TRIBHUVAN UNIVERSITY
INSTITUTE OF ENGINEERING
PULCHOWK CAMPUS

PROJECT REPORT ON

"SEARCH QUALITY ANALYSIS"

SUBMITTED BY:

Anil Kumar Sah (072/BEX/450)

Anil Poudel (072/BEX/404)

Bibek Thapa Magar (072/BEX/409)

SUBMITTED TO:

Mr. Khagendra B. Shrestha

DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING

LALITPUR, NEPAL

26 July, 2019

Table of Content

List of tables:

List of figures:

ACKNOWLEDGEMENT

We would like express our first and foremost gratitude towards the Department of Electronics and Computer Engineering, Pulchowk Campus for providing us the opportunity to undertake this project which has helped us learn and implement our skills and knowledge on the field of Big Data.

We would like to acknowledge Mr. Khagendra B. Shrestha for his continuous supervision, unwavering support, invaluable suggestions and consultation throughout the project which have propelled us to come a long way into accomplishing the project with the finest result.

Our thanks and appreciations also go to the authors of various papers that helped us gain the in-depth knowledge of various aspects. Finally, we would like to thank all the people who are directly or indirectly related for the successful completion of this project.

Sincerely,

Anil Kumar Sah

Bibek Thapa Magar

Anil Poudel

ABSTRACT

Truly, Search was Big Data before there was Big Data. Search has always been concerned with extremely large datasets, and statistical analysis of those sets, both for indexing i.e. for large-scale batch processing as well as at query-time i.e. for high-speed real-time processing. Billion-document databases have existed in search engines for decades.

Search has always lacked is a good, inclusive framework. Do we have that framework now? No. But we have the vision of the framework, and it lives on Big Data. All sorts of wonderful, amazing things that we've always wanted to do, but which were too hard, too expensive, and too unreliable to be used except in complex, hand-crafted implementations.

Parameters like linkage, performance, page rank link, popularity, etc. are involved in the organic search itself and for Google search result page to rank for the particular queries we made, depends on the metrices like total appearance, average rank of each domain and the coverage they made for that queries by the domains. All those effects of each metrices have been visualized clearly through the data frame we made and the chart we plot with the plotty. And for search relevancy of e-commerce site, proper description of the product was found to be the most important feature for search relevancy in customer search.

LIST OF ABBREVIATIONS

SERPs	Search Engine Result Pages
API	Application Programming Interface
CSE	Custom Search Engine
GL	Geo Location
WEB	World Wide Web

Search Quality Implementation I

"Analyzing SERPs Ranking All Over the World on Google"

1 INTRODUCTION

1.1 Background

Big Data will make search better and easier. Truly, Search was Big Data before there was Big Data. Search has always been concerned with extremely large datasets, and statistical analysis of those sets, both for indexing (i.e. large-scale batch processing) as well as at query-time (i.e. high-speed real-time processing). Billion-document databases have existed in search engines for decades.

But what Search has always lacked is a good, inclusive framework. Do we have that framework now? No. But we have the vision of the framework, and it lives on Big Data.

But what would we do with such a framework? Well, all sorts of wonderful, amazing things that we've always wanted to do, but which were too hard, too expensive, and too unreliable to be used except in complex, hand-crafted implementations. Things like:

- Link Counting
- Page Rank
- Anchor Text
- Popularity

These are all techniques using external references to improve search relevancy.

Link counting uses inbound links into a document (i.e. links from other documents) to boost relevancy. Google's "PageRank" has the same goal, but is mathematically more sophisticated. "Anchor text" can influence relevancy by taking into account how other people reference your document. Popularity boosts documents that are known to be more popular based on how often those documents are clicked.

How many implementations of these techniques have I seen over the years? Lots. And all of them have been terrible, hand-crafted, slow, awkward algorithms, which live inside document processing pipelines using relational databases.

But now, with Big Data, these algorithms become more than just possible; they are a natural evolution. After all, Google invented Map/Reduce specifically to handle Page Rank calculations for Google.com. It is because of the lack of an appropriate framework that all of this incredibly valuable external data is underused in most search implementations.

2 Methodology

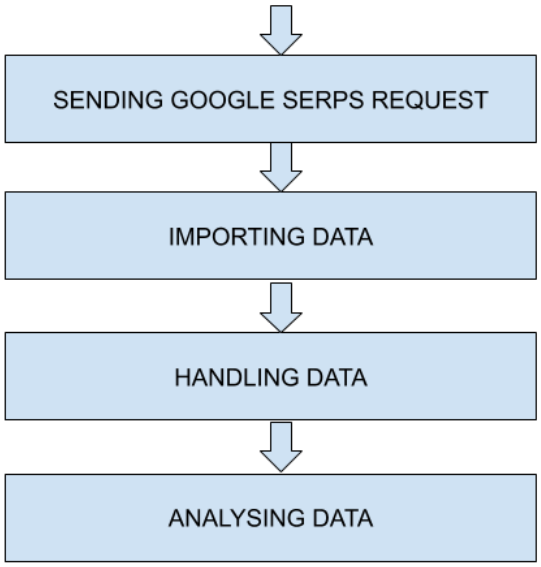

Figure 1: Workflow diagram of our proposed system

2.1 SENDING GOOGLE SERPs REQUEST

Google's Custom Search Engine is a service that allows you to create your own customized search engine, where you can specify the sites you want crawled and set your own relevancy rules (if you don't set any specific rules, then your custom search engine will essentially search the whole web by default).

You can further streamline your efforts by specifying parameters for your search queries, including the location of the user, the language of the site, image search, and much more.

You can also programmatically pull the data through its API, which is what we are going to do here. Here are the steps to set up an account to import data.

1. Create a custom search engine. At first, you might be asked to enter a site to search. Enter any domain, then go to the control panel and remove it. Make sure you enable "Search the entire web" and image search. You will also need to get your search engine ID, which you can find on the control panel page.
2. Enable the custom search API. The service will allow you to retrieve and display search results from your custom search engine programmatically. You will need to create a project for this first.
3. Create credentials for this project so you can get your key.
4. Enable billing for your project if you want to run more than 100 queries per day. The first 100 queries are free; then for each additional 1,000 queries, you pay USD $5.

The ‘gl’ parameter of the adv.serp_goog function stands for "geo-location". The available ‘gl’ are available as a dictionary as given below:

```
number of available locations: 239

country codes:
['ad', 'ae', 'af', 'ag', 'ai', 'al', 'am', 'an', 'ao', 'aq', 'ar', 'as', 'at', 'au', 'aw']
['az', 'ba', 'bb', 'bd', 'be', 'bf', 'bg', 'bh', 'bi', 'bj', 'bm', 'bn', 'bo', 'br', 'bs']
['bt', 'bv', 'bw', 'by', 'bz', 'ca', 'cc', 'cd', 'cf', 'cg', 'ch', 'ci', 'ck', 'cl', 'cm']
['cn', 'co', 'cr', 'cs', 'cu', 'cv', 'cx', 'cy', 'cz', 'de', 'dj', 'dk', 'dm', 'do', 'dz']
['ec', 'ee', 'eg', 'eh', 'er', 'es', 'et', 'fi', 'fj', 'fk', 'fm', 'fo', 'fr', 'ga', 'gd']
['ge', 'gf', 'gh', 'gi', 'gl', 'gm', 'gn', 'gp', 'gq', 'gr', 'gs', 'gt', 'gu', 'gw', 'gy']
['hk', 'hm', 'hn', 'hr', 'ht', 'hu', 'id', 'ie', 'il', 'in', 'io', 'iq', 'ir', 'is', 'it']
['jm', 'jo', 'jp', 'ke', 'kg', 'kh', 'ki', 'km', 'kn', 'kp', 'kr', 'kw', 'ky', 'kz', 'la']
['lb', 'lc', 'li', 'lk', 'lr', 'ls', 'lt', 'lu', 'lv', 'ly', 'ma', 'mc', 'md', 'mg', 'mh']
['mk', 'ml', 'mm', 'mn', 'mo', 'mp', 'mq', 'mr', 'ms', 'mt', 'mu', 'mv', 'mw', 'mx', 'my']
['mz', 'na', 'nc', 'ne', 'nf', 'ng', 'ni', 'nl', 'no', 'np', 'nr', 'nu', 'nz', 'om', 'pa']
['pe', 'pf', 'pg', 'ph', 'pk', 'pl', 'pm', 'pn', 'pr', 'ps', 'pt', 'pw', 'py', 'qa', 're']
['ro', 'ru', 'rw', 'sa', 'sb', 'sc', 'sd', 'se', 'sg', 'sh', 'si', 'sj', 'sk', 'sl', 'sm']
['sn', 'so', 'sr', 'st', 'sv', 'sy', 'sz', 'tc', 'td', 'tf', 'tg', 'th', 'tj', 'tk', 'tl']
['tm', 'tn', 'to', 'tr', 'tt', 'tv', 'tw', 'tz', 'ua', 'ug', 'uk', 'um', 'us', 'uy', 'uz']
['va', 'vc', 've', 'vg', 'vi', 'vn', 'vu', 'wf', 'ws', 'ye', 'yt', 'za', 'zm', 'zw']
```

Figure 2: Geo-location's Dictionary of 239 locations

Now, we send query request to all over the worlds. For this we use 'gl' parameter of Advertools.

```
coffee_df = adv.serp_goog(cx=cx, key=key, q='coffee',
                          gl=adv.SERP_GOOG_VALID_VALS['gl'])

cafe_df = adv.serp_goog(cx=cx, key=key, q='cafe',
                        gl=adv.SERP_GOOG_VALID_VALS['gl'])
```

```
2019-07-21 14:05:54,087 | INFO | serp.py:671 | serp_goog | Requesting: gl=py, key=AIzaSyCPGNo8hLYz9ep5Uv5H7Zg_4m6wveIrWRs, cx=007877757504657638886:cjk_v0ebh
2019-07-21 14:05:54,522 | INFO | serp.py:671 | serp_goog | Requesting: gl=ir, key=AIzaSyCPGNo8hLYz9ep5Uv5H7Zg_4m6wveIrWRs, cx=007877757504657638886:cjk_v0ebh
2019-07-21 14:05:54,942 | INFO | serp.py:671 | serp_goog | Requesting: gl=th, key=AIzaSyCPGNo8hLYz9ep5Uv5H7Zg_4m6wveIrWRs, cx=007877757504657638886:cjk_v0ebh
2019-07-21 14:05:55,345 | INFO | serp.py:671 | serp_goog | Requesting: gl=re, key=AIzaSyCPGNo8hLYz9ep5Uv5H7Zg_4m6wveIrWRs, cx=007877757504657638886:cjk_v0ebh
2019-07-21 14:05:55,736 | INFO | serp.py:671 | serp_goog | Requesting: gl=pn, key=AIzaSyCPGNo8hLYz9ep5Uv5H7Zg_4m6wveIrWRs, cx=007877757504657638886:cjk_v0ebh
2019-07-21 14:05:56,129 | INFO | serp.py:671 | serp_goog | Requesting: gl=sr, key=AIzaSyCPGNo8hLYz9ep5Uv5H7Zg_4m6wveIrWRs, cx=007877757504657638886:cjk_v0ebh
2019-07-21 14:05:56,564 | INFO | serp.py:671 | serp_goog | Requesting: gl=wf, key=AIzaSyCPGNo8hLYz9ep5Uv5H7Zg_4m6wveIrWRs, cx=007877757504657638886:cjk_v0ebh
2019-07-21 14:05:56,966 | INFO | serp.py:671 | serp_goog | Requesting: gl=ag, key=AIzaSyCPGNo8hLYz9ep5Uv5H7Zg_4m6wveIrWRs, cx=007877757504657638886:cjk_v0ebh
2019-07-21 14:05:57,331 | INFO | serp.py:671 | serp_goog | Requesting: gl=ht, key=AIzaSyCPGNo8hLYz9ep5Uv5H7Zg_4m6wveIrWRs, cx=007877757504657638886:cjk_v0ebh
2019-07-21 14:05:57,713 | INFO | serp.py:671 | serp_goog | Requesting: gl=sm, key=AIzaSyCPGNo8hLYz9ep5Uv5H7Zg_4m6wveIrWRs, cx=007877757504657638886:cjk_v0ebh
2019-07-21 14:05:58,122 | INFO | serp.py:671 | serp_goog | Requesting: gl=bt, key=AIzaSyCPGNo8hLYz9ep5Uv5H7Zg_4m6wveIrWRs, cx=007877757504657638886:cjk_v0ebh
2019-07-21 14:05:58,478 | INFO | serp.py:671 | serp_goog | Requesting: gl=fo, key=AIzaSyCPGNo8hLYz9ep5Uv5H7Zg_4m6wveIrWRs, cx=007877757504657638886:cjk_v0ebh
2019-07-21 14:05:58,802 | INFO | serp.py:671 | serp_goog | Requesting: gl=gq, key=AIzaSyCPGNo8hLYz9ep5Uv5H7Zg_4m6wveIrWRs, cx=007877757504657638886:cjk_v0ebh
```

Figure 3: Sending SERPs request to all over the world

2.2 Handling the Data

We will be using three Python packages for our work:

Advertools: To connect to the Google CSE API and receive SERPs in a table format. (This is a package that I wrote and maintain. It has a bunch of productivity and analysis tools for online marketing.)

Pandas: For data manipulation, reshaping, merging, sorting, etc.

Matplotlib: For data visualization.

2.2.1 Some issues

We'd like to start by mentioning some of the possible issues in the dataset:

Keywords: "coffee" doesn't say much about the intention of the user, and it is very generic. So is "cafe", although the intention is likely easier to figure out than "coffee".

Language: coffee is an English word, and cafe is used in many other languages. The requests were run in all available countries, many of which don't speak English, or the languages that have "cafe".

Country weights: probably the most important issue. The visualization and counting assumes all countries are equal in value. They are not, whether in terms of population, GDP, coffee consumption etc. Ideally, you would have your own weights for countries, or whatever filtering parameters you are using and you can apply them for a better evaluation of the SERPs ranks.

2.3 Analyzing the Data

For this analysis, we will be using the Jupyter Notebook as editor. It is basically a browser-based tool that combines regular text, programming code, as well as the output of the code that is run. The output could be text, tables, as well as images, which could be data visualizations as you will see below. It looks a lot like a word processor, and it is great for running analyses, creating campaigns, and general programming work.

By allowing you to store the steps that you made by keeping a copy of the code, the notebook enables you to go back and see how you came to your conclusions, and whether or not there are errors or areas that need improvement. It further allows others to work on the analysis from the point where you left off.

The notebook contains text boxes that are referred to as "cells", and this is where you can enter regular text or code. Here is how it renders, including a brief description of the cells.

2.3.1 Overview of the Domains

The coffee related domains were found to be 176 and cafe related domain to be 463. And among both the common domains was 15, showing the result of both related queries. Below is given the common domain of both queries “coffee” and “cafe”.

```
Coffee domains: 176
Cafe domains: 463
# Common domains: 15
{'de.wikipedia.org',
 'en.wikipedia.org',
 'es.wikipedia.org',
 'ja.wikipedia.org',
 'nestle.jp',
 'simple.wikipedia.org',
 'twitter.com',
 'www.abc.net.au',
 'www.bkkmenu.com',
 'www.facebook.com',
 'www.maxicoffee.com',
 'www.nespresso.com',
 'www.starbucks.cl',
 'www.starbucks.com.ar',
 'www.youtube.com'}
```

Figure 4: Result showing common domain for both queries.

In figure above, you can see, the number of domains ranking for "cafe" is almost 2.5 times that of "coffee". I think it makes sense, because the former is more of a local keyword, and the geo-location makes a difference. So, Google is probably giving more local domains for each country. On the other, hand "coffee" is a generic term about the plant/drink so the location doesn't play an important role.

It's also interesting to see that only fifteen domains are common. If you remove the local versions of some of those domains, you will notice that they are even less than that. Getting the top domains is done by simply getting the ones that appeared the most in the dataset. As mentioned above this is not ideal, but once you see the top ten or fifteen side by side, you will get a good idea of who is performing the best.

```
['www.ncausa.org',
 'en.wikipedia.org',
 'www.medicalnewstoday.com',
 'www.healthline.com',
 'www.mayoclinic.org',
 'www.coffeereview.com',
 'coffeecollective.dk',
 'www.blackriflecoffee.com',
 'www.drinktrade.com',
 'www.starbucks.com',
 'www.coffeebean.com',
 'www.peets.com',
 'app.starbucks.com',
 'www.webmd.com',
 'www.intelligentsiacoffee.com']
```

Figure 5: Top 15 domain in query for "coffee".

gl	searchTerms	rank	title	snippet	displayLink	link	queryTime	totalResults
nf	coffee	1	Coffee - Wikipedia	Coffee is a brewed drink prepared from roasted...	en.wikipedia.org	https://en.wikipedia.org/wiki/Coffee	2019-06-20 12:27:11.020962+00:00	1680000000
nf	coffee	2	Coffee: Benefits, nutrition, and risks	Health benefits, say some researchers, may ran...	www.medicalnewstoday.com	https://www.medicalnewstoday.com/articles/2702...	2019-06-20 12:27:11.020962+00:00	1680000000
nf	coffee	3	What is Coffee?	Coffee trees are pruned short to conserve thei	www.ncausa.org	http://www.ncausa.org/about-coffee/what-is-coffee	2019-06-20 12:27:11.020962+00:00	1680000000

Figure 6: Pandas Data frame of the above result.

2.3.2 Keywords used:

Total appearances: the number of times the domain appeared in SERPs (top 10)

Coverage: total appearances divided by the total queries (shown as a percentage)

Average position: Average rank of each of the domains.

	displayLink	rank_count	rank_mean
0	app.starbucks.com	75	8.093333
1	coffeecollective.dk	116	9.620690
2	en.wikipedia.org	245	1.106122
3	www.blackriflecoffee.com	105	7.714286
4	www.coffeebean.com	81	5.518519
5	www.coffeereview.com	127	8.118110
6	www.drinktrade.com	105	6.752381
7	www.healthline.com	202	5.282178
8	www.intelligentsiacoffee.com	56	8.017857
9	www.mayoclinic.org	136	6.316176
10	www.medicalnewstoday.com	225	2.240000
11	www.ncausa.org	299	4.364548
12	www.peets.com	80	6.550000
13	www.starbucks.com	98	3.632653
14	www.webmd.com	64	8.906250

Table 1: The table showing the domain list along with rank count and rank mean.

	displayLink	count	avg_rank	coverage
123	ncausa.org	299	4.4	125.1%
22	en.wikipedia.org	245	1.1	102.5%
120	medicalnewstoday.com	225	2.2	94.1%
100	healthline.com	202	5.3	84.5%
119	mayoclinic.org	136	6.3	56.9%
84	coffeereview.com	127	8.1	53.1%
7	coffeecollective.dk	116	9.6	48.5%
92	drinktrade.com	105	6.8	43.9%
66	blackriflecoffee.com	105	7.7	43.9%
145	starbucks.com	98	3.6	41.0%

Table 2: The table showing the domain list along with rank count, rank mean and coverage.

2.4 Visualizing the data:

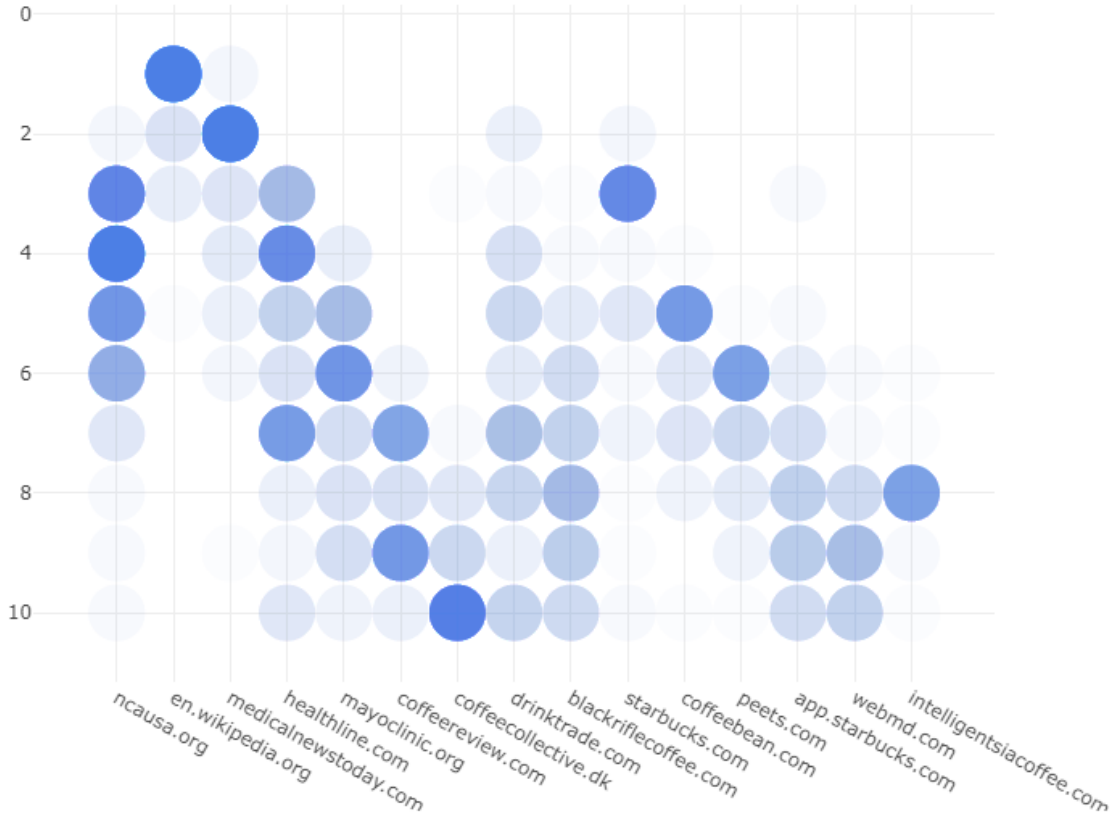

Figure 7: The figure below shows how often each domain ranks per position on SERPs.

The opacity of the circle makes it very easy to see the top spots. Yet, it is difficult to know really how different they are, since there are many levels of opacity. For this, we can add the actual numbers to the chart, and this way you can visually spot the top locations, and get the actual number of appearance per position.

We first create the DataFrame rank and count, which shows how many times each domain ranked per position as shown below:

	displayLink	rank	count
0	app.starbucks.com	3	2
1	app.starbucks.com	5	2
2	app.starbucks.com	6	6
3	app.starbucks.com	7	12
4	app.starbucks.com	8	19
5	app.starbucks.com	9	21
6	app.starbucks.com	10	13
7	coffeecollective.dk	3	1
8	coffeecollective.dk	7	2
9	coffeecollective.dk	8	8
10	coffeecollective.dk	9	15
11	coffeecollective.dk	10	90
12	en.wikipedia.org	1	228
13	en.wikipedia.org	2	10
14	en.wikipedia.org	3	6

Table 3 :Table showing DataFrame with rank and count

Now, we can easily add another layer to the chart, with the numbers mentioned.

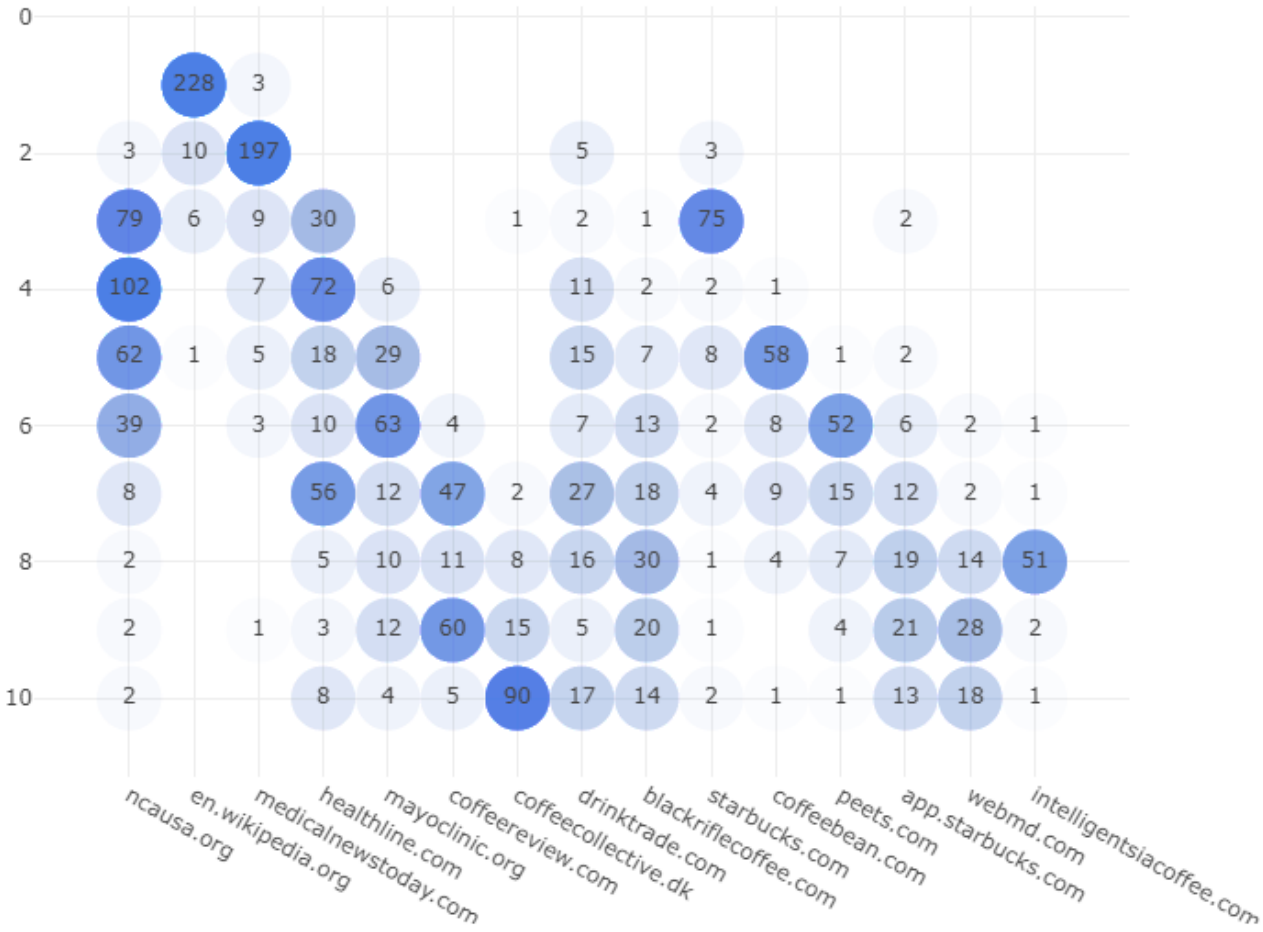

Figure 8: Chart showing rank and coverage along with the number

In the chart above does having ncausa.org before wikipedia.org make sense? It's clear that Wikipedia's average rank is higher, but what about the number of appearances and coverage?

We add those data points for each domain with the following loops, so we can have a final number summarizing each of those metrics, and place them right below the domain on the chart.

Now we have a fuller picture about the domains.

You can immediately see which domains are performing the best. You can also see how many times they ranked for each position. Then you can see aggregates (Total appearances, Coverage, and Avg. Pos.)You will also notice that I added the number of queries dynamically to the chart's title, as well as the list of queries that were used in the SERP DataFrame (in this case we only have one keyword).

Now let's make it a function by putting all the code in sequence, and providing a few options to customize the chart:

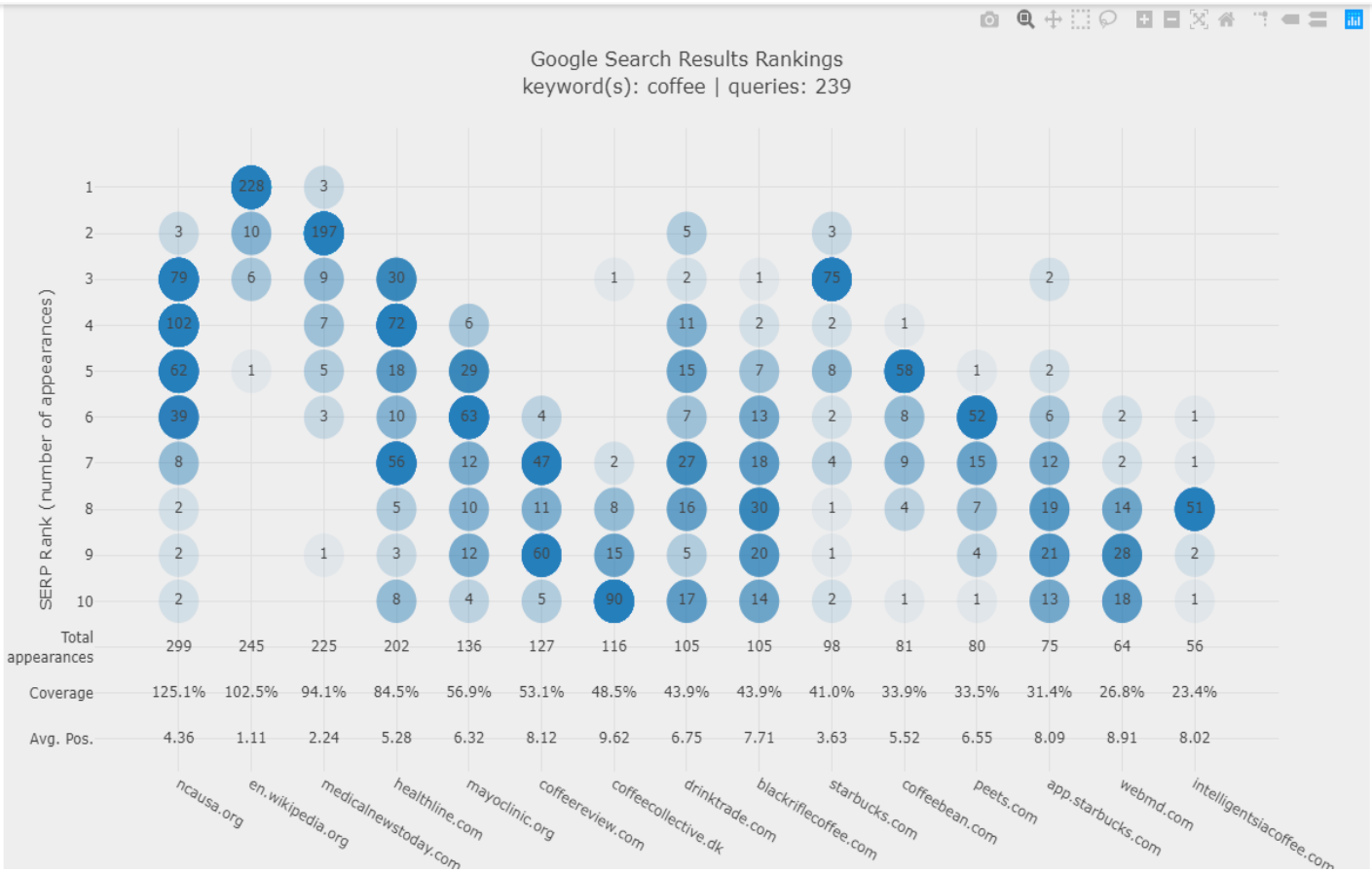

Figure 9: The chart below showing the Google Result Ranking for the queries "cafe" along with total appearances, coverage and average positions of each domain.

Now we have a fuller picture about the domains.
You can immediately see which domains are performing the best. You can also see how many times they ranked for each position. Then you can see aggregates (Total appearances, Coverage, and Avg. Pos.)

You will also notice that I added the number of queries dynamically to the chart's title, as well as the list of queries that were used in the SERP Data Frame (in this case we only have one keyword).

Now let's make it a function by putting all the code in sequence, and providing a few options to customize the chart:

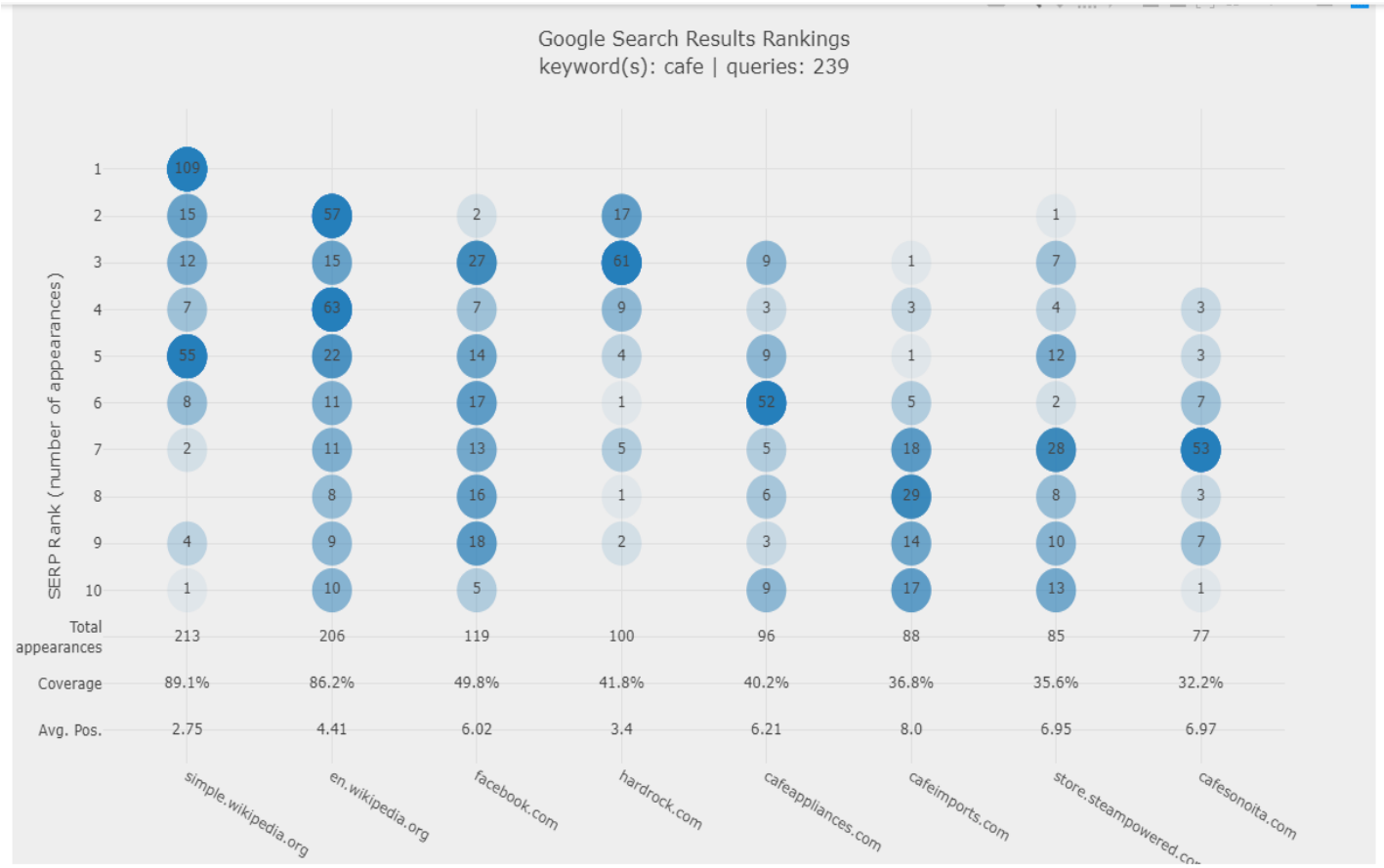

Figure 10: The similar chart below showing the Google Result Ranking for the queries "cafe" along with total appearances, coverage and average positions of each domain.

Search Quality Implementation II

"Analysis of Feature for E-commerce site search relevancy"

3 Methodology

The purpose of this implementation was identified relevancy determining factors query-based in real online shopping site search that can ultimately help consumer to find information/products

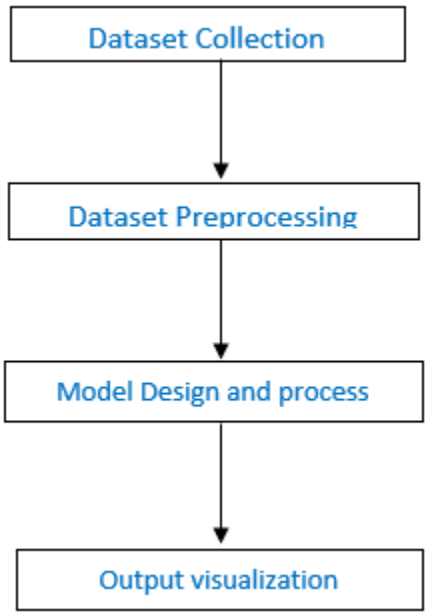

Figure 11:working process for feature analysis for e-commerce site search relevancy

3.1 Dataset collection

Dataset was real which was of number of products and real customer search terms from Home Depot's website which is an American home improvement supplies retailing company that sells tools, construction products, and services.

```
In [7]: df_all.head()
```

Out[7]:

	id	product_uid	product_title	search_term	relevance	product_description	brand
0	2	100001	Simpson Strong-Tie 12-Gauge Angle	angle bracket	3.00	Not only do angles make joints stronger, they ...	Simpson Strong-Tie
1	3	100001	Simpson Strong-Tie 12-Gauge Angle	l bracket	2.50	Not only do angles make joints stronger, they ...	Simpson Strong-Tie
2	9	100002	BEHR Premium Textured DeckOver 1-gal. #SC-141 ...	deck over	3.00	BEHR Premium Textured DECKOVER is an innovativ...	BEHR Premium Textured DeckOver
3	16	100005	Delta Vero 1-Handle Shower Only Faucet Trim Ki...	rain shower head	2.33	Update your bathroom with the Delta Vero Singl...	Delta
4	17	100005	Delta Vero 1-Handle Shower Only Faucet Trim Ki...	shower only faucet	2.67	Update your bathroom with the Delta Vero Singl...	Delta

Figure 12: Merged Dataset in form of table

3.2 Dataset preprocessing

Dataset preprocessing was in following steps

- Correction of search term based on Google dictionary
- Unify the unit.
- Remove special characters.
- Stemming text
- Tokenize text

3.3 Model Design and processing

Different statistical and counting features was taken as feature of the dataset. The XGBoost's plot importance function was used to plot features ordered by their importance. Following different parameters was selected based upon the necessity:

- n_estimator =Number of trees to fit
- Subsample=Subsample ratio of the training instance
- colsample_bytree=Subsample ratio of columns when constructing each tree
- Gamma=Minimum loss reduction required to make a further partition on a leaf node of the tree.
- max_depth =Maximum tree depth for base learners.

3.4 Output visualization

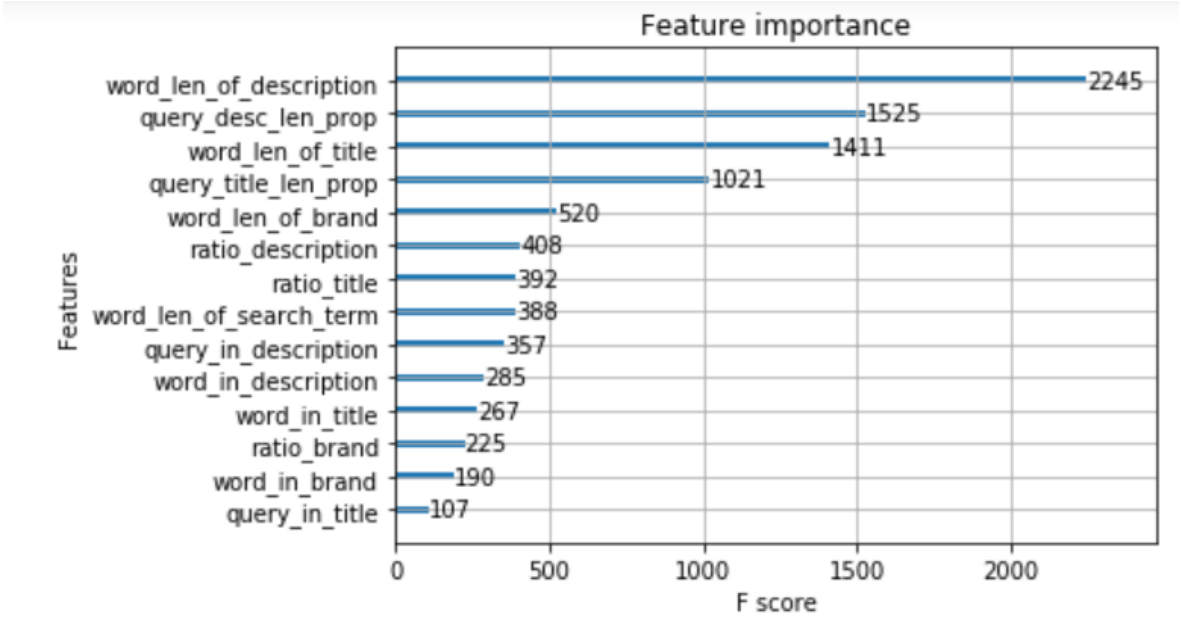

Figure 13: Visualization of feature importance vs f-score

From the above plot, we can see the relevancy was more dependent in features like word length of description, title word length etc. whereas it was less dependent on features such as word in brand, word in title etc.

4 Conclusion

Through this project, we conclude that a lot of parameters are involved in the organic search itself and for Google search result page to rank for the particular queries we made, depends on the metrices like total appearance, average rank of each domain and the coverage they made for that queries by the domains. All those effects of each metrices have been visualized clearly through the data frame we made and the chart we plot with the plotty. And for search relevancy of ecommerce site, proper description of the product was found to be the most important feature for search relevancy in customer search.

5 References

1. https://www.semrush.com/blog/analyzing-search-engine-results-pages/
2. https://www.searchtechnologies.com/search-quality-analysis
3. https://semrush.com/blog/analyzing-search-engine-results-pages/
4. https://developers.google.com/custom-search/v1/overview
5. https://www.algolia.com/doc/rest-api/search/
6. https://searchengineland.com/library/google/google-search-quality-raters

6 Source Code

6.1 Implementation I

```
# -*- coding: utf-8 -*-
"""Coffee" and "Cafe" Search Engine Rankings All Over the World.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1-8-
M3dz8oDTn6fOO5DZx6ECkwkIbIrqc
"""



"""# Analyzing of the SERP (Search Engine Results Pages) rankings of
"coffee" and "cafe" in all available countries on Google"""

!pip install advertools

# %config InlineBackend.figure_format = 'retina'
import matplotlib.pyplot as plt
import advertools as adv
import pandas as pd
pd.options.display.max_columns = None
import plotly.graph_objs as go
import plotly.offline as iplot
from plotly.offline import iplot, init_notebook_mode
init_notebook_mode()
pd.set_option('display.max_columns', None)
cx = '007877757504657638886:cjk_v0ebhm8'
key = 'AIzaSyCPGNo8hLYz9ep5Uv5H7Zg_4m6wveIrWRs'

from google.colab import drive
drive.mount('/content/drive')

coffee_df = adv.serp_goog(cx=cx, key=key, q='coffee',
                          gl=adv.SERP_GOOG_VALID_VALS['gl'])

cafe_df = adv.serp_goog(cx=cx, key=key, q='cafe',
                        gl=adv.SERP_GOOG_VALID_VALS['gl'])

coffee_df = pd.read_csv('coffee_serps.csv')
cafe_df = pd.read_csv('cafe_serps.csv')

country_codes = sorted(adv.SERP_GOOG_VALID_VALS['gl'])
print('number of available locations:', len(country_codes))
print()
print('country codes:')
print(*[country_codes[x:x+15] for x in range(0, len(country_codes), 15)],
      sep='\n')

print('Coffee domains:', coffee_df['displayLink'].nunique())
print('Cafe domains:', cafe_df['displayLink'].nunique())
```

```
common =
set(coffee_df['displayLink']).intersection(cafe_df['displayLink'])
print('# Common domains:', len(common))
common

"""**As you can see, the number of domains ranking for "cafe" is almost
2.5 times that of "coffee". I think it makes sense, because the fomer is
more of a local keyword, and the geo-location makes a difference. So
Google is probably giving more local domains for each country. On the
other hand "coffee" is a generic term about the plant/drink so the
location doesn't play an important role.**"""

num_domains = 15  # the number of domains to show in the chart
opacity = 0.02  # how opaque you want the circles to be
df = coffee_df  # which DataFrame you are using

top_domains =
df['displayLink'].value_counts()[:num_domains].index.tolist()
top_domains

top_domains =
df['displayLink'].value_counts()[:num_domains].index.tolist()
top_domains

top_df = df[df['displayLink'].isin(top_domains)]
top_df.head(3)

top_df_counts_means = (top_df
                       .groupby('displayLink', as_index=False)
                       .agg({'rank': ['count', 'mean']})
                       .set_axis(['displayLink', 'rank_count',
'rank_mean'],
                                 axis=1, inplace=False))
top_df_counts_means

top_df = (pd.merge(top_df, top_df_counts_means)
          .sort_values(['rank_count', 'rank_mean'],
                       ascending=[False, True]))
top_df.iloc[:3, list(range(8))+ [-2, -1]]

num_queries = df['queryTime'].nunique()

summary = (df
           .groupby(['displayLink'], as_index=False)
           .agg({'rank': ['count', 'mean']})
           .sort_values(('rank', 'count'), ascending=False)
           .assign(coverage=lambda df: (df[('rank',
'count')].div(num_queries))))
summary.columns = ['displayLink', 'count', 'avg_rank', 'coverage']
summary['displayLink'] = summary['displayLink'].str.replace('www.', '')
summary['avg_rank'] = summary['avg_rank'].round(1)
summary['coverage'] = (summary['coverage'].mul(100)
                       .round(1).astype(str).add('%'))
summary.head(10)
```

```
top10_domains = top_df.displayLink.value_counts()[:10].index
top_df = top_df[top_df['displayLink'].isin(top10_domains)]

top10_domains_coffee = coffee_df.displayLink.value_counts()[:10].index
top10_df_coffee = top_df[top_df['displayLink'].isin(top10_domains_coffee)]

top10_domains_cafe = cafe_df.displayLink.value_counts()[:10].index
top10_df_cafe = df[df['displayLink'].isin(top10_domains_cafe)]

configure_plotly_browser_state()
print('number of queries:', num_queries)
fig = go.Figure()
fig.add_scatter(x=top_df['displayLink'].str.replace('www.', ''),
                y=top_df['rank'], mode='markers',
                marker={'size': 35, 'opacity': opacity},
                showlegend=False)
fig.layout.height = 600
fig.layout.yaxis.autorange = 'reversed'
fig.layout.yaxis.zeroline = False
iplot(fig)



import plotly
configure_plotly_browser_state()
plotly.offline.init_notebook_mode(connected=False)
print('number of queries:', num_queries)
fig = go.Figure()
fig.add_scatter(x=top_df['displayLink'].str.replace('www.', ''),
                y=top_df['rank'], mode='markers',
                marker={'size': 35, 'opacity': opacity},
                showlegend=False)

fig.layout.height = 600
fig.layout.yaxis.autorange = 'reversed'
fig.layout.yaxis.zeroline = False
iplot(fig)

def configure_plotly_browser_state():
  import IPython
  display(IPython.core.display.HTML('''
        <script src="/static/components/requirejs/require.js"></script>
        <script>
          requirejs.config({
            paths: {
              base: '/static/base',
              plotly: 'https://cdn.plot.ly/plotly-latest.min.js?noext',
            },
          });
        </script>
        '''))

configure_plotly_browser_state()
```

```
for domain in rank_counts['displayLink'].unique():
    rank_counts_subset = rank_counts[rank_counts['displayLink']==domain]
    fig.add_scatter(x=[domain.replace('www.', '')],
                    y=[11], mode='text',
                    marker={'size': 50},
                    text=str(rank_counts_subset['count'].sum()))
    fig.add_scatter(x=[domain.replace('www.', '')],
                    y=[12], mode='text',
                    text=format(rank_counts_subset['count'].sum() /
num_queries, '.1%'))
    fig.add_scatter(x=[domain.replace('www.', '')],
                    y=[13], mode='text',
                    marker={'size': 50},
                    text=str(round(rank_counts_subset['rank']
                                   .mul(rank_counts_subset['count'])
                                   .sum() / rank_counts_subset['count']
                                   .sum(),2)))
fig.layout.title = ('Google Search Results Rankings<br>keyword(s): ' +
                    ', '.join(list(df['searchTerms'].unique())) +
                    ' | queries: ' + str(df['queryTime'].nunique()))
fig.layout.hovermode = False
fig.layout.yaxis.autorange = 'reversed'
fig.layout.yaxis.zeroline = False
fig.layout.yaxis.tickvals = list(range(1, 14))
fig.layout.yaxis.ticktext = list(range(1, 11)) +
['Total<br>appearances','Coverage', 'Avg. Pos.']
fig.layout.height = 700
fig.layout.width = 1200
fig.layout.yaxis.title = 'SERP Rank (number of appearances)'
fig.layout.showlegend = False
fig.layout.paper_bgcolor = '#eeeeee'
fig.layout.plot_bgcolor = '#eeeeee'
iplot(fig)

configure_plotly_browser_state()
def plot_serps(df, opacity=0.1, num_domains=10, width=1200, height=700):
    """
    df: a DataFrame resulting from running advertools.serp_goog
    opacity: the opacity of the markers [0, 1]
    num_domains: how many domains to plot
    """
    top_domains =
df['displayLink'].value_counts()[:num_domains].index.tolist()
    top_df = df[df['displayLink'].isin(top_domains)]
    top_df_counts_means = (top_df
                           .groupby('displayLink', as_index=False)
                           .agg({'rank': ['count', 'mean']})
                           .set_axis(['displayLink', 'rank_count',
'rank_mean'],
                                     axis=1, inplace=False))
    top_df = (pd.merge(top_df, top_df_counts_means)
          .sort_values(['rank_count', 'rank_mean'],
                       ascending=[False, True]))
    rank_counts = (top_df
```

```
                .groupby(['displayLink', 'rank'])
                .agg({'rank': ['count']})
                .reset_index()
                .set_axis(['displayLink', 'rank', 'count'],
                          axis=1, inplace=False))
    num_queries = df['queryTime'].nunique()
    fig = go.Figure()
    fig.add_scatter(x=top_df['displayLink'].str.replace('www.', ''),
                    y=top_df['rank'], mode='markers',
                    marker={'size': 35, 'opacity': opacity},
                    showlegend=False)
    fig.layout.height = 600
    fig.layout.yaxis.autorange = 'reversed'
    fig.layout.yaxis.zeroline = False
    fig.add_scatter(x=rank_counts['displayLink'].str.replace('www.', ''),
                y=rank_counts['rank'], mode='text',
                marker={'color': '#000000'},
                text=rank_counts['count'], showlegend=False)
    for domain in rank_counts['displayLink'].unique():
        rank_counts_subset =
rank_counts[rank_counts['displayLink']==domain]
        fig.add_scatter(x=[domain.replace('www.', '')],
                        y=[11], mode='text',
                        marker={'size': 50},
                        text=str(rank_counts_subset['count'].sum()))
        fig.add_scatter(x=[domain.replace('www.', '')],
                        y=[12], mode='text',
                        text=format(rank_counts_subset['count'].sum() /
num_queries, '.1%'))
        fig.add_scatter(x=[domain.replace('www.', '')],
                        y=[13], mode='text',
                        marker={'size': 50},
                        text=str(round(rank_counts_subset['rank']
                                       .mul(rank_counts_subset['count'])
                                       .sum() /
rank_counts_subset['count']
                                       .sum(),2)))
    fig.layout.title = ('Google Search Results Rankings<br>keyword(s): ' +
                        ', '.join(list(df['searchTerms'].unique())) +
                        ' | queries: ' + str(df['queryTime'].nunique()))
    fig.layout.hovermode = False
    fig.layout.yaxis.autorange = 'reversed'
    fig.layout.yaxis.zeroline = False
    fig.layout.yaxis.tickvals = list(range(1, 14))
    fig.layout.yaxis.ticktext = list(range(1, 11)) +
['Total<br>appearances','Coverage', 'Avg. Pos.']
    fig.layout.height = height
    fig.layout.width = width
    fig.layout.yaxis.title = 'SERP Rank (number of appearances)'
    fig.layout.showlegend = False
    fig.layout.paper_bgcolor = '#eeeeee'
    fig.layout.plot_bgcolor = '#eeeeee'
    iplot(fig)
```

```
configure_plotly_browser_state()
plot_serps(coffee_df, opacity=0.07, num_domains=15)

configure_plotly_browser_state()
plot_serps(cafe_df, opacity=0.07, num_domains=8)
```

6.2 Implementaion II

```
import numpy as np
import pandas as pd
from nltk.stem.porter import *
stemmer = PorterStemmer()
import re
import random

random.seed(2018)

df_train = pd.read_csv('train.csv', encoding="ISO-8859-1")
df_pro_desc = pd.read_csv('product_descriptions.csv')
df_attr = pd.read_csv('attributes.csv')

df_brand = df_attr[df_attr.name == "MFG Brand Name"][["product_uid",
"value"]].rename(columns={"value": "brand"})

df_brand.head()
df_brand.shape
df_brand.index = range(86250)
load_dictonary=”product_dict”
google_dict=load_dictonary

# In[12]:

df_all = pd.merge(df_train, df_pro_desc, how='left', on='product_uid')
df_all = pd.merge(df_all, df_brand, how='left', on='product_uid')


# In[13]:
df_all.head()


# Search_term spelling correction: using the Google dict from the forum
https://www.kaggle.com/steubk/home-depot-product-search-relevance/fixing-
typos

# In[14]:

df_all['search_term']=df_all['search_term'].map(lambda x: google_dict[x]
if x in google_dict.keys() else x)


# In[15]:
```

```
df_all.head(2)


# In[16]:


def str_stem(s):
    if isinstance(s, str):
        s = re.sub(r"([0-9])( *)\.( *)([0-9])", r"\1.\4", s)
        s = re.sub(r"([0-9]+)( *)(inches|inch|in|')\.?", r"\1in. ", s)
        s = re.sub(r"([0-9]+)( *)(foot|feet|ft|'')\.?", r"\1ft. ", s)
        s = re.sub(r"([0-9]+)( *)(pounds|pound|lbs|lb)\.?", r"\1lb. ", s)
        s = re.sub(r"([0-9]+)( *)(square|sq) ?\.?(feet|foot|ft)\.?",
r"\1sq.ft. ", s)
        s = re.sub(r"([0-9]+)( *)(cubic|cu) ?\.?(feet|foot|ft)\.?",
r"\1cu.ft. ", s)
        s = re.sub(r"([0-9]+)( *)(gallons|gallon|gal)\.?", r"\1gal. ", s)
        s = re.sub(r"([0-9]+)( *)(ounces|ounce|oz)\.?", r"\1oz. ", s)
        s = re.sub(r"([0-9]+)( *)(centimeters|cm)\.?", r"\1cm. ", s)
        s = re.sub(r"([0-9]+)( *)(milimeters|mm)\.?", r"\1mm. ", s)
        s = re.sub(r"([0-9]+)( *)(°|degrees|degree)\.?", r"\1 deg. ", s)
        s = re.sub(r"([0-9]+)( *)(v|volts|volt)\.?", r"\1 volt. ", s)
        s = re.sub(r"([0-9]+)( *)(wattage|watts|watt)\.?", r"\1 watt. ",
s)
        s = re.sub(r"([0-9]+)( *)(amperes|ampere|amps|amp)\.?", r"\1 amp.
", s)
        s = re.sub(r"([0-9]+)( *)(qquart|quart)\.?", r"\1 qt. ", s)
        s = re.sub(r"([0-9]+)( *)(hours|hour|hrs.)\.?", r"\1 hr ", s)
        s = re.sub(r"([0-9]+)( *)(gallons per minute|gallon per minute|gal
per minute|gallons/min.|gallons/min)\.?", r"\1 gal. per min. ", s)
        s = re.sub(r"([0-9]+)( *)(gallons per hour|gallon per hour|gal per
hour|gallons/hour|gallons/hr)\.?", r"\1 gal. per hr ", s)
        # Deal with special characters
        s = s.replace("$"," ")
        s = s.replace("?"," ")
        s = s.replace(" "," ")
        s = s.replace("&amp;","&")
        s = s.replace("&#39;","'")
        s = s.replace("/>/Agt/>","")
        s = s.replace("</a<gt/","")
        s = s.replace("gt/>","")
        s = s.replace("/>","")
        s = s.replace("<br","")
        s = s.replace("<.+?>","")
        s = s.replace("[ &<>)(_,;:!?\+^~@#\$]+"," ")
        s = s.replace("'s\\b","")
        s = s.replace("[']+","")
        s = s.replace("[\"]+","")
        s = s.replace("-"," ")
        s = s.replace("+"," ")
        # Remove text between paranthesis/brackets)
        s = s.replace("[ ]?[[(].+?[])]","")
        # remove sizes
```

```
        s = s.replace("size: .+$","")
        s = s.replace("size [0-9]+[.]?[0-9]+\\b","")


        return " ".join([stemmer.stem(re.sub('[^A-Za-z0-9-./]', ' ',
word)) for word in s.lower().split()])
    else:
        return "null"


# In[17]:

def str_common_word(str1, str2):
    str1, str2 = str1.lower(), str2.lower()
    words, count = str1.split(), 0
    for word in words:
        if str2.find(word)>=0:
            count+=1
    return count


# In[18]:

def str_whole_word(str1, str2, i_):
    str1, str2 = str1.lower().strip(), str2.lower().strip()
    count = 0
    while i_ < len(str2):
        i_ = str2.find(str1, i_)
        if i_ == -1:
            return count
        else:
            count += 1
            i_ += len(str1)
    return count


# In[19]:


df_all.isnull().sum()


# In[20]:


a = 0
for i in range(a,a+2):
    print(df_all.product_title[i])
    print(df_all.search_term[i])
    print(df_all.product_description[i])
    print(df_all.brand[i])
    print(df_all.relevance[i])
    print()
```

```
df_all.iloc[4,:]['search_term']
```

```
df_all.iloc[4,:]['product_title']
```

```
df_all.iloc[4,:]['product_description']
```

```
df_all['product_title'] = df_all['product_title'].apply(str_stem)
df_all['search_term'] = df_all['search_term'].apply(str_stem)
df_all['product_description'] =
df_all['product_description'].apply(str_stem)
df_all['brand'] = df_all['brand'].apply(str_stem)
```

```
a = 0
for i in range(a,a+2):
    print(df_all.product_title[i])
    print(df_all.search_term[i])
    print(df_all.product_description[i])
    print(df_all.brand[i])
    print(df_all.relevance[i])
    print()
```

```
df_all.head(2)
```

```
df_all['word_len_of_search_term'] = df_all['search_term'].apply(lambda
x:len(x.split())).astype(np.int64)
df_all['word_len_of_title'] = df_all['product_title'].apply(lambda
x:len(x.split())).astype(np.int64)
df_all['word_len_of_description'] =
df_all['product_description'].apply(lambda
x:len(x.split())).astype(np.int64)
df_all['word_len_of_brand'] = df_all['brand'].apply(lambda
x:len(x.split())).astype(np.int64)
# Create a new column that combine "search_term", "product_title" and
"product_description"
df_all['product_info'] =
df_all['search_term']+"\t"+df_all['product_title']
+"\t"+df_all['product_description']
# Number of times the entire search term appears in product title.
df_all['query_in_title'] = df_all['product_info'].map(lambda
x:str_whole_word(x.split('\t')[0],x.split('\t')[1],0))
# Number of times the entire search term appears in product description
df_all['query_in_description'] = df_all['product_info'].map(lambda
x:str_whole_word(x.split('\t')[0],x.split('\t')[2],0))
```

```
# Number of words that appear in search term also appear in product title.
df_all['word_in_title'] = df_all['product_info'].map(lambda
x:str_common_word(x.split('\t')[0],x.split('\t')[1]))
# Number of words that appear in search term also appear in production
description.
df_all['word_in_description'] = df_all['product_info'].map(lambda
x:str_common_word(x.split('\t')[0],x.split('\t')[2]))
# The ratio of product title word length to search term word length
df_all['query_title_len_prop']=df_all['word_len_of_title']/df_all['word_le
n_of_search_term']
# The ratio of product description word length to search term word length
df_all['query_desc_len_prop']=df_all['word_len_of_description']/df_all['wo
rd_len_of_search_term']
# The ratio of product title and search term common word count to search
term word count
df_all['ratio_title']=df_all['word_in_title']/df_all['word_len_of_search_t
erm']

# The ratio of product description and search term common word cout to
search term word count.
df_all['ratio_description'] =
df_all['word_in_description']/df_all['word_len_of_search_term']
# new column that combine "search_term", "brand" and "product_title".
df_all['attr'] =
df_all['search_term']+"\t"+df_all['brand']+"\t"+df_all['product_title']

# Number of words that appear in search term also apprears in brand.
df_all['word_in_brand'] = df_all['attr'].map(lambda
x:str_common_word(x.split('\t')[0],x.split('\t')[1]))

# The ratio of search term and brand common word count to brand word count
df_all['ratio_brand'] =
df_all['word_in_brand']/df_all['word_len_of_brand']

df_all.head(2)

df_all.isnull().sum()

df_all.head(2)

print(df_all.loc[[0]].to_dict())

df_all.columns

df_all.drop(['id', 'product_uid', 'product_title', 'search_term',
'product_description', 'brand', 'product_info', 'attr'], axis=1,
inplace=True)
df_all.columns

from sklearn.model_selection import train_test_split

X = df_all.loc[:, df_all.columns != 'relevance']
y = df_all.loc[:, df_all.columns == 'relevance']
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=0)



from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error

rf = RandomForestRegressor(n_estimators=100, max_depth=6, random_state=0)
rf.fit(X_train, y_train.values.ravel())
y_pred = rf.predict(X_test)
rf_mse = mean_squared_error(y_pred, y_test)
rf_rmse = np.sqrt(rf_mse)
print('RandomForest RMSE: %.4f' % rf_rmse)

from sklearn.linear_model import Ridge

rg= Ridge(alpha=.1)
rg.fit(X_train, y_train.values.ravel())
y_pred = rg.predict(X_test)
rg_mse = mean_squared_error(y_pred, y_test)
rg_rmse = np.sqrt(rg_mse)
print('Ridge RMSE: %.4f' % rg_rmse)

from sklearn.ensemble import GradientBoostingRegressor

est = GradientBoostingRegressor(n_estimators=100, learning_rate=0.1,
max_depth=1, random_state=0, loss='ls').fit(X_train,
y_train.values.ravel())
y_pred = est.predict(X_test)
est_mse = mean_squared_error(y_pred, y_test)
est_rmse = np.sqrt(est_mse)
print('Gradient boosting RMSE: %.4f' % est_rmse)


import xgboost
from sklearn.metrics import mean_squared_error
xgb = xgboost.XGBRegressor(n_estimators=100, learning_rate=0.08, gamma=0,
subsample=0.75, colsample_bytree=1, max_depth=7)
xgb.fit(X_train, y_train.values.ravel())
y_pred = xgb.predict(X_test)
xgb_mse = mean_squared_error(y_pred, y_test)
xgb_rmse = np.sqrt(xgb_mse)
print('Xgboost RMSE: %.4f' % xgb_rmse)


# The XGBoost library provides a built-in function called
plot_importance() to plot features ordered by their importance.



from xgboost import plot_importance
plot_importance(xgb);
```