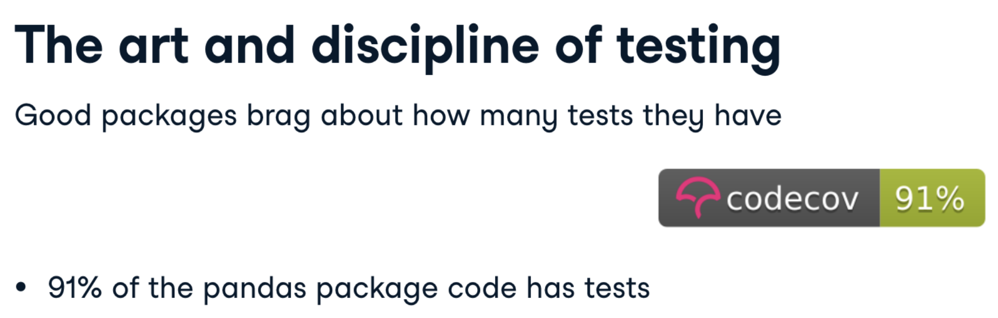
Upload your distributions to PyPI

```
twine upload dist/*
```

Upload your distributions to TestPyPI

```
twine upload -r testpypi dist/*
```

```
mysklearn/
|-- mysklearn
|-- setup.py
|-- requirements.txt
|-- LICENSE
|-- README.md
|-- dist
|    |-- mysklearn-0.1.0-py3-none-any.whl
|    |-- mysklearn-0.1.0.tar.gz
|-- build
|-- mysklearn.egg-info
```



The art and discipline of testing

Good packages brag about how many tests they have

codecov 91%

- 91% of the pandas package code has tests

Organizing tests inside your package

Test directory layout

```
mysklearn/tests/
|-- __init__.py
|-- preprocessing
```

Code directory layout

```
mysklearn/mysklearn/
|-- __init__.py
|-- preprocessing
```

Organizing a test module

Inside `test_normalize.py`

```python
from mysklearn.preprocessing.normalize import (
    find_max, find_min, normalize_data
)

def test_find_max(x):
    assert find_max([1,4,7,1])==7

def test_find_min(x):
    assert ...

def test_normalize_data(x):
    assert ...
```

Inside `normalize.py`

```python
def find_max(x):
    ...
    return x_max

def find_min(x):
    ...
    return x_min

def normalize_data(x):
    ...
    return x_norm
```

Running tests with pytest

```
pytest
```

- `pytest` looks inside the `test` directory
- It looks for modules like `test_modulename.py`

```
mysklearn/ <-- navigate to here
|-- mysklearn
|-- tests
|-- setup.py
|-- LICENSE
|-- MANIFEST.in
```

- You can make a new directory from the terminal using the command `mkdir DirectoryName`.
- If a source module is at `mypackage/mysubpackage/mymodule.py` then the test module for this file should be at `tests/mysubpackage/test_mymodule.py`.
- You can create an empty file from the terminal using the command `touch filename.py`.
- An absolute import starts with the package name, i.e. `from mypackage.mysubpackage.mymodule import myfunction1, myfunction2`

```
from impyrial.length.core import inches_to_feet, inches_to_yards

# Define tests for inches_to_feet function
def test_inches_to_feet():
  # Check that 12 inches is converted to 1.0 foot
    assert inches_to_feet(12) == 1.0
    # Check that 2.5 feet is converted to 30.0 inches
    assert inches_to_feet(2.5, reverse=True) == 30.0
```

Testing your package with different environments

Testing multiple versions of Python

This `setup.py` allows any version of Python from version 2.7 upwards.

```
from setuptools import setup, find_packages

setup(
    ...
    python_requires='>=2.7',
)
```

To test these Python versions you must:

- Install all these Python versions
- Run `tox`

Configure tox

Configuration file - `tox.ini`

```
[tox]
envlist = py27, py35, py36, py37

[testenv]
deps = pytest
commands =
    pytest
    echo "run more commands"
    ...
```

- Headings are surrounded by square brackets `[...]` .
- To test Python version X.Y add `pyXY` to `envlist` .
- The versions of Python you test need to be installed already.
- The `commands` parameter lists the terminal commands `tox` will run.
- The `commands` list can be any commands which will run from the terminal, like `ls` , `cd` , `echo` etc.

tox output

```
py27 run-test: commands[0] | pytest
======================== test session starts ========================
platform linux2 -- Python 2.7.17, ...
rootdir: /home/workspace/mypackages/mysklearn
collected 6 items

tests/preprocessing/test_normalize.py ...                     [ 50%]
tests/preprocessing/test_standardize.py ...                   [100%]

========================= 6 passed in 0.23s =========================
```