Thoth

How to find the best application stack

Fridolín Pokorný <fridolin@redhat.com>

Thoth

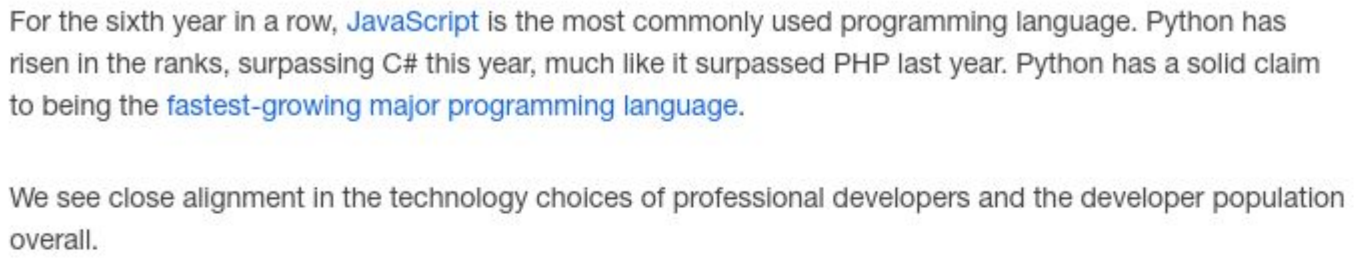
~~How to find the best application stack~~

How to give the right DNA to our applications

Fridolín Pokorný <fridolin@redhat.com>

Let’s develop a machine learning application

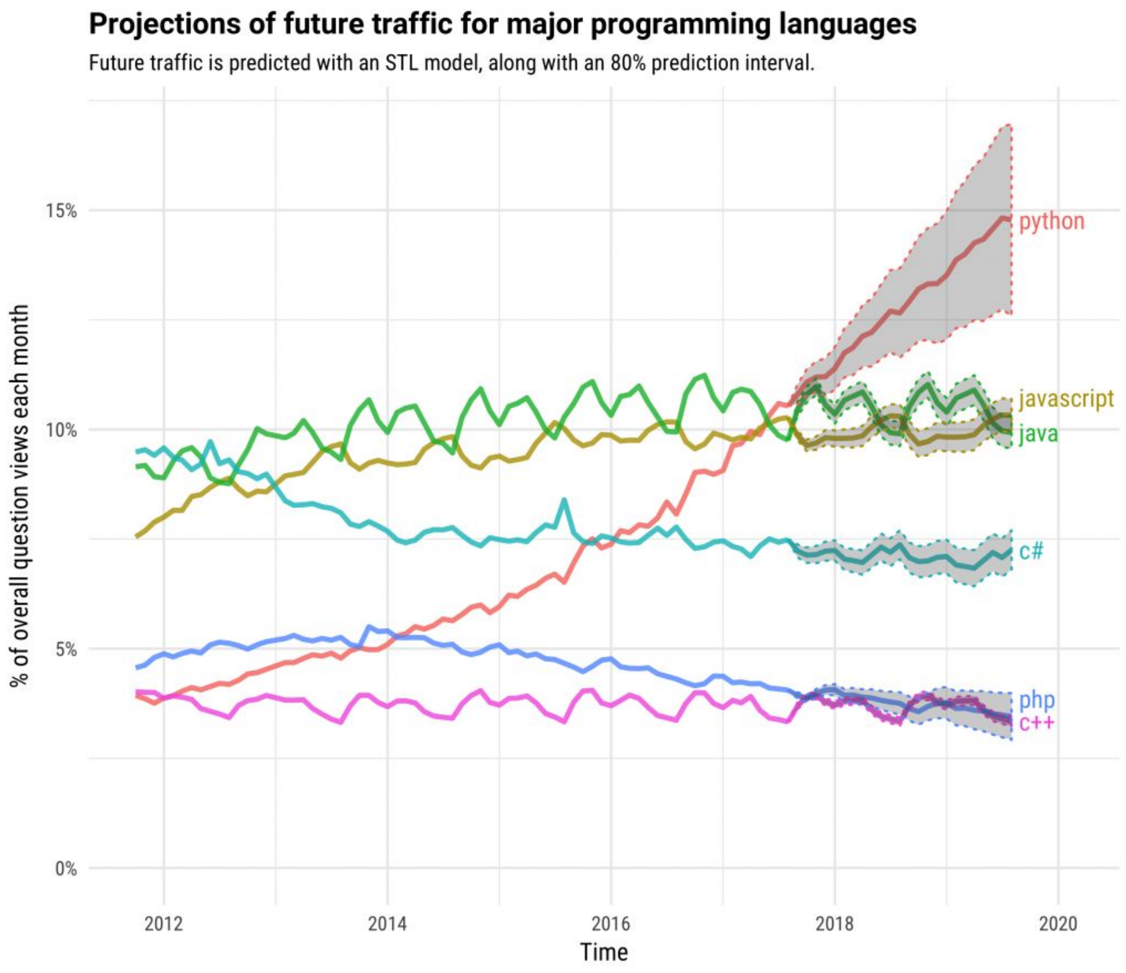
What programming language should I use?



For the sixth year in a row, JavaScript is the most commonly used programming language. Python has risen in the ranks, surpassing C# this year, much like it surpassed PHP last year. Python has a solid claim to being the fastest-growing major programming language.

We see close alignment in the technology choices of professional developers and the developer population overall.

Source: Developer Survey Results 2018



Source: The Incredible Growth of Python

python™

TensorFlow

Flask

pandas

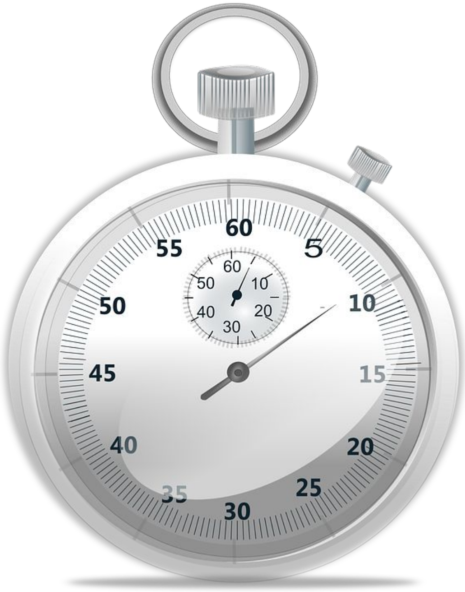
$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$

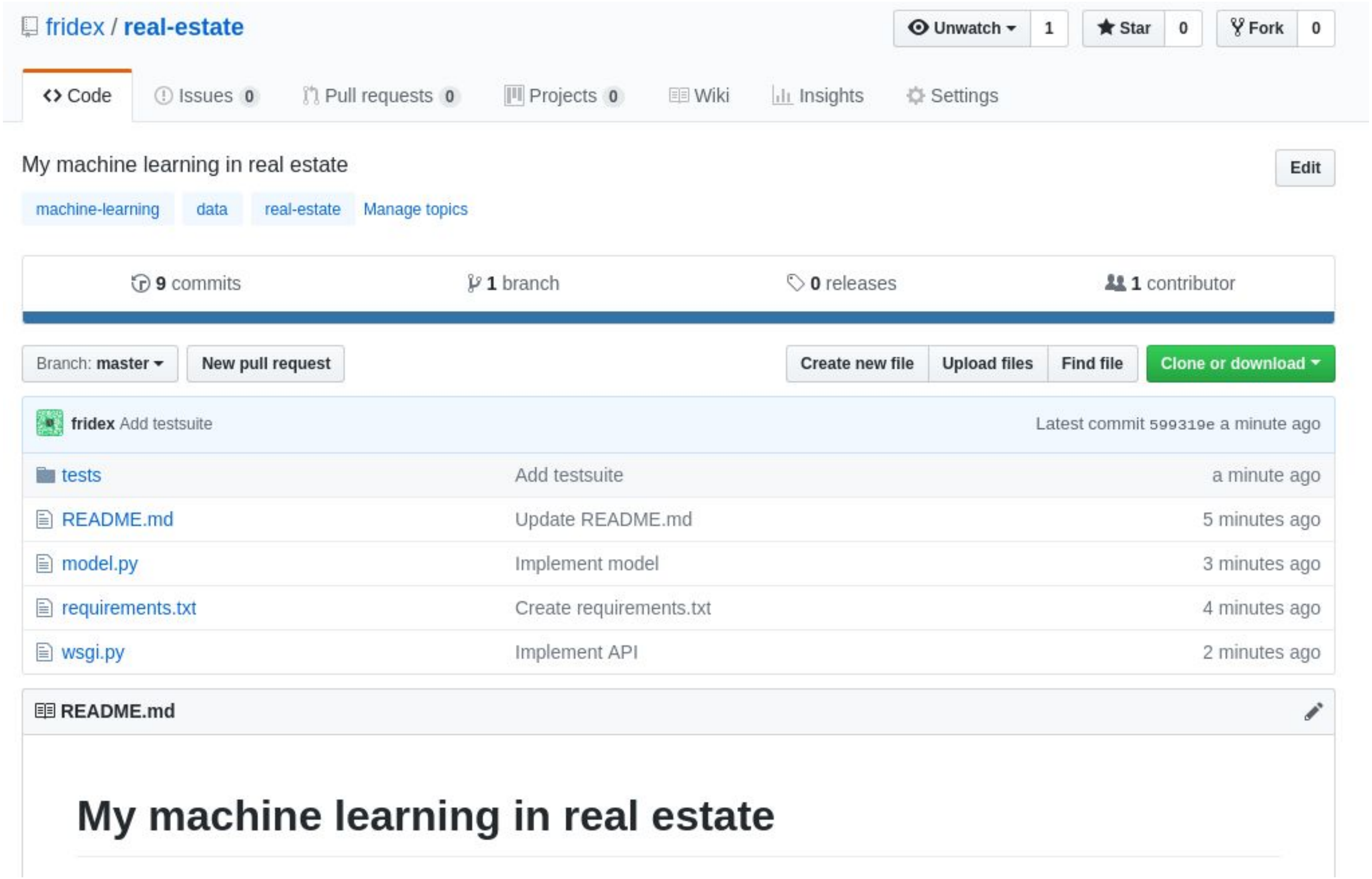
gunicorn

Requirements of my application

- requirements.txt file

```
tensorflow
pandas
flask
gunicorn
```




fridex / real-estate

Unwatch 1 | Star 0 | Fork 0

Code | Issues 0 | Pull requests 0 | Projects 0 | Wiki | Insights | Settings

My machine learning in real estate

machine-learning | data | real-estate | Manage topics

9 commits | 1 branch | 0 releases | 1 contributor

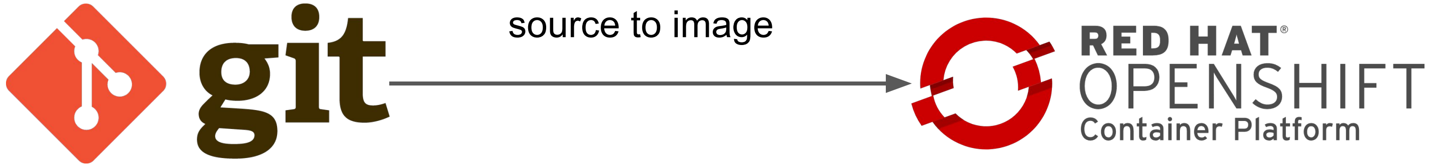
Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

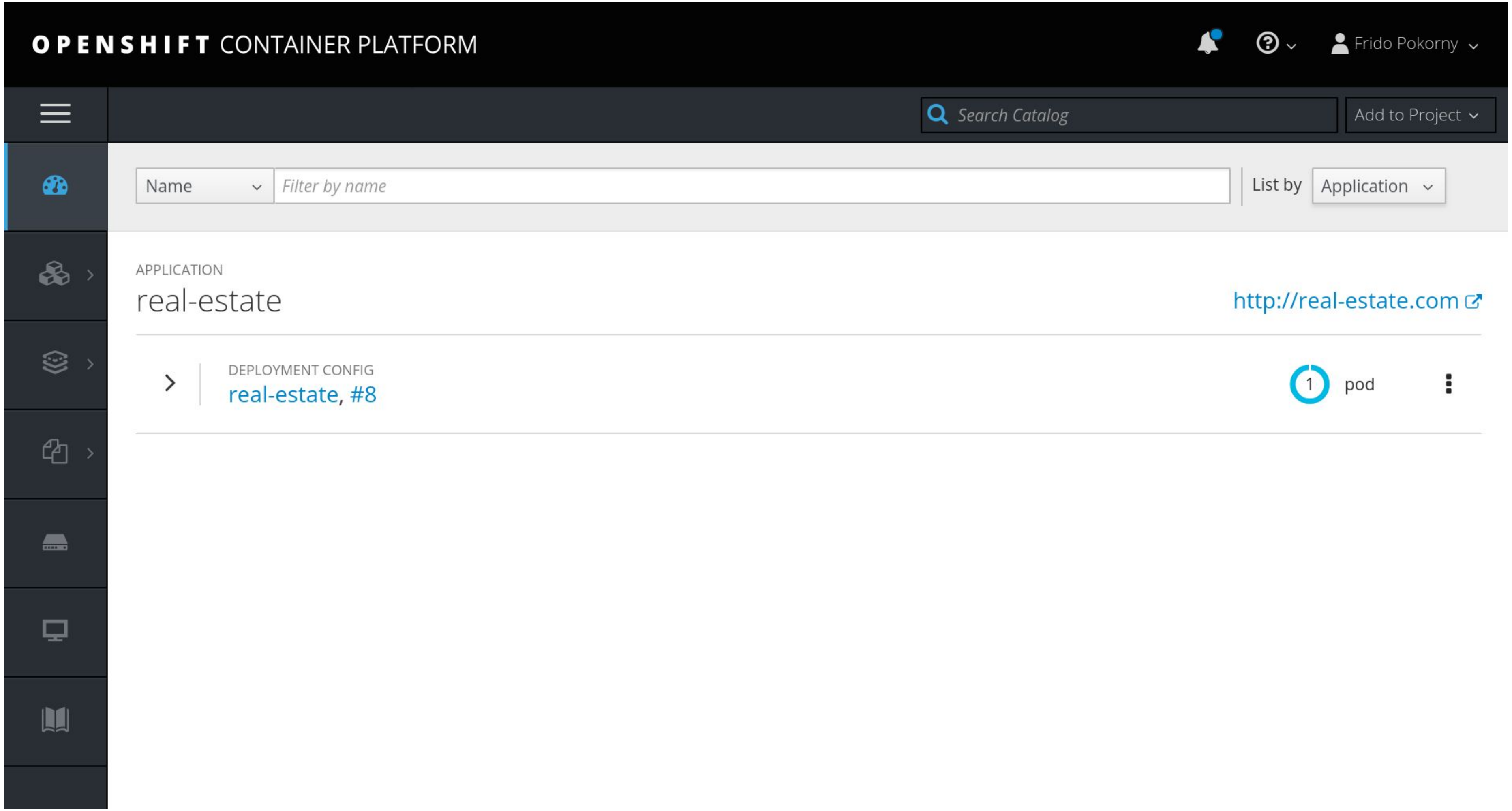
fridex Add testsuite — Latest commit 599319e a minute ago

tests	Add testsuite	a minute ago
README.md	Update README.md	5 minutes ago
model.py	Implement model	3 minutes ago
requirements.txt	Create requirements.txt	4 minutes ago
wsgi.py	Implement API	2 minutes ago

README.md

My machine learning in real estate


git
source to image
RED HAT®
OPENSHIFT
Container Platform



APPLICATION

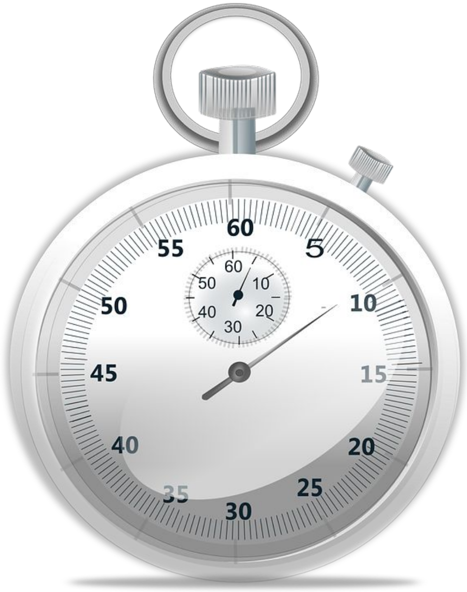
real-estate

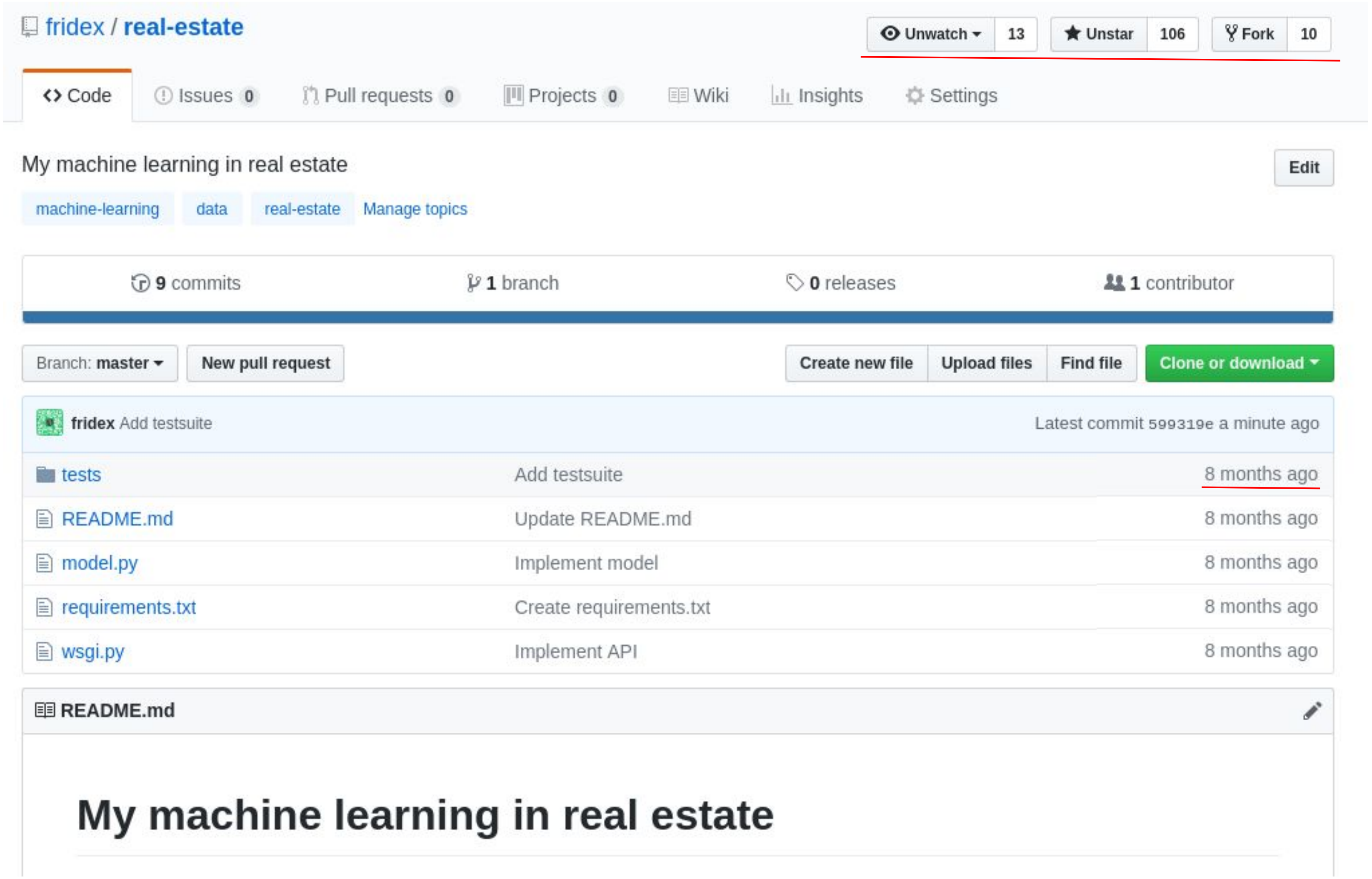
http://real-estate.com

DEPLOYMENT CONFIG

real-estate, #8

1 pod




fridex / real-estate

Unwatch 13 | Unstar 106 | Fork 10

Code | Issues 0 | Pull requests 0 | Projects 0 | Wiki | Insights | Settings

My machine learning in real estate

Edit

machine-learning | data | real-estate | Manage topics

9 commits | 1 branch | 0 releases | 1 contributor

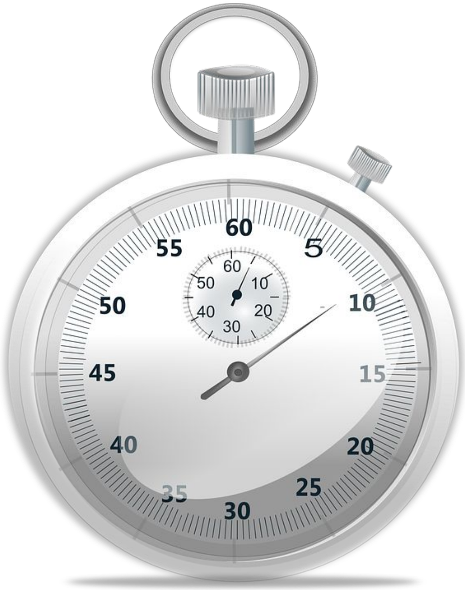
Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

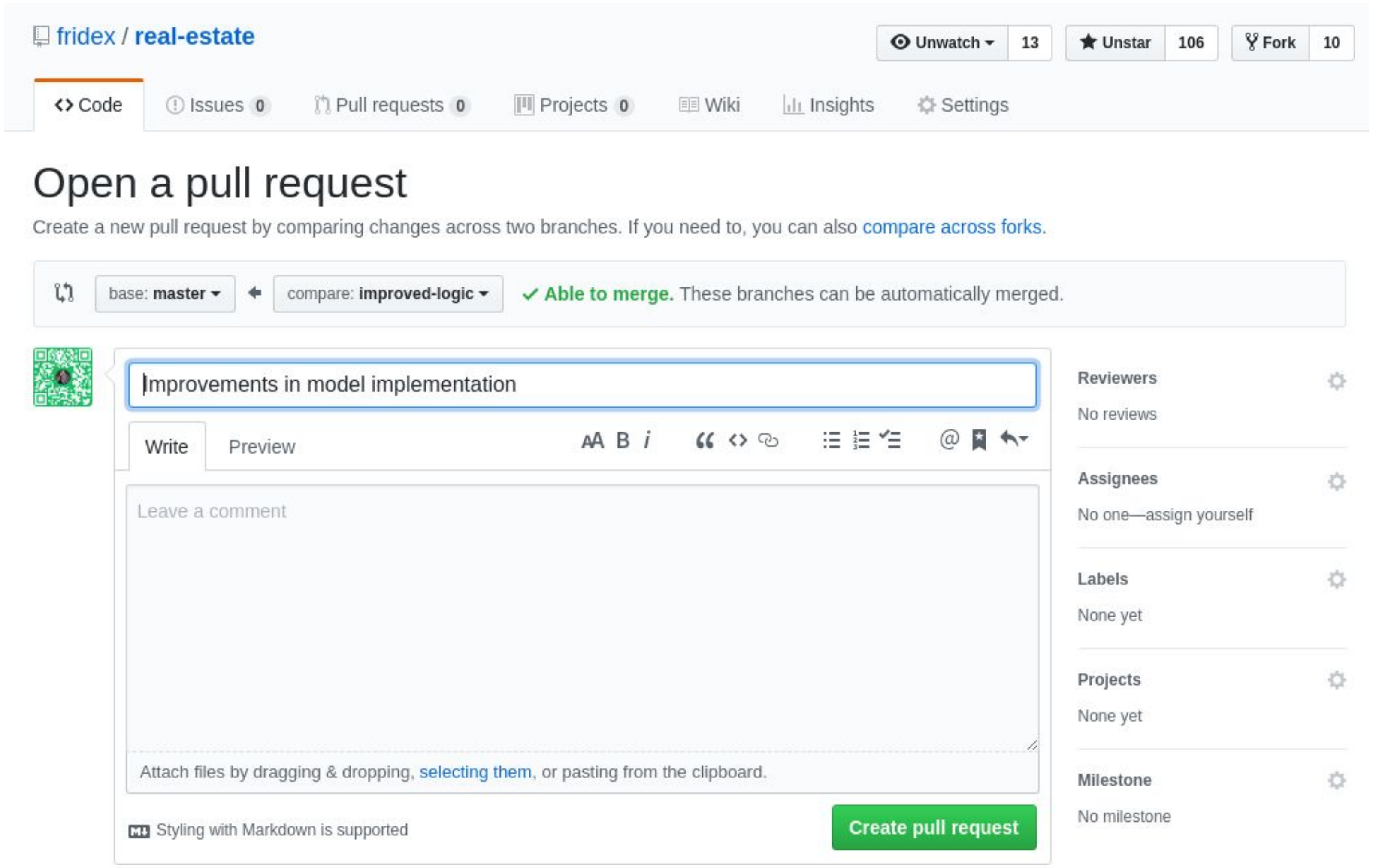
fridex Add testsuite — Latest commit 599319e a minute ago

tests	Add testsuite	8 months ago
README.md	Update README.md	8 months ago
model.py	Implement model	8 months ago
requirements.txt	Create requirements.txt	8 months ago
wsgi.py	Implement API	8 months ago

README.md

My machine learning in real estate


60
5
10
15
20
25
30
35
40
45
50
55
60
10
20
30
40
50


fridex / real-estate

Unwatch 13 | Unstar 106 | Fork 10

Code | Issues 0 | Pull requests 0 | Projects 0 | Wiki | Insights | Settings

Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also compare across forks.

base: **master** ← compare: **improved-logic** ✓ **Able to merge.** These branches can be automatically merged.

Improvements in model implementation

Write | Preview

Leave a comment

Attach files by dragging & dropping, selecting them, or pasting from the clipboard.

Styling with Markdown is supported

Create pull request

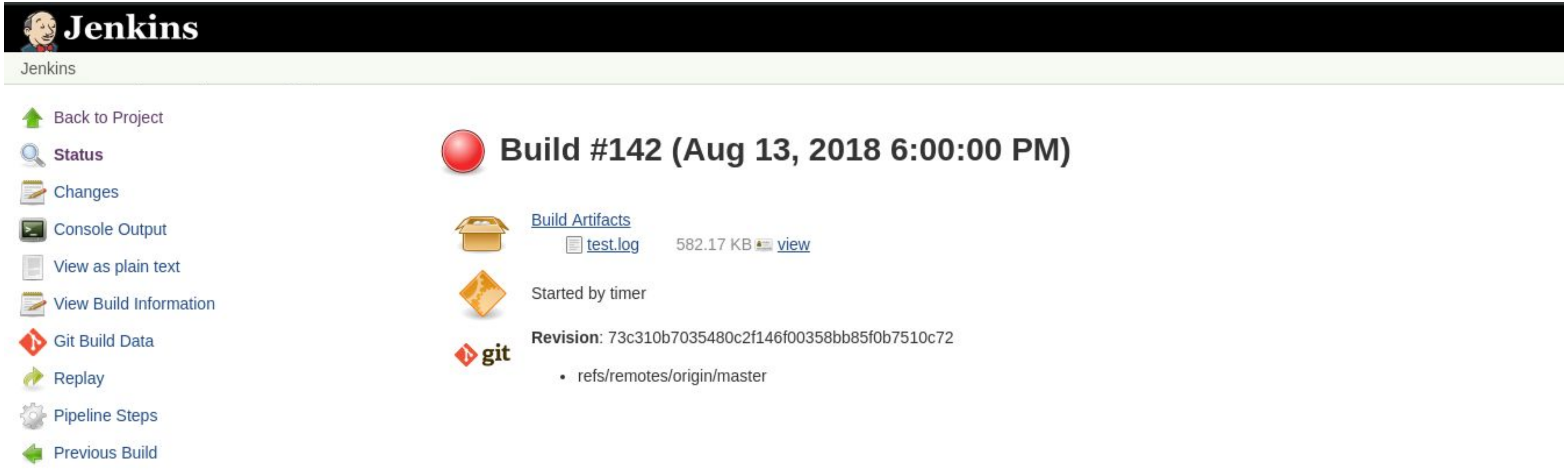
Reviewers
No reviews

Assignees
No one—assign yourself

Labels
None yet

Projects
None yet

Milestone
No milestone



Jenkins

Jenkins

- Back to Project
- Status
- Changes
- Console Output
- View as plain text
- View Build Information
- Git Build Data
- Replay
- Pipeline Steps
- Previous Build

Build #142 (Aug 13, 2018 6:00:00 PM)

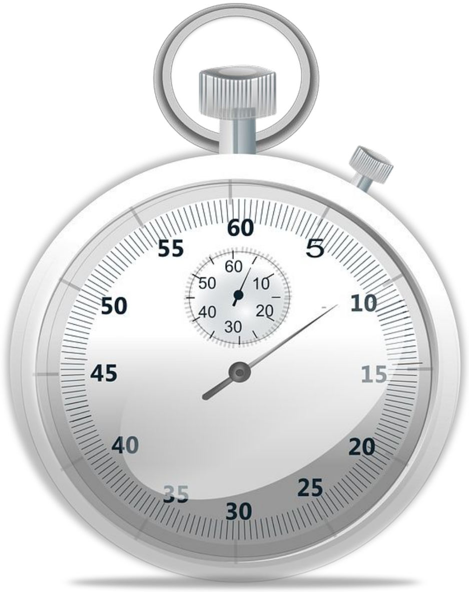
Build Artifacts
test.log 582.17 KB view

Started by timer

Revision: 73c310b7035480c2f146f00358bb85f0b7510c72

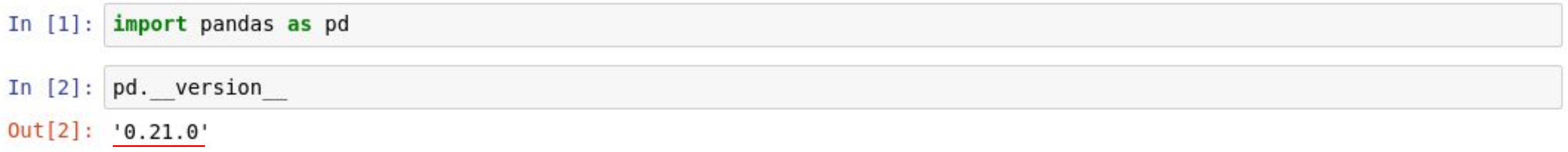
- refs/remotes/origin/master

Let’s debug application


60
5
10
15
20
25
30
35
40
45
50
55

Behavior change in library

Previous behavior



```
In [1]: import pandas as pd

In [2]: pd.__version__

Out[2]: '0.21.0'
```

In [1]:
```
import pandas as pd
```

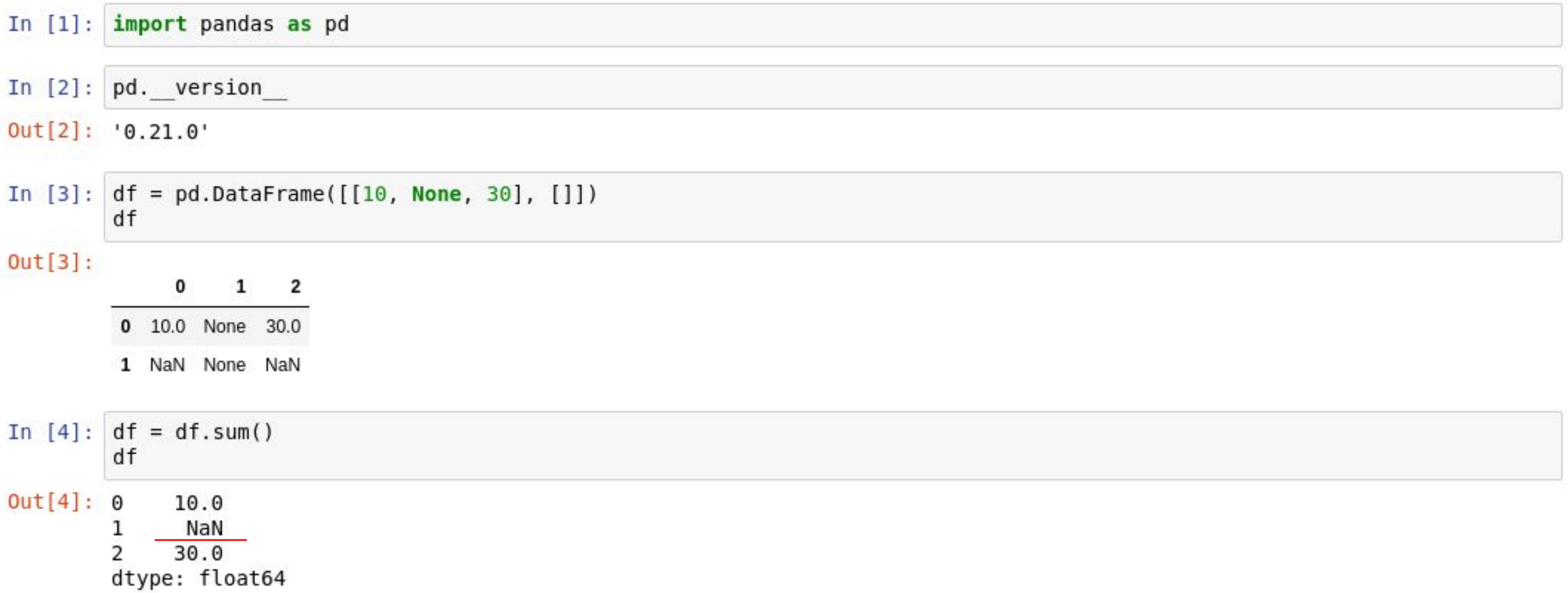
In [2]:
```
pd.__version__
```

Out[2]:
```
'0.21.0'
```

In [3]:
```
df = pd.DataFrame([[10, None, 30], []])
df
```

Out[3]:

	0	1	2
0	10.0	None	30.0
1	NaN	None	NaN



```
In [1]: import pandas as pd
```

```
In [2]: pd.__version__
```

```
Out[2]: '0.21.0'
```

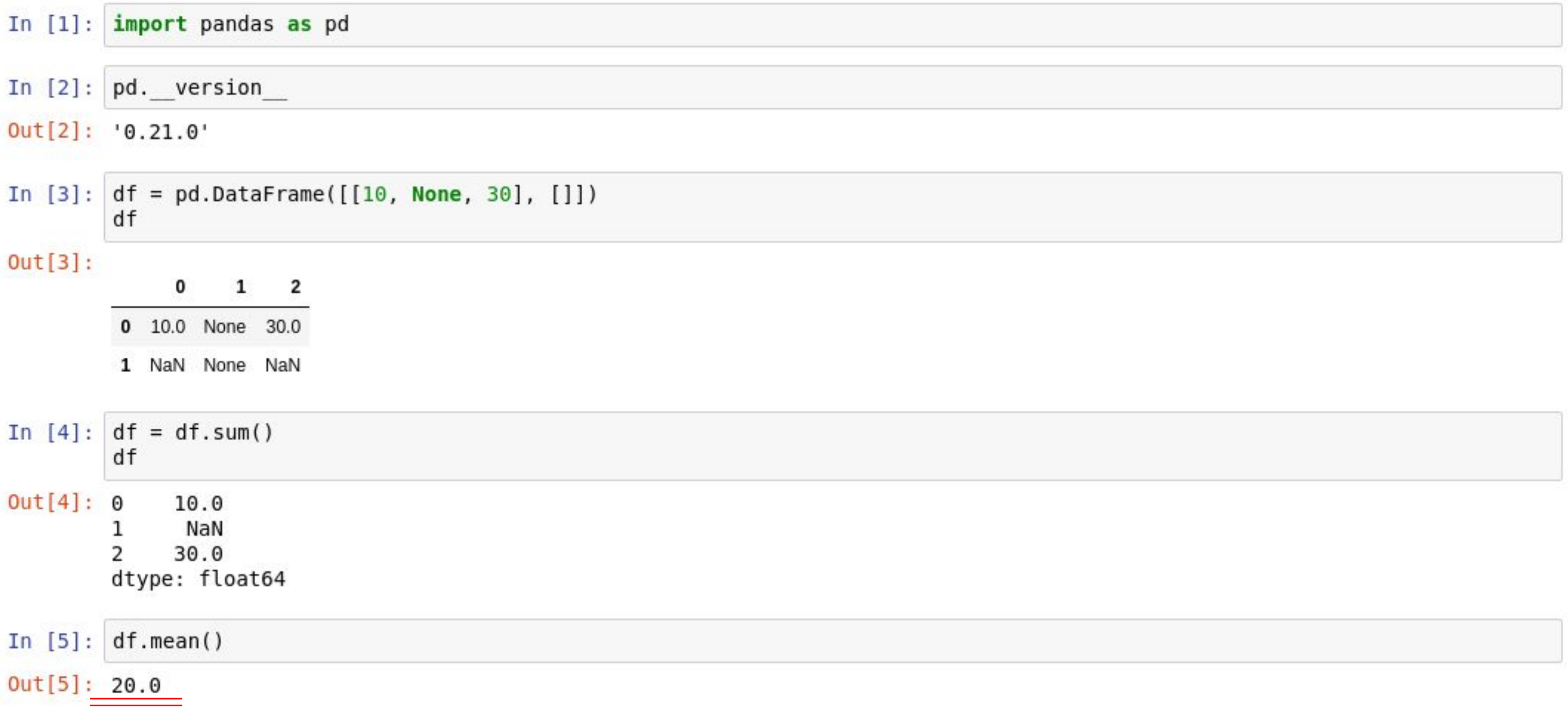
```
In [3]: df = pd.DataFrame([[10, None, 30], []])
        df
```

Out[3]:

	0	1	2
0	10.0	None	30.0
1	NaN	None	NaN

```
In [4]: df = df.sum()
        df
```

```
Out[4]: 0    10.0
        1     NaN
        2    30.0
        dtype: float64
```



```
In [1]: import pandas as pd
```

```
In [2]: pd.__version__
```

```
Out[2]: '0.21.0'
```

```
In [3]: df = pd.DataFrame([[10, None, 30], []])
        df
```

Out[3]:

	0	1	2
0	10.0	None	30.0
1	NaN	None	NaN

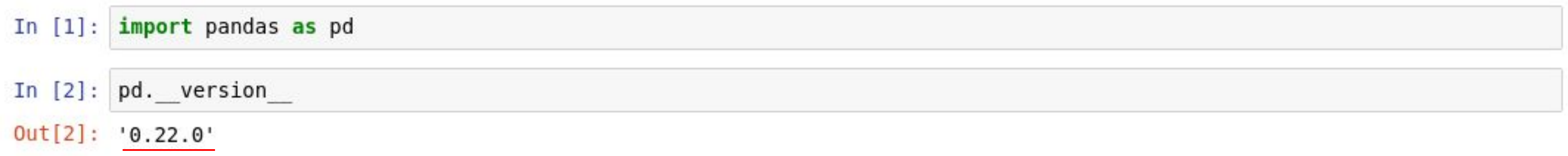
```
In [4]: df = df.sum()
        df
```

```
Out[4]: 0    10.0
        1     NaN
        2    30.0
        dtype: float64
```

```
In [5]: df.mean()
```

```
Out[5]: 20.0
```

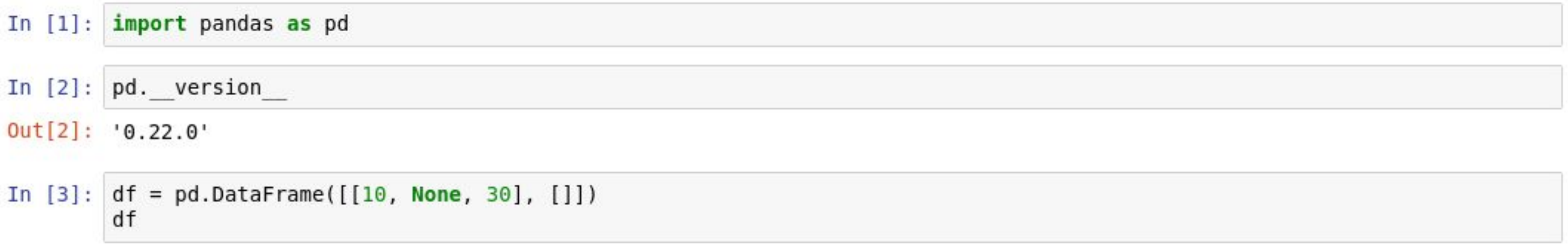
Behavior now



```
In [1]: import pandas as pd

In [2]: pd.__version__

Out[2]: '0.22.0'
```



```
In [1]: import pandas as pd
```

```
In [2]: pd.__version__
```

```
Out[2]: '0.22.0'
```

```
In [3]: df = pd.DataFrame([[10, None, 30], []])
        df
```

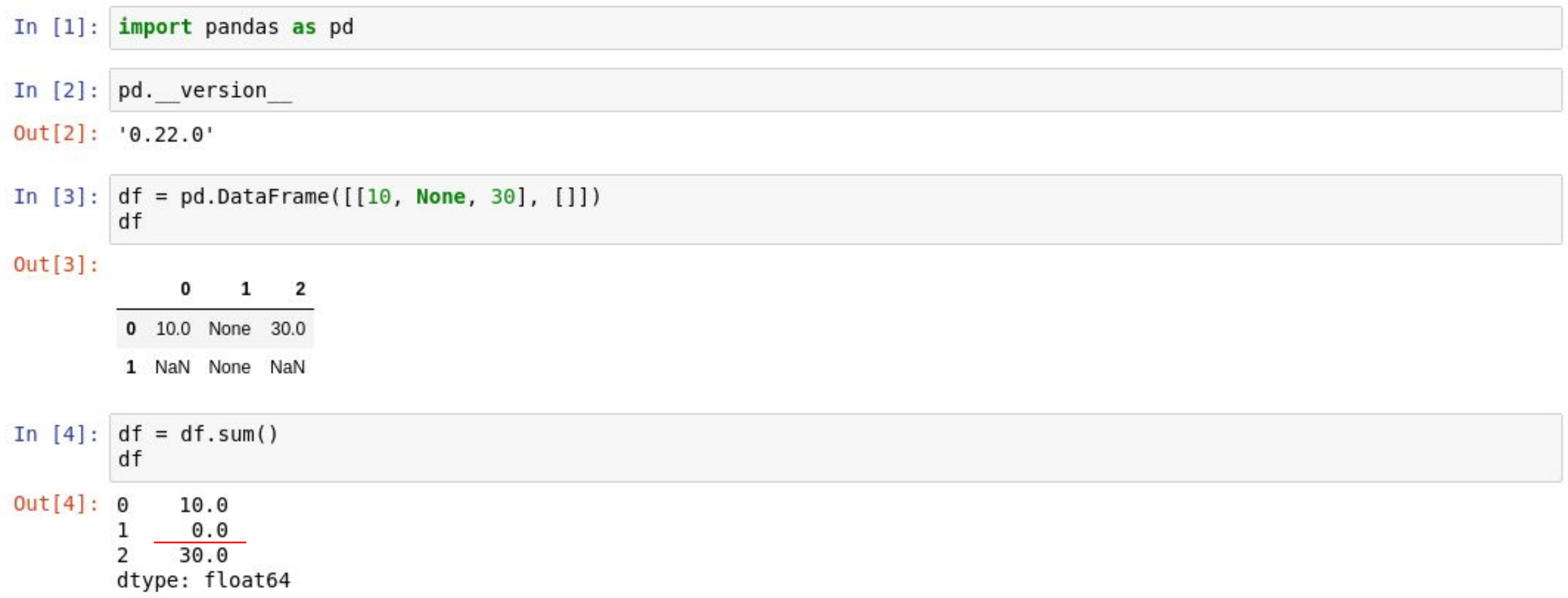
```
In [1]: import pandas as pd

In [2]: pd.__version__

Out[2]: '0.22.0'

In [3]: df = pd.DataFrame([[10, None, 30], []])
        df

Out[3]:
           0     1     2
        0  10.0  None  30.0
        1  NaN   None  NaN
```



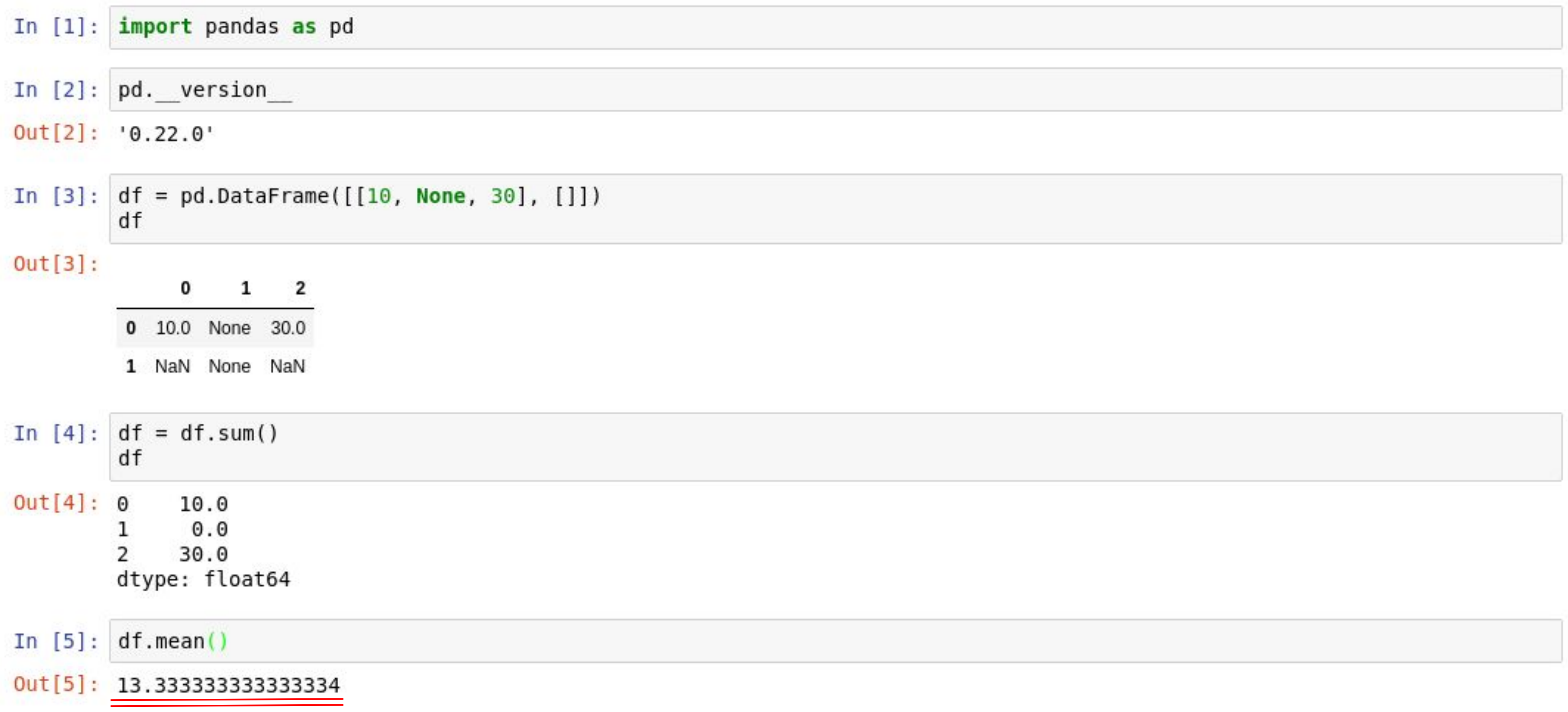
```
In [1]: import pandas as pd
```

```
In [2]: pd.__version__
Out[2]: '0.22.0'
```

```
In [3]: df = pd.DataFrame([[10, None, 30], []])
        df
Out[3]:
```

	0	1	2
0	10.0	None	30.0
1	NaN	None	NaN

```
In [4]: df = df.sum()
        df
Out[4]: 0    10.0
        1     0.0
        2    30.0
        dtype: float64
```



```
In [1]: import pandas as pd
```

```
In [2]: pd.__version__
Out[2]: '0.22.0'
```

```
In [3]: df = pd.DataFrame([[10, None, 30], []])
        df
Out[3]:
```

	0	1	2
0	10.0	None	30.0
1	NaN	None	NaN

```
In [4]: df = df.sum()
        df
Out[4]: 0    10.0
        1     0.0
        2    30.0
        dtype: float64
```

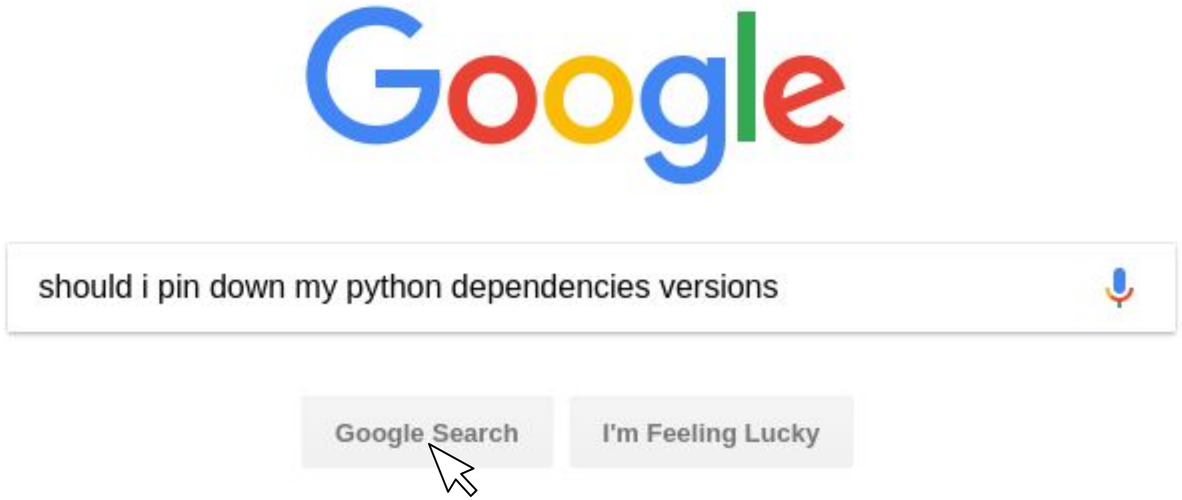
```
In [5]: df.mean()
Out[5]: 13.333333333333334
```

Change in behavior across versions

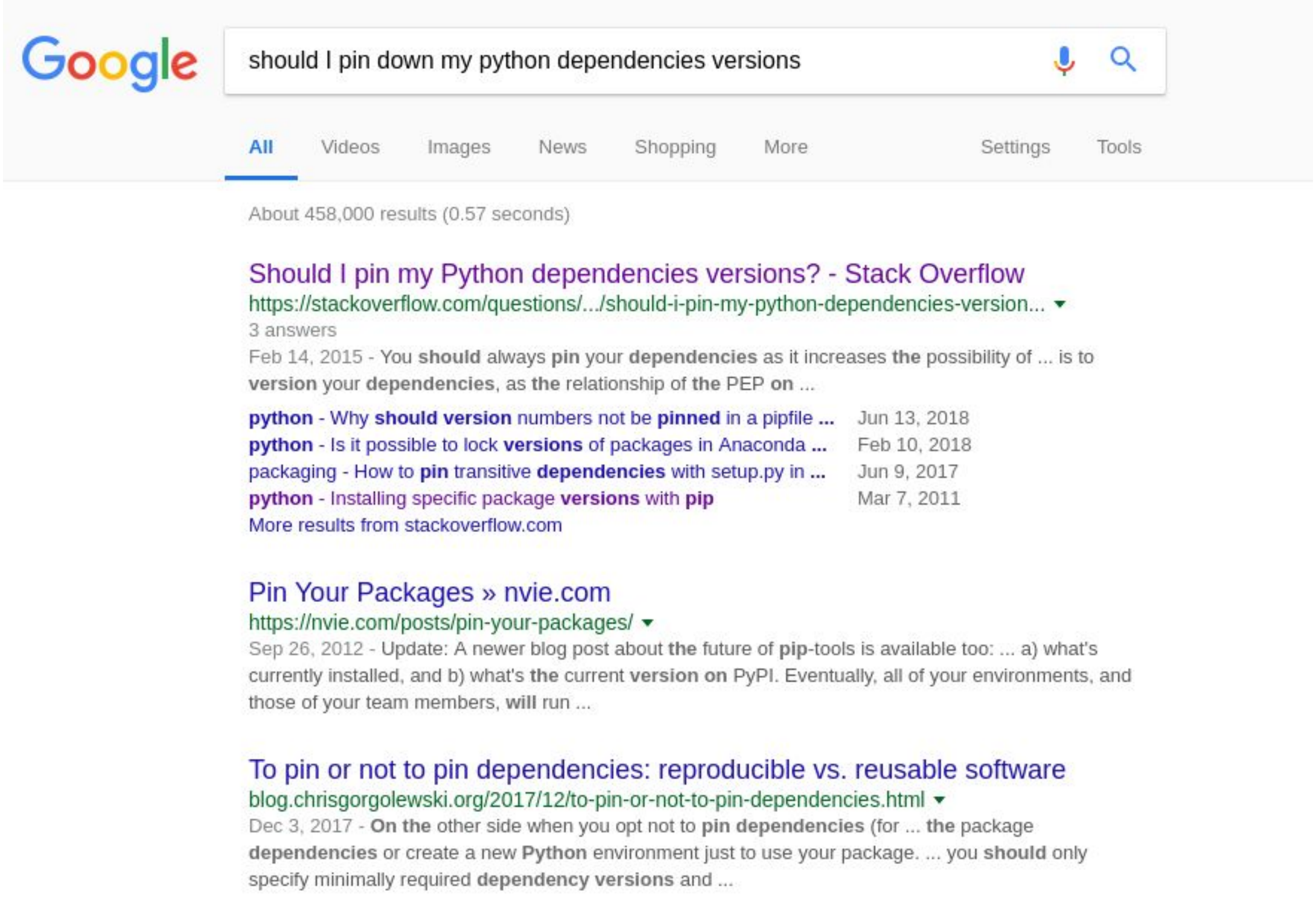
- What versions of libraries do I use to develop my application?

- What versions of libraries is my application compatible with?

- What if libraries I use change over time?
  - They change over time as can be seen!



should i pin down my python dependencies versions



should I pin down my python dependencies versions

About 458,000 results (0.57 seconds)

Should I pin my Python dependencies versions? - Stack Overflow
https://stackoverflow.com/questions/.../should-i-pin-my-python-dependencies-version...
3 answers
Feb 14, 2015 - You **should** always **pin** your **dependencies** as it increases **the** possibility of ... is to **version** your **dependencies**, as **the** relationship of **the** PEP **on** ...

python - Why **should version** numbers not be **pinned** in a pipfile **...**	Jun 13, 2018
python - Is it possible to lock **versions** of packages in Anaconda **...**	Feb 10, 2018
packaging - How to **pin** transitive **dependencies** with setup.py in **...**	Jun 9, 2017
python - Installing specific package **versions** with **pip**	Mar 7, 2011

More results from stackoverflow.com

Pin Your Packages » nvie.com
https://nvie.com/posts/pin-your-packages/
Sep 26, 2012 - Update: A newer blog post about **the** future of **pip**-tools is available too: ... a) what's currently installed, and b) what's **the** current **version on** PyPI. Eventually, all of your environments, and those of your team members, **will** run ...

To pin or not to pin dependencies: reproducible vs. reusable software
blog.chrisgorgolewski.org/2017/12/to-pin-or-not-to-pin-dependencies.html
Dec 3, 2017 - **On the** other side when you opt not to **pin dependencies** (for ... **the** package **dependencies** or create a new **Python** environment just to use your package. ... you **should** only specify minimally required **dependency versions** and ...

Should I pin my Python dependencies versions?

12

3

I am about to release a Python library I've been working on the past few weeks. I've read a lot about Python dependencies but something is not quite clear yet:

Some people pretend you should **never** pin your dependencies versions as it would prevent the users of your library from upgrading those dependencies.

Some other claim that you should **always** pin your dependencies versions as it is the only way of guaranteeing that your release works the way it did when you developped it and to prevent that a breaking change in a dependency wreaks havoc in your library.

I'm somehow went for an hybrid solution, where I assumed my dependencies used semantic versioning and pinned only the major version number (say `somelib >= 2.3.0, < 3`) except when the major version number is `0` (semantic versioning dictates that such versions are to be considered volatile and may break the API even if only the patch number is bumped).

As of now, I'm not sure which way is the best. Is there an official guideline (even a PEP perhaps ?) that dictates the best practice regarding Python dependencies and how to specify them ?

python dependencies pip versioning semantic-versioning

share improve this question

asked Feb 13 '15 at 21:59
ereOn
27.9k ●27 ●119 ●202

Good question. I think this should be addressed by the Python community. – pylang Dec 3 '15 at 19:52

add a comment

3 Answers

active oldest votes

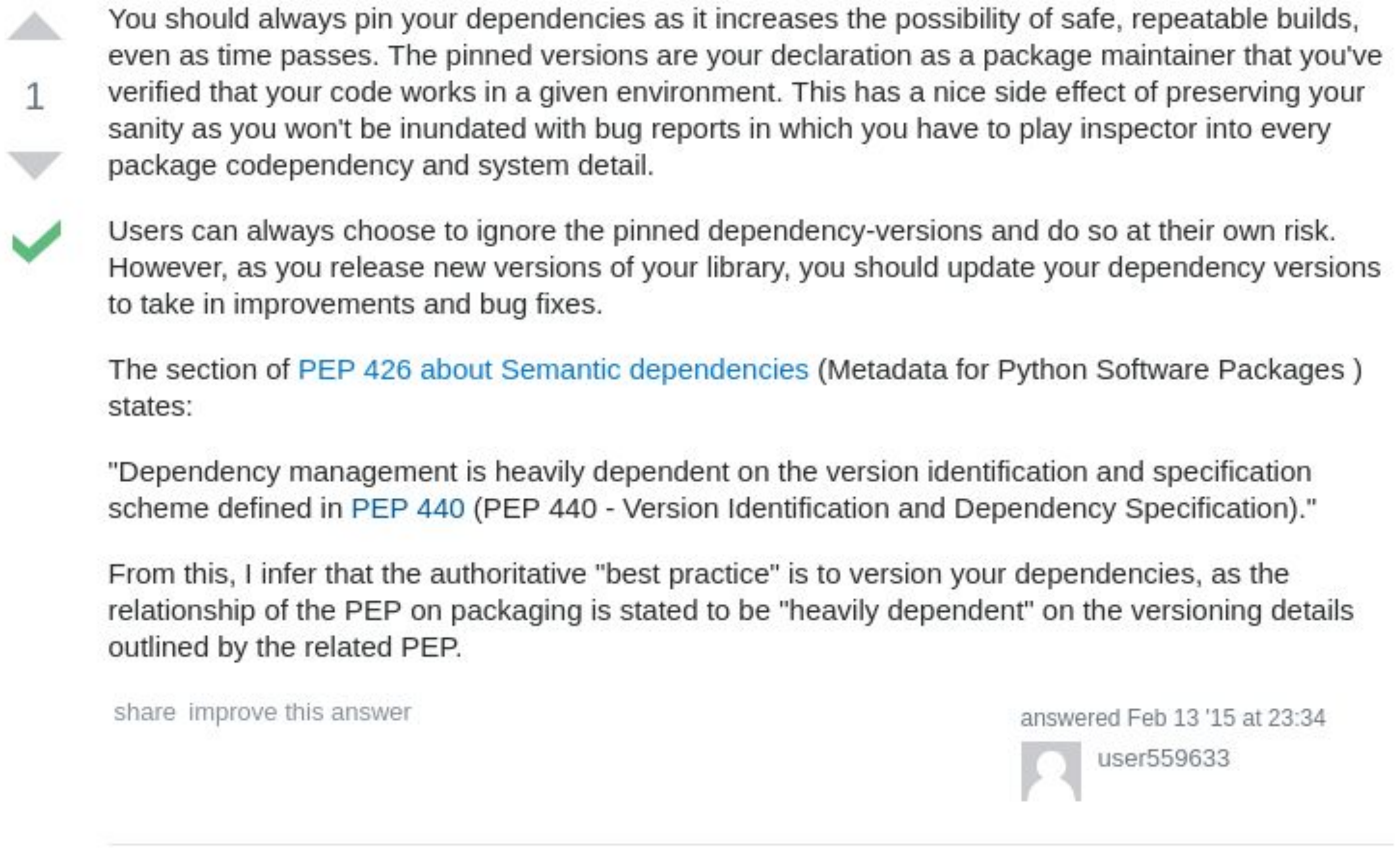
1

✓

You should always pin your dependencies as it increases the possibility of safe, repeatable builds, even as time passes. The pinned versions are your declaration as a package maintainer that you've verified that your code works in a given environment. This has a nice side effect of preserving your sanity as you won't be inundated with bug reports in which you have to play inspector into every package codependency and system detail.

Users can always choose to ignore the pinned dependency-versions and do so at their own risk. However, as you release new versions of your library, you should update your dependency versions to take in improvements and bug fixes.

The section of PEP 426 about Semantic dependencies (Metadata for Python Software Packages) states:



1

You should always pin your dependencies as it increases the possibility of safe, repeatable builds, even as time passes. The pinned versions are your declaration as a package maintainer that you've verified that your code works in a given environment. This has a nice side effect of preserving your sanity as you won't be inundated with bug reports in which you have to play inspector into every package codependency and system detail.

Users can always choose to ignore the pinned dependency-versions and do so at their own risk. However, as you release new versions of your library, you should update your dependency versions to take in improvements and bug fixes.

The section of PEP 426 about Semantic dependencies (Metadata for Python Software Packages) states:

"Dependency management is heavily dependent on the version identification and specification scheme defined in PEP 440 (PEP 440 - Version Identification and Dependency Specification)."

From this, I infer that the authoritative "best practice" is to version your dependencies, as the relationship of the PEP on packaging is stated to be "heavily dependent" on the versioning details outlined by the related PEP.

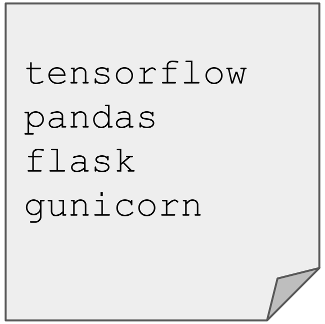
share improve this answer

answered Feb 13 '15 at 23:34

user559633

Requirements of my application

- requirements.txt file



```
tensorflow
pandas
flask
gunicorn
```

~~Requirements of my application~~

- requirements.txt file

```
tensorflow
pandas
flask
gunicorn
```

My requirements for the application

- requirements.**in** file

```
tensorflow
pandas
flask
gunicorn
```

My requirements for the application

- requirements.**in** file

```
tensorflow
pandas
flask
gunicorn
```

Requirements of the application

- requirements.**txt** file ?

Requirements of application

- requirements.txt file

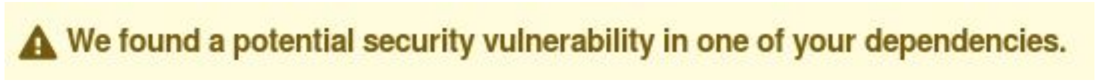
```
abs1-py==0.3.0              # via tensorflow
astor==0.7.1                # via tensorflow
click==6.7                  # via flask
flask==0.6.0
gast==0.2.0                 # via tensorflow
grpcio==1.14.1              # via tensorflow
gunicorn==19.9.0
itsdangerous==0.24          # via flask
jinja2==2.10                # via flask
markdown==2.6.11            # via tensorboard
markupsafe==1.0             # via jinja2
numpy==1.14.5               # via pandas, tensorboard, tensorflow
pandas==0.23.4
protobuf==3.6.0             # via tensorboard, tensorflow
python-dateutil==2.7.3      # via pandas
pytz==2018.5                # via pandas
six==1.11.0                 # via absl-py, grpcio, protobuf, python-dateutil, tensorboard, tensorflow
tensorboard==1.10.0         # via tensorflow
tensorflow==1.10.0
termcolor==1.1.0            # via tensorflow
werkzeug==0.14.1            # via flask, tensorboard
wheel==0.31.1               # via tensorboard, tensorflow
```

Requirements of application

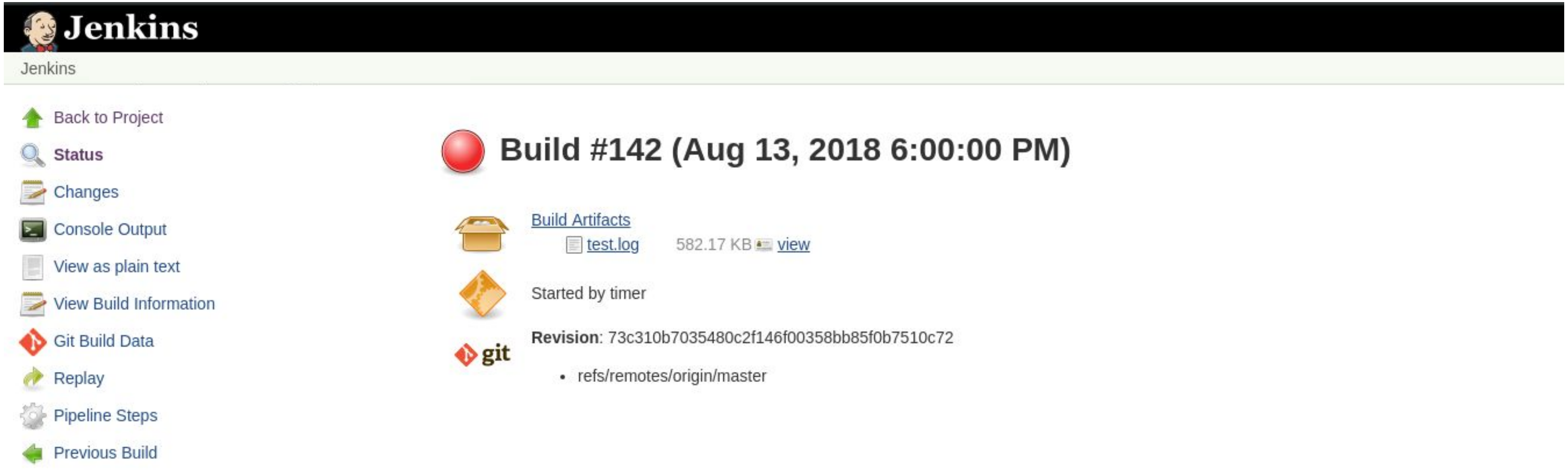
- Fully pinned down software stack
- Transitive dependencies
- Resolution is no more time dependent
- Recommended - use pipenv

```
$ git commit -m “Pin down dependencies”

$ git push
```


⚠ We found a potential security vulnerability in one of your dependencies.

```
$ git commit -m “Excluded vulnerable version”
$ git push
```



Jenkins

Jenkins

- Back to Project
- **Status**
- Changes
- Console Output
- View as plain text
- View Build Information
- Git Build Data
- Replay
- Pipeline Steps
- Previous Build

Build #142 (Aug 13, 2018 6:00:00 PM)

Build Artifacts

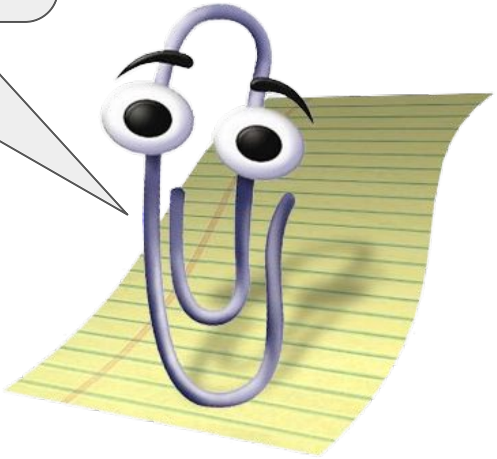
test.log 582.17 KB view

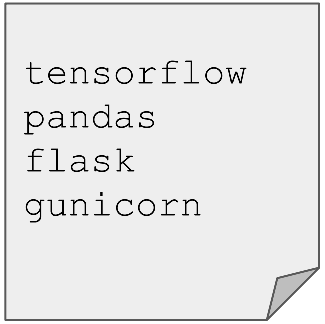
Started by timer

Revision: 73c310b7035480c2f146f00358bb85f0b7510c72

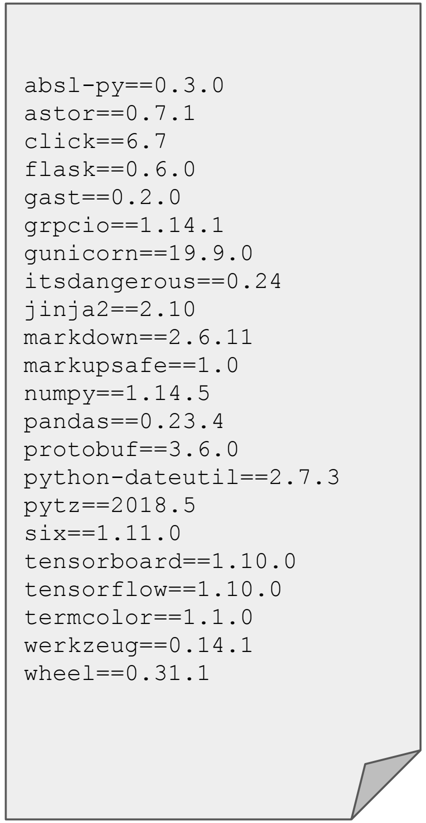
- refs/remotes/origin/master

Hey, do you know the fact you are definitely using wrong software?



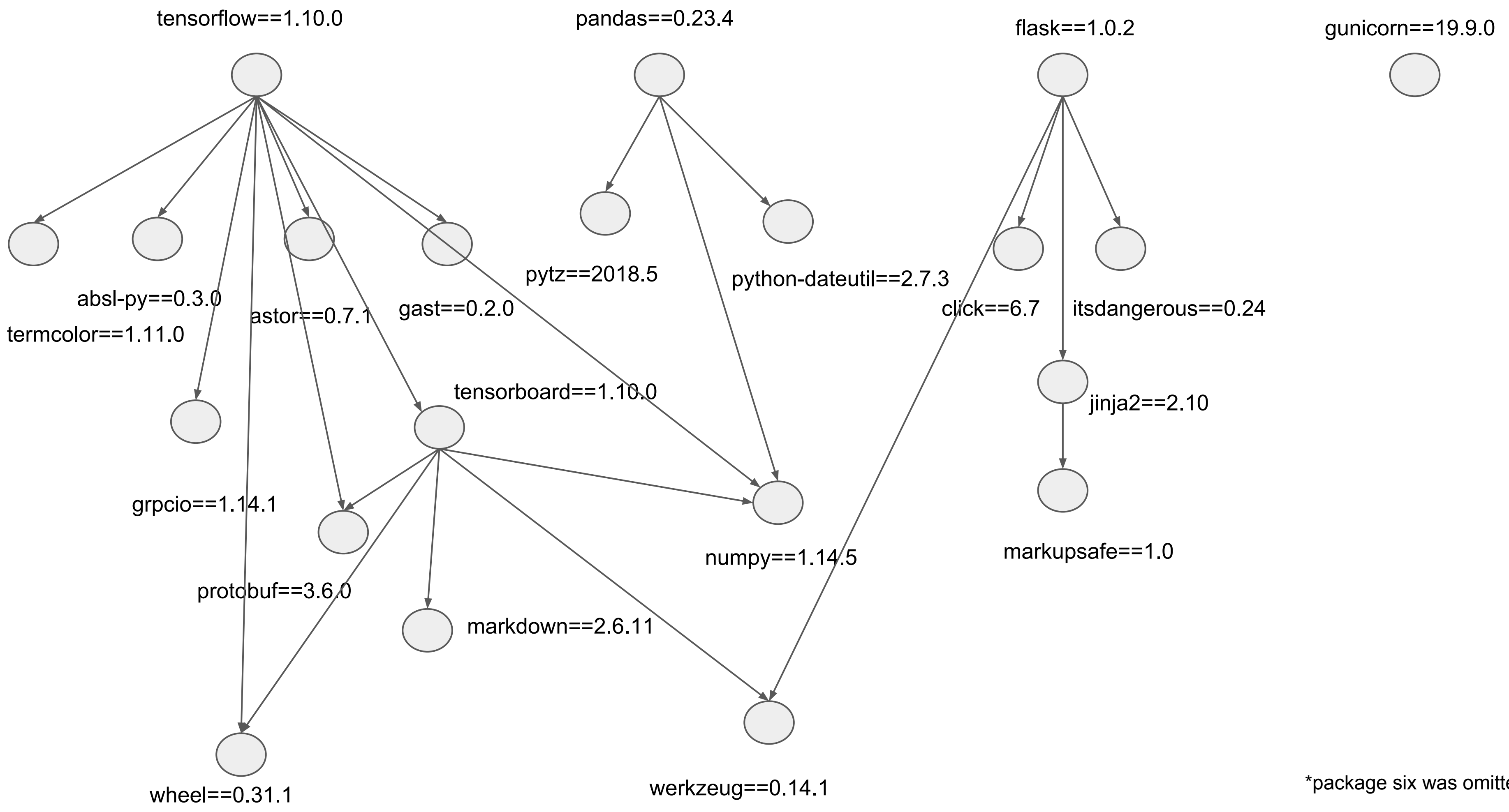


```
tensorflow
pandas
flask
gunicorn
```

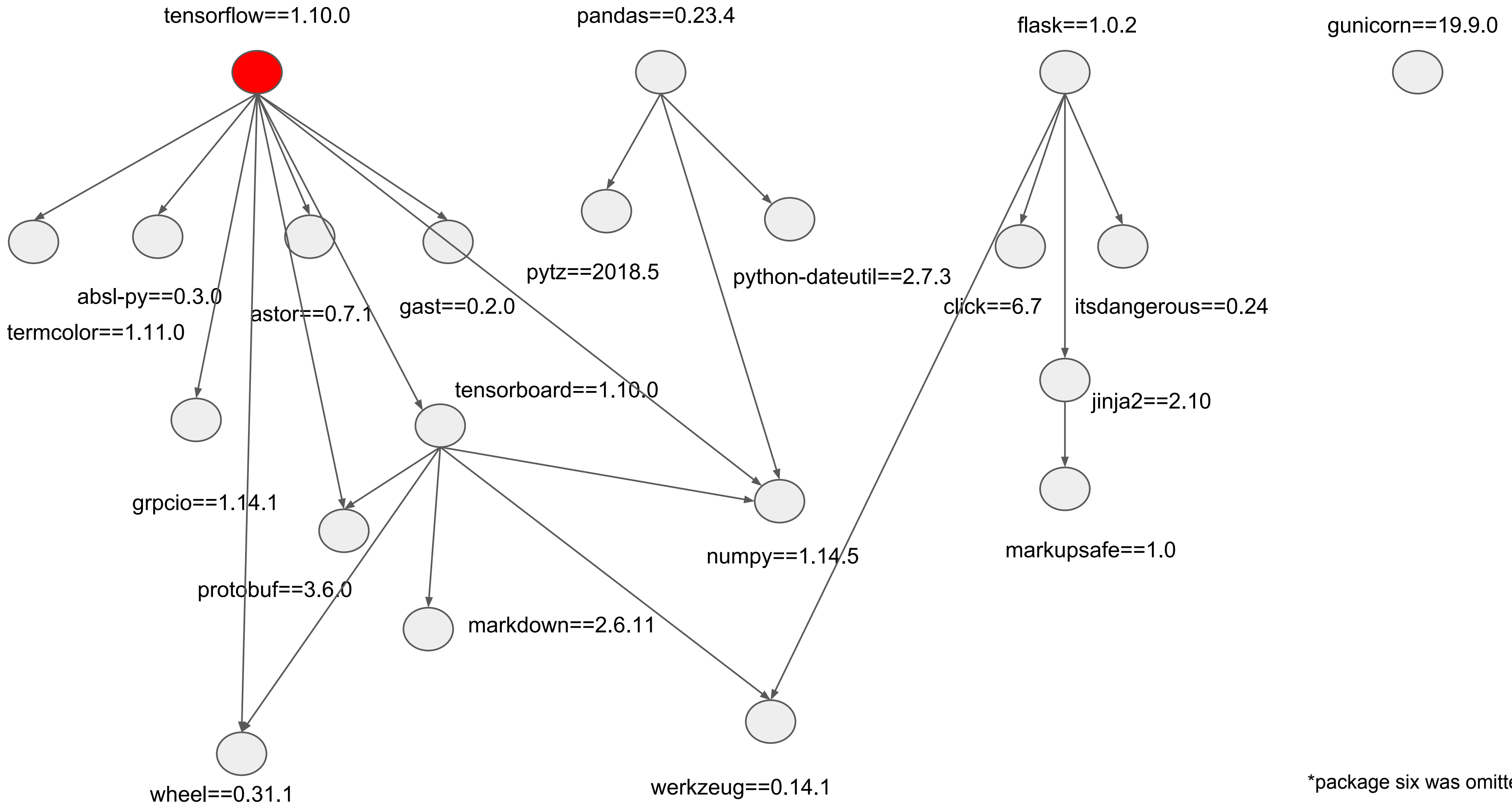


```
absl-py==0.3.0
astor==0.7.1
click==6.7
flask==0.6.0
gast==0.2.0
grpcio==1.14.1
gunicorn==19.9.0
itsdangerous==0.24
jinja2==2.10
markdown==2.6.11
markupsafe==1.0
numpy==1.14.5
pandas==0.23.4
protobuf==3.6.0
python-dateutil==2.7.3
pytz==2018.5
six==1.11.0
tensorboard==1.10.0
tensorflow==1.10.0
termcolor==1.1.0
werkzeug==0.14.1
wheel==0.31.1
```

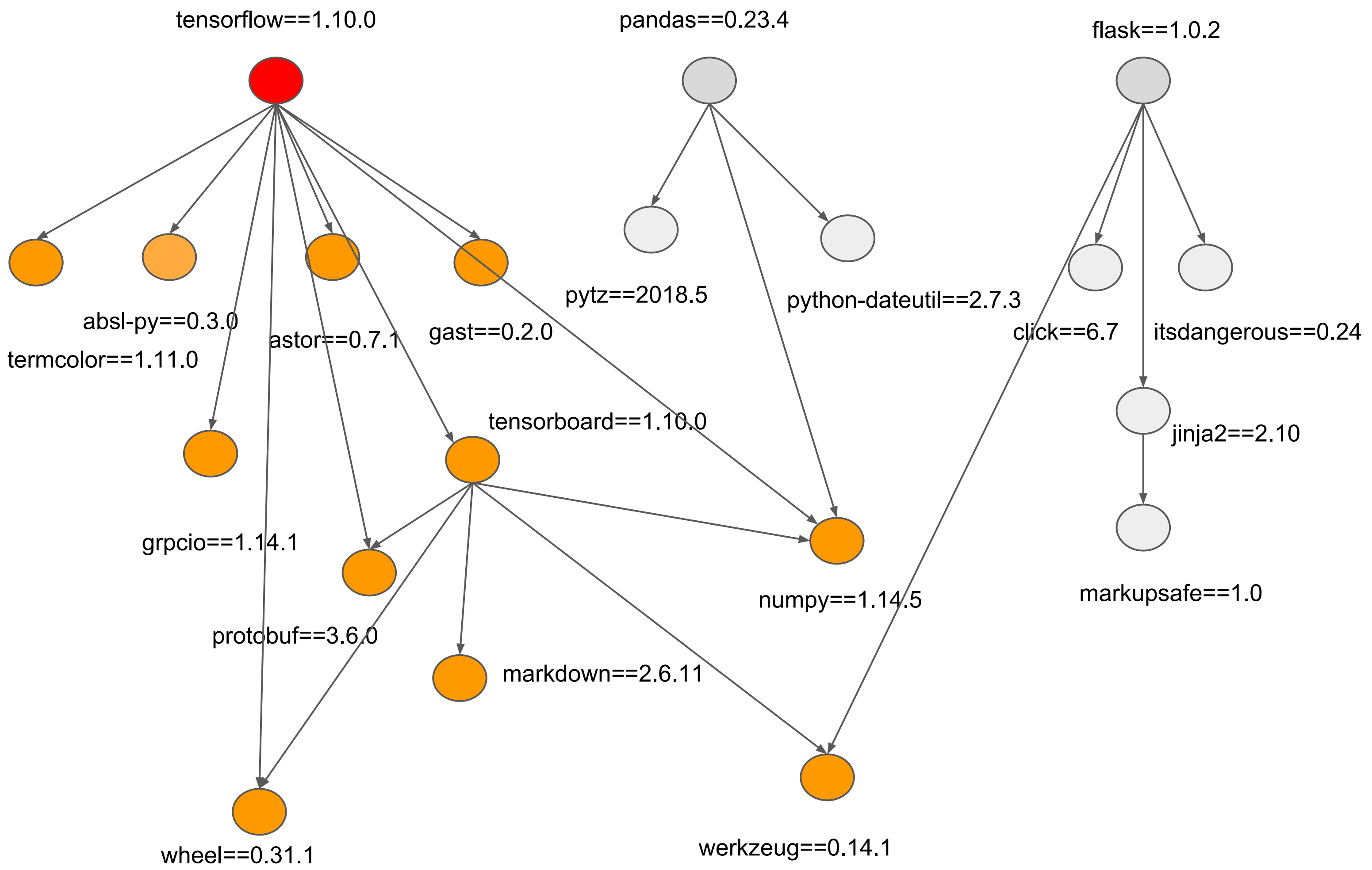
tensorflow==1.10.0
pandas==0.23.4
flask==1.0.2
gunicorn==19.9.0



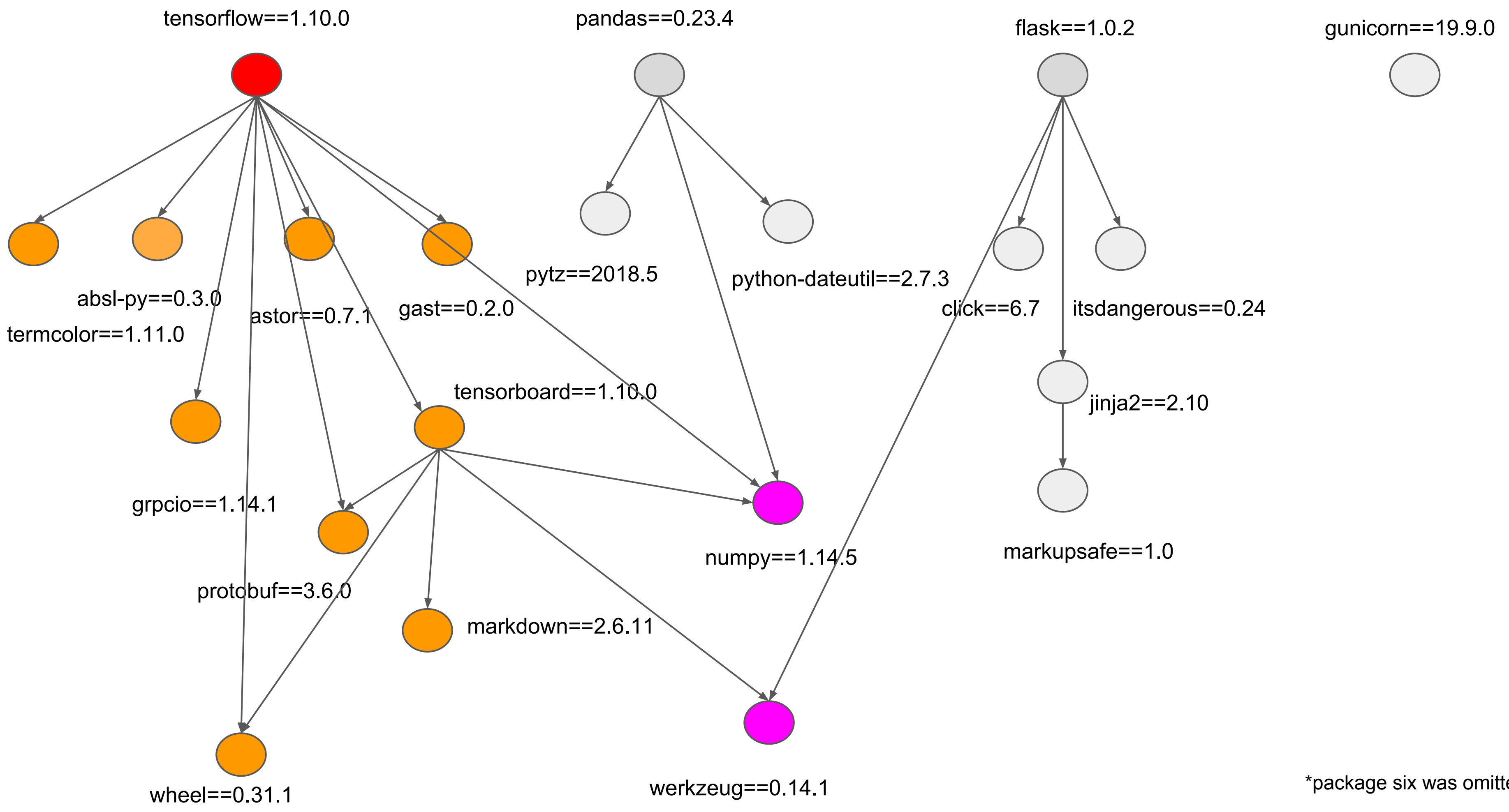
*package six was omitted



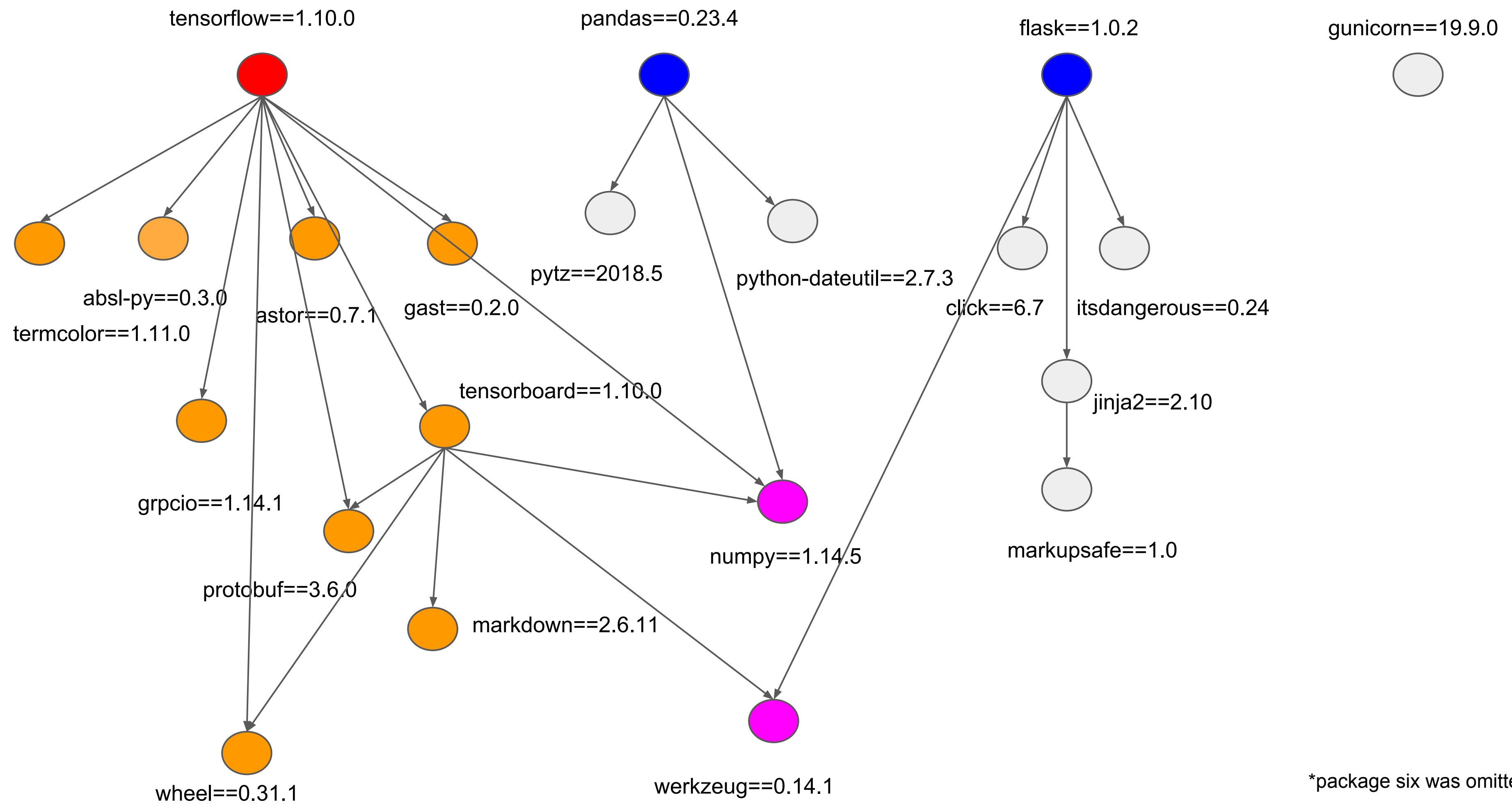
*package six was omitted



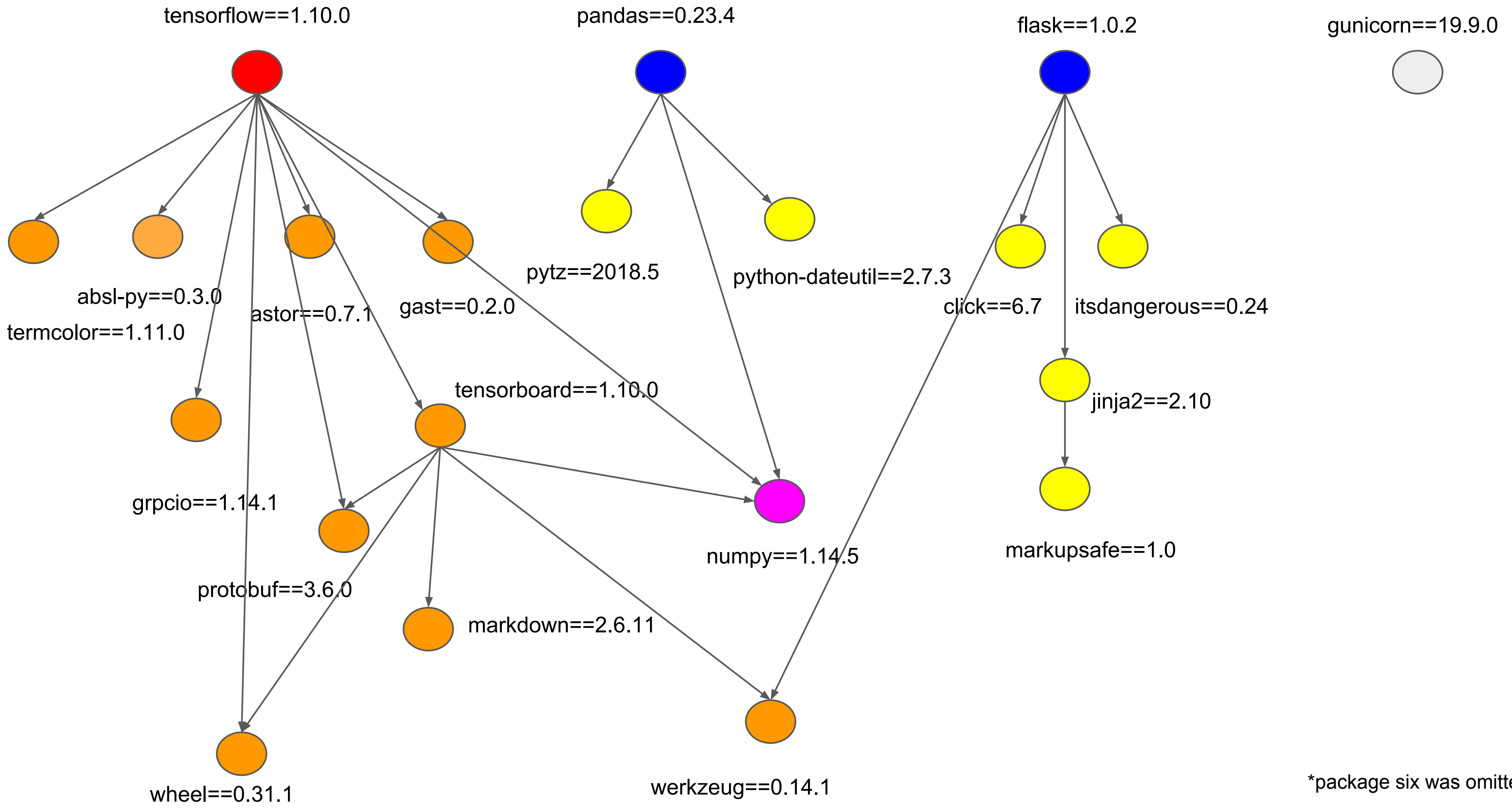
*package six was omitted



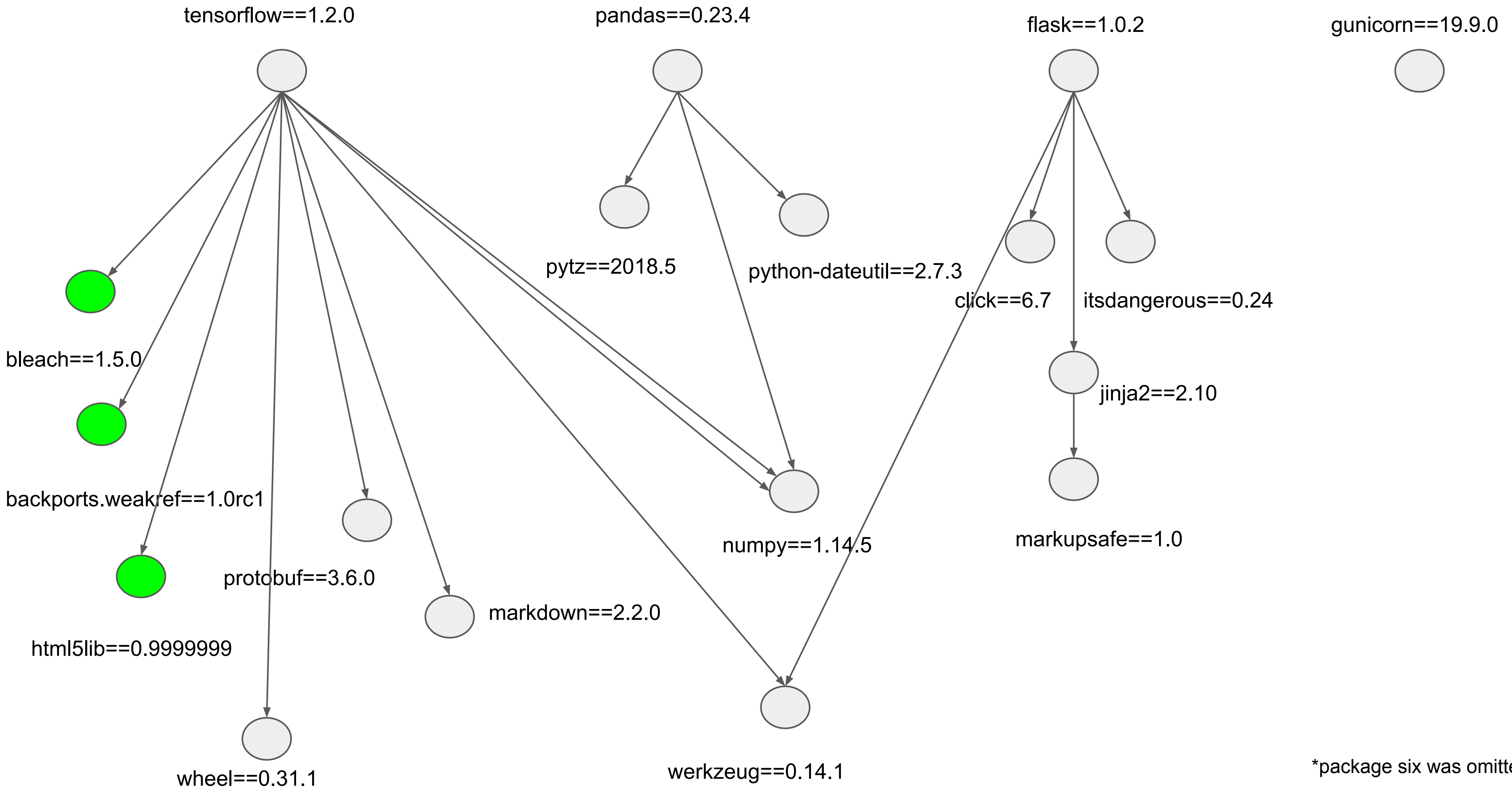
*package six was omitted



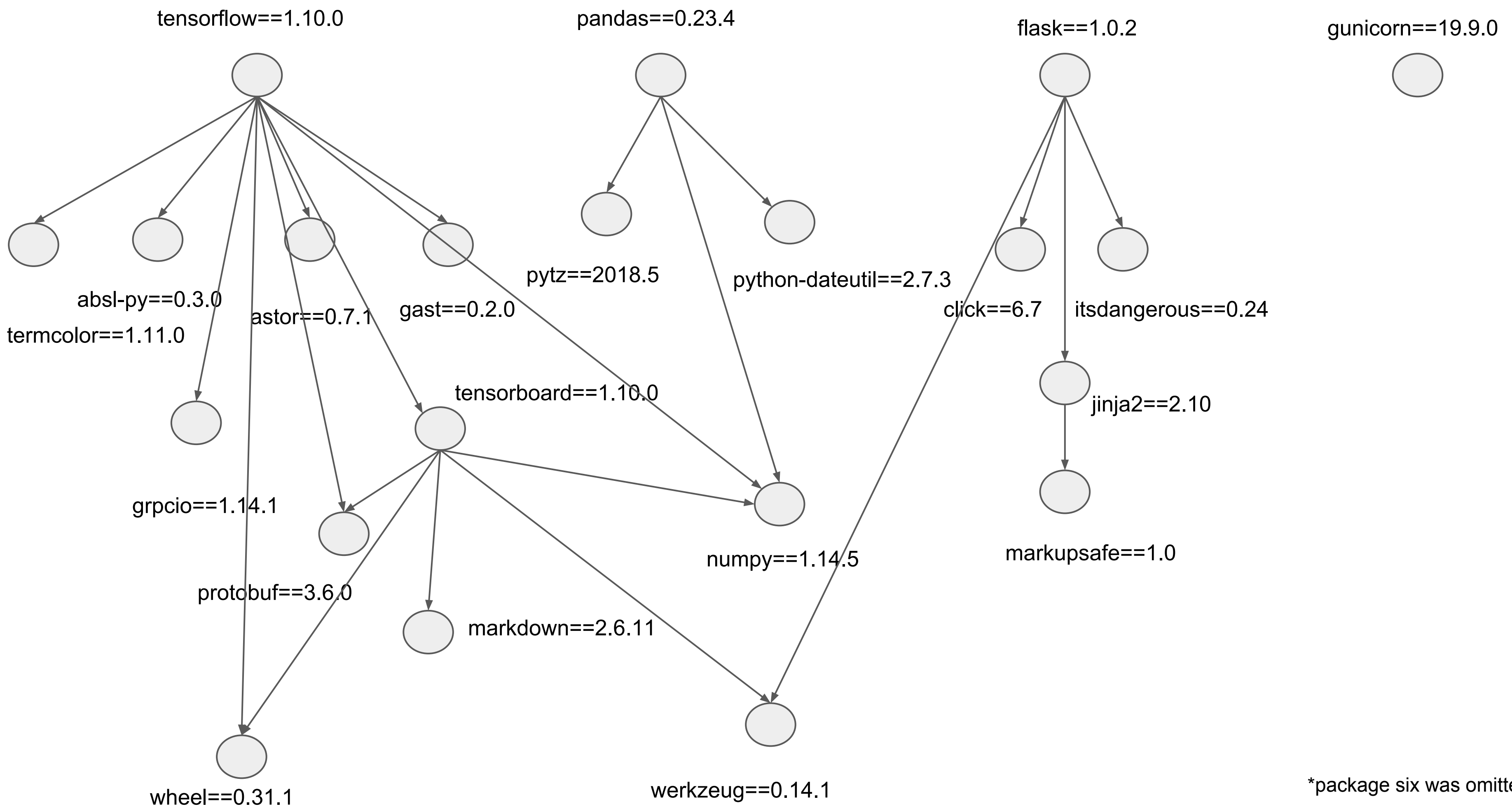
*package six was omitted



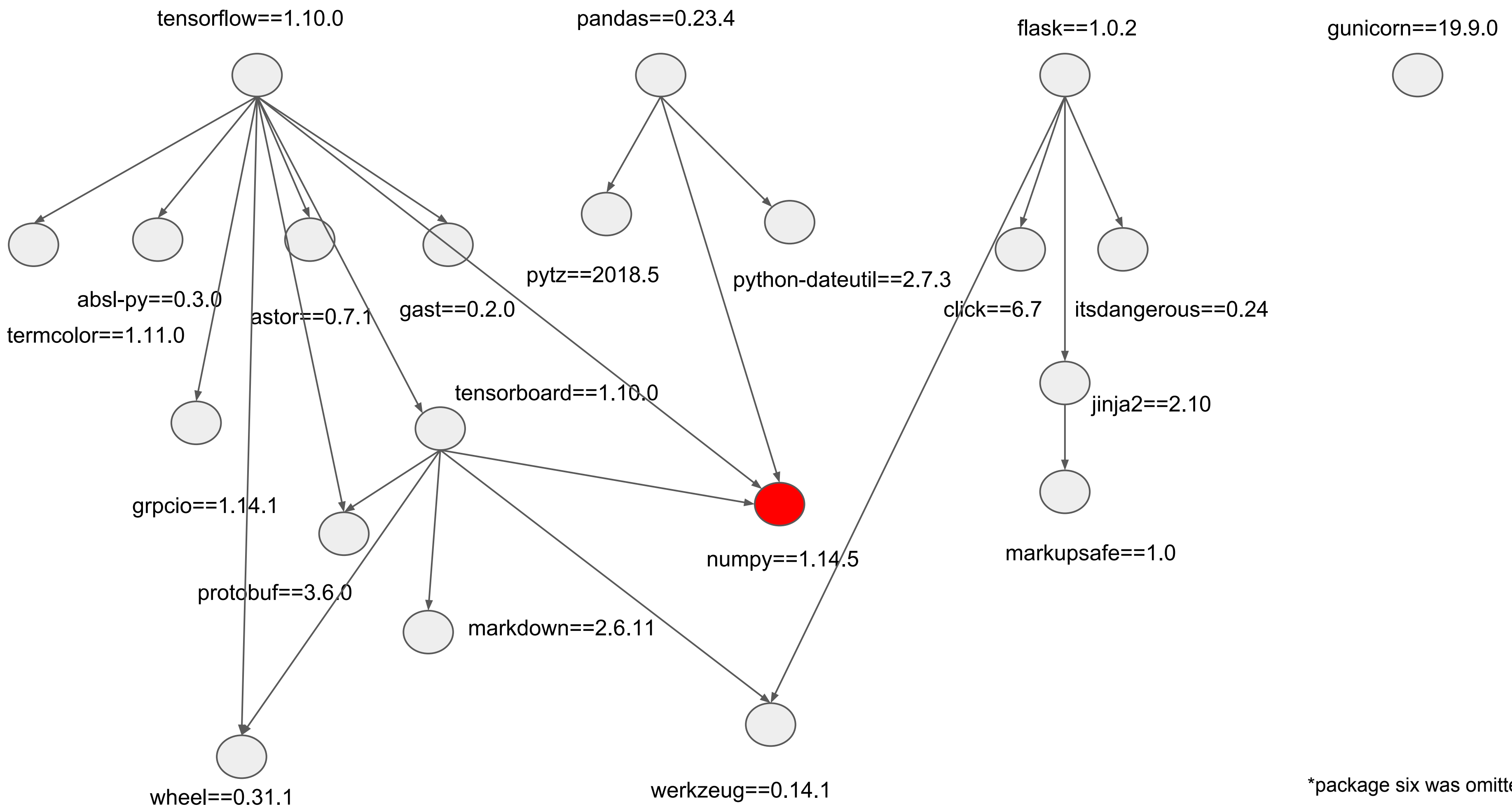
*package six was omitted



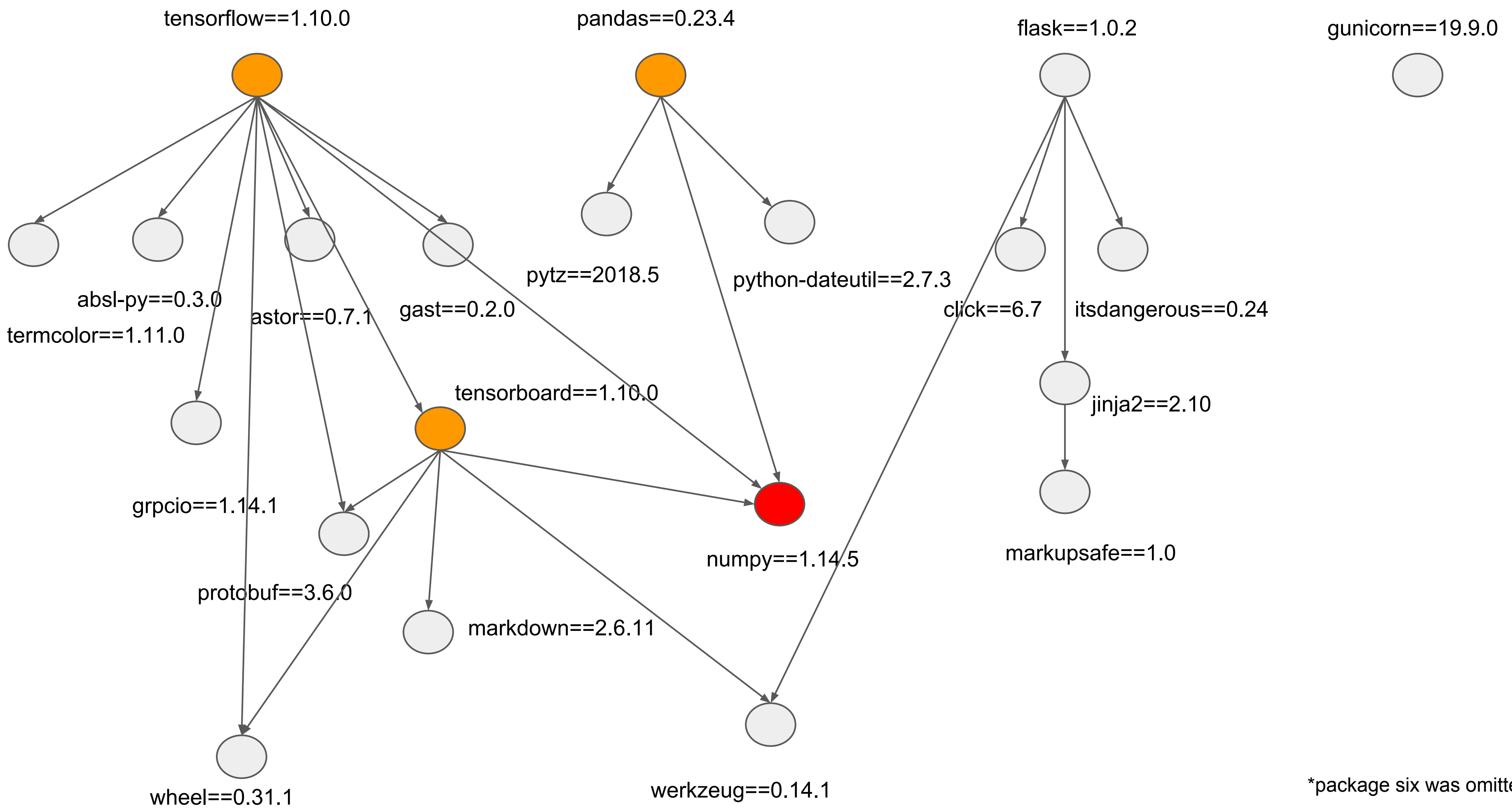
*package six was omitted



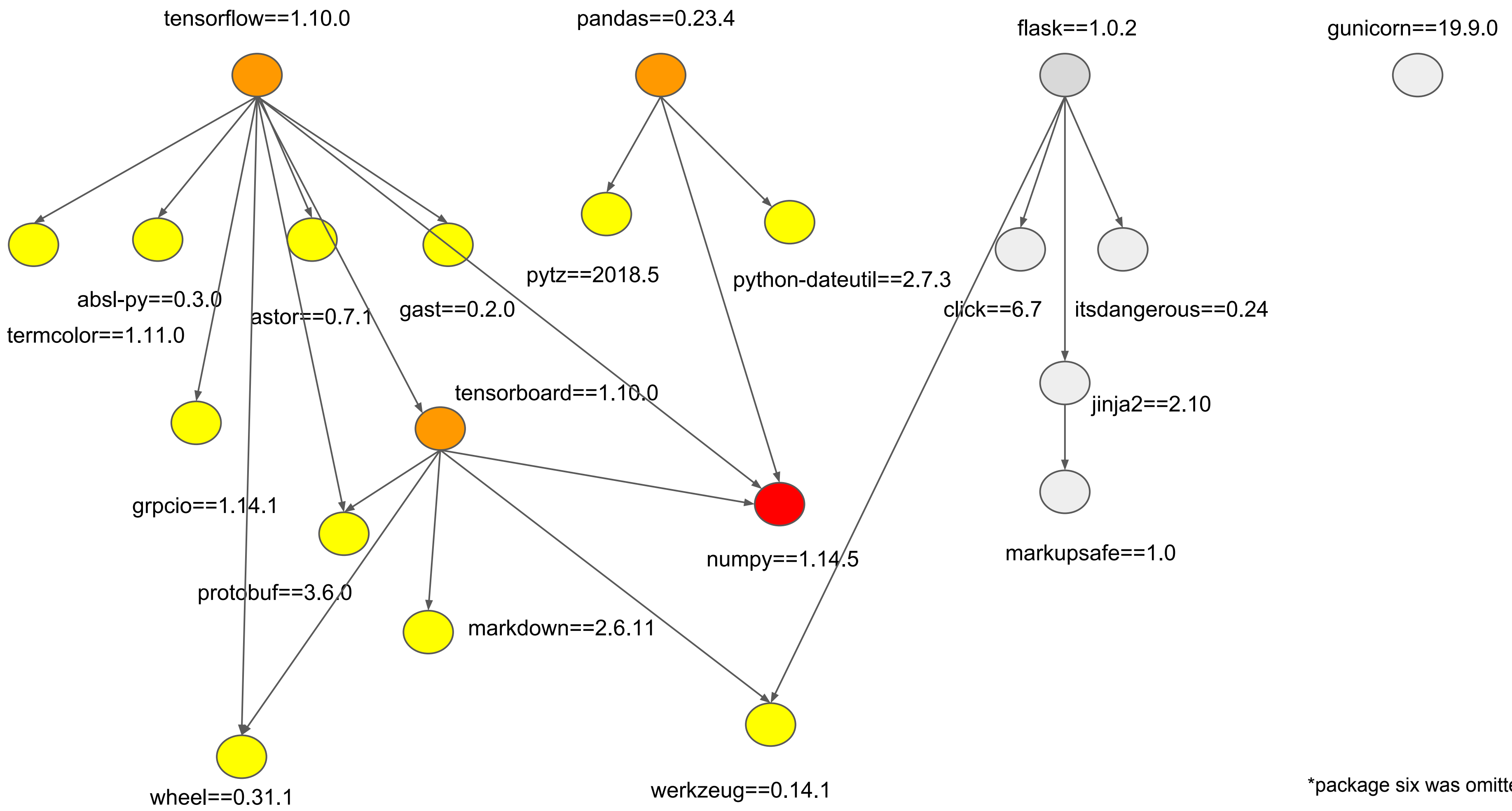
*package six was omitted



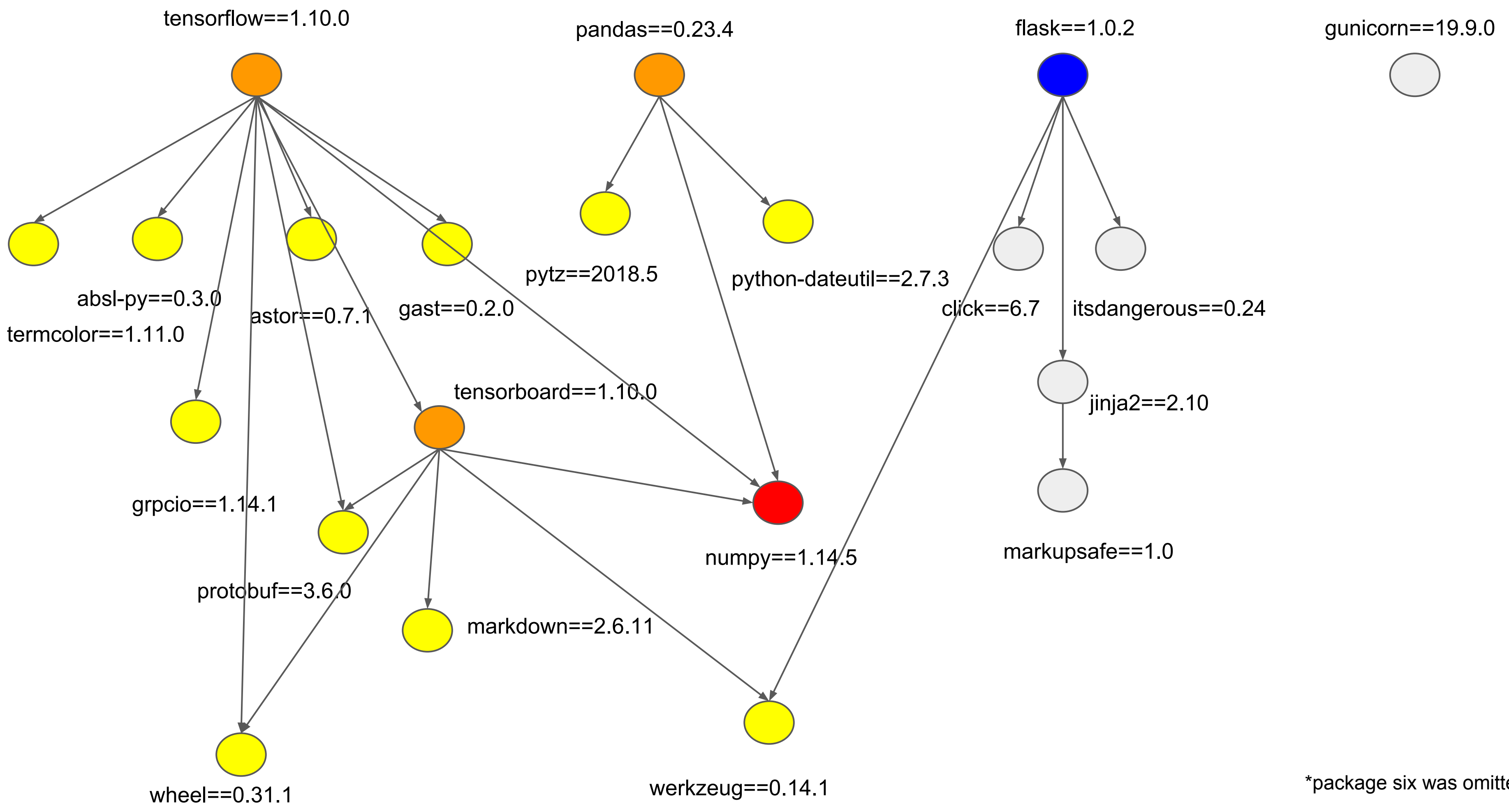
*package six was omitted



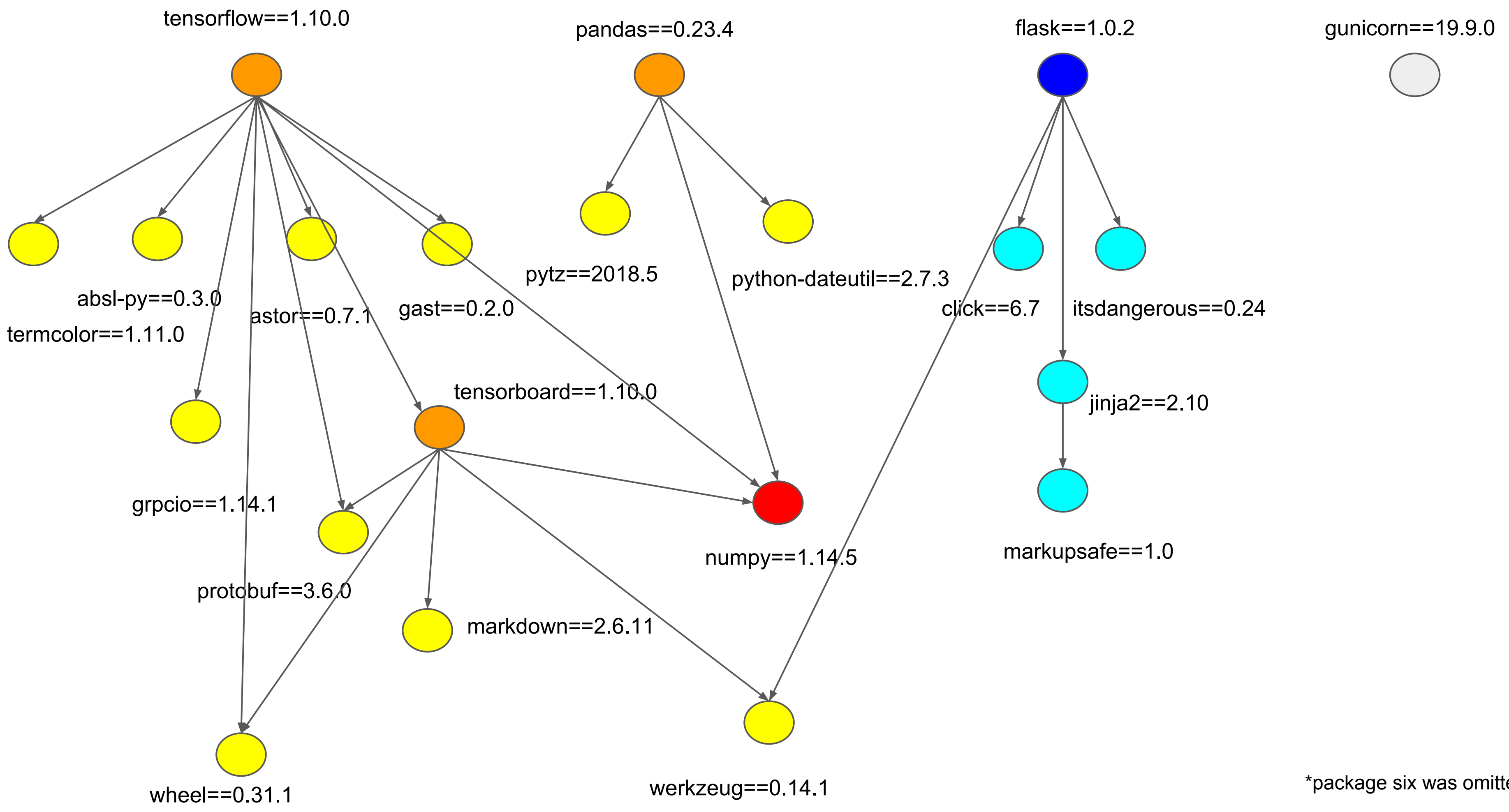
*package six was omitted



*package six was omitted



*package six was omitted



*package six was omitted

Thoth

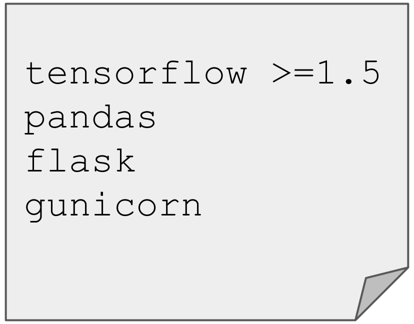
- Server side resolution
- You state what you want, we what the resolved stack should look like
    - Based on experience we have
    - Based on observations we have
    - Based on monitoring we have

Thoth

- Server side resolution
- You state what you want, we what the resolved stack should look like
    - Based on experience we have
    - Based on observations we have
    - Based on monitoring we have

Knowledge

Your requirements for application



```
tensorflow >=1.5
pandas
flask
gunicorn
```

Thoth’s recommendation on libraries you should use

- What is wrong
- What you should use
- Why you should use it

```
absl-py==0.3.0              # tensorflow
astor==0.7.1                # tensorflow
click==6.7                  # flask
flask==1.0.2
gast==0.2.0                 # tensorflow
grpcio==1.14.1              # tensorflow
gunicorn==19.9.0
itsdangerous==0.24          # flask
jinja2==2.10                # flask
markdown==2.6.11            # tensorboard
markupsafe==1.0             # jinja2
numpy==1.14.5               # pandas, tensorboard, tensorflow
pandas==0.23.4
protobuf==3.6.0             # tensorboard, tensorflow
python-dateutil==2.7.3      # pandas
pytz==2018.5                # pandas
six==1.11.0                 # absl-py, grpcio, protobuf, python-dateutil, tensorboard,
tensorboard==1.10.0         # tensorflow
tensorflow==1.10.0
termcolor==1.1.0            # tensorflow
werkzeug==0.14.1            # flask, tensorboard
wheel==0.31.1               # tensorboard, tensorflow
```

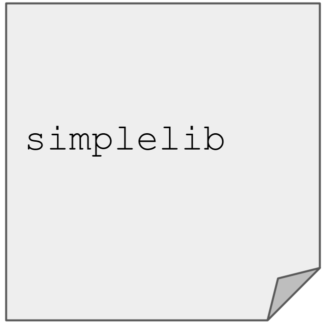
Thoth

- Your actual team member
- Bots proactively open pull requests
- You review

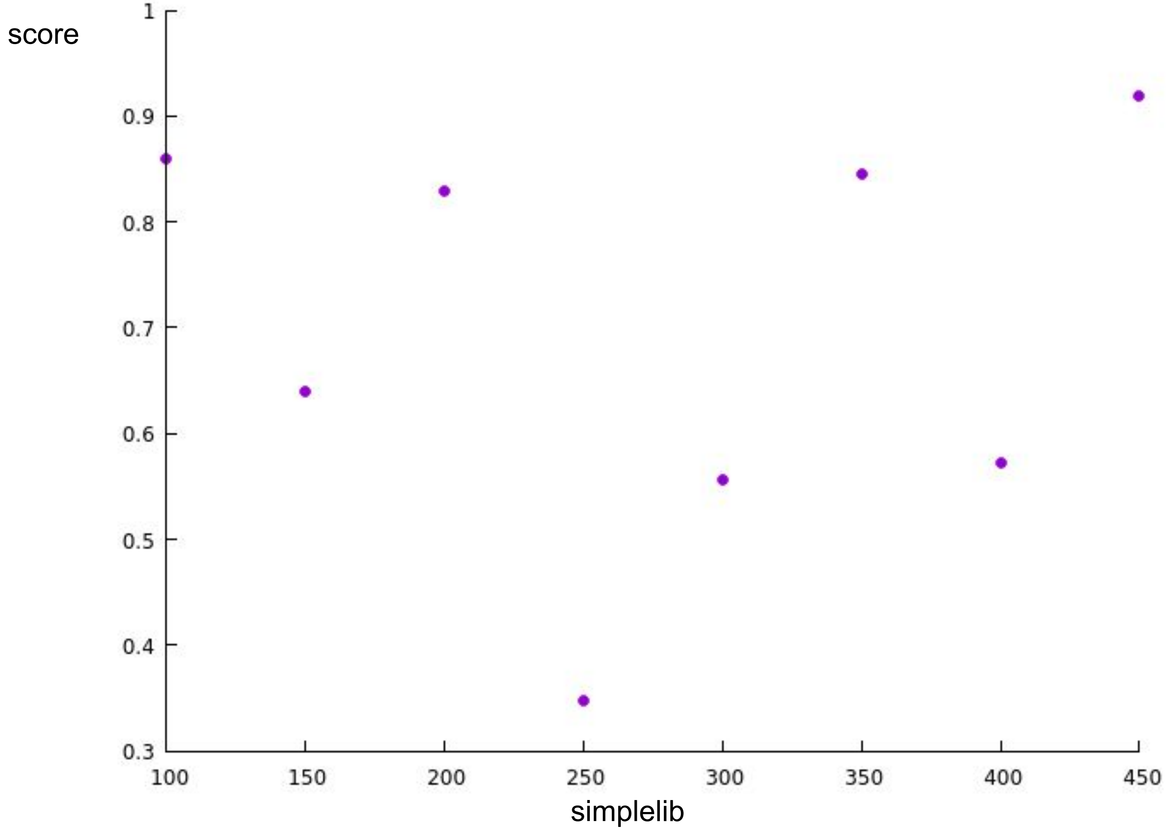
How to find the best software stack?

How good is my software stack?

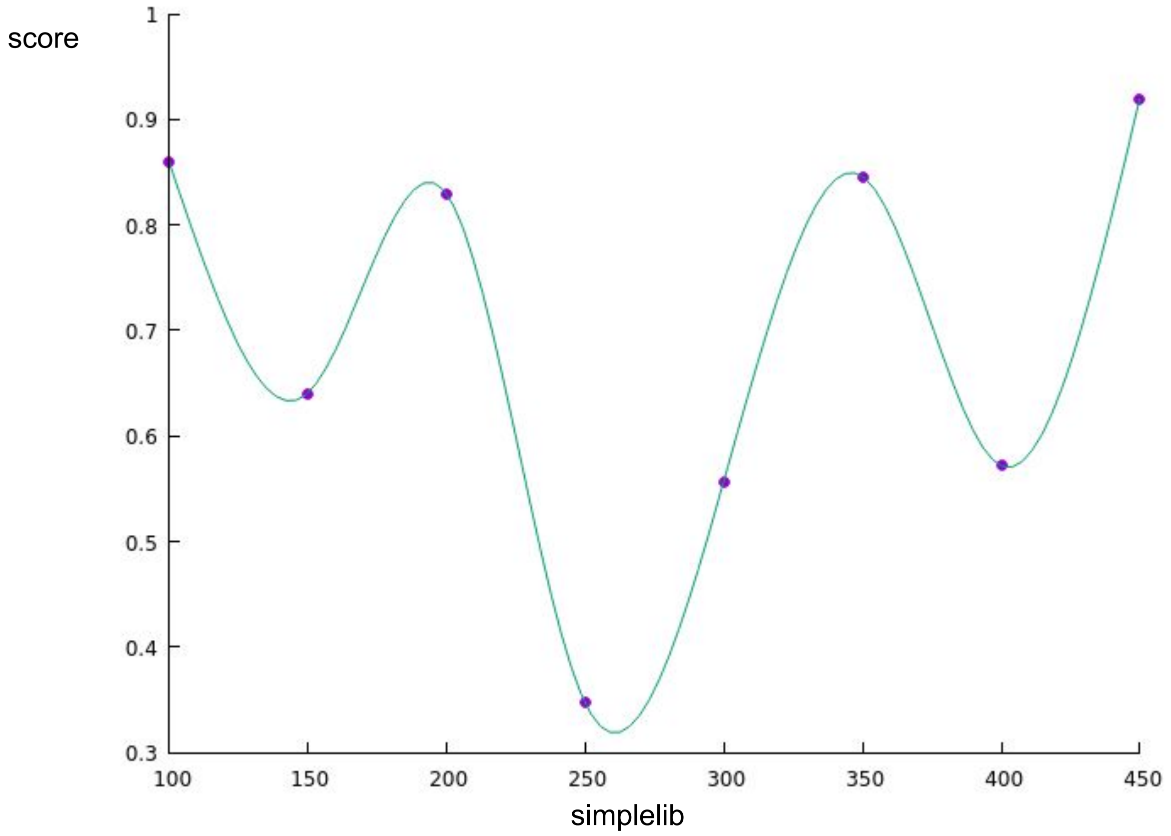
- Best = we need a mechanism to score software stack
- Let’s have an application with one depndency
- This dependency has no dependencies
  - no transitive/indirect dependencies



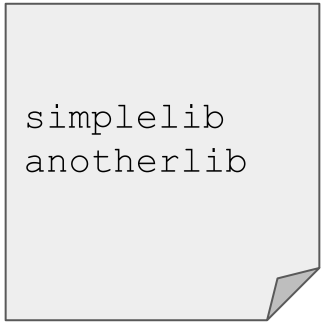
Scoring function - 2D



Scoring function - 2D

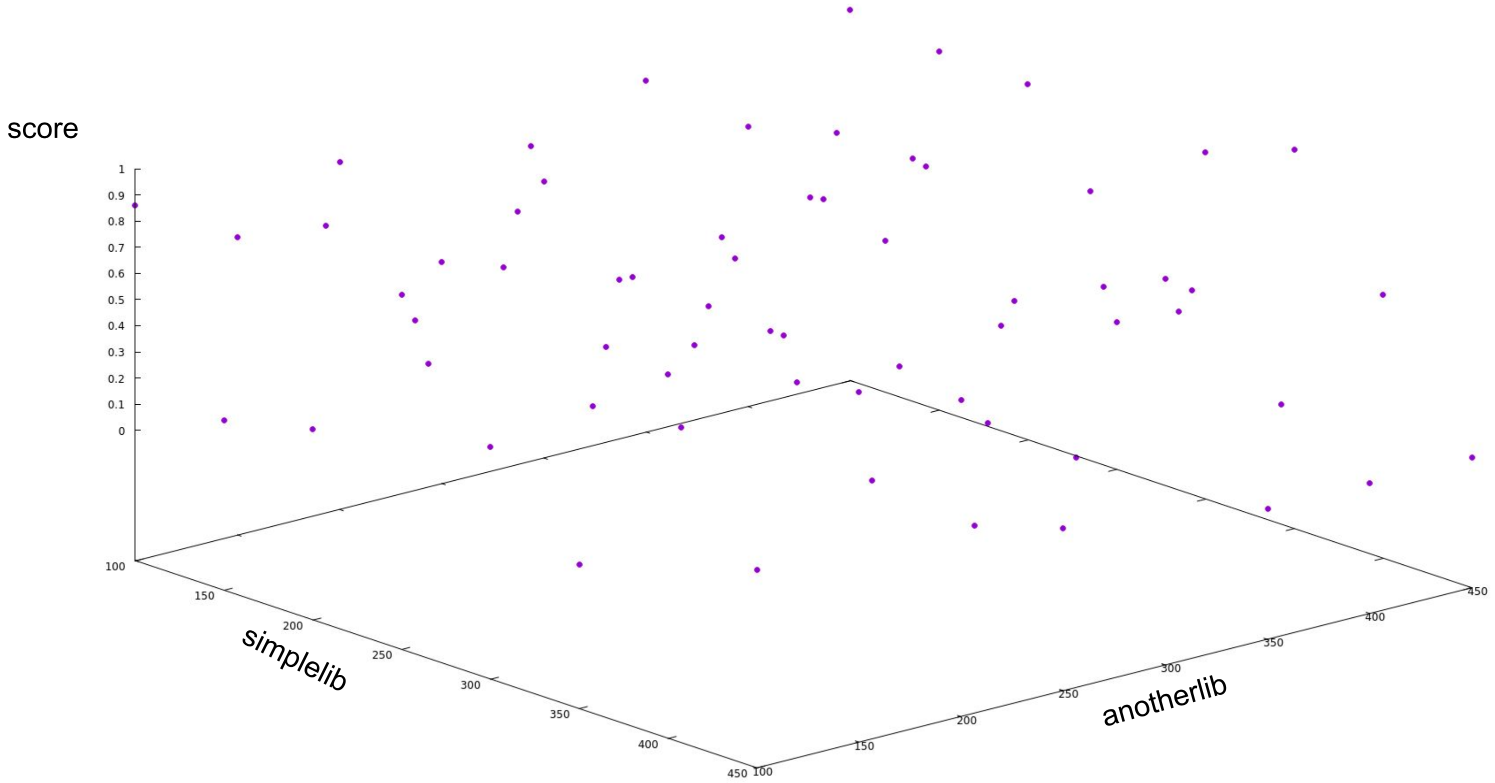


How good is my software stack?

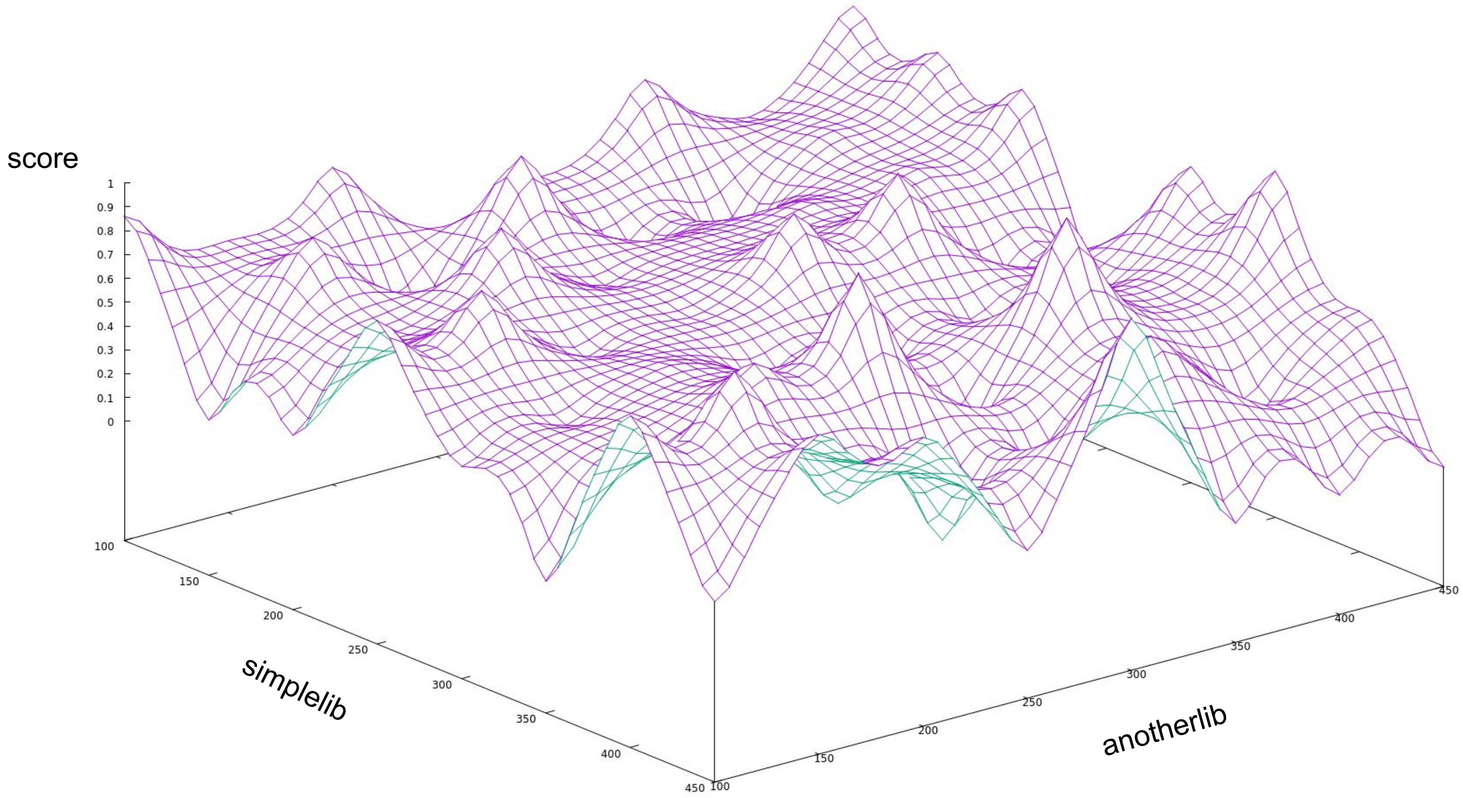


```
simplelib
anotherlib
```

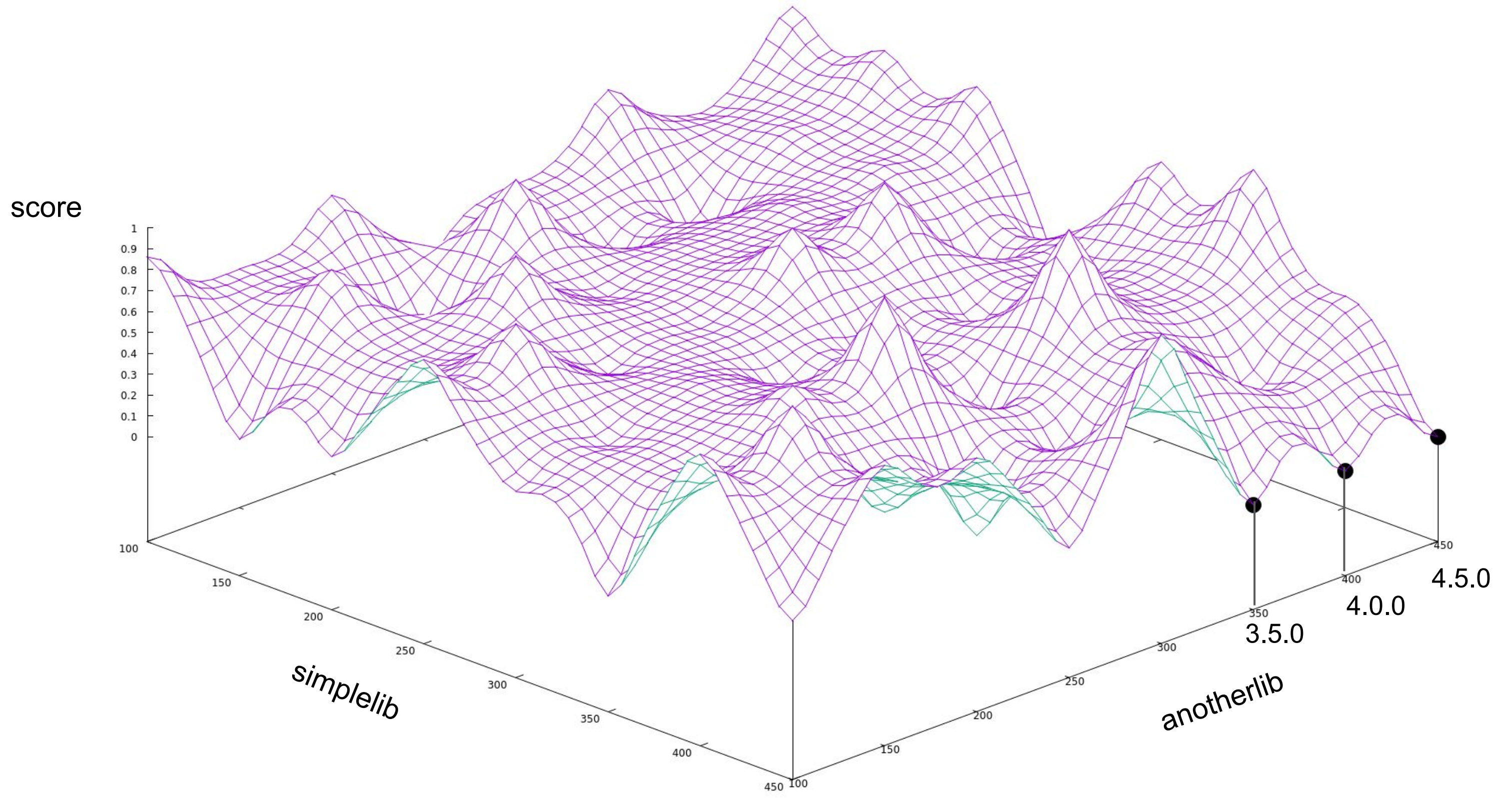
Scoring function - 3D



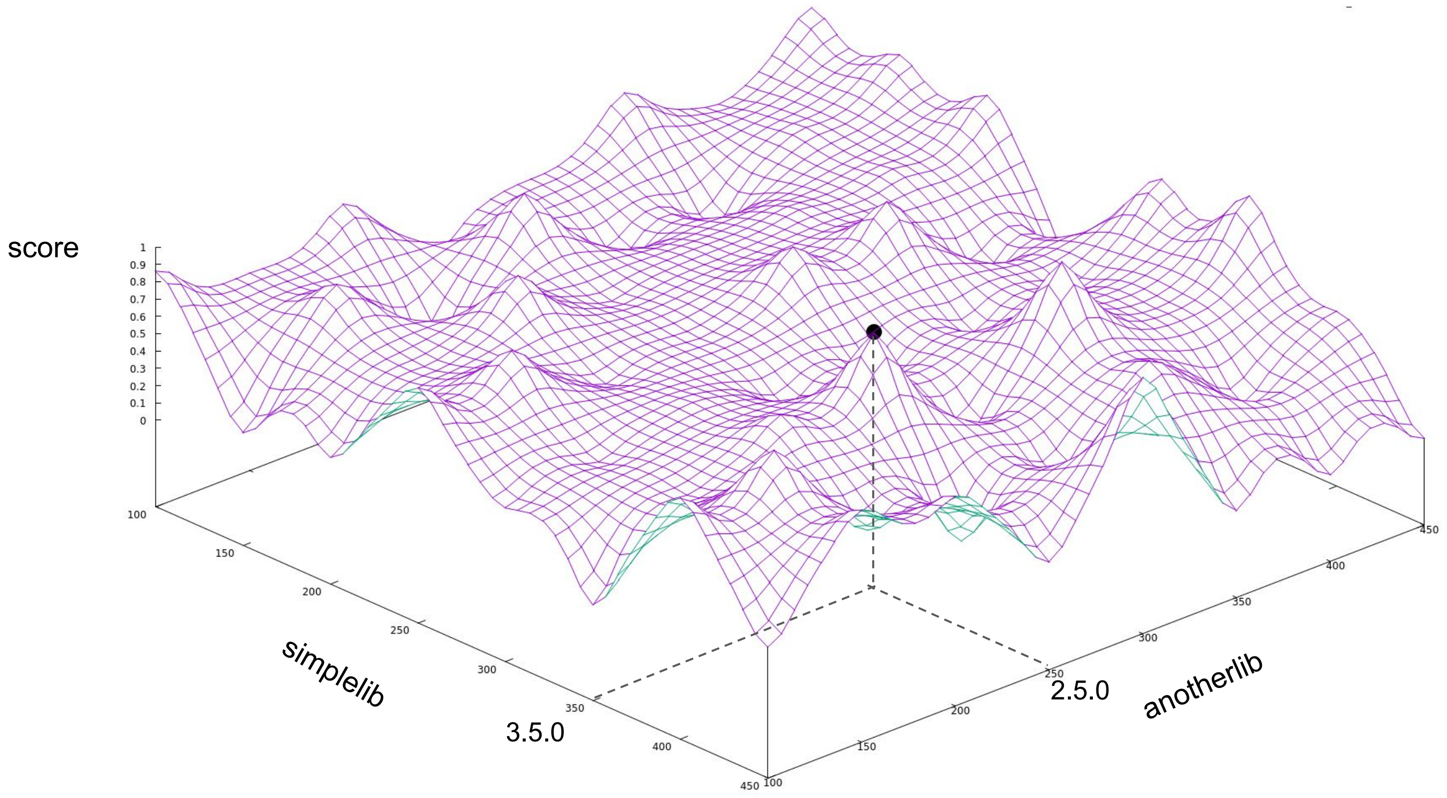
Scoring function - 3D



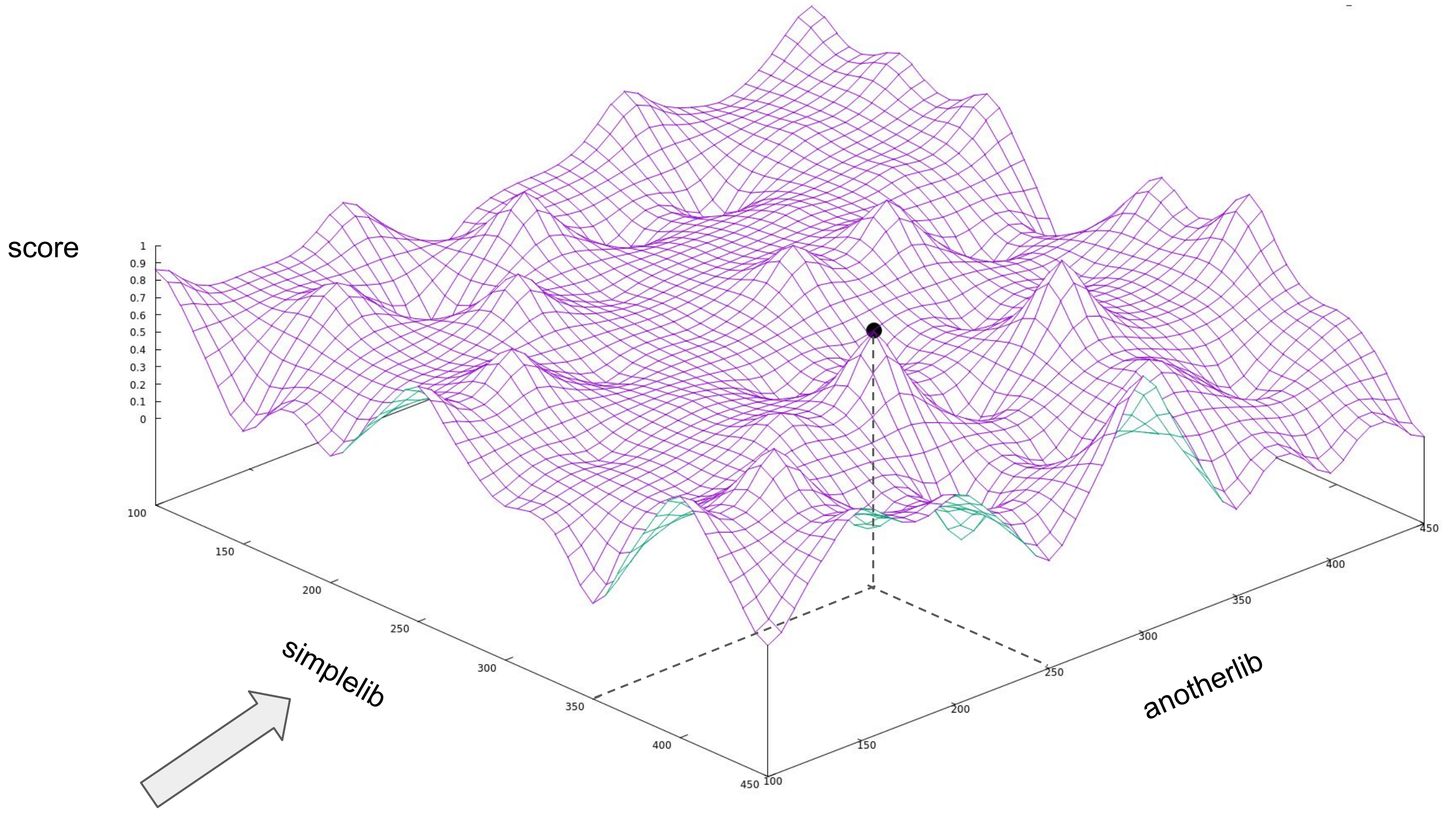
Scoring function - 3D



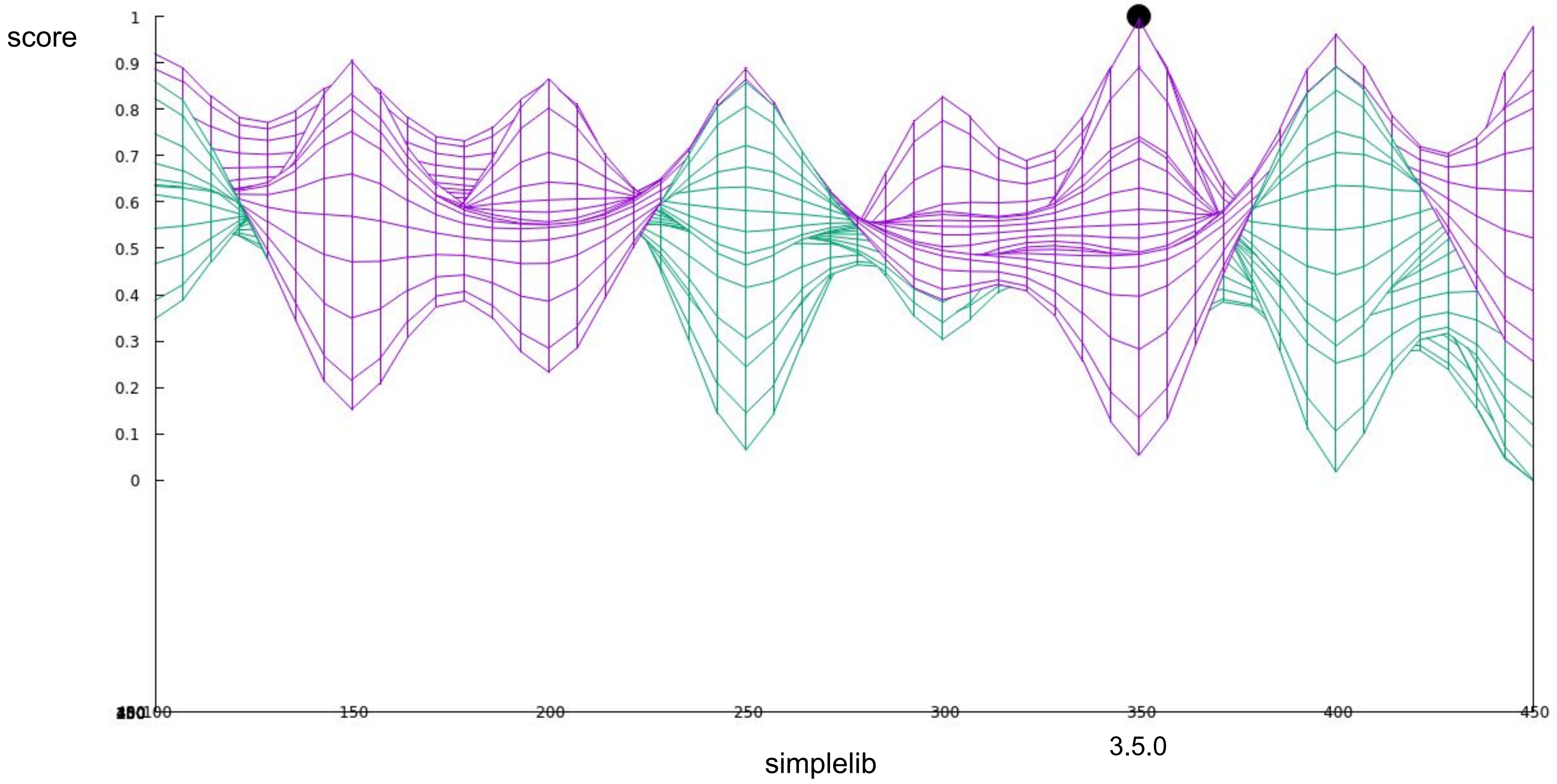
Scoring function - 3D



Scoring function - 3D



Scoring function - 3D



Scoring function - nD

- n = number of all packages possible in my software stack + 1 (score)
- Scoring - let’s find a scoring function for my application
  - How does it perform when it comes to performance?
  - How reliable are libraries?
  - How do libraries work together?
  - Do libraries have CVEs?
  - …

Knowledge

- Scoring
  - Package-level score
  - Cross-library score

How to find the best software stack?

- Nearly impossible - number of software stacks
    - Number of stacks with TensorFlow - 68+ bilion possible fully pinned down software stacks
- However we can find “a good enough” software stack

Thoth’s recommendation engine

- Resolve software stacks
- Score candidates based on knowledge
- Provide recommendation with reasoning
- Be close to source
  - Bots directly operate on source code
  - A user reviewes pull requests
  - If the pull request was closed => learn from it
  - Learn from opened pull requests and failures in CI
- Be proactive on changes around us
  - We monitor repositories
  - We proactively issue pull requests before something bad happens
- How to run your application

What does Thoth provide as of now?

- Automatic updates of your dependencies
    - Gathering information about software from logs
    - Does this update break your application
    - Log analyses
- Server side resolution
    - Finding package-level issues and excluding the given package from software stack
- Bots operating on source code
    - Automatic initial locks for your application
    - Automatic re-locking in case of issues
- Check provinence of your libraries
    - Am I really using well optimized TensorFlow on deployment?
    - Are packages installed from the configured source?
- Check your runtime environment

What is next?

- Aggregate more information about software stacks
  - Analyses of logs
- Add cross-library scoring
- Dependency Monkey
  - We already have dependency solver
  - Find out good TensorFlow configuration based on libraries and hardware used

Questions?

“Define future otherwise future defines you!”

Thanks for your attention!

https://github.com/thoth-station