Faster package development with templates

Templates

- Python packages have lots of extra files
- There is a lot to remember
- Using templates takes care of a lot of this

Package file tree

```
.
|-- example_package
|   |-- __init__.py
|   `-- example_package.py
|-- tests
|   |-- __init__.py
|   `-- test_example_package.py
|-- README.rst      <-
|-- LICENSE         <-- lots
|-- MANIFEST.in     <-- of
|-- tox.ini         <-- additional
|-- setup.py        <-- files
`-- setup.cfg       <--
```

cookiecutter

- Can be used to create empty Python packages
- Creates all the additional files your package needs

Package file tree

```
.
|-- example_package
|   |-- __init__.py
|   `-- example_package.py
|-- tests
|   |-- __init__.py
|   `-- test_example_package.py
|-- README.rst
|-- LICENSE
|-- MANIFEST.in
|-- tox.ini
|-- setup.py
`-- setup.cfg
```

Using cookiecutter

```
cookiecutter https://github.com/audreyr/cookiecutter-pypackage
```

```
full_name [Audrey Roy Greenfeld]: James Fulton
email [audreyr@example.com]: james@email.com
github_username [audreyr]:  MyUsername
project_name [Python Boilerplate]: mysklearn
project_slug [mysklearn]: mysklearn
```

- Project slug - the `name` used in `pip install name`

3. cookiecutter
Cookiecutter is a command line tool which creates projects from templates. You can use it to create a basic, empty Python package, like this one. These templates can create all of the additional files which your package needs, so you can focus more on the code, and won't need to worry you have forgotten something.

4. Using cookiecutter
You run cookiecutter from the terminal, and pass it the URL of the template you would like to use.

Using cookiecutter

```
cookiecutter https://github.com/audreyr/cookiecutter-pypackage
```

```
...
project_short_description [Python Boilerplate ...]: A Python package for linear
    regression.
pypi_username [MyUsername]:
version [0.1.0]:
```

Using cookiecutter

```
cookiecutter https://github.com/audreyr/cookiecutter-pypackage
```

```
...
use_pytest [n]: y
use_pypi_deployment_with_travis [y]: n
add_pyup_badge [n]: n
```

5. Using cookiecutter
The standard template for Python packages is this one. But there are lots more templates you can use choose from.
6. Using cookiecutter
When you run this, it will ask you to fill in some details about the package you are creating. The first field you must fill in is your full name. Each field has a default value in square brackets. In this case the default is the name of the template author.

Using cookiecutter

```
cookiecutter https://github.com/audreyr/cookiecutter-pypackage
```

```
...
Select command_line_interface:
1 - Click
2 - Argparse
3 - No command-line interface
Choose from 1, 2, 3 [1]: 3
create_author_file [y]: y
```

7. Using cookiecutter
This value will be used to set the author name inside the package setup-dot-py file. You'll need to type yours, and press enter to continue.

8. Using cookiecutter
It will prompt you for some more details used to fill in the template. This includes the package name and slug. The slug is just the name you want the package to be registered under. The package will be pip-installed with this name.

Using cookiecutter

```
cookiecutter https://github.com/audreyr/cookiecutter-pypackage
```

```
...
Select open_source_license:
1 - MIT license
2 - BSD license
3 - ISC license
4 - Apache Software License 2.0
5 - GNU General Public License v3
6 - Not open source
Choose from 1, 2, 3, 4, 5, 6 [1]: 6
```

Template output

```
mysklearn/
|-- mysklearn/
|   |-- __init__.py
|   `-- mysklearn.py
|-- tests/
|   |-- __init__.py
|   `-- test_mysklearn.py
|-- MANIFEST.in
|-- README.rst
|-- requirements_dev.txt
|-- setup.cfg
|-- setup.py
|-- tox.ini
|-- AUTHORS.rst
|-- CONTRIBUTING.rst
|-- HISTORY.rst
`-- Makefile
```

Inside the `AUTHORS.rst` file

```
=======
Credits
=======

Development Lead
----------------

* James Fulton <james@example.com>

Contributors
------------

None yet. Why not be the first?
```

Version numbers and history

Final files

- `CONTRIBUTING.md`
- `HISTORY.md`

```
mysklearn/
|-- mysklearn/
|   |-- __init__.py
|   `-- mysklearn.py
|-- tests/
|   |-- __init__.py
|   `-- test_mysklearn.py
|-- MANIFEST.in
|-- README.md
|-- requirements_dev.txt
|-- setup.cfg
|-- setup.py
|-- tox.ini
|-- AUTHORS.md
|-- CONTRIBUTING.md  <-- new files
|-- HISTORY.md       <--
`-- Makefile
```

CONTRIBUTING.md

- Either markdown or reStructured-Text
- Invites other developers to work on your package
- Tells them how to get started

HISTORY.md

e.g. NumPy release notes

- Known as history, changelog or release notes
- Tells users what has changed between versions

HISTORY.md

- Section for each released version
- Bullet points of the important changes
- Subsections for
  - Improvements to existing functions
  - New additions
  - Bugs that have been fixed
  - Deprecations

```
# History

## 0.3.0
### Changed
- Regression fitting sped up using NumPy operations.
### Deprecated
- Support for Python 3.5 has ended.
- `regression.regression` module has been removed.

## 0.2.1
### Fixed
- Fixed bug causing intercepts of zero.

## 0.2.0
### Added
- Multiple linear regression now available in new
  `regression.multiple_regression` module.
### Deprecated
- 0.2.x will be the last version that supports Python 3.5.
- `regression.regression` module has been renamed
  `regression.single_regression`. `regression.regression` will be
  removed in next minor release.
```

History

0.3.0

Changed
- Regression fitting sped up using NumPy operations.

Deprecated
- Support for Python 3.5 has ended.
- `regression.regression` module has been removed.

0.2.1

Fixed
- Fixed bug causing intercepts of zero.

0.2.0

Added
- Multiple linear regression now available in new `regression.multiple_regression` module.

Deprecated
- 0.2.x will be the last version that supports Python 3.5.
- `regression.regression` module has been renamed `regression.single_regression`. `regression.regression` will be removed in next minor release.

Version number

- Increase version number when ready for new release
- Cannot upload to PyPI if not changed

```
mysklearn/
|-- mysklearn/
|   |-- __init__.py          <---
|   `-- mysklearn.py
|-- setup.py                 <---
...
```

The package version number

`setup.py`

```
# Import required functions
from setuptools import setup, find_packages

# Call setup function
setup(
    ...
    version='0.1.0',  <---
    ...

)
```

Top level `__init__.py`

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

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

__version__ = '0.1.0'  <---
```

```
print(mysklearn.__version__)
```

```
0.1.0
```

bumpversion

- Convenient tool to update all package version numbers

```
bumpversion major
```

```
bumpversion minor
```

```
bumpversion patch
```

```
mysklearn/    <--   navigate to here
|-- mysklearn/
|   |-- __init__.py
|   `-- mysklearn.py
|-- setup.py
...
```

Makefiles and classifiers

Classifiers

- Metadata for your package
- Helps users find your package on PyPI
- You should include
  - Package status
  - Your intended audience
  - License type
  - Language
  - Versions of Python supported
- Lots more classifiers exist
  https://pypi.org/classifiers

Inside `setup.py` of mysklearn

```
setup(
    ...
    classifiers=[
        'Development Status :: 2 - Pre-Alpha',
        'Intended Audience :: Developers',
        'License :: OSI Approved :: MIT License',
        'Natural Language :: English',
        'Programming Language :: Python :: 3',
        'Programming Language :: Python :: 3.6',
        'Programming Language :: Python :: 3.7',
        'Programming Language :: Python :: 3.8',
    ],
    ...
)
```

What is in a Makefile?

Inside `Makefile`

```
...

dist: ## builds source and wheel package
    python3 setup.py sdist bdist_wheel

clean-build: ## remove build artifacts
    rm -fr build/
    rm -fr dist/
    rm -fr .eggs/

test: ## run tests quickly with the default Python
    pytest

release: dist ## package and upload a release
    twine upload dist/*
```

How do I use the Makefile?

```
make <function-name>
```

```
mysklearn/ <--- navigate to here
...
|-- README.md
|-- setup.py
|-- Makefile
...
```

How do I use the Makefile?

To use the dist function type this in terminal

```
make dist
```

Inside `Makefile`

```
...

dist: ## builds source and wheel package
    python3 setup.py sdist bdist_wheel

clean-build: ## remove build artifacts
    rm -fr build/
    rm -fr dist/
    rm -fr .eggs/

test: ## run tests quickly with the default Python
    pytest

release: dist ## package and upload a release
    twine upload dist/*
```

Makefile summary

```
make help
```

```
clean                remove all build, test, coverage and Python artifacts
clean-build          remove build artifacts
clean-pyc            remove Python file artifacts
clean-test           remove test and coverage artifacts
lint                 check style with flake8
test                 run tests quickly with the default Python
test-all             run tests on every Python version with tox
release              package and upload a release
dist                 builds source and wheel package
install              install the package to the active Python's site-packages
```

Wrap-up

50 XP

Recap

- Modules vs subpackages vs packages
- Package structure and `__init__.py`
- Absolute and relative imports
- Documentation with `pyment`
- Code style with `flake8`
- Making your package installable with `setup.py`
- Dependencies with `install_requires` and `requirements.txt`
- Supporting files like `LICENSE`, `README.md`, `CONTRIBUTING.md` and `HISTORY.md`
- Building and uploading distributions to PyPI with `twine`
- Testing with `pytest` and `tox`
- Using package templates with `cookiecutter`
- Efficient package care with `Makefile` s