Exercises chapter 17: Working with APIs

Saul SL

June 2023

1 17-1. Other Languages

Modify the API call in `python_repos.py` so it generates a chart showing the most popular projects in other languages. Try languages such as JavaScript, Ruby, C, Java, Perl, Haskell, and Go.

```python
import os
import requests

from plotly.graph_objs import Bar
from plotly import offline

language = input('Type the language you want to analyze:\n')
# language = 'r'
language = language.lower()
# Make an API call and store the response.
url = f'https://api.github.com/search/repositories?q=language:{language}&sort=stars'
headers = {'Accept': 'application/vnd.github.v3+json'}
r = requests.get(url, headers=headers)
status_code = r.status_code
if status_code == 200:

    # Store API response in a variable.
    response_dict = r.json()

    print(f"Total repositories: {response_dict['total_count']}")

    # Explore information about the repositories.
    repo_dicts = response_dict['items']
    print(f"Repositories returned: {len(repo_dicts)}")
    # Does it always returns 30?

    # PLot a graph
    repo_links, stars, labels = [], [], []
    for repo_dict in repo_dicts:
        repo_name = repo_dict['name']
        repo_url = repo_dict['html_url']
        repo_link = f"<a href='{repo_url}'>{repo_name}</a>"
        repo_links.append(repo_link)

        stars.append(repo_dict['stargazers_count'])
        owner = repo_dict['owner']['login']
        description = repo_dict['description']
        label = f"{owner}<br />{description}"
        labels.append(label)

    data = [{
        'type': 'bar',
        'x': repo_links,
        'y': stars,
        'hovertext': labels,
        'marker': {
```

```
            'color': 'rgb(250, 50, 20)',  # Red tone
            'line': {
                'width': 1.5,
                'color': 'rgb(25, 25, 25)'
            }  # Grey line
        },
        'opacity': 0.8,
    }]

    my_layout = {
        'title': f'Most-Starred {language.title()} Projects on GitHub',
        'titlefont': {
            'size': 28
        },
        'xaxis': {
            'title': 'Repository',
            'titlefont': {
                'size': 24
            },
            'tickfont': {
                'size': 14
            },
        },
        'yaxis': {
            'title': 'Stars',
            'titlefont': {
                'size': 24
            },
            'tickfont': {
                'size': 14
            },
        },
    }
    fig = {'data': data, 'layout': my_layout}
    icounter = 1
    out_file = f'{language}_repos_v{icounter}'
    while os.path.exists(out_file):
        icounter += 1
        out_file = f'{language}_repos_v{icounter}'

    if not os.path.exists("images"):
        os.mkdir("images")

    # Plot and save file as png
    # offline.plot(fig, filename=out_file)
    offline.plot(fig,
                 auto_open=True, image='png', image_filename=out_file,
                 output_type='file', image_width=1600, image_height=900,
                 filename=out_file+'.html', validate=False)
```

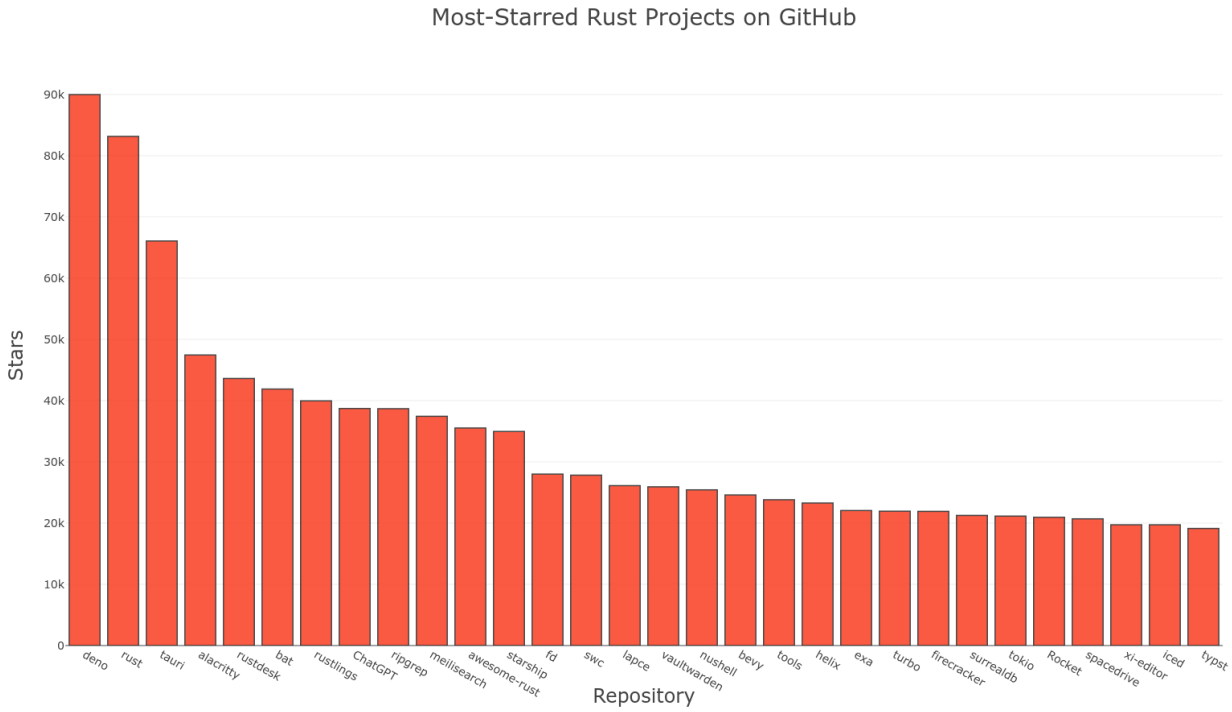

Figure 1: Top GitHub repositories (based on number of stars) for Rust language

2 17-2. Active Discussions

Using the data from hn_submissions.py, make a bar chart showing the most active discussions currently happening on Hacker News. The height of each bar should correspond to the number of comments each submission has. The label for each bar should include the submission's title and should act as a link to the discussion page for that submission.

```
import os
import requests
from operator import itemgetter
from plotly.graph_objs import Bar
from plotly import offline

# Make an API call and store the response.
url = 'https://hacker-news.firebaseio.com/v0/topstories.json'
r = requests.get(url)
# print(f"Status code: {r.status_code}")
if r.status_code == 200:
    # Process information about each submission.
    submission_ids = r.json()
    submission_dicts = []
    for submission_id in submission_ids[:30]:
        # Make a separate API call for each submission.

        url = f"https://hacker-news.firebaseio.com/v0/item/{submission_id}.json"
        r = requests.get(url)
        if r.status_code != 200:
            print(f"Failed for id: {submission_id}\tstatus: {r.status_code}")
        else:
            print('.', end='')
        print(' ')

        response_dict = r.json()

        # Build a dictionary for each article.
        submission_dict = {
            'title': response_dict['title'],
            'hn_link': f"http://news.ycombinator.com/item?id={submission_id}",
            'comments': response_dict['descendants'],
        }
        submission_dicts.append(submission_dict)

    # Sort by number of comments
    submission_dicts = sorted(submission_dicts,
                              key=itemgetter('comments'),
                              reverse=True)

    links, icounts, ilabels = [], [], []
    for i in range(0, len(submission_dicts)-1):
        ititle = submission_dicts[i]['title']
        if len(ititle) > 27:
            ititle_sub = ititle[:27] + '...'
        else:
            ititle_sub = ititle
        hn_link = submission_dicts[i]['hn_link']
        comments = submission_dicts[i]['comments']
        links.append(f"<a href='{hn_link}'>{ititle_sub}</a>")
        icounts.append(comments)
        ilabels.append(ititle)

    data = [{
        'type': 'bar',
        'x': links,
```

```
        'y': icounts,
        'hovertext': ilabels,
        'marker': {
            'color': 'rgb(250, 50, 20)',  # Red tone
            'line': {
                'width': 1.5,
                'color': 'rgb(25, 25, 25)'  # Grey line
            }
        },
        'opacity': 0.8,
    }]

    my_layout = {
        'title': 'Most-Comments on Hacker News',
        'titlefont': {
            'size': 28
        },
        'xaxis': {
            'title': 'News',
            'titlefont': {
                'size': 24
            },
            'tickfont': {
                'size': 14
            },
        },
        'yaxis': {
            'title': 'Comments',
            'titlefont': {
                'size': 24
            },
            'tickfont': {
                'size': 14
            },
        },
    }
    fig = {'data': data, 'layout': my_layout}
    icounter = 1
    out_file = f'Hacker_news_v{icounter}'
    if not os.path.exists("images"):
        os.mkdir("images")

    while os.path.exists(out_file):
        icounter += 1
        out_file = f'Hacker_news_v{icounter}'

    # Plot and save file as png
    offline.plot(fig,
                 auto_open=True, image='png', image_filename=out_file,
                 output_type='file', image_width=1600, image_height=900,
                 filename=out_file+'.html', validate=False)
    # offline.plot(fig, filename=out_file)
```

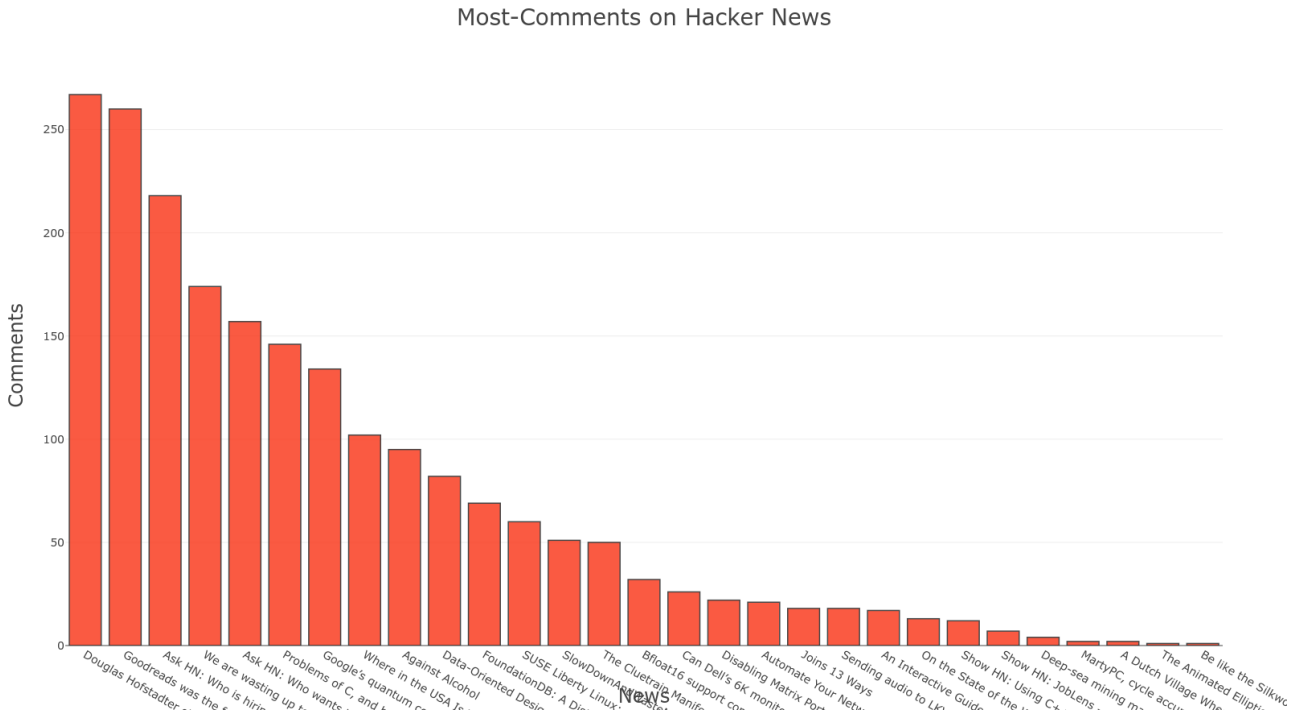

Figure 2: Top active discussions on Hacker News (based on the number of comments)

3 17-3. Testing python_repos.py

In `python_repos.py`, we printed the value of `status_code` to make sure the API call was successful. Write a program called `test_python_repos.py` that uses unittest to assert that the value of `status_code` is 200. Figure out some other assertions you can make—for example, that the number of items returned is expected and that the total number of repositories is greater than a certain amount.

3.1 Define functions to query GitHub

```
import requests

url_prefix = 'https://api.github.com/search/repositories?q=language:'
url_suffix = '&sort=stars'
headers = {'Accept': 'application/vnd.github.v3+json'}

def get_github_status(language):
    """Returns the status code of a API call to github for
    for repositories on a particular language
    """
    language = language.lower()
    url = url_prefix+language+url_suffix
    r = requests.get(url, headers=headers)
    return r.status_code

def query_github_repos(language):
    """search GitHub for repositories on a particular language
    if the query is successful (Status 200) returns a dictionary
    """
    repo_status = get_github_status(language)
    if repo_status == 200:
        language = language.lower()
        url = url_prefix+language+url_suffix
        r = requests.get(url, headers=headers)
        # Store API response in a variable.
        response_dict = r.json()
        return response_dict
```

3.2 Test functions

```
import unittest
from search_github_repos import get_github_status, query_github_repos


class GitHubQueryTestCase(unittest.TestCase):
    """Test an API call to GitHub
    """
    def test_get_github_status(self):
        istatus = get_github_status('python')
        self.assertEqual(istatus, 200)

    def test_n_items(self):
        idict = query_github_repos('python')
        n_items = len(idict['items'])
        n_repos = idict['total_count']
        self.assertGreater(n_items, 29)
        self.assertGreater(n_repos, 1_000_000)
```

```
if __name__ == '__main__':
    unittest.main()
```

4 17-4. Further Exploration

Visit the documentation for Plotly and either the GitHub API or the Hacker News API. Use some of the information you find there to either customize the style of the plots we've already made or pull some different information and create your own visualizations.

```
import os
import plotly.express as px
# from plotly import offline
import pandas as pd
import search_github_repos as sgh

language = 'python'
python_repos = sgh.query_github_repos(language)
repo_dicts = python_repos['items']

repo_links, stars, descriptions, names = [], [], [], []
for repo_dict in repo_dicts:
    repo_name = repo_dict['name']
    repo_url = repo_dict['html_url']
    repo_link = f"<a href='{repo_url}'>{repo_name}</a>"
    repo_links.append(repo_link)

    stars.append(repo_dict['stargazers_count'])
    # owner = repo_dict['owner']['login']
    description = repo_dict['description']
    # label = f"{owner}<br />{description}"
    descriptions.append(description)
    names.append(repo_name)

# Create a pandas dataframe
data = {'Repository': names, 'N stars': stars, 'Description': descriptions}
df = pd.DataFrame(data)
df_inv = df.sort_values(by='N stars')


# Create a plotly object
fig = px.bar(df_inv, y='Repository', x='N stars',
             orientation='h', text='N stars',
             hover_name='Description',
             color='N stars', color_continuous_scale='Burg',
             title=f"Most-Starred {language.title()} Projects on GitHub")

# Show figure
# fig.show()
icounter = 1
out_file = f'{language.title()}_repos_px_v{icounter}'
if not os.path.exists("images"):
    os.mkdir("images")

while os.path.exists(out_file):
    icounter += 1
    out_file = f'{language.title()}_repos_px_v{icounter}'

# Plot and save file as png
fig.write_image("./images/"+out_file+".png", width=1600, height=900)
```

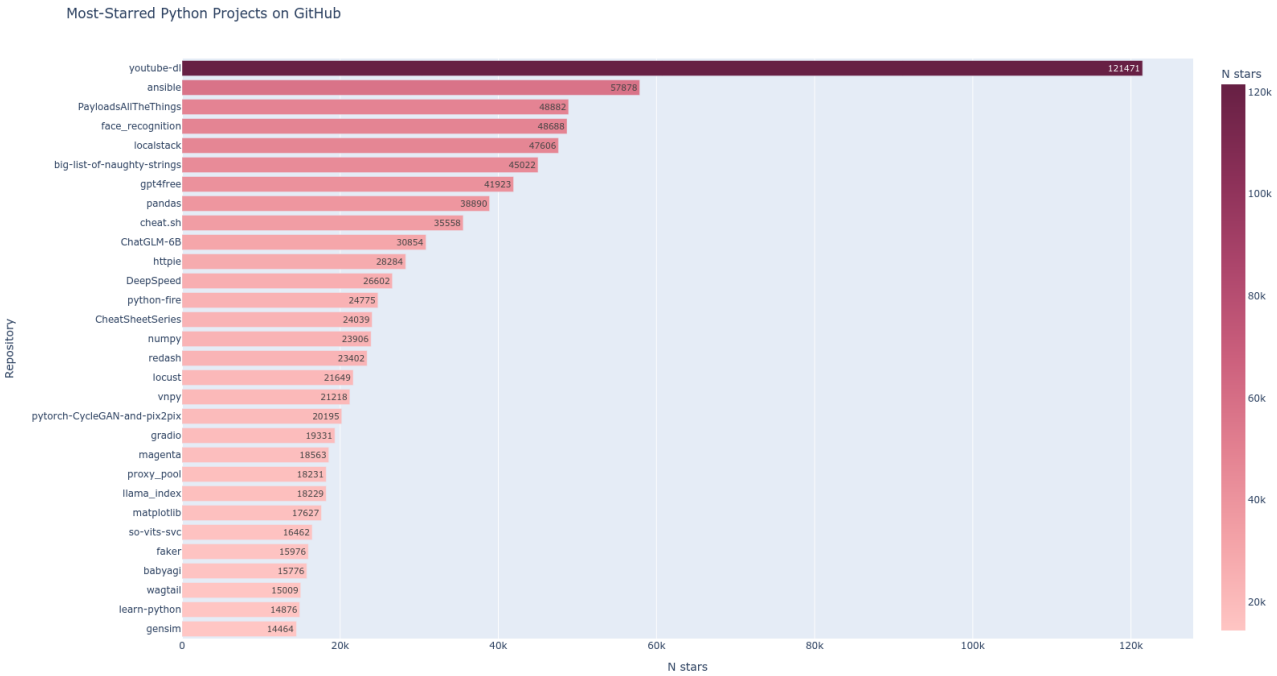

Figure 3: Top GitHub repositories (based on number of stars) for Python language.