University of Stuttgart

Text Technology Project Report

Instructor: Andre Blessing
Students: Brandon Sorensen, King DeLaney,
Haywood Shannon, Johannes Krämer

2019

1 Elasticsearch Project

Why did we choose Elasticsearch? "You know, for search."

But seriously, Elasticsearch is a state of the art tool. It has seen extensive use in research and industry applications alike. This should be reason enough. Hearing the buzzword "Elasticsearch" certainly motivated us to move in this direction. The real fascination, however, came from watching introductory YouTube videos about Elasticsearch. Most people were so excited while talking about it that we were convinced of its power before ever trying it out.

Our approach was a heterogeneous one. Our goal was to explore, experiment, and learn about the full power of Elasticsearch without binding ourselves to a specific downstream task. As a result we used different datasets to illustrate the various capabilities of the technology:

- A dataset of reported UFO sightings around the world
- A dataset of different types of beer
- A dataset containing political and census data at a county level in the US

2 The ELK Stack

The ELK Stack is comprised of Elasticsearch, Logstash, and Kibana. Often when people refer to Elasticsearch on its own, they are actually referring to the combination of these three technologies. Together they form a pipeline where Elasticsearch as a database is the core technology.
If we follow the chronological flow of data, Logstash is the first section of the stack that we encounter. It servers as a preprocessing tool that helps to properly

Figure 1: The Logstash configuration file which was used to preprocess the UFO dataset.

shape the data. The second tool is Elasticsearch itself, which stores and indexes the data. The third tool is Kibana, which provides a Graphical User Interface (GUI) to Elasticsearch's data and is responsible for visualizing the data.
The ELK Stack is available for personal use free of charge. It can also be employed professionally by using it as Software as a Service (SaaS) in the cloud or by integrating it in company server centers. With the last two approaches the above mentioned technologies are easily scalable to virtually any size of data/storage, computing power.

2.1 Logstash

As mentioned above, we can view Logstash as a preprocessing tool. It prepares the data and sends the transformed data to a specified Elasticsearch instance. It can also receive data from many kinds of sources in many formats from databases, HTTP-streams, .csv files, JSON files, Logfiles, among others.
Filters help to parse and transform the data into a desired format, making it possible to achieve more structure, consistency, readability, etc.
In the next paragraph the configuration file which was used for the UFO-sighting project will be explained (see Figure 1).
The UFO-sightings are stored in a .csv file. As such the configuration file contains definitions specific for the .csv format.
In lines 0-5 the input is defined (here the .csv file). Once dynamically read from the standard input [1] (stdin), and once with a hard-coded path to the .csv file.
In lines 7-31 different filters are defined that deal with the content of the .csv file:

[1] this means that the path to the file needs to be passed as an argument when running logstash

The .csv filter (lines 8-12) specifies the structure of the original .csv file. In this case the .csv file is comma separated (as opposed to tab separated), has the following columns (see image), and the header is skipped because it is not part of the actual data.
The mutate filters (lines 13-27) actually applied some changes to the data: In line 14 we see a new field *location* is constructed out of the existing *latitude* and *longitude* fields. The reason for that is that in order to represent geo-coordinates Elasticsearch requires geographical data to be stored as a tuple in a single field instead of their own separate fields.
On line 15 a couple of fields are removed, which we deemed irrelevant for purposes of our data analysis. For example, geographical information is now stored in the *location* field, so the original *lalitude*- and *longitude*-fields are not needed anymore.
The *gsup* mutation (lines 16-23) is a regular expression based string substitution. The first column defines the source field, the second column defines the regular expression, and the third field is the replacement string.
In this case the language codes are transformed into the original country names just for the sake of readability.
The last substitution is a little hack to deal with a specificity of the UFO-dataset. In the UFO-dataset, midnight was encoded as 24:00. However Elasticsearch's time format goes from 00:00:00-23:59:59. Although it is more common to represent midnight as 00:00, sometimes 24:00 is used to represent the end of the day and 00:00 the beginning of the day. Since the original midnight encoding was 24:00 (end of day) it was decided to remove one minute instead of adding one, such that the time still represents the end of a day instead of the beginning of the next day.
The date filter (lines 28-30) deals with different date representations. Elasticsearch requires a consistent date-time representation to be automatically recognized as such. Therefore all possible date combinations[2] must be given to the date filter. The first entry is the column name of the date field followed by the different possible combinations of date representation.

From line 32-42 the output is defined. This means defining where the transformed data is sent to: In line 33 debugging information is sent to the standard output (stdout) - e.g. the console. From line 34-41 the Elasticsearch target instance is defined together with the index the data should be stored at. In this case (line 36) Elasticsearch runs locally (localhost) and the index (line 37) is called *ufoindex*.
As mentioned earlier Elasticsearch can be used as a service in the cloud. In line 35 an Elasticsearch target instance can be seen which was hosted on Amazons AWS Cloud-system.[3].

[2] A day represented with one digit. A day represented as two digits. A month represented as one digit. A month represented as two digits.

[3] A free trial was used in order to have all the material for the presentation ready in one place. The trial (and hence the url) is not active anymore.

```
GET beers/_search
{
  "query":{
    "match_all": {
    }
  },
  "size":0,
  "aggs": {
    "Methods": {
      "terms": {
        "field": "Style.keyword",
        "size": 100
        }
    }
  }
}
GET beers/_search
```

Figure 2: Example for the Elasticsearch Query Language

2.2 Elasticsearch

As detailed above, Elasticsearch is a database and a search engine. The following query and search examples were made with the beer dataset.

2.2.1 Query Language

Elastic search queries use a structure similar to JSON (see Figure 2). All features of the query are nested within within the "query" field, and sub-fields such as "match", "bool", and "wildcard" are there to match the needs of the user. Within the query, filters, modifications, and precision adjustments can also be implemented. Attributes of the query such as "size" are beneath and not nested directly in the original query. The size field will control the number of results of the query or "buckets" during aggregation. Additionally, actions such as aggregations are also outside and below the query. Additional aggregations, can also be added to the end or within other aggregations.

2.2.2 Mapping

Data being stored in Elasticsearch is a stringified JSON format. Mappings (see Figure 3) are needed to provide the structure to values that are to be extracted and evaluated in methods other than those of plain text. JSON Mappings allow for basic types beyond strings such as integers, dates, Boolean values, and range

```
PUT beers
{
  "mappings": {
    "recipe": {
      "properties":{
        "ABV": {"type":"integer"},
        "BoilTime": {"type":"integer"},
        "OG": {"type":"integer"},
        "FG": {"type":"integer"},
        "Color": {"type":"integer"},
        "IBU": {"type":"integer"},
        "Size(L)":{"type":"integer"}
      }
    }
  }
}
```

Figure 3: Example for a Mapping

data. There are also more complex types, like arrays and dictionaries (or other JSON structured data), and additionally, there more specific data types such as Geo-point and IP data types. For values that are integers or a more complex data type and would like to be use for searches, aggregations within a range, plotted over a geographical map a mapping of the index must be used to create the index. In this mapping, the index "beers" has seven fields where the values will be type "integer" the remaining as well as any additional fields will be recognized as type "string".

2.2.3 Basic Search

The basic search allows for matching a value to specific fields or just searching over the whole of each document in the index. In the queries (see Figure 4), there is a basic search for all items in the index and a query for the string value "saaz" from the specific field "recipe" . The second search also has limit to which fields of the document within the index will be presented within the query results. By selecting "source", we specified the fields "recipe", "ABV", and "Style" to be present in the query results.

Figure 4: Example for a basic search usecase

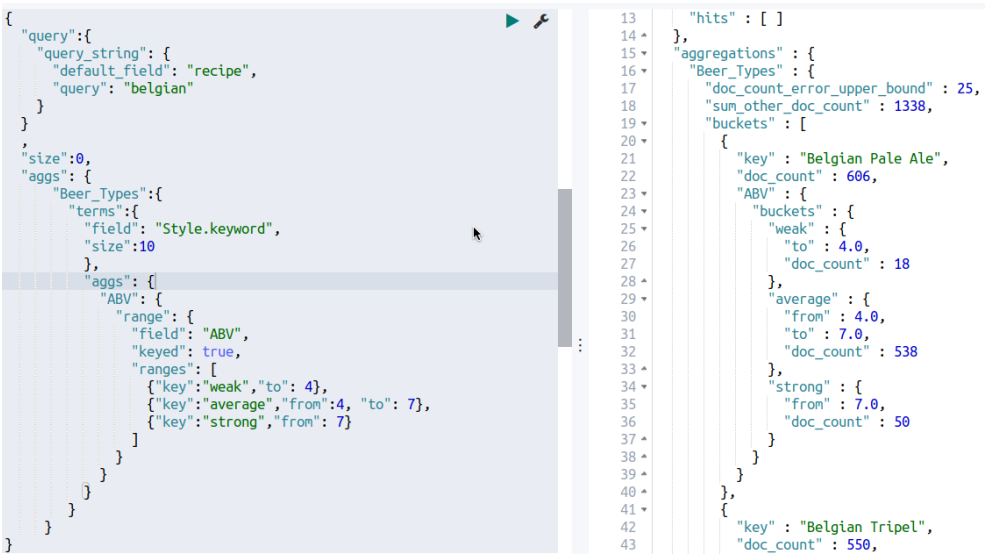

Figure 5: Example for aggregations

Figure 6: Example for weighted and wildcard queries

2.2.4 Aggregations

Elasticsearch allows for quick aggregations from a variety of different means. Aggregations can made on unique field values or ranges as well as a filter value that can be applied to a given aggregation. Limits can also be set with the "size" field within the query to limit or expand the number of query and/or aggregation results. In this query (see Figure 5), we have utilized the unique values of within the field "Style" to first aggregate the result documents of a string-query for the value "belgian" . The number of styles was limited to ten style "buckets", and within those buckets, we were once again able to aggregate them according the range value of "ABV". We have only three ranges here, but the limit is at least one, with an indetermined upper-bound limit.

2.2.5 Weighted and Wildcard Queries

Elasticsearch allows for weighting or boosting the significance of a query term. In the a query, the "^" symbol is used to begin the boosting of a specific term. This boosting can occur over many values and each will adjust the weighted ranking of the result documents. In this instance, we made the value of "centennial" three times more significant in the ranked results than the value "cascade". Wildcard queries are also easily performed in Elasticsearch. The "*" character is used for wildcard queries. In the query (see Figure 6), we used a wildcard for the value "Mittelfrüh". Terms with characters that are less common, in this case the "u" with an umlaut, might return results with correct and incorrect spellings of the term.

2.2.6 Proximity and Precision

A feature of Elasticsearch text search is proximity searches. The use of boundary "\" and quotes can define the values and a number outside will define the proximity space between the two terms. In the query (see Figure 7), it queries for "Hops:" and "Other:" separated by no more than one additional term between. Additionally, precision within the searches can be improved by targeting

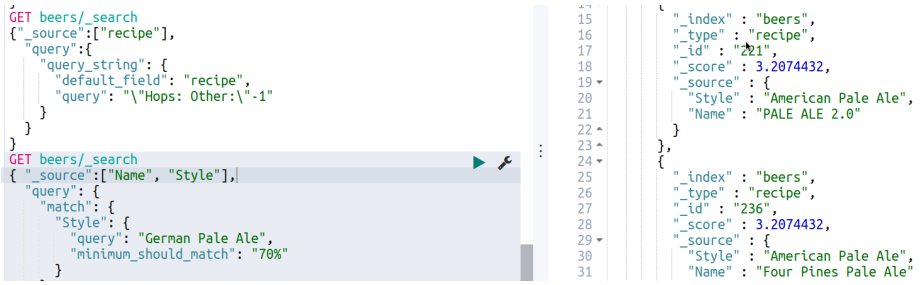

Figure 7: Example for proximity and precision

			cream-ale																
	2	Southern Tier Pumking clone	/homebrew/recipe/view/16367/southern-tier-pumking-clone	Holiday/Winter Special Spiced Beer	85	20.82	1.083	1.021	8.16	60.65	15.64	24.61	60	1.07	70	N/A	Specific Gravity	All Grain	N/A	N/A
	3	Zombie Dust Clone - EXTRACT	/homebrew/recipe/view/5920/zombie-dust-clone-extract	American IPA	7	18.93	1.063	1.018	5.91	59.25	8.98	22.71	60	N/A	70	N/A	Specific Gravity	extract	N/A	N/A

Figure 8: An excerpt of the beer dataset

a minimum match. Using the field "minimum-should-match", a string percentage value is given by which Elasticsearch tries to match to as a minimum. The percentage may be decreased to the nearest evenly distributed percentage based on the length of the query. In our query "German Pale Ale", our likely distribution is that each word constitutes 33.33% of the query. If we match to "70", it will try to match at least 66.67%. Within the results, "Pale Ale" is most likely to occur and returns many results with style "American Pale Ale".

2.2.7 Beer Dataset

The Brewer's Friend Beer recipe dataset (see Figure 8) that was selected started from Kaggle.com. It contained several fields already, including several aspects of a beers color, alcohol content, style, etc. The dataset was, however, lacking in areas involving methodology and ingredients. To make a more comprehensive and interesting dateset, more data needed to be gathered. For each entry, there was a link the recipe page where more information about the recipe could be ascertained. Extracting the data for each recipe would be the most effective way to compile more data into this data set.

2.2.8 Crawler

For this particular website, the recipe data was in several tables throughout the page. The corresponding data was also located in meta tags within the Html.

```python
# -*- coding: utf-8 -*-
import scrapy
import pickle
from beers.items import BeersItem
with open('/home/nadezh/Documents/Text Technology/project Beer/url_gen.pkl', 'rb') as f:
    urls = pickle.load(f)

class RecipeSpiderSpider(scrapy.Spider):
    name = 'recipe_spider'

    def start_requests(self):
        allowed_domains = ['https://www.brewersfriend.com']
        start_urls = [allowed_domains[0]+i for i in urls]
        headers = {'User-Agent': 'Mozilla/5.0 (X11; Linux x86_64; rv:48.0) Gecko/20100101 Firefox/48.0
        for url in start_urls:
            item=BeersItem()
            item['url'] = url.replace('http://www.brewersfriend.com','')
            yield scrapy.Request(url=url, meta=item, headers=headers, callback=self.parse)
    def parse(self, response):
        item=response.meta
        try:
            item['recipe'] = response.xpath('//meta[@property="og:description"]/@content').extract()
        except:
            item['recipe'] =''
        yield item
```

Figure 9: Scrapy Spider Code

```
recipes[9]['recipe']
```

```
'Method: All Grain, Style: Imperial IPA, ABV 8.88%, IBU 98.09, SRM 8.55, Fermentables: (Pale 2-Row, Caramel / Cryst
al 20L, Flaked Wheat, Rolled Oats) Hops: (Magnum, Centennial, Citra, Zythos) Other: (Whirlfloc Tab, Pureed Frozen M
ango, Pureed Habanero Pepper, Organic Mango Juice, Sliced Habanero Pepper) Notes: Add the pureed mango and habanero
to the boil with about 5 minutes left. When transferring to the primary do NOT strain the wort. One week in primary
Add mango juice, sliced habanero and dry hops when transferring to the secondary. Two Weeks in secondary OG 1.065 F
G 1.009 ABV 7.35% OG 1.080 - lots of sediment FG 1.012 ABV 8.9% Batch - San Diego Super Yeast - Very little pepper
flavor or heat OG 1.070 FG 1.010 ABV 7.9% OG 1.084 FG 1.012 ABV 9.5%'
```

Figure 10: Example of self-extracted beer data

This spider was constructed via Scrapy (see Figure 9). It sifted through the list of urls created from the Brewer's friend website and the recipe extension from the dataset. It extracted the attribute data from the meta tags. This data contained some of the data found in the original data set as well as various types of ingredients in up to four different categories and notes that the author felt necessary. All data was extracted to CSV files in smaller batches as the process of scraping 73,000 recipes took over 70 hours.

2.2.9 Extraction

As stated, some of the data that was extracted was identical to that which came in the original dataset. The additional data (see Figure 10) here often contained the author Username, fermentable ingredients, Steeping grains, Hops selection, additional ingredients, and notes from the author. This format was suitable for Elasticsearch full text queries, but it was limited in its use for further data analysis outside of aggregation counts of terms that occurred in the data.

```
|: {'Author': ' Stikks 2-2-12, Method: All Grain, Style: Cream Ale, ABV 5.48%, IBU 19.44, SRM 4.83',
 'Fermentables': ['pale 2-row',
  'white wheat',
  'pale 6-row',
  'flaked corn',
  'caramel',
  'crystal 20l',
  'carapils',
  'flaked barley',
  'honey'],
 'Hops': ['cascade', 'saaz'],
 'Other': ['pure vanilla extract',
  'yeast nutrient',
  'whirlfloc',
  'vanilla beans - in 2oz vodka'],
 'Notes': ' 1600 ml. starter at 1.04 Ferment at 58 to 60 degrees with Wyeast 2565,62 to 64 degrees with WLP 029 or
65 to 67 degrees with US-05 attenuation better than expected finished at 1.011 about 5.3 ABV carbonate 3 weeks at 6
8-70 degrees cold condition 1 week in fridge frothy head,nice lacing tasty summer beer,easy drinking. A great beer
for the female in your life. I`ve made this brew 7 times now,and it is a mainstay for my woman.I`ve used US-05 WLP
029 and WYEAST 2565 all with great results.Used Williamette in place of saaz and prefer it.Last batch came out at 6
.0 ABV with a modification to my fly sparge setup.Grain bill holds at 11 lbs. Last batch was with WLP 029,I think W
yeast 2565 was the best of the 3 yeast types that I have used. . Made batch #8 with US-05 on 4-19-13 used 4 Madagas
car vanilla beans soaked in vodka for a week and added to secondary It added a bit more vanilla flavor.Nice additio
n for added flavor.Will make this a regular step in future batches,recipe has not changed but 2565 rules the brew.
I made batch #9 on 9-22-13.Used original recipe with wyeast 2565, by far the best. Bottled on 11-1-2013.The only tw
eak was adding 12 oz clover honey at flameout.The Vanilla beans in secondary are the bomb. Recipe will not change f
rom here on out. This batch started at 1.057 and finished at 1.01 - 5.51 ABV.It is exactly where I Want it. SWMBO a
nd my friends love it.I like it alot too but I am more of a stout porter guy.I hope you like it and have good luck
with it if you try it. -enjoy- 12-8-2013 Vanilla flavor has mellowed nicely,lacing last`s till the end of the glass
.Very pleased with this batch. Batch #10 1-23-2015 recipe has not changed will be bottling in about a week or so. T
his brew is our house brew all year,it`s one for the ladies.Will be ready for St. Paddy`s day. batch #10 is long go
ne.Finally retired and will be moving to Florida in 2 weeks.I will update next batch when I brew in in the heat for
the first time,Goodbye Chicago.I hope you enjoy the recipe.-Will be back to brewing soon. -Cheers-'}
```

Figure 11: Example of a beer datapoint

2.2.10 Data Processing

Beyond extraction, there was a significant amount of cleaning and preprocessing that need to be done to make it more usable for analysis (see Figure 11). From the sing string that was extracted, there were several aspects that should have been extracted, such as ingredients and notes. There were four different categories of ingredients to extract and in addition, much of the data contained typos and short hand for different ingredients and descriptions in the notes. Much further refinements would also be necessary to make the data more presentable.

2.3 Kibana

The previous section showed the speed, flexibility, and powerful search capabilities of the ELK stack. Yet technical power alone rarely suffices. In order to work with data one has to have a good intuition of it. Since the human brain is better at interpreting images than a matrix of values, a common approach for better understanding data is to visualize it.

Kibana is a simple to use, yet powerful visualization tool that is built on top of Elasticsearch. It comes with a lot of built-in visualizations: histograms, pie charts, lines, heat maps, geographical maps, etc. These visualizations can be organized in dashboards. The dashboards are interactive, which means if one clicks on a value in a visualization (e.g. the biggest part in a pie chart) this value is used as a filter for all visualizations that are part of the same dashboard. This functionality makes exploring descriptive statistics really convenient because different aspects of the data can be looked at by experimenting and clicking around in the dashboard.

Additionally Kibana provides further functionalities, like Machine Learning-

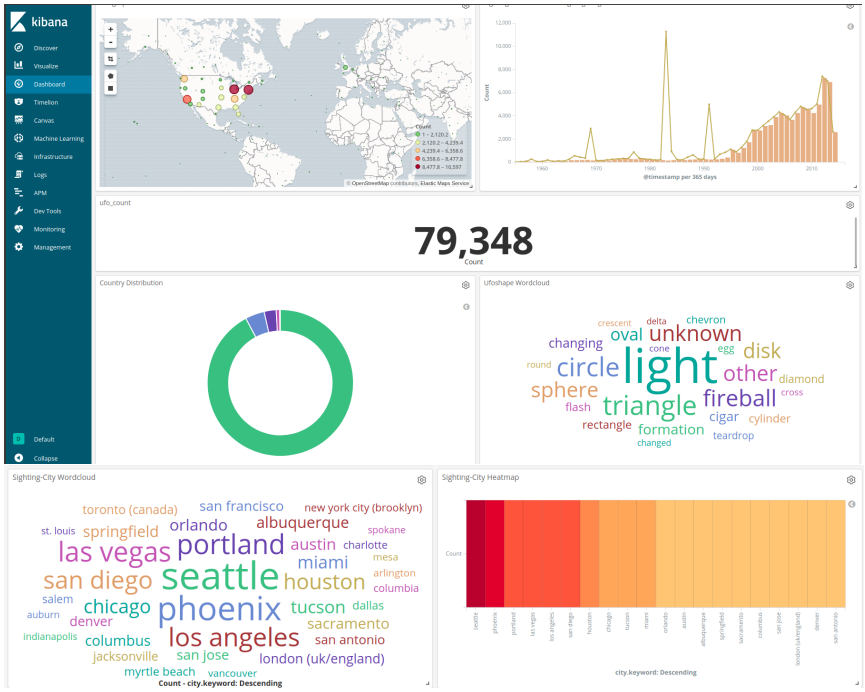

Figure 12: The dashboard for the UFO-dataset

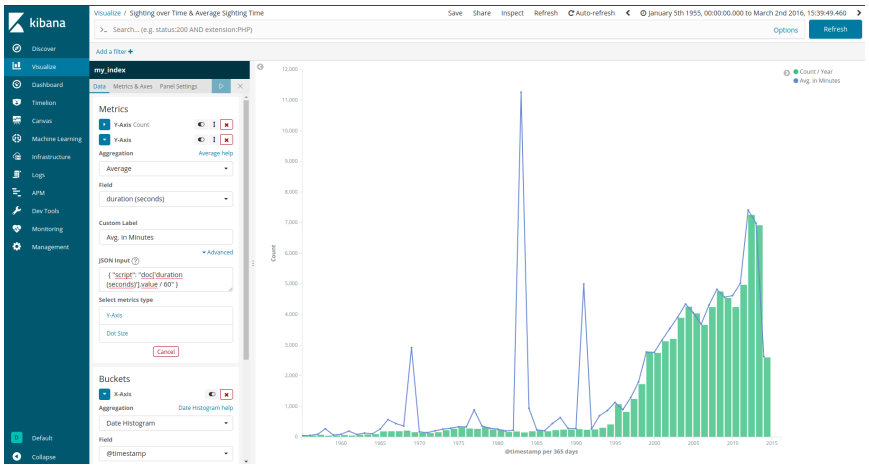

Figure 13: This is an visualization in edit mode.

based anomaly detection or customizable visualizations, that are not in the scope of this report.
But one additional functionality must be mentioned at this point. It is the ability to create online scripts that process data on the fly. In Figure 13 the *JSON Input* field is an example of such a script. This script turns seconds into minutes by dividing the current value by sixty. That way the original data need not be touched and the calculation is done only for the visualization. The drawback of this scripting functionality is that the script needs to be run every time the visualization is loaded. For huge datasets this can lead to long waiting times, especially when the scripting functionality is used in many visualizations.

In Figure 12 different visualizations are shown for the UFO dataset, which are organized into a dashboard:
In the upper left, a geographical visualization is shown. It marks all the points where UFOs were seen. Bigger circles are collections of data points. This can be seen by zooming in, then the bigger circle is divided into multiple smaller circles.
In the upper right a histogram and a line is shown. The histogram describes how many UFO sightings were recorded per year. The line is the average time (in minutes) per year that UFOs were seen[4]. Since the line visualization contains an outlier around 1964, this would be an ideal scenario for applying a filter by clicking on the outlier and inspecting the results of the other visualizations. That way it could be investigated whether something interesting happened in 1964 or an error occurred during data collection.
The next visualization below is just a simple count of data points (size of the dataset). In a live application this would be more interesting, because one could inspect how the dataset grows.
Below that on the left there is a pie chart that shows the distribution over Countries: green = America, blue = Canada, purple = Great Britain, purple = Australia, red = Germany (not visible anymore on the PDF).
On the right of the pie chart there is a frequency-based word cloud, showing the different UFO shapes that people described during the sighting. It should be mentioned that it is only possible to use a word cloud properly on (pseudo) categorical values - only one word per field. There is no preprocessing step (like tokenization) involved.
Below on the left is again a word cloud that shows roughly the distribution over cities.
To the right of the city word cloud, the information about cities is encoded as a heat map (just for the sake of showing exploring different visualizations).

As a quick summary and interpretation of the visualization the following can be recorded: most of the UFO sightings happened in the US; in 1964 there might have been an UFO boom; lights, circles, triangles, fireballs are popular

[4]The line is not on the same scale as the histogram. In order to see the actual minutes-values one needs to hover with the mouse to the point of interest.

descriptions of the look of UFOs; Western cities like Seattle, Phoenix, Portland, and Las Vegas are the most popular cities were people claim to have seen UFOs.

Indeed, the range of visualizations is wide. Our political/census dataset allowed us to explore the depth of features that Kibana provides. We experimented with many kinds of aggregation (detailed above). For reference, each data point in this dataset is a county, or a subdivision of a US state. The attributes of these data points are various census or electoral information about those counties, such as the vote distribution of various national elections or the percentage of population that is African-American. Kibana allows us to visualize the data based on any number of aggregations. We might aggregate by state or the states by the percentages of their populations that live in rural areas on the x-axis, then plot their presidential voting records on the y-axis. This gives us insight into the voting trends of rural voters.

These aggregations can be easily created in the Kibana UI via simple drop-down menus and other tool bars pictured below. For this example, data points are aggregated based on "Terms", here the field "State", which every "county" data point contains. The effect is that of SQL's "GROUP BY" keyword. All counties are grouped on their state (or district in the case of Washington, D.C.). The aggregations are ordered by the average rural percentage. (Washington, D.C. is omitted as it is a single city and as such its rural population is effectively zero. On the y-axis, the two fields represented the percentage of that went to Barack Obama in the 2012 Presidential Elections and that that went to Donald Trump in 2016 are plotted. This allows us to see the voting trends of rural voters and we accomplished it with roughly half a dozen simple drop-down menus.

This is a single example. One might bucket counties by their voting records or their Hispanic population and derive data from those buckets. One could aggregate by state and sub-aggregate by the voting records.

Another useful feature is 'control' in Kibana dashboard. It is essentially a filter function to visualize a subset of the dataset. It is a useful interactive tool to investigate the relationships between different fields in the dataset.

For example, the most common hops in whole beer recipe dataset is "cascade". (see Figure 16) After selecting the "Pilsner" in the "Fermentable" filter (Figure **??**), the distribution of the resulting dataset is altered, showing that the most common hops for "Pilsner" is "saaz", instead of "cascade" (Figure **??**).

3 Conclusion

Perhaps the fourteenth page of a report is not such a convincing time to boast of brevity, but it is indeed just that — the relative conciseness of this report — that best testifies to the power of Elasticsearch and the ELK stack more broadly. In the space of just a few pages we were able to take various raw data, manipulate them to our liking, tease out both superficial and complex features

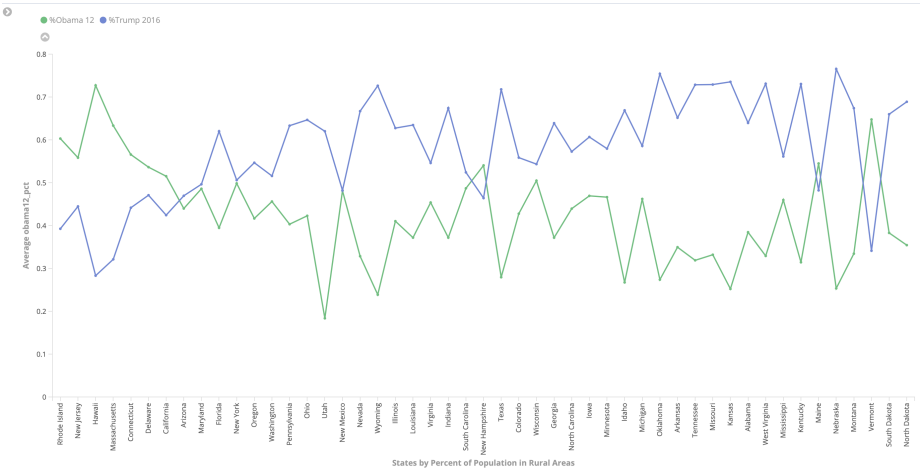

Figure 14: Voting trends of states based on their rural populations

thereof, and present it all in a manner both aesthetically appealing and informationally dense without sacrificing the complexity or nuance of the findings. What's more, our results lend themselves to easy replication in the future. It is these reasons, among a plethora of others, that have caught the attention of industry executives and academics.

Yet many of Elasticsearch's most appealing properties do not properly come through in a written report. Such marquee features as the breakneck speed of its Lucene-based back-end or its impressive scalability simply cannot be done justice here on the page but are best appreciated during implementation. However, we believe that the above précis of our research sufficiently demonstrates the many advantages of Elasticsearch and how they can be leveraged to the benefit of real world applications.

4 Personal Reflection

Here we can put our personal opinions and say what we liked most.

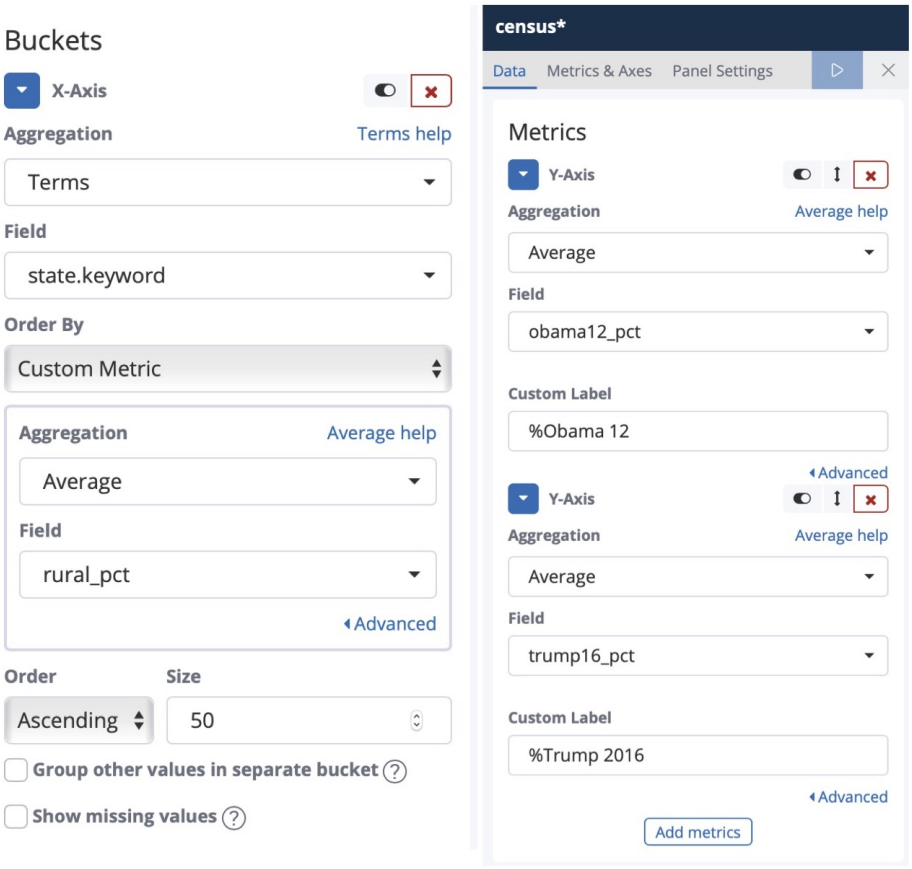

Figure 15: The settings of rural voter visualization

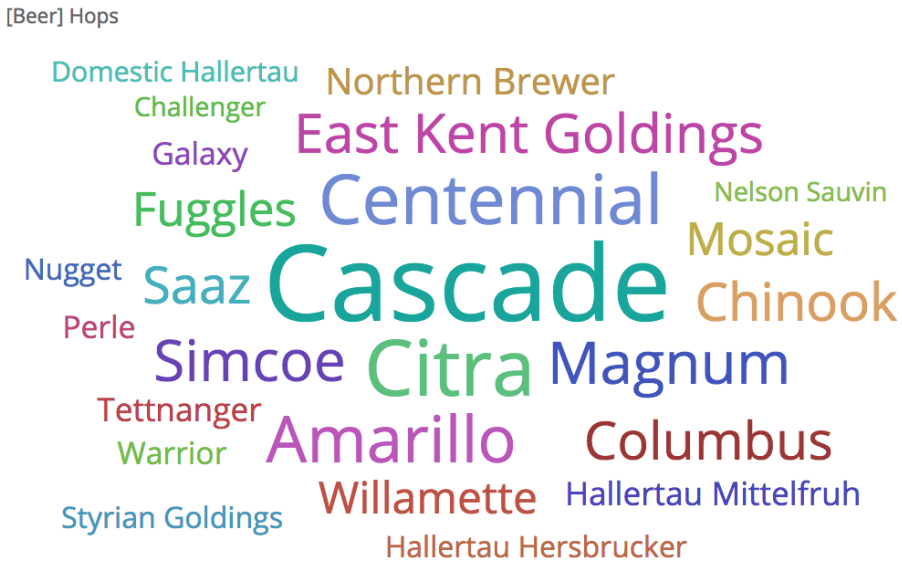

Figure 16: Word cloud for "hops" field for the whole dataset on Kibana Dashboard