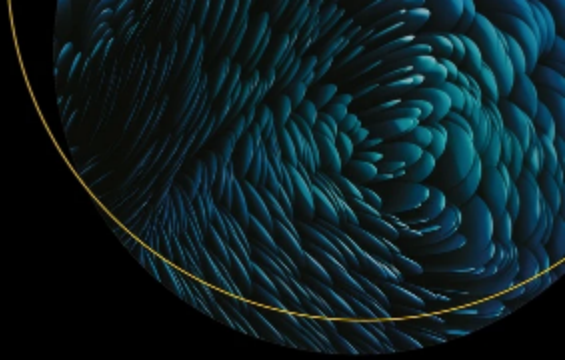

Data-Driven SEO with Python

Solve SEO Challenges with Data Science Using Python

—

Andreas Voniatis

Foreword by Will Critchlow, Founder and CEO, SearchPilot

Apress®

Data-Driven SEO with Python

Solve SEO Challenges with Data Science Using Python

Andreas Voniatis
Foreword by Will Critchlow, Founder and CEO, SearchPilot

Apress®

Data-Driven SEO with Python: Solve SEO Challenges with Data Science Using Python

Andreas Voniatis
Surrey, UK

ISBN-13 (pbk): 978-1-4842-9174-0 ISBN-13 (electronic): 978-1-4842-9175-7
https://doi.org/10.1007/978-1-4842-9175-7

Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: Celestin Suresh John
Development Editor: James Markham
Coordinating Editor: Mark Powers

Cover designed by eStudioCalamar

Cover image by Pawel Czerwinski on Unsplash (www.unsplash.com)

Distributed to the book trade worldwide by Apress Media, LLC, 1 New York Plaza, New York, NY 10004, U.S.A. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail booktranslations@springernature.com; for reprint, paperback, or audio rights, please e-mail bookpermissions@springernature.com.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at http://www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub (https://github.com/Apress). For more detailed information, please visit http://www.apress.com/source-code.

Printed on acid-free paper

To Julia.

Table of Contents

About the Author

Andreas Voniatis is the founder of Artios and a SEO consultant with over 20 year's experience working with ad agencies (PHD, Havas, Universal Mcann, Mindshare and iProspect), and brands (Amazon EU, Lyst, Trivago, GameSys). Andreas founded Artios in 2015 – to apply an advanced mathematical approach and cloud AI/Machine Learning to SEO.

With a background in SEO, data science and cloud engineering, Andreas has helped companies gain an edge through data science and automation. His work has been featured in publications worldwide including The Independent, PR Week, Search Engine Watch, Search Engine Journal and Search Engine Land.

Andreas is a qualified accountant, holds a degree in Economics from Leeds University and has specialised in SEO science for over a decade. Through his firm Artios, Andreas helps grow startups providing ROI guarantees and trains enterprise SEO teams on data driven SEO.

About the Contributing Editor

Simon Dance is the Chief Commercial Officer at Lyst.com, a fashion shopping platform serving over 200M users a year; an angel investor; and an experienced SEO having spent a 15-year career working in senior leadership positions including Head of SEO for Amazon's UK and European marketplaces and senior SEO roles at large-scale marketplaces in the flights and vacation rental space as well as consulting venture–backed companies including Depop, Carwow, and HealthUnlocked. Simon has worn multiple hats over his career from building links, manually auditing vast backlink profiles, carrying our comprehensive bodies of keyword research, and writing technical audit documents spanning hundreds of pages to building, mentoring, and leading teams who have unlocked significant improvements in SEO performance, generating hundreds of millions of dollars of incremental revenue. Simon met Andreas in 2015 when he had just built a rudimentary set of Python scripts designed to vastly increase the scale, speed, and accuracy of carrying out detailed keyword research and classification. They have worked together almost ever since.

About the Technical Reviewer

Joos Korstanje is a data scientist with over five years of industry experience in developing machine learning tools. He has a double MSc in Applied Data Science and in Environmental Science and has extensive experience working with geodata use cases. He has worked at a number of large companies in the Netherlands and France, developing machine learning for a variety of tools.

Acknowledgments

It's my first book and it wouldn't have been possible without the help of a few people. I'd like to thank Simon Dance, my contributing editor, who has asked questions and made suggested edits using his experience as an SEO expert and commercial director. I'd also like to thank all of the people at Springer Nature and Apress for their help and support. Wendy for helping me navigate the commercial seas of getting published. Will Critchlow for providing the foreword to this book. All of my colleagues, clients, and industry peers including SEOs, data scientists, and cloud engineers that I have had the pleasure of working with. Finally, my family, Petra and Julia.

Why I Wrote This Book

Since 2003, when I first got into SEO (by accident), much has changed in the practice of SEO. The ingredients were lesser known even though much of the focus was on getting backlinks, be they reciprocal, one-way links or from private networks (which are still being used in the gaming space). Other ingredients include transitioning to becoming a recognized brand, producing high-quality content which is valuable to users, a delightful user experience, producing and organizing content by search intent, and, for now and tomorrow, optimizing the search journey.

Many of the ingredients are now well known and are more complicated with the advent of mobile, social media, and voice and the increasing sophistication of search engines.

Now more than ever, the devil is in the details. There is more data being generated than ever before from ever more third-party data sources and tools. Spreadsheets alone won't hack it. You need a sharper blade, and data science (combined with your SEO knowledge) is your best friend.

I created this book for you, to make your SEO data driven and therefore the best it can be.

And why now in 2023? Because COVID-19 happened which gave me time to think about how I could add value to the world and in particular the niche world of SEO.

Even more presciently, there are lots of conversations on Twitter and LinkedIn about SEOs and the use of Python in SEO. So we felt the timing is right as the SEO industry has the appetite and we have knowledge to share.

I wish you the very best in your new adventure as a data-driven SEO specialist!

Who This Book Is For

We wrote this book to help you get ahead in your career as an SEO specialist. Whether you work in-house for a brand, an advertising agency, a consultant, or someone else (please write to us and introduce yourself!), this book will help you see SEO from a different angle and probably in a whole new way. Our goals for you are as follows:

- *A data science mindset to solving SEO challenges*: You'll start thinking about the outcome metrics, the data sources, the data structures to feed data into the model, and the models required to help you solve the problem or at the very least remove some of the disinformation surrounding the SEO challenge, all of which will take you several steps nearer to producing great SEO recommendations and ideas for split testing.
- *A greater insight into search engines*: You'll also have a greater appreciation for search engines like Google and a more contextual understanding of how they are likely to rank websites. After all, search engines are computer algorithms, not people, and so building your own algorithms and models to solve SEO challenges will give you some insight into how a search engine may reward or not reward certain features of a website and its content.
- *Code to get going*: The best way to learn naturally is by doing. While there are many courses in SEO, the most committed students of SEO will build their own websites and test SEO ideas and practices. Data science for SEO is no different if you want to make your SEO data driven. So, you'll be provided with starter scripts in Python to try your own hand in clustering pages and content, analyzing ranking factors. There will be code for most things but not for everything, as not everything has been coded for (yet). The code is there to get you started and can always be improved upon.
- *Familiarity with Python*: Python is the mainstay of data science in industry, even though R is still widely used. Python is free (open source) and is highly popular with the SEO community, data scientists, and the academic community alike. In fact, R and Python are quite similar in syntax and structure. Python is easy to use, read, and learn. To be clear, in no way do we suggest or advocate one language is better than the other, it's purely down to user preference and convenience.

Beyond the Scope

While this book promises and delivers on making your SEO data driven, there are a number of things that are better covered by other books out there, such as

- *How to become an SEO specialist*: What this book won't cover is how to become an SEO expert although you'll certainly come away with a lot of knowledge on how to be a better SEO specialist. There are some fundamentals that are beyond the scope of this book.

For example, we don't get into how a search engine works, what a content management system is, how it works, and how to read and code HTML and CSS. We also don't expose all of the ranking factors that a search engine might use to rank websites or how to perform a site relaunch or site migration.

This book assumes you have a rudimentary knowledge of how SEO works and what SEO is. We will give a data-driven view of the many aspects of SEO, and that is to reframe the SEO challenge from a data science perspective so that you have a useful construct to begin with.

- *How to become a data scientist*: This book will certainly expose the data science techniques to solve SEO challenges. What it won't do is teach you to become a data scientist or teach you how to program in the Python computing language.

To become a data science professional requires a knowledge of maths (linear algebra, probability, and statistics) in addition to programming. A true data scientist not only knows the theory and underpinnings of the maths and the software engineering to obtain and transform the data, they also know how and when to deploy certain models, the pros and cons of each (say Random Forest vs. AdaBoost), and how to rebuild each model from scratch. While we won't teach you how to become a fully fledged data scientist, you'll understand the intuition behind the models and how a data scientist would approach an SEO challenge.

There is no one answer of course; however, the answers we provide are based on experience and will be the best answer we believe at the time of writing. So you'll certainly be a data-driven SEO specialist, and if you take the trouble to learn data science properly, then you're well on your way to becoming an SEO scientist.

How This Book Works

Each chapter covers major work streams of SEO which will be familiar to you:

1. Keyword research
2. Technical
3. Content and UX
4. Authority
5. Competitor analysis
6. Experiments
7. Dashboards
8. Migration planning and postmortems
9. Google updates
10. Future of SEO

Under each chapter, we will define as appropriate

- SEO challenge(s) from a data perspective
- Data sources
- Data structures
- Models
- Model output evaluation
- Activation suggestions

I've tried to apply data science to as many SEO processes as possible in the areas identified earlier. Naturally, there will be some areas that could be applied that have not. However, technology is changing, and Google is already releasing updates to combat AI-written content. So I'd imagine in the very near future, more and more areas of SEO will be subject to data science.

Foreword

The data we have access to as SEOs has changed a lot during my 17 years in the industry. Although we lost analytics-level keyword data, and Yahoo! Site Explorer, we gained a wealth of opportunity in big data, proprietary metrics, and even some from the horse's mouth in Google Search Console.

You don't have to be able to code to be an effective SEO. But there is a certain kind of approach and a certain kind of mindset that benefits from wrangling data in all its forms. If that's how you prefer to work, you will very quickly hit the limits of spreadsheets and text editors. When you do, you'll do well to turn to more powerful tools to help you scale what you're capable of, get things done that you wouldn't even have been able to do without a computer helping, and speed up every step of the process.

There are a lot of programming languages, and a lot of ways of learning them. Some people will tell you there is only one right way. I'm not one of those people, but my personal first choice has been Python for years now. I liked it initially for its relative simplicity and ease of getting started, and very quickly fell for the magic of being able to import phenomenal power written by others with a single line of code. As I got to know the language more deeply and began to get some sense of the "pythonic" way of doing things, I came to appreciate the brevity and clarity of the language. I am no expert, and I'm certainly not a professional software engineer, but I hope that makes me a powerful advocate for the approach outlined in this book - because I have been the target market.

When I was at university, I studied neural networks among many other things. At the time, they were fairly abstract concepts in operations research. At that point in the late 90s, there wasn't the readily available computing power plus huge data sets needed to realise the machine learning capabilities hidden in those nodes, edges, and statistical relationships. I've remained fascinated by what is possible and with the help of magical import statements and remarkably mature frameworks, I have even been able to build and train my own neural networks in Python. As a stats geek, I love that it's all stats under the hood, but at the same time, I appreciate the beauty in a computer being able to do something a person can't.

A couple of years after university, I founded the SEO agency Distilled with my co-founder Duncan Morris, and one of the things that we encouraged among our SEO

consultants was taking advantage of the data and tools at their disposal. This led to fun innovation - both decentralised, in individual consultants building scripts and notebooks to help them scale their work, do it faster, or be more effective, and centrally in our R&D team.

That R&D team would be the group who built the platform that would become SearchPilot and launched the latest stage of my career where we are very much leading the charge for data aware decisions in SEO. We are building the enterprise SEO A/B testing platform to help the world's largest websites prove the value of their on-site SEO initiatives. All of this uses similar techniques to those outlined in the pages that follow to decide how to implement tests, to consume data from a variety of APIs, and to analyse their results with neural networks.

I believe that as Google implements more and more of their own machine learning into their ranking algorithms, that SEO becomes fundamentally harder as the system becomes harder to predict, and has a greater variance across sites, keywords, and topics. It's for this reason that I am investing so much time, energy, and the next phase of my career into our corner of data driven SEO. I hope that this book can set a whole new cohort of SEOs on a similar path.

I first met Andreas over a decade ago in London. I've seen some of the things he has been able to build over the years, and I'm sure he is going to be an incredible guide through the intricacies of wrangling data to your benefit in the world of SEO. Happy coding!

Will Critchlow, CEO, SearchPilot

September 2022

CHAPTER 1

Introduction

Before the Google Search Essentials (formerly Webmaster Guidelines), there was an unspoken contract between SEOs and search engines which promised traffic in return for helping search engines extract and index website content. This chapter introduces you to the challenges of applying data science to SEO and why you should use data.

The Inexact (Data) Science of SEO

There are many trends that motivate the application of data science to SEO; however, before we get into that, why isn't there a rush of data scientists to the industry door of SEO? Why are they going into areas of paid search, programmatic advertising, and audience planning instead?

Here's why:

- Noisy feedback loop
- Diminishing value of the channel
- Making ads look more like organic listings
- Lack of sample data
- Things that can't be measured
- High costs

Noisy Feedback Loop

Unlike paid search campaigns where changes can be live after 15 mins, the changes that affect SEO, be it on a website or indeed offsite, can take anywhere between an hour and many weeks for Google and other search engines to take note of and process the change

A. Voniatis, *Data-Driven SEO with Python*, https://doi.org/10.1007/978-1-4842-9175-7_1

within their systems before it gets reflected in the search engine results (which may or may not result in a change of ranking position).

Because of this variable and unpredictable time lag, this makes it rather difficult to undertake cause and effect analysis to learn from SEO experiments.

Diminishing Value of the Channel

The diminishing value of the channel will probably put off any decision by a data scientist to move into SEO when weighing up the options between computational advertising, financial securities, and other industries. SEO is likely to fall by the wayside as Google and others do as much as possible to reduce the value of organic traffic in favor of paid advertising.

Making Ads Look More like Organic Listings

Google is increasing the amount of ads shown before displaying the organic results, which diminishes the return of SEO (and therefore the appeal) to businesses. Google is also monetizing organic results such as Google Jobs, Flights, Credit Cards, and Shopping, which displaces the organic search results away from the top.

Lack of Sample Data

It's the lack of data points that makes data-driven SEO analysis more challenging. How many times has an SEO run a technical audit and taken this as a reflection of the SEO reality? How do we know this website didn't have an off moment during that particular audit?

Thank goodness, the industry-leading rank measurement tools are recording rankings on a daily basis. So why aren't SEO teams auditing on a more regular basis?

Many SEO teams are not set up to take multiple measurements because most do not have the infrastructure to do so, be it because they

- Don't understand the value of multiple measurements for data science
- Don't have the resources or don't have the infrastructure

- Rely on knowing when the website changes before having to run another audit (albeit tools like ContentKing have automated the process)

To have a dataset that has a true representation of the SEO reality, it must have multiple audit measurements which allow for statistics such as average and standard deviations per day of

- Server status codes
- Duplicate content
- Missing titles

With this type of data, data scientists are able to do meaningful SEO science work and track these to rankings and UX outcomes.

Things That Can't Be Measured

Even with the best will to collect the data, not everything worth measuring can be measured. Although this is likely to be true of all marketing channels, not just SEO, it's not the greatest reason for data scientists not to move into SEO. If anything, I'd argue the opposite in the sense that many things in SEO are measurable and that SEO is data rich.

There are things we would like to measure such as

- *Search query*: Google, for some time, has been hiding the search query detail of organic traffic, of which the keyword detail in Google Analytics is shown as "Not Provided." Naturally, this would be useful as there are many keywords to one URL relationship, so getting the breakdown would be crucial for attribution modeling outcomes, such as leads, orders, and revenue.
- *Search volume*: Google Ads does not fully disclose search volume per search query. The search volume data for long tail phrases provided by Ads is reallocated to broader matches because it's profitable for Google to encourage users to bid on these terms as there are more bidders in the auction. Google Search Console (GSC) is a good substitute, but is first-party data and is highly dependent on your site's presence for your hypothesis keyword.

- *Segment*: This would tell us who is searching, not just the keyword, which of course would in most cases vastly affect the outcomes of any machine-learned SEO analysis because a millionaire searching for "mens jeans" would expect different results to another user of more modest means. After all, Google is serving personalized results. Not knowing the segment simply adds noise to any SERPs model or otherwise.

High Costs

Can you imagine running a large enterprise crawling technology like Botify daily? Most brands run a crawl once a month because it's cost prohibitive, and not just on your site. To get a complete dataset, you'd need to run it on your competitors, and that's only one type of SEO data.

Cost won't matter as much to the ad agency data scientist, but it will affect whether they will get access to the data because the agency may decide the budget isn't worthwhile.

Why You Should Turn to Data Science for SEO

There are many reasons to turn to data science to make your SEO campaigns and operations data driven.

SEO Is Data Rich

We don't have the data to measure everything, including Google's user response data to the websites listed in the Search Engine Results Pages (SERPs), which would be the ultimate outcome data. What we do have is first-party (your/your company's data like Google/Adobe Analytics) and third-party (think rank checking tools, cloud auditing software) export data.

We also have the open source data science tools which are free to make sense of this data. There are also many free highly credible sources online that are willing to teach you how to use these tools to make sense of the ever-increasing deluge of SEO data.

SEO Is Automatable

At least in certain aspects. We're not saying that robots will take over your career. And yet, we believe there is a case that some aspects of your job as an SEO a computer can do instead. After all, computers are extremely good at doing repetitive tasks, they don't get tired nor bored, can "see" beyond three dimensions, and only live on electricity.

Andreas has taken over teams where certain members spent time constantly copying and pasting information from one document to another (the agency and individual will remain unnamed to spare their blushes).

Doing repetitive work that can be easily done by a computer is not value adding, emotionally engaging, nor good for your mental health. The point is we as humans are at our best when we're thinking and synthesizing information about a client's SEO; that's when our best work gets done.

Data Science Is Cheap

We also have the open source data science tools (R, Python) which are free to make sense of this data. There are also many free highly credible sources online that are willing to teach you how to use these tools to make sense of the ever-increasing deluge of SEO data.

Also, if there is too much data, cloud computing services such as Amazon Web Services (AWS) and Google Cloud Platform (GCP) are also rentable by the hour.

Summary

This brief introductory chapter has covered the following:

- The inexact science of SEO
- Why you should turn to data science for SEO

CHAPTER 2

Keyword Research

Behind every query a user enters within a search engine is a word or series of words. For instance, a user may be looking for a "hotel" or perhaps a "hotel in New York City." In search engine optimization (SEO), keywords are invariably the target, providing a helpful way of understanding demand for said queries and helping to more effectively understand various ways that users search for products, services, organizations, and, ultimately, answers.

As well as SEO starting from keywords, it also tends to end with the keyword as an SEO campaign may be evaluated on the value of the keyword's contribution. Even if this information is hidden from us by Google, attempts have been made by a number of SEO tools to infer the keyword used by users to reach a website.

In this chapter, we will give you data-driven methods for finding valuable keywords for your website (to enable you to have a much richer understanding of user demand).

It's also worth noting that given keyword rank tracking comes at a cost (usually charged per keyword tracked or capped at a total number of keywords), it makes sense to know which keywords are worth the tracking cost.

Data Sources

There are a number of data sources when it comes to keyword research, which we'll list as follows:

- **Google Search Console**
- **Competitor Analytics**
- **SERPs**
- **Google Trends**
- Google Ads
- Google Suggest

A. Voniatis, *Data-Driven SEO with Python*, https://doi.org/10.1007/978-1-4842-9175-7_2

We'll cover the ones highlighted in bold as they are not only the more informative of the data sources, they also scale as data science methods go. Google Ads data would only be so appealing if it were based on actual impression data.

We will also show you how to make forecasts of keyword data both in terms of the amount of impressions you get if you achieve a ranking on page 1 (within positions 1 to 10) and what this impact would be over a six-month horizon.

Armed with a detailed understanding of *how* customers search, you're in a much stronger position to benchmark where you index vs. this demand (in order to understand the available opportunity you can lean into), as well as be much more customer focused when orienting your website and SEO activity to target that demand.

Let's get started.

Google Search Console (GSC)

Google Search Console (GSC) is a (free) first-party data source, which is rich in market intelligence. It's no wonder Google does everything possible to make it difficult to parse, let alone obfuscate, the data when attempting to query the API at date and keyword levels.

GSC data is my first port of call when it comes to keyword research because the numbers are consistent, and unlike third-party numbers, you'll get data which isn't based on a generic click through a rate mapped to ranking.[1]

The overall strategy is to look for search queries that have impressions that are significantly above the average for their ranking position. Why impressions? Because impressions are more plentiful and they represent the opportunity, whereas clicks tend to come "after the fact," that is, they are the outcome of the opportunity.

What is significant? This could be any search query with impression levels more than two standard deviations (sigmas) above the mean (average), for example.

[1] In 2006, AOL shared click-through rate data based upon over 35 million search queries, and since then it has inspired numerous models to try and estimate the click-through rate (CTR) by search engine ranking position. That is, for every 100 people searching for "hotels in New York," 30% (for example) click on the position 1 ranking, with just 16% clicking on position 2 (hence the importance of achieving the top ranked position, in order to, effectively, double your traffic (for that keyword))

There is no hard and fast rule. Two sigmas simply mean that there's a less than 5% chance that the search query is actually less like the average search query, so a lower significance threshold like one sigma could easily suffice.

Import, Clean, and Arrange the Data

```
import pandas as pd
import numpy as np
import glob
import os
```

The data are several exports from Google Search Console (GSC) of the top 1000 rows based on a number of filters. The API could be used, and some code is provided in Chapter 10 showing how to do so.

For now, we're reading multiple GSC export files stored in a local folder.

Set the path to read the files:

```
data_dir = os.path.join('data', 'csvs')
gsc_csvs = glob.glob(data_dir + "/*.csv")
```

Initialize an empty list that will store the data being read in:

```
gsc_li = []
```

The for loop iterates through each export file and takes the filename as the modifier used to filter the results and then appends it to the preceding list:

```
for cf in gsc_csvs:
    df = pd.read_csv(cf, index_col=None, header=0)
    df['modifier'] = os.path.basename(cf)
    df.modifier = df.modifier.str.replace('_queries.csv', '')
    gsc_li.append(df)
```

Once the list is populated with the export data, it's combined into a single dataframe:

```
gsc_raw_df = pd.DataFrame()
gsc_raw_df = pd.concat(gsc_li, axis=0, ignore_index=True)
```

The columns are formatted to be more data-friendly:

```
gsc_raw_df.columns = gsc_raw_df.columns.str.strip().str.lower().str.
replace(' ', '_').str.replace('(', '').str.replace(')', '')

gsc_raw_df.head()
```

This produces the following:

	top_queries	clicks	impressions	ctr	position	modifier
0	ps4 cd keys	17	206	8.25%	13.40	cdkeys
1	cheap ps4 cd keys	13	40	32.5%	9.38	cdkeys
2	ps4 cd key	12	34	35.29%	19.44	cdkeys
3	cheap cd keys ps4	11	21	52.38%	8.71	cdkeys
4	xbox cd keys	8	89	8.99%	13.46	cdkeys

With the data imported, we'll want to format the column values to be capable of being summarized. For example, we'll remove the percent signs in the ctr column and convert it to a numeric format:

```
gsc_clean_ctr_df['ctr'] = gsc_clean_ctr_df['ctr'].str.replace('%', '')
gsc_clean_ctr_df['ctr'] = pd.to_numeric(gsc_clean_ctr_df['ctr'])
```

GSC data contains a funny character "<" in the impressions and clicks columns for values less than 10; our job is to clean this up by removing them and then arranging impressions in descending order. In Python, this would look like

```
gsc_clean_ctr_df['impressions'] = gsc_clean_ctr_df.impressions.str.
replace('<', '')
pd.to_numeric(gsc_import_df.impressions)
```

We'll also deduplicate the top_queries column:

```
gsc_dedupe_df = gsc_clean_ctr_df.drop_duplicates(subset='top_queries',
keep="first")
```

Segment by Query Type

The next step is to segment the queries by type. The reason for this is that we want to compare the impression volumes within a segment as opposed to the overall website.

This makes numbers more meaningful in terms of highlighting opportunities within segments. Otherwise, if we compared impressions to the website average, then we may miss out on valuable search query opportunities.

The approach we're using in Python is to categorize based on modifier strings found in the query column:

```
retail_vex = ['cdkeys', 'argos', 'smyth', 'amazon', 'cyberpunk', 'GAME']
platform_vex = ['ps5', 'xbox', 'playstation', 'switch', 'ps4', 'nintendo']
title_vex = ['blackops', 'pokemon', 'minecraft', 'mario',
'outriders','fifa', 'animalcrossing', 'resident', 'spiderman',
'newhorizons', 'callofduty']
network_vex = ['ee', 'o2', 'vodafone','carphone']

gsc_segment_strdetect = gsc_dedupe_df[['query', 'clicks', 'impressions',
'ctr', 'position']]
```

Create a list of our conditions:

```
query_conds = [
    gsc_segment_strdetect['query'].str.contains('|'.join(retail_vex)),
    gsc_segment_strdetect['query'].str.contains('|'.join(platform_vex)),
    gsc_segment_strdetect['query'].str.contains('|'.join(title_vex)),
    gsc_segment_strdetect['query'].str.contains('|'.join(network_vex))
]
```

Create a list of the values we want to assign for each condition:

```
segment_values = ['Retailer', 'Console', 'Title', 'Network'] #, 'Title',
'Accessories', 'Network', 'Top1000', 'Broadband']
```

Create a new column and use np.select to assign values to it using our lists as arguments:

```
gsc_segment_strdetect['segment'] = np.select(query_conds, segment_values)

gsc_segment_strdetect
```

Here is the output:

	query	clicks	impressions	ctr	position	segment
0	ps4 cd keys	17	206	8.25	13.40	Console
1	cheap ps4 cd keys	13	40	32.5	9.38	Console
2	ps4 cd key	12	34	35.29	19.44	Console
3	cheap cd keys ps4	11	21	52.38	8.71	Console
4	xbox cd keys	8	89	8.99	13.46	Console
...	...	...	...	...	...	...
18567	nintendo switch lite on credit	48	94	51.06	2.73	Console
18568	nintendo switch limited edition	47	3430	1.37	13.27	Console
18569	the cheapest nintendo switch	47	1182	3.98	5.16	Console
18570	nintendo switch lite grey bundle	47	721	6.52	8.03	Console
18571	where to buy a nintendo switch uk	47	620	7.58	11.03	Console

15867 rows × 6 columns

Round the Position Data into Whole Numbers

Given the position column is a floating number (i.e., contains decimals), the reason we'd like to do this is because we'll be calculating the impression statistics per rounded ranking position. This will give us 100 statistics. Now imagine if we didn't round it, we could have impression statistics for 10,000 ranking positions and not all of them are useful.

```
gsc_segment_strdetect['rank_bracket'] = gsc_segment_strdetect.position.
round(0)
gsc_segment_strdetect
```

This results in the following:

	query	clicks	impressions	ctr	position	segment	rank_bracket
0	ps4 cd keys	17	206	8.25	13.40	Console	13.0
1	cheap ps4 cd keys	13	40	32.5	9.38	Console	9.0
2	ps4 cd key	12	34	35.29	19.44	Console	19.0
3	cheap cd keys ps4	11	21	52.38	8.71	Console	9.0
4	xbox cd keys	8	89	8.99	13.46	Console	13.0
...	...	...	...	...	...	...	...
18567	nintendo switch lite on credit	48	94	51.06	2.73	Console	3.0
18568	nintendo switch limited edition	47	3430	1.37	13.27	Console	13.0
18569	the cheapest nintendo switch	47	1182	3.98	5.16	Console	5.0
18570	nintendo switch lite grey bundle	47	721	6.52	8.03	Console	8.0
18571	where to buy a nintendo switch uk	47	620	7.58	11.03	Console	11.0

Calculate the Segment Average and Variation

Now the data is segmented, we compute the average impressions and the lower and upper percentiles of impressions for the ranking position. The aim is to identify queries that have impressions two standard deviations or more above the ranking position. This means the query is likely to be a great opportunity for SEO and well worth monitoring.

The reason we're doing it this way, as opposed to just selecting high impression keywords per se, is because many keyword queries have high impressions just by virtue of being in the top 20 in the first place. This would make the number of queries to track rather large and expensive.

```
queries_rank_imps = gsc_segment_strdetect[['rank_bracket', 'impressions']]
group_by_rank_bracket = queries_rank_imps.groupby(['rank_bracket'], as_
index=False)
```

```
def imp_aggregator(col):
    d = {}
    d['avg_imps'] = col['impressions'].mean()
    d['imps_median'] = col['impressions'].quantile(0.5)
    d['imps_lq'] = col['impressions'].quantile(0.25)
    d['imps_uq'] = col['impressions'].quantile(0.95)
    d['n_count'] = col['impressions'].count()
```

```
    return pd.Series(d, index=['avg_imps', 'imps_median', 'imps_lq', 'imps_
    uq', 'n_count'])

overall_rankimps_agg = group_by_rank_bracket.apply(imp_aggregator)
overall_rankimps_agg
```

This results in the following:

	rank_bracket	avg_imps	imps_median	imps_lq	imps_uq	n_count
0	1.0	784.795848	132.0	43.25	2970.35	578.0
1	2.0	991.002639	153.0	24.00	2930.20	1137.0
2	3.0	1816.848187	159.5	35.00	6628.15	1324.0
3	4.0	2234.595041	151.0	22.00	8387.55	1452.0
4	5.0	2529.486692	153.0	22.00	9174.60	1315.0
...	...	...	...	...	...	...
97	98.0	36.000000	36.0	20.50	63.90	2.0
98	99.0	5.666667	6.0	5.00	6.90	3.0
99	105.0	1.000000	1.0	1.00	1.00	1.0
100	108.0	1.000000	1.0	1.00	1.00	1.0
101	110.0	4.000000	4.0	4.00	4.00	1.0

In this case, we went with the 25th and 95th percentiles. The lower percentile number doesn't matter as much as we're far more interested in finding queries with averages beyond the 95th percentile. If we can do that, we have a juicy keyword. Quick note, in data science, a percentile is known as a "quantile."

Could we make a table for each and every segment? For example, show the statistics for impressions by ranking position by section. Yes, of course, you could, and in theory, it would provide a more contextual analysis of queries performed vs. their segment average. The deciding factor on whether to do so or not depends on how many data points (i.e., ranked queries) you have for each rank bracket to make it worthwhile (i.e., statistically robust). You'd want at least 30 data points in each to go that far.

Compare Impression Levels to the Average

Okay, now let's left join (think vlookup or index match) the table from the previous set and then join it to the segmented data. Then we have a dataframe that shows the query data vs. the expected average and upper quantile.

Join accessories_rankimps_agg onto accessory_queries by rank_bracket:

```
query_quantile_stats = gsc_segment_strdetect.merge(overall_rankimps_agg, on
=['rank_bracket'], how='left')
query_quantile_stats
```

This results in the following:

	query	clicks	impressions	ctr	position	segment	rank_bracket	avg_imps	imps_median	imps_lq	imps_uq	n_count
0	ps4 cd keys	17	206	8.25	13.40	Retailer	13.0	5619.412587	50.0	6.00	12414.50	286.0
1	cheap ps4 cd keys	13	40	32.5	9.38	Retailer	9.0	2823.633851	72.0	8.25	12480.15	1158.0
2	ps4 cd key	12	34	35.29	19.44	Retailer	19.0	1385.688889	9.0	1.00	8170.05	90.0
3	cheap cd keys ps4	11	21	52.38	8.71	Retailer	9.0	2823.633851	72.0	8.25	12480.15	1158.0
4	xbox cd keys	8	89	8.99	13.46	Retailer	13.0	5619.412587	50.0	6.00	12414.50	286.0
...	...	...	...	...	...	...	...	...	...	...	...	...
15862	nintendo switch lite on credit	48	94	51.06	2.73	Console	3.0	1816.848187	159.5	35.00	6628.15	1324.0
15863	nintendo switch limited edition	47	3430	1.37	13.27	Console	13.0	5619.412587	50.0	6.00	12414.50	286.0
15864	the cheapest nintendo switch	47	1182	3.98	5.16	Console	5.0	2529.486692	153.0	22.00	9174.60	1315.0
15865	nintendo switch lite grey bundle	47	721	6.52	8.03	Console	8.0	2164.032761	59.0	8.00	10308.25	1282.0
15866	where to buy a nintendo switch uk	47	620	7.58	11.03	Console	11.0	4645.429894	22.0	5.00	13906.00	756.0

Explore the Data

Now you might be wondering, how many keywords are punching above and below their weight (i.e., above and below their quantile limits relative to their ranking position) and what are those keywords?

Get the number of keywords with high volumes of impressions:

```
query_stats_uq = query_quantile_stats.loc[query_quantile_stats.impressions
> query_quantile_stats.imps_uq]
query_stats_uq['query'].count()
```

This results in the following:

```
8390
```

Get the number of keywords with impressions and ranking beyond page 1:

```
query_stats_uq_p2b = query_quantile_stats.loc[(query_quantile_stats.
impressions > query_quantile_stats.imps_uq) & (query_quantile_stats.rank_
bracket > 10)]
query_stats_uq_p2b['query'].count()
```

This results in the following:

```
2510
```

Depending on your resources, you may wish to track all 8390 keywords or just the 2510. Let's see how the distribution of impressions looks visually across the range of ranking positions:

```
import seaborn as sns
import matplotlib.pyplot as plt
from pylab import savefig
```

Set the plot size:

```
sns.set(rc={'figure.figsize':(15, 6)})
```

Plot impressions vs. rank_bracket:

```
imprank_plt = sns.relplot(x = "rank_bracket", y = "impressions",
                hue = "quantiled", style = "quantiled",
                kind = "line", data = overall_rankimps_agg_long)
```

Save Figure 2-1 to a file for your PowerPoint deck or others:

```
imprank_plt.savefig("images/imprank_plt.png")
```

What's interesting is the upper quantile impression keywords are not all in the top 10, but many are on pages 2, 4, and 6 of the SERP results (Figure 2-1). This indicates that the site is either targeting the high-volume keywords but not doing a good job of achieving a high ranking position or not targeting these high-volume phrases.

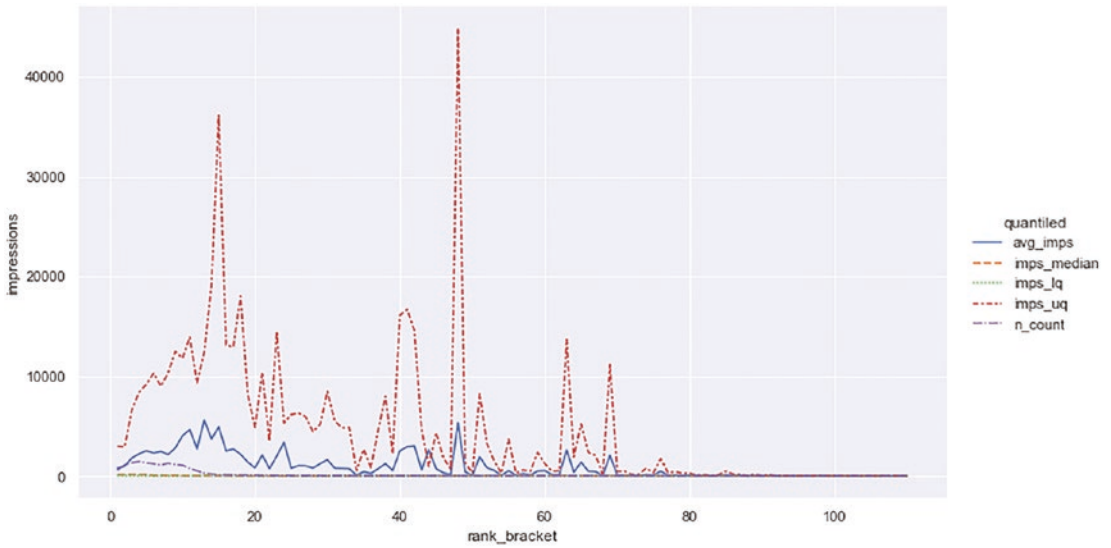

Figure 2-1. *Line chart showing GSC impressions per ranking position bracket for each distribution quantile*

Let's break this segment down.

Plot impressions vs. rank_bracket by segment:

```
imprank_seg = sns.relplot(x="rank_bracket", y="impressions",
                hue="quantiled", col="segment",
                kind="line", data = overall_rankimps_agg_long, facet_
                kws=dict(sharex=False))
```

Export the file:

```
imprank_seg.savefig("images/imprank_seg.png")
```

Most of the high impression keywords are in Accessories, Console, and of course Top 1000 (Figure 2-2).

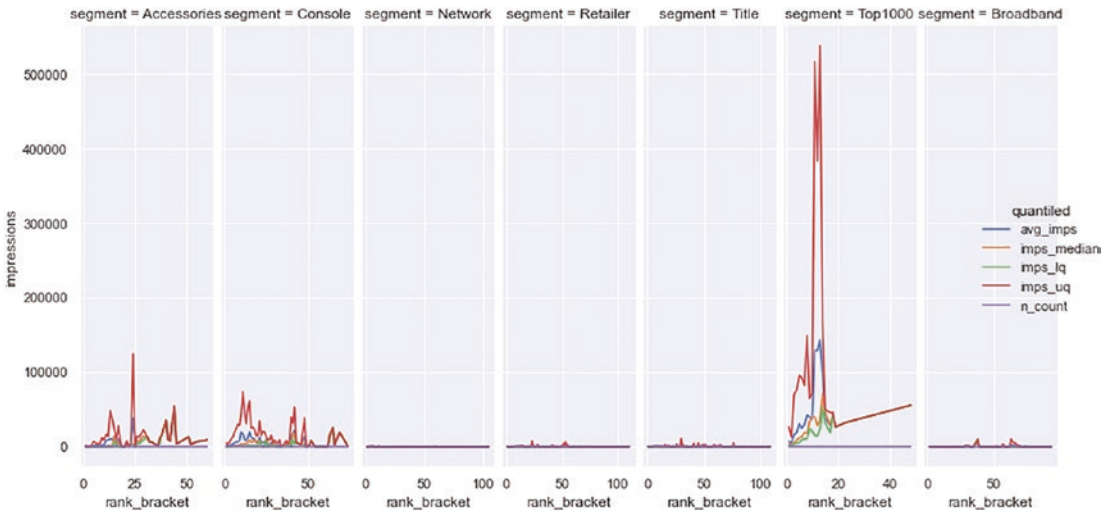

Figure 2-2. *Line chart showing GSC impressions per ranking position bracket for each distribution quantile faceted by segment*

Export Your High Value Keyword List

Now that you have your keywords, simply filter and export to CSV.

Export the dataframe to CSV:

```
query_stats_uq_p2b.to_csv('exports/query_stats_uq_p2b_TOTRACK.csv')
```

Activation

Now that you've identified high impression value keywords, you can

- Replace or add those keywords to the ones you're currently tracking and campaigning
- Research the content experience required to rank on the first page
- Think about how to integrate these new targets into your strategy
- Explore levels of on-page optimization for these keywords, including where there are low-hanging fruit opportunities to more effectively interlink landing pages targeting these keywords (such as through blog posts or content pages)

- Consider whether increasing external link popularity (through content marketing and PR) across these new landing pages is appropriate

Obviously, the preceding list is reductionist, and yet as a minimum, you have better nonbrand targets to better serve your SEO campaign.

Google Trends

Google Trends is another (free) third-party data source, which shows time series data (data points over time) up to the last five years for any search phrase that has demand. Google Trends can also help you compare whether a search is on the rise (or decline) while comparing it to other search phrases. It can be highly useful for forecasting.

Although no Google Trends API exists, there are packages in Python (i.e., pytrends) that can automate the extraction of this data as we'll see as follows:

```
import pandas as pd
from pytrends.request import TrendReq
import time
```

Single Keyword

Now that you've identified high impression value keywords, you can see how they've trended over the last five years:

```
kw_list = ["Blockchain"]
pytrends.build_payload(kw_list, cat=0, timeframe='today 5-y', geo='GB',
gprop='')
pytrends.interest_over_time()
```

This results in the following:

date	Blockchain	isPartial
2016-06-19	8	False
2016-06-26	6	False
2016-07-03	7	False
2016-07-10	6	False
2016-07-17	7	False
...	...	...
2021-05-09	42	False
2021-05-16	55	False
2021-05-23	43	False
2021-05-30	30	False
2021-06-06	34	True

260 rows × 2 columns

Multiple Keywords

As you can see earlier, you get a dataframe with the date, the keyword, and the number of hits (scaled from 0 to 100), which is great, and what if you had 10,000 keywords that you wanted trends for?

In that case, you'd want a for loop to query the search phrases one by one and stick them all into a dataframe like so:

Read in your target keyword data:

```
csv_raw = pd.read_csv('data/your_keyword_file.csv')
keywords_df = csv_raw[['query']]
keywords_list = keywords_df['query'].values.tolist()
keywords_list
```

Here's the output of what keywords_list looks like:

```
['nintendo switch',
 'ps4',
 'xbox one controller',
 'xbox one',
 'xbox controller',
 'ps4 vr',
 'Ps5' ...]
```

Let's now get Google Trends data for all of your keywords in one dataframe:

```
dataset = []
exceptions = []

for q in keywords_list:
    q_lst = [q]
    try:
        pytrends.build_payload(kw_list=q_lst, timeframe='today 5-y',
        geo='GB', gprop='')
        data = pytrends.interest_over_time()
        data = data.drop(labels=['isPartial'],axis='columns')
        dataset.append(data)
        time.sleep(3)
    except:
        exceptions.append(q_lst)

gtrends_long = pd.concat(dataset, axis=1)
```

This results in the following:

date	nintendo switch	ps4	xbox one controller	xbox one	xbox controller	ps4 vr	ps5	ps5 console	ps5 pre order	xbox series x
2016-06-19	0	30	30	37	29	9	0	0	0	0
2016-06-26	0	31	26	34	31	6	0	0	0	0
2016-07-03	0	31	26	36	26	7	0	0	0	0
2016-07-10	0	26	23	31	26	6	0	0	0	0
2016-07-17	0	27	17	29	19	4	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...
2021-05-09	20	24	18	14	28	12	13	14	1	8
2021-05-16	18	22	14	15	29	9	12	10	0	8
2021-05-23	19	22	17	14	30	7	10	10	1	7
2021-05-30	20	23	13	15	33	9	10	9	1	6
2021-06-06	17	20	12	14	27	7	11	10	1	7

260 rows × 10 columns

Let's convert to long format:

```
gtrends_long = gtrends_raw.melt(id_vars=['date'], var_name = 'query',
value_name = 'hits')
gtrends_long
```

This results in the following:

	date	query	hits
0	2016-06-19	nintendo switch	0
1	2016-06-26	nintendo switch	0
2	2016-07-03	nintendo switch	0
3	2016-07-10	nintendo switch	0
4	2016-07-17	nintendo switch	0
...	...	...	...
3115	2021-05-09	id	hits
3116	2021-05-16	id	hits
3117	2021-05-23	id	hits
3118	2021-05-30	id	hits
3119	2021-06-06	id	hits

Looking at Google Trends raw, we now have data in long format showing

- Date
- Keyword
- Hits

Let's visualize some of these over time. We start by subsetting the dataframe:

```
k_list = ['ps5',  'xbox one',  'ps4',  'xbox series x', 'nintendo switch']
keyword_gtrends = gtrends_long.loc[gtrends_long['query'].isin(k_list)]
keyword_gtrends
```

This results in the following:

	date	query	hits
0	2016-06-19	nintendo switch	0
1	2016-06-26	nintendo switch	0
2	2016-07-03	nintendo switch	0
3	2016-07-10	nintendo switch	0
4	2016-07-17	nintendo switch	0
...	...	...	...
2595	2021-05-09	xbox series x	8
2596	2021-05-16	xbox series x	7
2597	2021-05-23	xbox series x	6
2598	2021-05-30	xbox series x	6
2599	2021-06-06	xbox series x	7

Visualizing Google Trends

Okay, so we're now ready to plot the time series data as a chart, starting with the library import:

```
import seaborn as sns
```

Set the plot size:

```
sns.set(rc={'figure.figsize':(15, 6)})
```

Build and plot the chart:

```
keyword_gtrends_plt = sns.lineplot(data = keyword_gtrends, x = 'date', y =
'hits', hue = 'query')
```

Save the image to a file for your PowerPoint deck or others:

```
keyword_gtrends_plt.figure.savefig("images/keyword_gtrends.png")
keyword_gtrends_plt
```

Here, we can see that the "ps5" and "xbox series x" show a near identical trend which ramp up significantly, while other models are fairly stable and seasonal until the arrival of the new models.

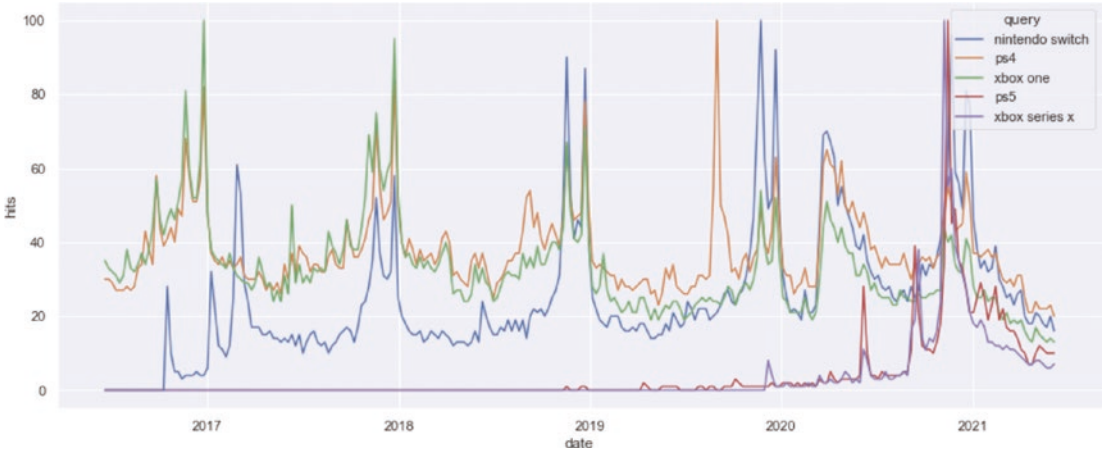

Figure 2-3. *Time series plot of Google Trends keywords*

Forecast Future Demand

While it's great to see what's happened in the last five years, it's also great to see what might happen in the future. Thankfully, Python provides the tools to do so. The most obvious use cases for forecasts are client pitches and reporting.

Exploring Your Data

```
import pandas as pd
from statsmodels.tsa.statespace.sarimax import SARIMAX
from statsmodels.graphics.tsaplots import plot_acf,plot_pacf
from statsmodels.tsa.seasonal import seasonal_decompose
from sklearn.metrics import mean_squared_error
from statsmodels.tools.eval_measures import rmse
import warnings
warnings.filterwarnings("ignore")
from pmdarima import auto_arima
```

Import Google Trends data:

```
df = pd.read_csv("exports/keyword_gtrends_df.csv", index_col=0)
df.head()
```

This results in the following:

	date	query	hits
1815	2021-05-09	ps5	12
1816	2021-05-16	ps5	11
1817	2021-05-23	ps5	10
1818	2021-05-30	ps5	10
1819	2021-06-06	ps5	10

As we'd expect, the data from Google Trends is a very simple time series with date, query, and hits spanning a five-year period. Time to format the dataframe to go from long to wide:

```
df_unstacked = ps_trends.set_index(["date", "query"]).unstack(level=-1)
df_unstacked.columns.set_names(['hits', 'query'], inplace=True)
ps_unstacked = df_unstacked.droplevel('hits', axis=1)
ps_unstacked.columns = [c.replace(' ', '_') for c in ps_unstacked.columns]
ps_unstacked = ps_unstacked.reset_index()
ps_unstacked.head()
```

This results in the following:

	date	ps4	ps5
0	2016-06-19	30	0
1	2016-06-26	30	0
2	2016-07-03	29	0
3	2016-07-10	27	0
4	2016-07-17	27	0

We no longer have a hits column as these are the values of the queries in their respective columns. This format is not only useful for SARIMA[2] (which we will be exploring here) but also neural networks such as long short-term memory (LSTM). Let's plot the data:

```
ps_unstacked.plot(figsize=(10,5))
```

From the plot (Figure 2-4), you'll note that the profiles of both "PS4" and "PS5" are different.

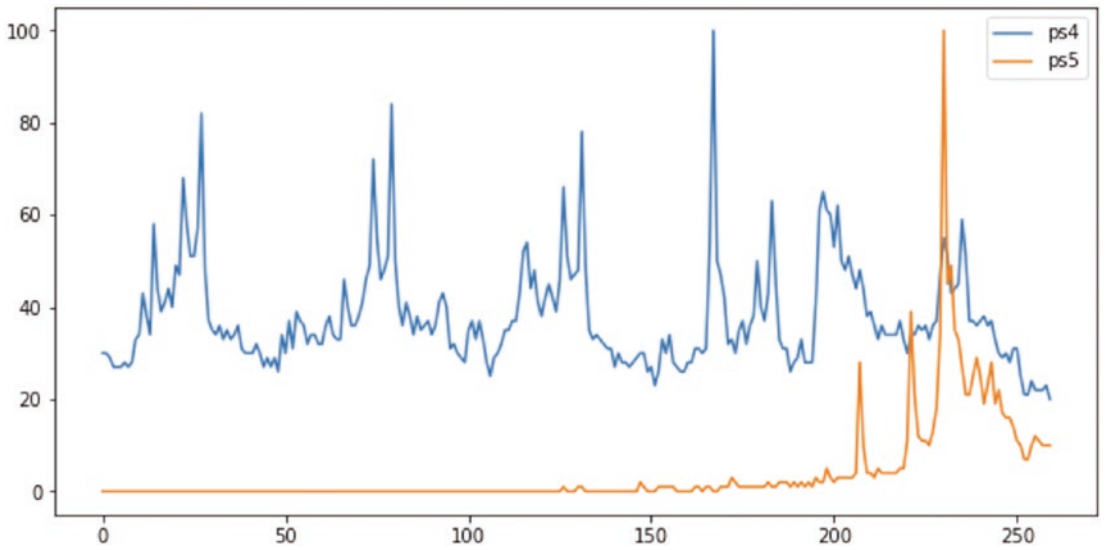

Figure 2-4. *Time series plot of both ps4 and ps5*

[2] Seasonal Autoregressive Integrated Moving Average

For the nongamers among you, "PS4" is the fourth generation of the Sony PlayStation console, and "PS5" the fifth. "PS4" searches are highly seasonal and have a regular pattern apart from the end when the "PS5" emerged. The "PS5" didn't exist five years ago, which would explain the absence of trend in the first four years of the preceding plot.

Decomposing the Trend

Let's now decompose the seasonal (or nonseasonal) characteristics of each trend:

```
ps_unstacked.set_index("date", inplace=True)
ps_unstacked.index = pd.to_datetime(ps_unstacked.index)

query_col = 'ps5'
a = seasonal_decompose(ps_unstacked[query_col], model = "add")
a.plot();
```

Figure 2-5 shows the time series data and the overall smoothed trend showing it rises from 2020.

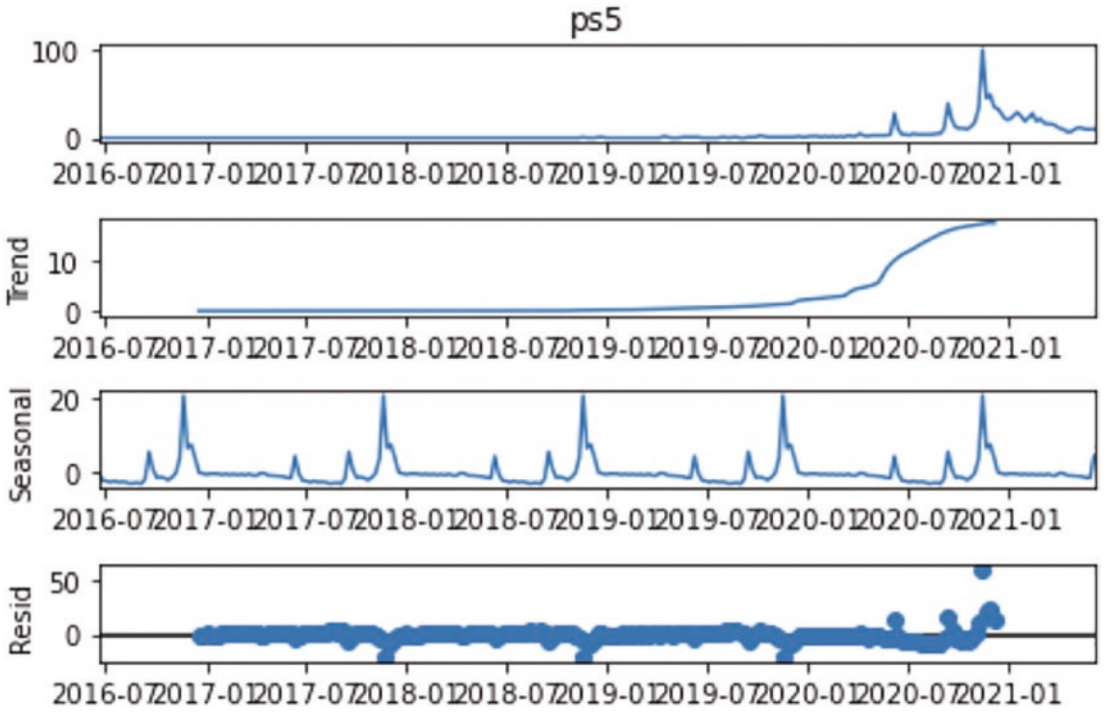

Figure 2-5. *Decomposition of the ps5 time series*

The seasonal trend box shows repeated peaks which indicates that there is seasonality from 2016, although it doesn't seem particularly reliable given how flat the time series is from 2016 until 2020. Also suspicious is the lack of noise as the seasonal plot shows a virtually uniform pattern repeating periodically.

The Resid (which stands for "Residual") shows any pattern of what's left of the time series data after accounting for seasonality and trend, which in effect is nothing until 2020 as it's at zero most of the time.

For "ps4," see Figure 2-6.

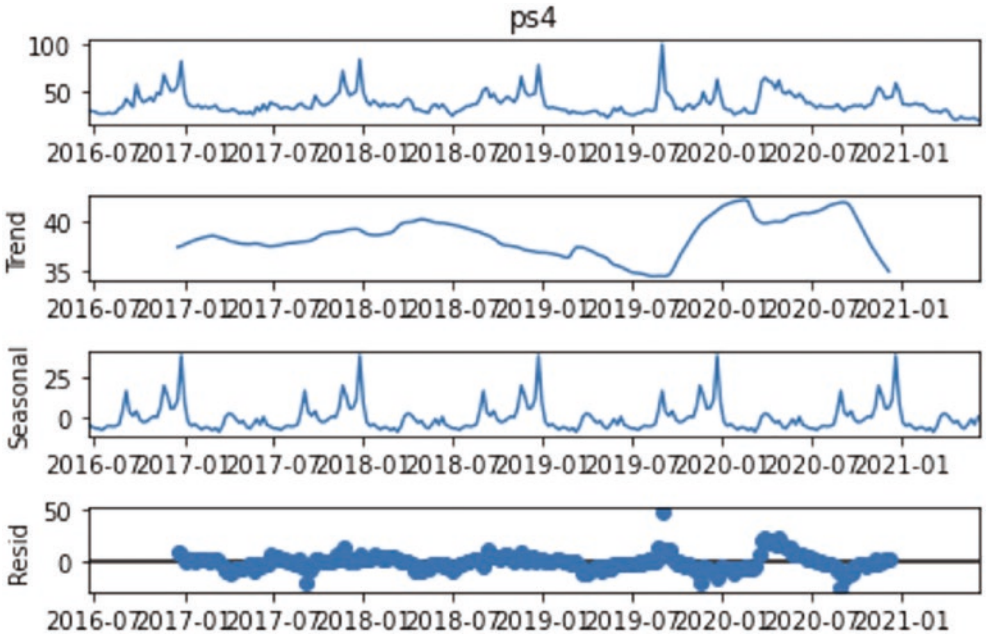

Figure 2-6. *Decomposition of the ps4 time series*

We can see fluctuation over the short term (Seasonality) and long term (Trend), with some noise (Resid). The next step is to use the augmented Dickey-Fuller method (ADF) to statistically test whether a given time series is stationary or not:

```
from pmdarima.arima import ADFTest
adf_test = ADFTest(alpha=0.05)
adf_test.should_diff(ps_unstacked[query_col])
```

```
PS4: (0.09760939899434763, True)
PS5: (0.01, False)
```

We can see that the p-value of "PS5" shown earlier is more than 0.05, which means that the time series data is not stationary and therefore needs differencing. "PS4" on the other hand is less than 0.05 at 0.01, meaning it's stationery and doesn't require differencing.

The point of all this is to understand the parameters that would be used if we were manually building a model to forecast Google searches.

Fitting Your SARIMA Model

Since we'll be using automated methods to estimate the best fit model parameters (later), we're not going to estimate the number of parameters for our SARIMA model.

To estimate the parameters for our SARIMA model, note that we set m to 52 as there are 52 weeks in a year which is how the periods are spaced in Google Trends. We also set all of the parameters to start at 0 so that we can let the auto_arima do the heavy lifting and search for the values that best fit the data for forecasting:

```
ps5_s = auto_arima(ps_unstacked['ps4'],
           trace=True,
           m=52, #there are 52 period per season (weekly data)
           start_p=0,
           start_d=0,
           start_q=0,
           seasonal=False)
```

This results in the following:

```
Performing stepwise search to minimize aic

 ARIMA(3,0,3)(0,0,0)[0]             : AIC=1842.301, Time=0.26 sec
 ARIMA(0,0,0)(0,0,0)[0]             : AIC=2651.089, Time=0.01 sec
 ARIMA(1,0,0)(0,0,0)[0]             : AIC=1865.936, Time=0.02 sec
 ARIMA(0,0,1)(0,0,0)[0]             : AIC=2370.569, Time=0.05 sec
 ARIMA(2,0,3)(0,0,0)[0]             : AIC=1845.911, Time=0.12 sec
 ARIMA(3,0,2)(0,0,0)[0]             : AIC=1845.959, Time=0.16 sec
 ARIMA(4,0,3)(0,0,0)[0]             : AIC=1838.349, Time=0.34 sec
 ARIMA(4,0,2)(0,0,0)[0]             : AIC=1846.701, Time=0.22 sec
 ARIMA(5,0,3)(0,0,0)[0]             : AIC=1843.754, Time=0.25 sec
 ARIMA(4,0,4)(0,0,0)[0]             : AIC=1842.801, Time=0.27 sec
 ARIMA(3,0,4)(0,0,0)[0]             : AIC=1841.447, Time=0.36 sec
 ARIMA(5,0,2)(0,0,0)[0]             : AIC=1841.893, Time=0.24 sec
 ARIMA(5,0,4)(0,0,0)[0]             : AIC=1845.734, Time=0.29 sec
 ARIMA(4,0,3)(0,0,0)[0] intercept   : AIC=1824.187, Time=0.82 sec
 ARIMA(3,0,3)(0,0,0)[0] intercept   : AIC=1824.769, Time=0.34 sec
 ARIMA(4,0,2)(0,0,0)[0] intercept   : AIC=1826.970, Time=0.34 sec
 ARIMA(5,0,3)(0,0,0)[0] intercept   : AIC=1826.789, Time=0.44 sec
```

```
 ARIMA(4,0,4)(0,0,0)[0] intercept   : AIC=1827.114, Time=0.43 sec
 ARIMA(3,0,2)(0,0,0)[0] intercept   : AIC=1831.587, Time=0.32 sec
 ARIMA(3,0,4)(0,0,0)[0] intercept   : AIC=1825.359, Time=0.42 sec
 ARIMA(5,0,2)(0,0,0)[0] intercept   : AIC=1827.292, Time=0.40 sec
 ARIMA(5,0,4)(0,0,0)[0] intercept   : AIC=1829.109, Time=0.51 sec

Best model:  ARIMA(4,0,3)(0,0,0)[0] intercept
Total fit time: 6.601 seconds
```

The preceding printout shows that the parameters that get the best results are

```
PS4: ARIMA(4,0,3)(0,0,0)
PS5: ARIMA(3,1,3)(0,0,0)
```

The PS5 estimate is further detailed when printing out the model summary:

```
ps5_s.summary()
```

This results in the following:

SARIMAX Results

Dep. Variable:	y	**No. Observations:**	260
Model:	SARIMAX(4, 0, 3)	**Log Likelihood**	-903.094
Date:	Fri, 30 Jul 2021	**AIC**	1824.187
Time:	13:02:02	**BIC**	1856.233
Sample:	0	**HQIC**	1837.070
	- 260		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
intercept	8.7756	2.661	3.298	0.001	3.561	13.990
ar.L1	1.2577	0.325	3.872	0.000	0.621	1.894
ar.L2	-0.1521	0.701	-0.217	0.828	-1.527	1.223
ar.L3	-0.8086	0.652	-1.241	0.215	-2.086	0.469
ar.L4	0.4698	0.240	1.961	0.050	0.000	0.939
ma.L1	-0.6256	0.316	-1.979	0.048	-1.245	-0.006
ma.L2	-0.2806	0.455	-0.617	0.537	-1.172	0.611
ma.L3	0.7784	0.294	2.648	0.008	0.202	1.355
sigma2	60.0816	3.039	19.771	0.000	54.125	66.038

Ljung-Box (L1) (Q):	0.02	**Jarque-Bera (JB):**	1449.21
Prob(Q):	0.90	**Prob(JB):**	0.00
Heteroskedasticity (H):	0.54	**Skew:**	2.24
Prob(H) (two-sided):	0.00	**Kurtosis:**	13.66

What's happening is the function is looking to minimize the probability of error measured by both the Akaike information criterion (AIC) and Bayesian information criterion:

```
AIC = -2Log(L) + 2(p + q + k + 1)
```

such that L is the likelihood of the data, k = 1 if $c \neq 0$, and k = 0 if c = 0.

```
BIC = AIC + [log(T) - 2] + (p + q + k + 1)
```

By minimizing AIC and BIC, we get the best estimated parameters for p and q.

Test the Model

Now that we have the parameters, we can now start making forecasts for both products:

```
ps4_order = ps4_s.get_params()['order']
ps4_seasorder = ps4_s.get_params()['seasonal_order']

ps5_order = ps5_s.get_params()['order']
ps5_seasorder = ps5_s.get_params()['seasonal_order']

params = {
    "ps4": {"order": ps4_order, "seasonal_order": ps4_seasorder},
    "ps5": {"order": ps5_order, "seasonal_order": ps5_seasorder}
}
```

Create an empty list to store the forecast results:

```
results = []
fig, axs = plt.subplots(len(X.columns), 1, figsize=(24, 12))
```

Iterate through the columns to fit the best SARIMA model:

```
for i, col in enumerate(X.columns):
    arima_model = SARIMAX(train_data[col],
                          order = params[col]["order"],
                          seasonal_order = params[col]["seasonal_order"])
    arima_result = arima_model.fit()
```

Make forecasts:

```
    arima_pred = arima_result.predict(start = len(train_data),
                                      end = len(X)-1, typ="levels")\
                             .rename("ARIMA Predictions")
```

Plot predictions:

```
test_data[col].plot(figsize = (8,4), legend=True, ax=axs[i])
arima_pred.plot(legend = True, ax=axs[i])

arima_rmse_error = rmse(test_data[col], arima_pred)
mean_value = X[col].mean()

results.append((col, arima_pred, arima_rmse_error, mean_value))
print(f'Column: {col} --> RMSE Error: {arima_rmse_error} - Mean: {mean_
value}\n')
```

This results in the following:

```
Column: ps4 --> RMSE Error: 8.626764032898576 - Mean: 37.83461538461538
Column: ps5 --> RMSE Error: 27.552818032476257 - Mean: 3.973076923076923
```

For ps4, the forecasts are pretty accurate from the beginning until March when the search values start to diverge (Figure 2-7), while the ps5 forecasts don't appear to be very good at all, which is unsurprising.

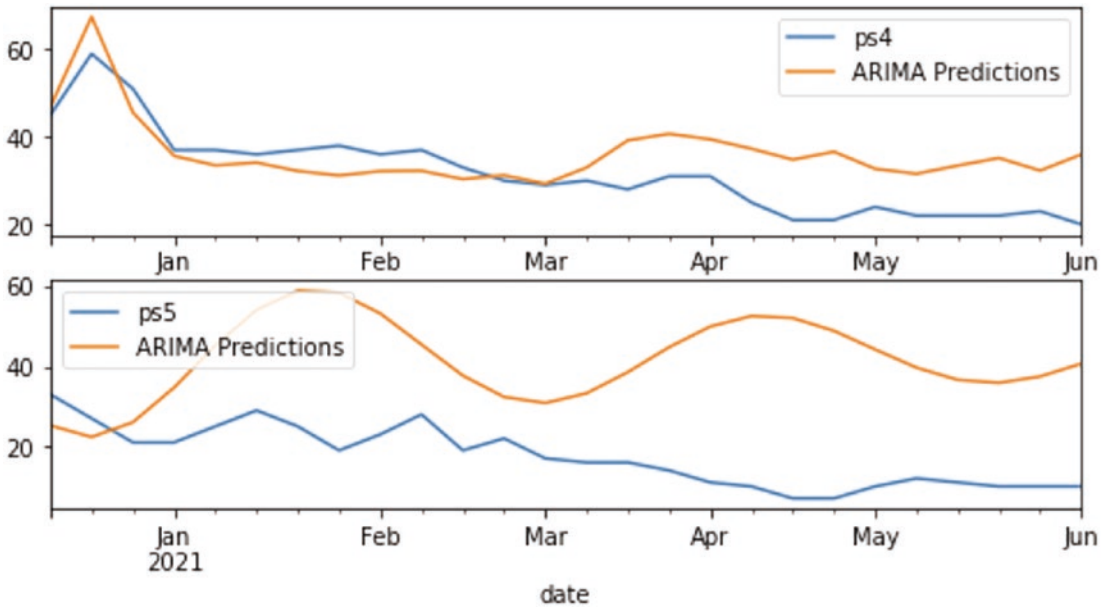

Figure 2-7. *Time series line plots comparing forecasts and actual data for both ps4 and ps5*

The forecasts show the models are good when there is enough history until they suddenly change like they have for PS4 from March onward. For PS5, the models are hopeless virtually from the get-go. We know this because the Root Mean Squared Error (RMSE) is 8.62 for PS4 which is more than a third of the PS5 RMSE of 27.5, which, given Google Trends varies from 0 to 100, is a 27% margin of error.

Forecast the Future

At this point, we'll now make the foolhardy attempt to forecast the future based on the data we have to date:

```
oos_train_data = ps_unstacked
oos_train_data.tail()
```

This results in the following:

date	ps4	ps5
2021-05-09	22	12
2021-05-16	22	11
2021-05-23	22	10
2021-05-30	23	10
2021-06-06	20	10

As you can see from the preceding table extract, we're now using all available data. Now we shall predict the next six months (defined as 26 weeks) in the following code:

```
oos_results = []
weeks_to_predict = 26
fig, axs = plt.subplots(len(ps_unstacked.columns), 1, figsize=(24, 12))
```

Again, iterate through the columns to fit the best model each time:

```
for i, col in enumerate(ps_unstacked.columns):
    s = auto_arima(oos_train_data[col], trace=True)
```

```
    oos_arima_model = SARIMAX(oos_train_data[col],
                          order = s.get_params()['order'],
                          seasonal_order = s.get_params()['seasonal_
                          order'])
    oos_arima_result = oos_arima_model.fit()
```

Make forecasts:

```
    oos_arima_pred = oos_arima_result.predict(start = len(oos_train_data),
                                      end = len(oos_train_data) + weeks_to_
                                      predict, typ="levels").rename("ARIMA
                                      Predictions")
```

Plot predictions:

```
    oos_arima_pred.plot(legend = True, ax=axs[i])
    axs[i].legend([col]);
    mean_value = ps_unstacked[col].mean()

    oos_results.append((col, oos_arima_pred, mean_value))
    print(f'Column: {col} - Mean: {mean_value}\n')
```

Here's the output:

```
Performing stepwise search to minimize aic
 ARIMA(2,0,2)(0,0,0)[0] intercept   : AIC=1829.734, Time=0.21 sec
 ARIMA(0,0,0)(0,0,0)[0] intercept   : AIC=1999.661, Time=0.01 sec
 ARIMA(1,0,0)(0,0,0)[0] intercept   : AIC=1827.518, Time=0.03 sec
 ARIMA(0,0,1)(0,0,0)[0] intercept   : AIC=1882.388, Time=0.05 sec
 ARIMA(0,0,0)(0,0,0)[0]             : AIC=2651.089, Time=0.01 sec
 ARIMA(2,0,0)(0,0,0)[0] intercept   : AIC=1829.254, Time=0.04 sec
 ARIMA(1,0,1)(0,0,0)[0] intercept   : AIC=1829.136, Time=0.09 sec
 ARIMA(2,0,1)(0,0,0)[0] intercept   : AIC=1829.381, Time=0.26 sec
 ARIMA(1,0,0)(0,0,0)[0]             : AIC=1865.936, Time=0.02 sec

Best model:  ARIMA(1,0,0)(0,0,0)[0] intercept
Total fit time: 0.722 seconds
Column: ps4 - Mean: 37.83461538461538
```

```
Performing stepwise search to minimize aic
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=1657.990, Time=0.19 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=1696.958, Time=0.01 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=1673.340, Time=0.04 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=1666.878, Time=0.05 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=1694.967, Time=0.01 sec
 ARIMA(1,1,2)(0,0,0)[0] intercept   : AIC=1656.899, Time=0.14 sec
 ARIMA(0,1,2)(0,0,0)[0] intercept   : AIC=1663.729, Time=0.04 sec
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=1656.787, Time=0.07 sec
 ARIMA(2,1,1)(0,0,0)[0] intercept   : AIC=1656.351, Time=0.16 sec
 ARIMA(2,1,0)(0,0,0)[0] intercept   : AIC=1672.668, Time=0.04 sec
 ARIMA(3,1,1)(0,0,0)[0] intercept   : AIC=1657.661, Time=0.11 sec
 ARIMA(3,1,0)(0,0,0)[0] intercept   : AIC=1670.698, Time=0.05 sec
 ARIMA(3,1,2)(0,0,0)[0] intercept   : AIC=1653.392, Time=0.33 sec
 ARIMA(4,1,2)(0,0,0)[0] intercept   : AIC=inf, Time=0.40 sec
 ARIMA(3,1,3)(0,0,0)[0] intercept   : AIC=1643.872, Time=0.45 sec
 ARIMA(2,1,3)(0,0,0)[0] intercept   : AIC=1659.698, Time=0.23 sec
 ARIMA(4,1,3)(0,0,0)[0] intercept   : AIC=inf, Time=0.48 sec
 ARIMA(3,1,4)(0,0,0)[0] intercept   : AIC=inf, Time=0.47 sec
 ARIMA(2,1,4)(0,0,0)[0] intercept   : AIC=1645.994, Time=0.52 sec
 ARIMA(4,1,4)(0,0,0)[0] intercept   : AIC=1647.585, Time=0.56 sec
 ARIMA(3,1,3)(0,0,0)[0]             : AIC=1641.790, Time=0.37 sec
 ARIMA(2,1,3)(0,0,0)[0]             : AIC=1648.325, Time=0.38 sec
 ARIMA(3,1,2)(0,0,0)[0]             : AIC=1651.416, Time=0.24 sec
 ARIMA(4,1,3)(0,0,0)[0]             : AIC=1650.077, Time=0.59 sec
 ARIMA(3,1,4)(0,0,0)[0]             : AIC=inf, Time=0.58 sec
 ARIMA(2,1,2)(0,0,0)[0]             : AIC=1656.290, Time=0.10 sec
 ARIMA(2,1,4)(0,0,0)[0]             : AIC=1644.099, Time=0.38 sec
 ARIMA(4,1,2)(0,0,0)[0]             : AIC=inf, Time=0.38 sec
 ARIMA(4,1,4)(0,0,0)[0]             : AIC=1645.756, Time=0.56 sec

Best model:  ARIMA(3,1,3)(0,0,0)[0]
Total fit time: 7.954 seconds
Column: ps5 - Mean: 3.973076923076923
```

This time, we automated the finding of the best-fitting parameters and fed that directly into the model.

The forecasts don't look great (Figure 2-8) because there's been a lot of change in the last few weeks of the data; however, that's in the case of those two keywords.

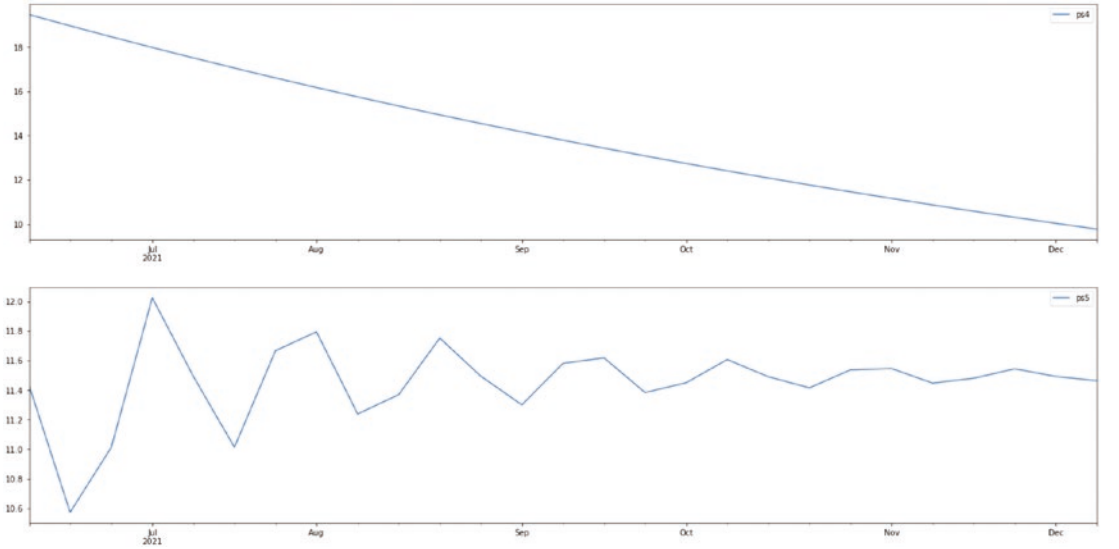

Figure 2-8. *Out-of-sample forecasts of Google Trends for ps4 and ps5*

The forecast quality will be dependent on how stable the historic patterns are and will obviously not account for unforeseeable events like COVID-19.

Export your forecasts:

```
df_pred = pd.concat([pd.Series(res[1]) for res in oos_results], axis=1)
df_pred.columns = [x + str('_preds') for x in ps_unstacked.columns]
df_pred.to_csv('your_forecast_data.csv')
```

What we learn here is where forecasting using statistical models are useful or are likely to add value for forecasting, particularly in automated systems like dashboards, that is, when there's historical data and not when there is a sudden spike like PS5.

Clustering by Search Intent

Search intent is the meaning behind the search queries that users of Google type in when searching online. So you may have the following queries:

"Trench coats"

"Ladies trench coats"

"Life insurance"

"Trench coats" will share the same search intent as "Ladies trench coats" but won't share the same intent as "Life insurance." To work this out, a simple comparison of the top 10 ranking sites for both search phrases in Google will offer a strong suggestion of what Google thinks of the search intent between the two phrases.

It's not a perfect method, but it works well because you're using the ranking results which are a distillation of everything Google has learned to date on what content satisfies the search intent of the search query (based upon the trillions of global searches per year). Therefore, it's reasonable to surmise that if two search queries have similar enough SERPs, then the search intent is shared between keywords.

This is useful for a number of reasons:

- *Rank tracking costs*: If your budget is limited, then knowing the search intent means you can avoid incurring further expense by not tracking keywords with the same intent as those you're tracking. This comes with a risk as consumers change and the keyword not tracked may not share the same intent anymore.
- *Core updates*: With changing consumer search patterns come changing intents, which means you can see if keywords change clusters or not by comparing the search intent clusters of keywords before and after the update, which will help inform your response.
- *Keyword content mapping*: Knowing the intent means you can successfully map keywords to landing pages. This is especially useful in ensuring your site architecture consists of landing pages which map to user search demand.
- *Paid search ads*: Good keyword content mappings also mean you can improve the account structure and resulting quality score of your paid search activity.

Starting Point

Okay, time to cluster. We'll assume you already have the top 100 SERPs[3] results for each of your keywords stored as a Python dataframe "serps_input." The data is easily obtained from a rank tracking tool, especially if they have an API:

```
serps_input
```

This results in the following:

	keyword	rank	url	se_results_count
0	xbox one x controller	1	https://www.xbox.com/en-GB/accessories	144000000
1	xbox one x controller	2	None	144000000
2	xbox one x controller	3	https://www.xbox.com/en-GB/accessories/control...	144000000
3	xbox one x controller	4	https://www.argos.co.uk/browse/technology/vide...	144000000
4	xbox one x controller	5	https://www.game.co.uk/en/accessories/xbox-one...	144000000
5	xbox one x controller	6	https://www.currys.co.uk/gbuk/xbox-one-control...	144000000
6	xbox one x controller	7	https://www.amazon.co.uk/xbox-one-controller/s...	144000000
7	xbox one x controller	8	None	144000000
8	xbox one x controller	9	https://www.ebay.co.uk/b/Microsoft-Xbox-One-Co...	144000000
9	xbox one x controller	10	https://www.amazon.com/Xbox-Wireless-Controlle...	144000000
10	xbox one x controller	11	https://www.powera.com/product_platform/xbox-one/	144000000
11	xbox one x controller	12	https://www.pricerunner.com/sp/xbox-one-x-cont...	144000000
12	xbox one x controller	13	https://en.wikipedia.org/wiki/Xbox_Wireless_Co...	144000000
13	xbox one x controller	14	https://scufgaming.com/uk/xbox	144000000
14	xbox one x controller	15	https://www.digitaltrends.com/gaming/how-to-sy...	144000000

Here, we're using DataForSEO's SERP API,[4] and we have renamed the column from "rank_absolute" to "rank."

[3] Search Engine Results Pages (SERP)

[4] Available at https://dataforseo.com/apis/serp-api/

Filter Data for Page 1

Because DataForSEO's numbers to individual results are contained within carousels, People Also Ask, etc., we'll want to compare the top 20 results of each SERP to each other to get the approximate results for page 1. We'll also filter out URLs that have the value "None." The programming approach we'll take is "Split-Apply-Combine." What is Split-Apply-Combine?

- Split the dataframe into keyword groups
- Apply the filtering formula to each group
- Combine the keywords of each group

Here it goes:
Split:

```
serps_grpby_keyword = serps_input.groupby("keyword")
```

Apply the function, before combining:

```
def filter_twenty_urls(group_df):
    filtered_df = group_df.loc[group_df['url'].notnull()]
    filtered_df = filtered_df.loc[filtered_df['rank'] <= 20]
    return filtered_df
filtered_serps = serps_grpby_keyword.apply(filter_twenty_urls)
```

Combine and add prefix to column names:

```
normed = normed.add_prefix('normed_')
```

Concatenate with an initial dataframe:

```
filtered_serps_df = pd.concat([filtered_serps],axis=0)
```

Convert Ranking URLs to a String

To compare the SERPs for each keyword, we need to convert the SERPs URL into a string. That's because there's a one (keyword) to many (SERP URLs) relationship. The way we achieve that is by simply concatenating the URL strings for each keyword, using the Split-Apply-Combine approach (again). Convert results to strings using SAC:

```
filtserps_grpby_keyword = filtered_serps_df.groupby("keyword")

def string_serps(df):
    df['serp_string'] = ''.join(df['url'])
    return df

    Combine
strung_serps = filtserps_grpby_keyword.apply(string_serps)
```

Concatenate with an initial dataframe and clean:

```
strung_serps = pd.concat([strung_serps],axis=0)
strung_serps = strung_serps[['keyword', 'serp_string']]#.head(30)
strung_serps = strung_serps.drop_duplicates()
strung_serps
```

This results in the following:

	keyword	serp_string
0	fifa 19 ps4	https://www.amazon.co.uk/Electronic-Arts-22154...
18	gaming broadband	https://www.bt.com/products/broadband/gaminght...
37	playstation vr	https://www.playstation.com/en-gb/ps-vr/https:...
54	ps4	https://www.playstation.com/en-gb/ps4/https://...
72	ps4 console	https://www.game.co.uk/en/hardware/playstation...
91	ps4 controller	https://www.playstation.com/en-gb/ps4/ps4-acce...
109	ps4 controllers	https://www.playstation.com/en-gb/ps4/ps4-acce...
127	ps4 vr	https://www.playstation.com/en-gb/ps-vr/https:...
146	ps5	https://direct.playstation.com/en-us/ps5https:...
162	ps5 console	https://direct.playstation.com/en-us/ps5https:...

Now we have a table showing the keyword and their SERP string, we're ready to compare SERPs. Here's an example of the SERP string for "fifa 19 ps4":

```
strung_serps.loc[1, 'serp_string']
```

This results in the following:

```
'https://www.amazon.co.uk/Electronic-Arts-221545-FIFA-PS4/dp/
B07DLXBGN8https://www.amazon.co.uk/FIFA-19-GAMES/dp/B07DL2SY2Bhttps://
www.game.co.uk/en/fifa-19-2380636https://www.ebay.co.uk/b/FIFA-19-Sony-
PlayStation-4-Video-Games/139973/bn_711513427Ohttps://www.pricerunner.com/
pl/1422-4602670/PlayStation-4-Games/FIFA-19-Compare-Priceshttps://pricespy.
co.uk/games-consoles/computer-video-games/ps4/fifa-19-ps4--p4766432https://
store.playstation.com/en-gb/search/fifa%2019https://www.amazon.com/FIFA-19-
Standard-PlayStation-4/dp/B07DL2SY2Bhttps://www.tesco.com/groceries/
en-GB/products/301926084https://groceries.asda.com/product/ps-4-games/
ps-4-fifa-19/1000076097883https://uk.webuy.com/product-detail/?id=503094
5121916&categoryName=playstation4-software&superCatName=gaming&title=fi
fa-19https://www.pushsquare.com/reviews/ps4/fifa_19https://en.wikipedia.
org/wiki/FIFA_19https://www.amazon.in/Electronic-Arts-Fifa19SEPS4-Fifa-
PS4/dp/B07DVWWF44https://www.vgchartz.com/game/222165/fifa-19/https://www.
metacritic.com/game/playstation-4/fifa-19https://www.johnlewis.com/fifa-19-
ps4/p3755803https://www.ebay.com/p/22045274968'
```

Compare SERP Distance

The SERPs comparison will use string distance techniques which allow us to see how similar or dissimilar one keyword's SERPs are. This technique is similar to how geneticists would compare one DNA sequence to another.

Naturally, we need to get the SERPs into a format ready for Python to compare SERPs. To do this, we need to convert each SERP to a string and then put them side by side. Group the table by keyword:

```
filtserps_grpby_keyword = filtered_serps_df.groupby("keyword")
def string_serps(df):
    df['serp_string'] = ' '.join(df['url'])
    return df
```

Combine using the preceding function:

```
strung_serps = filtserps_grpby_keyword.apply(string_serps)
```

Concatenate with an initial dataframe and clean:

```
strung_serps = pd.concat([strung_serps],axis=0)
```

```
strung_serps = strung_serps[['keyword', 'serp_string']]#.head(30)
strung_serps = strung_serps.drop_duplicates()
#strung_serps['serp_string'] = strung_serps.serp_string.str.
replace("https://www\.", "")
strung_serps.head(15)
```

This results in the following:

	keyword	serp_string
0	beige trench coats	https://www.zalando.co.uk/womens-clothing-coat...
9	blue trench coats	https://www.johnlewis.com/browse/women/womens-...
19	buy ps4	https://www.playstation.com/en-gb/ps4/buy-ps4/...
24	ladies trench coats	https://www.johnlewis.com/browse/women/womens-...
34	mens trench coats	https://uk.burberry.com/mens-trench-coats/ htt...
43	ps4	https://www.playstation.com/en-gb/ps4/ https:/...
51	ps4 console	https://www.game.co.uk/en/hardware/playstation...
60	ps4 vr	https://www.playstation.com/en-gb/ps-vr/ https...
69	ps5	https://direct.playstation.com/en-us/ps5 https...
78	ps5 console	https://www.game.co.uk/playstation-5 https://d...
86	ps5 news	https://www.pushsquare.com/ps5 https://www.pla...
95	ps5 pre order	https://www.playstation.com/en-gb/ps5/buy-now/...
104	trench coats	https://uk.burberry.com/womens-trench-coats/ h...
112	xbox controller	https://www.xbox.com/en-GB/accessories/control...
120	xbox one	https://www.xbox.com/ https://www.xbox.com/en-...

Here, we now have the keywords and their respective SERPs all converted into a string which fits into a single cell. For example, the search result for "beige trench coats" is

```
'https://www.zalando.co.uk/womens-clothing-coats-trench-coats/_beige/
https://www.asos.com/women/coats-jackets/trench-coats/cat/?cid=15143
https://uk.burberry.com/womens-trench-coats/beige/ https://www2.hm.com/
```

```
en_gb/productpage.0751992002.html https://www.hobbs.com/clothing/
coats-jackets/trench/beige/ https://www.zara.com/uk/en/woman-outerwear-
trench-l1202.html https://www.ebay.co.uk/b/Beige-Trench-Coats-for-
Women/63862/bn_7028370345 https://www.johnlewis.com/browse/women/womens-
coats-jackets/trench-coats/_/N-flvZ1z0rnyl https://www.elle.com/uk/fashion/
what-to-wear/articles/g30975/best-trench-coats-beige-navy-black/'
```

Time to put these side by side. What we're effectively doing here is taking a product of the column to itself, that is, squaring it, so that we get all the SERPs combinations possible to put the SERPs side by side.

Add a function to align SERPs:

```
def serps_align(k, df):
    prime_df = df.loc[df.keyword == k]
    prime_df = prime_df.rename(columns = {"serp_string" : "serp_string_a",
    'keyword': 'keyword_a'})
    comp_df = df.loc[df.keyword != k].reset_index(drop=True)
    prime_df = prime_df.loc[prime_df.index.repeat(len(comp_df.index))].
    reset_index(drop=True)
    prime_df = pd.concat([prime_df, comp_df], axis=1)
    prime_df = prime_df.rename(columns = {"serp_string" : "serp_string_b",
    'keyword': 'keyword_b', "serp_string_a" : "serp_string", 'keyword_a':
    'keyword'})
    return prime_df
```

Test the function on a single keyword:

```
serps_align('ps4', strung_serps)
```

Set up desired dataframe columns:

```
columns = ['keyword', 'serp_string', 'keyword_b', 'serp_string_b']
matched_serps = pd.DataFrame(columns=columns)
matched_serps = matched_serps.fillna(0)
```

Call the function for each keyword:

```
for q in queries:
    temp_df = serps_align(q, strung_serps)
    matched_serps = matched_serps.append(temp_df)
```

This results in the following:

	keyword	serp_string	keyword_b	serp_string_b
0	ps4	https://www.playstation.com/en-gb/ps4/ https:/...	beige trench coats	https://www.zalando.co.uk/womens-clothing-coat...
1	ps4	https://www.playstation.com/en-gb/ps4/ https:/...	blue trench coats	https://www.johnlewis.com/browse/women/womens-...
2	ps4	https://www.playstation.com/en-gb/ps4/ https:/...	buy ps4	https://www.playstation.com/en-gb/ps4/buy-ps4/...
3	ps4	https://www.playstation.com/en-gb/ps4/ https:/...	ladies trench coats	https://www.johnlewis.com/browse/women/womens-...
4	ps4	https://www.playstation.com/en-gb/ps4/ https:/...	mens trench coats	https://uk.burberry.com/mens-trench-coats/ htt...
...	...	...	...	...
267	blue trench coats	https://www.johnlewis.com/browse/women/womens-...	trench coats	https://uk.burberry.com/womens-trench-coats/ h...
268	blue trench coats	https://www.johnlewis.com/browse/women/womens-...	xbox controller	https://www.xbox.com/en-GB/accessories/control...
269	blue trench coats	https://www.johnlewis.com/browse/women/womens-...	xbox one	https://www.xbox.com/ https://www.xbox.com/en-...
270	blue trench coats	https://www.johnlewis.com/browse/women/womens-...	xbox one controller	https://www.xbox.com/en-GB/accessories https:/...
271	blue trench coats	https://www.johnlewis.com/browse/women/womens-...	xbox series x	https://www.xbox.com/en-GB/consoles/xbox-serie...

The preceding result shows all of the keywords with SERPs compared side by side with other keywords and their SERPs. Next, we'll infer keyword intent similarity by comparing serp_strings, but first here's a note on the methods like Levenshtein, Jaccard, etc.

Levenshtein distance is edit based, meaning the number of edits required to transform one string (in our case, serp_string) into the other string (serps_string_b). This doesn't work very well because the websites within the SERP strings are individual tokens, that is, not a single continuous string.

Sorensen-Dice is better because it is token based, that is, it treats the individual websites as individual items or tokens. Using set similarity methods, the logic is to find the common tokens and divide them by the total number of tokens present by combining both sets. It doesn't take the order into account, so we must go one better.

M Measure which looks at both the token overlap and the order of the tokens, that is, weighting the order tokens earlier (i.e., the higher ranking sites/tokens) more than the later tokens. There is no API for this unfortunately, so we wrote the function for you here:

```
import py_stringmatching as sm
ws_tok = sm.WhitespaceTokenizer()
```

Only compare the top k_urls results:

```
def serps_similarity(serps_str1, serps_str2, k=15):
    denom = k+1
    norm = sum([2*(1/i - 1.0/(denom)) for i in range(1, denom)])
    #use to tokenize the URLs
```

```
    ws_tok = sm.WhitespaceTokenizer()
    #keep only first k URLs
    serps_1 = ws_tok.tokenize(serps_str1)[:k]
    serps_2 = ws_tok.tokenize(serps_str2)[:k]
    #get positions of matches
    match = lambda a, b: [b.index(x)+1 if x in b else None for x in a]
    #positions intersections of form [(pos_1, pos_2), ...]
    pos_intersections = [(i+1,j) for i,j in enumerate(match(serps_1,
    serps_2)) if j is not None]
    pos_in1_not_in2 = [i+1 for i,j in enumerate(match(serps_1, serps_2)) if
    j is None]
    pos_in2_not_in1 = [i+1 for i,j in enumerate(match(serps_2, serps_1)) if
    j is None]

    a_sum = sum([abs(1/i -1/j) for i,j in pos_intersections])
    b_sum = sum([abs(1/i -1/denom) for i in pos_in1_not_in2])
    c_sum = sum([abs(1/i -1/denom) for i in pos_in2_not_in1])

    intent_prime = a_sum + b_sum + c_sum
    intent_dist = 1 - (intent_prime/norm)
    return intent_dist
```

Apply the function:

```
matched_serps['si_simi'] = matched_serps.apply(lambda x: serps_
similarity(x.serp_string, x.serp_string_b), axis=1)
matched_serps[["keyword", "keyword_b", "si_simi"]]
```

This is the resulting dataframe:

	keyword	keyword_b	si_simi
0	ps4	beige trench coats	0.058203
1	ps4	blue trench coats	0.050328
2	ps4	buy ps4	0.314561
3	ps4	ladies trench coats	0.050328
4	ps4	mens trench coats	0.058203
...	...	...	...
267	blue trench coats	trench coats	0.096999
268	blue trench coats	xbox controller	0.050328
269	blue trench coats	xbox one	0.050328
270	blue trench coats	xbox one controller	0.040118
271	blue trench coats	xbox series x	0.063454

272 rows × 3 columns

Before sorting the keywords into topic groups, let's add search volumes for each. This could be an imported table like the following one called "keysv_df":

```
keysv_df
```

This results in the following:

	keyword	search_volume
0	best isa rates	40500
1	isa	49500
2	savings account	60500
3	cash isa	14800
4	isa account	9900
5	child savings account	14800
6	fixed rate bonds	12100
7	isa rates	8100
8	savings account interest rate	9900
9	fixed rate isa	5400
10	isa interest rates	5400
11	savings accounts uk	6600
12	cash isa rates	4400
13	easy access savings account	3600
14	savings rates	5400
15	easy access savings	3600
16	fixed rate savings	4400
17	isa savings	3600
18	kids savings account	4400
19	online savings account	2400

Let's now join the data. What we're doing here is giving Python the ability to group keywords according to SERP similarity and name the topic groups according to the keyword with the highest search volume.

Group keywords by search intent according to a similarity limit. In this case, keyword search results must be 40% or more similar. This is a number based on trial and error of which the right number can vary by the search space, language, or other factors.

```
simi_lim = 0.4
```

Append topic vols:

```
keywords_crossed_vols = serps_compared.merge(keysv_df, on = 'keyword', how
= 'left')
keywords_crossed_vols = keywords_crossed_vols.rename(columns = {'keyword':
'topic', 'keyword_b': 'keyword', 'search_volume': 'topic_volume'})
```

Append keyword vols:

```
keywords_crossed_vols = keywords_crossed_vols.merge(keysv_df, on =
'keyword', how = 'left')
```

Simulate si_simi:

```
#keywords_crossed_vols['si_simi'] = np.random.rand(len(keywords_crossed_
vols.index))
keywords_crossed_vols.sort_values('topic_volume', ascending = False)
```

Strip the dataframe of NAN:

```
keywords_filtered_nonnan = keywords_crossed_vols.dropna()
```

We now have the potential topic name, keyword SERP similarity, and search volumes of each. You'll note the keyword and keyword_b have been renamed to topic and keyword, respectively. Now we're going to iterate over the columns in the dataframe using list comprehensions.

List comprehension is a technique for looping over lists. We applied it to the Pandas dataframe because it's much quicker than the .iterrows() function. Here it goes.

Add a dictionary comprehension to create numbered topic groups from keywords_ filtered_nonnan:

```
# {1: [k1, k2, ..., kn], 2: [k1, k2, ..., kn], ..., n: [k1, k2, ..., kn]}
```

Convert the top names into a list:

```
queries_in_df = list(set(keywords_filtered_nonnan.topic.to_list()))
```

Set empty lists and dictionaries:

```
topic_groups_numbered = {}
topics_added = []
```

Define a function to find the topic number:

```
def latest_index(dicto):
    if topic_groups_numbered == {}:
        i = 0
    else:
        i = list(topic_groups_numbered)[-1]
    return i
```

Define a function to allocate keyword to topic:

```
def find_topics(si, keyw, topc):
    i = latest_index(topic_groups_numbered)
    if (si >= simi_lim) and (not keyw in topics_added) and (not topc in
    topics_added):
        #print(si, ', kw=' , keyw,', tpc=', topc,', ', i,', ', topic_
        groups_numbered)
        i += 1
        topics_added.extend([keyw, topc])
        topic_groups_numbered[i] = [keyw, topc]
    elif si >= simi_lim and (keyw in topics_added) and (not topc in
    topics_added):
        #print(si, ', kw=' , keyw,', tpc=', topc,', ', i,', ', topic_
        groups_numbered)
        j = [key for key, value in topic_groups_numbered.items() if keyw
        in value]
        topics_added.extend(topc)
        topic_groups_numbered[j[0]].append(topc)
    elif si >= simi_lim and (not keyw in topics_added) and (not topc in
    topics_added):
        #print(si, ', kw=' , keyw,', tpc=', topc,', ', i,', ', topic_
        groups_numbered)
        j = list(mydict.keys())[list(mydict.values()).index(keyw)]
        topic_groups_numbered[j[0]].append(topc)
```

The list comprehension will now apply the function to group keywords into clusters:

```
[find_topics(x, y, z) for x, y, z in zip(keywords_filtered_nonnan.si_simi,
keywords_filtered_nonnan.keyword, keywords_filtered_nonnan.topic)]
topic_groups_numbered
```

This results in the following:

```
{1: ['easy access savings',
  'savings account',
  'savings accounts uk',
  'savings rates',
  'online savings account',
  'online savings account',
  'online savings account'],
 2: ['isa account', 'isa', 'isa savings', 'isa savings'],
 3: ['kids savings account', 'child savings account'],
 4: ['best isa rates',
  'cash isa',
  'fixed rate isa',
  'fixed rate isa',
  'isa rates',
  'isa rates',
  'isa rates'],
 5: ['savings account interest rate',
  'savings accounts uk',
  'online savings account'],
 6: ['easy access savings account', 'savings rates', 'online savings
    account'],
 7: ['cash isa rates', 'fixed rate isa', 'isa rates'],
 8: ['isa interest rates', 'isa rates'],
 9: ['fixed rate savings', 'fixed rate bonds', 'online savings account']}
```

The preceding results are statements printing out what keywords are in which topic group. We do this to make sure we don't have duplicates or errors, which is crucial for the next step to perform properly. Now we're going to convert the dictionary into a dataframe so you can see all of your keywords grouped by search intent:

```
topic_groups_lst = []
for k, l in topic_groups_numbered.items():
    for v in l:
        topic_groups_lst.append([k, v])

topic_groups_dictdf = pd.DataFrame(topic_groups_lst, columns=['topic_group_
no', 'keyword'])
topic_groups_dictdf
```

This results in the following:

	topic_group_no	keyword
0	1	easy access savings
1	1	savings account
2	1	savings accounts uk
3	1	savings rates
4	1	online savings account
5	1	online savings account
6	1	online savings account
7	2	isa account
8	2	isa
9	2	isa savings
10	2	isa savings
11	3	kids savings account
12	3	child savings account
13	4	best isa rates
14	4	cash isa
15	4	fixed rate isa
16	4	fixed rate isa
17	4	isa rates

As you can see, the keywords are grouped intelligently, much like a human SEO analyst would group these, except these have been done at scale using the wisdom of Google which is distilled from its vast number of users. Name the clusters:

```
topic_groups_vols = topic_groups_dictdf.merge(keysv_df, on = 'keyword', how
= 'left')

def highest_demand(df):
```

```
    df = df.sort_values('search_volume', ascending = False)
    del df['topic_group_no']
    max_sv = df.search_volume.max()
    df = df.loc[df.search_volume == max_sv]
    return df

topic_groups_vols_keywgrp = topic_groups_vols.groupby('topic_group_no')
topic_groups_vols_keywgrp.get_group(1)
```

Apply and combine:

```
high_demand_topics = topic_groups_vols_keywgrp.apply(highest_demand).
reset_index()
del high_demand_topics['level_1']
high_demand_topics = high_demand_topics.rename(columns = {'keyword':
'topic'})

def shortest_name(df):
    df['k_len'] = df.topic.str.len()
    min_kl = df.k_len.min()
    df = df.loc[df.k_len == min_kl]
    del df['topic_group_no']
    del df['k_len']
    del df['search_volume']
    return df

high_demand_topics_spl = high_demand_topics.groupby('topic_group_no')
```

Apply and combine:

```
named_topics = high_demand_topics_spl.apply(shortest_name).reset_index()
del named_topics['level_1']
```

Name topic numbered keywords:

```
topic_keyw_map = pd.merge(named_topics, topic_groups_dictdf, on = 'topic_
group_no', how = 'left')
topic_keyw_map
```

The resulting table shows that we now have keywords clustered by topic:

	topic_group_no	topic	keyword
0	1	savings account	savings accounts uk
1	1	savings account	savings account
2	1	savings account	savings account interest rate
3	1	savings account	easy access savings
4	1	savings account	savings rates
5	1	savings account	fixed rate savings
6	1	savings account	fixed rate bonds
7	1	savings account	online savings account
8	1	savings account	easy access savings account
9	2	isa	isa
10	2	isa	isa account
11	2	isa	isa savings
12	3	child savings account	kids savings account
13	3	child savings account	child savings account
14	4	best isa rates	cash isa
15	4	best isa rates	best isa rates

Let's add keyword search volumes:

```
topic_keyw_vol_map = pd.merge(topic_keyw_map, keysv_df, on = 'keyword', how
= 'left')
topic_keyw_vol_map
```

This results in the following:

	topic_group_no	topic	keyword	search_volume
0	1	savings account	savings accounts uk	6600
1	1	savings account	savings account	60500
2	1	savings account	savings account interest rate	9900
3	1	savings account	easy access savings	3600
4	1	savings account	savings rates	5400
5	1	savings account	fixed rate savings	4400
6	1	savings account	fixed rate bonds	12100
7	1	savings account	online savings account	2400
8	1	savings account	easy access savings account	3600
9	2	isa	isa	49500
10	2	isa	isa account	9900
11	2	isa	isa savings	3600
12	3	child savings account	kids savings account	4400
13	3	child savings account	child savings account	14800
14	4	best isa rates	cash isa	14800
15	4	best isa rates	best isa rates	40500

This is really starting to take shape, and you can quickly see opportunities emerging.

SERP Competitor Titles

If you don't have much Google Search Console data or Google Ads data to mine, then you may need to resort to your competitors. You may or may not want to use third-party keyword research tools such as SEMRush. And you don't have to.

Tools like SEMRush, Keyword.io, etc., certainly have a place in the SEO industry. In the absence of any other data, they are a decent ready source of intelligence on what search queries generate relevant traffic.

However, some work will need to be done in order to weed out the noise and extract high value phrases – assuming a competitive market. Otherwise, if your website (or

niche) is so new in terms of what it offers that there's insufficient demand (that has yet to be created by advertising and PR to generate nonbrand searches), then these external tools won't be as valuable. So, our approach will be to

1. Crawl your own website
2. Filter and clean the data for sections covering only what you sell
3. Extract keywords from your site's title tags
4. Filter using SERPs data (next section)

Filter and Clean the Data for Sections Covering Only What You Sell

The required data for this exercise is to literally take a site auditor[5] and crawl your website. Let's assume you've exported the crawl data with just the columns: URL and title tag; we'll import and clean:

```
import pandas as pd
import numpy as np

crawl_import_df = pd.read_csv('data/crawler-filename.csv')
crawl_import_df
```

This results in the following:

	deeprank	page_title	url	redirected_to_url	http_status_code	indexable
0	0.71	Growing blueberries in pots - Saga	https://www.saga.co.uk/magazine/home-garden/ga...	NaN	200	True
1	0.66	How to grow succulents - Saga	https://www.saga.co.uk/magazine/home-garden/ga...	NaN	200	True
2	0.49	Creating a wildlife garden: courtyards & small...	https://www.saga.co.uk/magazine/home-garden/ga...	NaN	200	True
3	0.55	Plants for clay soil - Saga	https://www.saga.co.uk/magazine/home-garden/ga...	NaN	200	True
4	0.28	The brambling: diet, identifcation & location ...	https://www.saga.co.uk/magazine/home-garden/ga...	NaN	200	True
...	...	...	...	...	...	...
7026	0.49	The best plants to complement roses - Saga	https://www.saga.co.uk/magazine/home-garden/ga...	NaN	200	True
7027	0.77	How to buy outdoor security lights for your ga...	https://www.saga.co.uk/magazine/home-garden/ga...	NaN	200	True
7028	0.39	Wildlife watch: lesser spotted woodpecker - Saga	https://www.saga.co.uk/magazine/home-garden/ga...	NaN	200	True
7029	0.25	Has my border collie's coat been damaged by cl...	https://www.saga.co.uk/magazine/home-garden/pe...	NaN	200	True
7030	0.32	The best daffodil varieties that bloom all spr...	https://www.saga.co.uk/magazine/home-garden/ga...	NaN	200	False

[5] Like Screaming Frog, OnCrawl, or Botify, for instance

The preceding result shows the dataframe of the crawl data we've just imported. We're most interested in live indexable[6] URLs, so let's filter and select the page_title and URL columns:

```
titles_urls_df = crawl_import_df.loc[crawl_import_df.indexable == True]
titles_urls_df = titles_urls_df[['page_title', 'url']]
titles_urls_df
```

This results in the following:

	page_title	url
0	Growing blueberries in pots - Saga	https://www.saga.co.uk/magazine/home-garden/ga...
1	How to grow succulents - Saga	https://www.saga.co.uk/magazine/home-garden/ga...
2	Creating a wildlife garden: courtyards & small...	https://www.saga.co.uk/magazine/home-garden/ga...
3	Plants for clay soil - Saga	https://www.saga.co.uk/magazine/home-garden/ga...
4	The brambling: diet, identifcation & location ...	https://www.saga.co.uk/magazine/home-garden/ga...
...	...	...
7025	Avoid these online tolls and road charges scam...	https://www.saga.co.uk/magazine/motoring/cars/...
7026	The best plants to complement roses - Saga	https://www.saga.co.uk/magazine/home-garden/ga...
7027	How to buy outdoor security lights for your ga...	https://www.saga.co.uk/magazine/home-garden/ga...
7028	Wildlife watch: lesser spotted woodpecker - Saga	https://www.saga.co.uk/magazine/home-garden/ga...
7029	Has my border collie's coat been damaged by cl...	https://www.saga.co.uk/magazine/home-garden/pe...

Now we're going to clean the title tags to make these nonbranded, that is, remove the site name and the magazine section.

```
titles_urls_df['page_title'] = titles_urls_df.page_title.str.replace(' -
Saga', '')
titles_urls_df = titles_urls_df.loc[~titles_urls_df.url.str.contains('/
magazine/')]
titles_urls_df
```

This results in the following:

[6] That is, pages with a 200 HTTP response that do block search indexing with "noindex"

	page_title	url
9	Travel money | Saga | Safe and quick order today	https://www.saga.co.uk/insurance/travel-money
21	MySaga	https://www.saga.co.uk/mysaga/manage-policy/html/
31	Direct Choice Contact Us	https://www.saga.co.uk/insurance/car-insurance...
48	Personal SOS Alarms Service | Saga Healthcare	https://www.saga.co.uk/sos-personal-alarm
49	Tailor Your Cover With Optional Extras | Saga ...	https://www.saga.co.uk/insurance/car-insurance...
...	...	...
6991	At home with John Sergeant	https://www.saga.co.uk/membership/articles/at-...
6994	Policy Documents | Policy Books | Saga Home In...	https://www.saga.co.uk/insurance/home-insuranc...
6995	Health and Beauty Offers From Saga	https://www.saga.co.uk/membership/categories/h...
6996	Single Trip Travel Insurance for Over 50s | Sa...	https://www.saga.co.uk/insurance/travel-insura...
7008	Oleanna	https://www.saga.co.uk/membership/tickets/lond...

We now have 349 rows, so we will query some of the keywords to illustrate the process.

Extract Keywords from the Title Tags

We now desire to extract keywords from the page title in the preceding dataframe. A typical data science approach would be to break down the titles into all kinds of combinations and then do a frequency count, maybe weighted by ranking.

Having tried it, we wouldn't recommend this approach; it's overkill and there is probably not enough data to make it worthwhile. A more effective and simpler approach is to break down the titles by punctuation marks. Why? Because humans (or probably some AI nowadays) wrote those titles, so these are likely to be natural breakpoints for target search phrases.

Let's try it; break the titles into n grams:

```
pd.set_option('display.max_rows', 1000)
serps_ngrammed = filtered_serps_df.set_index(["keyword", "rank_absolute"])\
                 .apply(lambda x: x.str.split('[-,|?()&:;\[\]=]').
                 explode())\
                 .dropna()\
                 .reset_index()
serps_ngrammed.head(10)
```

This results in the following:

	keyword	rank_absolute	title
0	Care Funding Advice Service	1	care funding advice service
1	Care Funding Advice Service	1	saga
2	Care Funding Advice Service	2	how does the saga care funding advice service ...
3	Care Funding Advice Service	2	
4	Care Funding Advice Service	3	paying for care
5	Care Funding Advice Service	3	money advice service
6	Care Funding Advice Service	4	care funding advice
7	Care Funding Advice Service	4	hub financial solutions
8	Care Funding Advice Service	5	care advice service
9	Care Funding Advice Service	5	paying for care fees in sussex and ...

Courtesy of the explode function, the dataframe has been unnested such that we can see the keyword rows expanded for the different text previously within the same title and conjoined by the punctuation mark.

Filter Using SERPs Data

Now all we have to do is perform a frequency count of the top three titles and then filter for any that appear three times or more:

```
serps_ngrammed_grp = serps_ngrammed.groupby(['keyword', 'title'])
keyword_ideas_df = serps_ngrammed_grp.size().reset_index(name='freq').sort_
values(['keyword', 'freq'], ascending = False)
keyword_ideas_df = keyword_ideas_df[keyword_ideas_df.freq > 2]
keyword_ideas_df = keyword_ideas_df[keyword_ideas_df.title.str.
contains('[a-z]')]
keyword_ideas_df = keyword_ideas_df.rename(columns = {'title':
'keyword_idea'})
keyword_ideas_df
```

This results in the following:

	keyword	rank_absolute	title
0	Care Funding Advice Service	1	care funding advice service
1	Care Funding Advice Service	1	saga
2	Care Funding Advice Service	2	how does the saga care funding advice service ...
3	Care Funding Advice Service	2	
4	Care Funding Advice Service	3	paying for care
5	Care Funding Advice Service	3	money advice service
6	Care Funding Advice Service	4	care funding advice
7	Care Funding Advice Service	4	hub financial solutions
8	Care Funding Advice Service	5	care advice service
9	Care Funding Advice Service	5	paying for care fees in sussex and ...

Eh voila, the preceding result shows a dataframe of keywords obtained from the SERPs. Most of it makes sense and can now be added to your list of keywords for serious consideration and tracking.

Summary

This chapter has covered data-driven keyword research, enabling you to

- Find standout keywords from GSC data
- Obtain trend data from Google Trends
- Forecast future organic traffic using time series techniques
- Cluster keywords by search intent
- Find keywords from your competitors using SERPs data

In the next chapter, we will cover the mapping of those keywords to URLs.

CHAPTER 3

Technical

Technical SEO mainly concerns the interaction of search engines and websites such that

- Website content is made discoverable by search engines.
- The priority of content is made apparent to search engines implied by its proximity to the home page.
- Search engine resources are conserved to access content (known as crawling) intended for search result inclusion.
- Extract the content meaning from those URLs again for search result inclusion (known as indexing).

In this chapter, we'll look at how data-driven approach can be taken toward improving technical SEO in the following manner:

- *Modeling page authority*: This is useful for helping fellow SEO and non-SEOs understand the impact of technical SEO changes.
- *Internal link optimization*: To improve the use of internal links used to make content more discoverable and help signal to search engines the priority of content.
- *Core Web Vitals (CWV)*: While the benefits to the UX are often lauded, there are ranking boost benefits to an improved CWV because of the conserved search engine resources used to extract content from a web page.

By no means will we claim that this is the final word on data-driven SEO from a technical perspective. What we will do is expose data-driven ways of solving technical SEO issues using some data science such as distribution analysis.

A. Voniatis, *Data-Driven SEO with Python*, https://doi.org/10.1007/978-1-4842-9175-7_3

Where Data Science Fits In

An obvious challenge of SEO is deciding which pages should be made accessible to the search engines and users and which ones should not. While many crawling tools provide visuals of the distributions of pages by site depth, etc., it never hurts to use data science, which we will go into more detail and complexity, which will help you

- Optimize internal links
- Allocate keywords to pages based on the copy
- Allocate parent nodes to the orphaned URLs

Ultimately, the preceding list will help you build better cases for getting technical recommendations implemented.

Modeling Page Authority

Technical optimization involves recommending changes that often make URLs nonindexable or canonicalized (for a number of reasons such as duplicate content). These changes are recommended with the aim of consolidating page authority onto URLs which will remain eligible for indexing.

The following section aims to help data-driven SEO quantify the beneficial extra page authority. The approach will be to

1. Filter in web pages
2. Examine the distribution of authority before optimization
3. Calculate the new distribution (to quantify the incremental page authority following a decision on which URLs will no longer be made indexable, making their authority available for reallocation)

First, we need to load the necessary packages:

```
import re
import time
import random
import pandas as pd
import numpy as np
import datetime
```

```
import requests
import json
from datetime import timedelta
from glob import glob
import os
from client import RestClient # If using the Data For SEO API
from textdistance import sorensen_dice
from plotnine import *
import matplotlib.pyplot as plt
from pandas.api.types import is_string_dtype
from pandas.api.types import is_numeric_dtype
import uritools
from urllib.parse import urlparse
import tldextract

pd.set_option('display.max_colwidth', None)
%matplotlib inline
```

Set variables:

```
root_domain = 'boundlesshq.com'
hostdomain = 'www.boundlesshq.com'
hostname = 'boundlesshq'
full_domain = 'https://www.boundlesshq.com'
client_name = 'Boundless'
audit_monthyear = 'jul_2022'
```

Import the crawl data from the Sitebulb desktop crawler. Screaming Frog or any other site crawling software can be used; however, the column names may differ:

```
crawl_csv = pd.read_csv('data/boundlesshq_com_all_urls__excluding_
uncrawled__filtered.csv')
```

Clean the column names using a list comprehension:

```
crawl_csv.columns = [col.lower().replace('.','').replace('(','').
replace(')','').replace(' ','_')
                     for col in crawl_csv.columns]

crawl_csv
```

Here is the result of crawl_csv:

	url	crawl_depth	craw
0	https://boundlesshq.com/	0	
1	https://boundlesshq.com/wp-content/uploads/oxygen/css/2920.css	1	
2	https://bat.bing.com/action/0?ti=149000342&tm=gtm002&Ver=2&mid=40c49240-b5a7-4d30-9cc3-29f89fc1e165&sid=d7008f20086b11edbbe6afef94e29e7c&vid=e23f2650fc5911ec911285f6dcda096a&vids=0&gtm_tag_source=1&pi=2083220816&lg=en-US&sw=1512&sh=982&sc=30&tl=Blog%20-%20Boundless&p=https%3A%2F%2Fboundlesshq.com%2Fblog%2F&r=&lt=1347&evt=pageLoad&msclkid=N&sv=1&rn=541225	1	
3	https://boundlesshq.com/wp-content/uploads/2022/06/touch-handy.png	1	
4	https://boundlesshq.com/wp-content/uploads/oxygen/css/5983.css	1	
...	...	...	
5417	https://www.convertcalculator.com/_next/static/chunks/462-a4fa6446d46110f5.js	1	
5418	https://static.cloudflareinsights.com/beacon.min.js	1	
5419	https://fonts.gstatic.com/s/inter/v12/UcC73FwrK3iLTeHuS_fvQtMwCp50KnMa1ZL7W0Q5nw.woff2	1	
5420	https://www.convertcalculator.com/_next/static/4t_ps-wEsto6ZaMDrAxlf/_ssgManifest.js	1	
5421	https://www.convertcalculator.com/_next/static/chunks/webpack-ab4bf6124310705c.js	1	

5422 rows × 25 columns

The dataframe is loaded into a Pandas dataframe. The most important fields are as follows:

- *url*: To detect patterns for noindexing and canonicalizing
- *ur*: URL Rating, Sitebulb's in-house metric for measuring internal page authority
- *content_type*: For filtering
- *passes_pagerank*: So we know which pages have authority
- *indexable*: Eligible for search engine index inclusion

Filtering in Web Pages

The next step is to filter in actual web pages that belong to the site and are capable of passing authority:

```
crawl_html = crawl_csv.copy()
crawl_html = crawl_html.loc[crawl_html['content_type'] == 'HTML']
crawl_html = crawl_html.loc[crawl_html['host'] == root_domain]
crawl_html = crawl_html.loc[crawl_html['passes_pagerank'] == 'Yes']

crawl_html
```

	url	crawl_depth	crawl_status	host	is_subdomain	scheme	crawl_source	
0	https://boundlesshq.com/	0	Success	boundlesshq.com	No	https	Crawler	
6	https://boundlesshq.com/blog/	1	Success	boundlesshq.com	No	https	Crawler	https:/
7	https://boundlesshq.com/summer-sale/	1	Success	boundlesshq.com	No	https	Crawler	https:/
28	https://boundlesshq.com/blog/employment/what-is-an-employer-of-record/	1	Success	boundlesshq.com	No	https	Crawler	https:/
47	https://boundlesshq.com/how-it-works/	1	Success	boundlesshq.com	No	https	Crawler	https:/
...	...	...	...	...	...	...	...	
5354	https://boundlesshq.com/category/blog/page/3/	4	Success	boundlesshq.com	No	https	Crawler	https://boundlesshq.
5356	https://boundlesshq.com/category/blog/page/2/	4	Success	boundlesshq.com	No	https	Crawler	https://boundlesshq.
5358	https://boundlesshq.com/category/blog/page/4/	4	Success	boundlesshq.com	No	https	Crawler	https://boundlesshq.
5370	https://boundlesshq.com/category/blog/page/1/	5	Redirect	boundlesshq.com	No	https	Crawler	https://boundlesshq.com/cat
5385	https://boundlesshq.com/uk-custom-report/	Not Set	Success	boundlesshq.com	No	https	XML Sitemap	

309 rows × 25 columns

The dataframe has been reduced to 309 rows. For ease of data handling, we'll select some columns:

```
crawl_select = crawl_html[['url', 'ur', 'crawl_depth', 'crawl_source',
'http_status_code', 'indexable',
                'indexable_status', 'passes_pagerank', 'total_impressions',
'first_parent_url', 'meta_robots_response']].copy()
```

Examine the Distribution of Authority Before Optimization

It is useful for groupby aggregation and counting:

```
crawl_select['project'] = client_name
crawl_select['count'] = 1
```

Let's get some quick stats:

```
print(crawl_select['ur'].sum(), crawl_select['ur'].sum()/crawl_select.
shape[0])
```

```
10993 35.57605177993528
```

URLs on this site have an average page authority level (measured as UR). Let's look at some further stats, indexable and nonindexable pages. We'll dimension on (I) indexable and (II) passes pagerank to sum the number of URLs and UR (URL Rating):

```
overall_pagerank_agg = crawl_select.groupby(['indexable',
```

```
                                        'passes_pagerank']).agg
                                        ({'count': 'sum',
                                                              'ur':
                                                              'sum'}).
                                                              reset_
                                                              index()
```

Then we derive the page authority per URL by dividing the total UR by the total number of URLs:

```
overall_pagerank_agg['PA'] = overall_pagerank_agg['ur'] / overall_pagerank_
agg['count']
overall_pagerank_agg
```

This results in the following:

	indexable	passes_pagerank	count	ur	PA
0	No	Yes	32	929	29.03125
1	Yes	Yes	277	10064	36.33213

We see that there are 32 nonindexable URLs with a total authority of 929 that could be consolidated to the indexable URLs.

There are some more stats, this time analyzed by site level purely out of curiosity:

```
site_pagerank_agg = crawl_select.groupby(['indexable',
                                          'crawl_depth']).
                                          agg({'count': 'sum',
                                                              'ur':
                                                              'sum'}).
                                                              reset_
                                                              index()
site_pagerank_agg['PA'] = site_pagerank_agg['ur'] / site_pagerank_
agg['count']

site_pagerank_agg
```

This results in the following:

	indexable	crawl_depth	count	ur	PA
0	No	1	2	186	93.000000
1	No	2	1	20	20.000000
2	No	3	19	640	33.684211
3	No	4	9	73	8.111111
4	No	5	1	10	10.000000
5	Yes	0	1	95	95.000000
6	Yes	1	13	1127	86.692308
7	Yes	2	46	2052	44.608696
8	Yes	3	196	6470	33.010204
9	Yes	4	20	320	16.000000
10	Yes	Not Set	1	0	0.000000

Most of the URLs that have the authority for reallocation are four clicks away from the home page.

Let's visualize the distribution of the authority preoptimization, using the geom_histogram function:

```
pageauth_dist_plt = (
    ggplot(crawl_select, aes(x = 'ur')) +
    geom_histogram(alpha = 0.7, fill = 'blue', bins = 20) +
    labs(x = 'Page Authority', y = 'URL Count') +
    theme(legend_position = 'none', axis_text_x=element_text(rotation=0,
    hjust=1, size = 12))
)

pageauth_dist_plt.save(filename = 'images/1_pageauth_dist_plt.png',
                            height=5, width=8, units = 'in', dpi=1000)
pageauth_dist_plt
```

As we'd expect from looking at the stats computed previously, most of the pages have between 25 and 50 UR, with the rest spread out (Figure 3-1).

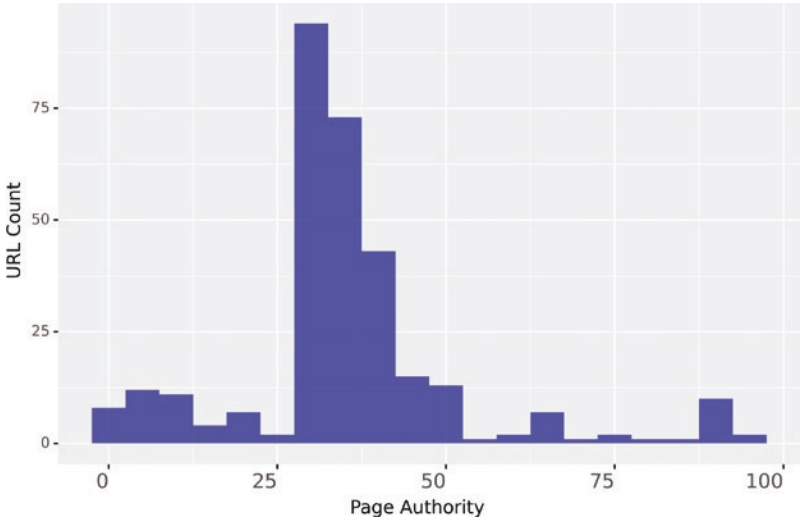

Figure 3-1. *Histogram plot showing URL count of URL Page Authority scores*

Calculating the New Distribution

With the current distribution examined, we'll now go about quantifying the new page authority distribution following optimization.

We'll start by getting a table of URLs by the first parent URL and the URL's UR values which will be our mapping for how much extra authority is available:

```
parent_pa_map = crawl_select[['first_parent_url', 'ur']].copy()
parent_pa_map = parent_pa_map.rename(columns = {'first_parent_url': 'url' ,
'ur': 'extra_ur'})

parent_pa_map
```

This results in the following:

	url	extra_ur
0	None	95
6	https://boundlesshq.com/	80
7	https://boundlesshq.com/	70
28	https://boundlesshq.com/	88
47	https://boundlesshq.com/	91
...	...	...
5354	https://boundlesshq.com/category/blog/	10
5356	https://boundlesshq.com/category/blog/	10
5358	https://boundlesshq.com/category/blog/	10
5370	https://boundlesshq.com/category/blog/page/2/	10
5385	None	0

309 rows × 2 columns

The table shows all the parent URLs and their mapping.

The next step is to mark pages that will be noindexed, so we can reallocate their authority:

```
crawl_optimised = crawl_select.copy()
```

Create a list of URL patterns for noindex:

```
reallocate_conds = [
    crawl_optimised['url'].str.contains('/page/[0-9]/'),
    crawl_optimised['url'].str.contains('/country/')
]
```

Values if the URL pattern conditions are met.

```
reallocate_vals = [1, 1]
```

The reallocate column uses the np.select function to mark URLs for noindex. Any URLs not for noindex are marked as "0," using the default parameter:

```
crawl_optimised['reallocate'] = np.select(reallocate_conds, reallocate_
vals, default = 0)
```

```
crawl_optimised
```

This results in the following:

first_parent_url	meta_robots_response	project	count	reallocate
None	index, follow, max-image-preview:large, max-snippet:-1, max-video-preview:-1	Boundless	1	0
https://boundlesshq.com/	index, follow, max-image-preview:large, max-snippet:-1, max-video-preview:-1	Boundless	1	0
https://boundlesshq.com/	index, follow, max-image-preview:large, max-snippet:-1, max-video-preview:-1	Boundless	1	0
https://boundlesshq.com/	index, follow, max-image-preview:large, max-snippet:-1, max-video-preview:-1	Boundless	1	0
https://boundlesshq.com/	index, follow, max-image-preview:large, max-snippet:-1, max-video-preview:-1	Boundless	1	0
...	...	...	...	...

The reallocate column is added so we can start seeing the effect of the reallocation, that is, the potential upside of technical optimization.

As usual, a groupby operation by reallocate and the average PA are calculated:

```
reallocate_agg = crawl_optimised.groupby('reallocate').agg({'count': sum,
'ur': sum}).reset_index()
reallocate_agg['PA'] = reallocate_agg['ur'] / reallocate_agg['count']
reallocate_agg
```

This results in the following:

	reallocate	count	ur	PA
0	0	285	10312	36.182456
1	1	24	681	28.375000

So we'll be actually reallocating 681 UR from the noindex URLs to the 285 indexable URLs. These noindex URLs have an average UR of 28.

We filter the URLs just for the ones that will be noindexed to help us in determining what the extra page authority will be:

```
no_indexed = crawl_optimised.loc[crawl_optimised['reallocate'] == 1]
```

We aggregate by the first parent URL (the parent node) for the total URLs within and their URL, because the UR is likely to be reallocated to the remaining indexable URLs that share the same parent node:

```
no_indexed_map = no_indexed.groupby('first_parent_url').agg({'count':
'sum', 'ur': sum}).reset_index()
```

add_ur is a new column created representing the additional authority as a result of the optimization. This is the total UR divided by the number of URLs:

```
no_indexed_map['add_ur'] = (no_indexed_map['ur'] / no_indexed_
map['count']).round(0)
```

Drop columns not required for joining later:

```
no_indexed_map.drop(['ur', 'count'], inplace = True,  axis = 1)
no_indexed_map
```

This results in the following:

	first_parent_url	add_ur
0	https://boundlesshq.com/category/blog/	10.0
1	https://boundlesshq.com/category/blog/employment/	2.0
2	https://boundlesshq.com/category/blog/employment/page/2/	2.0
3	https://boundlesshq.com/category/blog/page/2/	10.0
4	https://boundlesshq.com/guides/australia/	35.0
5	https://boundlesshq.com/guides/brazil/	38.0
6	https://boundlesshq.com/guides/canada/	35.0
7	https://boundlesshq.com/guides/chile/end-of-employment/	31.0

The preceding table will be merged into the indexable URLs by the first parent URL.

Filter the URLs just for the indexable and add more authority as a result of the noindexing reallocate URLs:

```
crawl_new = crawl_optimised.copy()
crawl_new = crawl_new.loc[crawl_new['reallocate'] == 0]
```

Join the no_indexed_map to get the amount of authority to be added:

```
crawl_new = crawl_new.merge(no_indexed_map, on = 'first_parent_url', how
= 'left')
```

Often, when joining data, there will be null values for first parent URLs not in the mapping. np.where() is used to replace those null values with zeros. This enables further data manipulation to take place as you'll see shortly.

```
crawl_new['add_ur'] = np.where(crawl_new['add_ur'].isnull(), 0, crawl_
new['add_ur'])
```

New_ur is the new authority score calculated by adding ur to add_ur:

```
crawl_new['new_ur'] = crawl_new['ur'] + crawl_new['add_ur']

crawl_new
```

This results in the following:

first_parent_url	meta_robots_response	project	count	reallocate	add_ur	new_ur
None	index, follow, max-image-preview:large, max-snippet:-1, max-video-preview:-1	Boundless	1	0	0.0	95.0
https://boundlesshq.com/	index, follow, max-image-preview:large, max-snippet:-1, max-video-preview:-1	Boundless	1	0	0.0	80.0
https://boundlesshq.com/	index, follow, max-image-preview:large, max-snippet:-1, max-video-preview:-1	Boundless	1	0	0.0	70.0
https://boundlesshq.com/	index, follow, max-image-preview:large, max-snippet:-1, max-video-preview:-1	Boundless	1	0	0.0	88.0
https://boundlesshq.com/	index, follow, max-image-preview:large, max-snippet:-1, max-video-preview:-1	Boundless	1	0	0.0	91.0

The indexable URLs now have their authority scores post optimization, which we'll visualize as follows:

```
pageauth_newdist_plt = (
    ggplot(crawl_new, aes(x = 'new_ur')) +
    geom_histogram(alpha = 0.7, fill = 'lightgreen', bins = 20) +
    labs(x = 'Page Authority', y = 'URL Count') +
    theme(legend_position = 'none', axis_text_x=element_text(rotation=0,
    hjust=1, size = 12))
)

pageauth_newdist_plt.save(filename = 'images/2_pageauth_newdist_plt.png',
                          height=5, width=8, units = 'in', dpi=1000)
pageauth_newdist_plt
```

The pageauth_newdist_plt in Figure 3-2 shows the distribution of page-level authority (page authority).

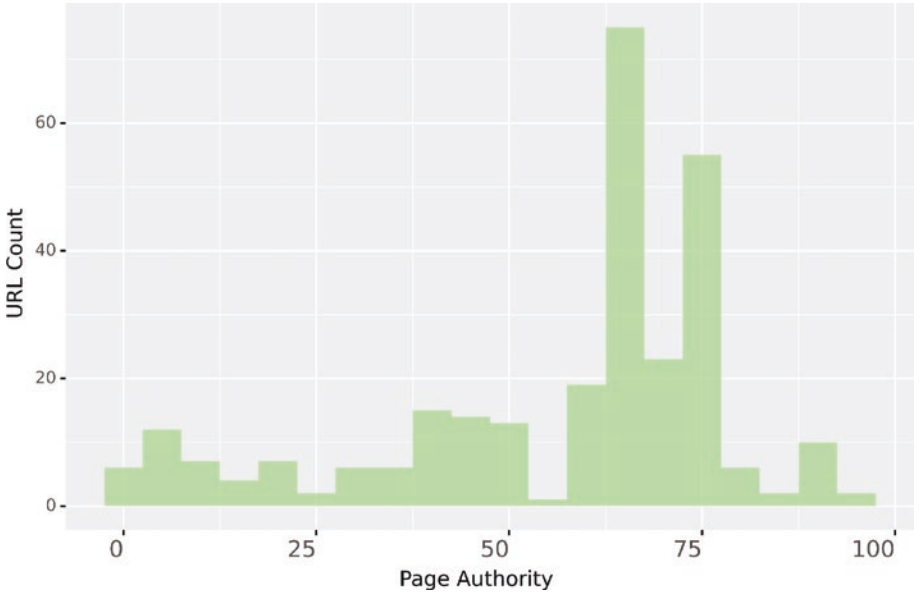

Figure 3-2. *Histogram of the distribution of page-level authority (page authority)*

The impact is noticeable, as we see most pages are above 60 UR post optimization, should the implementation move forward.

There are some quick stats to confirm:

```
new_pagerank_agg = crawl_new.groupby(['reallocate']).agg({'count': 'sum',
                                                          'ur': 'sum',
                                                          'new_ur':
                                                          'sum'}).
                                                          reset_ex()

new_pagerank_agg['PA'] = new_pagerank_agg['new_ur'] / new_pagerank_
agg['count']

print(new_pagerank_agg)

  reallocate  count     ur   new_ur    PA
0           0    285  10312  16209.0  57.0
```

The average page authority is now 57 vs. 36, which is a significant improvement. While this method is not an exact science, it could help you to build a case for getting your change requests for technical SEO fixes implemented.

Internal Link Optimization

Search engines are highly dependent on links in order to help determine the relative importance of pages within a website. That's because search engines work on the basis of assigning probability that content will be found by users at random based on the random surfer concept. That is, a content is more likely to be discovered if there are more links to the content.

If the content has more inbound links, then search engines also assume the content has more value, having earned more links.

Search engines also rely on the anchor text to signal what the hyperlinked URL's content will be about and therefore its relevance to keywords.

Thus, for SEO, internal links play a key role in website optimization, helping search engines decide which pages on the site are important and their associated keywords.

Here, we shall provide methods to optimize internal links using some data science, which will cover

1. Distributing authority by site level
2. Distributing authority by external page authority accrued from external sites
3. Anchor text

```
import pandas as pd
import numpy as np
from textdistance import sorensen_dice
from plotnine import *
import matplotlib.pyplot as plt
from mizani.formatters import comma_format

target_name = 'ON24'
target_filename = 'on24'
website = 'www.on24.com'
```

The link data is sourced from the Sitebulb auditing software which is being imported along with making the column names easier to work with:

```
link_raw = pd.read_csv('data/'+ client_filename + '_links.csv')
link_data = link_raw.copy()
```

```
link_data.drop('Unnamed: 13', axis = 1, inplace = True)

link_data.columns = [col.lower().replace('.','').replace('(','').
replace(')','').replace(' ','_')
                     for col in link_data.columns]

link_data
```

	referring_url	referring_url_rank_ur	target_url	target_url_rank_ur	anchor_text	location	crawl_status	no
0	https://www.on24.com/	96	https://www.on24.com/resources/asset/webinar-b...	92	Register Now	Content	Success	
1	https://www.on24.com/	96	https://www.on24.com/contact-us/	96	Contact Us	Header	Success	
2	https://www.on24.com/	96	https://www.on24.com/login/	100	Login	Header	Success	
3	https://www.on24.com/	96	https://www.on24.com/resources/	96	Resources	Header	Success	
4	https://www.on24.com/	96	https://www.on24.com/live-demo/	96	Live Demo	Header	Success	
...	...	...	...	...	...	...	...	
406022	https://www.on24.com/zapier/	0	https://www.on24.com/about-us/careers/	96	Careers	Footer	Success	
406023	https://www.on24.com/zapier/	0	https://www.on24.com/newsroom/	96	In The News	Footer	Success	
406024	https://www.on24.com/zapier/	0	https://www.on24.com/press-releases/	96	Press Releases	Footer	Success	
406025	https://www.on24.com/zapier/	0	https://www.on24.com/contact-us/	96	Contact Sales	Footer	Success	
406026	https://www.on24.com/zapier/	0	https://www.on24.co.jp/	0	Japan Website	Footer	Success	

406027 rows × 13 columns

The link dataframe shows us a list of links in terms of

- *Referring URL*: Where they are found
- *Target URL*: Where they point to
- *Referring URL Rank UR*: The page authority of the referring page
- *Target URL Rank UR*: The page authority of the target page
- *Anchor text*: The words used in the hyperlink
- *Location*: Where the link can be found

Let's import the crawl data, also sourced from Sitebulb:

```
crawl_data = pd.read_csv('data/'+ client_filename + '_crawl.csv')

crawl_data.drop('Unnamed: 103', axis = 1, inplace = True)
```

```
crawl_data.columns = [col.lower().replace('.','').replace('(','').
replace(')','').replace(' ','_')
                      for col in crawl_data.columns]

crawl_data
```

This results in the following:

	url	base_url	crawl_depth	crawl_status	host	http_protocol	is_subdomain	no_query_string_keys
0	https://www.on24.com/	No Data	0	Success	www.on24.com	http/1.1	No	0
1	https://www.on24.com/customer-stories/align-te...	No Data	1	Not Found	www.on24.com	http/1.1	No	0
2	https://www.on24.com/contact-us/	No Data	1	Success	www.on24.com	http/1.1	No	0
3	https://www.on24.com/resources/	No Data	1	Success	www.on24.com	http/1.1	No	0
4	https://www.on24.com/blog/how-juniper-networks...	No Data	1	Success	www.on24.com	http/1.1	No	0
...	...	...	...	...	...	...	...	...
8606	https://on24.influitive.com/saml/initialize	No Data	Not Set	Redirect	on24.influitive.com	http/1.1	No	0
8607	https://www.contenttechsummit.com/	No Data	Not Set	Redirect	www.contenttechsummit.com	http/1.1	No	0
8608	https://newsnetwork.mayoclinic.org/	No Data	Not Set	Success	newsnetwork.mayoclinic.org	http/1.1	No	0
8609	https://www.eetimes.com/	No Data	Not Set	Success	www.eetimes.com	http/1.1	No	0
8610	http://offers.hubspot.com/how-to-promote-a-wor...	No Data	Not Set	Redirect	offers.hubspot.com	http/1.1	No	0

8611 rows × 104 columns

So we have the usual list of URLs and how they were found (crawl source) with other features spanning over 100 columns.

As you'd expect, the number of rows in the link data far exceeds the crawl dataframe as there are many more links than pages!

Import the external inbound link data:

```
ahrefs_raw = pd.read_csv('data/'+ client_filename + '_ahrefs.csv')

ahrefs_raw.columns = [col.lower().replace('.','').replace('(','').
replace(')','').replace(' ','_')
                      for col in ahrefs_raw.columns]

ahrefs_raw
```

This results in the following:

	#	url_rating_desc	page_url	page_title	referring_domains	dofollow	nofollow	redirects	first_seen	size	co
0	1	81	https://www.on24.com/	Webinar Software & Virtual Event Platform | ON24	3215	64144	18915	48	20/12/2018 17:02	32 kB	2
1	2	54	http://www.on24.com/	NaN	856	4666	1017	173	08/08/2013 16:21	162 B	3
2	3	52	https://vshow.on24.com/view/vts/error.html?cod...	Error	643	3455	299	0	12/12/2014 12:07	3 kB	2
3	4	43	https://on24.com/	NaN	668	46591	4628	0	17/12/2018 03:07	0	3
4	5	41	https://www.on24.com/live-webcast-elite/	Deliver Better Webcasts on ON24 Webcast Elite ...	208	173	363	0	04/01/2019 01:57	33 kB	2
...	...	...	...	...	...	...	...	...	...	...	
210399	210400	0	https://www.on24.com/ww19internal/	NaN	1	0	10	0	NaN	0	
210400	210401	0	https://www.on24.com/youre-in/	NaN	1	0	10	0	NaN	0	
210401	210402	0	https://www-staging.on24.com/	NaN	1	2	0	0	NaN	0	
210402	210403	0	http://acueductoportachuelo.xyzevent.on24.com/...	NaN	1	5	0	0	NaN	0	
210403	210404	0	https://golearnership.co.zaevent.on24.com/even...	NaN	1	22	0	0	NaN	0	

210404 rows × 13 columns

There are over 210,000 URLs with backlinks, which is very nice! There's quite a bit of data, so let's simplify a little by removing columns and renaming some columns so we can join the data later:

```
ahrefs_df = ahrefs_raw[['page_url', 'url_rating_desc', 'referring_
domains']]
ahrefs_df = ahrefs_df.rename(columns = {'url_rating_desc': 'page_
authority', 'page_url': 'url'})
ahrefs_df
```

This results in the following:

	url	page_authority	referring_domains
0	https://www.on24.com/	81	3215
1	http://www.on24.com/	54	856
2	https://vshow.on24.com/view/vts/error.html?cod...	52	643
3	https://on24.com/	43	668
4	https://www.on24.com/live-webcast-elite/	41	208
...	...	...	...
210399	https://www.on24.com/ww19internal/	0	1
210400	https://www.on24.com/youre-in/	0	1
210401	https://www-staging.on24.com/	0	1
210402	http://acueductoportachuelo.xyzevent.on24.com/...	0	1
210403	https://golearnership.co.zaevent.on24.com/even...	0	1

210404 rows × 3 columns

Now we have the data in its simplified form which is important because we're not interested in the detail of the links but rather the estimated page-level authority that they import into the target website.

By Site Level

With the data imported and cleaned, the analysis can now commence.

We're always curious to see how many URLs we have at different site levels. We'll achieve this with a quick groupby aggregation function:

```
redir_live_urls.groupby(['crawl_depth']).size()
```

This results in the following:

```
crawl_depth
0              1
1             70
10             5
11             1
```

```
12             1
13             2
14             1
2            303
3            378
4            347
5            253
6            194
7             96
8             33
9             19
Not Set     2351
dtype: int64
```

We can see how Python is treating the crawl depth as a string character rather than a numbered category, which we can fix shortly.

Most of the site URLs can be found in the site depths of 2 to 6. There are over 2351 orphaned URLs, which means these won't inherit any authority unless they have backlinks.

We'll now filter for redirected and live links:

```
redir_live_urls = crawl_data[['url', 'crawl_depth', 'http_status_code',
'indexable', 'no_internal_links_to_url', 'host', 'title']]
```

The dataframe is filtered to include URLs that are indexable:

```
redir_live_urls = redir_live_urls = redir_live_urls.loc[redir_live_
urls['indexable'] == 'Yes']
```

Crawl depth is set as a category and ordered so that Python treats the column variable as a number as opposed to a string character type:

```
redir_live_urls['crawl_depth'])
redir_live_urls['crawl_depth'] = redir_live_urls['crawl_depth'].
astype('category')
redir_live_urls['crawl_depth'] = redir_live_urls['crawl_depth'].cat.
reorder_categories(['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '10',
'Not Set'
```

```
            ])
redir_live_urls = redir_live_urls.loc[redir_live_urls.host == website]
redir_live_urls.drop('host', axis = 1, inplace = True)

redir_live_urls
```

This results in the following:

	url	crawl_depth	http_status_code	indexable	indexable_status	no_internal_links_to_url	title
0	https://www.on24.com/	0	200	Yes	Indexable	4139	Webinar Software & Virtual Event Platform | ON24
2	https://www.on24.com/contact-us/	1	200	Yes	Indexable	11189	Contact Us | Global Office Locations | ON24 Te...
3	https://www.on24.com/resources/	1	200	Yes	Indexable	11155	Webinar, White Paper, and Video Resources | ON24
4	https://www.on24.com/blog/how-juniper-networks...	1	200	Yes	Indexable	17	How Juniper Networks Set Up a Global Virtual S...
6	https://www.on24.com/solutions/manufacturing/	1	200	Yes	Indexable	7414	Platform for Manufacturing Industry | ON24
...	...	...	...	...	...	...	...
6949	https://www.on24.com/customer-stories/shell-ex...	Not Set	200	Yes	Indexable	1	Shell expands global brand awareness and drive...
6950	https://www.on24.com/blog/category/feature-fri...	Not Set	200	Yes	Indexable	0	Feature Friday Archives | ON24
6951	https://www.on24.com/blog/use_cases/certificat...	Not Set	200	Yes	Indexable	0	Certification Archives | ON24
6955	https://www.on24.com/customer-stories/agilent-...	Not Set	200	Yes	Indexable	1	Agilent Optimizes its Global Digital Marketing...
6956	https://www.on24.com/blog/types/beginner/	Not Set	200	Yes	Indexable	0	Beginner Archives | ON24

3483 rows × 7 columns

Let's look at the number of URLs by site level.

```
redir_live_urls.groupby(['crawl_depth']).size()
```

```
crawl_depth
0              1
1             66
2            169
3            280
4            253
5            201
6            122
7             64
```

```
8             17
9              6
10             1
Not Set     2303
dtype: int64
```

Note how the size has dropped slightly to 2303 URLs. The 48 nonindexable URLs were probably paginated pages.

Let's visualize the distribution:

```
from plotnine import *
import matplotlib.pyplot as plt
pd.set_option('display.max_colwidth', None)
%matplotlib inline

# Distribution of internal links to URL by site level
ove_intlink_dist_plt = (ggplot(redir_live_urls, aes(x = 'no_internal_links_
to_url')) +
                    geom_histogram(fill = 'blue', alpha = 0.6, bins = 7) +
                    labs(y = '# Internal Links to URL') +
                    theme_classic() +
                    theme(legend_position = 'none')
                   )

ove_intlink_dist_plt.save(filename = 'images/1_overall_intlink_dist_
plt.png',
                      height=5, width=5, units = 'in', dpi=1000)
ove_intlink_dist_plt
```

The plot ove_intlink_dist_plt in Figure 3-3 is a histogram of the number of internal links to a URL.

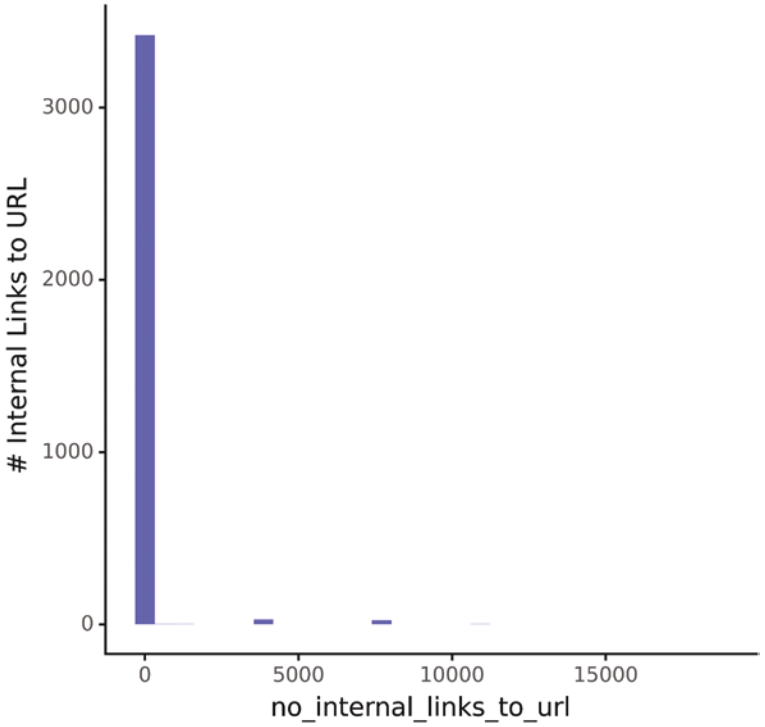

Figure 3-3. *Histogram of the number of internal links to a URL*

The distribution is negatively skewed such that most pages have close to zero links. This would be of some concern to an SEO manager.

While the overall distribution gives one view, it would be good to deep dive into the distribution of internal links by crawl depth:

```
redir_live_urls.groupby('crawl_depth').agg({'no_internal_links_to_url':
['describe']}).sort_values('crawl_depth')
```

This results in the following:

	no_internal_links_to_url							
	describe							
	count	mean	std	min	25%	50%	75%	max
crawl_depth								
0	1.0	4139.000000	NaN	4139.0	4139.00	4139.0	4139.0	4139.0
1	66.0	4808.257576	3507.312178	1.0	3713.00	3732.0	7442.5	18625.0
2	169.0	141.792899	640.547774	0.0	2.00	5.0	8.0	3720.0
3	280.0	3.928571	7.953397	0.0	1.00	2.0	4.0	82.0
4	253.0	2.754941	1.934243	0.0	2.00	2.0	4.0	15.0
5	201.0	2.477612	1.389513	0.0	2.00	2.0	3.0	10.0
6	122.0	2.319672	0.911509	0.0	2.00	2.0	3.0	5.0
7	64.0	1.984375	0.745190	0.0	2.00	2.0	2.0	4.0
8	17.0	1.882353	0.696631	0.0	2.00	2.0	2.0	3.0
9	6.0	1.500000	0.836660	0.0	1.25	2.0	2.0	2.0
10	1.0	2.000000	NaN	2.0	2.00	2.0	2.0	2.0
Not Set	2303.0	0.067304	0.489049	0.0	0.00	0.0	0.0	11.0

The table describes the distribution of internal links by crawl depth or site level. Any URL that is 3+ clicks away from the home page can expect two internal links on average. This is probably the blog content as the marketing team produces a lot of it.

To visualize it graphically

```
from plotnine import *
import matplotlib.pyplot as plt
pd.set_option('display.max_colwidth', None)
%matplotlib inline

intlink_dist_plt = (ggplot(redir_live_urls, aes(x = 'crawl_depth', y = 'no_
internal_links_to_url')) +
                    geom_boxplot(fill = 'blue', alpha = 0.8) +
                    labs(y = '# Internal Links to URL', x = 'Site Level') +
                    theme_classic() +
```

```
        theme(legend_position = 'none')
    )

intlink_dist_plt.save(filename = 'images/1_intlink_dist_plt.png', height=5,
width=5, units = 'in', dpi=1000)
intlink_dist_plt
```

The plot intlink_dist_plt in Figure 3-4 is a histogram of the number of internal links to a URL by site level.

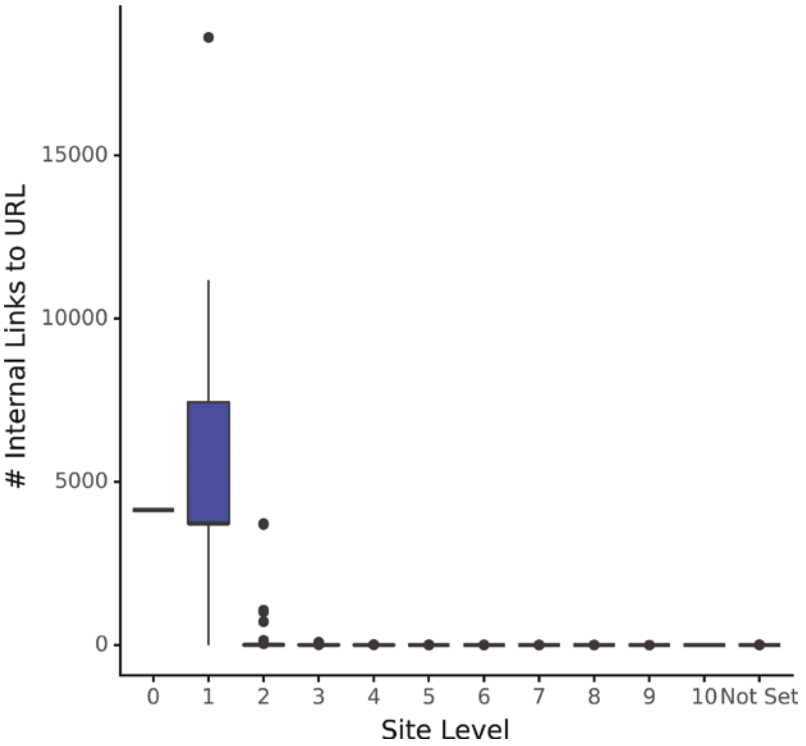

Figure 3-4. *Box plot distributions of the number of internal links to a URL by site level*

As suspected, the most variation is in the first level directly below the home page, with very little variation beyond.

However, we can compare the variation between site levels for content in level 2 and beyond. For a quick peek, we'll use a logarithmic scale for the number of internal links to a URL:

```
from mizani.formatters import comma_format
```

```
intlink_dist_plt = (ggplot(redir_live_urls, aes(x = 'crawl_depth', y = 'no_
internal_links_to_url')) +
                    geom_boxplot(fill = 'blue', alpha = 0.8) +
                    labs(y = '# Internal Links to URL', x = 'Site Level') +
                    scale_y_log10(labels = comma_format()) +
                    theme_classic() +
                    theme(legend_position = 'none')
                   )

intlink_dist_plt.save(filename = 'images/1_log_intlink_dist_plt.png',
height=5, width=5, units = 'in', dpi=1000)
intlink_dist_plt
```

The picture is clearer and more insightful, as we can see how much better and varied the distribution of the lower site levels compared to each other (Figure 3-5).

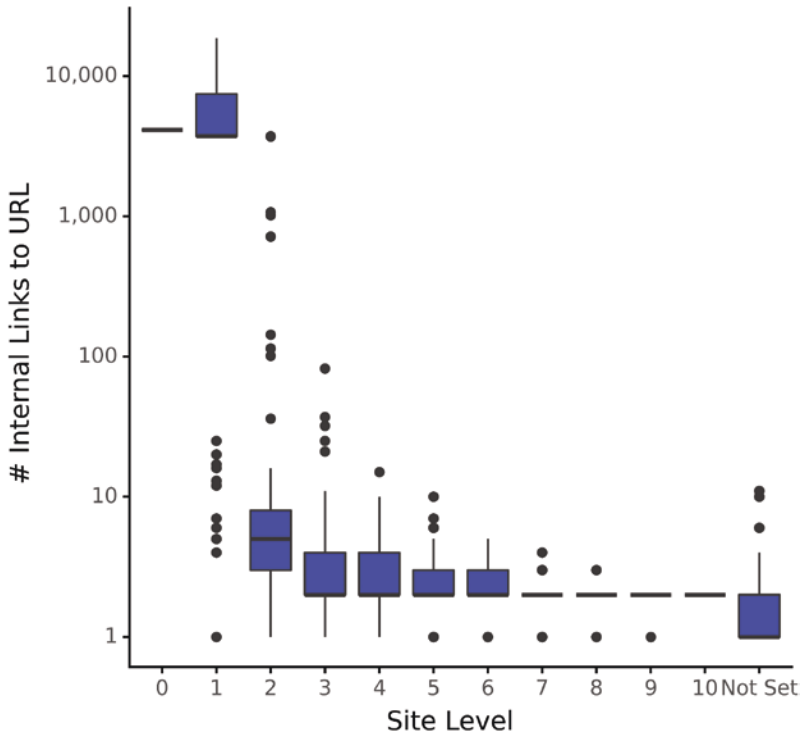

Figure 3-5. *Box plot distribution of the number of internal links by site level with logarized vertical axis*

For example, it's much more obvious that the median number of inbound internal links for pages on site level 2 is much higher than the lower levels.

It's also very obvious that the variation in internal inbound links for pages in site levels 3 and 4 is higher than those in levels 5 and 6.

Remember though the preceding example was achieved using a log scale of the same input variable.

What we've learned here is that having a new variable which is taking a log of the internal links would yield a more helpful picture to compare levels from 2 to 10.

We'll achieve this by creating a new column variable "log_intlinks" which is a log of the internal link count. To avoid negative infinity values from taking a log of zero, we'll add 0.01 to the calculation:

```
redir_live_urls['log_intlinks'] = np.log2(redir_live_urls['no_internal_
links_to_url'] + .01)
```

```
Now we'll plot using the new logarized variable:
```

```
intlink_dist_plt = (ggplot(redir_live_urls, aes(x = 'crawl_depth', y =
'log_intlinks')) +
                    geom_boxplot(fill = 'blue', alpha = 0.8) +
                    labs(y = '# Log Internal Links to URL', x = 'Site
                    Level') +
                    theme_classic() +
                    theme(legend_position = 'none')
                   )
```

```
intlink_dist_plt.save(filename = 'images/1c_loglinks_dist_plt.png',
height=5, width=5, units = 'in', dpi=1000)
intlink_dist_plt
```

The intlink_dist_plt plot (Figure 3-6) is quite similar to the logarized scale, only this time the numbers are easier to read because we're using normal scales for the vertical axis. The comparative averages and variations are easier to compare.

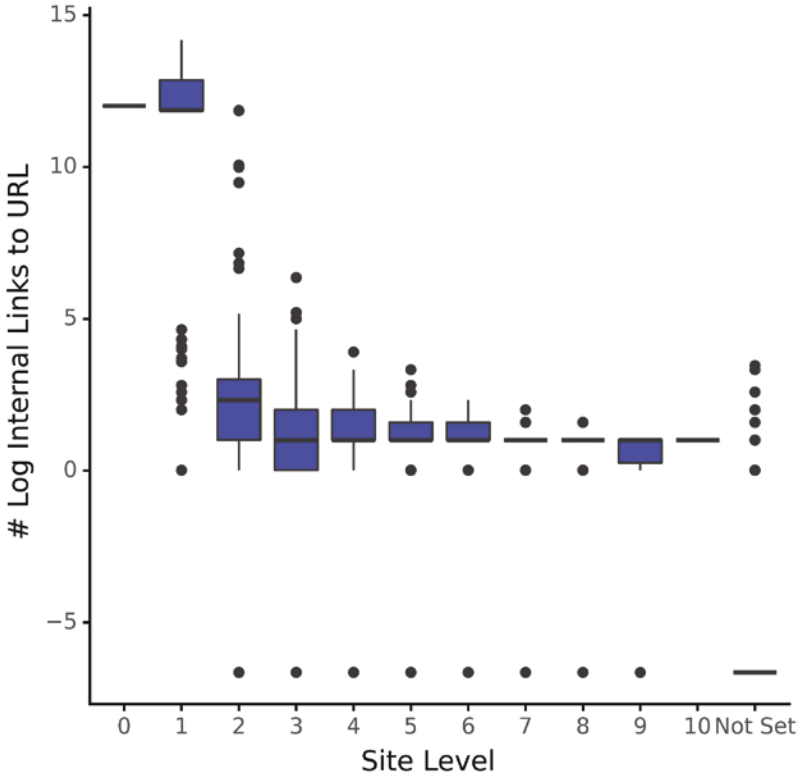

Figure 3-6. *Box plot distributions of logarized internal links by site level*

Site-Level URLs That Are Underlinked

Now that we know the lay of the land in terms of what the distributions look like at the site depth level, we're ready to start digging deeper and see how many URLs are underlinked per site level.

For example, if the 35th percentile number of internal links to a URL is 10 for URLs at a given site level, how many URLs are below that percentile?

That's what we aim to find out. Why 35th and not 25th? It doesn't really matter, a low cutoff point just needs to be picked as the cutoff is arbitrary.

The first step is to calculate the averages of internal links for both nonlog and log versions, which will be joined onto the main dataframe later:

```
intlink_dist = redir_live_urls.groupby('crawl_depth').agg({'no_internal_
links_to_url': ['mean'],
                                                          'log_intlinks':
                                                          ['mean']
```

```
                                                                }).reset_index()
intlink_dist.columns = ['_'.join(col) for col in intlink_dist.
columns.values]
intlink_dist = intlink_dist.rename(columns = {'no_internal_links_to_url_
mean': 'avg_int_links',
                                              'log_intlinks_mean': 'logavg_
                                              int_links',
                                             })
intlink_dist
```

This results in the following:

	crawl_depth_	avg_int_links	logavg_int_links
0	0	4139.000000	12.015070
1	1	4808.257576	10.481776
2	2	141.792899	2.081835
3	3	3.928571	0.907680
4	4	2.754941	0.822429
5	5	2.477612	0.789220
6	6	2.319672	0.843477
7	7	1.984375	0.558034
8	8	1.882353	0.508866
9	9	1.500000	-0.433453
10	10	2.000000	1.007196
11	Not Set	0.067304	-6.396314

The averages are in place by site level. Notice how the log column helps make the range of values between crawl depths less extreme and skewed, that is, 4239 to 0.06 for the average vs. 12 to –6.39 for the log average, which makes it easier to normalize the data.

Now we'll set the lower quantile at 35% for all site levels. This will use a customer function quantile_lower:

```
def quantile_lower(x):
    return x.quantile(.35).round(0)

quantiled_intlinks = redir_live_urls.groupby('crawl_depth').agg({'log_
intlinks':
                                                                [quantile_
                                                                lower]}).
                                                                reset_
                                                                index()
quantiled_intlinks.columns = ['_'.join(col) for col in quantiled_intlinks.
columns.values]
quantiled_intlinks = quantiled_intlinks.rename(columns = {'crawl_depth_':
'crawl_depth',
                                                        'log_intlinks_
                                                        quantile_lower':
                                                        'sd_intlink_
                                                        lowqua'})
quantiled_intlinks
```

This results in the following:

	crawl_depth	sd_intlink_lowqua
0	0	12.0
1	1	12.0
2	2	2.0
3	3	1.0
4	4	1.0
5	5	1.0
6	6	1.0
7	7	1.0
8	8	1.0
9	9	1.0
10	10	1.0
11	Not Set	-7.0

The lower quantile stats are set. Quartiles are limited to the 25th percentile, whereas a quantile means the lower limits can be set to any number, such as 11th, 18th, 24th, etc., which is why we use quantiles instead of quartiles. The next steps are to join the data to the main dataframe, then we'll apply a function to mark URLs that are underlinked for their given site level:

```
redir_live_urls_underidx = redir_live_urls.merge(quantiled_intlinks, on =
'crawl_depth', how = 'left')
```

The following function assesses whether the URL has less links than the lower quantile. If yes, then the value of "sd_int_uidx" is 1, otherwise 0:

```
def sd_intlinkscount_underover(row):
    if row['sd_intlink_lowqua'] > row['log_intlinks']:
        val = 1
    else:
        val = 0
    return val

redir_live_urls_underidx['sd_int_uidx'] = redir_live_urls_underidx.
apply(sd_intlinkscount_underover, axis=1)
```

There's some code to account for "Not Set" which are effectively orphaned URLs. In this instance, we set these to 1 – meaning they're underlinked:

```
redir_live_urls_underidx['sd_int_uidx'] = np.where(redir_live_urls_
underidx['crawl_depth'] == 'Not Set', 1,
                                                   redir_live_urls_
                                                   underidx['sd_int_uidx'])

redir_live_urls_underidx
```

This results in the following:

url	crawl_depth	http_status_code	indexable	indexable_status	no_internal_links_to_url	title	log_intlinks	sd_intlink_lowqua	sd_int_uidx
ttps://www.on24.com/	0	200	Yes	Indexable	4139	Webinar Software & Virtual Event Platform | ON24	12.015070	12.0	0
on24.com/contact-us/	1	200	Yes	Indexable	11189	Contact Us | Global Office Locations | ON24 Teams | ON24	13.449795	12.0	0
.on24.com/resources/	1	200	Yes	Indexable	11155	Webinar, White Paper, and Video Resources | ON24	13.445404	12.0	0
now-juniper-networks-global-virtual-summit/	1	200	Yes	Indexable	17	How Juniper Networks Set Up a Global Virtual Summit | ON24 Blog	4.088311	12.0	1
lutions/manufacturing/	1	200	Yes	Indexable	7414	Platform for Manufacturing Industry | ON24	12.856038	12.0	0
...	...	...	...	...	...	...	...	...	...

The dataframe shows that the column is in place marking underlinked URLs as 1. With the URLs marked, we're ready to get an overview of how under-linked the URLs are, which will be achieved by aggregating by crawl depth and summing the total number of underlinked URLs:

```
intlinks_agged = redir_live_urls_underidx.groupby('crawl_depth').agg({'sd_
int_uidx': ['sum', 'count']}).reset_index()
```

The following line tidies up the column names by inserting an underscore using a list comprehension:

```
intlinks_agged.columns = ['_'.join(col) for col in intlinks_agged.
columns.values]
intlinks_agged = intlinks_agged.rename(columns = {'crawl_depth_': 'crawl_
depth'})
```

To get a proportion (or percentage), we divide the sum by the count and multiply by 100:

```
intlinks_agged['sd_uidx_prop'] = (intlinks_agged.sd_int_uidx_sum) /
intlinks_agged.sd_int_uidx_count * 100

print(intlinks_agged)
```

This results in the following:

	crawl_depth	sd_int_uidx_sum	sd_int_uidx_count	sd_uidx_prop
0	0	0	1	0.000000
1	1	38	66	57.575758
2	2	67	169	39.644970
3	3	75	280	26.785714
4	4	57	253	22.529644
5	5	31	201	15.422886
6	6	9	122	7.377049
7	7	9	64	14.062500
8	8	3	17	17.647059
9	9	2	6	33.333333
10	10	0	1	0.000000
11	Not Set	2303	2303	100.000000

So even though the content in levels 1 and 2 have more links than any of the lower levels, they have a higher proportion of underlinked URLs than any other site level (apart from the orphans in Not Set of course).

For example, 57% of pages just below the home page are underlinked.

Let's visualize:

```
# plot the table
depth_uidx_plt = (ggplot(intlinks_agged, aes(x = 'crawl_depth', y = 'sd_
int_uidx_sum')) +
                    geom_bar(stat = 'identity', fill = 'blue', alpha
                    = 0.8) +
                    labs(y = '# Under Linked URLs', x = 'Site Level') +
                    scale_y_log10() +
                    theme_classic() +
                    theme(legend_position = 'none')
                   )

depth_uidx_plt.save(filename = 'images/1_depth_uidx_plt.png', height=5,
width=5, units = 'in', dpi=1000)
depth_uidx_plt
```

It's good to visualize using depth_uidx_plt because we can also see (Figure 3-7) that levels 2, 3, and 4 have the most underlinked URLs by volume.

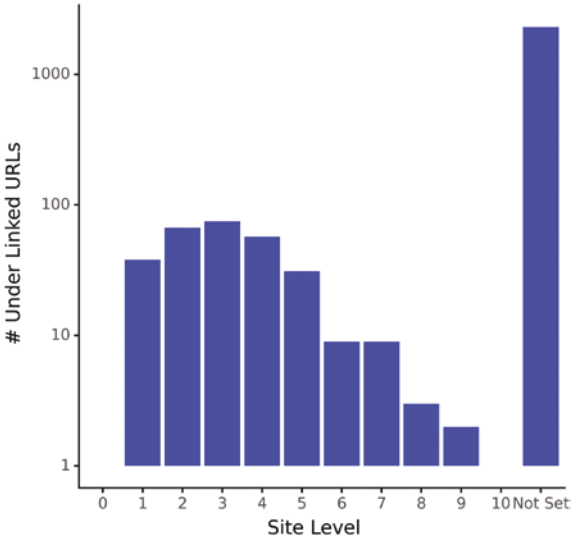

Figure 3-7. *Column chart of the number of internally under-linked URLs by site level*

Let's plot the intlinks_agged table:

```
depth_uidx_prop_plt = (ggplot(intlinks_agged, aes(x = 'crawl_depth', y =
'sd_uidx_prop')) +
                    geom_bar(stat = 'identity', fill = 'blue', alpha
                    = 0.8) +
                    labs(y = '% URLs Under Linked', x = 'Site Level') +
                    theme_classic() +
                    theme(legend_position = 'none')
                   )

depth_uidx_prop_plt.save(filename = 'images/1_depth_uidx_prop_plt.png',
height=5, width=5, units = 'in', dpi=1000)
depth_uidx_prop_plt
```

Plotting depth_uidx_prop_plt (Figure 3-8), we see it just so happens that although level 1 has a lower volume, the proportion is higher. Intuitively, this is indicative of too many pages being linked from the home page but unequally.

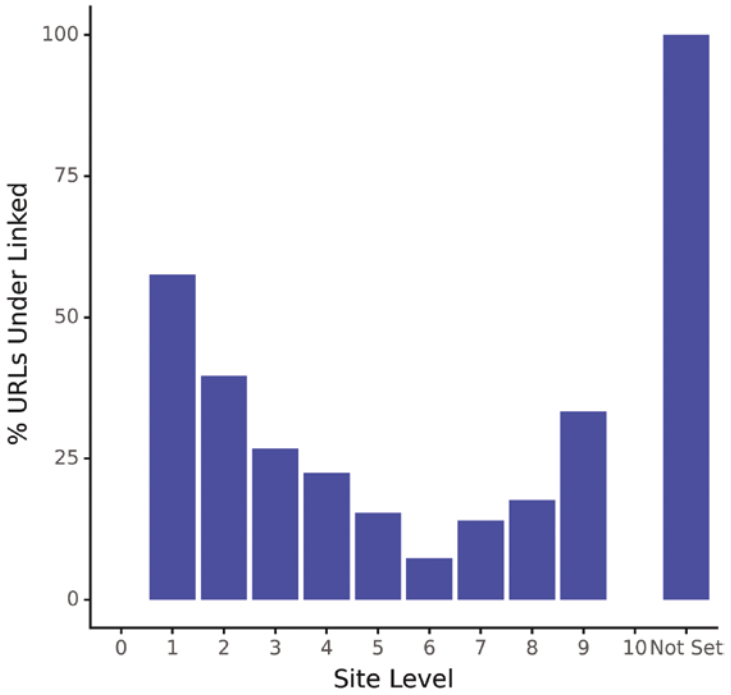

Figure 3-8. *Column chart of the proportion of under internally linked URLs by site level*

It's not a given that URLs in the site level that are underlinked are a problem or perhaps more so by design. However, they are worth reviewing as perhaps they should be at that site level or they do deserve more internal links after all.

The following code exports the underlinked URLs to a CSV which can be viewed in Microsoft Excel:

```
underlinked_urls = redir_live_urls_underidx.loc[redir_live_urls_underidx.
sd_int_uidx == 1]
underlinked_urls = underlinked_urls.sort_values(['crawl_depth', 'no_
internal_links_to_url'])
underlinked_urls.to_csv('exports/underlinked_urls.csv')
```

By Page Authority

Inbound links from external websites are a source of PageRank or, if we're going to be search engine neutral about it, page authority.

Given that not all pages earn inbound links, it is normally desired by SEOs to have pages without backlinks crawled more often. So it would make sense to analyze and explore opportunities to redistribute this PageRank to other pages within the website.

We'll start by tacking on the AHREFs data to the main dataframe so we can see internal links by page authority.

```
intlinks_pageauth = redir_live_urls_underidx.merge(ahrefs_df, on = 'url',
how = 'left')
intlinks_pageauth.head()
```

This results in the following:

url	crawl_depth	http_status_code	indexable_status	no_internal_links_to_url	title	log_intlinks	page_authority	referring_domains
https://www.on24.com/	0	200	Indexable	4139	Webinar Software & Virtual Event Platform | ON24	12.015070	81.0	3215.0
://www.on24.com/contact-us/	1	200	Indexable	11189	Contact Us | Global Office Locations | ON24 Teams | ON24	13.449795	36.0	55.0
s://www.on24.com/resources/	1	200	Indexable	11155	Webinar, White Paper, and Video Resources | ON24	13.445404	28.0	42.0
w.on24.com/blog/how-juniper-et-up-a-global-virtual-summit/	1	200	Indexable	17	How Juniper Networks Set Up a Global Virtual Summit | ON24 Blog	4.088311	13.0	1.0
.com/solutions/manufacturing/	1	200	Indexable	7414	Platform for Manufacturing Industry | ON24	12.856038	17.0	7.0

We now have page authority and referring domains at the URL level. Predictably, the home page has a lot of referring domains (over 3000) and the most page-level authority at 81.

As usual, we'll perform some aggregations and explore the distribution of the PageRank (interchangeable with page authority).

First, we'll clean up the data to make sure we replace null values with zero:

```
intlinks_pageauth['page_authority'] = np.where(intlinks_pageauth['page_
authority'].isnull(),
                                               0, intlinks_pageauth['page_
                                               authority'])
```

Aggregate by page authority:

```
intlinks_pageauth.groupby('page_authority').agg({'no_internal_links_to_
url': ['describe']})
```

This results in the following:

	no_internal_links_to_url							
	describe							
	count	mean	std	min	25%	50%	75%	max
page_authority								
0.0	1320.0	0.034848	0.240667	0.0	0.0	0.0	0.00	3.0
13.0	1077.0	7.839369	120.726053	0.0	0.0	2.0	3.00	3698.0
14.0	148.0	79.763514	524.905542	0.0	0.0	2.0	3.00	3720.0
15.0	725.0	13.477241	200.634714	0.0	0.0	0.0	0.00	3716.0
16.0	67.0	336.134328	1250.082093	0.0	0.0	2.0	5.00	7413.0
17.0	38.0	1082.684211	2095.249390	0.0	2.0	5.0	110.25	7414.0
18.0	22.0	1015.090909	2604.950231	0.0	2.0	3.0	6.75	7436.0
19.0	22.0	510.090909	1304.569147	0.0	2.0	3.5	8.00	3733.0
20.0	15.0	1735.666667	3093.162267	0.0	1.0	4.0	1877.00	7429.0

The preceding table shows the distribution of internal links by different levels of page authority.

At the lower levels, most URLs have around two internal links.

A graph will give us the full picture:

```
# distribution of page_authority
page_authority_dist_plt = (ggplot(intlinks_pageauth, aes(x = 'page_
authority')) +
                    geom_histogram(fill = 'blue', alpha = 0.6, bins
                    = 30 ) +
                    labs(y = '# URLs', x = 'Page Authority') +
                    #scale_y_log10() +
                    theme_classic() +
                    theme(legend_position = 'none')
                   )
```

```
page_authority_dist_plt.save(filename = 'images/2_page_authority_dist_
plt.png',
                             height=5, width=5, units = 'in', dpi=1000)
page_authority_dist_plt
```

The distribution, shown in page_authority_dist_plt (Figure 3-9), is heavily negatively skewed when plotting the raw numbers. Most of the site URLs have a PageRank of 15, of which the number of URLs with higher authority shrinks dramatically. A very high number of URLs have no authority, because they are orphaned.

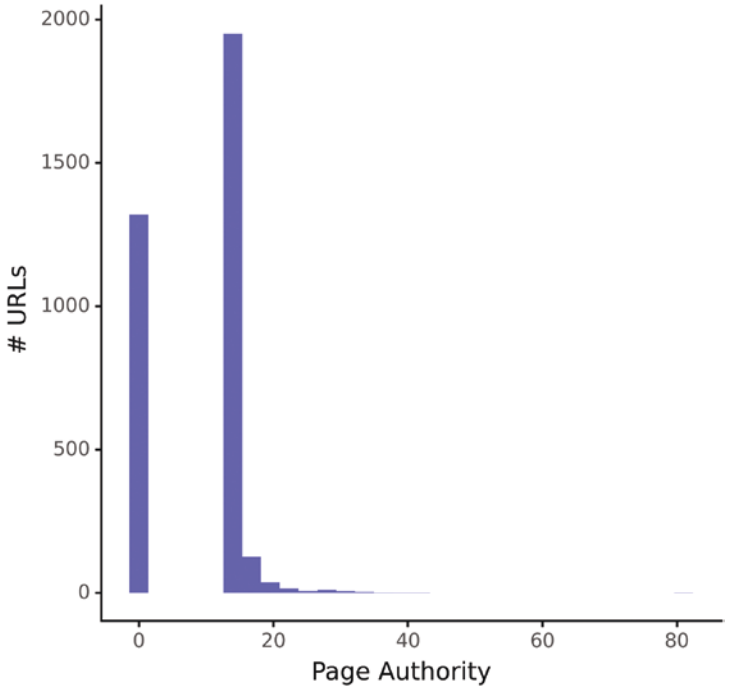

Figure 3-9. *Distribution of URLs by page authority*

Using the log scale, we can see how the higher levels of authority compare:

```
# distribution of page_authority
page_authority_dist_plt = (ggplot(intlinks_pageauth, aes(x = 'page_
authority')) +
                    geom_histogram(fill = 'blue', alpha = 0.6, bins
                    = 30 ) +
                    labs(y = '# URLs (Log)', x = 'Page Authority') +
```

```
    scale_y_log10() +
    theme_classic() +
    theme(legend_position = 'none')
  )

page_authority_dist_plt.save(filename = 'images/2_page_authority_dist_log_
plt.png',
                             height=5, width=5, units = 'in', dpi=1000)
page_authority_dist_plt
```

Suddenly, the view shown by page_authority_dist_plt (Figure 3-10) is more interesting because as authority increases by an increment of one, there are ten times less URLs than before – a pretty harsh distribution of PageRank.

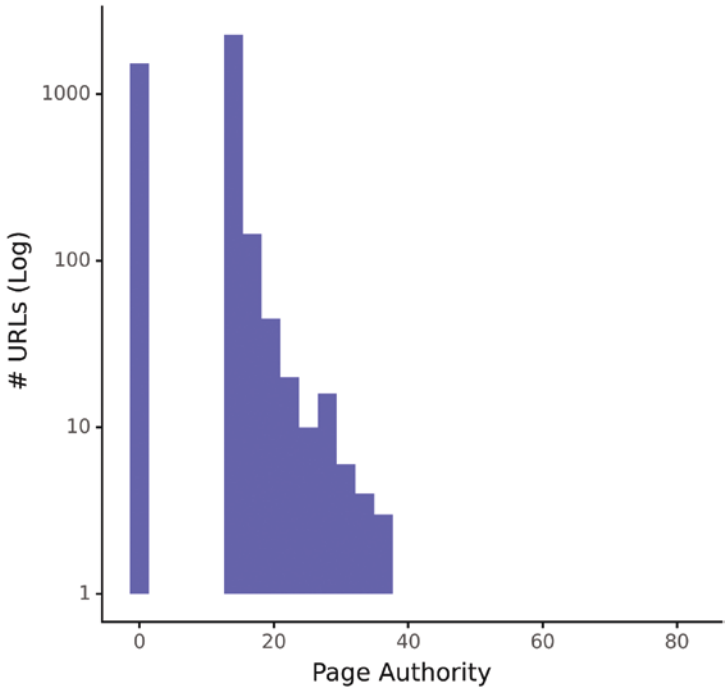

Figure 3-10. *Distribution plot of URLs by logarized scale*

Given this more insightful view, taking a log of "page_authority" to form a new column variable "log_pa" is justified:

```
intlinks_pageauth['page_authority'] = np.where(intlinks_pageauth['page_
authority'] == 0, .1, intlinks_pageauth['page_authority'])
intlinks_pageauth['log_pa'] = np.log2(intlinks_pageauth.page_authority)
intlinks_pageauth.head()
```

	url	crawl_depth	http_status_code	indexable_status	no_internal_links_to_url	title	log_intlinks	page_authority	referring_domains	log_pa
	://www.on24.com/	0	200	Indexable	4139	Webinar Software & Virtual Event Platform | ON24	12.015070	81.0	3215.0	6.339850
	4.com/contact-us/	1	200	Indexable	11189	Contact Us | Global Office Locations | ON24 Teams | ON24	13.449795	36.0	55.0	5.169925
	24.com/resources/	1	200	Indexable	11155	Webinar, White Paper, and Video Resources | ON24	13.445404	28.0	42.0	4.807355
	/blog/how-juniper-bal-virtual-summit/	1	200	Indexable	17	How Juniper Networks Set Up a Global Virtual Summit | ON24 Blog	4.088311	13.0	1.0	3.700440
	ons/manufacturing/	1	200	Indexable	7414	Platform for Manufacturing Industry | ON24	12.856038	17.0	7.0	4.087463

The log_pa column is in place; let's visualize:

```
page_authority_trans_dist_plt = (ggplot(intlinks_pageauth, aes(x =
'log_pa')) +
                    geom_histogram(fill = 'blue', alpha = 0.6, bins
                    = 30 ) +
                    labs(y = '# URLs (Log)', x = 'Log Page Authority') +
                    scale_y_log10() +
                    theme_classic() +
                    theme(legend_position = 'none')
                   )

page_authority_trans_dist_plt.save(filename = 'images/2_page_authority_
trans_dist_plt.png',
                              height=5, width=5, units = 'in', dpi=1000)
page_authority_trans_dist_plt
```

Taking a log has compressed the range of PageRank, as shown by page_authority_trans_dist_plt (Figure 3-11), by making it less extreme as the home page has a log_pa value of 6, bringing it closer to the rest of the site.

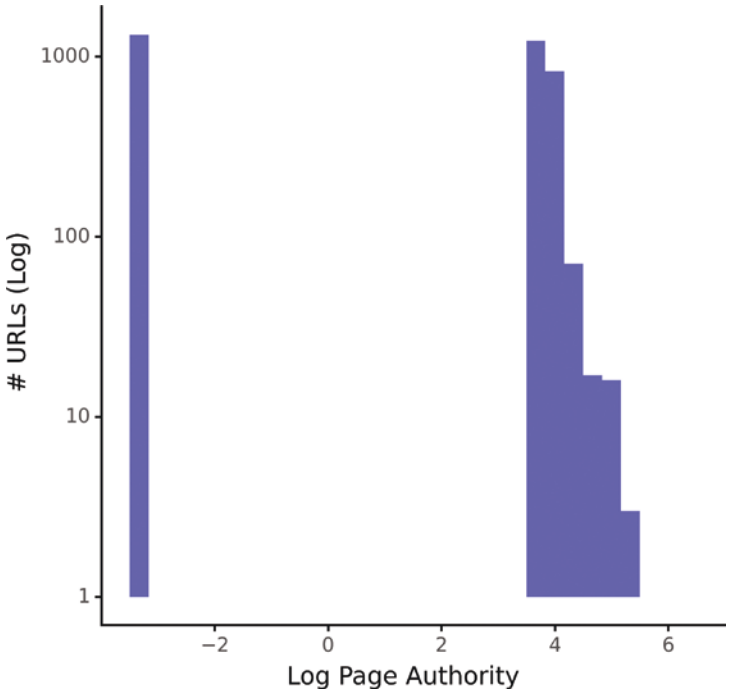

Figure 3-11. *Distribution of URLs by log page authority*

The decimal points will be rounded to make the 3000+ URLs easier to categorize:

```
intlinks_pageauth['pa_band'] = intlinks_pageauth['log_pa'].apply(np.floor)

# display updated DataFrame
intlinks_pageauth
```

wl_depth	http_status_code	indexable	indexable_status	no_internal_links_to_url	title	log_intlinks	page_authority	referring_domains	log_pa	pa_band
0	200	Yes	Indexable	4139	Webinar Software & Virtual Event Platform | ON24	12.015070	81.0	3215.0	6.339850	6.0
1	200	Yes	Indexable	11189	Contact Us | Global Office Locations | ON24 Teams | ON24	13.449795	36.0	55.0	5.169925	5.0
1	200	Yes	Indexable	11155	Webinar, White Paper, and Video Resources | ON24	13.445404	28.0	42.0	4.807355	4.0
1	200	Yes	Indexable	17	How Juniper Networks Set Up a Global Virtual Summit | ON24 Blog	4.088311	13.0	1.0	3.700440	3.0
1	200	Yes	Indexable	7414	Platform for Manufacturing Industry | ON24	12.856038	17.0	7.0	4.087463	4.0

Page Authority URLs That Are Underlinked

With the URLs categorized into PA bands, we want to see if they have less internal links for their authority level than they should. We've set the threshold at 40% so that any URL that has less internal links for their level of PA will be counted as underlinked.

The choice of 40% is not terribly important at this stage as each website (or market even) is different. There are more scientific ways of arriving at the optimal threshold, such as analyzing top-ranking competitors for a search space; however, for now we'll choose 40% as our threshold.

```
def quantile_lower(x):
    return x.quantile(.4).round(0)

quantiled_pageau = intlinks_pageauth.groupby('pa_band').agg({'no_internal_
links_to_url': [quantile_lower]}).reset_index()
quantiled_pageau.columns = ['_'.join(col) for col in quantiled_pageau.
columns.values]
quantiled_pageau = quantiled_pageau.rename(columns = {'pa_band_':
'pa_band',
                                                      'no_internal_links_
                                                      to_url_quantile_
                                                      lower': 'pa_intlink_
                                                      lowqua'})
quantiled_pageau
```

This results in the following:

	pa_band	pa_intlink_lowqua
0	-4.0	0.0
1	3.0	0.0
2	4.0	3.0
3	5.0	7446.0
4	6.0	4139.0

Going by PageRank, we now have the minimum threshold of inbound internal links we would expect. Time to join the data and mark the URLs that are underlinked for their authority level:

```
intlinks_pageauth_underidx = intlinks_pageauth.merge(quantiled_pageau, on =
'pa_band', how = 'left')
```

```
def pa_intlinkscount_underover(row):
    if row['pa_intlink_lowqua'] > row['no_internal_links_to_url']:
        val = 1
    else:
        val = 0
    return val
```

```
intlinks_pageauth_underidx['pa_int_uidx'] = intlinks_pageauth_underidx.
apply(pa_intlinkscount_underover, axis=1)
```

This function will allow us to make some aggregations to see how many URLs there are at each PageRank band and how many are under-linked:

```
pageauth_agged = intlinks_pageauth_underidx.groupby('pa_band').agg({'pa_
int_uidx': ['sum', 'count']}).reset_index()
pageauth_agged.columns = ['_'.join(col) for col in pageauth_agged.
columns.values]
```

```
pageauth_agged['uidx_prop'] = pageauth_agged.pa_int_uidx_sum / pageauth_
agged.pa_int_uidx_count * 100
```

```
print(pageauth_agged)
```

This results in the following:

```
  pa_band_  pa_int_uidx_sum  pa_int_uidx_count  uidx_prop
0      -4.0               0               1320   0.000000
1       3.0               0               1950   0.000000
2       4.0              77                203  37.931034
3       5.0               4                  9  44.444444
4       6.0               0                  1   0.000000
```

Most of the underlinked content appears to be those that have the highest page authority, which is slightly contrary to what the site-level approach suggests (that pages lower down are underlinked). That's assuming most of the high authority pages are closer to the home page.

What is the right answer? It depends on what we're trying to achieve. Let's continue with more analysis for now and visualize the authority stats:

```
# distribution of page_authority
pageauth_agged_plt = (ggplot(intlinks_pageauth_underidx.loc[intlinks_
pageauth_underidx['pa_int_uidx'] == 1],
                           aes(x = 'pa_band')) +
                     geom_histogram(fill = 'blue', alpha = 0.6, bins = 10) +
                     labs(y = '# URLs Under Linked', x = 'Page Authority
                     Level') +
                     theme_classic() +
                     theme(legend_position = 'none')
                    )

pageauth_agged_plt.save(filename = 'images/2_pageauth_agged_hist.png',
                        height=5, width=5, units = 'in', dpi=1000)
pageauth_agged_plt
```

We see in pageauth_agged_plt (Figure 3-12) that there are almost 80 URLs underlinked at PageRank level 4 and a few at PageRank level 5. This is quite an abstract concept admittedly.

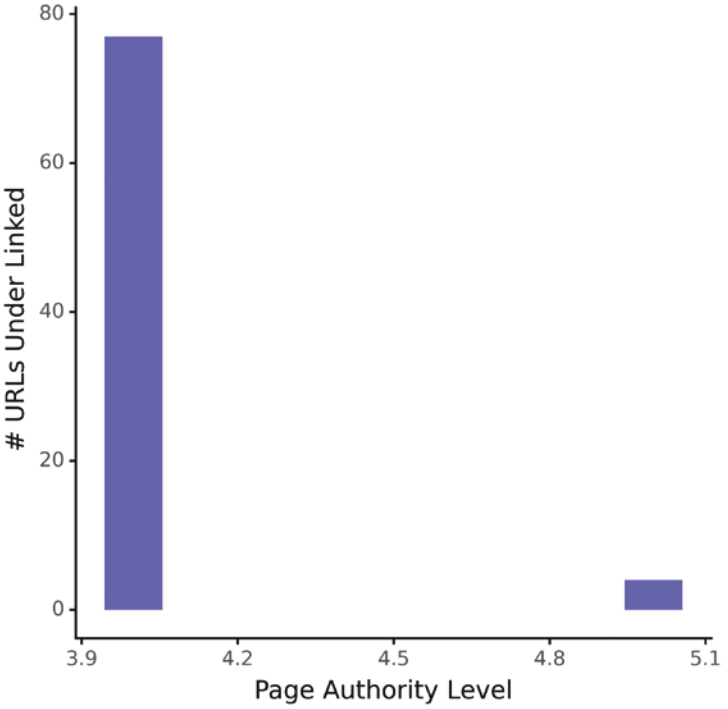

Figure 3-12. *Distribution of under internally linked URLs by page authority level*

Content Type

Perhaps it would be more useful to visualize this by content type just by a "quick and dirty" analysis using the first subdirectory:

```
intlinks_content_underidx = intlinks_depthauth_underidx.copy()
```

To get the first subfolder, we'll define a function that allows the operation to continue in case of a fail (which would happen for the home page URL because there is no subfolder). The k parameter specifies the number of slashes in the URL to find the desired folder and parse the subdirectory name:

```
def get_folder(fp, k=3):
    try:
        return os.path.split(fp)[0].split(os.sep)[k]
    except:
        return 'home'
```

```
intlinks_content_underidx['content'] = intlinks_content_underidx['url'].
apply(lambda x: get_folder(x))
```

Inspect the distribution of links by subfolder:

```
intlinks_content_underidx.groupby('content').agg({'no_internal_links_to_
url': ['describe']})
```

This results in the following:

	no_internal_links_to_url							
	describe							
	count	mean	std	min	25%	50%	75%	max
content								
about-us	6.0	6203.000000	1921.783443	3713.0	4658.50	7442.0	7443.75	7446.0
accelerate-pipeline-on24	1.0	3.000000	NaN	3.0	3.00	3.0	3.00	3.0
accessibility	1.0	0.000000	NaN	0.0	0.00	0.0	0.00	0.0
act-on	1.0	0.000000	NaN	0.0	0.00	0.0	0.00	0.0
add-on-services	1.0	0.000000	NaN	0.0	0.00	0.0	0.00	0.0
...	...	...	...	...	...	...	...	...
webinarworldondemand-london	1.0	0.000000	NaN	0.0	0.00	0.0	0.00	0.0
webinarworldondemand-sydney	1.0	0.000000	NaN	0.0	0.00	0.0	0.00	0.0
webinerd-community	14.0	3.928571	3.452185	0.0	2.00	3.5	5.75	11.0
webinerd-education	2.0	7.500000	10.606602	0.0	3.75	7.5	11.25	15.0
zapier	1.0	0.000000	NaN	0.0	0.00	0.0	0.00	0.0

183 rows × 8 columns

Wow, 183 subfolders! That's way too much for categorical analysis. We could break it down and aggregate it into fewer categories using the ngram techniques described in Chapter 9; feel free to try.

In any case, it looks like the site architecture is too flat and could be better structured to be more hierarchical, that is, more pyramid like.

Also, many of the content folders only have one inbound internal link, so even without the benefit of data science, it's obvious these require SEO attention.

Combining Site Level and Page Authority

Perhaps it would be more useful to visualize by combining site level and page authority?

```
intlinks_depthauth_underidx = intlinks_pageauth_underidx.copy()
intlinks_depthauth_underidx['depthauth_uidx'] = np.where((intlinks_
depthauth_underidx['sd_int_uidx'] +
                                                         intlinks_
                                                         depthauth_
                                                         underidx['pa_
                                                         int_uidx'] ==
                                                         2), 1, 0)

'''intlinks_depthauth_underidx['depthauth_uidx'] = np.where((intlinks_
depthauth_underidx['sd_int_uidx'] == 1) &
                                                         (intlinks_
                                                         depthauth_
                                                         underidx['pa_int_
                                                         uidx'] == 1),
                                                         1, 0)'''

depthauth_uidx = intlinks_depthauth_underidx.groupby(['crawl_depth',
'pa_band']).agg({'depthauth_uidx': 'sum'}).reset_index()
depthauth_urls = intlinks_depthauth_underidx.groupby(['crawl_depth',
'pa_band']).agg({'url': 'count'}).reset_index()

depthauth_stats = depthauth_uidx.merge(depthauth_urls,
                                                        on = ['crawl_depth',
                                                        'pa_band'], how = 'left')
depthauth_stats['depthauth_uidx_prop'] = (depthauth_stats['depthauth_uidx']
/ depthauth_stats['url']).round(2)
depthauth_stats.sort_values('depthauth_uidx', ascending = False)
```

This results in the following:

	crawl_depth	pa_band	depthauth_uidx	url	depthauth_uidx_prop
57	Not Set	4.0	42	44	0.95
12	2	4.0	7	50	0.14
17	3	4.0	5	24	0.21
22	4	4.0	4	18	0.22
27	5	4.0	2	12	0.17
58	Not Set	5.0	1	1	1.00
32	6	4.0	1	4	0.25
47	9	4.0	1	1	1.00
0	0	-4.0	0	0	NaN
40	8	-4.0	0	0	NaN
41	8	3.0	0	15	0.00

Most of the underlinked URLs are orphaned and have page authority (probably from backlinks).

Visualize to get a fuller picture:

```
depthauth_stats_plt = (
    ggplot(depthauth_stats,
           aes(x = 'pa_band', y = 'crawl_depth', fill = 'depthauth_
           uidx')) +
    geom_tile(stat = 'identity', alpha = 0.6) +
    labs(y = '', x = '') +
    theme_classic() +
    theme(legend_position = 'right')
)

depthauth_stats_plt.save(filename = 'images/3_depthauth_stats_plt.png',
                         height=5, width=10, units = 'in', dpi=1000)
depthauth_stats_plt
```

There we have it, depthauth_stats_plt (Figure 3-13) shows most of the focus should go into the orphaned URLs (which they should anyway), but more importantly we know which orphaned URLs to prioritize over others.

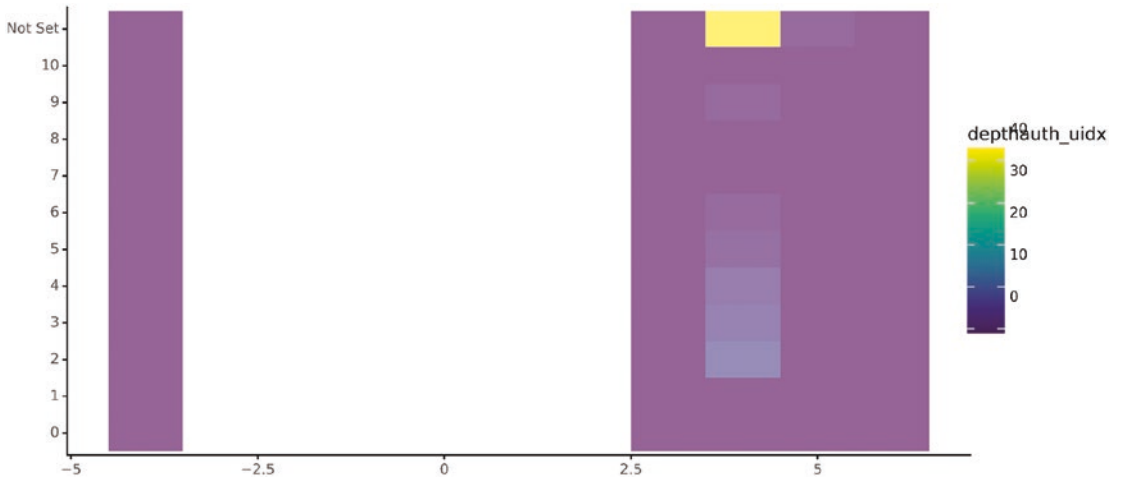

Figure 3-13. *Heatmap of page authority level, site level, and underlinked URLs*

We can also see the extent of the issue. The second highest priority group of underindexed URLs are at site levels 2, 3, and 4.

Anchor Texts

If the count and their distribution represent the quantitative aspect of internal links, then the anchor texts could be said to represent their quality.

Anchor texts signal to search engines and users what content to expect after accessing the hyperlink. This makes anchor texts an important signal and one worth optimizing.

We'll start by aggregating the crawl data from Sitebulb to get an overview of the issues:

```
anchor_issues_agg = crawl_data.agg({'no_anchors_with_empty_href': ['sum'],
                'no_anchors_with_leading_or_trailing_whitespace_in_href':
                ['sum'],
                'no_anchors_with_local_file': ['sum'],
                'no_anchors_with_localhost': ['sum'],
                'no_anchors_with_malformed_href': ['sum'],
                'no_anchors_with_no_text': ['sum'],
                'no_anchors_with_non_descriptive_text': ['sum'],
                'no_anchors_with_non-http_protocol_in_href': ['sum'],
```

```
                'no_anchors_with_url_in_onclick': ['sum'],
                'no_anchors_with_username_and_password_in_href': ['sum'],
                'no_image_anchors_with_no_alt_text': ['sum']
               }).reset_index()

anchor_issues_agg = pd.melt(anchor_issues_agg, var_name=['issues'],
                            value_vars=['no_anchors_with_empty_href',
                                        'no_anchors_with_leading_or_
                                        trailing_whitespace_in_href',
                                        'no_anchors_with_local_file','no_
                                        anchors_with_localhost',
                                        'no_anchors_with_malformed_href',
                                        'no_anchors_with_no_text',
                                        'no_anchors_with_non_
                                        descriptive_text',
                                        'no_anchors_with_non-http_protocol_
                                        in_href',
                                        'no_anchors_with_url_in_onclick',
                                        'no_anchors_with_username_and_
                                        password_in_href',
                                        'no_image_anchors_with_no_
                                        alt_text'],
                            value_name='instances'
                           )
anchor_issues_agg
```

This results in the following:

	issues	instances
0	no_anchors_with_empty_href	19
1	no_anchors_with_leading_or_trailing_whitespace_in_href	3724
2	no_anchors_with_local_file	0
3	no_anchors_with_localhost	0
4	no_anchors_with_malformed_href	11
5	no_anchors_with_no_text	297
6	no_anchors_with_non_descriptive_text	4047
7	no_anchors_with_non-http_protocol_in_href	0
8	no_anchors_with_url_in_onclick	0
9	no_anchors_with_username_and_password_in_href	0
10	no_image_anchors_with_no_alt_text	112

Over 4000 links with no descriptive anchor text jump out as the most common issue, not to mention the 19 anchors with empty HREF (albeit very low in number).

To visualize

```
anchor_issues_count_plt = (ggplot(anchor_issues_agg, aes(x =
'reorder(issues, instances)', y = 'instances')) +
                    om_bar(stat = 'identity', fill = 'blue', alpha = 0.6) +
                    labs(y = '# instances of Anchor Text Issues', x = '') +
                    theme_classic() +
                    coord_flip() +
                    theme(legend_position = 'none')
                   )

anchor_issues_count_plt.save(filename = 'images/4_anchor_issues_count_
plt.png',
                         height=5, width=5, units = 'in', dpi=1000)
anchor_issues_count_plt
```

anchor_issues_count_plt (Figure 3-14) visually confirms the number of internal links with nondescriptive anchor text.

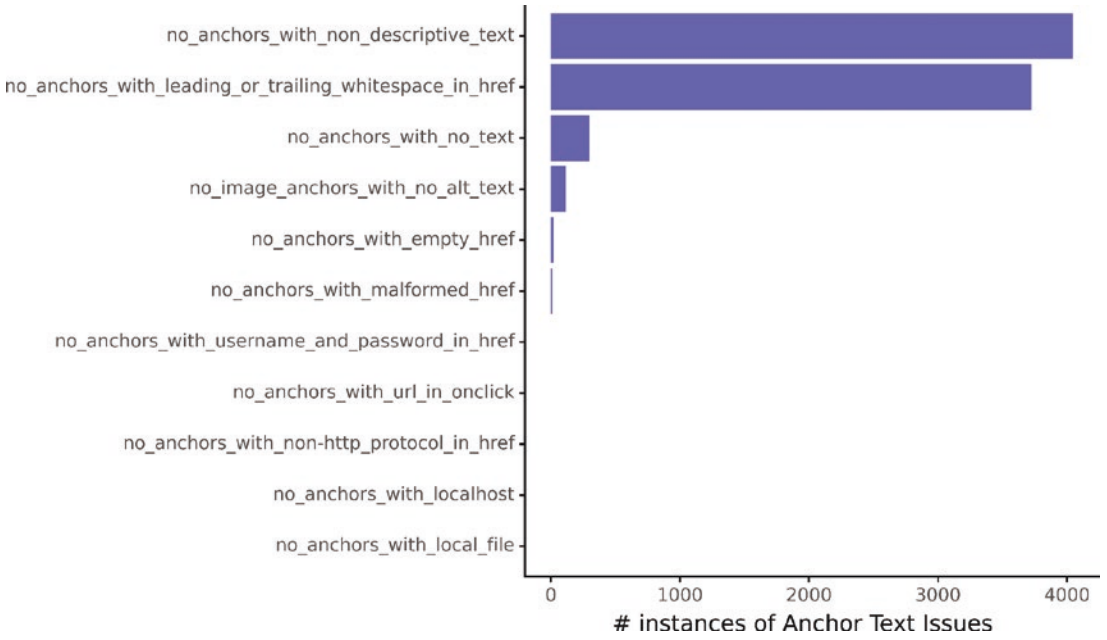

Figure 3-14. *Bar chart of anchor text issues*

Anchor Issues by Site Level

We'll drill down on the preceding example by site level to get a bit more insight to see where the problems are happening:

```
anchor_issues_levels = crawl_data.groupby('crawl_depth').agg({'no_anchors_
with_empty_href': ['sum'],
                'no_anchors_with_leading_or_trailing_whitespace_in_href':
                ['sum'],
                'no_anchors_with_local_file': ['sum'],
                'no_anchors_with_localhost': ['sum'],
                'no_anchors_with_malformed_href': ['sum'],
                'no_anchors_with_no_text': ['sum'],
                'no_anchors_with_non_descriptive_text': ['sum'],
                'no_anchors_with_non-http_protocol_in_href': ['sum'],
                'no_anchors_with_url_in_onclick': ['sum'],
                'no_anchors_with_username_and_password_in_href': ['sum'],
                'no_image_anchors_with_no_alt_text': ['sum']
               }).reset_index()
```

```
anchor_issues_levels.columns = ['_'.join(col) for col in anchor_issues_
levels.columns.values]
anchor_issues_levels.columns = [str.replace(col, '_sum', '') for col in
anchor_issues_levels.columns.values]
anchor_issues_levels.columns = [str.replace(col, 'no_anchors_with_', '')
for col in anchor_issues_levels.columns.values]
anchor_issues_levels = anchor_issues_levels.rename(columns = {'crawl_
depth_': 'crawl_depth'})

anchor_issues_levels = pd.melt(anchor_issues_levels, id_vars=['crawl_
depth'], var_name=['issues'],
                           value_vars=['empty_href',
                                       'leading_or_trailing_whitespace_
                                       in_href',
                                       'local_file','localhost',
                                       'malformed_href', 'no_text',
                                       'non_descriptive_text',
                                       'non-http_protocol_in_href',
                                       'url_in_onclick',
                                       'username_and_password_in_href',
                                       'no_image_anchors_with_no_
                                       alt_text'],
                           value_name='instances'
                          )

print(anchor_issues_levels)
```

This results in the following:

```
   crawl_depth                                  issues  instances
111     Not Set                  non_descriptive_text       2458
31      Not Set  leading_or_trailing_whitespace_in_href       2295
104           3                  non_descriptive_text        350
24            3  leading_or_trailing_whitespace_in_href        328
105           4                  non_descriptive_text        307
..          ...                                   ...        ...
85           13                               no_text          0
```

```
84          12                                  no_text          0
83          11                                  no_text          0
82          10                                  no_text          0
0            0                                 empty_href        0

[176 rows x 3 columns]
```

Most of the issues are on orphaned pages followed by URLs three to four levels deep. To visualize

```
anchor_levels_issues_count_plt = (ggplot(anchor_issues_levels, aes
(x = 'crawl_depth',
                                                                y = 'issues',
                                                                fill =
                                                                'instances'
                                                                )) +
                    geom_tile() +
                    labs(y = '# instances of Anchor Text Issues', x = '') +
                    scale_fill_cmap(cmap_name='viridis') +
                    theme_classic()
                   )

anchor_levels_issues_count_plt.save(filename = 'images/4_anchor_levels_
issues_count_plt.png',
                        height=5, width=5, units = 'in', dpi=1000)
anchor_levels_issues_count_plt
```

The anchor_levels_issues_count_plt graphic (Figure 3-15) makes it clearer; the technical issues with anchor text lay with the orphaned pages.

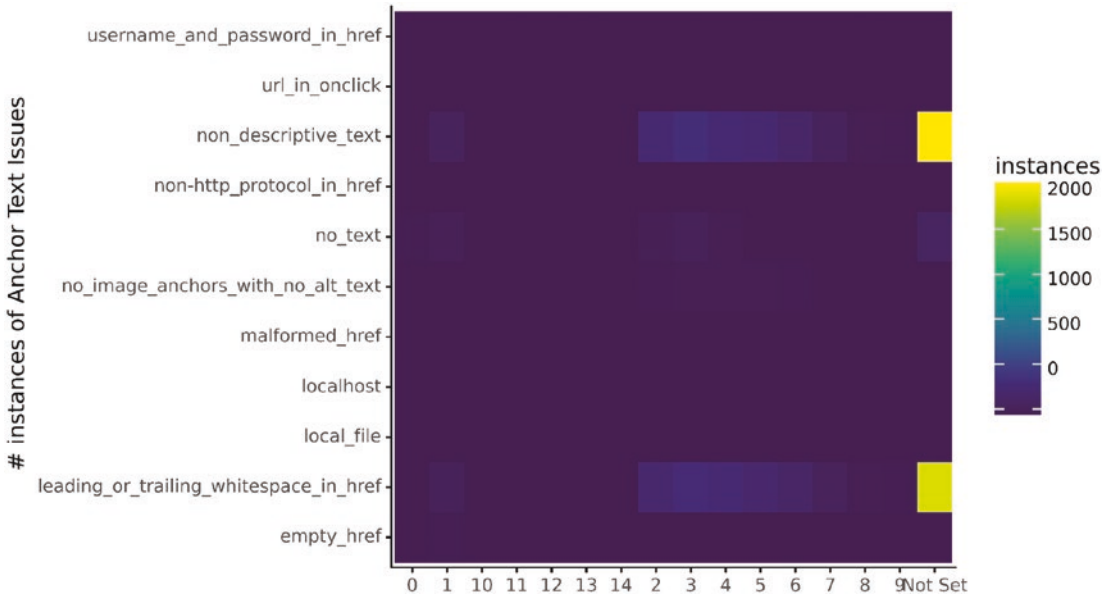

Figure 3-15. *Heatmap of site level, anchor text issues, and instances*

Anchor Text Relevance

Of course, that's not the only aspect of anchor text that SEOs are interested in. SEOs want to know the extent of the relevance between the anchor text and the destination URL.

For that task, we'll use string matching techniques on the Sitebulb link report to measure that relevance and then aggregate to see the overall picture:

```
link_df = link_data[['target_url', 'referring_url', 'anchor_text',
'location']]
link_df = link_df.rename(columns = {'target_url':'url'})
```

Merge with the crawl data using the URL as the primary key and then filter for indexable URLs only:

```
anchor_merge = crawl_data.merge(link_df, on = 'url', how = 'left')
anchor_merge = anchor_merge.loc[anchor_merge['host'] == website]

anchor_merge = anchor_merge.loc[anchor_merge['indexable'] == 'Yes']

anchor_merge['crawl_depth'] = anchor_merge['crawl_depth'].
astype('category')
```

```
anchor_merge['crawl_depth'] = anchor_merge['crawl_depth'].cat
.reorder_categories(['0', '1', '2', '3', '4', '5', '6', '7', '8', '9',
'10', 'Not Set'])
```

Then we compare the string similarity of the anchor text and title tag of the destination URLs:

```
anchor_merge['anchor_relevance'] = anchor_merge.loc[:, ['title',
                                                        'anchor_text']].
apply(lambda x: sorensen_dice(*x), axis=1)
```

And any URLs with less than 70% relevance score will be marked as irrelevant under the new column "irrel_anchors" as a 1.

Why 70%? This is from experience, and you're more than welcome to try different thresholds.

With Sorensen-Dice, which is not only fast but meets SEO needs for measuring relevance, 70% seems to be the right limit between relevance and irrelevance, especially when accounting for the site markers in the title tag string:

```
anchor_merge['irrel_anchors'] = np.where(anchor_merge['anchor_relevance'] <
.7, 1, 0)
```

```
Having a single factor makes it easier to aggregate the entire dataframe by
column although there are alternative methods to this:
```

```
anchor_merge['project'] = target_name
```

```
anchor_merge
```

This results in the following:

rect_url	redirect_url_status	redirect_url_status_code	referring_url	anchor_text	anchor_relevance	irrel_anchors	project
No Data	Not Set	Not Set	https://www.on24.com/about-us/	ON24	0.153846	1	ON24
No Data	Not Set	Not Set	https://www.on24.com/about-us/board-of-directors/	ON24	0.153846	1	ON24
No Data	Not Set	Not Set	https://www.on24.com/about-us/careers/	ON24	0.153846	1	ON24
No Data	Not Set	Not Set	https://www.on24.com/about-us/careers/	https://www.on24.com	0.289855	1	ON24
No Data	Not Set	Not Set	https://www.on24.com/about-us/careers/	https://www.on24.com	0.289855	1	ON24
...	...	...	...	...	...	...	...
No Data	Not Set	Not Set	https://www.on24.com/blog/business_types/enter...	Shell expands global brand awareness and drive...	0.965517	0	ON24
No Data	Not Set	Not Set	NaN	nan	0.060606	1	ON24
No Data	Not Set	Not Set	NaN	nan	0.125000	1	ON24
No Data	Not Set	Not Set	https://www.on24.com/blog/solutions/manufactur...	Agilent Optimizes its Global Digital Marketing...	0.956522	0	ON24
No Data	Not Set	Not Set	NaN	nan	0.148148	1	ON24

Because there is a many-to-many relationship between referring pages and destination URLs (i.e., a destination URL can receive links from multiple URLs, and the former can link to multiple URLs), the dataframe has expanded to over 350,000 rows from 8611.

Let's aggregate by counting the number of URLs per referring URL:

```
anchor_rel_stats_site_agg = anchor_merge.groupby('project').agg({'irrel_
anchors': 'sum'}).reset_index()
anchor_rel_stats_site_agg['total_urls'] = anchor_merge.shape[0]
anchor_rel_stats_site_agg['irrel_anchors_prop'] = anchor_rel_stats_site_
agg['irrel_anchors'] /anchor_rel_stats_site_agg['total_urls']
print(anchor_rel_stats_site_agg)
```

```
 project  irrel_anchors  total_urls  irrel_anchors_prop
0    ON24         333946      350643            0.952382
```

About 95% of anchor texts on this site are irrelevant. How does this compare to their competitors? That's your homework.

Let's go slightly deeper and analyze this by site depth:

```
anchor_rel_depth_irrels = anchor_merge.groupby(['crawl_depth']).
agg({'irrel_anchors': 'sum'}).reset_index()
anchor_rel_depth_urls = anchor_merge.groupby(['crawl_depth']).
agg({'project': 'count'}).reset_index()
anchor_rel_depth_stats = anchor_rel_depth_irrels.merge(anchor_rel_depth_
urls, on = 'crawl_depth', how = 'left')
```

```
anchor_rel_depth_stats['irrel_anchors_prop'] = anchor_rel_depth_
stats['irrel_anchors'] / anchor_rel_depth_stats['project']

anchor_rel_depth_stats
```

This results in the following:

	crawl_depth	irrel_anchors	project	irrel_anchors_prop
0	0	4139	4139	1.000000
1	1	306156	317345	0.964742
2	2	19162	23974	0.799283
3	3	655	1117	0.586392
4	4	608	713	0.852735
5	5	467	510	0.915686
6	6	278	289	0.961938
7	7	128	131	0.977099
8	8	33	33	1.000000
9	9	10	10	1.000000
10	10	2	2	1.000000
11	Not Set	2308	2380	0.969748

Virtually, all content at all site levels with the exception of those three clicks away from the home page (probably blog posts) have irrelevant anchors.

Let's visualize:

```
# anchor issues text
anchor_rel_stats_site_agg_plt = (ggplot(anchor_rel_depth_stats,
                                        aes(x = 'crawl_depth', y = 'irrel_
                                        anchors_prop')) +
                    geom_bar(stat = 'identity', fill = 'blue', alpha
                    = 0.6) +
                    labs(y = '# irrel_anchors', x = '') +
                    #scale_y_log10() +
                    theme_classic() +
                    coord_flip() +
```

```
                    theme(legend_position = 'none')
                  )

anchor_rel_stats_site_agg_plt.save(filename = 'images/3_anchor_rel_stats_
site_agg_plt.png',
                          height=5, width=5, units = 'in', dpi=1000)
anchor_rel_stats_site_agg_plt
```

Irrelevant anchors by site level are shown in the anchor_rel_stats_site_agg_plt plot (Figure 3-16), where we can see it is pretty much sitewide with less instances on URLs in site level 3.

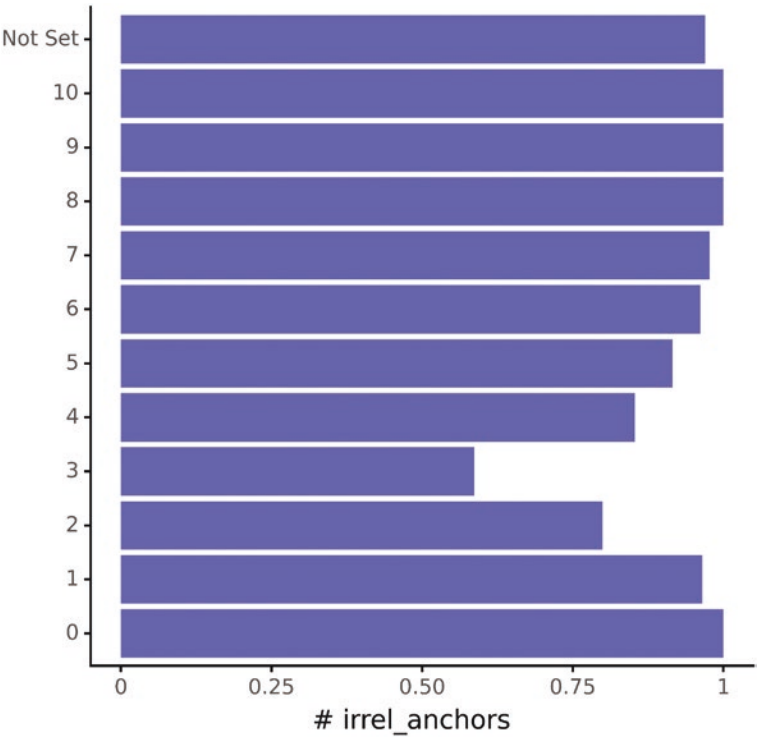

Figure 3-16. *Bar chart of irrelevant anchor texts by site level*

Location

More insight could be gained by looking at the location of the anchors:

```
anchor_rel_locat_irrels = anchor_merge.groupby(['location']).agg({'irrel_
anchors': 'sum'}).reset_index()
```

```
anchor_rel_locat_urls = anchor_merge.groupby(['location']).agg({'project':
'count'}).reset_index()
anchor_rel_locat_stats = anchor_rel_locat_irrels.merge(anchor_rel_locat_
urls, on = 'location', how = 'left')
anchor_rel_locat_stats['irrel_anchors_prop'] = anchor_rel_locat_
stats['irrel_anchors'] / anchor_rel_locat_stats['project']

anchor_rel_locat_stats
```

This results in the following:

	location	irrel_anchors	project	irrel_anchors_prop
0	Footer	137470	148609	0.925045
1	Header	194183	199741	0.972174

The irrelevant anchors are within the header or footer which make these relatively easy to solve.

Anchor Text Words

Let's look at the anchor texts themselves. Anchor texts are the words that make up the HTML hyperlinks. Search engines use these words to assign some meaning to the page that is being linked to.

Naturally, search engines will score anchor texts that accurately describe the content of the page they're linking to, because if a user does click the link, then they will receive a good experience of the content such that it matches their expectations created by the anchor text.

We'll start by looking at the most common words anchor texts used in the website:

```
anchor_count = anchor_merge[['anchor_text']].copy()
anchor_count['count'] = 1

anchor_count_agg = anchor_count.groupby('anchor_text').agg({'count':
'sum'}).reset_index()
anchor_count_agg = anchor_count_agg.sort_values('count', ascending = False)

anchor_count_agg
```

This results in the following:

	anchor_text	count
203	Contact Us	7427
551	Live Demo Discover how to create engaging webi...	7426
876	Resources	7426
550	Live Demo	7426
724	ON24 Webcast Elite	3851
...	...	...
916	Selling to Tech Means Investing in Customer Co...	1
915	Seismic shifted its APAC marketing strategy	1
176	Call-to-Action	1
181	Cassandra Clark Senior Manager of Webinar Prog...	1
1807	“The Ultimate Guide to Planning the Perfect We...	1

1808 rows × 2 columns

There are over 1,808 variations of anchor texts of which “Contact Us” is the most popular along with “Live Demo” and “Resources.”

Let’s visualize using a word cloud. We’ll have to import the WordCloud package and convert the dataframe into a dictionary:

```
from wordcloud import WordCloud

data = anchor_count_agg.set_index('anchor_text').to_dict()['count']
data
```

```
{'Contact Us ': 7427,
 'Live Demo Discover how to create engaging webinar experiences designed to
 cativate and convert your audience. ': 7426,
 'Resources ': 7426,
 'Live Demo ': 7426,
 'ON24 Webcast Elite ': 3851,
 'ON24 platform ': 3806,
```

```
 'Press Releases ': 3799, …}
```

Once converted, we feed this into the wordcloud function, limiting the data to the 200 most popular anchors:

```
wc = WordCloud(background_color='white',
               width=800, height=400,
               max_words=30).generate_from_frequencies(anchor_count_agg)

import matplotlib.pyplot as plt

plt.figure(figsize=(10, 10))
plt.imshow(wc, interpolation='bilinear')
plt.axis('off')

# Save image
wc.to_file("images/wordcloud.png")

plt.show()
```

The word cloud (Figure 3-17) could be used in a management presentation. There are some pretty long anchors there!

Figure 3-17. *Word cloud of the most commonly used anchor texts*

The activation from this point would be to see about finding semiautomated rules to improve the relevance of anchor texts, which is made easier by virtue of the fact that these are within the header or footer.

Core Web Vitals (CWV)

Core Web Vitals (CWV) is a Google initiative to help websites deliver a better UX. This includes speed, page stability during load, and the time it takes for the web page to become user interactive. So if CWV is about users, why is this in the technical section?

The technical SEO benefits which are less advertised help Google (and other search engines) mainly conserve computing resources to crawl and render websites. So it's a massive win-win-win for search engines, users, and webmasters.

So by pursuing CWV, you're effectively increasing your crawl and render budget which benefits your technical SEO.

However, technical SEO doesn't hold great appeal to marketing teams, whereas it's a much easier sell to marketing teams if you can imply the ranking benefits to justify web developments of improving CWV. And that is what we'll aim to do in this section.

We'll start with the landscape to show the overall competitive picture before drilling down on the website itself for the purpose of using data to prioritize development.

Landscape

```
import re
import time
import random
import pandas as pd
import numpy as np
import requests
import json
import plotnine
import tldextract
from plotnine import *
from mizani.transforms import trans
from client import RestClient

target_bu = 'boundless'
```

```
target_site = 'https://boundlesshq.com/'
target_name = target_bu
```

We start by obtaining the SERPs for your target keywords using the pandas read_csv function. We're interested in the URL which will form the input for querying the Google PageSpeed API which gives us the CWV metric values:

```
desktop_serps_df = pd.read_csv('data/1_desktop' + client_name +
'_serps.csv')
desktop_serps_df
```

This results in the following:

	keyword	rank_absolute	url	device
0	permanent establishment risk	1	https://papayaglobal.com/blog/how-to-avoid-per...	desktop
1	permanent establishment risk	2	None	desktop
2	permanent establishment risk	3	https://www.pwc.co.uk/assets/pdf/permanent-est...	desktop
3	permanent establishment risk	4	https://www.safeguardglobal.com/resources/blog...	desktop
4	permanent establishment risk	5	https://www.omnipresent.com/resources/permanen...	desktop
5	permanent establishment risk	6	https://www.airswift.com/blog/permanent-establ...	desktop
6	permanent establishment risk	7	https://www.gtn.com/resources/newsletters/perm...	desktop
7	permanent establishment risk	8	https://www.letsdeel.com/blog/permanent-establ...	desktop
8	permanent establishment risk	9	https://freemanlaw.com/the-tax-risk-of-a-perma...	desktop
9	permanent establishment risk	10	https://www.oysterhr.com/library/avoid-permane...	desktop
10	permanent establishment risk	11	https://nhglobalpartners.com/what-is-permanent...	desktop

The SERPs data can get a bit noisy, and ultimately the business is only interested in their direct competitors, so we'll create a list of them to filter the SERPs accordingly:

```
selected_sites = [target_site, 'https://papayaglobal.com/', 'https://www.
airswift.com/', 'https://shieldgeo.com/',
                  'https://remote.com/', 'https://www.letsdeel.com/',
'https://www.omnipresent.com/']

desktop_serps_select = desktop_serps_df[~desktop_serps_df['url'].
isnull()].copy()
```

```
desktop_serps_select = desktop_serps_select[desktop_serps_select['url'].
str.contains('|'.join(selected_sites))]
desktop_serps_select
```

	keyword	rank_absolute	url	device
0	permanent establishment risk	1	https://papayaglobal.com/blog/how-to-avoid-permanent-establishment-risk/	desktop
4	permanent establishment risk	5	https://www.omnipresent.com/resources/permanent-establishment-risk-a-remote-workforce	desktop
5	permanent establishment risk	6	https://www.airswift.com/blog/permanent-establishment-risks	desktop
7	permanent establishment risk	8	https://www.letsdeel.com/blog/permanent-establishment-risk	desktop
14	permanent establishment risk	15	https://shieldgeo.com/ultimate-guide-permanent-establishment/	desktop
...	...	...	...	...
2355	eor country	32	https://papayaglobal.com/blog/differences-between-icps-why-it-matters/	desktop
2370	eor country	47	https://www.omnipresent.com/resources/peo-vs-eor-tapping-into-global-talent	desktop
2374	eor country	51	https://boundlesshq.com/guides/croatia/	desktop
2376	eor country	53	https://www.letsdeel.com/blog/international-employee-experience	desktop
2403	eor country	80	https://www.airswift.com/blog/what-is-an-employer-of-record	desktop

114 rows × 4 columns

There are much less rows as a result, which means less API queries and less time required to get the data.

Note the data is just for desktop, so this process would need to be repeated for mobile SERPs also.

To query the PageSpeed API efficiently and avoid duplicate requests, we want a unique set of URLs. We achieve this by

Exporting the URL column to a list

```
desktop_serps_urls = desktop_serps_select['url'].to_list()
```

Deduplicating the list

```
desktop_serps_urls = list(dict.fromkeys(desktop_serps_urls))
desktop_serps_urls
```

```
['https://papayaglobal.com/blog/how-to-avoid-permanent-
establishment-risk/',
 'https://www.omnipresent.com/resources/permanent-establishment-risk-a-
 remote-workforce',
 'https://www.airswift.com/blog/permanent-establishment-risks',
 'https://www.letsdeel.com/blog/permanent-establishment-risk',
```

```
 'https://shieldgeo.com/ultimate-guide-permanent-establishment/',
 'https://remote.com/blog/what-is-permanent-establishment',
 'https://remote.com/lp/global-payroll',
 'https://remote.com/services/global-payroll?nextInternalLocale=
en-us', . . . ]
```

With the list, we query the API, starting by setting the parameters for the API itself, the device, and the API key (obtained by getting a Google Cloud Platform account which is free):

```
base_url = 'https://www.googleapis.com/pagespeedonline/v5/
runPagespeed?url='
strategy = '&strategy=desktop'
api_key = '&key=[Your PageSpeed API key]'
```

Initialize an empty dictionary and set i to zero which will be used as a counter to help us keep track of how many API calls have been made and how many to go:

```
desktop_cwv = {}
i = 1

for url in desktop_serps_urls:
    request_url = base_url + url + strategy + api_key
    response = json.loads(requests.get(request_url).text)
    i += 1
    print(i, " ", request_url)
    desktop_cwv[url] = response
```

The result is a dictionary containing the API response. To get this output into a usable format, we iterate through the dictionary to pull out the actual CWV scores as the API has a lot of other micro measurement data which doesn't serve our immediate objectives.

Initialize an empty list which will store the API response data:

```
desktop_psi_lst = []
```

Loop through the API output which is a JSON dictionary, so we need to pull out the relevant "keys" and add them to the list initialized earlier:

```
for key, data in desktop_cwv.items():
```

```
    if 'lighthouseResult' in data:
        FCP = data['lighthouseResult']['audits']['first-contentful-paint']
        ['numericValue']
        LCP = data['lighthouseResult']['audits']['largest-contentful-
        paint']['numericValue']
        CLS = data['lighthouseResult']['audits']['cumulative-layout-shift']
        ['numericValue']
        FID = data['lighthouseResult']['audits']['max-potential-fid']
        ['numericValue']
        SIS = data['lighthouseResult']['audits']['speed-index']
        ['score'] * 100

        desktop_psi_lst.append([key, FCP, LCP, CLS, FID, SIS])
```

Convert the list into a dataframe:

```
desktop_psi_df = pd.DataFrame(desktop_psi_lst, columns = ['url', 'FCP',
'LCP', 'CLS', 'FID', 'SIS'])
desktop_psi_df
```

This results in the following:

	url	FCP	LCP	CLS	FID	SIS
0	https://papayaglobal.com/blog/how-to-avoid-permanent-establishment-risk/	890.00000	5241.5	0.140066	103	17.0
1	https://www.omnipresent.com/resources/permanent-establishment-risk-a-remote-workforce	1010.00000	1720.0	0.144360	138	92.0
2	https://www.airswift.com/blog/permanent-establishment-risks	662.13298	2734.0	0.118718	99	28.0
3	https://www.letsdeel.com/blog/permanent-establishment-risk	1093.00000	1736.5	0.000000	81	67.0
4	https://shieldgeo.com/ultimate-guide-permanent-establishment/	1030.00000	1254.5	0.121591	81	96.0
...	...	...	...	...	...	...
72	https://www.letsdeel.com/blog/remote-interview-guide	942.00000	1320.0	0.000000	122	68.0
73	https://www.omnipresent.com/resources/how-to-recruit-and-hire-remote-employees	1070.00000	1640.0	0.138055	94	72.0
74	https://papayaglobal.com/blog/differences-between-icps-why-it-matters/	930.00000	3432.0	0.089238	129	33.0
75	https://boundlesshq.com/guides/croatia/	730.00000	2050.0	0.051751	78	99.0
76	https://www.letsdeel.com/blog/international-employee-experience	972.00000	1411.0	0.000000	129	67.0

77 rows × 6 columns

The PageSpeed data on all of the ranking URLs is in a dataframe with all of the CWV metrics:

- *FCP*: First Contentful Paint

- *LCP*: Largest Contentful Paint
- *CLS*: Cumulative Layout Shift
- *SIS*: Speed Index Score

To show the relevance of the ranking (and hopefully the benefit to ranking by improving CWV), we want to merge this with the rank data:

```
dtp_psi_serps = desktop_serps_select.merge(desktop_psi_df, on = 'url', how
= 'left')
dtp_psi_serps_bu = dtp_psi_serps.merge(target_keywords_df, on = 'keyword',
how = 'left')
dtp_psi_serps_bu.to_csv('data/'+ target_bu +'_dtp_psi_serps_bu.csv')
dtp_psi_serps_bu
```

This results in the following:

	keyword	rank_absolute	url	device	FCP	LCP	CLS	FID	SIS	bu
0	permanent establishment risk	1	https://papayaglobal.com/blog/how-to-avoid-permanent-establishment-risk/	desktop	890.00000	5241.5	0.140066	103.0	17.0	boundless
1	permanent establishment risk	5	https://www.omnipresent.com/resources/permanent-establishment-risk-a-remote-workforce	desktop	1010.00000	1720.0	0.144360	138.0	92.0	boundless
2	permanent establishment risk	6	https://www.airswift.com/blog/permanent-establishment-risks	desktop	662.13298	2734.0	0.118718	99.0	28.0	boundless
3	permanent establishment risk	8	https://www.letsdeel.com/blog/permanent-establishment-risk	desktop	1093.00000	1736.5	0.000000	81.0	67.0	boundless
4	permanent establishment risk	15	https://shieldgeo.com/ultimate-guide-permanent-establishment/	desktop	1030.00000	1254.5	0.121591	81.0	96.0	boundless
...	...	...	...	...	...	...	...	...	...	...
112	eor country	32	https://papayaglobal.com/blog/differences-between-icps-why-it-matters/	desktop	930.00000	3432.0	0.089238	129.0	33.0	boundless
113	eor country	47	https://www.omnipresent.com/resources/peo-vs-eor-tapping-into-global-talent	desktop	1070.00000	1740.0	0.138055	128.0	94.0	boundless
114	eor country	51	https://boundlesshq.com/guides/croatia/	desktop	730.00000	2050.0	0.051751	78.0	99.0	boundless
115	eor country	53	https://www.letsdeel.com/blog/international-employee-experience	desktop	972.00000	1411.0	0.000000	129.0	67.0	boundless
116	eor country	80	https://www.airswift.com/blog/what-is-an-employer-of-record	desktop	662.13298	2986.0	0.120042	94.0	64.0	boundless

117 rows × 10 columns

The dataframe is complete with the keyword, its rank, URL, device, and CWV metrics.

At this point, rather than repeat near identical code for mobile, you can assume we have the data for mobile which we have combined into a single dataframe using the pandas concat function (same headings).

To add some additional features, we have added another column is_target indicating whether the ranking URL is the client or not:

```
overall_psi_serps_bu['is_target'] = np.where(overall_psi_serps_bu['url'].
str.contains(target_site), '1', '0')
```

Parse the site name:

```
overall_psi_serps_bu['site'] = overall_psi_serps_bu['url'].apply(lambda
url: tldextract.extract(url).domain)
```

Count the column for easy aggregation:

```
overall_psi_serps_bu['count'] = 1
```

The resultant dataframe is overall_psi_serps_bu shown as follows:

keyword	rank_absolute	url	device	FCP	LCP	CLS	FID	SIS	bu	is_target	site
permanent stablishment risk	1	https://papayaglobal.com/blog/how-to-avoid-permanent-establishment-risk/	mobile	3990.00000	4140.0	0.051357	349.0	4.0	boundless	0	papayaglobal
permanent stablishment risk	4	https://www.omnipresent.com/resources/permanent-establishment-risk-a-remote-workforce	mobile	4065.00000	4515.0	0.040469	471.0	30.0	boundless	0	omnipresent
permanent stablishment risk	5	https://www.airswift.com/blog/permanent-establishment-risks?hs_amp=true	mobile	927.00000	2495.0	0.000000	316.0	97.0	boundless	0	airswift
permanent stablishment risk	8	https://www.letsdeel.com/blog/permanent-establishment-risk	mobile	5763.00000	7131.0	0.000000	286.0	25.0	boundless	0	letsdeel
permanent stablishment risk	16	https://shieldgeo.com/ultimate-guide-permanent-establishment/	mobile	4623.00000	6695.0	0.000343	299.0	70.0	boundless	0	shieldgeo
...	...	...	...	...	...	...	...	...	...	...	...
eor country	32	https://papayaglobal.com/blog/differences-between-icps-why-it-matters/	desktop	930.00000	3432.0	0.089238	129.0	33.0	boundless	0	papayaglobal
eor country	47	https://www.omnipresent.com/resources/peo-vs-eor-tapping-into-global-talent	desktop	1070.00000	1740.0	0.138055	128.0	94.0	boundless	0	omnipresent
eor country	51	https://boundlesshq.com/guides/croatia/	desktop	730.00000	2050.0	0.051751	78.0	99.0	boundless	1	boundlesshq
eor country	53	https://www.letsdeel.com/blog/international-employee-experience	desktop	972.00000	1411.0	0.000000	129.0	67.0	boundless	0	letsdeel
eor country	80	https://www.airswift.com/blog/what-is-an-employer-of-record	desktop	662.13298	2986.0	0.120042	94.0	64.0	boundless	0	airswift

The aggregation will be executed at the site level so we can compare how each site scores on average for their CWV metrics and correlate that with performance:

```
overall_psi_serps_agg = overall_psi_serps_bu.groupby('site').
agg({'LCP': 'mean',
                                                    'FCP': 'mean',
                                                    'CLS': 'mean',
                                                    'FID': 'mean',
                                                    'SIS': 'mean',
```

```
                                                        'rank_
                                                        absolute':
                                                        'mean',
                                                        'count':
                                                        'sum'}).
                                                        reset_
                                                        index()
overall_psi_serps_agg = overall_psi_serps_agg.rename(columns = {'count':
'reach'})
```

Here are some operations to make the site names shorter for the graphs later:

```
overall_psi_serps_agg['site'] = np.where(overall_psi_serps_agg['site'] ==
'papayaglobal', 'papaya',
                                         overall_psi_serps_agg['site'])
overall_psi_serps_agg['site'] = np.where(overall_psi_serps_agg['site'] ==
'boundlesshq', 'boundless',
                                         overall_psi_serps_agg['site'])
overall_psi_serps_agg
```

This results in the following:

	site	LCP	FCP	CLS	FID	SIS	rank_absolute	reach
0	airswift	2850.282609	952.014339	0.060763	150.347826	72.521739	30.043478	23
1	boundless	4563.175000	2043.200000	0.186265	158.900000	92.450000	35.900000	20
2	letsdeel	5153.855556	3340.166667	0.000000	229.955556	38.755556	36.044444	45
3	omnipresent	3809.153846	2380.897436	0.064785	242.000000	72.820513	34.948718	39
4	papaya	10835.166753	2640.666667	0.057782	312.800000	8.833333	42.562500	32
5	remote	5358.801887	2169.452830	0.074436	845.716981	89.660377	17.641509	53
6	shieldgeo	5756.100000	3017.533333	0.056910	234.833333	71.500000	33.333333	30

That's the summary which is not so easy to discern trends, and now we're ready to plot the data, starting with the overall speed index. The Speed Index Score (SIS) is scaled between 0 and 100, 100 being the fastest and therefore best.

Note that in all of the charts that will compare Google rank with the individual CWV metrics, the vertical axis will be inverted such that the higher the position, the higher the ranking. This is to make the charts more intuitive and easier to understand.

```
SIS_cwv_landscape_plt = (
    ggplot(overall_psi_serps_agg,
           aes(x = 'SIS', y = 'rank_absolute', fill = 'site', colour = 'site',
                                size = 'reach')) +
    geom_point(alpha = 0.8) +
    geom_text(overall_psi_serps_agg, aes(label = 'site'),
    position=position_stack(vjust=-0.08)) +
    labs(y = 'Google Rank', x = 'Speed Score') +
    scale_y_reverse() +
  scale_size_continuous(range = [7, 17]) +
    theme(legend_position = 'none', axis_text_x=element_text(rotation=0,
    hjust=1, size = 12))
)

SIS_cwv_landscape_plt.save(filename = 'images/0_SIS_cwv_landscape.png',
                           height=5, width=8, units = 'in', dpi=1000)
SIS_cwv_landscape_plt
```

Already we can see in SIS_cwv_landscape_plt (Figure 3-18) that the higher your speed score, the higher you rank in general which is a nice easy sell to the stakeholders, acting as motivation to invest resources into improving CWV.

Figure 3-18. *Scatterplot comparing speed scores and Google rank of different websites*

Boundless in this instance are doing relatively well. Although they don't rank the highest, this could indicate that either some aspects of CWV are not being attended to or something non-CWV related or more likely a combination of both.

```
LCP_cwv_landscape_plt = (
    ggplot(overall_psi_serps_agg,
           aes(x = 'LCP', y = 'rank_absolute', fill = 'site', colour
           = 'site',
                              size = 'reach')) +
    geom_point(alpha = 0.8) +
    geom_text(overall_psi_serps_agg, aes(label = 'site'),
    position=position_stack(vjust=-0.08)) +
    labs(y = 'Google Rank', x = 'Largest Contentful Paint') +
    scale_y_reverse() +
  scale_size_continuous(range = [7, 17]) +
    theme(legend_position = 'none', axis_text_x=element_text(rotation=0,
    hjust=1, size = 12))
)

LCP_cwv_landscape_plt.save(filename = 'images/0_LCP_cwv_landscape.png',
                           height=5, width=8, units = 'in', dpi=1000)
LCP_cwv_landscape_plt
```

The LCP_cwv_landscape_plt plot (Figure 3-19) shows that Papaya and Remote look like outliers; in any case, the trend does indicate that the less time it takes to load the largest content element, the higher the rank.

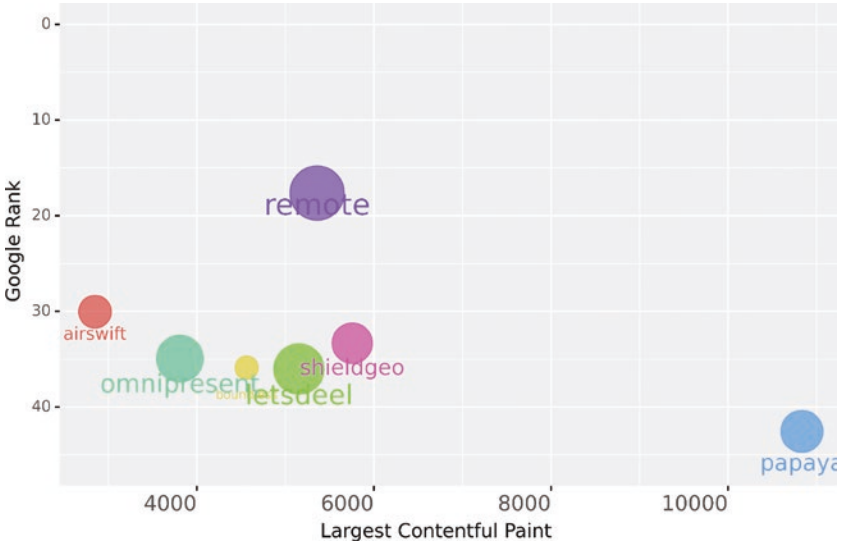

Figure 3-19. *Scatterplot comparing Largest Contentful Paint (LCP) and Google rank by website*

```
FID_cwv_landscape_plt = (
    ggplot(overall_psi_serps_agg,
           aes(x = 'FID', y = 'rank_absolute', fill = 'site', colour
           = 'site',
                                size = 'reach')) +
    geom_point(alpha = 0.8) +
    geom_text(overall_psi_serps_agg, aes(label = 'site'),
    position=position_stack(vjust=-0.08)) +
    labs(y = 'Google Rank', x = 'First Input Delay') +
    scale_y_reverse() +
    scale_x_log10() +
  scale_size_continuous(range = [7, 17]) +
    theme(legend_position = 'none', axis_text_x=element_text(rotation=0,
    hjust=1, size = 12))
)

FID_cwv_landscape_plt.save(filename = 'images/0_FID_cwv_landscape.png',
                           height=5, width=8, units = 'in', dpi=1000)
FID_cwv_landscape_plt
```

Remote looks like an outlier in FID_cwv_landscape_plt (Figure 3-20). Should the outlier be removed? Not in this case, because we don't remove outliers just because it doesn't show us what we wanted it to show.

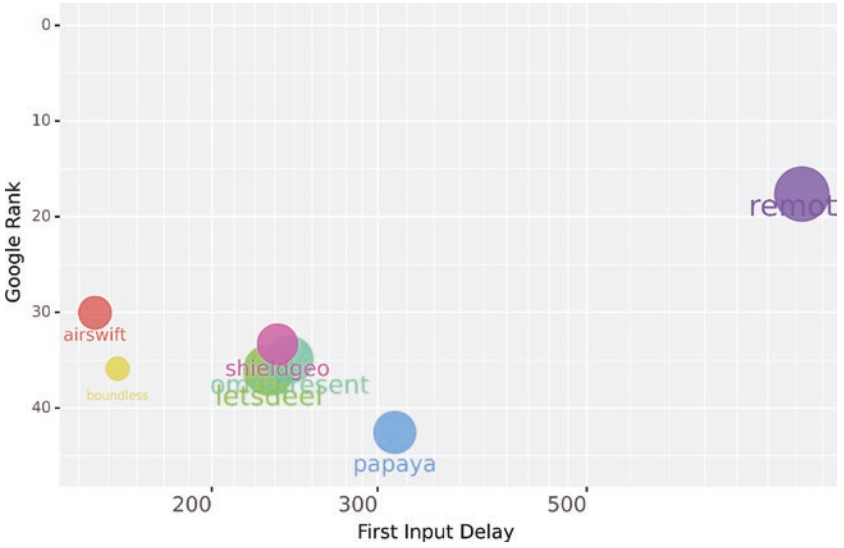

Figure 3-20. *Scatterplot comparing First Input Delay (FID) and Google rank by website*

The trend indicates that the less time it takes to make the page interactive for users, the higher the rank.

Boundless are doing well in this respect.

```
CLS_cwv_landscape_plt = (
    ggplot(overall_psi_serps_agg,
           aes(x = 'CLS', y = 'rank_absolute', fill = 'site', colour
           = 'site',
                                size = 'reach')) +
    geom_point(alpha = 0.8) +
    geom_text(overall_psi_serps_agg, aes(label = 'site'),
    position=position_stack(vjust=-0.08)) +
    labs(y = 'Google Rank', x = 'Cumulative Layout Shift') +
    scale_y_reverse() +
  scale_size_continuous(range = [7, 17]) +
```

```
    theme(legend_position = 'none', axis_text_x=element_text(rotation=0,
    hjust=1, size = 12))
)

CLS_cwv_landscape_plt.save(filename = 'images/0_CLS_cwv_landscape.png',
                           height=5, width=8, units = 'in', dpi=1000)
CLS_cwv_landscape_plt
```

Okay, CLS where Boundless don't perform as well is shown in CLS_cwv_landscape_plt (Figure 3-21). The impact on improving rank is quite unclear too.

Figure 3-21. *Scatterplot comparing Cumulative Layout Shift (CLS) and Google rank by website*

```
FCP_cwv_landscape_plt = (
    ggplot(overall_psi_serps_agg,
           aes(x = 'FCP', y = 'rank_absolute', fill = 'site', colour
           = 'site',
                                size = 'reach')) +
    geom_point(alpha = 0.8) +
    geom_text(overall_psi_serps_agg, aes(label = 'site'),
    position=position_stack(vjust=-0.08)) +
```

```
    labs(y = 'Google Rank', x = 'First Contentful Paint') +
    scale_y_reverse() +
  scale_size_continuous(range = [7, 17]) +
    theme(legend_position = 'none', axis_text_x=element_text(rotation=0,
    hjust=1, size = 12))
)

FCP_cwv_landscape_plt.save(filename = 'images/0_FCP_cwv_landscape.png',
                           height=5, width=8, units = 'in', dpi=1000)
FCP_cwv_landscape_plt
```

Papaya and Remote look like outliers in FCP_cwv_landscape_plt (Figure 3-22); in any case, the trend does indicate that the less time it takes to load the largest content element, the higher the rank.

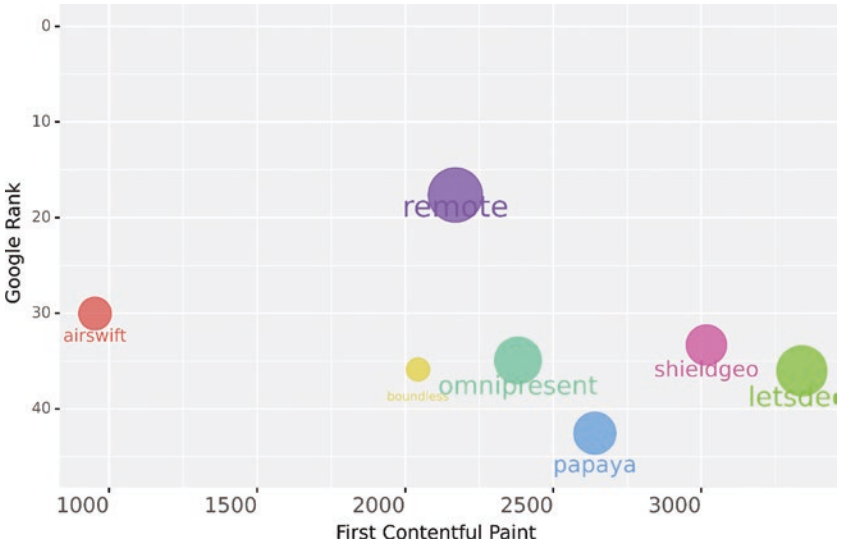

Figure 3-22. *Scatterplot comparing First Contentful Paint (FCP) and Google rank by website*

That's the deep dive into the overall scores. The preceding example can be repeated for both desktop and mobile scores to drill down into, showing which specific CWV metrics should be prioritized. Overall, for boundless, CLS appears to be its weakest point.

In the following, we'll summarize the analysis on a single chart by pivoting the data in a format that can be used to power the single chart:

```
overall_psi_serps_long = overall_psi_serps_agg.copy()
```

We select the columns we want:

```
overall_psi_serps_long = overall_psi_serps_long[['site', 'LCP', 'FCP',
'CLS', 'FID', 'SIS']]
```

and use the melt function to pivot the table:

```
overall_psi_serps_long = overall_psi_serps_long.melt(id_vars=['site'],
                                                     value_vars=['LCP',
'FCP', 'CLS', 'FID', 'SIS'],
                                                     var_name='Metric',
value_name='Index')
overall_psi_serps_long['x_axis'] = overall_psi_serps_long['Metric']
overall_psi_serps_long['site'] = np.where(overall_psi_serps_long['site'] ==
'papayaglobal', 'papaya',
                                          overall_psi_serps_long['site'])
overall_psi_serps_long['site'] = np.where(overall_psi_serps_long['site'] ==
'boundlesshq', 'boundless',
                                          overall_psi_serps_long['site'])

overall_psi_serps_long
```

This results in the following:

	site	Metric	Index	x_axis
0	airswift	LCP	2807.090909	LCP
1	airswift	LCP	2889.875000	LCP
2	boundless	LCP	1935.050000	LCP
3	boundless	LCP	7191.300000	LCP
4	letsdeel	LCP	1926.391304	LCP
...	...	...	...	...
65	papaya	SIS	2.333333	SIS
66	remote	SIS	96.227273	SIS
67	remote	SIS	85.000000	SIS
68	shieldgeo	SIS	88.785714	SIS
69	shieldgeo	SIS	56.375000	SIS

70 rows × 4 columns

That's the long format in place, ready to plot.

```
speed_ex_plt = (
    ggplot(overall_psi_serps_long,
           aes(x = 'site', y = 'Index', fill = 'site')) +
    geom_bar(stat = 'identity', alpha = 0.8) +
    labs(y = '', x = '') +
    theme(legend_position = 'right',
          axis_text_x =element_text(rotation=90, hjust=1, size = 12),
          legend_title = element_blank()
         ) +
    facet_grid('Metric ~ .', scales = 'free')
)

speed_ex_plt.save(filename = 'images/0_CWV_Metrics_plt.png',
                            height=5, width=8, units = 'in', dpi=1000)
speed_ex_plt
```

The speed_ex_plt chart (Figure 3-23) shows the competitors being compared for each metric. Remote seem to perform the worst on average, so their prominent rankings are probably due to non-CWV factors.

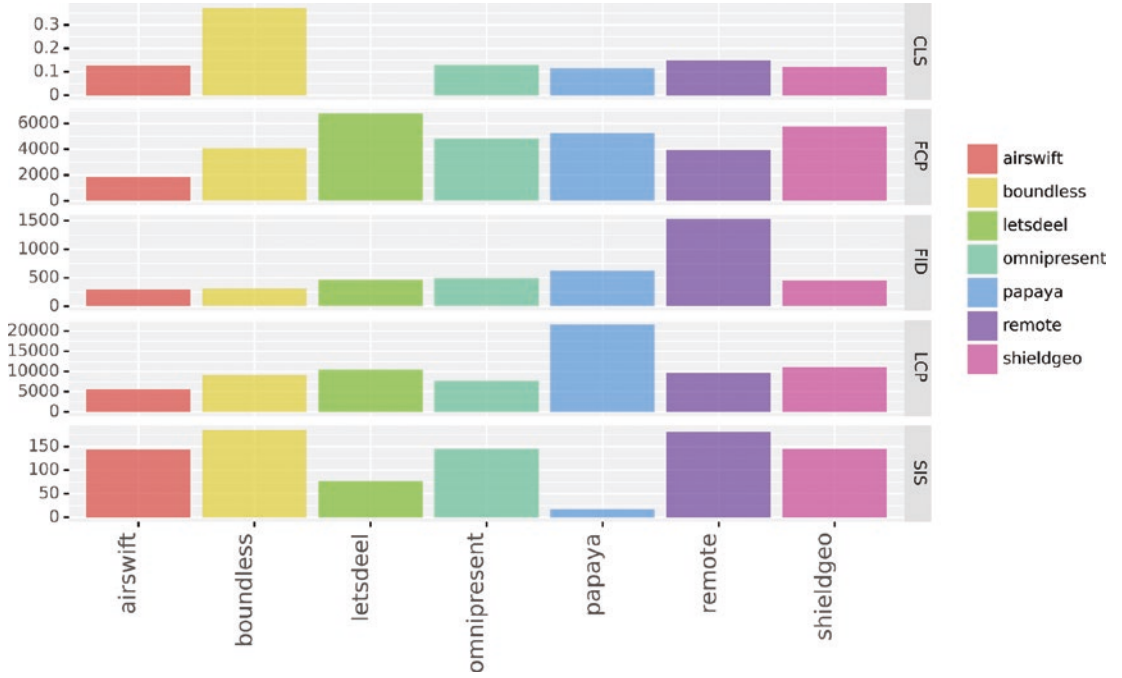

Figure 3-23. *Faceted column chart of different sites by CWV metric*

Onsite CWV

The purpose of the landscape was to use data to motivate the client, colleagues, and stakeholders of the SEO benefits that would follow CWV improvement. In this section, we're going to drill into the site itself to see where the improvements could be made.

We'll start by importing the data and cleaning up the columns as usual:

```
target_crawl_raw = pd.read_csv('data/boundlesshq_com_all_urls__excluding_
uncrawled__filtered_20220427203402.csv')

target_crawl_raw.columns = [col.lower() for col in target_crawl_raw.
columns]
target_crawl_raw.columns = [col.replace('(', '') for col in target_crawl_
raw.columns]
```

```
target_crawl_raw.columns = [col.replace(')', '') for col in target_crawl_
raw.columns]
target_crawl_raw.columns = [col.replace('@', '') for col in target_crawl_
raw.columns]
target_crawl_raw.columns = [col.replace('/', '') for col in target_crawl_
raw.columns]
target_crawl_raw.columns = [col.replace(' ', '_') for col in target_crawl_
raw.columns]
print(target_crawl_raw.columns)
```

We're using Sitebulb crawl data, and we want to only include onsite indexable URLs since those are the ones that rank, which we will filter as follows:

```
target_crawl_raw = target_crawl_raw.loc[target_crawl_raw['host'] ==
target_host]
target_crawl_raw = target_crawl_raw.loc[target_crawl_raw['indexable_
status'] == 'Indexable']
target_crawl_raw = target_crawl_raw.loc[target_crawl_raw['content_type']
== 'HTML']

target_crawl_raw
```

This results in the following:

	url	crawl_depth	crawl_status	host	is_subdomain	scheme	crawl_source	first_parent_url	h
0	https://boundlesshq.com/	0	Success	boundlesshq.com	No	https	Crawler	None	
13	https://boundlesshq.com/pricing/	1	Success	boundlesshq.com	No	https	Crawler	https://boundlesshq.com/	
17	https://boundlesshq.com/how-it-works/countries/	1	Success	boundlesshq.com	No	https	Crawler	https://boundlesshq.com/	
23	https://boundlesshq.com/borderless-benefits/	1	Success	boundlesshq.com	No	https	Crawler	https://boundlesshq.com/	
36	https://boundlesshq.com/blog/employment/what-is-an-employer-of-record/	1	Success	boundlesshq.com	No	https	Crawler	https://boundlesshq.com/	
...	...	...	...	...	...	...	...	...	
4404	https://boundlesshq.com/guides/united-arab-emirates/	Not Set	Success	boundlesshq.com	No	https	Google Search Analytics	None	
4407	https://boundlesshq.com/guides/vietnam/	Not Set	Success	boundlesshq.com	No	https	Google Search Analytics	None	
4477	https://boundlesshq.com/hr-tech-ireland/	Not Set	Success	boundlesshq.com	No	https	Google Search Analytics	None	
4486	https://boundlesshq.com/guides/uruguay/	Not Set	Success	boundlesshq.com	No	https	Google Search Analytics	None	
4498	https://boundlesshq.com/guides/slovenia/	Not Set	Success	boundlesshq.com	No	https	Google Search Analytics	None	

279 rows × 71 columns

With 279 rows, it's a small website. The next step is to select the desired columns which will comprise the CWV measures and anything that could possibly explain it:

```
target_speedDist_df = target_crawl_raw[['url', 'cumulative_layout_shift',
'first_contentful_paint',
                                        'largest_contentful_paint',
                                        'performance_score', 'time_to_
                                        interactive',
                                        'total_blocking_time', 'images_
                                        without_dimensions', 'perf_
                                        budget_fonts',
                                        'font_transfer_size_kib', 'fonts_
                                        files', 'images_files',
                                        'images_not_efficiently_encoded',
                                        'images_size_kib',
                                        'images_transfer_size_kib',
                                        'images_without_dimensions',
                                        'media_files',
                                        'media_size_kib', 'media_transfer_
                                        size_kib',
                                        'next-gen_format_savings_kib',
                                        'offscreen_images_not_deferred',
                                        'other_files', 'other_size_kib',
                                        'other_transfer_size_kib',
                                        'passed_font-face_display_urls',
                                        'render_blocking_savings',
                                        'resources_not_http2', 'scaled_
                                        images', 'perf_budget_total']]

target_speedDist_df
```

This results in the following:

	url	cumulative_layout_shift	first_contentful_paint	largest_contentful_paint	performance_score	time_to_interactive
0	https://boundlesshq.com/	0.339	1757	3518	83	2111
13	https://boundlesshq.com/pricing/	0.096	2459	2635	84	3273
17	https://boundlesshq.com/how-it-works/countries/	0.104	2395	2616	75	3825
23	https://boundlesshq.com/borderless-benefits/	0.139	1952	2578	90	2629
36	https://boundlesshq.com/blog/employment/what-is-an-employer-of-record/	0.236	2052	5380	59	3648
...	...	...	...	...	...	...
4404	https://boundlesshq.com/guides/united-arab-emirates/	0.103	2103	9684	64	2711
4407	https://boundlesshq.com/guides/vietnam/	0.104	1868	9427	71	2529
4477	https://boundlesshq.com/hr-tech-ireland/	0.214	2190	2616	83	3297
4486	https://boundlesshq.com/guides/uruguay/	0.104	2057	9460	66	2551
4498	https://boundlesshq.com/guides/slovenia/	0.104	2113	8651	64	2760

279 rows × 29 columns

The dataframe columns have reduced from 71 to 29, and the CWV scores are more apparent.

Attempting to analyze the sites at the URL will not be terribly useful, so to make pattern identification easier, we will classify the content by folder location:

```
section_conds = [
    target_speedDist_df['url'] == 'https://boundlesshq.com/',
    target_speedDist_df['url'].str.contains('/guides/'),
    target_speedDist_df['url'].str.contains('/how-it-works/')
]

section_vals = ['home', 'guides', 'commercial']

target_speedDist_df['content'] = np.select(section_conds, section_vals,
default = 'blog')
```

We'll also convert the main metrics to a number:

```
cols = ['cumulative_layout_shift', 'first_contentful_paint', 'largest_
contentful_paint', 'performance_score',
        'time_to_interactive', 'total_blocking_time']

target_speedDist_df[cols] = pd.to_numeric(target_speedDist_df[cols].
stack(), errors='coerce').unstack()

target_speedDist_df
```

This results in the following:

other_files	other_size_kib	other_transfer_size_kib	passed_font-face_display_urls	render_blocking_savings	resources_not_http2	scaled_images	perf_budget_total	content
0	0	0	0	10550	0	20	Yes	home
0	0	0	0	8348	0	0	Yes	blog
0	0	0	0	8805	0	5	Yes	commercial
0	0	0	0	8572	0	0	Yes	blog
0	0	0	0	8895	0	0	Yes	blog
...	...	...	...	...	...	...	...	...
0	0	0	0	8415	0	0	Yes	guides
0	0	0	0	8327	0	0	Yes	guides
0	0	0	0	8501	0	1	Yes	blog
0	0	0	0	8501	0	0	Yes	guides
0	0	0	0	8660	0	0	Yes	guides

A new column has been created in which each indexable URL is labeled by their content category.

Time for some aggregation using groupby on "content":

```
speed_dist_agg = target_speedDist_df.groupby('content').agg({'url':
'count', 'performance_score'}).reset_index()
speed_dist_agg
```

This results in the following:

	content	url	performance_score
0	blog	66	68.636364
1	commercial	3	66.333333
2	guides	209	76.631579
3	home	1	83.000000

Most of the content are guides followed by blog posts with three offer pages. To visualize, we're going to use a histogram showing the distribution of the overall performance score and color code the URLs in the score columns by their segment.

The home page and the guides are by far the fastest.

```
target_speedDist_plt = (
    ggplot(target_speedDist_df,
           aes(x = 'performance_score', fill = 'content')) +
    geom_histogram(alpha = 0.8, bins = 20) +
    labs(y = 'Page Count', x = '\nSpeed Score') +
```

```
    #scale_x_continuous(breaks=range(0, 100, 20)) +
    theme(legend_position = 'right',
          axis_text_x = element_text(rotation=90, hjust=1, size = 7))
)

target_speedDist_plt.save(filename = 'images/3_target_speedDist_plt.png',
                          height=5, width=8, units = 'in', dpi=1000)
target_speedDist_plt
```

The target_speedDist_plt plot (Figure 3-24) shows the home page (in purple) performs reasonably well with a speed score of 84. The guides vary, but most of these have a speed above 80, and the majority of blog posts are in the 70s.

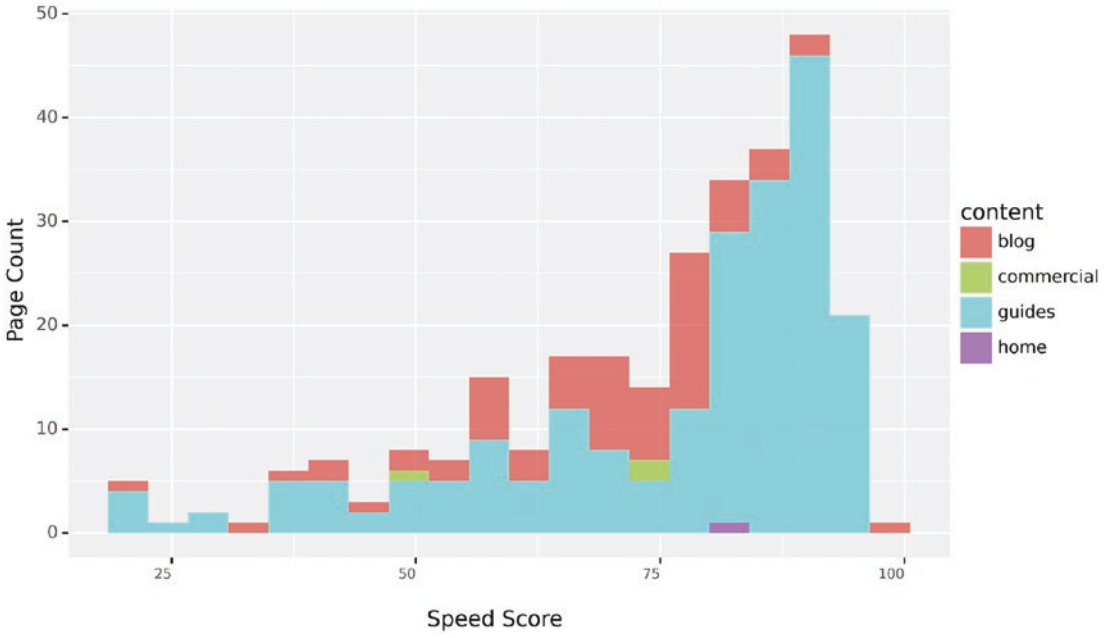

Figure 3-24. *Distribution of speed score by content type*

Let's drill down by CWV score category, starting with CLS:

```
target_CLS_plt = (
    ggplot(target_speedDist_df,
           aes(x = 'cumulative_layout_shift', fill = 'content')) +
    geom_histogram(alpha = 0.8, bins = 20) +
    labs(y = 'Page Count', x = '\ncumulative_layout_shift') +
```

```
    #scale_x_continuous(breaks=range(0, 100, 20)) +
    theme(legend_position = 'right',
          axis_text_x = element_text(rotation=90, hjust=1, size = 7))
)

target_CLS_plt.save(filename = 'images/3_target_CLS_plt.png',
                            height=5, width=8, units = 'in', dpi=1000)
target_CLS_plt
```

As shown in target_CLS_plt (Figure 3-25), guides have the least amount of shifting during browser rendering, whereas the blogs and the home page shift the most.

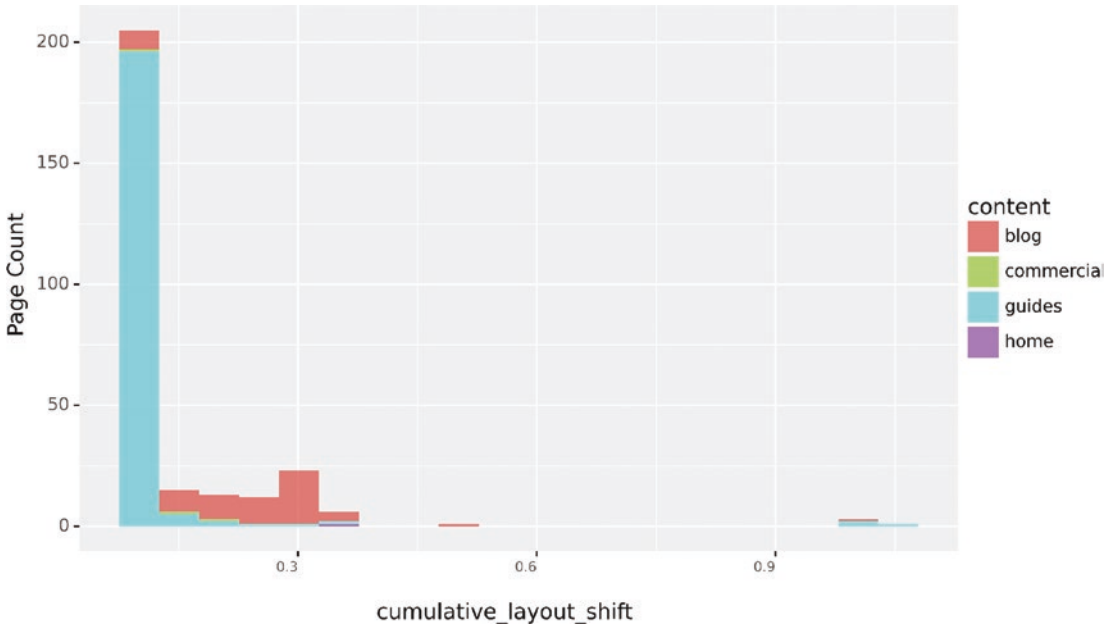

Figure 3-25. *Distribution of CLS by content type*

So we now know which content templates to focus our CLS development efforts.

```
target_FCP_plt = (
    ggplot(target_speedDist_df,
           aes(x = 'first_contentful_paint', fill = 'content')) +
    geom_histogram(alpha = 0.8, bins = 30) +
    labs(y = 'Page Count', x = '\nContentful paint') +
    theme(legend_position = 'right',
```

```
          axis_text_x = element_text(rotation=90, hjust=1, size = 7))
)

target_FCP_plt.save(filename = 'images/3_target_FCP_plt.png',
                            height=5, width=8, units = 'in', dpi=1000)
target_FCP_plt
```

In this area, target_FCP_plt (Figure 3-26) shows no discernible trends here which indicates it's an overall site problem. So digging into the Chrome Developer Tools and looking into the network logs would be the obvious next step.

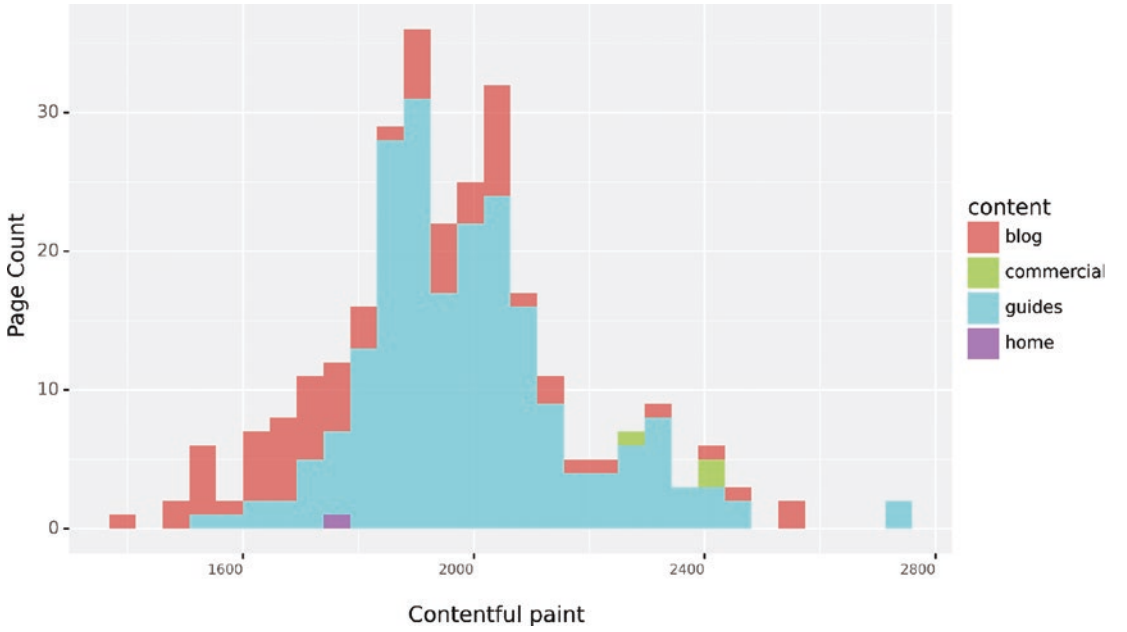

Figure 3-26. *Distribution of FCP by content type*

```
target_LCP_plt = (
    ggplot(target_speedDist_df,
           aes(x = 'largest_contentful_paint', fill = 'content')) +
    geom_histogram(alpha = 0.8, bins = 20) +
    labs(y = 'Page Count', x = '\nlargest_contentful_paint') +
    theme(legend_position = 'right',
          axis_text_x = element_text(rotation=90, hjust=1, size = 7))
)
```

```
target_LCP_plt.save(filename = 'images/3_target_LCP_plt.png',
                                height=5, width=8, units = 'in', dpi=1000)
target_LCP_plt
```

target_LCP_plt (Figure 3-27) shows most guides and some blogs have the fastest LCP scores; in any case, the blog template and the rogue guides would be the areas of focus.

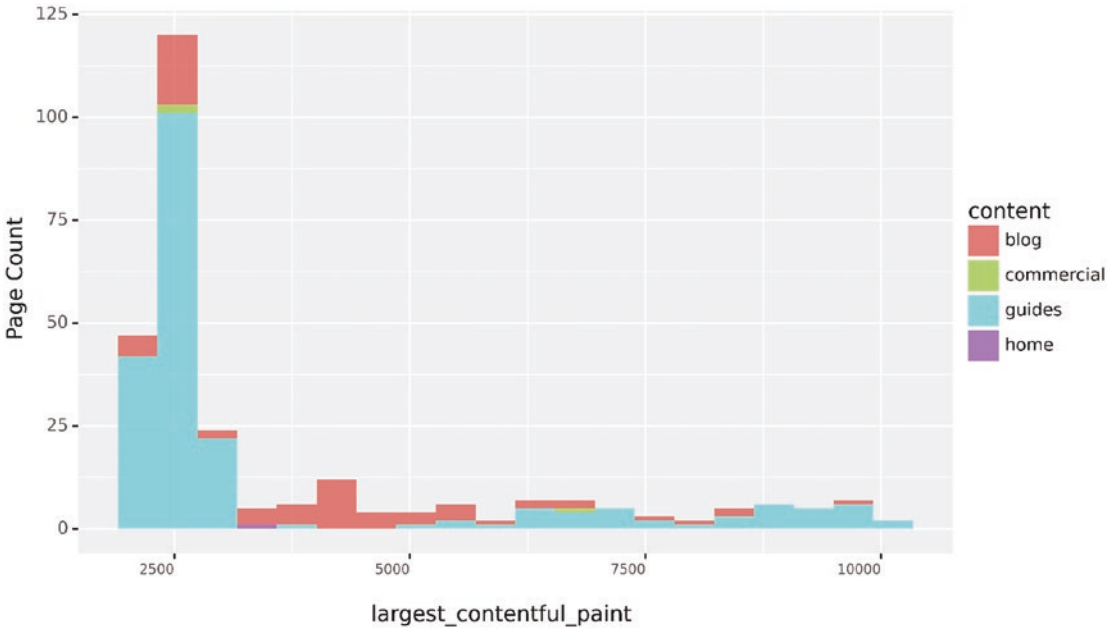

Figure 3-27. *Distribution of LCP by content type*

```
target_FID_plt = (
    ggplot(target_speedDist_df,
           aes(x = 'time_to_interactive', fill = 'content')) +
    geom_histogram(alpha = 0.8, bins = 20) +
    labs(y = 'Page Count', x = '\ntime_to_interactive') +
    theme(legend_position = 'right',
          axis_text_x = element_text(rotation=90, hjust=1, size = 7))
)

target_FID_plt.save(filename = 'images/3_target_FID_plt.png',
                                height=5, width=8, units = 'in', dpi=1000)
target_FID_plt
```

The majority of the site appears in target_FID_plt (Figure 3-28) to enjoy fast FID times, so this would be the least priority for CWV improvement.

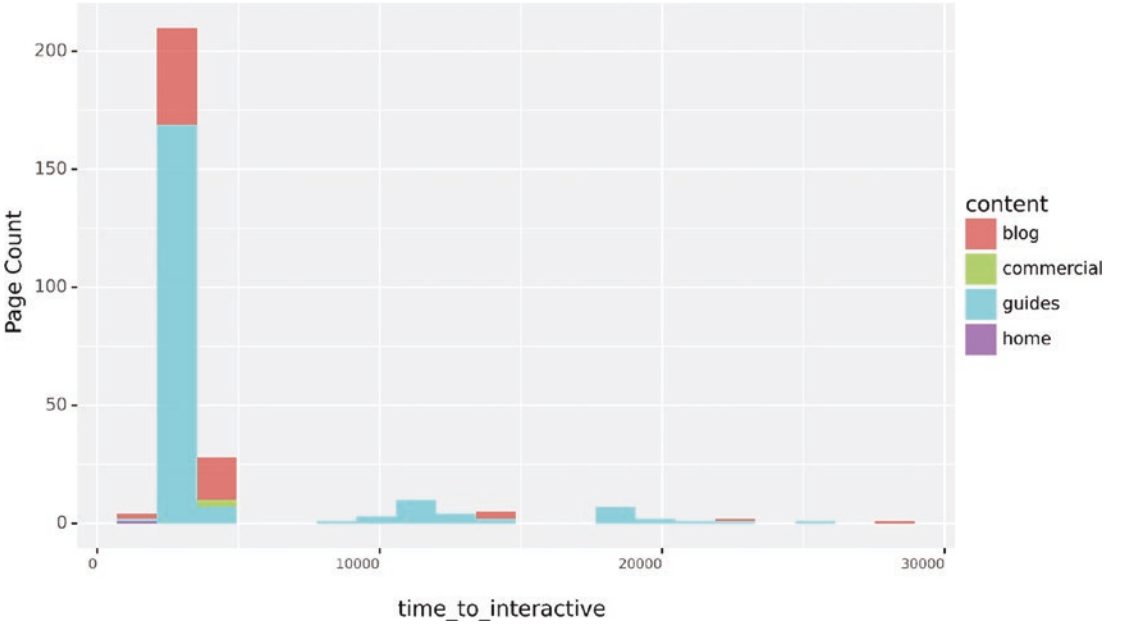

Figure 3-28. *Distribution of FID by content type*

Summary

In this chapter, we covered how data-driven approach could be taken toward technical SEO by way of

- Modeling page authority to estimate the benefit of technical SEO recommendations to colleagues and clients
- Internal link optimization analyzed in different ways to improve content discoverability and labeling via anchor text
- Core Web Vitals to see which metrics require improvement and by content type

The next chapter will focus on using data to improve content and UX.

CHAPTER 4

Content and UX

Content and UX for SEO is about the quality of the experience you're delivering to your website users, especially when they are referred from search engines. This means a number of things including but not limited to

- Having the content your target audiences are searching for
- Content that best satisfies the user query
  - *Content creation*: Planning landing page content
  - *Content consolidation*: (I) Splitting content (in instances where "too much" content might be impacting user satisfaction or hindering search engines from understanding the search intent the content is targeting) and (II) merging content (in instances where multiple pages are competing for the same intent)
- Fast to load – ensuring you're delivering a good user experience (UX)
- Renders well on different device types

By no means do we claim that this is the final word on data-driven SEO from a content and UX perspective. What we will do is expose data-driven ways of solving the most important SEO challenge using data science techniques, as not all require data science.

For example, getting scientific evidence that fast page speeds are indicative of higher ranked pages uses similar code from Chapter 6. Our focus will be on the various flavors of content that best satisfies the user query: keyword mapping, content gap analysis, and content creation.

A. Voniatis, *Data-Driven SEO with Python*, https://doi.org/10.1007/978-1-4842-9175-7_4

Content That Best Satisfies the User Query

An obvious challenge of SEO is deciding which content should go on which pages. Arguably, getting this right means you're optimizing for Google's RankBrain (a component of Google's core algorithm which uses machine learning to help understand and process user search queries).

While many crawling tools provide visuals of the distributions of pages by site depth or by segment, for example, data science enables you to benefit from a richer level of detail. To help you work out the content that best satisfies the user query, you need to

- Map keywords to content
- Plan content sections for those landing pages
- Decide what content to create for target keywords that will satisfy users searching for them

Data Sources

Your most likely data sources will be a combination of

- Site auditor URL exports
- SERPs tracking tools

Keyword Mapping

While there is so much to be gained from creating value-adding content, there is also much to be gained from retiring or consolidating content. This is achieved by merging it with another on the basis that they share the same search intent. Assuming the keywords have been grouped together by search intent, the next stage is to map them.

Keyword mapping is the process of mapping target keywords to pages and then optimizing the page toward these – as a result, maximizing a site's rank position potential in the search result. There are a number of approaches to achieve this:

- TF-IDF
- String matching

- Third-party neural network models (BERT, GPT-3)
- Build your own AI

We recommend string matching as it's fast, reasonably accurate, and the easiest to deploy.

String Matching

String matching works to see how many strings overlap and is used in DNA sequencing. String matching can work in two ways, which are to either treat strings as one object or strings made up of tokens (i.e., words within a string). We're opting for the latter because words mean something to humans and are not serial numbers. For that reason, we'll be using Sorensen-Dice which is fast and accurate compared to others we've tested.

The following code extract shows how we use string distance to map keywords to content by seeking the most similar URL titles to the target keyword. Let's go, importing libraries:

```
import requests
from requests.exceptions import ReadTimeout
from json.decoder import JSONDecodeError
import re
import time
import random
import pandas as pd
import numpy as np
import datetime
from client import RestClient
import json
import py_stringmatching as sm
from textdistance import sorensen_dice
from plotnine import *
import matplotlib.pyplot as plt

target = 'wella'
```

We'll start by importing the crawl data, which is a CSV export of website auditing software, in this case from "Sitebulb":

```
crawl_raw = pd.read_csv('data/www_wella_com_internal_html_urls_by_
indexable_status_filtered_20220629220833.csv')
```

Clean up the column heading title texts using a list comprehension:

```
crawl_raw.columns = [col.lower().replace('(','').replace(')','').
replace('%','').replace(' ', '_')
  for col in crawl_raw.columns]

crawl_df = crawl_raw.copy()
```

We're only interested in indexable pages as those are the URLs available for mapping:

```
crawl_df = crawl_df.loc[crawl_df['indexable'] == 'Yes']
crawl_df
```

This results in the following:

	url	ur	crawl_depth	scheme	crawl_source	first_parent_url	url_source	http_status	http_status_code	indexable
0	https://www.wella.com/international/wella-tuto...	7	Not Set	https	XML Sitemap	None	XML Sitemap	OK	200	Yes
1	https://www.wella.com/international/hair-style...	1	Not Set	https	XML Sitemap	None	XML Sitemap	OK	200	Yes
2	https://www.wella.com/international/hair-style...	4	Not Set	https	XML Sitemap	None	XML Sitemap	OK	200	Yes

The crawl import is complete. However, we're only interested in the URL and title as that's all we need for mapping keywords to URLs. Still it's good to import the whole file to visually inspect it, to be more familiar with the data.

```
urls_titles = crawl_df[['url', 'title']].copy()
urls_titles
```

This results in the following:

	url	title
0	https://www.wella.com/international/wella-tuto...	How to Style a Faux Hawk Like a Headline Act |...
1	https://www.wella.com/international/hair-style...	Silvikrin Classic Voluminous Hold Hairspray 75...
2	https://www.wella.com/international/hair-style...	Wellaflex 2nd Day Volume Strong Hold Mousse, H...
3	https://www.wella.com/international/hair-color...	Wella Koleston Permanent Hair Color Cream Fore...
4	https://www.wella.com/international/hair-color...	Wella Koleston Permanent Hair Color Cream With...
5	https://www.wella.com/international/hair-style...	Wellaflex Mega Strong Hold Hairspray, Hold: 5+...
6	https://www.wella.com/international/hair-style...	Wella Deluxe 24 Hour Wonder Volume Mousse 75 m...
7	https://www.wella.com/international/hair-color...	Wella Koleston Permanent Hair Color Cream With...
8	https://www.wella.com/international/hair-style...	Wellaflex Hydro Style Extra Strong Hold Hairsp...
9	https://www.wella.com/international/hair-style...	Wella Shockwaves Ultra Strong Power Hold Gel S...
10	https://www.wella.com/international/hair-color...	Wella Koleston Permanent Hair Color Cream With...

The dataframe is showing the URLs and titles. Let's load the keywords we want to map that have been clustered using techniques in Chapter 2:

```
keyword_discovery = pd.read_csv('data/keyword_discovery.csv')
```

This results in the following:

	topic	keyword	se_results_count	topic_results	topic_group
7	black hair	black hair	7340000000	7340000000	1
14	brown hair	brown hair	5170000000	5170000000	2
8	blonde hair	blonde hair	2730000000	2730000000	3
20	color perfect	color perfect	2270000000	2270000000	4
104	virtual try on	virtual try on	1670000000	2270000000	4
103	virtual try on	virtual try on tool	600000000	2270000000	4
61	how hair coloring works	how hair coloring works	2140000000	2140000000	5
26	cover roots	quick hair root touch ups	816000000	1941000000	6
27	cover roots	the best root touch	169000000	1941000000	6
28	cover roots	cover roots at home fast	197000000	1941000000	6
29	cover roots	cover roots	390000000	1941000000	6

The dataframe shows the topics, keywords, number of search engine results for the keywords, topic web search results, and the topic group. Note these were clustered using the methods disclosed in Chapter 2.

We'll map the topic as this is the central keyword that would also rank for their topic group keywords. This means we only require the topic column.

```
total_mapping_simi = keyword_discovery[['topic']].copy().drop_duplicates()
```

We want all the combinations of topics and URL titles before we can test each combination for string similarity. We achieve this using the cross-product merge:

```
total_mapping_simi = total_mapping_simi.merge(urls_titles, how = 'cross')
```

A new column "test" is created which will be formatted to remove boilerplate brand strings and force lowercase. This will make the string matching values more accurate.

```
total_mapping_simi['test'] = total_mapping_simi['title']
total_mapping_simi['test'] = total_mapping_simi['test'].str.lower()
total_mapping_simi['test'] = total_mapping_simi['test'].str.replace(' \|
wella', '')

total_mapping_simi
```

This results in the following:

	topic	url	title	test
0	black hair	https://www.wella.com/international/wella-tuto...	How to Style a Faux Hawk Like a Headline Act |...	how to style a faux hawk like a headline act
1	black hair	https://www.wella.com/international/hair-style...	Silvikrin Classic Voluminous Hold Hairspray 75...	silvikrin classic voluminous hold hairspray 75ml
2	black hair	https://www.wella.com/international/hair-style...	Wellaflex 2nd Day Volume Strong Hold Mousse, H...	wellaflex 2nd day volume strong hold mousse, h...
3	black hair	https://www.wella.com/international/hair-color...	Wella Koleston Permanent Hair Color Cream Fore...	wella koleston permanent hair color cream fore...
4	black hair	https://www.wella.com/international/hair-color...	Wella Koleston Permanent Hair Color Cream With...	wella koleston permanent hair color cream with...
...	...	...	...	...
30815	half up top knot	https://www.wella.com/international/curly-and-...	Style Wavy & Curly Hair | Curly Hair Products ...	style wavy & curly hair | curly hair products
30816	half up top knot	https://www.wella.com/international/wella-maga...	Coloring at home for the first time? | Wella	coloring at home for the first time?
30817	half up top knot	https://www.wella.com/international/styling	Wella - Hair passion and expertise, shared wit...	wella - hair passion and expertise, shared wit...
30818	half up top knot	https://www.wella.com/international/about-well...	Wella Color by You | Mix, match and wear the h...	wella color by you | mix, match and wear the h...
30819	half up top knot	https://www.wella.com/international/wella-x-you	Wella X You LP | Wella	wella x you lp

30820 rows × 4 columns

Now we're ready to compare strings by creating a new column "simi," meaning string similarity. The scores will take the topic and test columns as inputs and feed the sorensen_dice function imported earlier:

```
total_mapping_simi['simi'] = total_mapping_simi.loc[:, ['topic',
      'test']].apply(lambda x: sorensen_dice(*x), axis=1)
total_mapping_simi
```

	topic	url	title	test	simi
0	black hair	https://www.wella.com/international/wella-tuto...	How to Style a Faux Hawk Like a Headline Act |...	how to style a faux hawk like a headline act	0.296296
1	black hair	https://www.wella.com/international/hair-style...	Silvikrin Classic Voluminous Hold Hairspray 75...	silvikrin classic voluminous hold hairspray 75ml	0.310345
2	black hair	https://www.wella.com/international/hair-style...	Wellaflex 2nd Day Volume Strong Hold Mousse, H...	wellaflex 2nd day volume strong hold mousse, h...	0.166667
3	black hair	https://www.wella.com/international/hair-color...	Wella Koleston Permanent Hair Color Cream Fore...	wella koleston permanent hair color cream fore...	0.227273
4	black hair	https://www.wella.com/international/hair-color...	Wella Koleston Permanent Hair Color Cream With...	wella koleston permanent hair color cream with...	0.202020
...	...	...	...	...	...
30815	half up top knot	https://www.wella.com/international/curly-and-...	Style Wavy & Curly Hair | Curly Hair Products ...	style wavy & curly hair | curly hair products	0.360656
30816	half up top knot	https://www.wella.com/international/wella-maga...	Coloring at home for the first time? | Wella	coloring at home for the first time?	0.461538
30817	half up top knot	https://www.wella.com/international/styling	Wella - Hair passion and expertise, shared wit...	wella - hair passion and expertise, shared wit...	0.382353
30818	half up top knot	https://www.wella.com/international/about-well...	Wella Color by You | Mix, match and wear the h...	wella color by you | mix, match and wear the h...	0.296296
30819	half up top knot	https://www.wella.com/international/wella-x-you	Wella X You LP | Wella	wella x you lp	0.533333

30820 rows × 5 columns

The simi column has been added complete with scores. A score of 1 is identical, and 0 is completely dissimilar. The next stage is to select the closest matching URLs to topic keywords:

```
keyword_mapping_grp = total_mapping_simi.copy()
```

The dataframe is first sorted by similarity score and topic in descending order so that the first row by topic is the closest matching:

```
keyword_mapping_grp = keyword_mapping_grp.sort_values(['simi', 'topic'],
ascending = False)
```

After sorting, we use the first() function to select the top matching URL for each topic using the groupby() function:

```
keyword_mapping_grp = keyword_mapping_grp.groupby('topic').first().
reset_index()

keyword_mapping_grp
```

This results in the following:

	topic	url	title	test	simi
0	80s rock'n'roll hairstyle	https://www.wella.com/international/wella-tuto...	Get the 80's rock'n'roll hairstyle.	get the 80's rock'n'roll hairstyle.	0.833333
1	accessorize braided hairstyle	https://www.wella.com/international/wella-tuto...	Accessorize your braided hairstyle for a chic ...	accessorize your braided hairstyle for a chic ...	0.725000
2	back comb safely	https://www.wella.com/international/wella-tuto...	Back comb safely for stylish results!	back comb safely for stylish results!	0.603774
3	bardot look	https://www.wella.com/international/wella-tuto...	BARDOT LOOK | Wella	bardot look	1.000000
4	beachy waves hairstyle	https://www.wella.com/international/wella-maga...	Hair color safety tests | Wella	hair color safety tests	0.666667
5	best hair color results	https://www.wella.com/international/wella-maga...	Hair color safety tests | Wella	hair color safety tests	0.826087
6	big hair volume	https://www.wella.com/international/blonde-hair	Blonde Hair | Wella	blonde hair	0.692308
7	black hair	https://www.wella.com/international/black-hair	Black Hair | Wella	black hair	1.000000
8	blonde hair	https://www.wella.com/international/blonde-hair	Blonde Hair | Wella	blonde hair	1.000000

Each topic now has its closest matching URL. The next stage is to decide whether these matches are good enough or not:

```
keyword_mapping = keyword_mapping_grp[['topic', 'url', 'title',
'simi']].copy()
```

At this point, we eyeball the data to see what threshold number is good enough. I've gone with 0.7 or 70% as it seems to do the job mostly correctly, which is to act as the natural threshold for matching test content to URLs.

Using np.where(), which is equivalent to Excel's IF formula, we'll make any rows exceeding 0.7 as "mapped" and the rest as "unmatched":

```
keyword_mapping['url'] = np.where(keyword_mapping['simi'] < 0.7,
'unmatched', keyword_mapping['url'])
keyword_mapping['mapped'] = np.where(keyword_mapping['simi'] =< 0.7,
'No', 'Yes')

keyword_mapping
```

This results in the following:

	topic	url	title	simi	mapped
0	80s rock'n'roll hairstyle	https://www.wella.com/international/wella-tuto...	Get the 80's rock'n'roll hairstyle.	0.833333	Yes
1	accessorize braided hairstyle	https://www.wella.com/international/wella-tuto...	Accessorize your braided hairstyle for a chic ...	0.725000	Yes
2	back comb safely	unmatched	Back comb safely for stylish results!	0.603774	No
3	bardot look	https://www.wella.com/international/wella-tuto...	BARDOT LOOK | Wella	1.000000	Yes
4	beachy waves hairstyle	unmatched	Hair color safety tests | Wella	0.666667	No
5	best hair color results	https://www.wella.com/international/wella-maga...	Hair color safety tests | Wella	0.826087	Yes
6	big hair volume	unmatched	Blonde Hair | Wella	0.692308	No
7	black hair	https://www.wella.com/international/black-hair	Black Hair | Wella	1.000000	Yes
8	blonde hair	https://www.wella.com/international/blonde-hair	Blonde Hair | Wella	1.000000	Yes
9	blunt cut bob	unmatched	Fun and functional braids.	0.512821	No
10	boost fine & thinning hair	https://www.wella.com/international/fine-and-t...	Boost Fine & Thinning Hair | Styling Essential...	0.712329	Yes

Finally, we have keywords mapped to URLs and some stats on the overall exercise.

```
keyword_mapping_aggs = keyword_mapping.copy()
keyword_mapping_aggs = keyword_mapping_aggs.groupby('mapped').count().
reset_index()
```

```
Keyword_mapping_aggs
```

This results in the following:

	mapped	topic	url	title	simi
0	No	32	32	32	32
1	Yes	60	60	60	60

String Distance to Map Keyword Evaluation

So 65% of the 92 URLs got mapped – not bad and for the minimum code too. Those unmapped will have to be done manually, probably because

- Existing unmapped URL titles are not optimized.
- New content needs to be created.

Content Gap Analysis

Search engines require content to rank as a response to a keyword search by their users. Content gap analysis helps your site extend its reach to your target audiences by identifying keywords (and topics) where your direct competitors are visible, and your site is not.

The analysis is achieved by using search analytics data sources such as SEMRush overlaying your site data with your competitors to find

- *Core content set*: Of which keywords are common to multiple competitors
- *Content gaps*: The extent to which the brand is not visible for keywords that form the content set

Without this analysis, your site risks being left behind in terms of audience reach and also appearing less authoritative because your site appears less knowledgeable about the topics covered by your existing content. This is particularly important when considering the buying cycle. Let's imagine you're booking a holiday, and now imagine the variety of search queries that you might use as you carry out that search, perhaps searching by destination ("beach holidays to Spain"), perhaps refining by a specific requirement ("family beach holidays in Spain"), and then more specific including a destination (Majorca), and perhaps ("family holidays with pool in Majorca"). Savvy SEOs think deeply about mapping customer demand (right across the search journey) to compelling landing page (and website) experiences that can satisfy this demand. Data science enables you to manage this opportunity at a significant scale.

Warnings and motivations over, let's roll starting with the usual package loading:

```
import re
import time
import random
import pandas as pd
import numpy as np
```

OS and Glob allow the environment to read the SEMRush files from a folder:

```
import os
import glob

from pandas.api.types import is_string_dtype
from pandas.api.types import is_numeric_dtype
import uritools
```

Combinations is particularly useful for generating combinations of list elements which will be used to work out which datasets to intersect and in a given order:

```
from itertools import combinations
```

To see all columns of a dataframe and without truncation:

```
pd.set_option('display.max_colwidth', None)
```

These variables are set in advance so that when copying this script over for another site, the script can be run with minimal changes to the code:

```
root_domain = 'wella.com'
hostdomain = 'www.wella.com'
hostname = 'wella'
full_domain = 'https://www.wella.com'
target_name = 'Wella'
```

With the variables set, we're now ready to start importing data.

Getting the Data

We set the directory path where all of the SEMRush files are stored:

```
data_dir = os.path.join('data/semrush/')
```

Glob reads all of the files in the folder, and we store the output in a variable "semrush_csvs":

```
semrush_csvs = glob.glob(data_dir + "/*.csv")
Semrush_csvs
```

Print out the files in the folder:

```
['data/hair.com-organic.Positions-uk-20220704-2022-07-05T14_04_59Z.csv',
 'data/johnfrieda.com-organic.Positions-
 uk-20220704-2022-07-05T13_29_57Z.csv',
 'data/madison-reed.com-organic.Positions-
 uk-20220704-2022-07-05T13_38_32Z.csv',
 'data/sebastianprofessional.com-organic.Positions-
 uk-20220704-2022-07-05T13_39_13Z.csv',
 'data/matrix.com-organic.Positions-uk-20220704-2022-07-05T14_04_12Z.csv',
 'data/wella.com-organic.Positions-uk-20220704-2022-07-05T13_30_29Z.csv',
 'data/redken.com-organic.Positions-uk-20220704-2022-07-05T13_37_31Z.csv',
 'data/schwarzkopf.com-organic.Positions-
 uk-20220704-2022-07-05T13_29_03Z.csv',
 'data/garnier.co.uk-organic.Positions-
 uk-20220704-2022-07-05T14_07_16Z.csv']
```

Initialize the final dataframe where we'll be storing the imported SEMRush data:

```
semrush_raw_df = pd.DataFrame()
```

Initialize a list where we'll be storing the imported SEMRush data:

```
semrush_li = []
```

The for loop uses the pandas read_csv() function to read the SEMRush CSV file and extract the filename which is put into a new column "filename." A bit superfluous to requirements but it will help us know where the data came from.

Once the data is read, it is added to the semrush_li list we initialized earlier:

```
for cf in semrush_csvs:
    df = pd.read_csv(cf, index_col=None, header=0)
    df['filename'] = os.path.basename(cf)
    df['filename'] = df['filename'].str.replace('.csv', '')
    df['filename'] = df['filename'].str.replace('_', '.')
    semrush_li.append(df)

semrush_raw_df = pd.concat(semrush_li, axis=0, ignore_index=True)
```

Clean up the columns to make these lowercase and data-friendly. A list comprehension can also be used, but we used a different approach to show an alternative.

```
semrush_raw_df.columns = semrush_raw_df.columns.str.strip().str.lower().
str.replace(' ', '_').str.replace('(', '').str.replace(')', '')
```

A site column is created so we know which content the site belongs to. Here, we used regex on the filename column, but we could have easily derived this from the URL also:

```
semrush_raw_df['site'] = semrush_raw_df['filename'].str.extract('(.*?)\-')
semrush_raw_df.head()
```

This results in the following:

	keyword	position	previous_position	search_volume	keyword_difficulty	cpc	url	traffic	traffic_%	traffic_cost	competition	number_
0	blonde balayage	6	6	14800	47	3.55	https://www.hair.com/blonde-balayage-ideas.html	518	2.24	1838.0	0.51	
1	ginger hair color	2	2	2400	47	2.11	https://www.hair.com/ginger-hair-color.html	316	1.36	668.0	0.99	2
2	hair colors for pale skin	1	1	1000	51	0.94	https://www.hair.com/hair-color-for-pale-skin.html	248	1.07	233.0	0.83	
3	dark brown hair	9	12	9900	62	3.72	https://www.hair.com/dark-brown-hair-ideas.html	237	1.02	883.0	0.66	2
4	best purple shampoo	5	5	4400	66	0.40	https://www.hair.com/best-purple-shampoo.html	193	0.83	77.0	1.00	

That's the dataframe, although we're more interested in the keywords and the site it belongs to.

```
semrush_raw_presect = semrush_raw_sited.copy()
semrush_raw_presect = semrush_raw_presect[['keyword', 'site']]
semrush_raw_presect
```

This results in the following:

	keyword	site
0	blonde balayage	hair.com
1	ginger hair color	hair.com
2	hair colors for pale skin	hair.com
3	dark brown hair	hair.com
4	best purple shampoo	hair.com
...	...	...
118676	honey blonde hair dye for dark hair	garnier.co.uk
118677	dye hair blonde	garnier.co.uk
118678	colour touch shade chart	garnier.co.uk
118679	red and blue hair dye	garnier.co.uk
118680	garnier 3-1	garnier.co.uk

118681 rows × 2 columns

The aim of the exercise is to find keywords to two or more competitors which will define the core content set.

To achieve this, we will use a list comprehension to split the semrush_raw_presect dataframe by site into unnamed dataframes:

```
df1, df2, df3, df4, df5, df6, df7, df8, df9 = [x for _, x in semrush_raw_
presect.groupby(semrush_raw_presect['site'])]
```

Now that each dataframe has the site and keywords, we can dispense with the site column as we're only interested in the keywords and not where they come from.

We start by defining a list of dataframes, df_list:

```
df_list = [df1, df2, df3, df4, df5, df6, df7, df8, df9]
```

Here's an example; df1 is Garnier:

```
df1
```

This results in the following:

	keyword	site
100596	garnier	garnier.co.uk
100597	hair colour	garnier.co.uk
100598	garnier.co.uk	garnier.co.uk
100599	garnier hair color	garnier.co.uk
100600	garnier hair colour	garnier.co.uk
...	...	...
118676	honey blonde hair dye for dark hair	garnier.co.uk
118677	dye hair blonde	garnier.co.uk
118678	colour touch shade chart	garnier.co.uk
118679	red and blue hair dye	garnier.co.uk
118680	garnier 3-1	garnier.co.uk

18085 rows × 2 columns

Define the function drop_col, which as the name suggests

1. Drops the column (col) of the dataframe (df)
2. Takes the desired column (list_col)
3. Converts the desired column to a list
4. Adds the column to a big list (master_list)

```
def drop_col(df, col, listcol, master_list):
    df.drop(col, axis = 1, inplace = True)
    df_tolist = df[listcol].tolist()
    master_list.append(df_tolist)
```

Our master list is initiated as follows:

```
keywords_lists = []
```

List comprehension which will go through all of the keyword sets in df_list, and these as lists to get a list of keyword lists.

```
_ = [drop_col(x, 'site', 'keyword', keywords_lists) for x in df_list]
```

The lists within the list of lists are too long to print here; however, the double bracket at the beginning should show this is indeed a list of lists.

```
keywords_lists
```

This results in the following:

```
[['garnier',
  'hair colour',
  'garnier.co.uk',
  'garnier hair color',
  'garnier hair colour',
  'garnier micellar water',
  'garnier hair food',
  'garnier bb cream',
  'garnier face mask',
  'bb cream from garnier',
  'garnier hair mask',
  'garnier shampoo',
  'hair dye',
```

The list of keyword lists is exported into separated lists:

```
lst_1, lst_2, lst_3, lst_4, lst_5, lst_6, lst_7, lst_8, lst_9 =
keywords_lists
```

List 1 is shown as follows:

```
lst_1
```

This results in the following:

```
['garnier',
 'hair colour',
 'garnier.co.uk',
 'garnier hair color',
 'garnier hair colour',
 'garnier micellar water',
 'garnier hair food',
 'garnier bb cream',
 'garnier face mask',
 'bb cream from garnier',
 'garnier hair mask',
 'garnier shampoo',
 'hair dye',
 'garnier hair dye',
 'garnier shampoo bar',
 'garnier vitamin c serum',
```

Now we want to generate combinations of lists so we can control how each of the site's keywords get intersected:

```
values_list = [lst_1, lst_2, lst_3, lst_4, lst_5, lst_6, lst_7,
lst_8, lst_9]
```

The dictionary comprehension will append each list into a dictionary we create called keywords_dict, where the key (index) is the number of the list:

```
keywords_dict = {listo: values_list[listo]  for listo in
range(len(values_list))}
```

When we print the keywords_dict keys

```
keywords_dict.keys()
```

we get the list numbers. The reason it goes from 0 to 8 and not 1 to 9 is because Python uses zero indexing which means it starts from zero:

```
dict_keys([0, 1, 2, 3, 4, 5, 6, 7, 8])
```

Now we'll convert the keys to a list for ease of manipulation shortly:

```
keys_list = list(keywords_dict.keys())
keys_list
```

This results in the following:

```
[0, 1, 2, 3, 4, 5, 6, 7, 8]
```

With the list, we can construct combinations of the site's keywords to intersect. The intersection of the website keyword lists will be the words that are common to the websites.

Creating the Combinations

Initialize list_combos which will be a list of the combinations generated:

```
list_combos = []
```

List comprehension using the combinations function picking four site keywords at random and storing it in list combos using the append() function:

```
_ = [list_combos.append(comb) for comb in combinations(keys_list, 4)]
```

This line converts the combination into a list so that list_combos will be a list of lists:

```
list_combos = [list(combo) for combo in list_combos]


list_combos
```

This results in the following:

```
[[0, 1, 2, 3],
 [0, 1, 2, 4],
 [0, 1, 2, 5],
 [0, 1, 2, 6],
 [0, 1, 2, 7],
 [0, 1, 2, 8],
 [0, 1, 3, 4],
 [0, 1, 3, 5],
 [0, 1, 3, 6], ...
```

With the list of lists, we're ready to start intersecting the keyword lists to build the core content (keyword) set.

Finding the Content Intersection

Initialize an empty list keywords_intersected:

```
keywords_intersected = []
```

Define the multi_intersect function which takes a list of dictionaries and their keys, then finds the common keywords (i.e., intersection), and adds it to the keywords_intersected list.

The function can be adapted to just compare two sites, three sites, and so on. Just ensure you rerun the combinations function with the number of lists desired and edit the function as follows:

```
def multi_intersect(list_dict, combo):
    a = list_dict[combo[0]]
    b = list_dict[combo[1]]
    c = list_dict[combo[2]]
    d = list_dict[combo[3]]
    intersection = list(set(a) & set(b) & set(c) & set(d))
    keywords_intersected.append(intersection)
```

Using the list comprehension, we loop through the list of combinations list_combos to run the multi_intersect function which takes the dictionary containing all the site keywords (keywords_dict), pulls the appropriate keywords, and finds the common ones, before adding to keywords_intersected:

```
_ = [multi_intersect(keywords_dict, combo) for combo in list_combos]
```

And we get a list of lists, because each list is an iteration of the function for each combination:

```
keywords_intersected
```

This results in the following:

```
[['best way to cover grey hair',
  'rich red hair colour',
  'hair dye colors chart',
  'different shades of blonde hair',
  'adding colour to grey hair',
  'cool hair colors',
  'dark red hair',
  'light brown toner',
  'medium light brown hair',
  'hair color on brown skin',
  'highlights to cover grey in dark brown hair',
  'auburn color swatch', ..
```

Let's turn the list of lists into a single list:

```
flat_keywords_intersected = [elem for sublist in keywords_intersected for
elem in sublist]
```

Then deduplicate it. list(set(the_list_you_want_to_de-duplicate)) is a really helpful technique to deduplicate lists.

```
unique_keywords_intersected = list(set(flat_keywords_intersected))
print(len(flat_keywords_intersected), len(unique_keywords_intersected))
```

This results in the following:

```
87031 8380
```

There were 87K keywords originally and 8380 keywords post deduplication.

```
unique_keywords_intersected
```

This results in the following:

```
['hairspray for holding curls',
 'burgundy colour hair',
 'cool hair colors',
 'dark red hair',
 'color stripes hair',
```

```
 'for frizzy hair products',
 'blue purple hair',
 'autumn balayage 2021',
 'ash brown hair color',
 'blonde highlights in black hair',
 'what hair colour will suit me',
 'hair gloss treatment at home',
 'dark roots with red hair',
 'silver shoulder length hair',
 'mens curly hair',
 'ash brunette hair',
 'toners for grey hair',
```

That's the list, but it's not over yet as we need to establish the gap, which we all want to know.

Establishing Gap

The question is which keywords are "Wella" not targeting and how many are there?

We'll start by filtering the SEMRush site for the target site Wella.com:

```
target_semrush = semrush_raw_sited.loc[semrush_raw_sited['site'] ==
root_domain]
```

And then we include only the keywords in the core content set:

```
target_on = target_semrush.loc[target_semrush['keyword'].isin(unique_
keywords_intersected)]
target_on
```

This results in the following:

	keyword	position	previous_position	search_volume	keyword_difficulty	cpc	url	traffic	traffic_%	traffic_cost	comp
60526	balayage	9	11	60500	72	3.32	https://blog.wella.com/gb/what-is-balayage	1452	1.04	4820.0	
60537	brown to blonde hair	1	1	2400	48	1.49	https://blog.wella.com/gb/foolproof-way-go-brown-blonde-hair	595	0.42	886.0	
60542	copper hair	4	6	8100	40	2.78	https://blog.wella.com/gb/copper-red-hair-color	526	0.37	1463.0	
60545	auburn hair	10	9	22200	54	2.24	https://blog.wella.com/gb/auburn-hair-color-ideas-and-formulas	488	0.34	1094.0	
60546	yellow hair	1	1	1900	30	0.00	https://blog.wella.com/gb/how-to-tone-yellow-hair	471	0.33	0.0	

Let's get some stats starting with the number of keywords in the preceding dataframe and the number of keywords in the core content set:

```
print(target_on[['keyword'].drop_duplicates().shape[0], len(unique_
keywords_intersected))
```

This results in the following:

```
6936 8380
```

So just under 70% of Wella's keyword content is in the core content set, which is about 1.4K keywords short.

To find the 6.9K intersect keywords, we can use the list and set functions:

```
target_on_list = list(set(target_semrush['keyword'].tolist()) & set(unique_
keywords_intersected))
target_on_list[:10]
```

This results in the following:

```
['hairspray for holding curls',
 'burgundy colour hair',
 'cool hair colors',
 'dark red hair',
 'blue purple hair',
```

```
 'autumn balayage 2021',
 'ash brown hair color',
 'blonde highlights in black hair',
 'what hair colour will suit me',
 'hair gloss treatment at home']
```

To find the keywords that are not in the core content set, that is, the content gap, we'll remove the target SEMRush keywords from the core content set:

```
target_gap = list(set(unique_keywords_intersected) - set(target_
semrush['keyword'].tolist()))
print(len(target_gap), len(unique_keywords_intersected))
target_gap[:10]
```

This results in the following:

```
['bleaching hair with toner',
 'color stripes hair',
 'for frizzy hair products',
 'air dry beach waves short hair',
 'does semi permanent black dye wash out',
 'balayage for dark skin',
 'matte hairspray',
 'mens curly hair',
 'how to change hair color',
 'ginger and pink hair']
```

Now that we know what these gap keywords are, we can filter the dataframe by listing keywords:

```
cga_semrush = semrush_raw_sited.loc[semrush_raw_sited['keyword'].
isin(target_gap)]

cga_semrush
```

This results in the following:

	keyword	position	previous_position	search_volume	keyword_difficulty	cpc	url	traffic	traffic_%	traffic_cost	competition
42	hair dye ideas	8	7	2900	64	0.94	https://www.hair.com/dark-brown-hair-ideas.html	69	0.29	65.0	0.56
78	best hair color for pale skin	3	3	590	53	0.52	https://www.hair.com/hair-color-for-pale-skin.html	48	0.20	25.0	0.64
92	best hair color for pale skin and blue eyes	1	1	170	51	0.52	https://www.hair.com/best-hair-colors-blue-eyes.html	42	0.18	21.0	0.51
109	color stripes hair	11	11	1900	38	0.26	https://www.hair.com/skunk-stripe-hair.html	36	0.15	9.0	1.00
112	best hair color for blue eyes and fair skin	1	1	140	49	0.87	https://www.hair.com/best-hair-colors-blue-eyes.html	34	0.14	30.0	0.95
...	...	...	...	...	...	...	...	...	...	...	...

We only want the highest ranked target URLs per keyword, which we'll achieve with a combination of sort_values(), groupby(), and first():

```
cga_unique = cga_semrush.sort_values('position').groupby('keyword').
first().reset_index()
cga_unique['project'] = target_name
```

To make the dataframe more user-friendly, we'll prioritize keywords by

```
cga_unique = cga_unique.sort_values('search_volume', ascending = False)
```

Ready to export:

```
cga_unique.to_csv('exports/cga_unique.csv')
cga_unique
```

Now it's time to decide what content should be on these pages.

Content Creation: Planning Landing Page Content

Of course, now that you know which keywords belong together and which ones don't, and which keywords to pursue thanks to the content gap analysis, the question becomes what content should be on these pages?

One strategy we're pursuing is to

1. Look at the top 10 ranking URLs for each keyword
2. Extract the headings (<h1>, <h2>) from each ranking URL
3. Check the search results for each heading as writers can phrase the intent differently
4. Cluster the headings and label them
5. Count the frequency of the clustered headings for a given keyword, to see which ones are most popular and are being rewarded by Google (in terms of rankings)
6. Export the results for each search phrase

This strategy won't work for all verticals as there's a lot of noise in some market sectors compared to others. For example, with hair styling articles, a lot of the headings (and their sections) are celebrity names which will not have the same detectable search intent as another celebrity.

In contrast, in other verticals this method works really well because there aren't endless lists with the same HTML heading tags shared with related article titles (e.g., "Drew Barrymore" and "54 ways to wear the modern Marilyn").

Instead, the headings are fewer in number and have a meaning in common, for example, "What is account-based marketing?" and "Defining ABM," which is something Google is likely to understand.

With those caveats in mind, let's go.

```
import requests
from requests.exceptions import ReadTimeout
from json.decoder import JSONDecodeError
import re
import time
import random
import pandas as pd
import numpy as np
import datetime
import requests
import json
```

```
from datetime import timedelta
from glob import glob
import os
from client import RestClient
from textdistance import sorensen_dice
from plotnine import *
import matplotlib.pyplot as plt
from mizani.transforms import trans
from pandas.api.types import is_string_dtype
from pandas.api.types import is_numeric_dtype
import uritools
```

This is the website we're creating content for:

```
target = 'on24'
```

These are the keywords the target website wants to rank for. There's only eight keywords, but as you'll see, this process generates a lot of noisy data, which will need cleaning up:

```
queries = ['webinar best practices',
           'webinar marketing guide',
           'webinar guide',
           'funnel marketing guide',
           'scrappy marketing guide',
           'b2b marketing guide',
           'how to run virtual events',
           'webinar benchmarks']
```

Getting SERP Data

Import the SERP data which will form the basis of finding out what content is Google rewarding for the sites to rank in the top 10:

```
serps_input = pd.read_csv('data/serps_input_' + target + '.csv')
```

```
serps_input
```

This results in the following:

	keyword	rank	url	se_results_count	domain	title	is_video
0	funnel marketing guide	1	https://www.mageplaza.com/blog/marketing-funnel.html	14900000	www.mageplaza.com	What Is A Marketing Funnel? A Step-By-Step Guide - Mageplaza	None
1	funnel marketing guide	2	https://www.singlegrain.com/blog-posts/content-marketing/how-to-create-marketing-funnel/	14900000	www.singlegrain.com	How to Create a Powerful Marketing Funnel Step-by-Step	False
2	funnel marketing guide	3	None	14900000	None	None	None
3	funnel marketing guide	4	https://ahrefs.com/blog/marketing-funnels/	14900000	ahrefs.com	Marketing Funnels for Beginners: A Comprehensive Guide	False
4	funnel marketing guide	5	https://www.hotjar.com/blog/marketing-funnel/	14900000	www.hotjar.com	The Marketing Funnel: Stages, Strategies, & How to Optimize	False
...	...	...	...	...	...	...	...

The extract function from the TLD extract package is useful for extracting the hostname and domain name from URLs:

```
from tldextract import extract
```

```
serps_input_clean = serps_input.copy()
```

Set the URL column as a string:

```
serps_input_clean['url'] = serps_input_clean['url'].astype(str)
```

Use lambda to apply the extract function to the URL column:

```
serps_input_clean['host'] = serps_input_clean['url'].apply(lambda x:
extract(x))
```

Convert the function output (which is a tuple) to a list:

```
serps_input_clean['host'] = [list(lst) for lst in serps_input_
clean['host']]
```

Extract the hostname by taking the penultimate list element from the list using the string get method:

```
serps_input_clean['host'] = serps_input_clean['host'].str.get(-2)
```

The site uses a similar logic as before:

```
serps_input_clean['site'] = serps_input_clean['url'].apply(lambda x:
extract(x))
serps_input_clean['site'] = [list(lst) for lst in serps_input_
clean['site']]
```

Only this time, we want both the hostname and the top-level domain (TLD) which we will join to form the site or domain name:

```
serps_input_clean['site'] = serps_input_clean['site'].str.get(-2) + '.'
+serps_input_clean['site'].str.get(-1)
```

```
serps_input_clean
```

This results in the following:

eyword	rank	url	se_results_count	domain	title	is_video	host	site
funnel arketing guide	1	https://www.mageplaza.com/blog/marketing-funnel.html	14900000	www.mageplaza.com	What Is A Marketing Funnel? A Step-By-Step Guide - Mageplaza	NaN	mageplaza	mageplaza.com
funnel arketing guide	2	https://www.singlegrain.com/blog-posts/content-marketing/how-to-create-marketing-funnel/	14900000	www.singlegrain.com	How to Create a Powerful Marketing Funnel Step-by-Step	False	singlegrain	singlegrain.com
funnel arketing guide	3	nan	14900000	NaN	NaN	NaN	nan	nan.
funnel arketing guide	4	https://ahrefs.com/blog/marketing-funnels/	14900000	ahrefs.com	Marketing Funnels for Beginners: A Comprehensive Guide	False	ahrefs	ahrefs.com
funnel arketing guide	5	https://www.hotjar.com/blog/marketing-funnel/	14900000	www.hotjar.com	The Marketing Funnel: Stages, Strategies, & How to Optimize	False	hotjar	hotjar.com
...	...	...	...	...	...	...	...	...

The augmented dataframe shows the host and site columns added.

This line allows the column values to be read by setting the column widths to their maximum value:

```
pd.set_option('display.max_colwidth', None)
```

Crawling the Content

The next step is to get a list of top ranking URLs that we'll crawl for their content sections:

```
serps_to_crawl_df = serps_input_clean.copy()
```

There are some sites not worth crawling because they won't let you, which are defined in the following list:

```
dont_crawl = ['wikipedia', 'google', 'youtube', 'linkedin', 'foursquare',
'amazon', 'twitter', 'facebook', 'pinterest', 'tiktok', 'quora',
'reddit', 'None']
```

The dataframe is filtered to exclude sites in the don't crawl list:

```
serps_to_crawl_df = serps_to_crawl_df.loc[~serps_to_crawl_df['host'].
isin(dont_crawl)]
```

We'll also remove nulls and sites outside the top 10:

```
serps_to_crawl_df = serps_to_crawl_df.loc[~serps_to_crawl_df['domain'].
isnull()]
serps_to_crawl_df = serps_to_crawl_df.loc[serps_to_crawl_df['rank'] < 10]

serps_to_crawl_df.head(10)
```

This results in the following:

url	se_results_count	domain	title	is_video	host	site
https://www.mageplaza.com/blog/marketing-funnel.html	14900000	www.mageplaza.com	What Is A Marketing Funnel? A Step-By-Step Guide - Mageplaza	NaN	mageplaza	mageplaza.com
https://www.singlegrain.com/blog-posts/content-marketing/how-to-create-marketing-funnel/	14900000	www.singlegrain.com	How to Create a Powerful Marketing Funnel Step-by-Step	False	singlegrain	singlegrain.com
https://ahrefs.com/blog/marketing-funnels/	14900000	ahrefs.com	Marketing Funnels for Beginners: A Comprehensive Guide	False	ahrefs	ahrefs.com
https://www.hotjar.com/blog/marketing-funnel/	14900000	www.hotjar.com	The Marketing Funnel: Stages, Strategies, & How to Optimize	False	hotjar	hotjar.com
https://sproutsocial.com/insights/social-media-marketing-funnel/	14900000	sproutsocial.com	How to Build a Social Media Marketing Funnel That Converts	False	sproutsocial	sproutsocial.com

With the dataframe filtered, we just want the URLs to export to our desktop crawler.

Some URLs may rank for multiple search phrases. To avoid crawling the same URL multiple times, we'll use drop_duplicates() to make the URL list unique:

```
serps_to_crawl_upload = serps_to_crawl_df[['url']].drop_duplicates()
serps_to_crawl_upload.to_csv('data/serps_to_crawl_upload.csv', index=False)
```

```
serps_to_crawl_upload
```

This results in the following:

	url
0	https://www.mageplaza.com/blog/marketing-funnel.html
1	https://www.singlegrain.com/blog-posts/content-marketing/how-to-create-marketing-funnel/
3	https://ahrefs.com/blog/marketing-funnels/
4	https://www.hotjar.com/blog/marketing-funnel/
5	https://sproutsocial.com/insights/social-media-marketing-funnel/
...	...
719	https://www.netline.com/netline003h/?d=on24scrappymarketer&k=190815nlw24sm
721	https://blog.marketo.com/2016/08/get-scrappy-7-tips-for-smarter-digital-marketing.html
722	https://www.scootermediaco.com/2021/05/scrappy-marketing-strategies/
723	https://www.slideshare.net/kflanagan/the-scrappy-guide-to-marketing
724	https://www.bookdepository.com/Scrappy-Marketing-Handbook-Ann-Handley/9781118929636

62 rows × 1 columns

Now we have a list of 62 URLs to crawl, which cover the eight target keywords.

Let's import the results of the crawl:

```
crawl_raw = pd.read_csv('data/all_inlinks.csv')
pd.set_option('display.max_columns', None)
```

Using a list comprehension, we'll clean up the column names to make it easier to work with:

```
crawl_raw.columns = [col.lower().replace(' ', '_') for col in crawl_raw.
columns]
```

Print out the column names to see how many extractor fields were extracted:

```
print(crawl_raw.columns)
```

This results in the following:

```
Index(['type', 'source', 'destination', 'form_action_link', 'indexability',
       'indexability_status', 'hreflang', 'size_(bytes)', 'alt_text',
       'length',
       'anchor', 'status_code', 'status', 'follow', 'target', 'rel',
       'path_type', 'unlinked', 'link_path', 'link_position', 'link_
       origin',
       'extractor_1_1', 'extractor_1_2', 'extractor_1_3', 'extractor_1_4',
       'extractor_1_5', 'extractor_1_6', 'extractor_1_7', 'extractor_2_1',
       'extractor_2_2', 'extractor_2_3', 'extractor_2_4', 'extractor_2_5',
       'extractor_2_6', 'extractor_2_7', 'extractor_2_8', 'extractor_2_9',
       'extractor_2_10', 'extractor_2_11', 'extractor_2_12',
       'extractor_2_13',
       'extractor_2_14', 'extractor_2_15', 'extractor_2_16',
       'extractor_2_17',
       'extractor_2_18', 'extractor_2_19', 'extractor_2_20',
       'extractor_2_21',
       'extractor_2_22', 'extractor_2_23', 'extractor_2_24',
       'extractor_2_25',
       'extractor_2_26', 'extractor_2_27', 'extractor_2_28',
       'extractor_2_29',
       'extractor_2_30', 'extractor_2_31', 'extractor_2_32',
       'extractor_2_33',
       'extractor_2_34', 'extractor_2_35', 'extractor_2_36',
       'extractor_2_37',
       'extractor_2_38', 'extractor_2_39', 'extractor_2_40',
       'extractor_2_41',
       'extractor_2_42', 'extractor_2_43', 'extractor_2_44',
       'extractor_2_45',
       'extractor_2_46', 'extractor_2_47', 'extractor_2_48',
       'extractor_2_49',
```

```
        'extractor_2_50', 'extractor_2_51', 'extractor_2_52',
        'extractor_2_53',
        'extractor_2_54', 'extractor_2_55', 'extractor_2_56',
        'extractor_2_57',
        'extractor_2_58', 'extractor_2_59', 'extractor_2_60',
        'extractor_2_61',
        'extractor_2_62', 'extractor_2_63', 'extractor_2_64',
        'extractor_2_65'],
       dtype='object')
```

There are 6 primary headings (H1 in HTML) and 65 H2 headings altogether. These will form the basis of our content sections which tell us what content should be on those pages.

`crawl_raw`

This results in the following:

link_position	link_origin	extractor_1_1	extractor_1_2	extractor_1_3	extractor_1_4	extractor_1_5	extractor_1_6	extractor_1_7	extractor_2_1	extractor_2_2	ext
Header	HTML	The ultimate guide to creating engaging webinars.	NaN	NaN	NaN	NaN	NaN	NaN	Webinar guide contents	Presentations vs Webinars	
Header	HTML	The ultimate guide to creating engaging webinars.	NaN	NaN	NaN	NaN	NaN	NaN	Webinar guide contents	Presentations vs Webinars	
Header	HTML	The ultimate guide to creating engaging webinars.	NaN	NaN	NaN	NaN	NaN	NaN	Webinar guide contents	Presentations vs Webinars	
Content	HTML	The ultimate guide to creating engaging webinars.	NaN	NaN	NaN	NaN	NaN	NaN	Webinar guide contents	Presentations vs Webinars	
Content	HTML	The ultimate guide to creating engaging webinars.	NaN	NaN	NaN	NaN	NaN	NaN	Webinar guide contents	Presentations vs Webinars	
...	...	...	...	...	...	...	...	...	...	...	

Extracting the Headings

Since we're only interested in the content, we'll filter for it:

```
crawl_headings = crawl_raw.loc[crawl_raw['link_position'] ==
'Content'].copy()
```

The dataframe also contains columns that are superfluous to our requirements such as link_position and link_origin. We can remove these by listing the columns by position (saves space and typing out the names of which there are many!).

```
drop_cols = [0, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17,
18, 19, 20]
```

Using the .drop() method, we can drop multiple columns in place (i.e., without having to copy the result onto itself):

```
crawl_headings.drop(crawl_headings.columns[drop_cols], axis = 1,
inplace = True)
```

Rename the columns from source to URL, which will be useful for joining later:

```
crawl_headings = crawl_headings.rename(columns = {'source': 'url'})

crawl_headings
```

This results in the following:

	url	extractor_1_1	extractor_1_2	extractor_1_3	extractor_1_4	extractor_1_5	extractor_1_6	extractor_1_7	extra
3	https://buffalo7.co.uk/blog/webinar-guide/	The ultimate guide to creating engaging webinars.	NaN	NaN	NaN	NaN	NaN	NaN	Web
4	https://buffalo7.co.uk/blog/webinar-guide/	The ultimate guide to creating engaging webinars.	NaN	NaN	NaN	NaN	NaN	NaN	Web
5	https://buffalo7.co.uk/blog/webinar-guide/	The ultimate guide to creating engaging webinars.	NaN	NaN	NaN	NaN	NaN	NaN	Web
6	https://buffalo7.co.uk/blog/webinar-guide/	The ultimate guide to creating engaging webinars.	NaN	NaN	NaN	NaN	NaN	NaN	Web
7	https://buffalo7.co.uk/blog/webinar-guide/	The ultimate guide to creating engaging webinars.	NaN	NaN	NaN	NaN	NaN	NaN	Web
...	...	...	...	...	...	...	...	...	

With the desired columns of URL and their content section columns, these need to be converted to long format, where all of the sections will be in a single column called "heading":

```
crawl_headings_long = crawl_headings.copy()
```

We'll want a list of the extractor column names (again to save typing) by subsetting the dataframe from the second column onward using .iloc and extracting the column names (.columns.values):

```
heading_cols = crawl_headings_long.iloc[:, 1:].columns.values.tolist()
```

Using the .melt() function, we'll pivot the dataframe to reshape the content sections into a single column "heading" using the preceding list:

```
crawl_headings_long = pd.melt(crawl_headings_long, id_vars='url', value_
name = 'heading', var_name = 'position',
            value_vars= heading_cols)
```

Remove the null values:

```
crawl_headings_long = crawl_headings_long.loc[~crawl_headings_
long['heading'].isnull()]
```

Remove the duplicates:

```
crawl_headings_long = crawl_headings_long.drop_duplicates()
```

```
crawl_headings_long
```

This results in the following:

	url	position	heading
0	https://buffalo7.co.uk/blog/webinar-guide/	extractor_1_1	The ultimate guide to creating engaging webinars.
6	https://blog.hubspot.com/marketing/what-is-a-webinar	extractor_1_1	The Ultimate Guide to Creating Compelling Webinars
16	https://blog.hubspot.com/marketing/are-webinars-dead-how-to-make-a-webinar	extractor_1_1	The Ultimate Guide to Creating Compelling Webinars
18	https://blog.hubspot.com/blog/tabid/6307/bid/2391/10-best-practices-for-webinars-or-webcasts.aspx	extractor_1_1	The Ultimate Guide to Creating Compelling Webinars
20	https://www.creativebloq.com/advice/virtual-event-tips	extractor_1_1	How to host a virtual event: 10 expert tips
...	...	...	...
16179	https://surveysparrow.com/blog/how-to-conduct-a-webinar-guide/	extractor_2_61	Company
16420	https://surveysparrow.com/blog/how-to-conduct-a-webinar-guide/	extractor_2_62	Resources
16661	https://surveysparrow.com/blog/how-to-conduct-a-webinar-guide/	extractor_2_63	Free Tools
16902	https://surveysparrow.com/blog/how-to-conduct-a-webinar-guide/	extractor_2_64	Sales
17143	https://surveysparrow.com/blog/how-to-conduct-a-webinar-guide/	extractor_2_65	Connect

647 rows × 3 columns

The resulting dataframe shows the URL, the heading, and the position where the first number denotes whether it was an h1 or h2 and the second number indicates the order of the heading on the page. The heading is the text value.

You may observe that the heading contains some values that are not strictly content but boilerplate content that is sitewide, such as Company, Resources, etc. These will require removal at some point.

```
serps_headings = serps_to_crawl_df.copy()
```

Let's join the headings to the SERPs data:

```
serps_headings = serps_headings.merge(crawl_headings_long, on = 'url',
how = 'left')
```

Replace null headings with '' so that these can be aggregated:

```
serps_headings['heading'] = np.where(serps_headings['heading'].isnull(),
'', serps_headings['heading'])

serps_headings['project'] = 'target'

serps_headings
```

This results in the following:

url	se_results_count	domain	title	is_video	host	site	position	heading	project
plaza.com/blog/marketing-funnel.html	14900000	www.mageplaza.com	What Is A Marketing Funnel? A Step-By-Step Guide - Mageplaza	NaN	mageplaza	mageplaza.com	extractor_1_1	What Is A Marketing Funnel? A Step-By-Step Guide!	target
plaza.com/blog/marketing-funnel.html	14900000	www.mageplaza.com	What Is A Marketing Funnel? A Step-By-Step Guide - Mageplaza	NaN	mageplaza	mageplaza.com	extractor_2_1	What is Marketing funnel?	target
plaza.com/blog/marketing-funnel.html	14900000	www.mageplaza.com	What Is A Marketing Funnel? A Step-By-Step Guide - Mageplaza	NaN	mageplaza	mageplaza.com	extractor_2_2	Understanding the stages of a marketing funnel	target
plaza.com/blog/marketing-funnel.html	14900000	www.mageplaza.com	What Is A Marketing Funnel? A Step-By-Step Guide - Mageplaza	NaN	mageplaza	mageplaza.com	extractor_2_3	Marketing and sales funnel: What's the difference?	target
plaza.com/blog/marketing-funnel.html	14900000	www.mageplaza.com	What Is A Marketing Funnel? A Step-By-Step Guide - Mageplaza	NaN	mageplaza	mageplaza.com	extractor_2_4	Do you need a Marketing funnel?	target
...	...	...	...	...	...	...	...	...	...

With the data joined, we'll take the domain, heading, and the position:

```
headings_tosum = serps_headings[['domain', 'heading', 'position']].copy()
```

Split position by underscore and extract the last number in the list (using -1) to get the order the heading appears on the page:

```
headings_tosum['pos_n'] = headings_tosum['position'].str.split('_').str[-1]
```

Convert the data type into a number:

```
headings_tosum['pos_n'] = headings_tosum['pos_n'].astype(float)
```

Add a count column for easy aggregation:

```
headings_tosum['count'] = 1
headings_tosum
```

This results in the following:

	domain	heading	position	pos_n	count
0	www.mageplaza.com	What Is A Marketing Funnel? A Step-By-Step Guide!	extractor_1_1	1.0	1
1	www.mageplaza.com	What is Marketing funnel?	extractor_2_1	1.0	1
2	www.mageplaza.com	Understanding the stages of a marketing funnel	extractor_2_2	2.0	1
3	www.mageplaza.com	Marketing and sales funnel: What's the difference?	extractor_2_3	3.0	1
4	www.mageplaza.com	Do you need a Marketing funnel?	extractor_2_4	4.0	1
...	...	...	...	...	...
674	www.bookdepository.com	Books By Language	extractor_2_9	9.0	1
675	www.bookdepository.com	Description	extractor_2_10	10.0	1
676	www.bookdepository.com	\n Product details	extractor_2_11	11.0	1
677	www.bookdepository.com	People who viewed this also viewed	extractor_2_12	12.0	1
678	www.bookdepository.com	Bestsellers in Sales & Marketing Management	extractor_2_13	13.0	1

679 rows × 5 columns

Cleaning and Selecting Headings

We're ready to aggregate and start removing nonsense headings.

We'll start by removing boilerplate headings that are particular to each site. This is achieved by summing the number of times a heading appears by domain and removing any that appear more than once as that will theoretically mean the heading is not unique.

```
domsheadings_tosum_agg = headings_tosum.groupby(['domain', 'heading']).
agg({'count': sum,
'pos_n': 'mean'
          }).reset_index().sort_values(['domain', 'count'],
          ascending = False)
domsheadings_tosum_agg['heading'] = domsheadings_tosum_agg['heading'].
str.lower()
domsheadings_tosum_agg.head(50)
```

Stop headings is a list containing headings that we want to remove.

Include those that appear more than once:

```
stop_headings = domsheadings_tosum_agg.loc[domsheadings_tosum_
agg['count'] > 1]
```

and contain line break characters like "\n":

```
stop_headings = stop_headings.loc[stop_headings['heading'].str.
contains('\n')]
stop_headings = stop_headings['heading'].tolist()

stop_headings
```

This results in the following:

```
['\n  \n    the scrappy guide to marketing\n  \n',
 '\n                \n                    danny goodwin
\n            ',
 '\n         \n             how to forecast seo with better precision &
transparency         \n     ',
 '\n         \n             should you switch to ga4 now? what you need to
know         \n     ',
 '\n            the ultimate guide to webinars: 41 tips for successful
webinars        ',
 '\n        \n        \n            \n            \n                \n
\n            \n            \n            \n               \n get timely
updates and fresh ideas delivered to your inbox. \n                \n
\n                \n                \n            \n            \n
\n        \n    ',
 '4 best webinar practices for marketing and promotion in 2020\n',
 '\n    company\n  ',
 '\n    customers\n  ',
 '\n    free tools\n  ',
 '\n    partners\n  ',
 '\n    popular features\n  ']
```

The list of boilerplate has been reasonably successful on a domain level, but there is more work to do.

We'll now analyze the headings per se, starting by counting the number of headings:

```
headings_tosum_agg = headings_tosum.groupby(['heading']).agg({'count': sum,
'pos_n': 'mean'
          }).reset_index().sort_values('count',
          ascending = False)
headings_tosum_agg['heading'] = headings_tosum_agg['heading'].str.lower()
```

Remove the headings containing the boilerplate items:

```
headings_tosum_agg = headings_tosum_agg.loc[~headings_tosum_agg['heading'].
isin(stop_headings)]
```

Subset away from headings containing nothing (''):

```
headings_tosum_agg = headings_tosum_agg.loc[headings_tosum_
agg['heading'] != '']

headings_tosum_agg.head(10)
```

This results in the following:

	heading	count	pos_n
195	company	4	25.000000
467	webinar marketing strategy	3	2.000000
281	how to record a webinar	3	5.000000
161	b2b marketing examples	3	5.666667
507	what is a webinar?	3	1.000000
163	b2b marketing strategies	3	5.333333
494	what is b2b marketing?	3	1.000000
476	webinar statistics	3	5.000000
263	how does a webinar work?	3	4.000000
273	how to create a webinar	3	1.000000

The dataframe looks to contain more sensible content headings with the exception of "company," which also is much further down the order of the page at 25.

Let's filter further:

```
headings_tosum_filtered = headings_tosum_agg.copy()
```

Remove headings with a position of 10 or above as these are unlikely to contain actual content sections. Note 10 is an arbitrary number and could be more or less depending on the nature of content.

```
headings_tosum_filtered = headings_tosum_filtered.loc[headings_tosum_
filtered['count'] < 10 ]
```

Measure the number of words in the heading:

```
headings_tosum_filtered['tokens'] = headings_tosum_filtered['heading'].str.
count(' ') + 1
```

Clean up the headings by removing spaces on either side of the text:

```
headings_tosum_filtered['heading'] = headings_tosum_filtered['heading'].
str.strip()
```

Split heading using colons as a punctuation mark and extract the right-hand side of the colon:

```
headings_tosum_filtered['heading'] =  headings_tosum_filtered['heading'].
str.split(':').str[-1]
```

Apply the same principle to the full stop:

```
headings_tosum_filtered['heading'] =  headings_tosum_filtered['heading'].
str.split('.').str[-1]
```

Remove headings containing pagination, for example, 1 of 9:

```
headings_tosum_filtered = headings_tosum_filtered.loc[~headings_tosum_
filtered['heading'].str.contains('[0-9] of [0-9]', regex = True)]
```

Remove headings that are less than 5 words long or more than 12:

```
headings_tosum_filtered = headings_tosum_filtered.loc[headings_tosum_
filtered['tokens'].between(5, 12)]
headings_tosum_filtered = headings_tosum_filtered.sort_values('count',
ascending = False)
```

```
headings_tosum_filtered = headings_tosum_filtered.loc[headings_tosum_
filtered['heading'] != '' ]

headings_tosum_filtered.head(10)
```

This results in the following:

	heading	count	pos_n	tokens
281	how to record a webinar	3	5.0	5
273	how to create a webinar	3	1.0	5
274	how to create an amazing webinar in 2022	3	1.0	8
276	how to host a webinar for free	3	7.0	7
409	the ultimate guide to creating compelling webinars	3	1.0	7
356	reach your target audience with webinars	3	8.0	6
263	how does a webinar work?	3	4.0	5
166	b2b marketing trends to watch in 2022 [new data]	2	4.0	9
509	what is scrappy marketing – and is it an answer?	2	1.0	10
499	what is scrappy marketing and why is it beneficial?	2	1.0	9

Now we have headings that look more like actual content sections. These are now ready for clustering.

Cluster Headings

The reason for clustering is that writers will describe the same section heading using different words and deliberately so as to avoid copyright infringement and plagiarism. However, Google is smart enough to know that "webinar best practices" and "best practices for webinars" are the same.

To make use of Google's knowledge, we'll make use of the SERPs to see if the search results of each heading are similar enough to know if they mean the same thing or not (i.e., whether the underlying meaning or intent is the same).

We'll create a list and use the search intent clustering code (see Chapter 2) to categorize the headings into topics:

```
headings_to_cluster = headings_tosum_filtered[['heading']].drop_
duplicates()
headings_to_cluster = headings_to_cluster.loc[~headings_to_
cluster['heading'].isnull()]
headings_to_cluster = headings_to_cluster.rename(columns = {'heading':
'keyword'})

headings_to_cluster
```

This results in the following:

	keyword
281	how to record a webinar
273	how to create a webinar
274	how to create an amazing webinar in 2022
276	how to host a webinar for free
409	the ultimate guide to creating compelling webinars
...	...
402	the difference between sales and marketing
401	the complete guide to virtual events in 2022
400	how to host a virtual event
429	understanding the difference between b2b and b2c marketing
280	how to protect your virtual events from cyberattacks

206 rows × 1 columns

With the headings clustered by search intent, we'll import the results:

```
topic_keyw_map = pd.read_csv('data/topic_keyw_map.csv')
```

Let's rename the keyword column to heading, which we can use to join to the SERP dataframe later:

```
topic_keyw_map = topic_keyw_map.rename(columns = {'keyword': 'heading'})

topic_keyw_map
```

This results in the following:

	topic	heading	topic_results
0	what is a virtual event	what is a virtual event?	6130000000
1	what is a virtual event	what is a virtual event	6130000000
2	how to create your own effective webinar	how to create your own effective webinar	2384900000
3	how to create your own effective webinar	how to design a webinar	2384900000
4	how to create your own effective webinar	how to create an amazing webinar in 2022	2384900000
5	how to create your own effective webinar	is creating a webinar right for you?	2384900000
6	how to create your own effective webinar	how to create a webinar	2384900000
7	how to create your own effective webinar	what's the best time to host a webinar?	2384900000
8	how to create your own effective webinar	how to create your webinar content	2384900000
9	the complete guide to virtual events in 2022	in-person or virtual - the fundamentals matter	1863000000

The dataframe shows the heading and the meaning of the heading as "topic." The next stage is to get some statistics and see how many headings constitute a topic. As the topics are the central meaning of the headings, this will form the core content sections per target keyword.

```
topic_keyw_map_agg = topic_keyw_map.copy()
topic_keyw_map_agg['count'] = 1
topic_keyw_map_agg = topic_keyw_map_agg.groupby('topic').agg({'count':
'sum'}).reset_index()
topic_keyw_map_agg = topic_keyw_map_agg.sort_values('count',
ascending = False)

topic_keyw_map_agg
```

This results in the following:

	topic	count
7	how to create your own effective webinar	7
17	what is scrappy marketing and why is it beneficial?	4
14	webinar attendance facts and statistic	4
3	building your webinar using virtual event software	4
8	how to optimize your marketing funnel for the customer journey	3
0	5 best webinar presentation design practices for 2020	3
5	how to conduct a webinar – the ultimate guide	3
2	b2b marketing the ultimate guide to b2b marketing	3
6	how to create a powerful marketing funnel?	2
4	how does this apply to webinars?	2
1	8 tactics for your b2b marketing strategy	2
10	the essential guide to webinar marketing for 2022 [with best practices]	2
11	understanding the stages of a marketing funnel	2
12	use webinar best practices to host a great webinar	2
13	webinar / webcast best practices	2
15	what is a social media marketing funnel?	2
16	what is a virtual event	2
9	the complete guide to virtual events in 2022	2

"Creating effective webinars" was the most popular content section.

These will now be merged with the SERPs so we can map suggested content to target keywords:

```
serps_topics_merge = serps_headings.copy()
```

For a successful merge, we'll require the heading to be in lowercase:

```
serps_topics_merge['heading'] = serps_topics_merge['heading'].str.lower()
```

```
serps_topics_merge = serps_topics_merge.merge(topic_keyw_map, on =
'heading', how = 'left')

serps_topics_merge
```

This results in the following:

_count	domain	title	is_video	host	site	position	heading	project	topic	topic_results	count
900000	www.mageplaza.com	What Is A Marketing Funnel? A Step-By-Step Guide - Mageplaza	NaN	mageplaza	mageplaza.com	extractor_1_1	what is a marketing funnel? a step-by-step guide!	target	NaN	NaN	1
900000	www.mageplaza.com	What Is A Marketing Funnel? A Step-By-Step Guide - Mageplaza	NaN	mageplaza	mageplaza.com	extractor_2_1	what is marketing funnel?	target	NaN	NaN	1
900000	www.mageplaza.com	What Is A Marketing Funnel? A Step-By-Step Guide - Mageplaza	NaN	mageplaza	mageplaza.com	extractor_2_2	understanding the stages of a marketing funnel	target	understanding the stages of a marketing funnel	33890000.0	1
900000	www.mageplaza.com	What Is A Marketing Funnel? A Step-By-Step Guide - Mageplaza	NaN	mageplaza	mageplaza.com	extractor_2_3	marketing and sales funnel: what's the difference?	target	NaN	NaN	1
900000	www.mageplaza.com	What Is A Marketing Funnel? A Step-By-Step Guide - Mageplaza	NaN	mageplaza	mageplaza.com	extractor_2_4	do you need a marketing funnel?	target	NaN	NaN	1
...	...	...	...	...	...	...	...	...	...	...	...

```
keyword_topics_summary = serps_topics_merge.groupby(['keyword', 'topic']).
agg({'count': 'sum'}).reset_index().sort_values(['keyword', 'count'],
ascending = False)
```

The count will be reset to 1, so we can count the number of suggested content sections per target keyword:

```
keyword_topics_summary['count'] = 1

keyword_topics_summary
```

This results in the following:

	keyword	topic	count
24	webinar marketing guide	how to create your own effective webinar	1
25	webinar marketing guide	the essential guide to webinar marketing for 2022 [with best practices]	1
22	webinar guide	how to conduct a webinar – the ultimate guide	1
23	webinar guide	how to create your own effective webinar	1
20	webinar guide	building your webinar using virtual event software	1
21	webinar guide	how does this apply to webinars?	1
17	webinar best practices	how to create your own effective webinar	1
13	webinar best practices	5 best webinar presentation design practices for 2020	1
15	webinar best practices	how does this apply to webinars?	1
19	webinar best practices	webinar / webcast best practices	1
14	webinar best practices	b2b marketing the ultimate guide to b2b marketing	1
16	webinar best practices	how to conduct a webinar – the ultimate guide	1
18	webinar best practices	use webinar best practices to host a great webinar	1
12	webinar benchmarks	webinar attendance facts and statistic	1
11	webinar benchmarks	use webinar best practices to host a great webinar	1
10	scrappy marketing guide	what is scrappy marketing and why is it beneficial?	1
9	scrappy marketing guide	how does this apply to webinars?	1
6	how to run virtual events	building your webinar using virtual event software	1
8	how to run virtual events	what is a virtual event	1
7	how to run virtual events	the complete guide to virtual events in 2022	1
3	funnel marketing guide	how to optimize your marketing funnel for the customer journey	1
2	funnel marketing guide	how to create a powerful marketing funnel?	1
4	funnel marketing guide	understanding the stages of a marketing funnel	1
5	funnel marketing guide	what is a social media marketing funnel?	1
0	b2b marketing guide	8 tactics for your b2b marketing strategy	1
1	b2b marketing guide	b2b marketing the ultimate guide to b2b marketing	1

The preceding dataframe shows the content sections (topic) that should be written for each target keyword.

```
keyword_topics_summary.groupby(['keyword']).agg({'count': 'sum'}).
reset_index()
```

This results in the following:

	keyword	count
0	b2b marketing guide	2
1	funnel marketing guide	4
2	how to run virtual events	3
3	scrappy marketing guide	2
4	webinar benchmarks	2
5	webinar best practices	7
6	webinar guide	4
7	webinar marketing guide	2

Webinar best practices will have the most content, while other target keywords will have around two core content sections on average.

Reflections

For B2B marketing, it works really well as it's a good way of automating a manual process most SEOs go through (i.e., seeing what content the top 10 ranking pages cover) especially when you have a lot of keywords to create content for.

We used the H1 and H2 because using even more copy from the body (such as H3 or <p> paragraphs even after filtering out stop words) would introduce more noise into the string distance calculations.

Sometimes, you get some reliable suggestions that are actually quite good; however, the output should be reviewed first before raising content requests from your creative team or agency.

Summary

There are many aspects of SEO that go into delivering content and UX better than your competitors. This chapter focused on

- *Keyword mapping*: Assigning keywords to existing content and identifying opportunities for new content creation
- *Content gap analysis*: Identifying critical content and the gaps in your website
- *Content creation*: Finding the core content common to top ranking articles for your target search phrases

The next chapter deals with the third major pillar of SEO: authority.

CHAPTER 5

Authority

Authority is arguably 50% of the Google algorithm. You could optimize your site to your heart's content by creating the perfect content and deliver it with the perfect UX that's hosted on a site with the most perfect information architecture, only to find it's nowhere in Google's search results when searching by the title of the page – assuming it's not a unique search phrase, so what gives?

You'll find out about this and the following in this chapter:

- What site authority is and how it impacts SEO
- How brand searches could impact search visibility
- Review single and multiple site analysis

Some SEO History

To answer the question, one must appreciate the evolution of search engines and just how wild things were before Google came along in 1998. And even when Google did come along, things were still wild and evolving quickly.

Before Google, most of the search engines like AltaVista, Yahoo!, and Ask (Jeeves) were primarily focused on the keywords embedded within the content on the page. This made search engines relatively easy to game using all kinds of tricks including hiding keywords in white text on white backgrounds or substantial repetition of keywords.

When Google arrived, they did a couple of things differently, which essentially turned competing search engines on their heads.

A. Voniatis, *Data-Driven SEO with Python*, https://doi.org/10.1007/978-1-4842-9175-7_5

The first thing is that their algorithm ranked pages based on their authority, in other words, how trustworthy the document (or website) was, as opposed to only matching a document on keyword relevance. Authority in those days was measured by Google as the amount of links from other sites linking to your site. This was much in the same way as citations in a doctoral dissertation. The more links (or citations), the higher the probability a random surfer on the Web would find your content. This made SEO harder to game and the results (temporarily yet significantly) more reliable relative to the competition.

The second thing they did was partner with Yahoo! which openly credited Google for powering their search results. So what happened next? Instead of using Yahoo!, people went straight to Google, bypassing the intermediary Yahoo! Search engine, and the rest is history – or not quite.

A Little More History

Although Google got the lion's share of searches, the SEO industry worked out the gist of Google's algorithm and started engineering link popularity schemes such as swapping links (known as reciprocal linking) and creating/renting links from private networks (still alive and well today, unfortunately). Google responded with antispam algorithms, such as Panda and Penguin, which more or less decimated these schemes to the point that most businesses in the brand space resorted to advertising and digital PR. And it works.

Authority, Links, and Other

While there is a widespread confusion in that back links are authority. We've seen plenty of evidence to show that authority is the effect of links and advertising, that is, authority is not only measured in links. Refer to Figure 5-1.

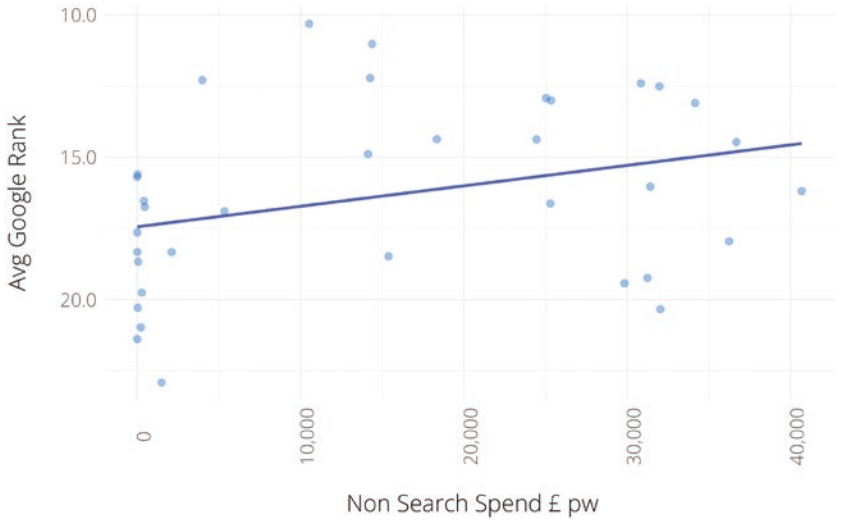

Figure 5-1. *Positive relationship between rankings and authority*

Figure 5-1 is just one example of many showing a positive relationship between rankings and authority. In this case, the authority is the product of nonsearch advertising. And why is that? It's because good links and effective advertising drive brand impressions, which are also positively linked.

What we will set out to do is show how data science can help you:

- Examine your own links
- Analyze your competitor's links
- Find power networks
- Determine the key ingredients for a good link

Examining Your Own Links

If you've ever wanted to analyze your site's backlinks, the chances are you'd use one of the more popular tools like AHREFs and SEMRush. These services trawl the Web to get a list of sites linking to your website with a domain rating and other info describing the quality of your backlinks, which they store in vast indexes which can be queried.

It's no secret that backlinks play a big part in Google's algorithm so it makes sense as a minimum to understand your own site before comparing it with the competition, of which the former is what we will do today.

While most of the analysis can be done on a spreadsheet, Python has certain advantages. Other than the sheer number of rows it can handle, it can also look at the statistical side more readily such as distributions.

Importing and Cleaning the Target Link Data

We're going to pick a small website from the UK furniture sector (for no particular reason) and walk through some basic analysis using Python.

So what is the value of a site's backlinks for SEO? At its simplest, I'd say quality and quantity. Quality is subjective to the expert yet definitive to Google by way of metrics such as authority and content relevance.

We'll start by evaluating the link quality with the available data before evaluating the quantity. Time to code.

```
import re
import time
import random
import pandas as pd
import numpy as np
import datetime
from datetime import timedelta
from plotnine import *
import matplotlib.pyplot as plt
from pandas.api.types import is_string_dtype
from pandas.api.types import is_numeric_dtype
import uritools

pd.set_option('display.max_colwidth', None)
%matplotlib inline

root_domain = 'johnsankey.co.uk'
hostdomain = 'www.johnsankey.co.uk'
hostname = 'johnsankey'
full_domain = 'https://www.johnsankey.co.uk'
target_name = 'John Sankey'
```

We start by importing the data and cleaning up the column names to make it easier to handle and quicker to type, for the later stages.

```
target_ahrefs_raw = pd.read_csv(
    'data/johnsankey.co.uk-refdomains-subdomains__2022-03-18_15-15-47.csv')
```

List comprehensions are a powerful and less intensive way to clean up the column names.

```
target_ahrefs_raw.columns = [col.lower() for col in target_ahrefs_raw.
columns]
```

The list comprehension instructs Python to convert the column name to lowercase for each column ("col") in the dataframe columns.

```
target_ahrefs_raw.columns = [col.replace(' ','_') for col in target_ahrefs_
raw.columns]
target_ahrefs_raw.columns = [col.replace('.','_') for col in target_ahrefs_
raw.columns]
target_ahrefs_raw.columns = [col.replace('__','_') for col in target_
ahrefs_raw.columns]
target_ahrefs_raw.columns = [col.replace('(','') for col in target_ahrefs_
raw.columns]
target_ahrefs_raw.columns = [col.replace(')','') for col in target_ahrefs_
raw.columns]
target_ahrefs_raw.columns = [col.replace('%','') for col in target_ahrefs_
raw.columns]
```

An alternative to repeating the preceding lines of code would be to chain the function calls to process the columns in a single line:

```
target_ahrefs_raw.columns = [col.lower().replace(' ','_').replace('.','_').
replace('__','_').replace('(','').replace(')','').replace('%','') for col
in target_ahrefs_raw.columns]
```

Though not strictly necessary, I like having a count column as standard for aggregations and a single value column "project" should I need to group the entire table:

```
target_ahrefs_raw['rd_count'] = 1
target_ahrefs_raw['project'] = target_name
Target_ahrefs_raw
```

This results in the following:

	domain	dr	dofollow_ref_domains	dofollow_linked_domains	traffic_	links_to_target	new_links	lost_links	dofollow_links
0	dribbble.com	93.0	598872	14388	3460486	1	0	0	0
1	msn.com	92.0	748243	265009	111799543	9	0	0	0
2	thetimes.co.uk	91.0	448410	28603	4977287	1	0	0	1
3	ow.ly	90.0	118913	2	42910	2	2	0	2
4	10times.com	78.0	19109	19346	522290	2	0	0	0
...	...	...	...	...	...	...	...	...	...
102	ikhatotherescue.blogspot.com	0.0	73	2314	0	1	0	0	1
103	peakedgehotel.blogspot.com	0.0	0	0	0	8	0	0	8
104	thelibertyscale.blogspot.com	0.0	0	0	1	1	0	0	1
105	plums-rhombus-lrg3.squarespace.com	0.0	0	0	0	1	1	1	1
106	upholsterycleaningdozaowa.blogspot.com	0.0	0	0	0	1	0	0	1

107 rows × 13 columns

Now we have a dataframe with clean column names. The next step is to clean the actual table values and make them more useful for analysis.

Make a copy of the previous dataframe and give it a new name:

```
target_ahrefs_clean_dtypes = target_ahrefs_raw.copy()
```

Clean the dofollow_ref_domains column which tells us how many ref domains the sitelinking has. In this case, we'll convert the dashes to zeros and then cast the whole column as a whole number.

Start with referring domains:

```
target_ahrefs_clean_dtypes['dofollow_ref_domains'] = np.where(target_
ahrefs_clean_dtypes['dofollow_ref_domains'] == '-',
                        0, target_ahrefs_clean_dtypes['dofollow_ref_
                        domains'])
target_ahrefs_clean_dtypes['dofollow_ref_domains'] = target_ahrefs_clean_
dtypes['dofollow_ref_domains'].astype(int)
```

then linked domains:

```
target_ahrefs_clean_dtypes['dofollow_linked_domains'] = np.where(target_
ahrefs_clean_dtypes['dofollow_linked_domains'] == '-',
                    0, target_ahrefs_clean_dtypes['dofollow_linked_
                    domains'])
target_ahrefs_clean_dtypes['dofollow_linked_domains'] = target_ahrefs_
clean_dtypes['dofollow_linked_domains'].astype(int)
```

"First seen" tells us the date when the link was first found (i.e., discovered and then added to the index of ahrefs). We'll convert the string to a date format that Python can process and then use this to derive the age of the links later on:

```
target_ahrefs_clean_dtypes['first_seen'] = pd.to_datetime(target_ahrefs_
clean_dtypes['first_seen'], format='%d/%m/%Y %H:%M')
```

Converting first_seen to a date also means we can perform time aggregations by month year, as it's not always the case that links for a site will get acquired on a daily basis:

```
target_ahrefs_clean_dtypes['month_year'] = target_ahrefs_clean_
dtypes['first_seen'].dt.to_period('M')
```

The link age is calculated by taking today's date and subtracting the first seen date. Then it's converted to a number format and divided by a huge number to get the number of days:

```
target_ahrefs_clean_dtypes['link_age'] = dt.datetime.now() - target_ahrefs_
clean_dtypes['first_seen']
target_ahrefs_clean_dtypes['link_age'] = target_ahrefs_clean_
dtypes['link_age']
target_ahrefs_clean_dtypes['link_age'] = target_ahrefs_clean_dtypes
['link_age'].astype(int)
target_ahrefs_clean_dtypes['link_age'] = (target_ahrefs_clean_dtypes
['link_age']/(3600 * 24 * 1000000000)).round(0)

target_ahrefs_clean_dtypes
```

This results in the following:

lomains	dofollow_linked_domains	traffic_	links_to_target	new_links	lost_links	dofollow_links	first_seen	lost	rd_count	project	month_year	link_age
598872	14388	3460486	1	0	0	0	2021-04-16 03:06:00	NaN	1	John Sankey	2021-04	403.0
748243	265009	111799543	9	0	0	0	2021-09-28 11:39:00	NaN	1	John Sankey	2021-09	238.0
448410	28603	4977287	1	0	0	1	2017-09-17 07:13:00	NaN	1	John Sankey	2017-09	1710.0
118913	2	42910	2	2	0	2	2022-02-16 21:31:00	NaN	1	John Sankey	2022-02	97.0
19109	19346	522290	2	0	0	0	2018-10-06 01:03:00	NaN	1	John Sankey	2018-10	1326.0
...	...	...	...	...	...	...	...	...	...	...	...	...
73	2314	0	1	0	0	1	2021-11-18 05:37:00	NaN	1	John Sankey	2021-11	187.0
0	0	0	8	0	0	8	2020-06-17 23:30:00	NaN	1	John Sankey	2020-06	705.0
0	0	1	1	0	0	1	2021-04-11 06:21:00	NaN	1	John Sankey	2021-04	408.0
0	0	0	1	1	1	1	2022-03-14 11:18:00	16/03/2022 16:51	1	John Sankey	2022-03	71.0
0	0	0	1	0	0	1	2019-04-04 13:29:00	NaN	1	John Sankey	2019-04	1146.0

With the data types cleaned, and some new data features created (note columns added earlier), the fun can begin.

Targeting Domain Authority

The first part of our analysis evaluates the link quality, which starts by summarizing the whole dataframe using the describe function to get descriptive statistics of all the columns:

```
target_ahrefs_analysis = target_ahrefs_clean_dtypes
target_ahrefs_analysis.describe()
```

This results in the following:

	dr	dofollow_ref_domains	dofollow_linked_domains	traffic_	links_to_target	new_links	lost_links	dofollow_links	rd_count	link_age
count	107.000000	107.000000	1.070000e+02	1.070000e+02	107.000000	107.000000	107.000000	107.000000	107.0	107.000000
mean	26.593458	22557.794393	3.661365e+05	1.334358e+06	3.383178	0.644860	0.093458	2.411215	1.0	492.121495
std	28.092862	103659.472991	1.477562e+06	1.088568e+07	6.180683	3.629679	0.445792	5.606521	0.0	555.336286
min	0.000000	0.000000	0.000000e+00	0.000000e+00	1.000000	0.000000	0.000000	0.000000	1.0	6.000000
25%	0.100000	29.000000	2.700000e+01	0.000000e+00	1.000000	0.000000	0.000000	1.000000	1.0	123.500000
50%	16.000000	184.000000	1.373000e+03	1.380000e+02	2.000000	0.000000	0.000000	1.000000	1.0	234.000000
75%	47.500000	1973.000000	7.001000e+03	7.329500e+03	2.500000	0.000000	0.000000	2.000000	1.0	645.500000
max	93.000000	748243.000000	7.923115e+06	1.117995e+08	42.000000	34.000000	4.000000	42.000000	1.0	2504.000000

So from the preceding table, we can see the average (mean), the number of referring domains (107), and the variation (the 25th percentiles and so on).

The average domain rating (equivalent to Moz's Domain Authority) of referring domains is 27. Is that a good thing? In the absence of competitor data to compare in this market sector, it's hard to know, which is where your experience as an SEO practitioner comes in. However, I'm certain we could all agree that it could be much higher – given that it falls on a scale between 0 and 100. How much higher to make a shift is another question.

The preceding table can be a bit dry and hard to visualize, so we'll plot a histogram to get more of an intuitive understanding of the referring domain authority:

```
dr_dist_plt = (
    ggplot(target_ahrefs_analysis,
           aes(x = 'dr')) +
    geom_histogram(alpha = 0.6, fill = 'blue', bins = 100) +
    scale_y_continuous() +
    theme(legend_position = 'right'))

dr_dist_plt
```

The distribution is heavily skewed, showing that most of the referring domains have an authority rating of zero (Figure 5-2). Beyond zero, the distribution looks fairly uniform with an equal amount of domains across different levels of authority.

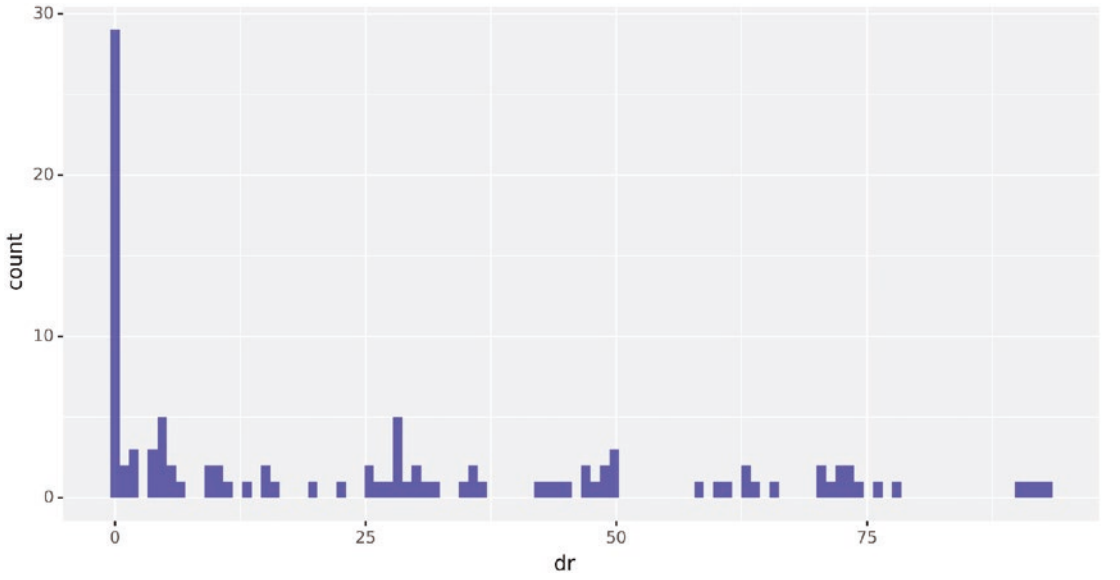

Figure 5-2. *Distribution of domain rating in the backlink profile*

Domain Authority Over Time

We'll now look at the domain authority as a proxy for the link quality as a time series. If we were to plot the number of links by date, the time series would look rather messy and less useful as follows:

```
dr_firstseen_plt = (
    ggplot(target_ahrefs_analysis, aes(x = 'first_seen', y = 'dr',
    group = 1)) +
    geom_line(alpha = 0.6, colour = 'blue', size = 2) +
    labs(y = 'Domain Rating', x = 'Month Year') +
    scale_y_continuous() +
    scale_x_date() +
    theme(legend_position = 'right',
          axis_text_x=element_text(rotation=90, hjust=1)
         )
)
```

```
dr_firstseen_plt.save(filename = 'images/1_dr_firstseen_plt.png',
                      height=5, width=10, units = 'in', dpi=1000)

dr_firstseen_plt
```

The plot looks very noisy as you'd expect and only really shows you what the DR (domain rating) of a referring domain was at a point in time (Figure 5-3). The utility of this chart is that if you have a team tasked with acquiring links, you can monitor the link quality over time in general.

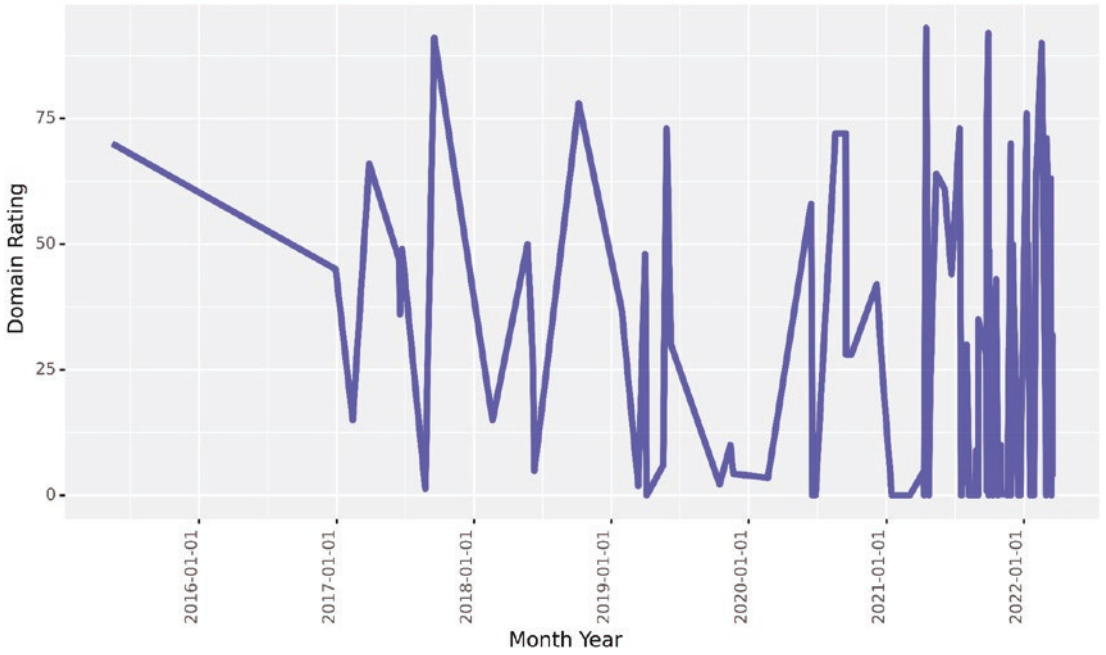

Figure 5-3. *Backlink domain rating acquired over time*

For a more smoother view:

```
dr_firstseen_smooth_plt = (
    ggplot(target_ahrefs_analysis, aes(x = 'first_seen', y = 'dr',
    group = 1)) +
    geom_smooth(alpha = 0.6, colour = 'blue', size = 3, se = False) +
    labs(y = 'Domain Rating', x = 'Month Year') +
    scale_y_continuous() +
    scale_x_date() +
```

```
    theme(legend_position = 'right',
          axis_text_x=element_text(rotation=90, hjust=1)
         ))

dr_firstseen_smooth_plt.save(filename = 'images/1_dr_firstseen_smooth_plt.
png', height=5, width=10, units = 'in', dpi=1000)

dr_firstseen_smooth_plt
```

The use of geom_smooth() gives a somewhat less noisy view and shows the variability of the domain rating over time to show how consistent the quality is (Figure 5-4). Again, this correlates to the quality of the links being acquired.

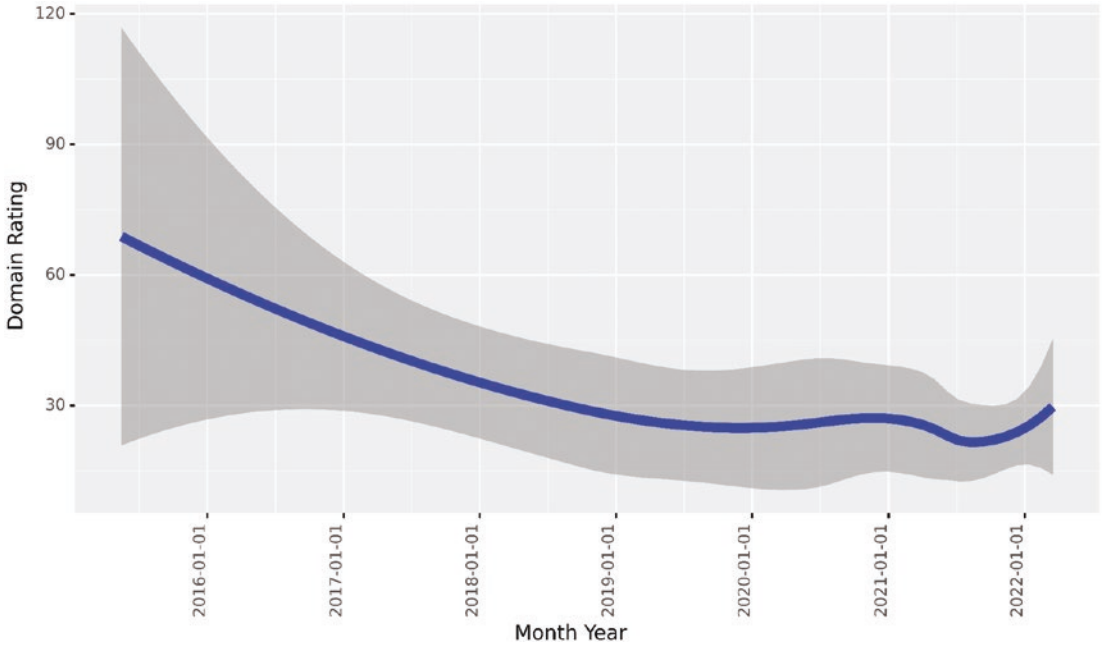

Figure 5-4. *Backlink domain rating acquired smoothed over time*

What this doesn't quite describe is the overall site authority over time, because the value of links acquired is retained over time; therefore, a different math approach is required.

To see the site's authority over time, we will calculate a running average of the domain rating by month of the year. Note the use of the expanding() function which instructs Pandas to include all previous rows with each new row:

```
target_rd_cummean_df = target_ahrefs_analysis
target_rd_mean_df = target_rd_cummean_df.groupby(['month_year'])['dr'].
sum().reset_index()

target_rd_mean_df['dr_runavg'] = target_rd_mean_df['dr'].expanding().mean()

target_rd_mean_df.head(10)
```

This results in the following:

	month_year	dr	dr_runavg
0	2015-05	70.0	70.000000
1	2016-12	45.0	57.500000
2	2017-02	15.0	43.333333
3	2017-03	66.0	49.000000
4	2017-06	132.0	65.600000
5	2017-08	1.3	54.883333
6	2017-09	91.0	60.042857
7	2018-02	15.0	54.412500
8	2018-05	50.0	53.922222
9	2018-06	32.9	51.820000

We now have a table which we can use to feed the graph and visualize.

```
dr_cummean_smooth_plt = (
    ggplot(target_rd_mean_df, aes(x = 'month_year', y = 'dr_runavg',
    group = 1)) +
    geom_line(alpha = 0.6, colour = 'blue', size = 2) +
    #labs(y = 'GA Sessions', x = 'Date') +
    scale_y_continuous() +
    scale_x_date() +
    theme(legend_position = 'right',
          axis_text_x=element_text(rotation=90, hjust=1)
         ))

dr_cummean_smooth_plt
```

So the target site started with high authority links (which may have been a PR campaign announcing the business brand), which faded soon after for four years and then rebooted with new acquisition of high authority links again (Figure 5-5).

Figure 5-5. *Cumulative average domain rating of backlinks over time*

Most importantly, we can see the site's general authority over time, which is how a search engine like Google may see it too.

A really good extension to this analysis would be to regenerate the dataframe so that we would plot the distribution over time on a cumulative basis. Then we could not only see the median quality but also the variation over time too.

That's the link quality, what about quantity?

Targeting Link Volumes

Quality is one thing; the volume of quality links is quite another, which is what we'll analyze next.

We'll use the expanding function like the previous operation to calculate a cumulative sum of the links acquired to date:

```
target_count_cumsum_df = target_ahrefs_analysis
print(target_count_cumsum_df.columns)
target_count_cumsum_df = target_count_cumsum_df.groupby(['month_year'])
['rd_count'].sum().reset_index()

target_count_cumsum_df['count_runsum'] = target_count_cumsum_df['rd_
count'].expanding().sum()
target_count_cumsum_df['link_velocity'] = target_count_cumsum_df['rd_
count'].diff()

target_count_cumsum_df
```

This results in the following:

	month_year	rd_count	count_runsum	link_velocity
0	2015-05	1	1.0	NaN
1	2016-12	1	2.0	0.0
2	2017-02	1	3.0	0.0
3	2017-03	1	4.0	0.0
4	2017-06	3	7.0	2.0
5	2017-08	1	8.0	-2.0
6	2017-09	1	9.0	0.0
7	2018-02	1	10.0	0.0
8	2018-05	1	11.0	0.0
9	2018-06	2	13.0	1.0

That's the data, now the graphs.

```
target_count_plt = (
    ggplot(target_count_cumsum_df, aes(x = 'month_year', y = 'rd_count',
    group = 1)) +
    geom_line(alpha = 0.6, colour = 'blue', size = 2) +
    labs(y = 'Count of Referring Domains', x = 'Month Year') +
    scale_y_continuous() +
    scale_x_date() +
    theme(legend_position = 'right',
```

```
          axis_text_x=element_text(rotation=90, hjust=1)
         ))

target_count_plt.save(filename = 'images/3_target_count_plt.png',
                          height=5, width=10, units = 'in', dpi=1000)

target_count_plt
```

This is a noncumulative view of the amount of referring domains. Again, this is useful for evaluating how effective a team is at acquiring links (Figure 5-6).

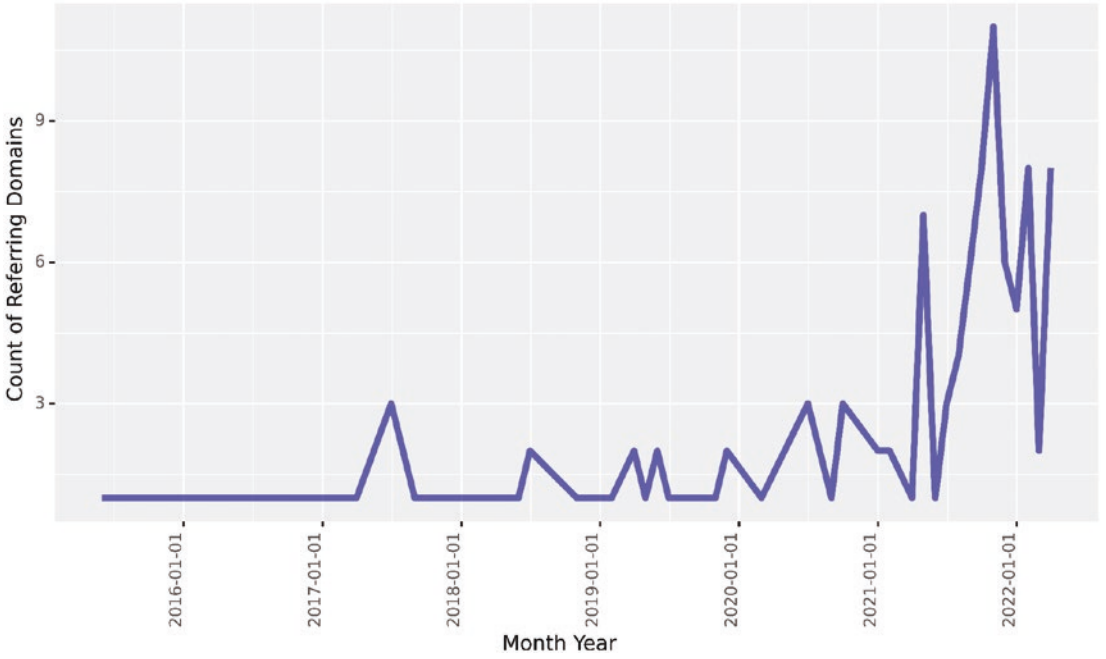

Figure 5-6. *Count of referring domains over time*

But perhaps it is not as useful for how a search engine would view the overall number of referring domains a site has.

```
target_count_cumsum_plt = (
    ggplot(target_count_cumsum_df, aes(x = 'month_year', y = 'count_
    runsum', group = 1)) +
    geom_line(alpha = 0.6, colour = 'blue', size = 2) +
    scale_y_continuous() +
    scale_x_date() +
```

```
    theme(legend_position = 'right',
          axis_text_x=element_text(rotation=90, hjust=1)
         ))

target_count_cumsum_plt
```

The cumulative view shows us the total number of referring domains (Figure 5-7). Naturally, this isn't the entirely accurate picture as some referring domains may have been lost, but it's good enough to get the gist of where the site is at.

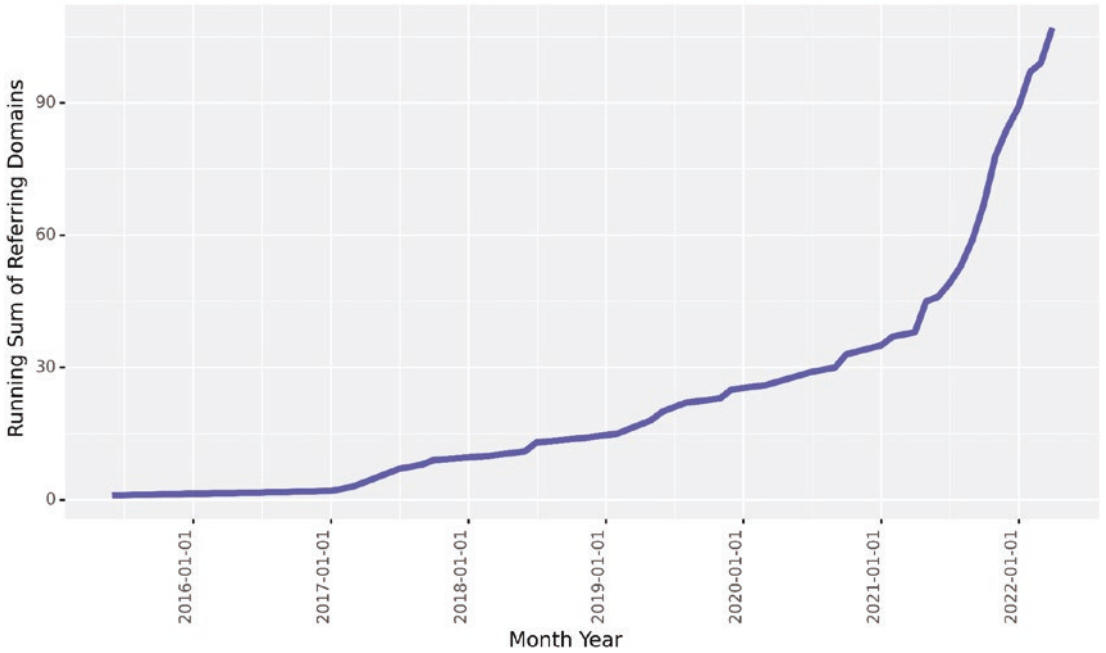

Figure 5-7. *Cumulative sum of referring domains over time*

We see that links were steadily added from 2017 for the next four years before accelerating again around March 2021. This is consistent with what we have seen with domain rating over time.

A useful extension to correlate that with performance may be to layer in

- Referring domain site traffic
- Average ranking over time

Analyzing Your Competitor's Links

Like last time, we defined the value of a site's backlinks for SEO as a product of quality and quantity – quality being the domain authority (or AHREF's equivalent domain rating) and quantity as the number of referring domains.

Again, we'll start by evaluating the link quality with the available data before evaluating the quantity. Time to code.

```
import re
import time
import random
import pandas as pd
import numpy as np
import datetime
from datetime import timedelta
from plotnine import *
import matplotlib.pyplot as plt
from pandas.api.types import is_string_dtype
from pandas.api.types import is_numeric_dtype
import uritools
```

```
pd.set_option('display.max_colwidth', None)
%matplotlib inline
```

```
root_domain = 'johnsankey.co.uk'
hostdomain = 'www.johnsankey.co.uk'
hostname = 'johnsankey'
full_domain = 'https://www.johnsankey.co.uk'
target_name = 'John Sankey'
```

Data Importing and Cleaning

We set up the file directories so we can read multiple AHREF exported data files in one folder, which is much faster, less boring, and more efficient than reading each file individually, especially when you have over ten of them:

```
ahrefs_path = 'data/'
```

The listdir() function from the OS module allows us to list all of the files in a subdirectory:

```
ahrefs_filenames = os.listdir(ahrefs_path)

ahrefs_filenames
```

This results in the following:

```
['www.davidsonlondon.com--refdomains-subdomain__2022-03-13_23-37-29.csv',
 'www.stephenclasper.co.uk--refdomains-subdoma__2022-03-13_23-47-28.csv',
 'www.touchedinteriors.co.uk--refdomains-subdo__2022-03-13_23-42-05.csv',
 'www.lushinteriors.co--refdomains-subdomains__2022-03-13_23-44-34.csv',
 'www.kassavello.com--refdomains-subdomains__2022-03-13_23-43-19.csv',
 'www.tulipinterior.co.uk--refdomains-subdomai__2022-03-13_23-41-04.csv',
 'www.tgosling.com--refdomains-subdomains__2022-03-13_23-38-44.csv',
 'www.onlybespoke.com--refdomains-subdomains__2022-03-13_23-45-28.csv',
 'www.williamgarvey.co.uk--refdomains-subdomai__2022-03-13_23-43-45.csv',
 'www.hadleyrose.co.uk--refdomains-subdomains__2022-03-13_23-39-31.csv',
 'www.davidlinley.com--refdomains-subdomains__2022-03-13_23-40-25.csv',
 'johnsankey.co.uk-refdomains-subdomains__2022-03-18_15-15-47.csv']
```

With the files listed, we'll now read each one individually using a for loop and add these to a dataframe. While reading in the file, we'll use some string manipulation to create a new column with the site name of the data we're importing:

```
ahrefs_df_lst = list()
ahrefs_colnames = list()

for filename in ahrefs_filenames:
    df = pd.read_csv(ahrefs_path + filename)
    df['site'] = filename
    df['site'] = df['site'].str.replace('www.', '', regex = False)
    df['site'] = df['site'].str.replace('.csv', '', regex = False)
    df['site'] = df['site'].str.replace('-.+', '', regex = True)
    ahrefs_colnames.append(df.columns)
    ahrefs_df_lst.append(df)

comp_ahrefs_df_raw = pd.concat(ahrefs_df_lst)

comp_ahrefs_df_raw
```

This results in the following:

	Domain	DR	Dofollow ref. domains	Dofollow linked domains	Traffic	Links to target	New links	Lost links	Dofollow links	First seen	Lost	site
0	pinterest.co.uk	92.0	189919	46703	14142143	2	0	0	2	18/08/2020 02:04	NaN	davidsonlondon.com
1	standard.co.uk	89.0	403937	7582	9931955	2	0	0	2	10/03/2021 11:53	NaN	davidsonlondon.com
2	myminifactory.com	78.0	40446	18726	293812	12	0	0	12	29/03/2017 16:29	NaN	davidsonlondon.com
3	idealhome.co.uk	77.0	18278	3133	2005577	2	0	0	0	15/02/2020 17:39	NaN	davidsonlondon.com
4	thomsonlocal.com	77.0	4895	24160	383648	22	0	6	0	18/06/2020 06:31	NaN	davidsonlondon.com
...	...	...	...	...	...	...	...	...	...	...	...	...
102	ikhatotherescue.blogspot.com	0.0	73	2314	0	1	0	0	1	18/11/2021 05:37	NaN	johnsankey.co.uk
103	peakedgehotel.blogspot.com	0.0	0	0	0	8	0	0	8	17/06/2020 23:30	NaN	johnsankey.co.uk
104	thelibertyscale.blogspot.com	0.0	0	0	1	1	0	0	1	11/04/2021 06:21	NaN	johnsankey.co.uk
105	plums-rhombus-lrg3.squarespace.com	0.0	0	0	0	1	1	1	1	14/03/2022 11:18	16/03/2022 16:51	johnsankey.co.uk
106	upholsterycleaningdozaowa.blogspot.com	0.0	0	0	0	1	0	0	1	04/04/2019 13:29	NaN	johnsankey.co.uk

5409 rows × 12 columns

Now we have the raw data from each site in a single dataframe, the next step is to tidy up the column names and make them a bit more friendlier to work with. A custom function could be used, but we'll just chain the function calls with a list comprehension:

```
competitor_ahrefs_cleancols = comp_ahrefs_df_raw.copy()
competitor_ahrefs_cleancols.columns = [col.lower().replace(' ','_').
replace('.','_').replace('__','_').replace('(','')
 .replace(')','').replace('%','')
 for col in competitor_ahrefs_cleancols.columns]
```

Having a count column and a single value column ("project") is useful for groupby and aggregation operations:

```
competitor_ahrefs_cleancols['rd_count'] = 1
competitor_ahrefs_cleancols['project'] = target_name

competitor_ahrefs_cleancols
```

This results in the following:

	domain	dr	dofollow_ref_domains	dofollow_linked_domains	traffic_	links_to_target	new_links	lost_links	dofollow_links
0	pinterest.co.uk	92.0	189919	46703	14142143	2	0	0	2
1	standard.co.uk	89.0	403937	7582	9931955	2	0	0	2
2	myminifactory.com	78.0	40446	18726	293812	12	0	0	12
3	idealhome.co.uk	77.0	18278	3133	2005577	2	0	0	0
4	thomsonlocal.com	77.0	4895	24160	383648	22	0	6	0
...	...	...	...	...	...	...	...	...	...
102	ikhatotherescue.blogspot.com	0.0	73	2314	0	1	0	0	1
103	peakedgehotel.blogspot.com	0.0	0	0	0	8	0	0	8
104	thelibertyscale.blogspot.com	0.0	0	0	1	1	0	0	1
105	plums-rhombus-lrg3.squarespace.com	0.0	0	0	0	1	1	1	1
106	upholsterycleaningdozaowa.blogspot.com	0.0	0	0	0	1	0	0	1

5409 rows × 14 columns

The columns are now cleaned up, so we'll now clean up the row data:

```
competitor_ahrefs_clean_dtypes = competitor_ahrefs_cleancols
```

For referring domains, we're replacing hyphens with zero and setting the data type as an integer (i.e., whole number). This will be repeated for linked domains, also:

```
competitor_ahrefs_clean_dtypes['dofollow_ref_domains'] =
np.where(competitor_ahrefs_clean_dtypes['dofollow_ref_domains'] == '-',
                    0, competitor_ahrefs_clean_dtypes['dofollow_ref_
                    domains'])
competitor_ahrefs_clean_dtypes['dofollow_ref_domains'] = competitor_ahrefs_
clean_dtypes['dofollow_ref_domains'].astype(int)

# linked_domains
competitor_ahrefs_clean_dtypes['dofollow_linked_domains'] =
np.where(competitor_ahrefs_clean_dtypes['dofollow_linked_domains'] == '-',
                    0, competitor_ahrefs_clean_dtypes['dofollow_linked_
                    domains'])
competitor_ahrefs_clean_dtypes['dofollow_linked_domains'] = competitor_
ahrefs_clean_dtypes['dofollow_linked_domains'].astype(int)
```

First seen gives us a date point at which links were found, which we can use for time series plotting and deriving the link age. We'll convert to date format using the to_datetime function:

```
competitor_ahrefs_clean_dtypes['first_seen'] = pd.to_datetime(competitor_
ahrefs_clean_dtypes['first_seen'],
                        format='%d/%m/%Y %H:%M')
competitor_ahrefs_clean_dtypes['first_seen'] = competitor_ahrefs_clean_
dtypes['first_seen'].dt.normalize()
competitor_ahrefs_clean_dtypes['month_year'] = competitor_ahrefs_clean_
dtypes['first_seen'].dt.to_period('M')
```

To calculate the link_age, we'll simply deduct the first seen date from today's date and convert the difference into a number:

```
competitor_ahrefs_clean_dtypes['link_age'] = dt.datetime.now() -
competitor_ahrefs_clean_dtypes['first_seen']
competitor_ahrefs_clean_dtypes['link_age'] = competitor_ahrefs_clean_
dtypes['link_age']
competitor_ahrefs_clean_dtypes['link_age'] = competitor_ahrefs_clean_
dtypes['link_age'].astype(int)
competitor_ahrefs_clean_dtypes['link_age'] = (competitor_ahrefs_clean_
dtypes['link_age']/(3600 * 24 * 1000000000)).round(0)
```

The target column helps us distinguish the "client" site vs. competitors, which is useful for visualization later:

```
competitor_ahrefs_clean_dtypes['target'] = np.where(competitor_ahrefs_
clean_dtypes['site'].str.contains('johns'),
                            1, 0)
competitor_ahrefs_clean_dtypes['target'] = competitor_ahrefs_clean_
dtypes['target'].astype('category')

competitor_ahrefs_clean_dtypes
```

This results in the following:

mains	traffic_	links_to_target	new_links	lost_links	dofollow_links	first_seen	lost	site	rd_count	project	month_year	link_age	target
46703	14142143	2	0	0	2	2020-08-18	NaN	davidsonlondon.com	1	John Sankey	2020-08	630.0	0
7582	9931955	2	0	0	2	2021-03-10	NaN	davidsonlondon.com	1	John Sankey	2021-03	426.0	0
18726	293812	12	0	0	12	2017-03-29	NaN	davidsonlondon.com	1	John Sankey	2017-03	1868.0	0
3133	2005577	2	0	0	0	2020-02-15	NaN	davidsonlondon.com	1	John Sankey	2020-02	815.0	0
24160	383648	22	0	6	0	2020-06-18	NaN	davidsonlondon.com	1	John Sankey	2020-06	691.0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
2314	0	1	0	0	1	2021-11-18	NaN	johnsankey.co.uk	1	John Sankey	2021-11	173.0	1
0	0	8	0	0	8	2020-06-17	NaN	johnsankey.co.uk	1	John Sankey	2020-06	692.0	1
0	1	1	0	0	1	2021-04-11	NaN	johnsankey.co.uk	1	John Sankey	2021-04	394.0	1
0	0	1	1	1	1	2022-03-14	16/03/2022 16:51	johnsankey.co.uk	1	John Sankey	2022-03	57.0	1
0	0	1	0	0	1	2019-04-04	NaN	johnsankey.co.uk	1	John Sankey	2019-04	1132.0	1

Now that the data is cleaned up both in terms of column titles and row values, we're ready to set forth and start analyzing.

Anatomy of a Good Link

When we analyzed the one target website earlier ("John Sankey"), we assumed (like the rest of the SEO industry the world over) that domain rating (DR) was the best and most reliable measure of the link quality.

But should we? Let's do a quick and dirty analysis to see if that is indeed the case or whether we can find something better. We'll start by aggregating the link features at the site level:

```
competitor_ahrefs_aggs = competitor_ahrefs_analysis.groupby('site').
agg({'link_age': 'mean',
          'dofollow_links': 'mean',    'domain': 'count', 'dr': 'mean',
'dofollow_ref_domains': 'mean',  'traffic_': 'mean', 'dofollow_
linked_domains': 'mean',    'links_to_target': 'mean',  'new_links':
'mean',     'lost_links': 'mean'}).reset_index()

competitor_ahrefs_aggs
```

This results in the following:

	site	link_age	dofollow_links	domain	dr	dofollow_ref_domains	traffic_	dofollow_linked_domains	links_to_target	new_links
0	davidlinley.com	794.675000	3.002000	1000	32.662200	22648.514000	6.456371e+06	83373.709000	4.031000	0.18800
1	davidsonlondon.com	329.118598	2.029650	371	11.003774	3515.447439	1.351592e+05	116973.943396	3.048518	0.328841
2	hadleyrose.co.uk	226.657316	1.326763	1319	1.944882	1297.971190	1.022266e+05	59388.066717	1.563306	0.146323
3	johnsankey.co.uk	540.121495	2.411215	107	26.593458	22557.794393	1.334358e+06	366136.514019	3.383178	0.64486
4	kassavello.com	301.018913	1.621749	423	6.077778	1279.536643	8.683198e+04	117990.945626	1.829787	0.10874
5	lushinteriors.co	227.137255	1.039216	306	4.706863	1202.562092	8.210043e+04	31187.594771	1.258170	0.15686
6	onlybespoke.com	267.407407	0.944444	108	16.330556	2934.129630	1.887193e+05	144268.425926	1.833333	0.29629
7	stephenclasper.co.uk	581.516129	1.516129	31	30.548387	8565.096774	7.907171e+04	302009.161290	3.612903	0.22580
8	tgosling.com	480.582474	1.783505	194	17.840722	14784.639175	1.181992e+06	197045.087629	2.221649	0.19072
9	touchedinteriors.co.uk	741.674627	5.549254	335	18.501194	8577.185075	4.250500e+05	222064.211940	7.188060	0.38806
10	tulipinterior.co.uk	273.664198	1.562963	810	7.666667	1884.629630	4.224160e+04	97459.551852	1.990123	0.14938
11	williamgarvey.co.uk	529.738272	4.024691	405	17.768889	8033.827160	1.909090e+05	325999.076543	5.728395	0.39506

The resulting table shows us aggregated statistics for each of the link features. Next, read in the list of SEMRush domain level data (which by way of manual data entry was literally typed in since it's only 11 sites):

```
semrush_viz = [10100, 2300, 931, 2400, 911, 2100, 1800, 136, 838, 428,
1100, 1700]

competitor_ahrefs_aggs['semrush_viz'] = semrush_viz

competitor_ahrefs_aggs
```

This results in the following:

	link_age	dofollow_links	domain	dr	dofollow_ref_domains	traffic_	dofollow_linked_domains	links_to_target	new_links	lost_links	semrush_viz
	794.675000	3.002000	1000	32.662200	22648.514000	6.456371e+06	83373.709000	4.031000	0.188000	0.193000	10100
	329.118598	2.029650	371	11.003774	3515.447439	1.351592e+05	116973.943396	3.048518	0.328841	0.121294	2300
	226.657316	1.326763	1319	1.944882	1297.971190	1.022266e+05	59388.066717	1.563306	0.146323	0.152388	931
	540.121495	2.411215	107	26.593458	22557.794393	1.334358e+06	366136.514019	3.383178	0.644860	0.093458	2400
	301.018913	1.621749	423	6.077778	1279.536643	8.683198e+04	117990.945626	1.829787	0.108747	0.120567	911
	227.137255	1.039216	306	4.706863	1202.562092	8.210043e+04	31187.594771	1.258170	0.156863	0.101307	2100
	267.407407	0.944444	108	16.330556	2934.129630	1.887193e+05	144268.425926	1.833333	0.296296	0.185185	1800
	581.516129	1.516129	31	30.548387	8565.096774	7.907171e+04	302009.161290	3.612903	0.225806	0.161290	136
	480.582474	1.783505	194	17.840722	14784.639175	1.181992e+06	197045.087629	2.221649	0.190722	0.164948	838
	741.674627	5.549254	335	18.501194	8577.185075	4.250500e+05	222064.211940	7.188060	0.388060	0.092537	428
	273.664198	1.562963	810	7.666667	1884.629630	4.224160e+04	97459.551852	1.990123	0.149383	0.139506	1100
	529.738272	4.024691	405	17.768889	8033.827160	1.909090e+05	325999.076543	5.728395	0.395062	0.708642	1700

The SEMRush visibility data has now been appended, so we're ready to find some r-squared, known as the coefficient of determination, which will tell which link feature can best explain the variation in SEMRush visibility:

```
competitor_ahrefs_r2 = competitor_ahrefs_aggs.corr() ** 2
competitor_ahrefs_r2 = competitor_ahrefs_r2[['semrush_viz']].reset_index()
competitor_ahrefs_r2 = competitor_ahrefs_r2.sort_values('semrush_viz',
ascending = False)

competitor_ahrefs_r2
```

This results in the following:

	index	semrush_viz
10	semrush_viz	1.000000
5	traffic_	0.890900
4	dofollow_ref_domains	0.336989
3	dr	0.214275
0	link_age	0.204189
2	domain	0.148347
6	dofollow_linked_domains	0.064904
1	dofollow_links	0.014366
7	links_to_target	0.007580
9	lost_links	0.001712
8	new_links	0.001055

Naturally, we'd expect the semrush_viz to correlate perfectly with itself. DR (domain rating) surprisingly doesn't explain the difference in SEMRush very well with an r_squared of 21%.

On the other hand, "traffic_" which is the referring domain's traffic value correlates better. From this alone, we're prepared to disregard "dr." Let's inspect this visually:

```
comp_correl_trafficviz_plt = (
    ggplot(competitor_ahrefs_aggs,
           aes(x = 'traffic_', y = 'semrush_viz')) +
    geom_point(alpha = 0.4, colour = 'blue', size = 2) +
```

```
    geom_smooth(method = 'lm', se = False, colour = 'red', size = 3,
    alpha = 0.4)
)

comp_correl_trafficviz_plt.save(filename = 'images/2_comp_correl_
trafficviz_plt.png',
                    height=5, width=10, units = 'in', dpi=1000)

comp_correl_trafficviz_plt
```

This is not terribly convincing (Figure 5-8), due to the lack of referring domains beyond 2,000,000. Does this mean we should disregard traffic_ as a measure?

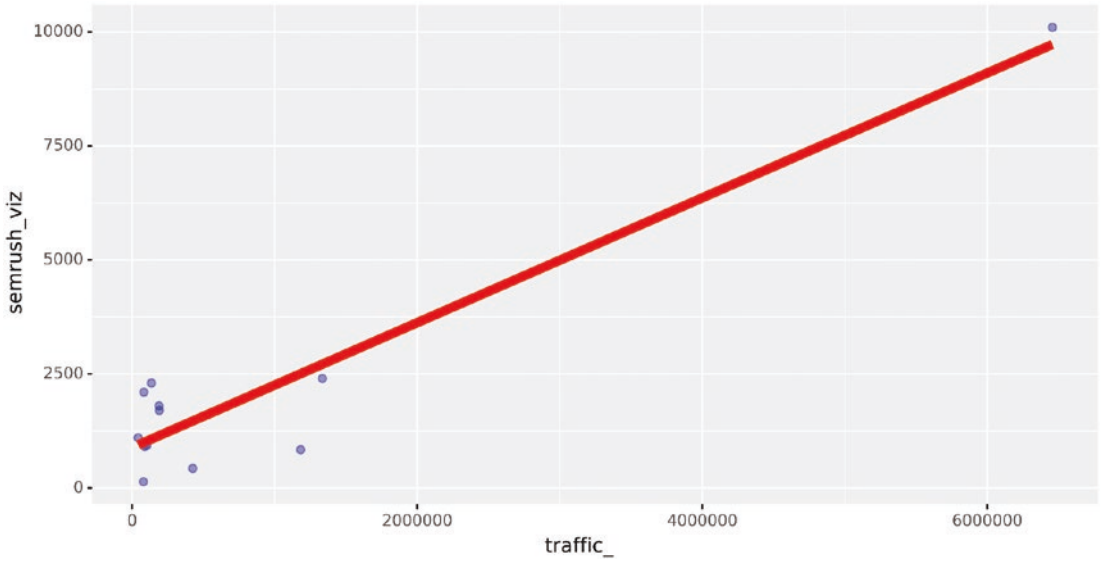

Figure 5-8. *Scatterplot of the SEMRush visibility (semrush_viz) vs. the total AHREFs backlink traffic (traffic_) of the site's backlinks*

Not necessarily. The outlier data point with 10,000 visibility isn't necessarily incorrect. The site does have superior visibility and more referring traffic in the real world, so it doesn't mean the site's data should be removed.

If anything, more data should be gathered with more domains in the same sector. Alternatively, pursuing a more thorough treatment would involve obtaining SEMRush visibility data at the page level and correlating this with page-level link feature metrics.

Going forward, we will use traffic_ as our measure of quality.

Link Quality

We start with link quality, which we've very recently discovered should be measured by "traffic_" as opposed to the industry accepted.

Let's start by inspecting the distributive properties of each link feature using the describe() function:

```
competitor_ahrefs_analysis = competitor_ahrefs_clean_dtypes
competitor_ahrefs_analysis[['traffic_']].describe()
```

The resulting table shows some basic statistics including the mean, standard deviation (std), and interquartile metrics (25th, 50th, and 75th percentiles), which give you a good idea of where most referring domains fall in terms of referring domain traffic.

	traffic_
count	5.409000e+03
mean	1.359225e+06
std	4.572404e+07
min	0.000000e+00
25%	0.000000e+00
50%	0.000000e+00
75%	8.400000e+01
max	3.191808e+09

So unsurprisingly, if we look at the median, then most of the competitors' referring domains have zero (estimated) traffic. Only domains in the 75th percentile or above have traffic.

We can also plot (and confirm visually) their distribution using the geom_boxplot function to compare sites side by side:

```
comp_dr_dist_box_plt = (
    ggplot(competitor_ahrefs_analysis, #.loc[competitor_ahrefs_
    analysis['dr'] > 0],
           aes(x = 'reorder(site, traffic_)', y = 'traffic_',
           colour = 'target')) +
```

```
    geom_boxplot(alpha = 0.6) +
    scale_y_log10() +
    theme(legend_position = 'none',
          axis_text_x=element_text(rotation=90, hjust=1)
         ))

comp_dr_dist_box_plt.save(filename = 'images/4_comp_traffic_dist_box_plt.
png', height=5, width=10, units = 'in', dpi=1000)

comp_dr_dist_box_plt
```

comp_dr_dist_box_plt compares a site's distribution of referring domain traffic side by side (Figure 5-9) and most notably the interquartile range (IQR). The competitors are in red, and the client is in blue.

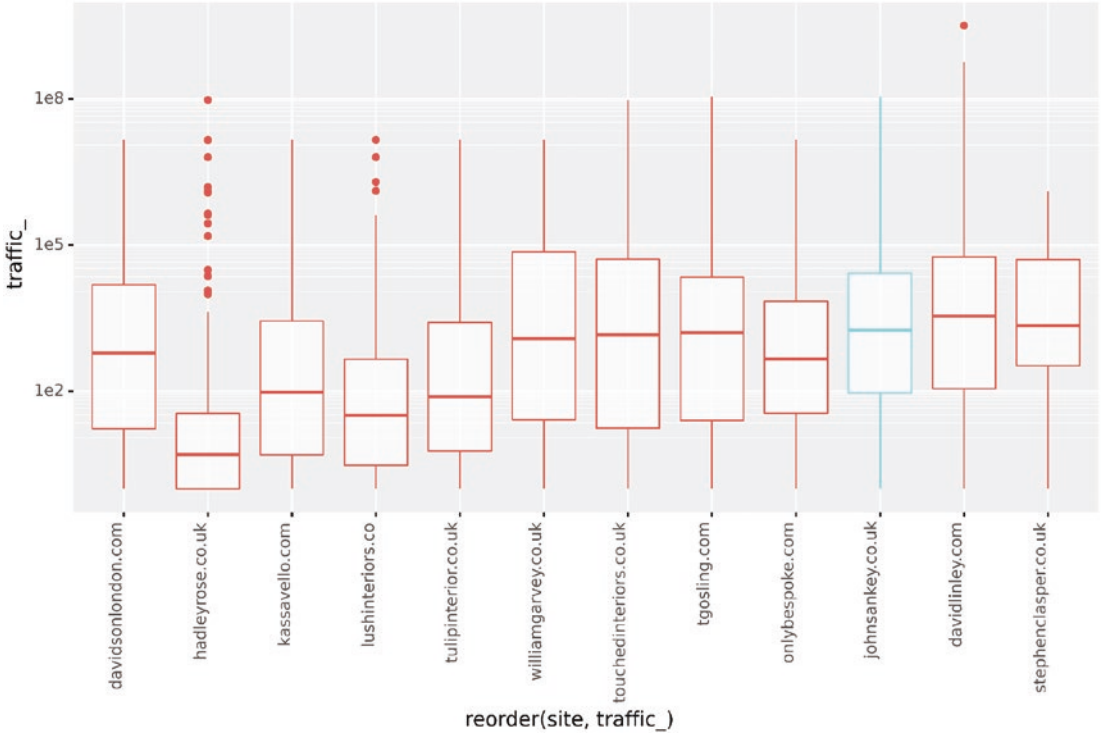

Figure 5-9. *Box plot of each website's backlink traffic (traffic_)*

The interquartile range is the range of data between its 25th percentile and 75th percentile. The purpose is to tell us

- Where most of the data is
- How much of the data is away from the median (the center)

In this case, the IQR is quantifying how much traffic each site's referring domains get and its variability.

We also see that "John Sankey" has the third highest median referring domain traffic which compares well in terms of link quality against their competitors. The size of the box (its IQR) is not the longest (quite consistent around its median) but not as short as Stephen Clasper (more consistent, with a higher median and more backlinks from referring domain sites higher than the median).

"Touched Interiors" has the most diverse range of DR compared with other domains, which could indicate an ever so slightly more relaxed criteria for link acquisition. Or is it the case that as your brand becomes more well known and visible online, this brand has naturally attracted more links from zero traffic referring domains? Maybe both.

Let's plot the domain quality over time for each competitor:

```
comp_traf_timeseries_plt = (
    ggplot(competitor_ahrefs_analysis,
           aes(x = 'first_seen', y = 'traffic_',
               group = 'site', colour = 'site')) +
    geom_smooth(alpha = 0.4, size = 2, se = False,
                method='loess'
               ) +
    scale_x_date() +
    theme(legend_position = 'right',
          axis_text_x=element_text(rotation=90, hjust=1)
         )
)

comp_traf_timeseries_plt.save(filename = 'images/4_comp_traffic_timeseries_
plt.png', height=5, width=10, units = 'in', dpi=1000)

comp_traf_timeseries_plt
```

We deliberately avoided using scale_y_log10() which would have transformed the vertical axis using logarithmic scales. Why? Because it would look very noisy and difficult to see any standout competitors.

Figure 5-10 shows the quality of links acquired over time of which the standout sites are David Linley, T Gosling, and John Sankey.

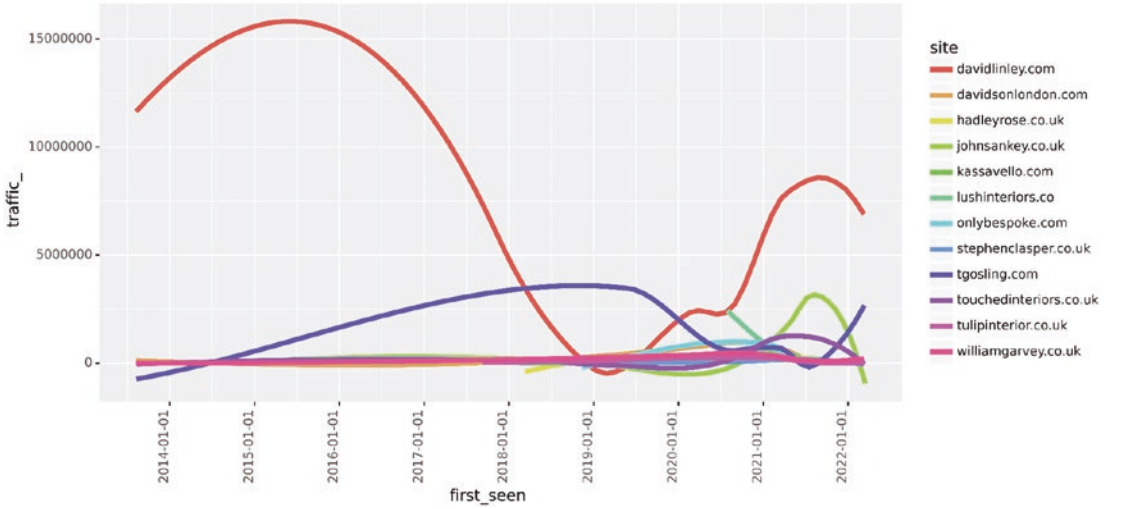

Figure 5-10. *Time series plot showing the amount of traffic each referring domain has over time for each website*

The remaining sites are more or less flat in terms of their link acquisition performance. David Linley started big, then dive-bombed in terms of link quality before improving again in 2020 and 2021.

Now that we have some concept of how the different sites perform, what we really want is a cumulative link quality by month_year as this is likely to be additive in the way search engines evaluate the authority of websites.

We'll use our trusted groupby() and expanding().mean() functions to compute the cumulative stats we want:

```
competitor_traffic_cummean_df = competitor_ahrefs_analysis.copy()

competitor_traffic_cummean_df = competitor_traffic_cummean_
df.groupby(['site', 'month_year'])['traffic_'].sum().reset_index()

competitor_traffic_cummean_df['traffic_runavg'] = competitor_traffic_
cummean_df['traffic_'].expanding().mean()

competitor_traffic_cummean_df
```

This results in the following:

	site	month_year	traffic_	traffic_runavg
0	davidlinley.com	2013-08	5770	5.770000e+03
1	davidlinley.com	2013-09	92	2.931000e+03
2	davidlinley.com	2013-10	32	1.964667e+03
3	davidlinley.com	2013-12	2	1.474000e+03
4	davidlinley.com	2014-02	0	1.179200e+03
...	...	...	...	...
502	williamgarvey.co.uk	2021-11	1940163	1.458292e+07
503	williamgarvey.co.uk	2021-12	14357281	1.458247e+07
504	williamgarvey.co.uk	2022-01	846774	1.455527e+07
505	williamgarvey.co.uk	2022-02	1628704	1.452973e+07
506	williamgarvey.co.uk	2022-03	3234	1.450108e+07

507 rows × 4 columns

Scientific formatted numbers aren't terribly helpful, nor is a table for that matter, but at least the dataframe is in a ready format to power the following chart:

```
competitor_traffic_cummean_plt = (
    ggplot(competitor_traffic_cummean_df, aes(x = 'month_year', y =
'traffic_runavg', group = 'site', colour = 'site')) +
    geom_line(alpha = 0.6, size = 2) +
    labs(y = 'Cumu Avg of traffic_', x = 'Month Year') +
    scale_y_continuous() +
    scale_x_date() +
    theme(legend_position = 'right',
          axis_text_x=element_text(rotation=90, hjust=1)
         ))

competitor_traffic_cummean_plt.save(filename = 'images/4_competitor_
traffic_cummean_plt.png', height=5, width=10, units = 'in', dpi=1000)

competitor_traffic_cummean_plt
```

The code is color coding the sites to make it easier to see which site is which.

So as we might expect, David Linley's link acquisition team has done well as their authority has made leaps and bounds over all of the competitors over time (Figure 5-11).

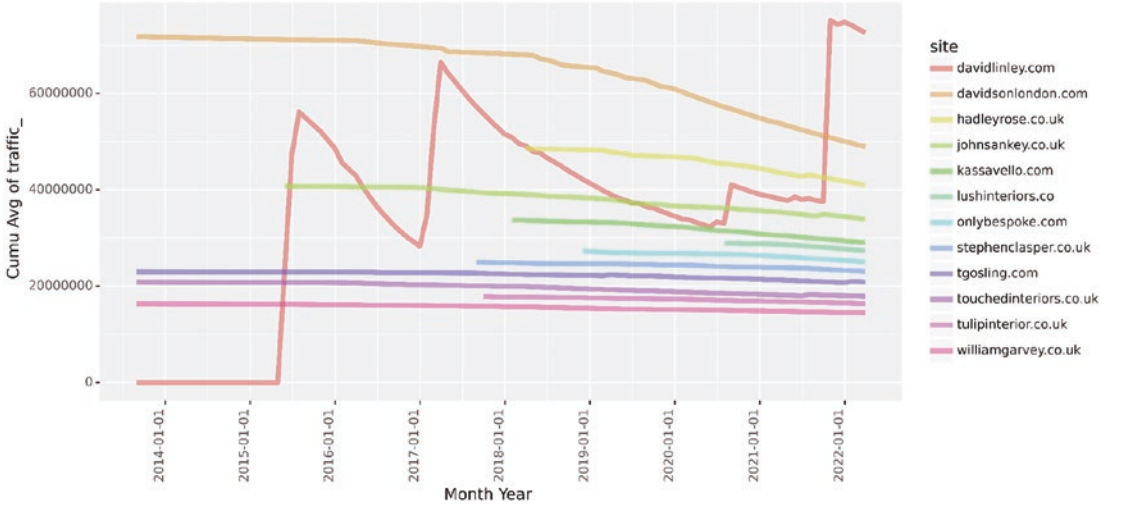

Figure 5-11. *Time series plot of the cumulative average backlink traffic for each website*

All of the other competitors have pretty much flatlined. This is reflected in David Linley's superior SEMRush visibility (Figure 5-12).

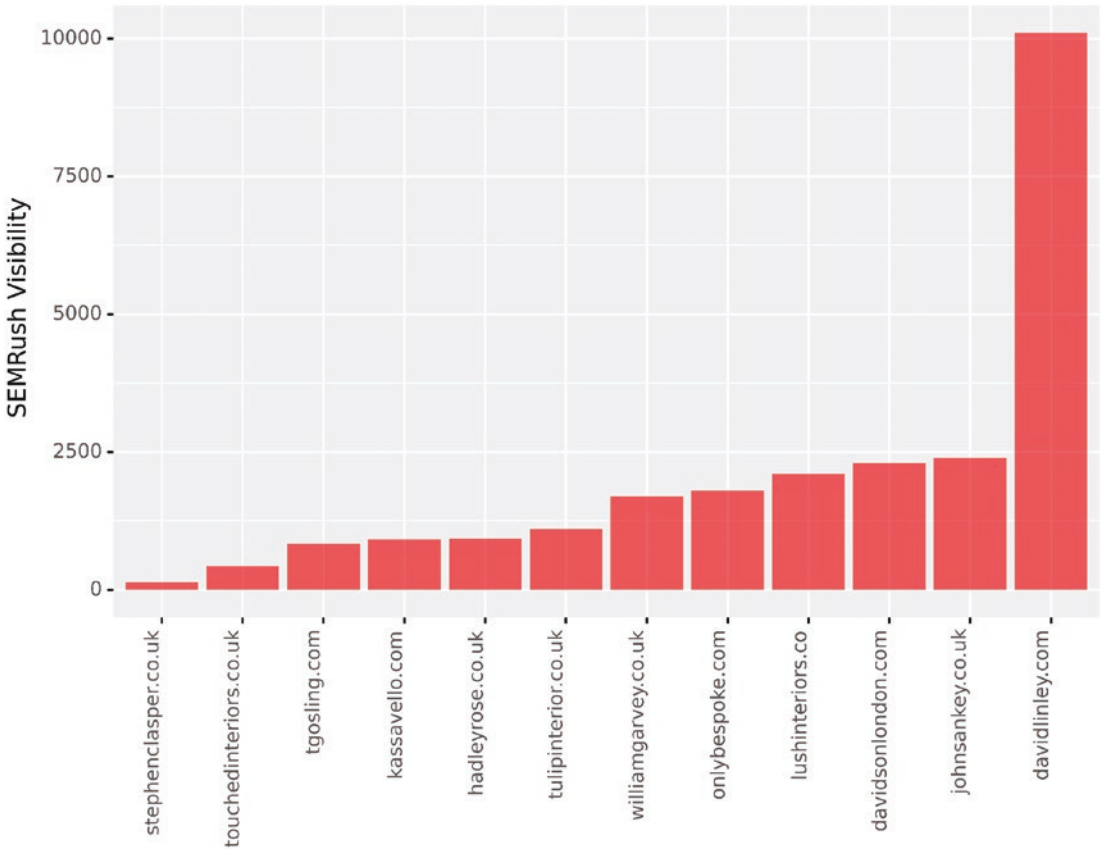

Figure 5-12. *Column chart showing the SEMRush visibility for each website*

What can we learn? So far in our limited data research, we can see that slow and steady does not win the day. By contrast, sites need to be going after links from high traffic sites in a big way.

Link Volumes

That's quality analyzed; what about the volume of links from referring domains?

Our approach will be to compute a cumulative sum of referring domains using the groupby() function:

```
competitor_count_cumsum_df = competitor_ahrefs_analysis

competitor_count_cumsum_df = competitor_count_cumsum_df.groupby(['site',
'month_year'])['rd_count'].sum().reset_index()
```

The expanding function allows the calculation window to grow with the number of rows, which is how we achieve our cumulative sum:

```
competitor_count_cumsum_df['count_runsum'] = competitor_count_cumsum_
df['rd_count'].expanding().sum()

competitor_count_cumsum_df
```

This results in the following:

	site	month_year	rd_count	count_runsum	link_velocity
0	davidlinley.com	2013-08	11	11.0	NaN
1	davidlinley.com	2013-09	1	12.0	-10.0
2	davidlinley.com	2013-10	1	13.0	0.0
3	davidlinley.com	2013-12	1	14.0	0.0
4	davidlinley.com	2014-02	1	15.0	0.0
...	...	...	...	...	...
502	williamgarvey.co.uk	2021-11	24	5324.0	-15.0
503	williamgarvey.co.uk	2021-12	36	5360.0	12.0
504	williamgarvey.co.uk	2022-01	22	5382.0	-14.0
505	williamgarvey.co.uk	2022-02	19	5401.0	-3.0
506	williamgarvey.co.uk	2022-03	8	5409.0	-11.0

507 rows × 5 columns

The result is a dataframe with the site, month_year, and count_runsum (the running sum), which is in the perfect format to feed the graph – which we will now run as follows:

```
competitor_count_cumsum_plt = (
    ggplot(competitor_count_cumsum_df, aes(x = 'month_year', y =
    'count_runsum',
     group = 'site', colour = 'site')) +
    geom_line(alpha = 0.6, size = 2) +
    labs(y = 'Running Sum of Referring Domains', x = 'Month Year') +
    scale_y_continuous() +
    scale_x_date() +
    theme(legend_position = 'right',
```

```
        axis_text_x=element_text(rotation=90, hjust=1)
       ))

competitor_count_cumsum_plt.save(filename = 'images/5_count_cumsum_smooth_
plt.png', height=5, width=10, units = 'in', dpi=1000)

competitor_count_cumsum_plt
```

The competitor_count_cumsum_plt plot (Figure 5-13) shows the number of referring domains for each site since 2014. What is quite interesting are the different starting positions for each site when they start acquiring links.

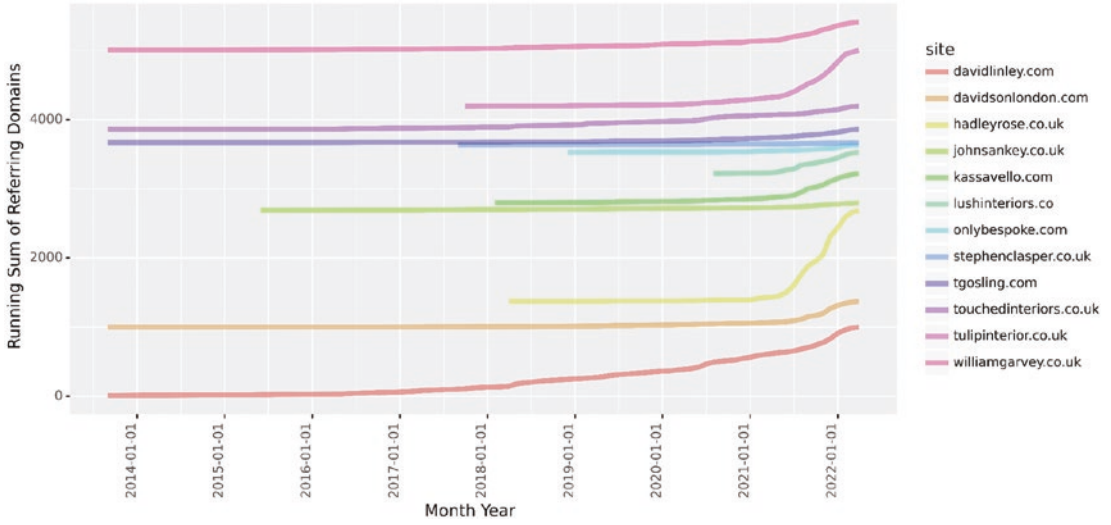

Figure 5-13. *Time series plot of the running sum of referring domains for each website*

For example, William Garvey started with over 5000 domains. I'd love to know who their digital PR team is.

We can also see the rate of growth, for example, although Hadley Rose started link acquisition in 2018, things really took off around mid-2021.

Link Velocity

Let's take a look at link velocity:

```
competitor_velocity_cumsum_plt = (
    ggplot(competitor_count_cumsum_df, aes(x = 'month_year', y = 'link_
    velocity',
     group = 'site', colour = 'site')) +
    geom_line(alpha = 0.6, size = 2) +
    labs(y = 'Running Sum of Referring Domains', x = 'Month Year') +
    scale_y_log10() +
    scale_x_date() +
    theme(legend_position = 'right',
          axis_text_x=element_text(rotation=90, hjust=1)
         ))

competitor_velocity_cumsum_plt.save(filename = 'images/5_competitor_
velocity_cumsum_plt.png',
                            height=5, width=10, units = 'in', dpi=1000)

competitor_velocity_cumsum_plt
```

The view shows the relative speed at which the sites are acquiring links (Figure 5-14). This is an unusual but useful view as for any given month you can see which site is acquiring the most links by virtue of the height of their lines.

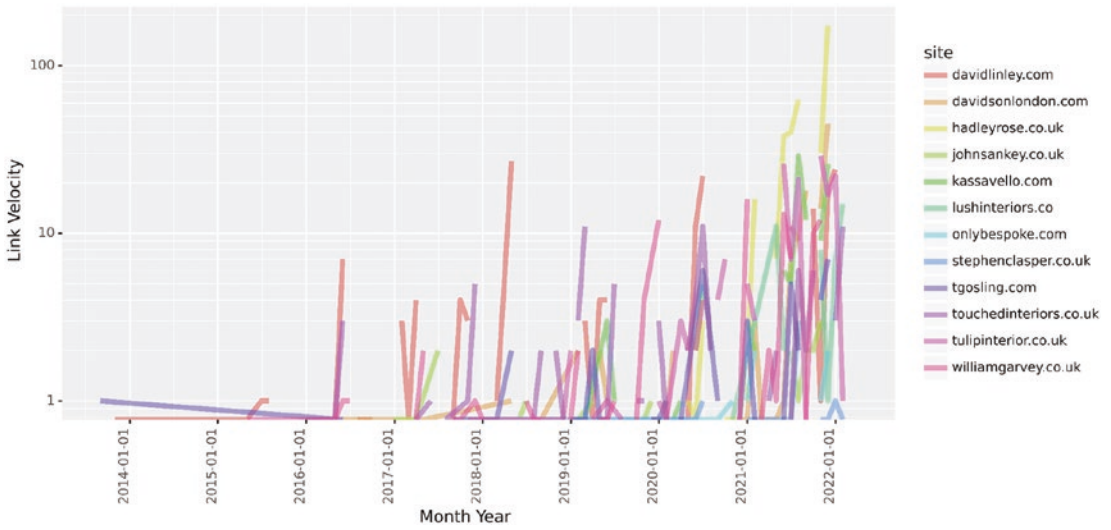

Figure 5-14. *Time series plot showing the link velocity of each website*

David Linley was winning the contest throughout the years until Hadley Rose came along.

Link Capital

Like most things that are measured in life, the ultimate value is determined by the product of their rate and volume. So we will apply the same principle to determine the overall value of a site's authority and call it "link capital."

We'll start by merging the running average stats for both link volume and average traffic (as our measure of authority):

```
competitor_capital_cumu_df = competitor_count_cumsum_df.merge(competitor_
traffic_cummean_df,
                          on = ['site', 'month_year'], how = 'left'
                         )

competitor_capital_cumu_df['auth_cap'] = (competitor_capital_cumu_
df['count_runsum'] * competitor_capital_cumu_df['traffic_runavg']).
round(1)*0.001

competitor_capital_cumu_df['auth_velocity'] = competitor_capital_cumu_
df['auth_cap'].diff()

competitor_capital_cumu_df
```

This results in the following:

	site	month_year	rd_count	count_runsum	link_velocity	traffic_	traffic_runavg	auth_cap	auth_velocity
0	davidlinley.com	2013-08	11	11.0	NaN	5770	5.770000e+03	6.347000e+01	NaN
1	davidlinley.com	2013-09	1	12.0	-10.0	92	2.931000e+03	3.517200e+01	-28.2980
2	davidlinley.com	2013-10	1	13.0	0.0	32	1.964667e+03	2.554070e+01	-9.6313
3	davidlinley.com	2013-12	1	14.0	0.0	2	1.474000e+03	2.063600e+01	-4.9047
4	davidlinley.com	2014-02	1	15.0	0.0	0	1.179200e+03	1.768800e+01	-2.9480
...	...	...	...	...	...	...	...	...	...
502	williamgarvey.co.uk	2021-11	24	5324.0	-15.0	1940163	1.458292e+07	7.763947e+07	216510.7948
503	williamgarvey.co.uk	2021-12	36	5360.0	12.0	14357281	1.458247e+07	7.816206e+07	522585.5084
504	williamgarvey.co.uk	2022-01	22	5382.0	-14.0	846774	1.455527e+07	7.833649e+07	174427.2243
505	williamgarvey.co.uk	2022-02	19	5401.0	-3.0	1628704	1.452973e+07	7.847506e+07	138573.1262
506	williamgarvey.co.uk	2022-03	8	5409.0	-11.0	3234	1.450108e+07	7.843632e+07	-38740.0970

507 rows × 9 columns

The merged table is produced with new columns auth_cap (measuring overall authority) and auth_velocity (the rate at which authority is being added).

Let's see how the competitors compare in terms of total authority over time in Figure 5-15.

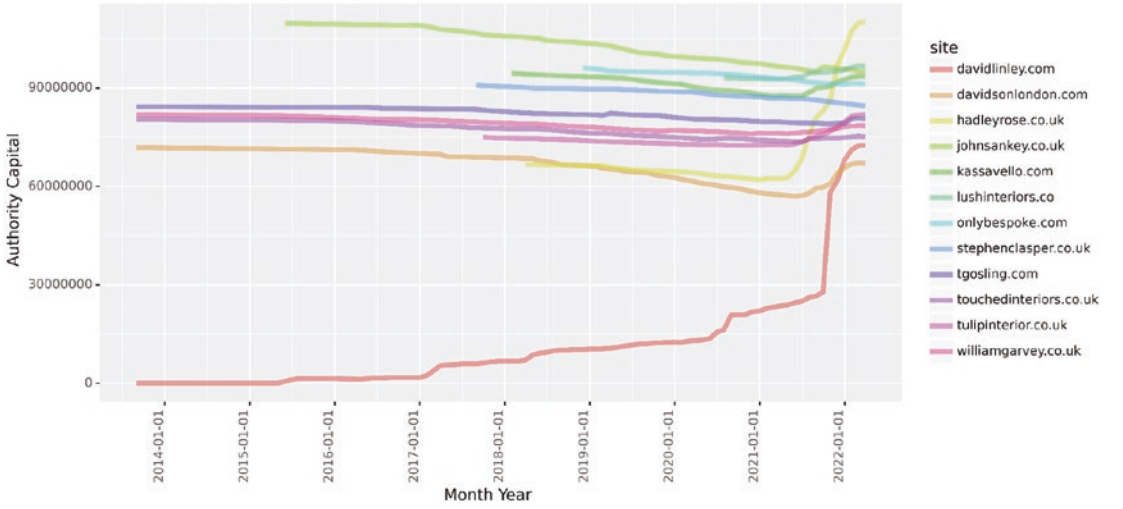

Figure 5-15. *Time series plot of authority capital over time by website*

The plot shows the link capital of several sites over time. What's quite interesting is how Hadley Rose emerged as the most authoritative with the third most consistently highest trafficked backlinking sites with a ramp-up in volume in less than a year. This has allowed them to overtake all of their competitors in the same time period (based on volume while maintaining quality).

What about the velocity in which authority has been added? In the following, we'll plot the authority velocity over time for each website:

```
competitor_capital_veloc_plt = (
    ggplot(competitor_capital_cumu_df, aes(x = 'month_year', y =
    'auth_velocity',
     group = 'site', colour = 'site')) +
    geom_line(alpha = 0.6, size = 2) +
    labs(y = 'Authority Capital', x = 'Month Year') +
    scale_y_continuous() +
    scale_x_date() +
    theme(legend_position = 'right',
          axis_text_x=element_text(rotation=90, hjust=1)
         ))

competitor_capital_veloc_plt.save(filename = 'images/6_auth_veloc_smooth_
plt.png',
                            height=5, width=10, units = 'in', dpi=1000)

competitor_capital_veloc_plt
```

The only standouts are David Linley and Hadley Rose (Figure 5-16). Should David Linley maintain the quality and the velocity of its link acquisition program?

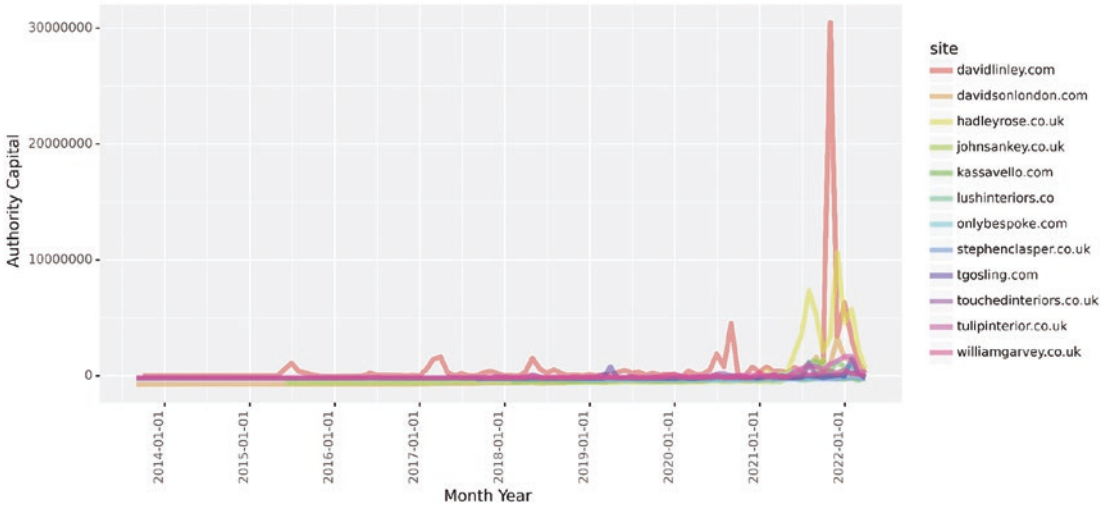

Figure 5-16. *Link capital velocity over time by website*

We're in no doubt that it will catch up and even surpass Hadley Rose, all other things being equal.

Finding Power Networks

A power network in SEO parlance is a group of websites that link to the top ranking sites for your desired keyword(s). So, getting a backlink from these websites to your website will improve your authority and thereby improve your site's ranking potential.

Does it work? From our experience, yes.

Before we go into the code, let's discuss the theory. In 1996, the quality of web search was in its infancy and highly dependent on the keyword(s) used on the page.

In response, Jon Kleinberg, a computer scientist, invented the Hyperlink-Induced Topic Search (HITS) algorithm which later formed the core algorithm for the Ask search engine.

The idea, as described in his paper "Authoritative sources in a hyperlinked environment" (1999), is a link analysis algorithm that ranks web pages for their authority and hub values. Authorities estimate the content value of the page, while hubs estimate the value of its links to other pages.

From a data-driven SEO perspective, we're not only interested in acquiring these links, we're also interested in finding out (in a data-driven manner) what these hubs are.

To achieve this, we'll group the referring domains and their traffic levels to calculate the number of sites:

```
power_doms_strata = competitor_ahrefs_analysis.groupby(['domain',
'traffic_']).agg({'rd_count': 'count'})
power_doms_strata = power_doms_strata.reset_index().sort_values('traffic_',
ascending = False)
```

A referring domain can only be considered a hub or power domain if it links to more than two domains, so we'll filter out those that don't meet the criteria. Why three or more? Because one is random, two is a coincidence, and three is directed.

```
power_doms_strata = power_doms_strata.loc[power_doms_strata['rd_
count'] > 2]

power_doms_strata
```

This results in the following:

	domain	traffic_	rd_count
3763	sitelike.org	14357011	11
1827	idcrawl.com	23024	10
3766	siteprice.org	1291812	9
1	1-2-3-4-5.com	25	9
1337	firmania.co.uk	2485	8
...	...	...	...
479	bizify.co.uk	4523	3
3891	storyandtoy.com	0	3
1748	homify.es	113111	3
1518	goldsir.ru	3	3
673	cgmood.com	6167	3

156 rows × 3 columns

The table shows referring domains, their traffic, and the number of (our furniture) sites that these backlinking domains are linking to.

Being data driven, we're not satisfied with a list, so we'll use statistics to help understand the distribution of power before filtering the list further:

```
pd.set_option('display.float_format', str)
power_doms_stats = power_doms_strata.describe()

power_doms_stats
```

This results in the following:

	traffic_	rd_count
count	156.0	156.0
mean	1015822.032051282	3.9038461538461537
std	7774169.411800991	1.4669645911413565
min	0.0	3.0
25%	0.75	3.0
50%	405.0	3.0
75%	51093.75	4.0
max	94579570.0	11.0

We see the distribution is heavily positively skewed where most of the highly trafficked referring domains are in the 75th percentile or higher. Those are the ones we want. Let's visualize:

```
power_doms_stats_plt = (
    ggplot(power_doms_strata, aes(x = 'traffic_')) +
    geom_histogram(alpha = 0.6, binwidth = 10) +
    labs(y = 'Power Domains Count', x = 'traffic_') +
    scale_y_continuous() +
    theme(legend_position = 'right',
          axis_text_x=element_text(rotation = 90, hjust=1)
         ))
```

```
power_doms_stats_plt.save(filename = 'images/7_power_doms_stats_plt.png',
                          height=5, width=10, units = 'in', dpi=1000)

power_doms_stats_plt
```

As mentioned, the distribution is massively skewed, which is more apparent from the histogram. Finally, we'll filter the domain list for the most powerful:

```
power_doms = power_doms_strata.loc[power_doms_strata['traffic_'] > power_
doms_stats['traffic_'][-2]]
```

Although we're interested in hubs, we're sorting the dataframe by traffic as these have the most authority:

```
power_doms = power_doms.sort_values('traffic_', ascending = False)

power_doms
```

This results in the following:

	domain	traffic_	rd_count
885	dailymail.co.uk	94579570	3
3763	sitelike.org	14357011	11
3135	pinterest.co.uk	14142143	6
3865	standard.co.uk	9931955	4
4614	yell.com	6335301	4
4106	thetimes.co.uk	4977287	3
1829	idealhome.co.uk	2005577	3
1726	homesandgardens.com	1975639	3
3766	siteprice.org	1291812	9
2619	minimalis.co.id	1187798	6
1743	homify.com.mx	867528	3
2389	livingetc.com	554980	3
1740	homify.com.br	485936	3
1781	houseandgarden.co.uk	462721	4
3767	sitesimilar.net	460603	3
2613	milesia.id	440193	4
1321	find-open.co.uk	411864	8
4139	thomsonlocal.com	383648	5
1751	homify.in	341049	3
1747	homify.de	287989	3
817	countryandtownhouse.co.uk	276936	5

By far, the most powerful is the daily mail, so in this case start budgeting for a good digital PR consultant or full-time employee. There are also other publisher sites like the Evening Standard (standard.co.uk) and The Times.

Some links are easier and quicker to get such as the yell.com and Thomson local directories.

Then there are more market-specific publishers such as the Ideal Home, Homes and Gardens, Livingetc, and House and Garden.

This should probably be your first port of call.

This analysis could be improved further in a number of ways, for example:

- Going more granular by looking for power pages (single backlink URLs that power your competitors)
- Checking the relevance of the backlink page (or home page) to see if it impacts visibility and filtering for relevance
- Combining relevance with traffic for a combined score for hub filtering

Taking It Further

Of course, the preceding discussion is just the tip of the iceberg, as it's a simple exploration of one site so it's very difficult to infer anything useful for improving rankings in competitive search spaces.

The following are some areas for further data exploration and analysis:

- Adding social media share data to destination URLs, referring domains, and referring pages
- Correlating overall site visibility with the running average referring domain traffic over time
- Plotting the distribution of referring domain traffic over time
- Adding search volume data on the hostnames to see how many brand searches the referring domains receive as an alternative measure of authority
- Joining with crawl data to the destination URLs to test for
    - Content relevance
    - Whether the page is indexable by confirming the HTTP response (i.e., 200)

Naturally, the preceding ideas aren't exhaustive. Some modeling extensions would require an application of the machine learning techniques outlined in Chapter 6.

Summary

Backlinks, the expression of website authority for search engines, are incredibly influential to search result positions for any website. In this chapter, you have learned about

- What site authority is and how it impacts SEO
- How brand searches could impact search visibility
- Single site analysis
- Competitor authority analysis
    - *Link anatomy*: How R^2 showed referring domain traffic was more of a predictor than domain rating for explaining visibility
    - How analyzing multiple sites adds richness and context to authority insights
- In both single and multiple site analyses
    - Authority – distribution and over time
    - Link volumes and velocity

In the next chapter, we will use data science to analyze keyword search result competitors.

Index

A

A. Voniatis, *Data-Driven SEO with Python*, https://doi.org/10.1007/978-1-4842-9175-7

B

C

D

E

F

G

H

I, J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X, Y

Z