ideas

April 29, 2022

```
[32]: import pypi_xmlrpc
      import requests
```

```
[3]: url = 'https://pypi.org/pypi/{}/json'
     packages = pypi_xmlrpc.list_packages()
```

```
[9]: package_info = requests.get(url.format('requests')).json()
```

```
[31]: print("requests:")
      print(json.dumps(package_info['releases']['0.10.0'][0], indent=4,␣
       ↪sort_keys=True))
```

```
requests:
{
    "comment_text": "",
    "digests": {
        "md5": "c90a48af18eb4170dbe4832c1104440c",
        "sha256":
"210a82e678c45d433a4ad1f105974b3102a8ab5198872dc0a3238a8750d4c65e"
    },
    "downloads": -1,
    "filename": "requests-0.10.0.tar.gz",
    "has_sig": false,
    "md5_digest": "c90a48af18eb4170dbe4832c1104440c",
    "packagetype": "sdist",
    "python_version": "source",
    "requires_python": null,
    "size": 62046,
    "upload_time": "2012-01-22T05:08:17",
    "upload_time_iso_8601": "2012-01-22T05:08:17.091441Z",
    "url": "https://files.pythonhosted.org/packages/62/35/0230421b8c4efad6624518
028163329ad0c2df9e58e6b3bee013427bf8f6/requests-0.10.0.tar.gz",
    "yanked": false,
    "yanked_reason": null
}
```

IDEA For packege in all packages:

```
1 get json
```

```
2 find the **first** release version with a date and save date
3 open a file in the format 'year-month-network.csv' representing the
    year and moth the package was created(grouping packages together
    that are released the same moth). If allready open do nothing
4 write in file repository|[dependencies...]
```

END; 5 : close all files.

Problem When making this kind of time-dependent network where we add nodes the dependencies may not yet exist, i.e. the development team decided to add dependencies after the first version. I have no way of checking the dependencies of each version

Addressing the problem*

The problem where the requirement of a package has not been released yet lets say we have a dictonary of pd.DataFrames, where each key corresponds to a year-date sorted based on the date of the data.

FOR each entry in the dictonary do: FOR each entry (package, requirement) in the dataframe do:

```
    1 check if the ***requirement*** is in any of the dataframes from before as package
    2 if yes : pass
    3 if not : delete the requirement and create a standalone package(node), additionally
        save both package and requirement together (as a tuple) in a list, say the cache list:
    4 check if the package is in the cache list as a ***requirement***
    5 if yes: append the pair package requirement found in the cache list to the
        dataframe and delete entry in the cache list
    6 if not: pass
    7 lastly update the dataframe by making a ***set*** out of it to avoid duplicate nodes, e.
        if requirement deleted (use pd.DataFrame.drop_duplicates())
```

```
DONE
```

DONE