Final-Project

December 7, 2021

API Websites Analysis (STA 141B - Final Project)

By Shih-Chi Chen

I. Introduction

- Use https://api.publicapis.org/entries api.
- Analyze if there exists some characteristics for some popular AIP websites and also analyze these websites' domain type and status code.
- Explore Questions:
  1. What categories of API are popular?
  2. What kinds of status code have high frequency in the dataset?
  3. For those API websites with status code of 200, what is the domain type of the most frequent occurrences?
  4. What are the failure rates by Category and by domain type?
  5. Which countries/regions tend to create more API websites?
  6. What is the relationship among variables?

API website is useful for many people to directly solve programming problems, and to save time. Programmer often use APIs to perform complex tasks easily in programming. APIs make our lives easier since they provide us proper access to data. So it is meaningful to analyze some characteristics on those API websites. Additionally, it is also useful to figure out status codes of those API websites before we use them.

In my project, I use two datasets. The first dataset is more than 1000 API websites from publicapis.org website. Then it is transformed to the pandas dataframe for analysis. It has 7+2 columns.

On the other hands, in order to in-dep analyze these AIP websites, I request from IP Location Finder website(https://tools.keycdn.com/) and obtain the detail infromation for each API webstie from publicapis.org. Then this dataset is transformed to the pandas dataframe for analysis. It has 8 columns.

II. Data Setting

Data one

[9]:
```
import requests
import pandas as pd
import requests_cache
import plotnine as p9
```

```
import sqlalchemy as sqla
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
requests_cache.install_cache('final_cache')
from urllib.parse import urlparse
from requests.structures import CaseInsensitiveDict
```

[3]:
```
#Data one
#use API and parse the JSON then form the dataframe
link_url = "https://api.publicapis.org/entries"
req_one = requests.get(link_url)
js = req_one.json()
js_data=js['entries']
df_onenormal = pd.json_normalize(js_data)
df_onenormal
```

[3]:
```
                API                                        Description  \
0          AdoptAPet                  Resource to help get pets adopted
1            Axolotl           Collection of axolotl pictures and facts
2          Cat Facts                                  Daily cat facts
3             Cataas          Cat as a service (cats pictures and gifs)
4             catAPI                              Random pictures of cats
…                ...                                              …
1396     weather-api            A RESTful free API to check the weather
1397      WeatherAPI  Weather API with other stuff like Astronomy an…
1398      Weatherbit                                            Weather
1399   Yahoo Weather                          Weather API from Yahoo
1400  Yandex.Weather   Assesses weather condition in specific locations

        Auth  HTTPS     Cors  \
0     apiKey   True      yes
1              True      yes
2              True       no
3              True       no
4              True      yes
…       ...    ...       …
1396           True       no
1397  apiKey   True      yes
1398  apiKey   True  unknown
1399   OAuth   True  unknown
1400  apiKey   True       no

                                                 Link Category
0     https://www.adoptapet.com/public/apis/pet_list…  Animals
1                   https://theaxolotlapi.netlify.app/  Animals
2           https://alexwohlbruck.github.io/cat-facts/  Animals
```

```
3                                    https://cataas.com/  Animals
4                               https://thatcopy.pw/catapi  Animals
…                                                      …      …
1396    https://github.com/robertoduessmann/weather-api  Weather
1397                           https://www.weatherapi.com/  Weather
1398                         https://www.weatherbit.io/api  Weather
1399                 https://developer.yahoo.com/weather/  Weather
1400                      https://yandex.com/dev/weather/  Weather

[1401 rows x 7 columns]
```

Columns information

1. API - name of entry
2. Description - description of entry
3. Auth - auth type of entry
4. HTTPS - return entries that support HTTPS or not
5. Cors - CORS support for entry ("yes", "no", or "unknown")
6. Link - API website link address
7. Category - return entries of a specific category

Add two more colums (status_code and domain)

In order to analyze these websites' domain type and status code. I create two colums for these two variables.

[5]:
```
#function of status code
def testurl(url):
    """
    Input: url
    Output: request status code
    """
    try:
        if url[:4] == 'http':
            r = requests.head(url).status_code
            return r
    except requests.ConnectionError:
        return None

#reate new column and calculate status code for all links
df_onenormal['status_codes']=df_onenormal.apply(lambda row:␣
 ↪testurl(row['Link']), axis=1)
```

[6]:
```
#functino of domain type
def extract_domain_type(link):
    return urlparse(link).netloc.split(".")[-1]

#create new column for domain
```

```
df_onenormal['domain']=df_onenormal.apply(lambda row:␣
 ↪extract_domain_type(row['Link']), axis=1)
df_onenormal.to_csv('df_onenormal')
```

Data two

[10]:
```
#Data two
def extract_domain(link):
    return ".".join(urlparse(link).netloc.split(".")[-2:])

data_two=pd.DataFrame()
for index, row in df_onenormal.iterrows():
    domain=extract_domain(row['Link'])
    linkme=row['Link']
    #print(domain)
    #now = datetime.now()
    #current_time = now.strftime("%H:%M:%S.%f")
    headers = CaseInsensitiveDict()
    headers["User-Agent"] = f"keycdn-tools:{linkme}"
    try:
        chreq_one = requests.get(f"https://tools.keycdn.com/geo.json?
 ↪host={domain}",headers=headers)
        getjson=chreq_one.json()
        df_twonormal = pd.json_normalize(getjson)
        data_two=pd.concat([data_two,df_twonormal])
    except :
        pass

data_two
```

[10]:

	status	description	data.geo.host	data.geo.ip	\
0	success	Data successfully received.	adoptapet.com	34.199.117.187	
0	success	Data successfully received.	netlify.app	161.35.218.98	
0	success	Data successfully received.	github.io	185.199.109.153	
0	success	Data successfully received.	cataas.com	37.187.12.73	
0	success	Data successfully received.	thatcopy.pw	172.67.186.215	
..	…	…	…	…	
0	success	Data successfully received.	github.com	140.82.121.4	
0	success	Data successfully received.	weatherapi.com	185.249.71.83	
0	success	Data successfully received.	weatherbit.io	192.155.89.79	
0	success	Data successfully received.	yahoo.com	74.6.231.21	
0	success	Data successfully received.	yandex.com	77.88.55.77	

	data.geo.rdns	data.geo.asn	\
0	ec2-34-199-117-187.compute-1.amazonaws.com	14618.0	
0	161.35.218.98	14061.0	
0	cdn-185-199-109-153.github.com	54113.0	

```
0                        ns3110573.ip-37-187-12.eu       16276.0
0                                  172.67.186.215       13335.0
..                                            …           …
0                  lb-140-82-121-4-fra.github.com       36459.0
0                                   185.249.71.83      204413.0
0                     li577-79.members.linode.com       63949.0
0   media-router-fp74.prod.media.vip.ne1.yahoo.com       36646.0
0                                        yandex.ru       13238.0

        data.geo.isp data.geo.country_name data.geo.country_code  \
0          AMAZON-AES         United States                    US
0   DIGITALOCEAN-ASN               Germany                    DE
0            FASTLY         United States                    US
0          OVH SAS                France                    FR
0      CLOUDFLARENET         United States                    US
..               …                   …                     …
0             GITHUB         United States                    US
0          Hyve Ltd        United Kingdom                    GB
0        Linode, LLC         United States                    US
0          YAHOO-NE1         United States                    US
0         YANDEX LLC                Russia                    RU

   data.geo.region_name data.geo.region_code      data.geo.city  \
0              Virginia                   VA             Ashburn
0                 Hesse                   HE  Frankfurt am Main
0            California                   CA      San Francisco
0                  None                 None               None
0                  None                 None               None
..                  …                    …                  …
0                  None                 None               None
0                  None                 None               None
0             New Jersey                   NJ       Cedar Knolls
0               Nebraska                   NE              Omaha
0                  None                 None               None

   data.geo.postal_code data.geo.continent_name data.geo.continent_code  \
0                 20149          North America                      NA
0                 60313                 Europe                      EU
0                 94107          North America                      NA
0                  None                 Europe                      EU
0                  None          North America                      NA
..                  …                    …                       …
0                  None          North America                      NA
0                  None                 Europe                      EU
0                 07927          North America                      NA
0                 68197          North America                      NA
0                  None                 Europe                      EU
```

```
   data.geo.latitude data.geo.longitude data.geo.metro_code  \
0            39.0469           -77.4903                 511
0            50.1188             8.6843                None
0            37.7642          -122.3993                 807
0            48.8582             2.3387                None
0             37.751            -97.822                None
..               …                 …                   …
0             37.751            -97.822                None
0            51.4964            -0.1224                None
0            40.8229           -74.4592                 501
0            41.2612           -95.9354                 652
0            55.7386            37.6068                None

      data.geo.timezone    data.geo.datetime
0      America/New_York  2021-12-01 20:46:42
0         Europe/Berlin  2021-12-02 02:46:43
0   America/Los_Angeles  2021-12-01 17:46:44
0          Europe/Paris  2021-12-02 02:46:45
0       America/Chicago  2021-12-01 19:46:46
..                  …                   …
0       America/Chicago  2021-12-01 19:49:04
0         Europe/London  2021-12-02 16:43:56
0      America/New_York  2021-12-02 11:43:57
0       America/Chicago  2021-12-02 10:43:59
0         Europe/Moscow  2021-12-02 19:29:25

[1392 rows x 20 columns]
```

[28]:
```
df_need_cols = ['data.geo.host','data.geo.ip','data.geo.country_name','data.geo.
 ↪region_name','data.geo.country_code','data.geo.region_code','data.geo.
 ↪latitude','data.geo.longitude']
data_two_re = data_two[df_need_cols].copy()
data_two_re.rename(columns={'data.geo.host': 'host', 'data.geo.ip': 'ip','data.
 ↪geo.country_name':'country_name','data.geo.country_code':
 ↪'country_code','data.geo.region_name':'region_name','data.geo.region_code':
 ↪'region_code','data.geo.latitude':'latitude','data.geo.longitude':
 ↪'longitude'}, inplace=True)
data_two_re
```

[28]:
```
              host               ip    country_name region_name country_code  \
0    adoptapet.com   34.199.117.187   United States    Virginia           US
0      netlify.app    161.35.218.98         Germany       Hesse           DE
0        github.io  185.199.109.153   United States  California           US
0       cataas.com     37.187.12.73          France        None           FR
0      thatcopy.pw   172.67.186.215   United States        None           US
..             …                 …               …           …           …
```

```
0       github.com     140.82.121.4   United States        None           US
0   weatherapi.com    185.249.71.83  United Kingdom        None           GB
0    weatherbit.io    192.155.89.79   United States  New Jersey           US
0        yahoo.com      74.6.231.21   United States    Nebraska           US
0       yandex.com      77.88.55.77          Russia        None           RU

   region_code latitude longitude
0           VA  39.0469  -77.4903
0           HE  50.1188    8.6843
0           CA  37.7642 -122.3993
0         None  48.8582    2.3387
0         None   37.751   -97.822
..         …      …       …
0         None   37.751   -97.822
0         None  51.4964   -0.1224
0           NJ  40.8229  -74.4592
0           NE  41.2612  -95.9354
0         None  55.7386   37.6068

[1392 rows x 8 columns]
```

Columns information

1. host - url of website
2. ip - ip of website
3. country_name - created country of website.
4. region_name - created region of website.
5. country_code - created country code of website.
6. region_code- created region code of website.
7. longitude - created region longitude of website.
8. latitude- - created region latitude of website.

II. Data Analysis - VIsualization

Category Pie chart

For those categories covering less than 1%, I group them up to one cateogry (Other).

Therefre, the pie chart shows that the top three popular API categories are Development, Games & Comics and Goverment except Other cateogry.

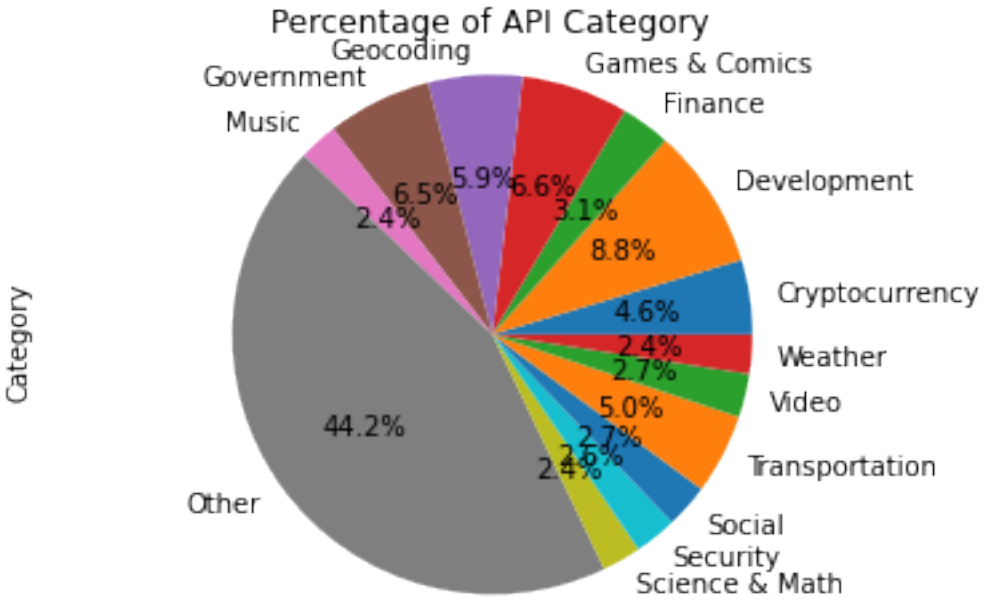
[18]:
```
#pie chart

def group_lower_ranking_values(column):
    pie_counts = df_onenormal.groupby(column).agg('count')
    pct_value = pie_counts[lambda df: df.columns[0]].quantile(.75)
    values_below_pct_value = pie_counts[lambda df: df.columns[0]].loc[lambda s:␣
 ↪s < pct_value].index.values
    def fix_values(row):
```

```
        if row[column] in values_below_pct_value:
            row[column] = 'Other'
        return row
    pie_grouped = df_onenormal.apply(fix_values, axis=1).groupby(column).
↪agg('count')
    return pie_grouped

new_df=group_lower_ranking_values('Category')

new_df.API.plot(kind='pie',ylabel="Category", autopct='%1.1f%%')
plt.axis('equal')
plt.title('Percentage of API Category')
plt.show()
```



Category and dommain Histgram

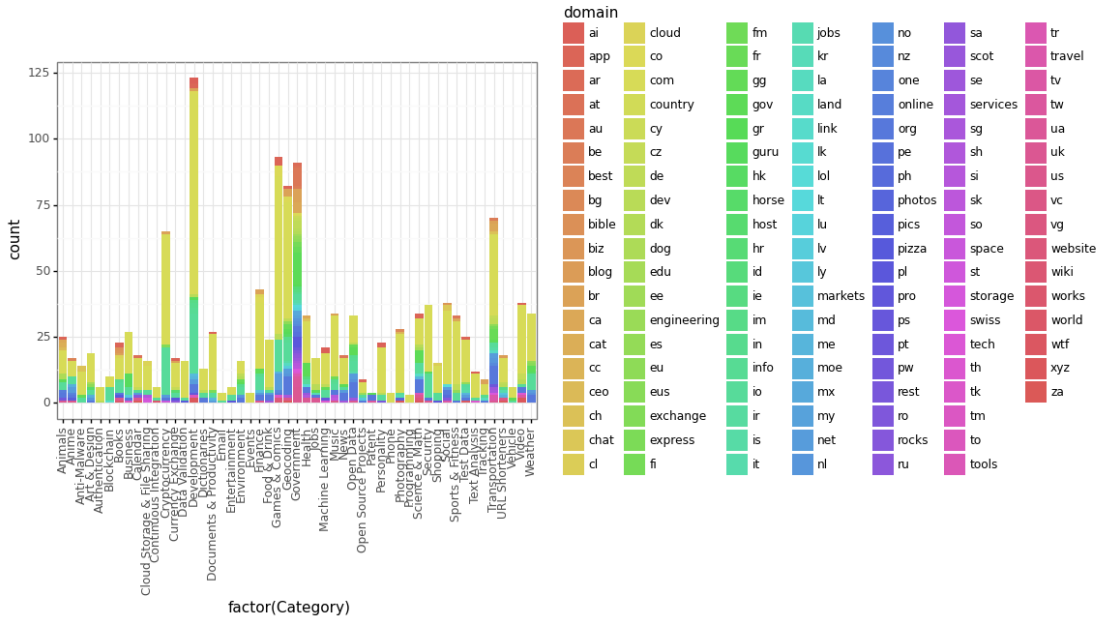
After adding domain in to the histgram, it shows that most of API website domains are com.

In addition, domain gov is the most frequent domain type for Government category API website.

[19]:
```
#histgram of Category and dommain

(p9.ggplot(data=df_onenormal,
           mapping=p9.aes(x='factor(Category)',fill = 'domain'))
    + p9.geom_bar()
    + p9.theme_bw()
    + p9.theme(axis_text_x = p9.element_text(angle=90))
```

```
)
```



[19]:
```
<ggplot: (8792090221300)>
```

Status Codes Plot

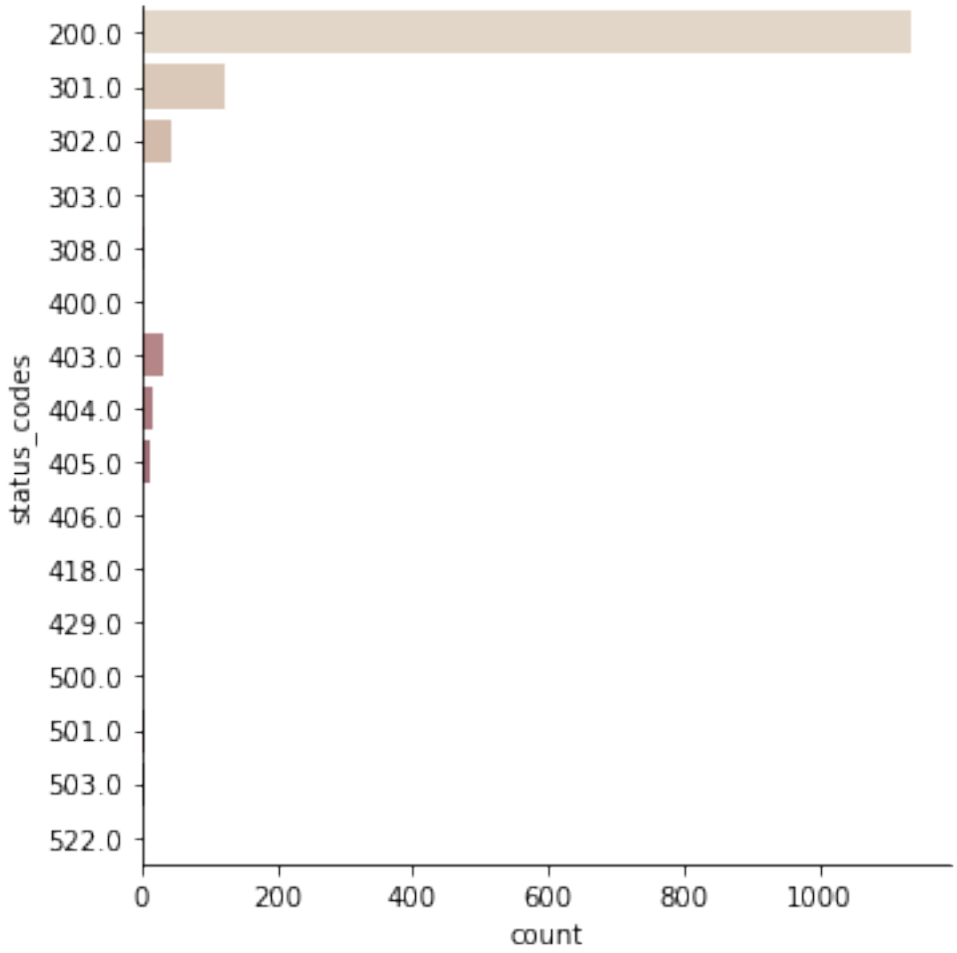
The plot shows that the most frequent status code is 200 in those API websites.

Therefore, most API websites are good for people to use.

However, there are still around 100 API websites with status code of 301, which indicates that the requested resource has been permanently moved to a new URL.

[20]:
```
#Status Codes plot
sns.catplot(y="status_codes", kind="count", palette="ch:.25", data=df_onenormal)
```

[20]:
```
<seaborn.axisgrid.FacetGrid at 0x7ff116b1c850>
```

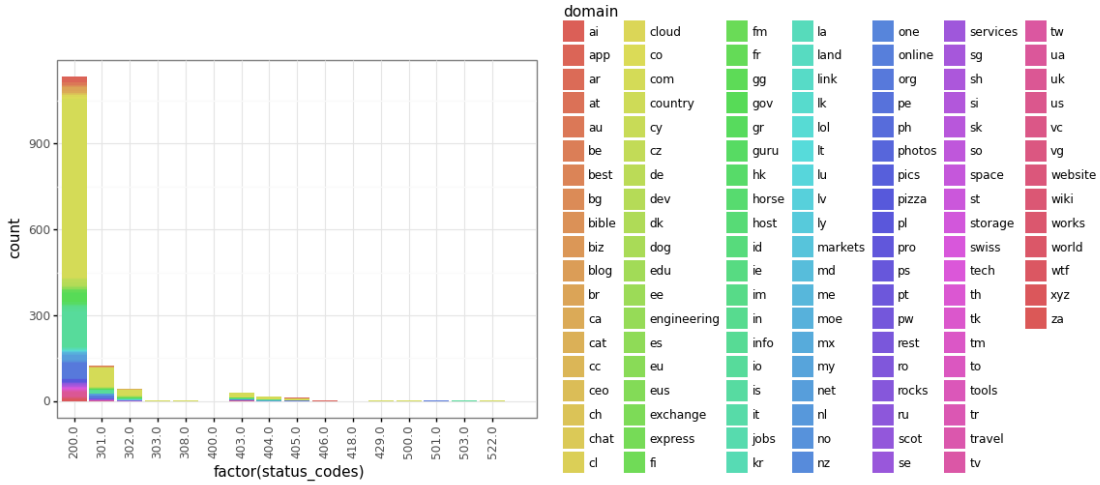


Status_codes and domain Histgram

From the histgram, it is found that most of the good API websites domain are com and the second one is gov.

[21]:
```
(p9.ggplot(data=df_onenormal,
           mapping=p9.aes(x='factor(status_codes)',fill = 'domain'))
    + p9.geom_bar()
    + p9.theme_bw()
    + p9.theme(axis_text_x = p9.element_text(angle=90))
)
```

```
/Users/chang/opt/anaconda3/lib/python3.8/site-packages/plotnine/layer.py:324:
PlotnineWarning: stat_count : Removed 15 rows containing non-finite values.
```



[21]: <ggplot: (8792090866173)>

Country / Region Websites Interactive Map

Please look at the map on juypter since it is an interactive map

From the map, it is found that most of the API websites are located at USA. And the second most of API websites are located at Germany.

By zooming up the map, it shows that WA and NY States have the most of API websites in the USA.

[17]:
```
#change longitude and latitude coulumns type to numeric
data_two_re['longitude']=pd.to_numeric(data_two_re.longitude,errors="coerce")
data_two_re['latitude']=pd.to_numeric(data_two_re.latitude,errors="coerce")

data_two_nona=data_two_re.dropna()
# Create a world map to show distributions of API websites
import folium
from folium.plugins import MarkerCluster
#empty map
world_map= folium.Map(tiles="cartodbpositron")
marker_cluster = MarkerCluster().add_to(world_map)
#for each coordinate, create circlemarker of host name
for i in range(len(data_two_nona)):
        lat = data_two_nona.iloc[i]['latitude']
        long = data_two_nona.iloc[i]['longitude']
        radius=5
        popup_text = """host : {} """
        popup_text = popup_text.format(data_two_nona.iloc[i]['host'])
```

```
        folium.CircleMarker(location = [lat, long], radius=radius, popup=␣
 ↪popup_text, fill =True).add_to(marker_cluster)
#show the map

display(world_map)
```

```
<folium.folium.Map at 0x7ff115579be0>
```

III. Data Analysis - Statistics

Since graphs sometime can not provide us sufficient information, some statistical results of these datasets are essential.

Failure Rates by Category

According to the following result, it shows that top three fail rates categories are Events, Open Source Projects and Text Analysis . Especially, Events category has the more high fail rate (0.75) than other websites, which means that the user will have high chances to get errors on API webistes when they use Events category.

On the other hands, there are six categories has zero fail rates on their websites, which are Phone,Documents & Productivity, Entertainment, Authentication, and Vehicle. The satatus codes of these websites are all equal to 200, which is good for user to use.

[22]:
```
#group by category and statis_code then count it
df_failed = df_onenormal.groupby(['Category','status_codes'])
failed_count = df_failed.count()[['API']].rename(columns={'API':'count'})
failed_count = failed_count.reset_index()
failed_count

#use unstack bring the lowest level of column index to the lowest level of row␣
 ↪index
newdf=failed_count.set_index(['Category','status_codes'],drop=True).
 ↪unstack('status_codes')
newdf = newdf.fillna(0) #in order to calculate fail ratio , fill na to 0

#add one column for fail ratio
newdf['fail_ratio']=newdf.iloc[:, 1:].sum(axis=1)/newdf.sum(axis=1)
ascend_df=newdf.sort_values(by='fail_ratio',ascending=False)
#find the top 3 fail ratio categories
copy_dfcat=ascend_df.groupby('Category')['fail_ratio'].max().reset_index().
 ↪sort_values(['fail_ratio'], ascending=False)
print(copy_dfcat[0:3])

#show zero fail ratio categories
goodcat=np.extract(copy_dfcat.fail_ratio == 0, copy_dfcat.Category)
goodcat
```

```
               Category  fail_ratio
20               Events    0.750000
```

```
32  Open Source Projects    0.444444
44          Text Analysis    0.416667
```

[22]:
```
array(['Phone', 'Patent', 'Documents & Productivity', 'Entertainment',
       'Authentication', 'Vehicle'], dtype=object)
```

Failure Rates by domain

From the table, it shows that 11 domain types have fail rates of one, which means that the user should get errors on API webistes when they use these types of domain (lu, services, my, online, ph, ps, engineering, pw, ro, eu, be).

Although there are 11 types of domain have 100% fail rate, there are still many types of domain(80 types) with fail rate of zero. Therefore, most API websites in these datasets are good to use.

[23]:
```
#group by domain and statis_code then count it
df_domafailed = df_onenormal.groupby(['domain','status_codes'])
failedoma_count = df_domafailed.count()[['API']].rename(columns={'API':'count'})
failedoma_count = failedoma_count.reset_index()
failedoma_count

#use unstack bring the lowest level of column index to the lowest level of row index
newdf_doma=failedoma_count.set_index(['domain','status_codes'],drop=True).unstack('status_codes')
newdf_doma = newdf_doma.fillna(0) #in order to calculate fail ratio , fill na to 0

#add one column for fail ratio
newdf_doma['fail_ratio']=newdf_doma.iloc[:, 1:].sum(axis=1)/newdf_doma.sum(axis=1)
ascenddoma_df=newdf_doma.sort_values(by='fail_ratio',ascending=False)

#find fail ratio =1 domain
copy_dfdom=ascenddoma_df.groupby('domain')['fail_ratio'].max().reset_index().sort_values(['fail_ratio'], ascending=False)
print(copy_dfdom[copy_dfdom.fail_ratio == 1])

#calculate how many domain have zero fail ratio
b=ascenddoma_df[ascenddoma_df.fail_ratio == 0].shape[0]
print('There are '+str(b)+' types of domain have zero fail ratio')
#show zero fail ratio domain
gooddom=np.extract(copy_dfdom.fail_ratio == 0, copy_dfdom.domain)
gooddom
```

```
        domain  fail_ratio
63          lu         1.0
95    services         1.0
71          my         1.0
```

```
77       online         1.0
80           ph         1.0
86           ps         1.0
31  engineering         1.0
88           pw         1.0
90           ro         1.0
33           eu         1.0
5            be         1.0
There are 80 types of domain have zero fail ratio
```

[23]:
```
array(['sk', 'bg', 'si', 'scot', 'bible', 'biz', 'space', 'ru', 'rocks',
       'blog', 'rest', 'cc', 'so', 'tk', 'st', 'tw', 'xyz', 'wtf',
       'world', 'works', 'wiki', 'website', 'vg', 'vc', 'ar', 'at',
       'storage', 'au', 'tr', 'tools', 'to', 'tm', 'pl', 'best', 'tech',
       'swiss', 'pro', 'dk', 'pizza', 'gr', 'is', 'im', 'ie', 'id',
       'host', 'horse', 'hk', 'guru', 'gg', 'pics', 'cy', 'fm', 'express',
       'exchange', 'eus', 'es', 'ee', 'edu', 'jobs', 'kr', 'la',
       'country', 'photos', 'ceo', 'pe', 'dog', 'chat', 'cl', 'no',
       'cloud', 'mx', 'moe', 'md', 'markets', 'lv', 'lt', 'lol', 'lk',
       'link', 'za'], dtype=object)
```

In addition, for the most frequent using domain COM, it has a fail rate of 0.176627, which is not bad for users to use.

[24]:
```
#show com fail rate
c=copy_dfdom[copy_dfdom.domain == 'com']
c
```

[24]:
```
   domain  fail_ratio
21    com    0.176316
```

Chi Square Independent Test

In order to figure out the relationship among variables, Chi Square Independent Test is used.

Chi squate independent test for some categorical variables in Data one is demonstrated.

Here, we want to find if there exists relationship between status code and other categorical variables(domain, Category, Auth, HTTPS, and Cors).

The plot below shows that Cors and HTTPS have higher p-values; it means that these two variables are independent of the status code. On the other hands, since Domain variable has the smallest p-value, there is a strong evidence to conclude that there exists a relationship between domain type and status code.

[25]:
```
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from sklearn.feature_selection import chi2
pd.options.mode.chained_assignment = None

#select columns from data one and drop na
```

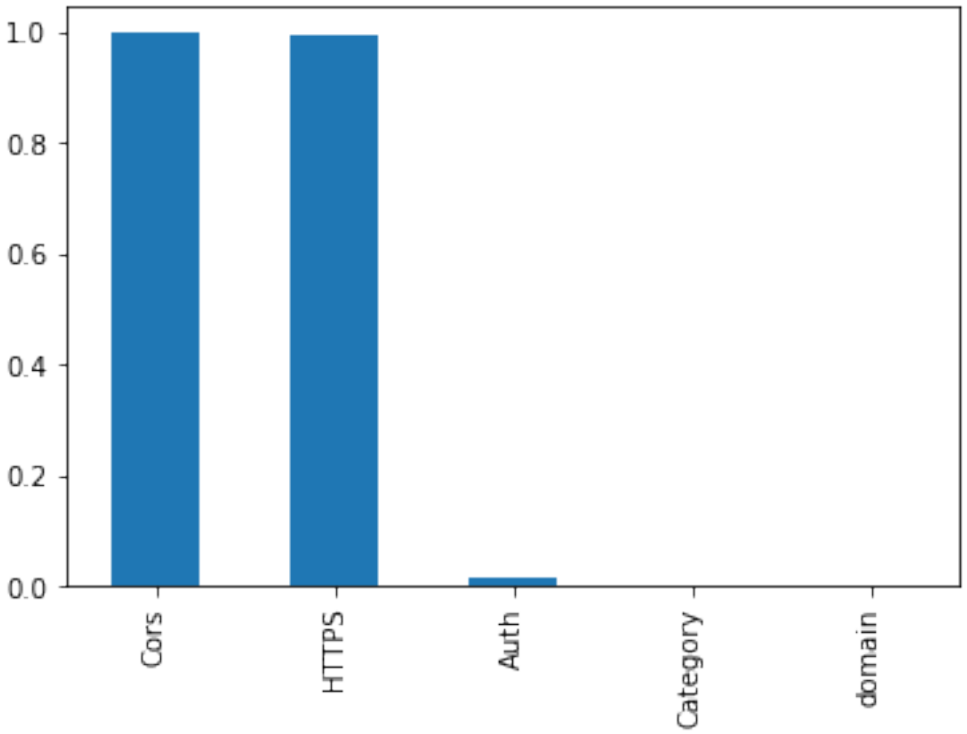
```
testchi_df=df_onenormal[['status_codes','domain','Category','Auth','HTTPS','Cors']]
testchi_df=testchi_df.dropna()

#label encoded
label_encoder = LabelEncoder()
testchi_df['status_codes'] = label_encoder.
 ↪fit_transform(testchi_df['status_codes'])
testchi_df['domain'] = label_encoder.fit_transform(testchi_df['domain'])
testchi_df['Category'] = label_encoder.fit_transform(testchi_df['Category'])
testchi_df['Auth'] = label_encoder.fit_transform(testchi_df['Auth'])
testchi_df['HTTPS'] = label_encoder.fit_transform(testchi_df['HTTPS'])
testchi_df['Cors'] = label_encoder.fit_transform(testchi_df['Cors'])


#chi squate test
X = testchi_df.drop('status_codes',axis=1)
y = testchi_df['status_codes']
chi_scores = chi2(X,y)
chi_scores #here first array represents chi square values and second array␣
 ↪represnts p-values

#show p-value and plot
p_values = pd.Series(chi_scores[1],index = X.columns)
p_values.sort_values(ascending = False , inplace = True)
p_values.plot.bar()
p_values
```

[25]:
```
Cors        9.984065e-01
HTTPS       9.954174e-01
Auth        1.842771e-02
Category    4.667775e-19
domain      1.478216e-42
dtype: float64
```



IV. Conclusion

From the plot, it shows the top 3 popular API categories even including the game & comic category, which indeed surprises me very much because I suppose that people will use these API websites mostly for commercial purposes. Therefore, in-deep analysis of "what kinds of game & comic API websites are more popular for users" are valuable in the future.

From these 1387 popular API websites, according to status codes, most of the websites work for users. In addition, it shows that most websites domains are COM. This result is the same as what we usually see on the internet. For example, most of the domain websites we used are COM such as yahoo.com and google.com.

From the map, most of the API websites are created in USA instead of Asian countries. This may be because most resourses are from USA. Besides, the most API websites are created in WA and NY States instead of CA which has the most population in USA. Therefore, the population may not have significant relationship with numbers of API webistes.

From the result of fail rates, it shows that COM domain has a fail rate of 0.176627. I believe that it is because COM has a larger sample size than other domains. Therefore, its fail rate is a little higher than those domains that people unusually see, such as eus, es, la and so on. In addition, according to the result of fail rate, it is found that edu domain type has zero fail rate, which is interesting since edu, gov and com are all top-level domains. However, from these top domains, only edu has zero fail rate. Therefore, taking more sample size to analyze the stability of these three top-level domains is another interesting topic.

Finally, the chi square test result shows that there is a strong relationship between domain type and status code. Therefore, in other to get the good AIP websites, the domain type of websites should be seriously considered since it is associated with status code.

V. Reference

1. YU,YINGXI , "Data Scientist Vs Software engineer from Cybercoder", https://madscientistkris.github.io/projects/cybercoders/Project_edited_version/ , 2017.
2. davemachado, "Public API for Public APIs", https://github.com/davemachado/public-api , 2017.
3. Chen, Jiewei and Xue, Da , "Recent Job Market Analysis" , https://celinechen0211.github.io/JobMarket/jobmarket.html, 2017.
4. "STA 141B - Lecture 21: Parsing HTML", https://github.com/a-f-farris/141b-lectures/blob/main/lecture21/lecture21.ipynb, 2021.
5. Oh, John, "Using Python to create a world map from a list of country names", https://towardsdatascience.com/using-python-to-create-a-world-map-from-a-list-of-country-names-cd7480d03b10, 2020.
6. gajawada, sampath kumar,"Chi-Square Test for Feature Selection in Machine learning", https://towardsdatascience.com/chi-square-test-for-feature-selection-in-machine-learning-206b1f0b8223, 2019.