TA Section Week 3

9/20/2018 8-9PM

Zhenyu Zhao

DevOps Engineer

Infrastructure Technology Services

Harvard University IT

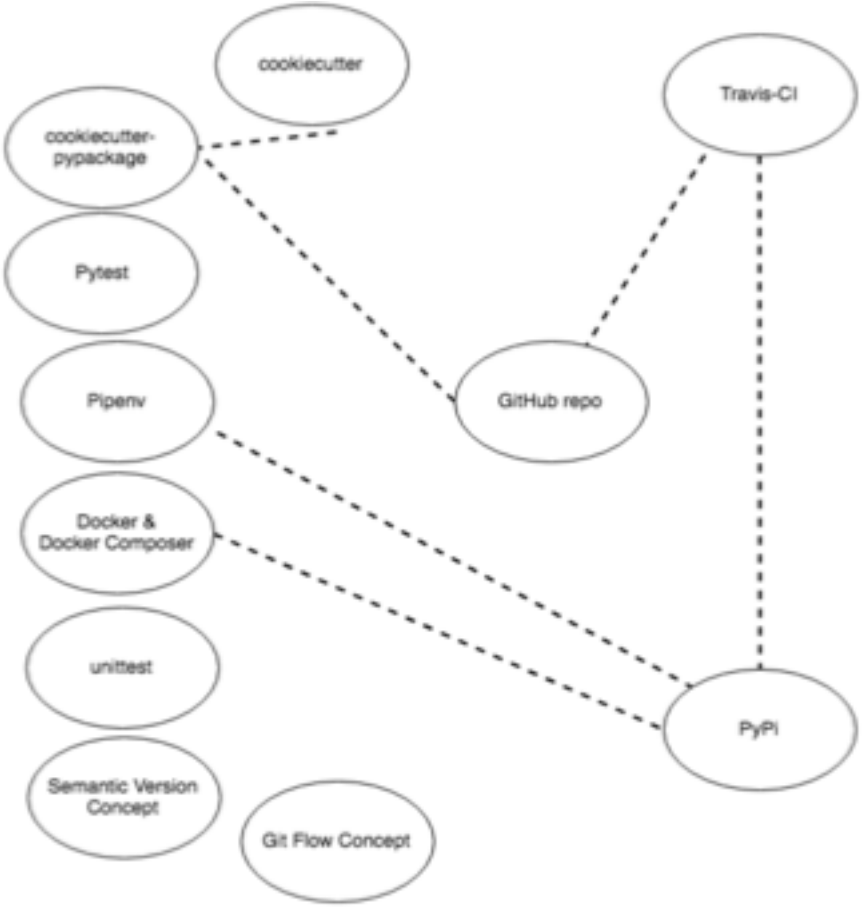
Email: Zhenyu_zhao@Harvard.edu

Agenda

❑Pset1 Anatomy

❑Word Embedding Example

Pset1 – Overview



Pset1 – Step 1

➢ **Two classroom repos that you need to work on**

❑cookiecutter-csci-package-<username>

❑pset_utils-<username>

You need to fork both repos into GitHub Classroom. The same for the future assignment.

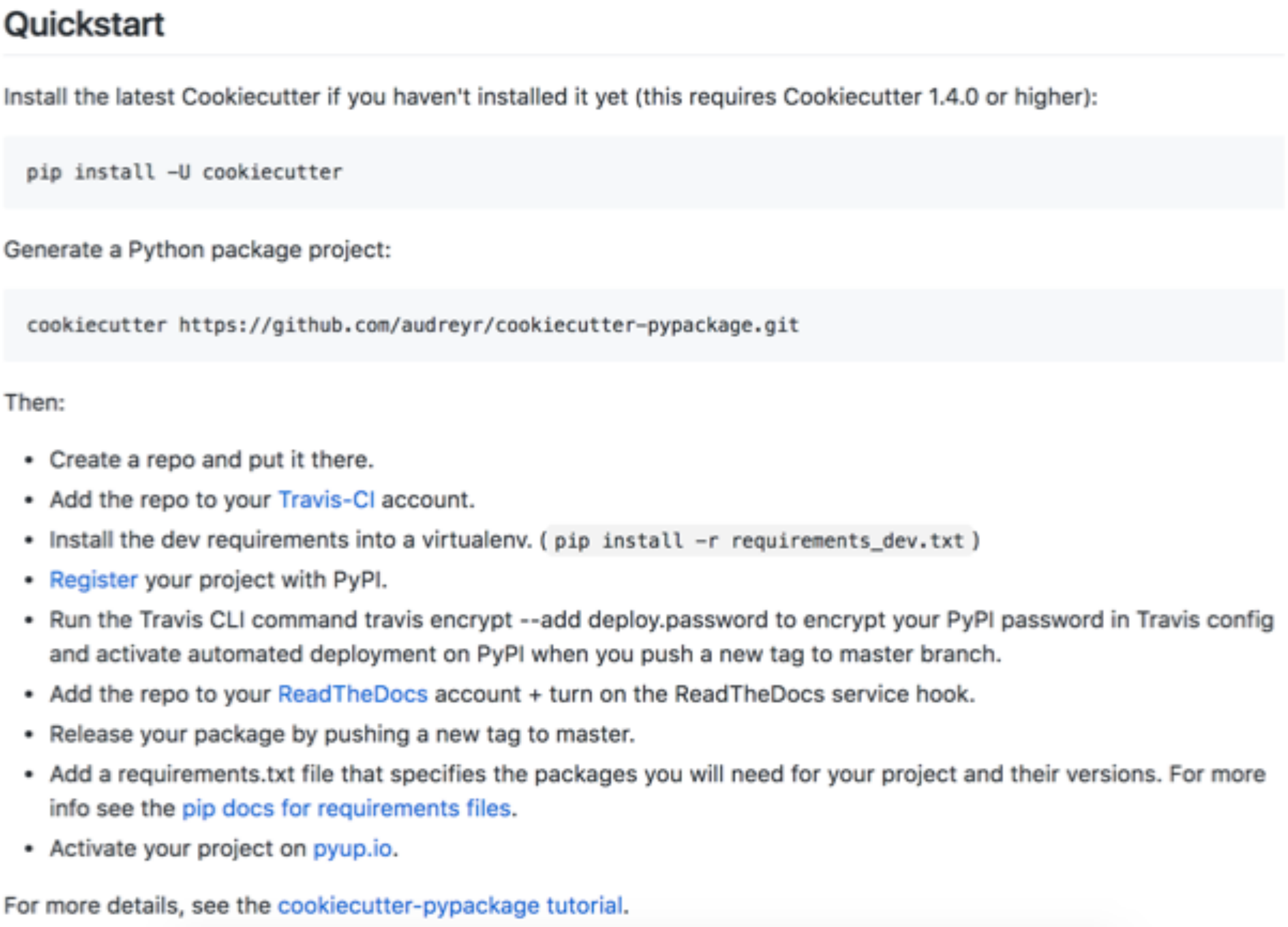
➢ cookiecutter-csci-<username> is actually a fork of the cookiecutter-pypackage repo (https://github.com/audreyr/cookiecutter-pypackage/) so it is just a template.

❑The good strategy is that you don't make a local clone on your computer yet but you follow the instructions in README.rst to try out each operations.

❑You will be in good shape to understand how the template works with the cookiecutter program.

❑How do you Install cookiecutter?

```
$ pip install cookiecutter
```



Quickstart

Install the latest Cookiecutter if you haven't installed it yet (this requires Cookiecutter 1.4.0 or higher):

```
pip install -U cookiecutter
```

Generate a Python package project:

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

Then:

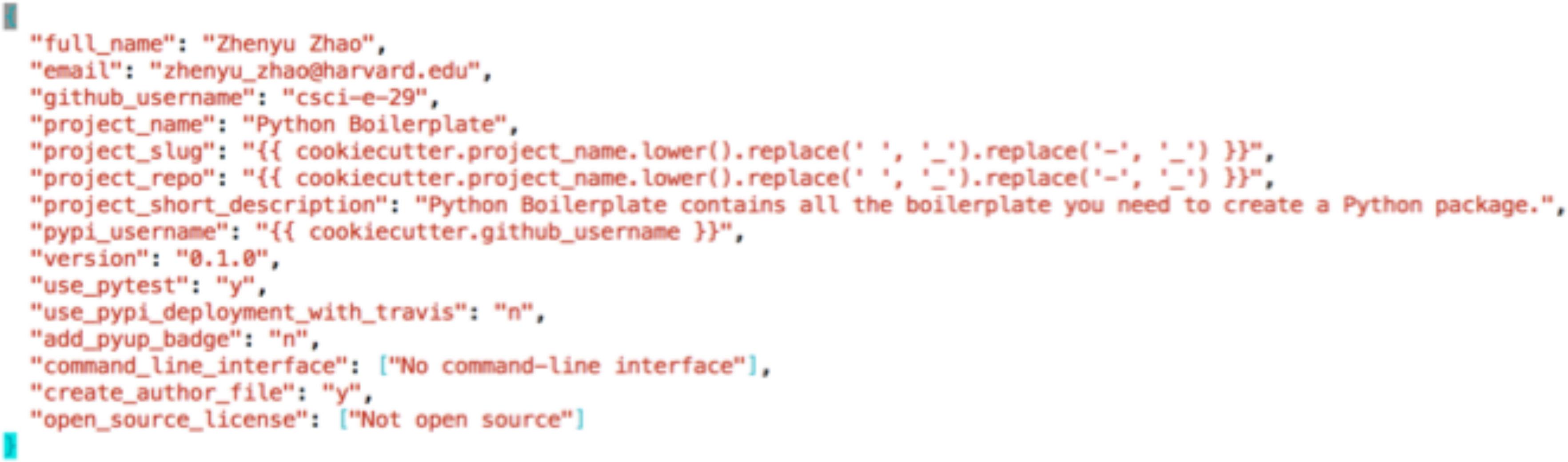
- Create a repo and put it there.
- Add the repo to your Travis-CI account.
- Install the dev requirements into a virtualenv. (`pip install -r requirements_dev.txt`)
- Register your project with PyPI.
- Run the Travis CLI command travis encrypt --add deploy.password to encrypt your PyPI password in Travis config and activate automated deployment on PyPI when you push a new tag to master branch.
- Add the repo to your ReadTheDocs account + turn on the ReadTheDocs service hook.
- Release your package by pushing a new tag to master.
- Add a requirements.txt file that specifies the packages you will need for your project and their versions. For more info see the pip docs for requirements files.
- Activate your project on pyup.io.

For more details, see the cookiecutter-pypackage tutorial.

Pset1 – Step 2

- **pset_utils-<username> is the classroom repo that you need to work on the most.**

❑ Inspect cookiecutter.json. Where is cookiecutter.json? It is the ~/.cookiecutters/cookiecutter-pypackage/cookiecutter.json file.



```
{
  "full_name": "Zhenyu Zhao",
  "email": "zhenyu_zhao@harvard.edu",
  "github_username": "csci-e-29",
  "project_name": "Python Boilerplate",
  "project_slug": "{{ cookiecutter.project_name.lower().replace(' ', '_').replace('-', '_') }}",
  "project_repo": "{{ cookiecutter.project_name.lower().replace(' ', '_').replace('-', '_') }}",
  "project_short_description": "Python Boilerplate contains all the boilerplate you need to create a Python package.",
  "pypi_username": "{{ cookiecutter.github_username }}",
  "version": "0.1.0",
  "use_pytest": "y",
  "use_pypi_deployment_with_travis": "n",
  "add_pyup_badge": "n",
  "command_line_interface": ["No command-line interface"],
  "create_author_file": "y",
  "open_source_license": ["Not open source"]
}
```

❑ Clone a local repo of cookiecutter-csci-package-<username>. This is the template that you need to modify.

$git clone git@github.com:username/cookiecutter-csci-package-<username>.git

Pset1 – Step 3

➢ **Start making changes to the local cookiecutter-csci-package-<username>**

❑Check the file system structure

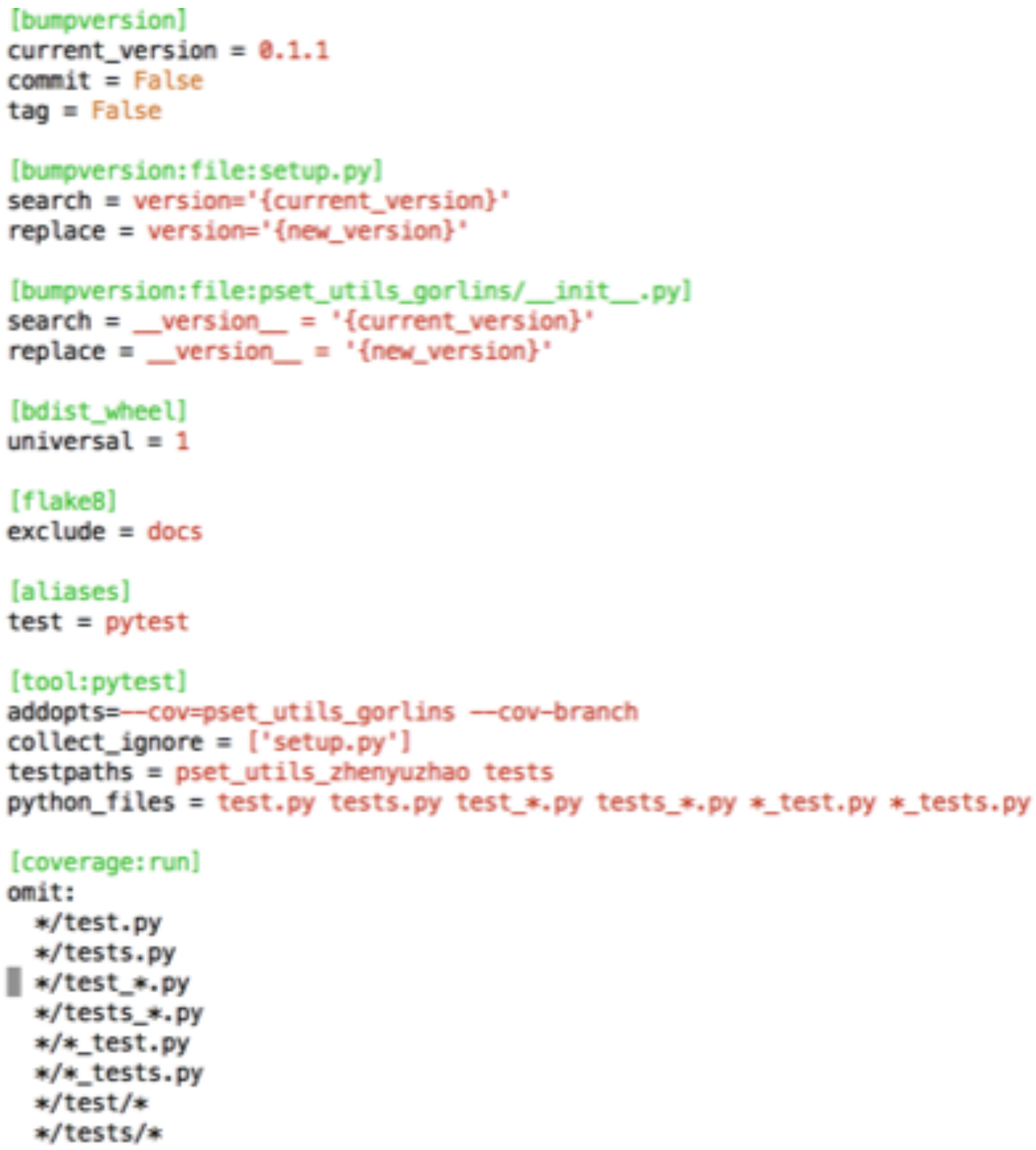
❑Remove the weird file {{project_slug}}/{{project_slug}}.py

❑rename the top-level {{cookiecutter.project_slug}} to {{cookiecutter.project_repo}}.

❑Fix the import in tests/test_{{project_slug}}.py

❑Remove python 2 and versions below 3.6 from tox.ini, setup.py, and .travis.yml. You can keep 3.7 if you like.

❑Tweak setup.cfg for pytest and coverage. You need to change both setup.cfg shown as follows:



```
[bumpversion]
current_version = 0.1.1
commit = False
tag = False

[bumpversion:file:setup.py]
search = version='{current_version}'
replace = version='{new_version}'

[bumpversion:file:pset_utils_gorlins/__init__.py]
search = __version__ = '{current_version}'
replace = __version__ = '{new_version}'

[bdist_wheel]
universal = 1

[flake8]
exclude = docs

[aliases]
test = pytest

[tool:pytest]
addopts=--cov=pset_utils_gorlins --cov-branch
collect_ignore = ['setup.py']
testpaths = pset_utils_zhenyuzhao tests
python_files = test.py tests.py test_*.py tests_*.py *_test.py *_tests.py

[coverage:run]
omit:
  */test.py
  */tests.py
  */test_*.py
  */tests_*.py
  */*_test.py
  */*_tests.py
  */test/*
  */tests/*
```

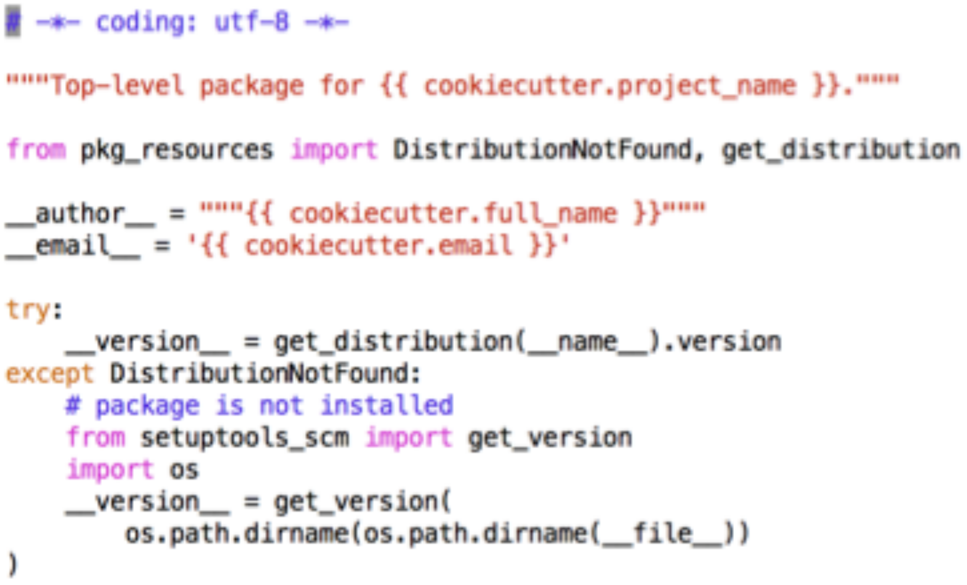
Pset1 – Step 4

➢ **Better versioning**

❑Delete all references to bumpversion in setup.cfg and the manual version specifiers it lists.

❑Update your setup.py and requirements_dev as specified in the help docs.

❑Make changes to {{ project_slug }}/__init__.py so it looks like:

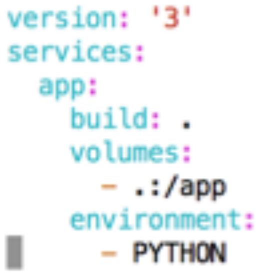


```
# -*- coding: utf-8 -*-

"""Top-level package for {{ cookiecutter.project_name }}."""

from pkg_resources import DistributionNotFound, get_distribution

__author__ = """{{ cookiecutter.full_name }}"""
__email__ = '{{ cookiecutter.email }}'

try:
    __version__ = get_distribution(__name__).version
except DistributionNotFound:
    # package is not installed
    from setuptools_scm import get_version
    import os
    __version__ = get_version(
        os.path.dirname(os.path.dirname(__file__))
)
```

Pset1 – Step 5

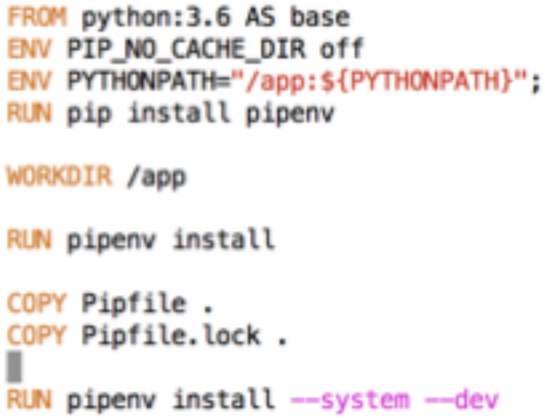
- ➢ **Convert to Docker and Pipenv**

❑Tweak docker-compose.yml



```
version: '3'
services:
  app:
    build: .
    volumes:
      - .:/app
    environment:
      - PYTHON
```

❑Tweak Dockerfile



```
FROM python:3.6 AS base
ENV PIP_NO_CACHE_DIR off
ENV PYTHONPATH="/app:${PYTHONPATH}";
RUN pip install pipenv

WORKDIR /app

RUN pipenv install

COPY Pipfile .
COPY Pipfile.lock .

RUN pipenv install --system --dev
```

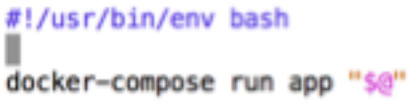
Pset1 – Step 5 (Continue)

❑**Create a shortcut executable drun_app**

```
$ touch drun_app
```

```
$ chmod +x drun_app
```

❑drun_app contains:



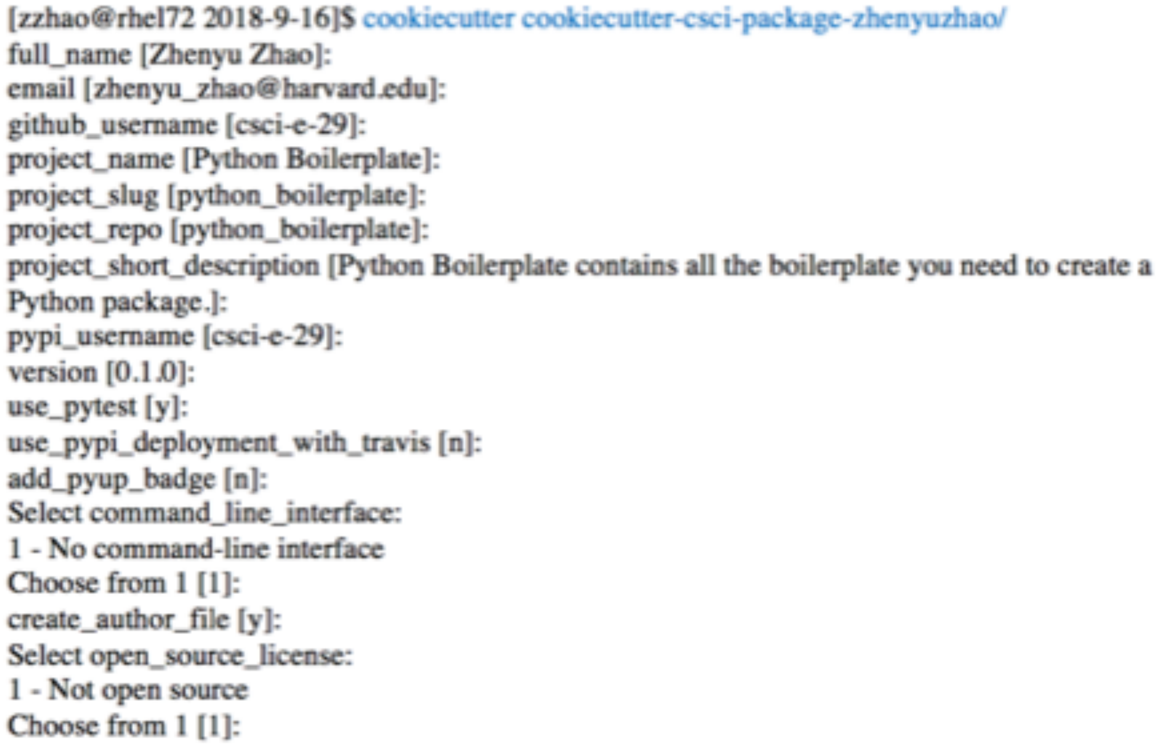
```
#!/usr/bin/env bash

docker-compose run app "$@"
```

Make sure that drun_app is placed in the {{cookiecutter.project_repo}} directory so it gets copied to the templace instances.

Pset1 – Step 6

➢ **Initialize the Pipfile**

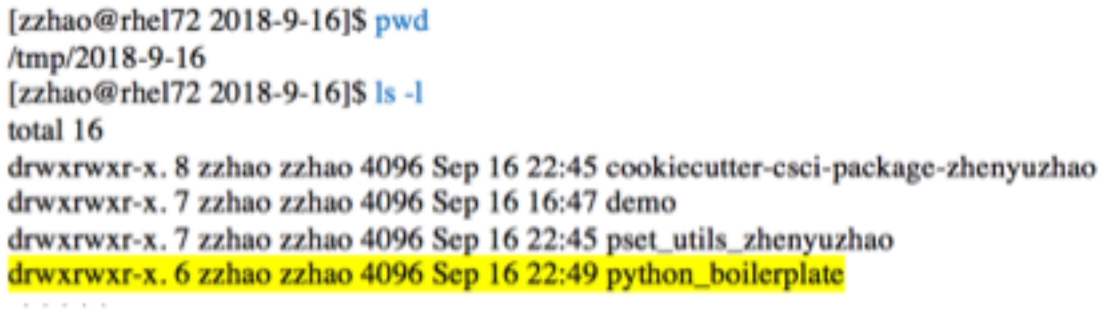
❑This is tricky step. So far we have worked on the template. Now we need to render a project based on the template:



```
[zzhao@rhel72 2018-9-16]$ cookiecutter cookiecutter-csci-package-zhenyuzhao/
full_name [Zhenyu Zhao]:
email [zhenyu_zhao@harvard.edu]:
github_username [csci-e-29]:
project_name [Python Boilerplate]:
project_slug [python_boilerplate]:
project_repo [python_boilerplate]:
project_short_description [Python Boilerplate contains all the boilerplate you need to create a
Python package.]:
pypi_username [csci-e-29]:
version [0.1.0]:
use_pytest [y]:
use_pypi_deployment_with_travis [n]:
add_pyup_badge [n]:
Select command_line_interface:
1 - No command-line interface
Choose from 1 [1]:
create_author_file [y]:
Select open_source_license:
1 - Not open source
Choose from 1 [1]:
```

Pset1 – Step 6 (continue)

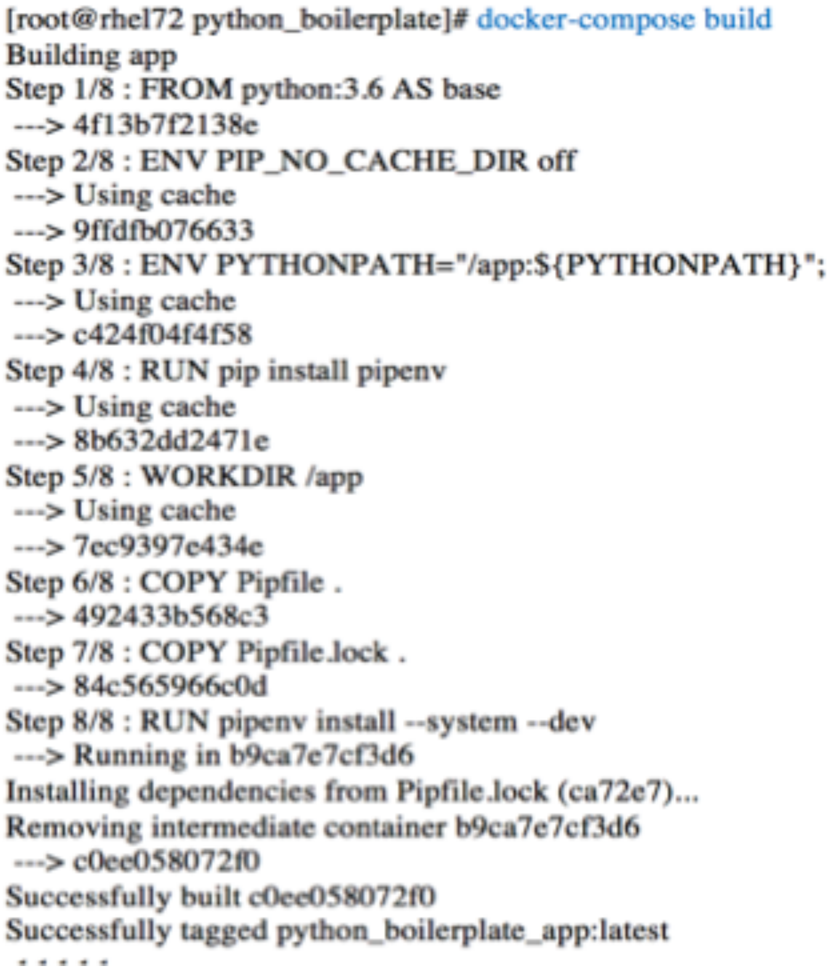
❑The result is that a new directory called python_boilerplate is created as follows:



```
[zzhao@rhel72 2018-9-16]$ pwd
/tmp/2018-9-16
[zzhao@rhel72 2018-9-16]$ ls -l
total 16
drwxrwxr-x. 8 zzhao zzhao 4096 Sep 16 22:45 cookiecutter-csci-package-zhenyuzhao
drwxrwxr-x. 7 zzhao zzhao 4096 Sep 16 16:47 demo
drwxrwxr-x. 7 zzhao zzhao 4096 Sep 16 22:45 pset_utils_zhenyuzhao
drwxrwxr-x. 6 zzhao zzhao 4096 Sep 16 22:49 python_boilerplate
. . . . .
```

Pset1 – Step 6 (continue)

❑Run the “docker-compose build” command:



```
[root@rhel72 python_boilerplate]# docker-compose build
Building app
Step 1/8 : FROM python:3.6 AS base
 ---> 4f13b7f2138e
Step 2/8 : ENV PIP_NO_CACHE_DIR off
 ---> Using cache
 ---> 9ffdfb076633
Step 3/8 : ENV PYTHONPATH="/app:${PYTHONPATH}";
 ---> Using cache
 ---> c424f04f4f58
Step 4/8 : RUN pip install pipenv
 ---> Using cache
 ---> 8b632dd2471e
Step 5/8 : WORKDIR /app
 ---> Using cache
 ---> 7ec9397e434e
Step 6/8 : COPY Pipfile .
 ---> 492433b568c3
Step 7/8 : COPY Pipfile.lock .
 ---> 84c565966c0d
Step 8/8 : RUN pipenv install --system --dev
 ---> Running in b9ca7e7cf3d6
Installing dependencies from Pipfile.lock (ca72e7)...
Removing intermediate container b9ca7e7cf3d6
 ---> c0ee058072f0
Successfully built c0ee058072f0
Successfully tagged python_boilerplate_app:latest
 . . . . .
```

Pset1 – Step 6 (continue)

❑Execute the “drun_app” comamnd

```
$ ./drun_app pipenv install pytest pytest-runner pytest-cov sphinx setuptools_scm --dev
```

Note that drun_app is a shortcut script that basically execute the command-line arguments and in this case pipenv install packages in dev environment.

❑What does pipenv do?

Pset1 – Step 6 (continue)

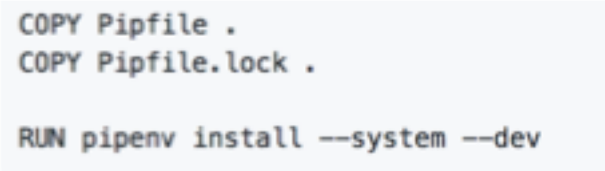
❑Copy Pipfile and Pipefile.lock back to the templace

```
$ mv Pipfile ../cookiecutter-csci-package-zhenyuzhao/\{\{cookiecutter.project_repo\}\}/
$ mv Pipfile.lock ../cookiecutter-csci-package-zhenyuzhai/\{\{cookiecutter.project_repo\}\}/
```

❑ Why copy Pipefile and Pipefile.lock?

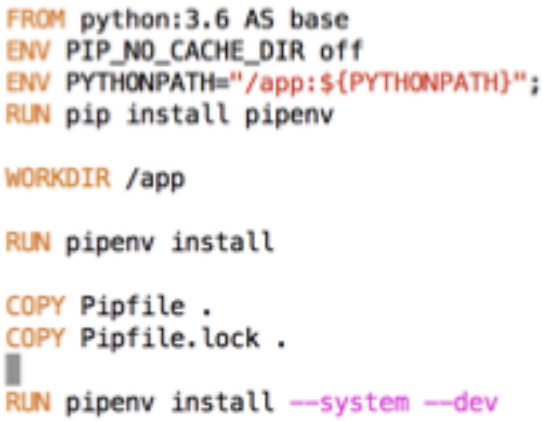
Pset1 – Step 6 (continue)

❑Add this to the end of the Dockerfile



```
COPY Pipfile .
COPY Pipfile.lock .

RUN pipenv install --system --dev
```

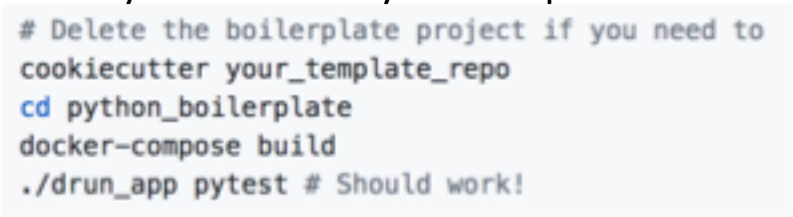
❑The final Dockerfile is as follows:



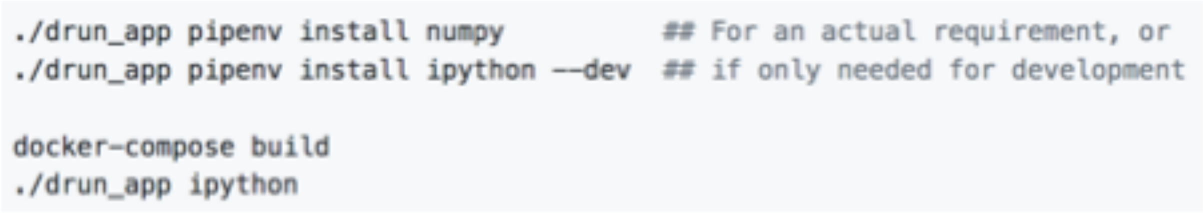
```
FROM python:3.6 AS base
ENV PIP_NO_CACHE_DIR off
ENV PYTHONPATH="/app:${PYTHONPATH}";
RUN pip install pipenv

WORKDIR /app

RUN pipenv install

COPY Pipfile .
COPY Pipfile.lock .

RUN pipenv install --system --dev
```

Pset1 – Step 6 (continue)

❑Finally test and verify the template



```
# Delete the boilerplate project if you need to
cookiecutter your_template_repo
cd python_boilerplate
docker-compose build
./drun_app pytest # Should work!
```

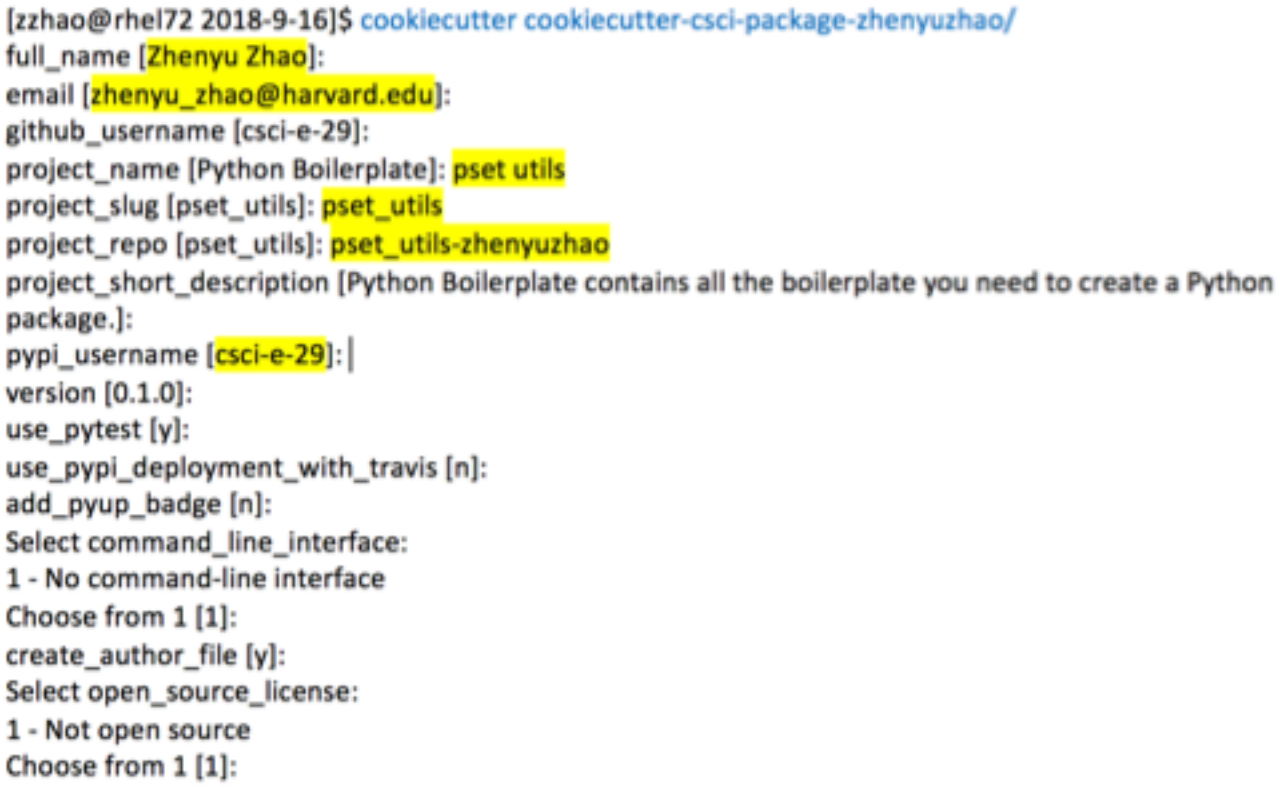


```
./drun_app pipenv install numpy          ## For an actual requirement, or
./drun_app pipenv install ipython --dev  ## if only needed for development

docker-compose build
./drun_app ipython
```

Pset1 – Step 7

- **Initialize your pset_utils repo**

□ Create a new project using cookiecutter with your template



```
[zzhao@rhel72 2018-9-16]$ cookiecutter cookiecutter-csci-package-zhenyuzhao/
full_name [Zhenyu Zhao]:
email [zhenyu_zhao@harvard.edu]:
github_username [csci-e-29]:
project_name [Python Boilerplate]: pset utils
project_slug [pset_utils]: pset_utils
project_repo [pset_utils]: pset_utils-zhenyuzhao
project_short_description [Python Boilerplate contains all the boilerplate you need to create a Python
package.]:
pypi_username [csci-e-29]:
version [0.1.0]:
use_pytest [y]:
use_pypi_deployment_with_travis [n]:
add_pyup_badge [n]:
Select command_line_interface:
1 - No command-line interface
Choose from 1 [1]:
create_author_file [y]:
Select open_source_license:
1 - Not open source
Choose from 1 [1]:
```

Pset1 – Step 7 (continued)

❑Turn the directory into a repo and link to your classroom repo

```
$ git init
$ git remote add origin git@github.com:csci-e-29/pset_utils-<username>.git
$ git fetch
$ git merge origin/master
```

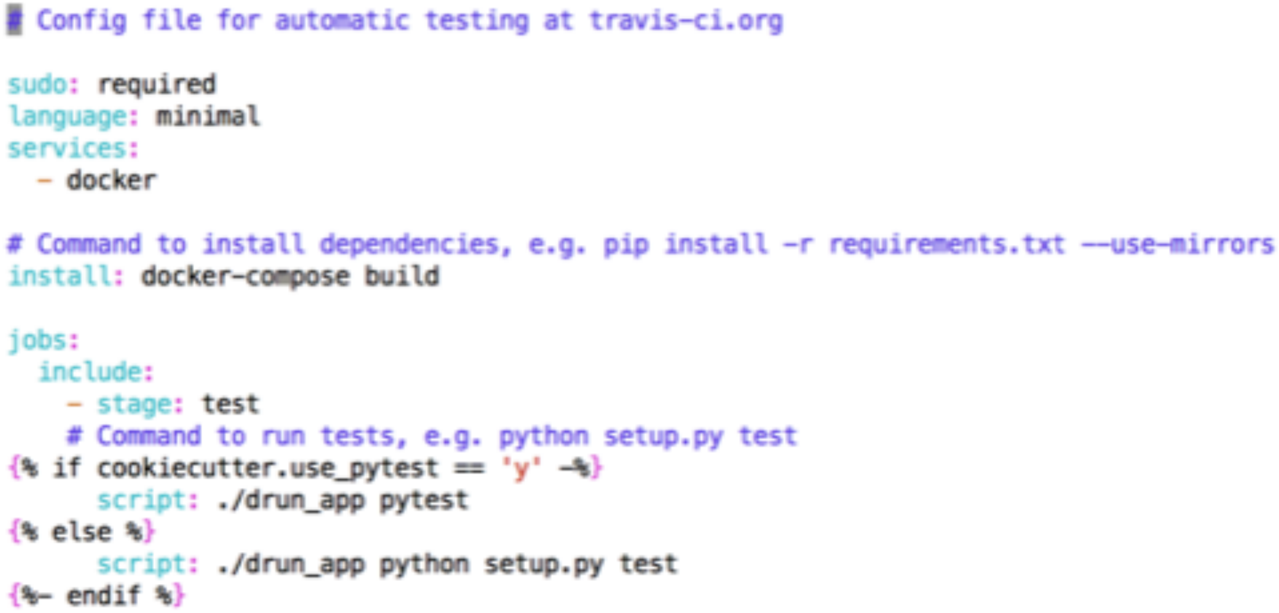
❑Remember to manage existing the non-master branches properly.

```
$git branch -d <branchname>
$git push origin --delete <branchname>
$ git branch -a
$ git branch -d <branchname>
$ git push origin --delete <branchname>
$ git branch -a develop
```

Pset1 – Step 8

- **Switch Travis to docker**

❑Update .travis.yml to look something like:



```
# Config file for automatic testing at travis-ci.org

sudo: required
language: minimal
services:
  - docker

# Command to install dependencies, e.g. pip install -r requirements.txt --use-mirrors
install: docker-compose build

jobs:
  include:
    - stage: test
    # Command to run tests, e.g. python setup.py test
{% if cookiecutter.use_pytest == 'y' -%}
      script: ./drun_app pytest
{% else %}
      script: ./drun_app python setup.py test
{%- endif %}
```

Pset1 – Step 9

➢ **Commit the rendered project files now and push to GitHub**

❑Because of the .travis.yml file, TravisCI should queue up a build.

Pset1 – Step 10

➢ **Implement atomic writer**

Atomic writes are used to ensure we never have an incomplete file target. Basically, they perform the operations:

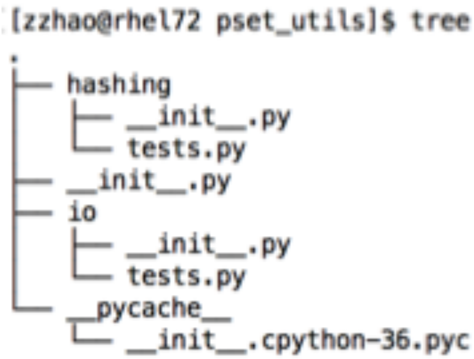
❑Create a temporary file which is unique (possibly involving a random file name)

❑Allow the code to take its sweet time writing to the file

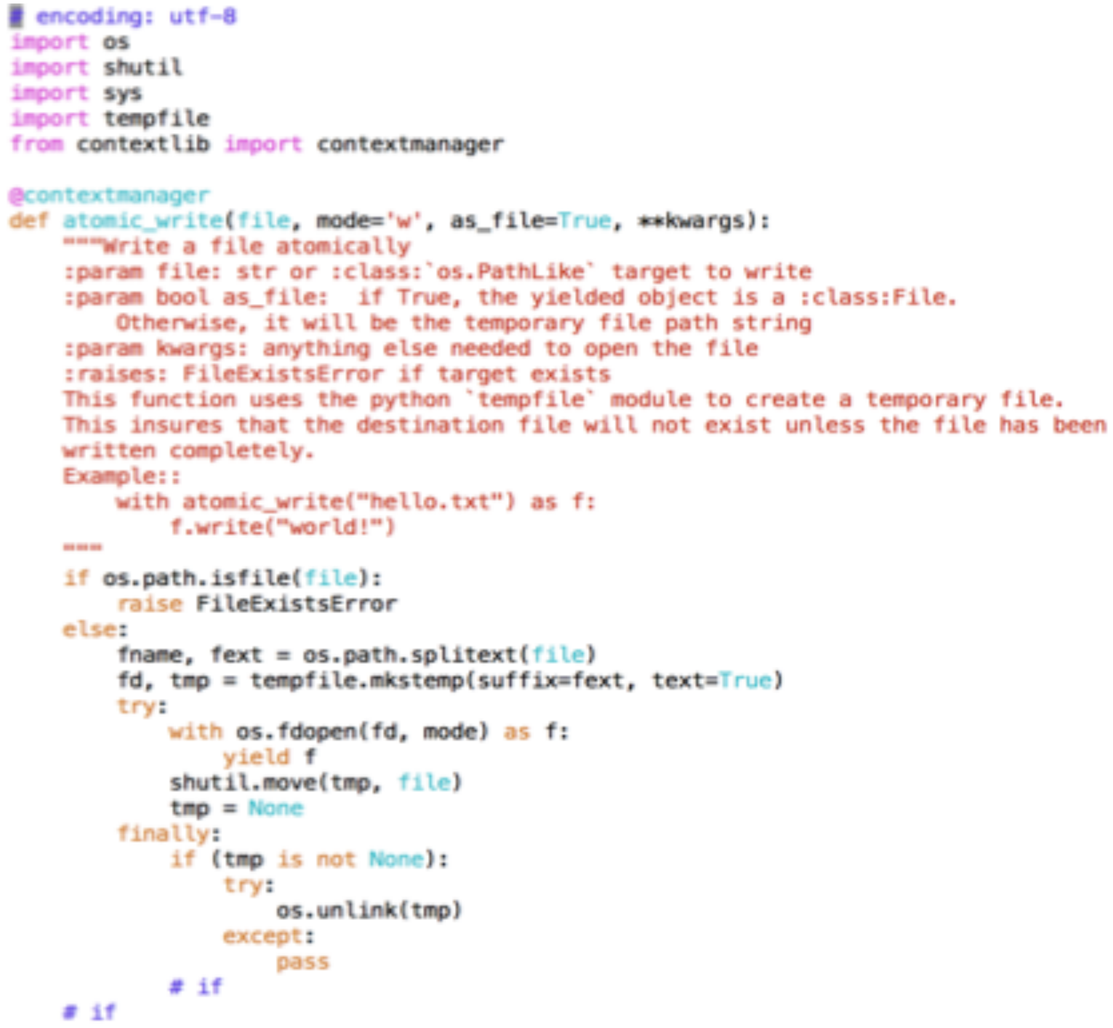
❑Rename the file to the target destination name.

If the target and temporary file are on the same filesystem, the rename operation is ***atomic*** - that is, it can only completely succeed or fail entirely, and you can never be left with a bad file state (assuming the code writes the data you wanted without failing).

Pset1 – Step 10 (continued)



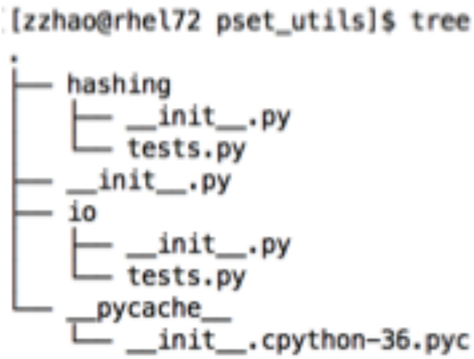
```
[zzhao@rhel72 pset_utils]$ tree
.
├── hashing
│   ├── __init__.py
│   └── tests.py
├── __init__.py
├── io
│   ├── __init__.py
│   └── tests.py
└── __pycache__
    └── __init__.cpython-36.pyc
```



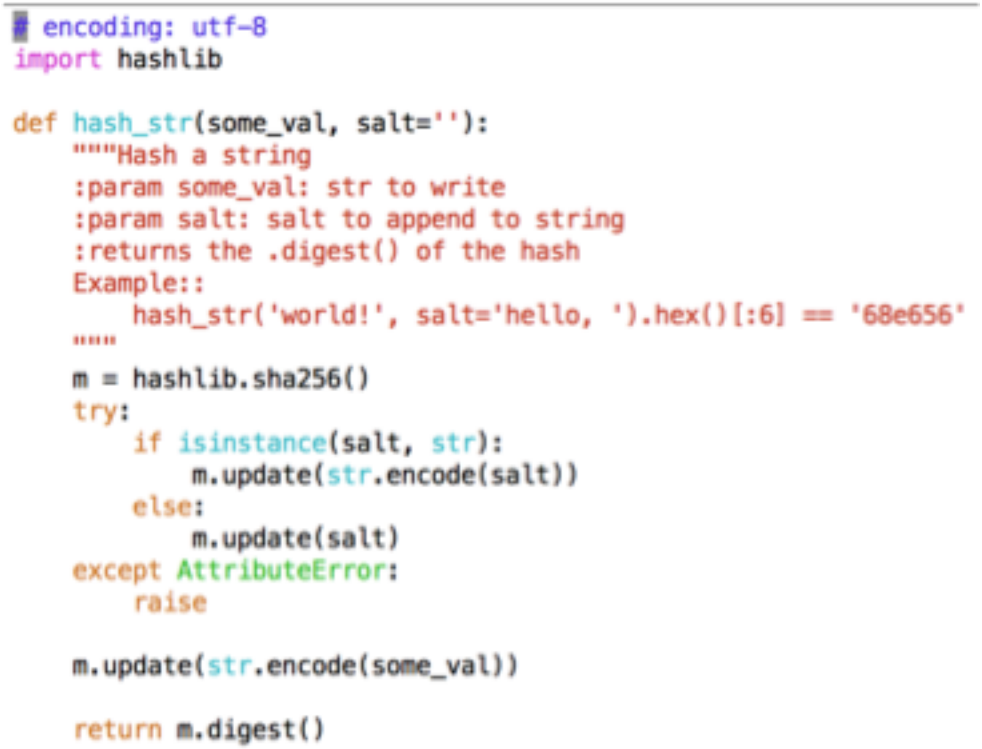
```python
# encoding: utf-8
import os
import shutil
import sys
import tempfile
from contextlib import contextmanager

@contextmanager
def atomic_write(file, mode='w', as_file=True, **kwargs):
    """Write a file atomically
    :param file: str or :class:`os.PathLike` target to write
    :param bool as_file:  if True, the yielded object is a :class:File.
        Otherwise, it will be the temporary file path string
    :param kwargs: anything else needed to open the file
    :raises: FileExistsError if target exists
    This function uses the python `tempfile` module to create a temporary file.
    This insures that the destination file will not exist unless the file has been
    written completely.
    Example::
        with atomic_write("hello.txt") as f:
            f.write("world!")
    """
    if os.path.isfile(file):
        raise FileExistsError
    else:
        fname, fext = os.path.splitext(file)
        fd, tmp = tempfile.mkstemp(suffix=fext, text=True)
        try:
            with os.fdopen(fd, mode) as f:
                yield f
            shutil.move(tmp, file)
            tmp = None
        finally:
            if (tmp is not None):
                try:
                    os.unlink(tmp)
                except:
                    pass
            # if
    # if
```

Pset1 – Step 11

- **Implement a standardized string hash**



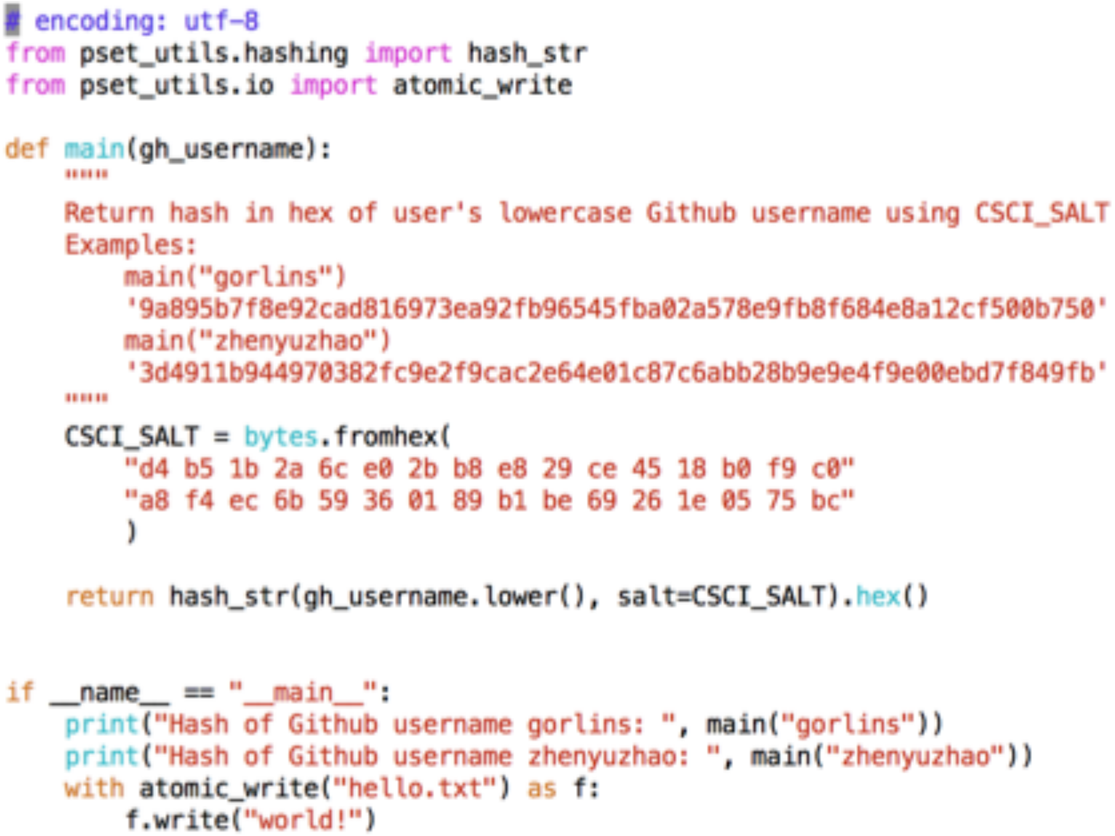
```
[zzhao@rhel72 pset_utils]$ tree
.
├── hashing
│   ├── __init__.py
│   └── tests.py
├── __init__.py
├── io
│   ├── __init__.py
│   └── tests.py
└── __pycache__
    └── __init__.cpython-36.pyc
```



```
# encoding: utf-8
import hashlib

def hash_str(some_val, salt=''):
    """Hash a string
    :param some_val: str to write
    :param salt: salt to append to string
    :returns the .digest() of the hash
    Example::
        hash_str('world!', salt='hello, ').hex()[:6] == '68e656'
    """
    m = hashlib.sha256()
    try:
        if isinstance(salt, str):
            m.update(str.encode(salt))
        else:
            m.update(salt)
    except AttributeError:
        raise

    m.update(str.encode(some_val))

    return m.digest()
```

Pset1 – Step 11 (continue)

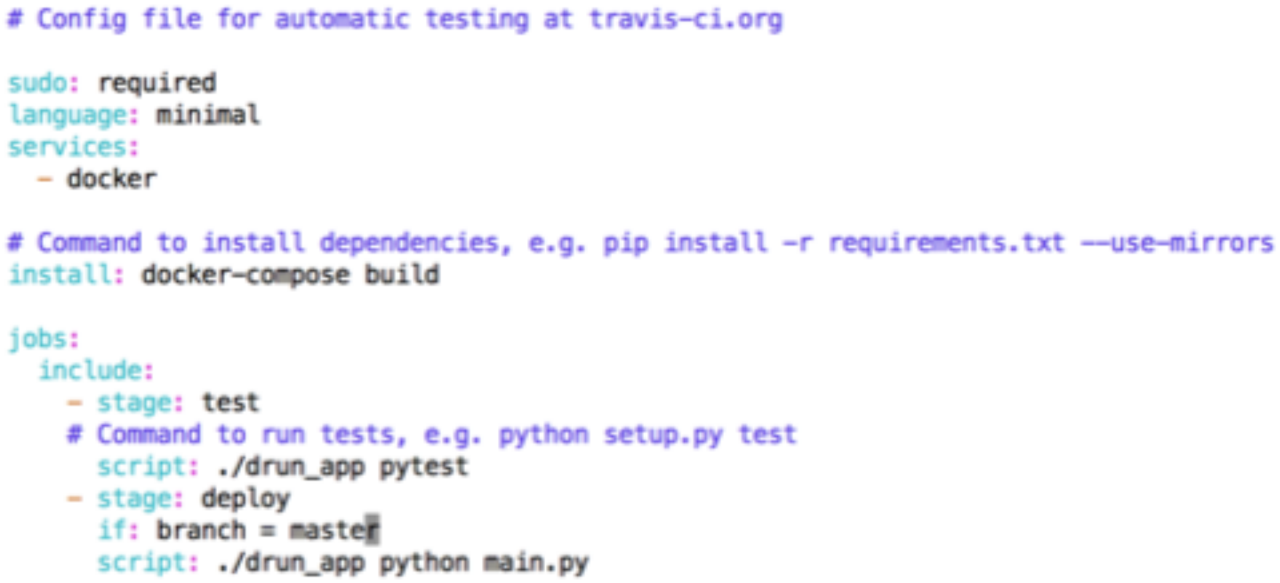
- ➢ **Prove your work**
- ❑Create a main.py



```python
# encoding: utf-8
from pset_utils.hashing import hash_str
from pset_utils.io import atomic_write

def main(gh_username):
    """
    Return hash in hex of user's lowercase Github username using CSCI_SALT
    Examples:
        main("gorlins")
        '9a895b7f8e92cad816973ea92fb96545fba02a578e9fb8f684e8a12cf500b750'
        main("zhenyuzhao")
        '3d4911b944970382fc9e2f9cac2e64e01c87c6abb28b9e9e4f9e00ebd7f849fb'
    """
    CSCI_SALT = bytes.fromhex(
        "d4 b5 1b 2a 6c e0 2b b8 e8 29 ce 45 18 b0 f9 c0"
        "a8 f4 ec 6b 59 36 01 89 b1 be 69 26 1e 05 75 bc"
        )

    return hash_str(gh_username.lower(), salt=CSCI_SALT).hex()


if __name__ == "__main__":
    print("Hash of Github username gorlins: ", main("gorlins"))
    print("Hash of Github username zhenyuzhao: ", main("zhenyuzhao"))
    with atomic_write("hello.txt") as f:
        f.write("world!")
```

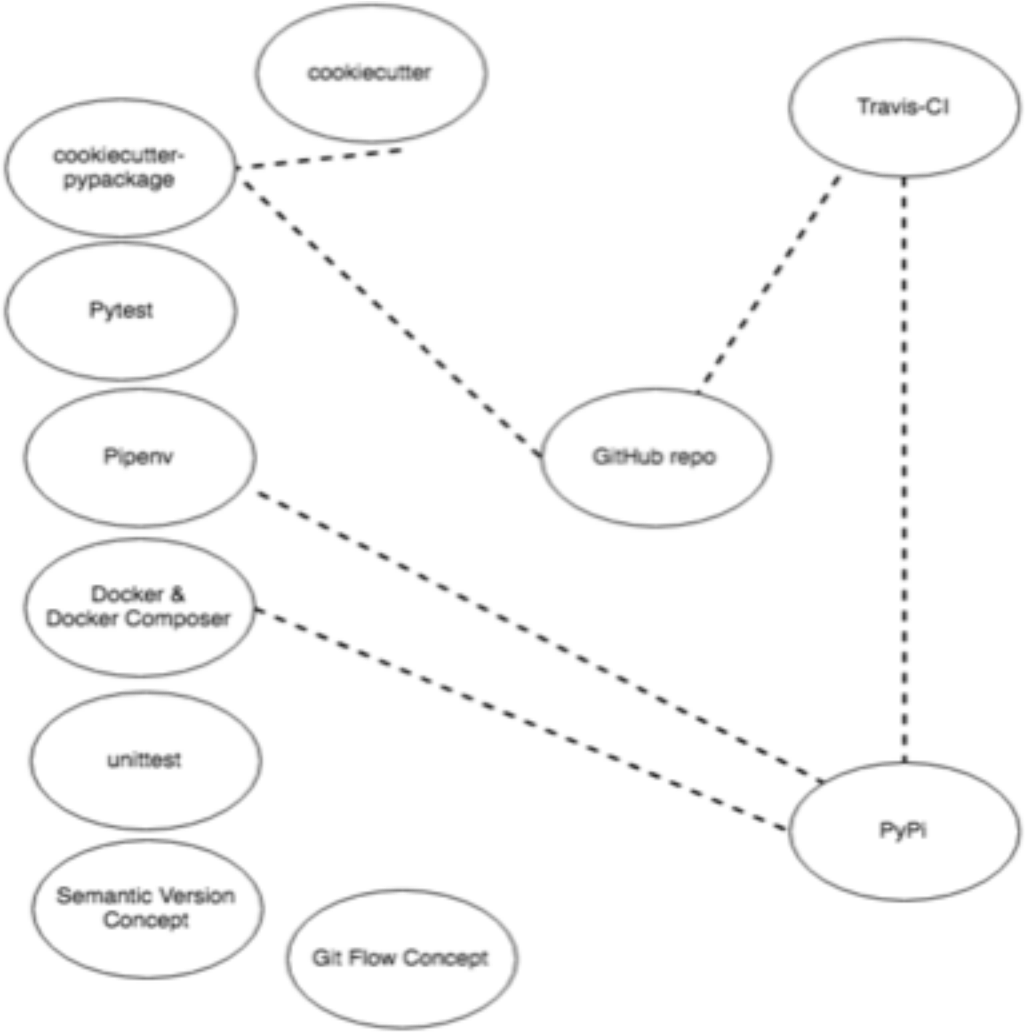
Pset1 – Step 12 (continue)

❑Add the following stage to your .travis.yml



```
# Config file for automatic testing at travis-ci.org

sudo: required
language: minimal
services:
  - docker

# Command to install dependencies, e.g. pip install -r requirements.txt --use-mirrors
install: docker-compose build

jobs:
  include:
    - stage: test
    # Command to run tests, e.g. python setup.py test
      script: ./drun_app pytest
    - stage: deploy
      if: branch = master
      script: ./drun_app python main.py
```

Pset1 – Conclusion



Word Embedding Introduction

- What is word embedding?

Word embedding is the collective name for a set of language modeling and feature learning techniques in natural language processing (NLP) where words or phrases from the vocabulary are mapped to vectors of real numbers.

- See an example

http://embeddings.macheads101.com/

Word Embedding Introduction

- Context

I laugh at his joke.

His joke didn’t make me laugh.

Word Embedding Introduction

- ➢ What does word embedding actually do?

It just converts a word into a vector.

Red ➔

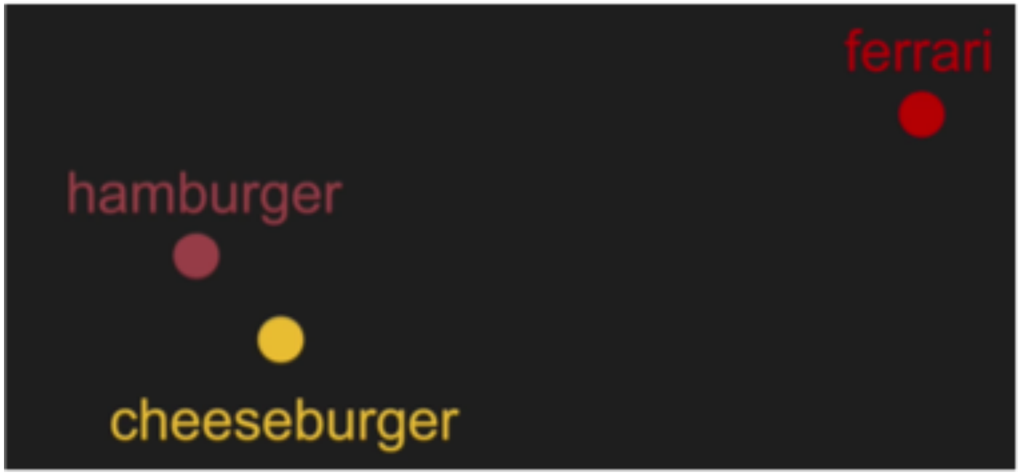
Yellow ➔

A vector is a mathematical object that has magnitude and direction and that is commonly represented by a directed line segment whose length represents the magnitude and whose orientation in space represents the direction.

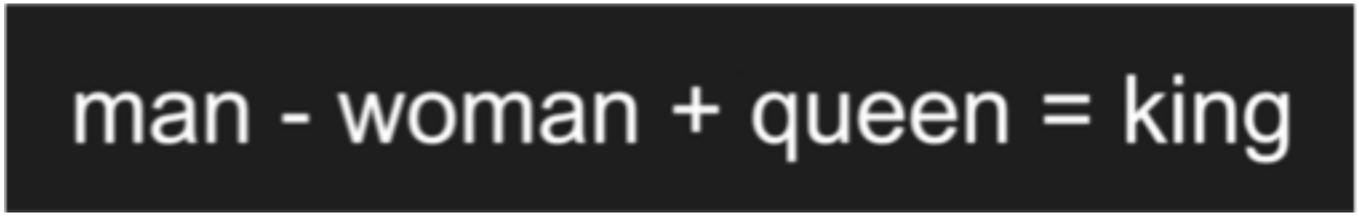
Word Embedding Introduction

➢ Benefits of quantifying words

❑Comparison



❑Calculation


man - woman + queen = king

Word Embedding Introduction

➢ We want to encapsulate as much information as possible to a vector

❑1,024-bit vector

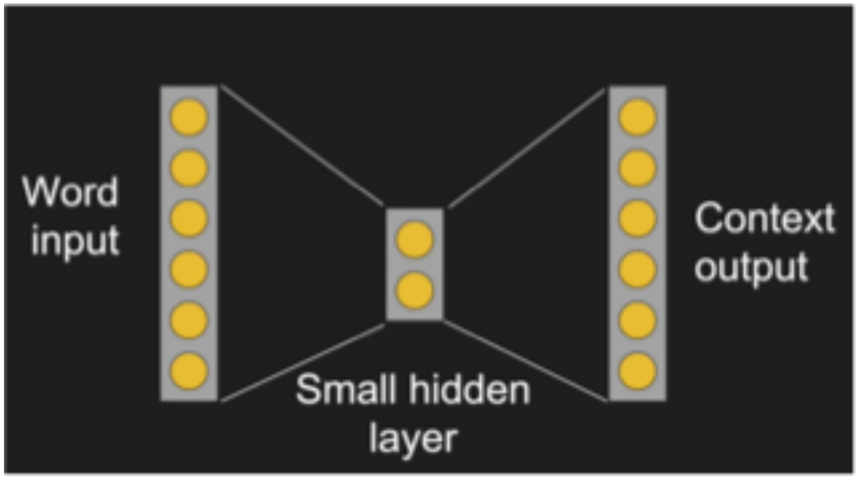
❑10,240-bit vector

❑102400-bit vector

❑Each bit represents a property or value related to context or

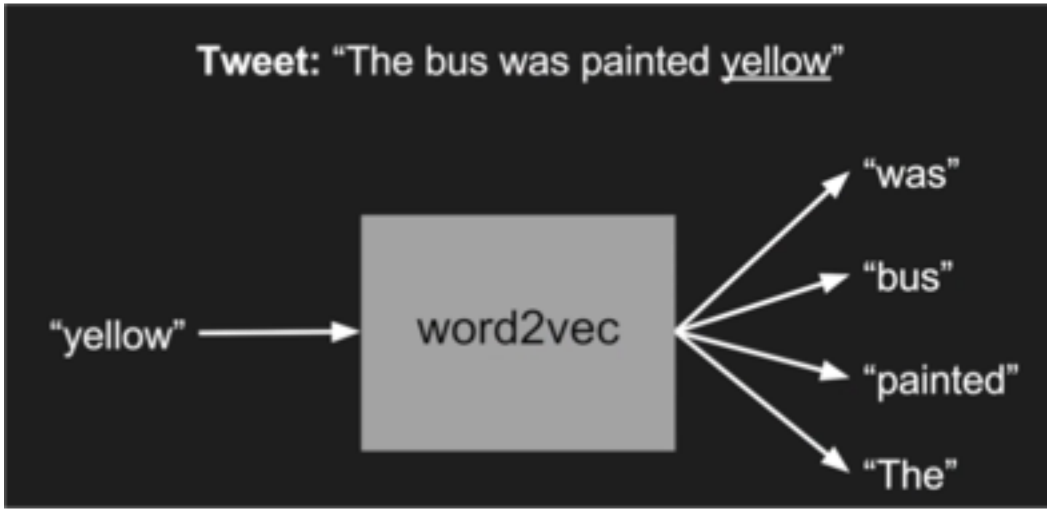
Word Embedding Introduction

- Word2vec



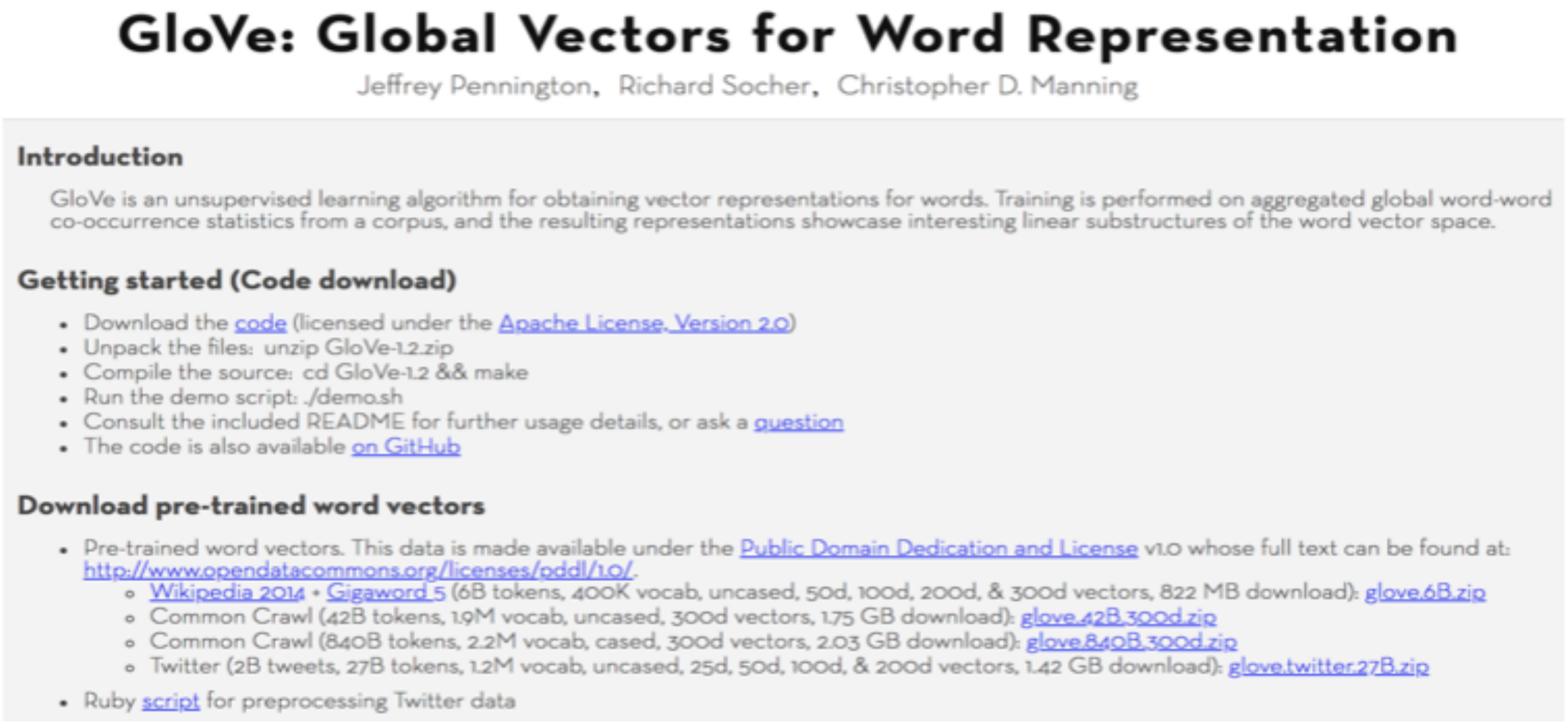
Word Embedding Introduction

- How does word2vec work?



Word Embedding Introduction

- Good algorithm: GloVe



GloVe: Global Vectors for Word Representation

Jeffrey Pennington, Richard Socher, Christopher D. Manning

Introduction

GloVe is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space.

Getting started (Code download)

- Download the code (licensed under the Apache License, Version 2.0)
- Unpack the files: unzip GloVe-1.2.zip
- Compile the source: cd GloVe-1.2 && make
- Run the demo script: ./demo.sh
- Consult the included README for further usage details, or ask a question
- The code is also available on GitHub

Download pre-trained word vectors

- Pre-trained word vectors. This data is made available under the Public Domain Dedication and License v1.0 whose full text can be found at: http://www.opendatacommons.org/licenses/pddl/1.0/.
    - Wikipedia 2014 + Gigaword 5 (6B tokens, 400K vocab, uncased, 50d, 100d, 200d, & 300d vectors, 822 MB download): glove.6B.zip
    - Common Crawl (42B tokens, 1.9M vocab, uncased, 300d vectors, 1.75 GB download): glove.42B.300d.zip
    - Common Crawl (840B tokens, 2.2M vocab, cased, 300d vectors, 2.03 GB download): glove.840B.300d.zip
    - Twitter (2B tweets, 27B tokens, 1.2M vocab, uncased, 25d, 50d, 100d, & 200d vectors, 1.42 GB download): glove.twitter.27B.zip
- Ruby script for preprocessing Twitter data