Visual Analytics – Assignment 2 Report

ElasticSearch, Kibana and Canvas: Analysis of the Restaurant Dataset

Airaghi Luca
Università della Svizzera italiana

Submission Date: 28 April 2025

Contents

1 Indexing

1.1 Dataset Preparation

I began my assignment by examining the data. I opened the original CSV file in VSCode, using an extension to preview its structure. Then, using Pandas, I wrote a small script, to convert the CSV data into NDJSON format. After that, I used a command provided by the professor in class to ingest the data into Elasticsearch. The following is the script I used, with comments included to clearly explain each step.

1.2 Scripts Used

Python Script to clean CSV File

```
import pandas as pd
import json
import ast

#Load CSV file
df = pd.read_csv('dataset/restaurants.csv', sep=";", skiprows=1)

#Rename columns
df.columns = ["ID", "RestaurantName", "AverageCostForTwo", "
   AggregateRating", "RatingText", "Votes", "Date", "Coordinates", "
   City/County/Continent"]

#Cast data types
df["Votes"] = df["Votes"].astype(int)
df["AverageCostForTwo"] = df["AverageCostForTwo"].astype(int)
df["AggregateRating"] = df["AggregateRating"].astype(float)

#Add columnn City, County, State for query #3
split_location = df['City/County/Continent'].str.split('/',expand=True)

#Ensure it has exactly 3 columns, filling missing values if needed
while split_location.shape[1] < 3:
    split_location[split_location.shape[1]] = None
df[['City', 'County', 'Continent']] = split_location.iloc[:, :3]

#Convert stringified coordinate list to {"lat": ..., "lon": ...}
def convert_coordinates(coord_str):
    try:
        coords = ast.literal_eval(coord_str)
        return {"lat": float(coords[0]), "lon": float(coords[1])}
    except Exception as e:
        print(f"Errore␣parsing␣'{coord_str}':␣{e}")
        return None

df["Coordinates"] = df["Coordinates"].apply(convert_coordinates)
df = df[df["Coordinates"].notnull()]

#Prepare NDJSON file for Elasticsearch bulk
with open('dataset/restaurants_elastic_bulk.json', 'w') as outfile:
    for _, row in df.iterrows():
        index_line = json.dumps({"index": {}})
        data_line = json.dumps(row.to_dict(), default=str)
        outfile.write(f"{index_line}\n{data_line}\n")
```

Listing 1: CSV to ElasticSearch JSON conversion

Explanation: In this Python script, I load the CSV file using `pandas` and rename the columns to remove leading white spaces and standardize the names.

I then split the `City/County/Continent` field into three separate columns: `City`, `County`, and `Continent`.

The `Coordinates` field, originally stored as a stringified list, is converted into a dictionary format with `"lat"` and `"lon"` keys, which is compatible with Elasticsearch's `geo_point` data type.

Finally, I iterate over the cleaned DataFrame and write each record in `NDJSON` format, including the necessary index metadata for bulk ingestion into Elasticsearch.

1.3 Ingestion Process

To ingest the file I use the following command:

```
curl -k --user "elastic:password" \
  -X POST "https://localhost:9200/restaurants_v5_fixed/_bulk" \
  -H "Content-Type:application/x-ndjson" \
  --data-binary "@restaurants_elastic_bulk.js"
```

Listing 2: Ingest data into Elasticsearch

Breakdown of the Command:

- `curl`:
  `curl` is a command-line tool to transfer data with URLs. In this case, it is used to interact with the Elasticsearch server over HTTP(S).

- `-k`:
  This option tells `curl` to allow insecure SSL connections and bypass certificate validation. It is typically used for testing purposes when the server is using self-signed certificates or in environments where the SSL/TLS configuration is not set up correctly.

- `--user "elastic:password"`:
  This flag provides authentication credentials to the Elasticsearch server. In this case, it is using the `elastic` user with the password `password`. If you have a different user or password, you should replace these values accordingly.

- `-X POST`:
  This option specifies the HTTP request method, in this case, `POST`. The `POST` method is used to send data to the server, which in this case is used for bulk insertion into the `restaurants` index.

- `"https://localhost:9200/restaurants_v5_fixed/_bulk"`:
  This is the URL of the Elasticsearch server. It specifies the **local Elasticsearch instance** (`localhost`) running on port 9200. The `/restaurants/_bulk` part of the URL indicates that the data will be inserted into the `restaurants` index, and the _bulk endpoint is used for bulk operations.

- `-H "Content-Type: application/x-ndjson"`:
  This flag sets the **Content-Type** of the request to `application/x-ndjson`. NDJSON (Newline Delimited JSON) is a format where each line is a valid JSON object. It is commonly used for bulk operations in Elasticsearch because each line can represent a separate document.

- `--data-binary "@restaurants_elastic_bulk.js"`:
  This option sends the contents of the file `restaurants_elastic_bulk.js` as the request

body. The `@` symbol is used to indicate that the data should come from a file (in this case, `restaurants_elastic_bulk.js`). The file should contain the bulk data in NDJSON format, where each line represents an individual document that will be indexed in Elasticsearch.

Purpose:

The command is performing a bulk data insertion into Elasticsearch. By using the _bulk endpoint, multiple documents are sent to Elasticsearch in a single HTTP request, making the operation more efficient compared to inserting documents one by one. The file `restaurants_elastic_bulk.js` contains the data that is being ingested into the `restaurants_v5_fixed` index. Each line in the file represents a single document, and the data is provided in NDJSON format for bulk ingestion.

2 Section 1 – Indexing, Queries and Aggregations

To index the data, I first ingest a document to observe how Elasticsearch generates the mapping. I then adapt the mapping to suit my requirements. For instance, I convert the location field to a geo-point data type instead of a string or float, which is assigned by default. After adjusting the mapping, I ingest the entire dataset into Elasticsearch and begin performing queries and aggregations.

2.1 Query 1: 'bar' in name but not 'barbecue/barbeque(s) or bar'

This query searches for restaurant names containing the word "bar" using a wildcard, but explicitly excludes names with "barbecue", "barbeque", or the exact word "bar" to avoid false positives.

```
GET restaurants_v5_fixed/_search
{
  "_source": ["RestaurantName", "City/County/Continent", "Votes"],
  "query": {
    "bool": {
      "must": [
        {
          "wildcard": {
            "RestaurantName.keyword": {
              "value": "*bar*"
            }
          }
        }
      ],
      "must_not": [
        {
          "wildcard": {
            "RestaurantName.keyword": {
              "value": "*barbecue*"
            }
          }
        },
        {
          "wildcard": {
            "RestaurantName.keyword": {
              "value": "*barbeque*"
            }
          }
        },
        {
          "match_phrase": {
            "RestaurantName": "bar"
          }
        }
      ]
    }
  }
}
```

Listing 3: Query 1

The query returns a total of 44 matching documents, meaning there are 44 restaurants whose names contain the substring "bar", but do not contain "barbecue", "barbeque" or exactly match

"bar" as a phrase. The result page include the top 10 matches. (I will put only the first 3 just to avoid a long list). Each hit includes:

- RestaurantName
- Votes
- City/Country/Continent

The first three results returned are:

- "Peribar"
- "Rhubarb Le Restaurant"
- "Kylin Skybar"

All these restaurant names contain "bar" somewhere in the name, but do not include "barbecue", "barbeque" or simply "bar" as an exact phrase. `_score` for all result is 1, which makes sense since wildcard queries don't use relevance scoring in a meaningful way unless combined with `full-text` search or ranking mechanisms.


```
{
  "took": 28,
  "timed_out": false,
  "_shards": {
    "total": 1,
    "successful": 1,
    "skipped": 0,
    "failed": 0
  },
  "hits": {
    "total": {
      "value": 44,
      "relation": "eq"
    },
    "max_score": 1,
    "hits": [
      {
        "_index": "restaurants_v5_fixed",
        "_id": "RWGxM5YBilAK5lv4USky",
        "_score": 1,
        "_source": {
          "RestaurantName": "Paribar",
          "Votes": 46,
          "City/County/Continent": "SaoPaulo/Brazil/South␣America"
        }
      },
      {
        "_index": "restaurants_v5_fixed",
        "_id": "vmGxM5YBilAK5lv4USoy",
        "_score": 1,
        "_source": {
          "RestaurantName": "Rhubarb␣Le␣Restaurant",
          "Votes": 33,
          "City/County/Continent": "Singapore/Singapore/Asia"
        }
      },
      {
```

```
        "_index": "restaurants_v5_fixed",
        "_id": "S2GxM5YBilAK5lv4USwy",
        "_score": 1,
        "_source": {
          "RestaurantName": "Doon␣Darbar",
          "Votes": 121,
          "City/County/Continent": "Dehradun/India/Asia"
        }
      }
      ...
```

Listing 4: Results from Query 1

2.2 Query 2: Top 3 Most Expensive Near Singapore (2016)

This query filters restaurants within a 20km radius of Singapore (lat 1.290270, lon 103.851959) that were active during the year 2016. It then sorts them by 'AverageCostForTwo' in descending order and returns only the top 3 most expensive ones.

```
GET restaurants_v5_fixed/_search
{
  "size": 3,
  "query": {
    "bool": {
      "filter": [
        {
          "range": {
            "Date": {
              "gte": "2016-01-01",
              "lt": "2017-01-01"
            }
          }
        },
        {
          "geo_distance": {
            "distance": "20km",
            "Coordinates": {
              "lat": 1.290270,
              "lon": 103.851959
            }
          }
        }
      ]
    }
  },
  "sort": [
    {
      "AverageCostForTwo": {
        "order": "desc"
      }
    }
  ]
}
```

Listing 5: Query 2

The query successfully returned the top 3 most expensive restaurants located within a 20 km radius of Singapore that were active in 2016. The results are sorted in descending order by the

'AverageCostForTwo' field. All results returned have "_score": null because the query used only filters (geo and date), not full-text search or relevance-based ranking, so scoring is not applied. Out of a total of 6 matching restaurants, the following three had the highest average cost:


```
{
  "took": 25,
  "timed_out": false,
  "_shards": {
    "total": 1,
    "successful": 1,
    "skipped": 0,
    "failed": 0
  },
  "hits": {
    "total": {
      "value": 6,
      "relation": "eq"
    },
    "max_score": null,
    "hits": [
      {
        "_index": "restaurants_v5_fixed",
        "_id": "u2GxM5YBilAK5lv4USoy",
        "_score": null,
        "_source": {
          "ID": 444,
          "RestaurantName": "Restaurant␣Andre",
          "AverageCostForTwo": 500,
          "AggregateRating": 3.8,
          "RatingText": "Good",
          "Votes": 33,
          "Date": "2016-01-28T14:00:04Z",
          "Coordinates": {
            "lat": 1.279419756,
            "lon": 103.8403602
          },
          "City/County/Continent": "Singapore/Singapore/Asia",
          "City": "Singapore",
          "Country": "Singapore",
          "Continent": "Asia"
        },
        "sort": [
          500
        ]
      },
      {
        "_index": "restaurants_v5_fixed",
        "_id": "xmGxM5YBilAK5lv4USoy",
        "_score": null,
        "_source": {
          "ID": 455,
          "RestaurantName": "Summer␣Pavilion",
          "AverageCostForTwo": 300,
          "AggregateRating": 3.9,
          "RatingText": "Good",
          "Votes": 34,
          "Date": "2016-05-22T10:07:17Z",
```

```
          "Coordinates": {
            "lat": 1.290800881,
            "lon": 103.8601766
          },
          "City/County/Continent": "Singapore/Singapore/Asia",
          "City": "Singapore",
          "Country": "Singapore",
          "Continent": "Asia"
        },
        "sort": [
          300
        ]
      },
      {
        "_index": "restaurants_v5_fixed",
        "_id": "wmGxM5YBilAK5lv4USoy",
        "_score": null,
        "_source": {
          "ID": 451,
          "RestaurantName": "The␣Refinery␣Singapore",
          "AverageCostForTwo": 80,
          "AggregateRating": 3.2,
          "RatingText": "Average",
          "Votes": 30,
          "Date": "2016-05-26T10:08:13Z",
          "Coordinates": {
            "lat": 1.310668316,
            "lon": 103.8621195
          },
          "City/County/Continent": "Singapore/Singapore/Asia",
          "City": "Singapore",
          "Country": "Singapore",
          "Continent": "Asia"
        },
        "sort": [
          80
        ]
      }
    ]
  }
}
```


Listing 6: Query results 2

2.3 Query 3: Pizza but not Pasta (min. Good rating)

This query finds restaurants that have 'pizza' in their name and not 'pasta'. Additionally, it ranks results that have received at least a 'Good' rating. (also includes "Very Good" and "Excellent"), sorted by descending aggregate rating.

```
GET restaurants_v5_fixed/_search
{
  "_source": ["City", "AggregateRating", "RestaurantName", "Votes"],
  "query": {
    "bool": {
      "must": [
        {
          "match": {
```

```
            "RestaurantName": "pizza"
          }
        }
      ],
      "must_not": [
        {
          "match": {
            "RestaurantName": "pasta"
          }
        }
      ],
      "should": [
        { "match": { "RatingText": "Good" } },
        { "match": { "RatingText": "Very Good" } },
        { "match": { "RatingText": "Excellent" } }
      ],
      "minimum_should_match": 1
    }
  },
  "sort": [
    {
      "AggregateRating": {
        "order": "desc"
      }
    }
  ]
}
```

Listing 7: Query 3

This query aims to retrieve restaurants with "pizza" in their name, while excluding those that mention "pasta". Additionally, it "boosts" (via should clauses) restaurants with positive ratings like "Good", "Very Good", or "Excellent". The results are then sorted by AggregateRating in descending order, highlighting the best-rated pizza places. Since the query uses filters and explicit sorting (not full-text ranking), the `_score` is null — Elasticsearch doesn't compute relevance scores unless needed. As I did in a query before, I only show the top 3 results, because otherwise the results will be too long.

```
{
  "took": 52,
  "timed_out": false,
  "_shards": {
    "total": 1,
    "successful": 1,
    "skipped": 0,
    "failed": 0
  },
  "hits": {
    "total": {
      "value": 52,
      "relation": "eq"
    },
    "max_score": null,
    "hits": [
      {
        "_index": "restaurants_v5_fixed",
        "_id": "PGGxM5YBilAK5lv4USoy",
        "_score": null,
```

```
      "_source": {
        "RestaurantName": "Ingleside␣Village␣Pizza",
        "AggregateRating": 4.9,
        "Votes": 478,
        "City": "Macon"
      },
      "sort": [
        4.9
      ]
    },
    {
      "_index": "restaurants_v5_fixed",
      "_id": "DWGxM5YBilAK5lv4UU42",
      "_score": null,
      "_source": {
        "RestaurantName": "Pizza␣Al␣Forno",
        "AggregateRating": 4.7,
        "Votes": 104,
        "City": "Ankara"
      },
      "sort": [
        4.7
      ]
    },
    {
      "_index": "restaurants_v5_fixed",
      "_id": "-GGxM5YBilAK5lv4USky",
      "_score": null,
      "_source": {
        "RestaurantName": "Fong's␣Pizza",
        "AggregateRating": 4.6,
        "Votes": 728,
        "City": "Des␣Moines"
      },
      "sort": [
        4.6
      ]
    }
    ...
```


Listing 8: Query results 3

2.4 Aggregation 1: Top 7 Cities by Avg. Price

This aggregation finds the top 7 cities (with at least 10 restaurants each) ranked by their average cost for two people. It only includes restaurants that have received 100 or more votes, ensuring statistical significance in the results.

```
GET restaurants_v5_fixed/_search
{
  "size": 0,
  "query": {
    "range": {
      "Votes": {
        "gte": 100
        }
      }
    },
  "aggs":{
    "cityUpTo10Restaurant": {
      "terms": {
        "field": "City.keyword",
        "size": 7,
        "min_doc_count": 10,
        "order": {
          "avg_cost": "desc"
        }
      },
      "aggs": {
        "avg_cost": {
          "avg": {
            "field": "AverageCostForTwo"
            }
          }
        }
      }
    }
}
```

Listing 9: Aggregation 1

This aggregation query filters the dataset to include only restaurants with 100 or more votes, then groups them by City (via terms aggregation on 'City.keyword'). It shows the top 7 cities (with at least 10 matching restaurants) sorted by their average 'AverageCostForTwo' in descending order. The results highlight cities with notable restaurant activity and higher dining costs:

```
{
  "took": 27,
  "timed_out": false,
  "_shards": {
    "total": 1,
    "successful": 1,
    "skipped": 0,
    "failed": 0
  },
  "hits": {
    "total": {
      "value": 2798,
      "relation": "eq"
    },
```

```
    "max_score": null,
    "hits": []
  },
  "aggregations": {
    "cityUpTo10Restaurant": {
      "doc_count_error_upper_bound": -1,
      "sum_other_doc_count": 2650,
      "buckets": [
        {
          "key": "Jakarta",
          "doc_count": 16,
          "avg_cost": {
            "value": 308437.5
          }
        },
        {
          "key": "Colombo",
          "doc_count": 14,
          "avg_cost": {
            "value": 2535.714285714286
          }
        },
        {
          "key": "Hyderabad",
          "doc_count": 17,
          "avg_cost": {
            "value": 1358.8235294117646
          }
        },
        {
          "key": "Pune",
          "doc_count": 20,
          "avg_cost": {
            "value": 1337.5
          }
        },
        {
          "key": "Jaipur",
          "doc_count": 18,
          "avg_cost": {
            "value": 1316.6666666666667
          }
        },
        {
          "key": "Kolkata",
          "doc_count": 20,
          "avg_cost": {
            "value": 1272.5
          }
        },
        {
          "key": "Bangalore",
          "doc_count": 20,
          "avg_cost": {
            "value": 1232.5
          }
        }
      ]
```

```
    }
  }
}
```

Listing 10: Aggregation result 1

2.5 Aggregation 2: Weighted Avg. Rating near Noida

This query calculates the weighted average rating of restaurants within 10km of Noida. Each rating is weighted by the number of votes it received, giving more influence to popular restaurants and ensuring more reliable aggregated feedback.

```
GET restaurants_v5_fixed/_search
{
  "size": 0,
  "query": {
    "geo_distance": {
      "distance": "10km",
      "Coordinates": {
        "lat": 28.535517,
        "lon": 77.391029
      }
    }
  },
  "aggs": {
    "weighted_avg": {
      "weighted_avg": {
        "value": {
          "field": "AggregateRating"
        },
        "weight": {
          "field": "Votes"
        }
      }
    }
  }
}
```

Listing 11: Aggregation 2

The average rating of restaurants near Noida, weighted by the number of votes, is approximately 3.48 out of 5, suggesting: The general restaurant quality is above average, though not outstanding.


```
{
  "took": 23,
  "timed_out": false,
  "_shards": {
    "total": 1,
    "successful": 1,
    "skipped": 0,
    "failed": 0
  },
  "hits": {
    "total": {
      "value": 1094,
      "relation": "eq"
    },
    "max_score": null,
```

```
    "hits": []
  },
  "aggregations": {
    "weighted_avg": {
      "value": 3.475335691819247
    }
  }
}
```

Listing 12: Aggregation result 2

2.6 Aggregation 3: Very Good Reviews, Not in Asia

This aggregation focuses on restaurants with 'Very Good' reviews located outside of Asia. It splits them into vote-based ranges and, for each range, returns the minimum and maximum cost for two people. This helps analyze how pricing varies with popularity across continents.

```
GET restaurants_v5_fixed/_search
{
  "size": 0,
  "query": {
    "bool": {
      "must": [
        {
          "match": {
            "RatingText.keyword": "Very␣Good"
          }
        }
      ],
      "must_not": [
        {
          "match": {
            "Continent.keyword": "Asia"
          }
        }
      ]
    }
  },
  "aggs": {
    "votes_ranges": {
      "range": {
        "field": "Votes",
        "ranges": [
          { "from": 100, "to": 250 },
          { "from": 250, "to": 400 },
          { "from": 400, "to": 550 }
        ]
      },
      "aggs": {
        "min_cost": {
          "min": {
            "field": "AverageCostForTwo"
          }
        },
        "max_cost": {
          "max": {
            "field": "AverageCostForTwo"
          }
```

```
        }
      }
    }
  }
}
```

Listing 13: Aggregation 3

This query explores pricing patterns among well-rated restaurants — those labeled with "Very Good" in their 'RatingText' — but outside of Asia. Restaurants are grouped by number of votes (a proxy for popularity), and for each group, the minimum and maximum average cost for two people is returned.

- Wide Price Range: In all vote brackets, there's a wide range of costs, indicating that popularity doesn't tightly correlate with pricing.
- High Max in Top Tier: Restaurants with 400–550 votes have the highest max cost (570) — possibly premium venues with strong reputations
- Low Entry Point: Surprisingly, even in the most popular buckets, some restaurants have very low minimum costs (10 or even 0), suggesting the presence of budget-friendly but well-rated places.
- More Mid-Vote Restaurants: The highest number of restaurants fall into the 100–250 range, suggesting it's a common zone for "Very Good" places outside Asia.


```
{
  "took": 30,
  "timed_out": false,
  "_shards": {
    "total": 1,
    "successful": 1,
    "skipped": 0,
    "failed": 0
  },
  "hits": {
    "total": {
      "value": 291,
      "relation": "eq"
    },
    "max_score": null,
    "hits": []
  },
  "aggregations": {
    "votes_ranges": {
      "buckets": [
        {
          "key": "100.0-250.0",
          "from": 100,
          "to": 250,
          "doc_count": 75,
          "max_cost": {
            "value": 535
          },
          "min_cost": {
            "value": 0
          }
        },
```

```
        {
          "key": "250.0-400.0",
          "from": 250,
          "to": 400,
          "doc_count": 61,
          "max_cost": {
            "value": 400
          },
          "min_cost": {
            "value": 10
          }
        },
        {
          "key": "400.0-550.0",
          "from": 400,
          "to": 550,
          "doc_count": 37,
          "max_cost": {
            "value": 570
          },
          "min_cost": {
            "value": 10
          }
        }
      ]
    }
  }
}
```


Listing 14: Aggregation result 3

2.7 Shard Size Discussion

I did not get the shard size problem.

3 Section 2 – Data Visualization (Dashboard & Canvas)

3.1 Dashboard

This subsection presents the development of an interactive Kibana dashboard, which aims to visualize key insights from the restaurant dataset. The dashboard highlights essential metrics such as the number of restaurants, cities covered, and average distribution. It includes dynamic visualizations like rating trends, continent-level comparisons, a geo-map of restaurant locations, and a heatmap of price and rating bins. Filters and a searchable table enable intuitive exploration and analysis of the dataset.

3.1.1 Metrics and Filters

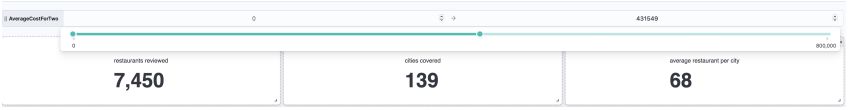

Figure 1: Price Filter, Total Restaurants, Total Cities, Average Restaurants per City

In this top part of the dashboard I firstly create some metrics like total restaurants, total cities and average restaurants per city. In Kibana to do this is very easy, you just have to select the field you want to aggregate on and choose the aggregation function.

- Total restaurants: Count of all the restaurant documents in the index. Computed as: `unique_count(RestaurantName.keyword)`
- Total cities: Count of unique values in the `City` field. Computed as: `unique_count(City.keyword)`
- Average restaurants per city: Total number of restaurants divided by the number of unique cities. Computed as: `count(ID)/unique_count(City.keyword)`

3.1.2 Continent Aggregations

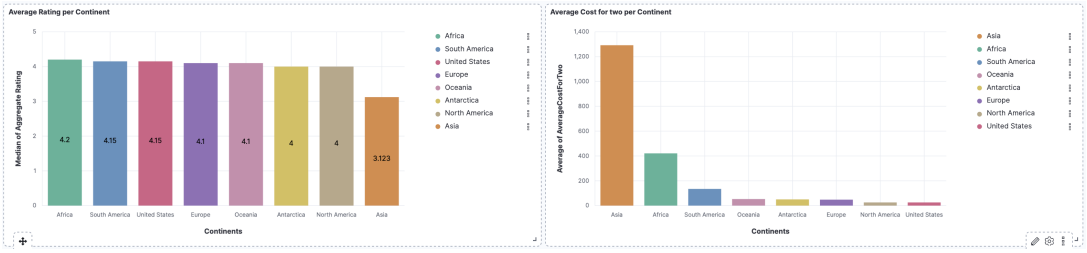

Figure 2: Bar charts of average of average cost for two per continent and average of average rating per continent

To create these two bar chart I use Lens, which allowed me to drag and drop attribute from the index and render it as a charts (combining also aggregation function as said before).

- First diagram: aims to show the average rating for each continent. To do so I set y axis equal to the average of the 'averageRatingForTwo', and on x the continent. For visual appealing each continent is assigned to a color (both graphs share same color for the same continent too). The elements are ordered based in a descending way.
- Second diagram: aims to shows the average cost for two for each continent. Here I set y axis to be the average of the averageRating, and as before on x the continents. Moreover, I

add a label to show better the value of each continent because some values are really near, and the difference cannot be seen at human eyes. Also the result as before are ordered in a descending way.

Few notes about the first charts:

- we can compare fairly the average of the price, because we do not have a currency of reference in the dataset, so 1200 in Asia maybe represent 50 euros in Europe
- elements even tough I put order descending when values are very low aren't ordered properly
- I did not put labels to show better values, because some how with these low values the label was not showing. Was showing up only for the first 3 continents, so I prefer to keep clean and coherent the chart.

3.1.3 Map

Figure 3: map shows the location of all the restaurants, with the size of each marker representing the number of votes

To do the map I did not use Lens, but maps, which provides directly the map. From there I set geo-points as position for my red marker. I then adjust the size of the marker using the field Votes, and rescale the original values becuase were too big. Then I added a tooltip field, so when a user click on a marker, is able to visualize information for the desired restaurant, like: averagePriceForTwo, TextRevies, Votes and so on.

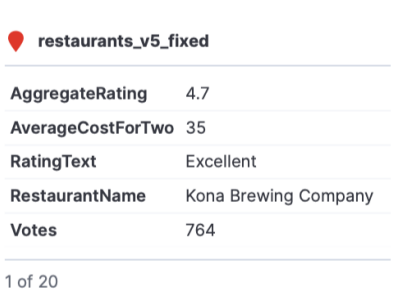

Figure 4: How the tooltip is rendered, when a user click on a restaurant in the map

3.1.4 Review Trends

This chart shows the trend of the restaurants reviews over time. We see tree line, because, we are showing:

- The number of reviews (blue line)

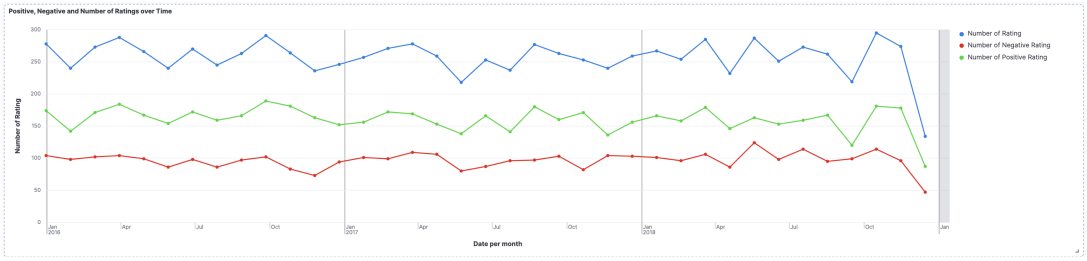

Figure 5: trend of restaurants review over time

- Positive reviews: have a score higher than 3 in green. To select positive reviews, there is not a built-in function, but I compute them as follow: `count(kql='AggregateRating > 3 and AggregateRating <= 5')`
- Negative reviews: have a score lower or equal than 3 in red. I compute them with a similar logic as before: `count(kql='AggregateRating <= 3 and AggregateRating > 0')`

3.1.5 Saved Search Table

To realize the saved search, I went to the discovery section. From there I only selected the necessary field requested to realize the map. I then save it, and added into the dashboard.

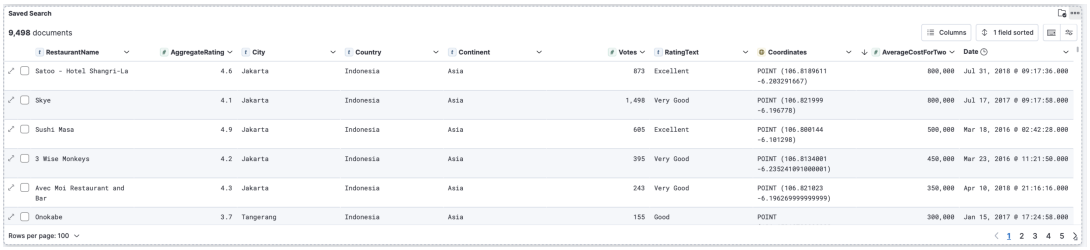

Figure 6: Saved search of restaurants statistic

3.1.6 Heatmap

For the heatmap I went under the section Lens, from there I selected the type of graph needed (so heatmap), I put as y axis the 'AverageCostForTwo', while on x axis the 'AggregateRating'. The text of exercise explicitly says that we have need to divide in bins both axis, and so I created the 20 bins over the y axis, and 5 bins on the x axis. In the map I also add a legend showing the number of votes for each bins.

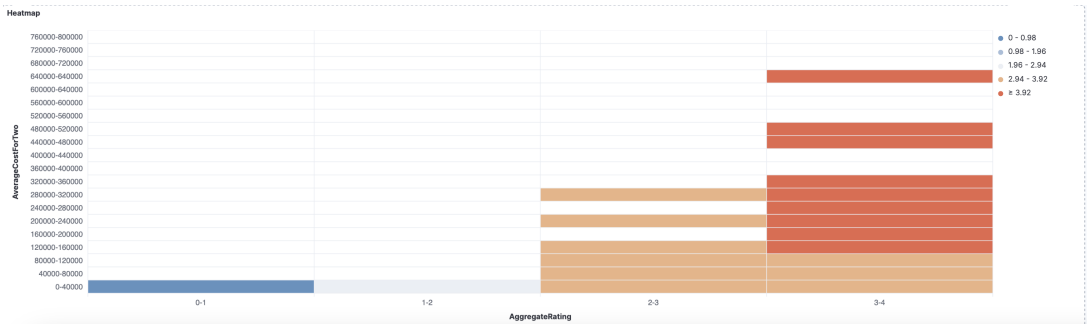

Figure 7: Distribution of Votes Across Price (Average Cost for Two) and Rating Ranges

3.2 Canvas

This section presents an interactive dashboard that visualizes restaurant ratings across cities. Users can filter data by city to explore the number of restaurants per defined price category — from Cheap to Super Pricy — based on average cost for two. It highlights the overall quality distribution of restaurants by categorizing them as Good, Medium, or Bad, according to their aggregate ratings. Additionally, a dynamic table lists key restaurant details, and high-level metrics showcase total reviews, the highest cost for two, and the most recent review date

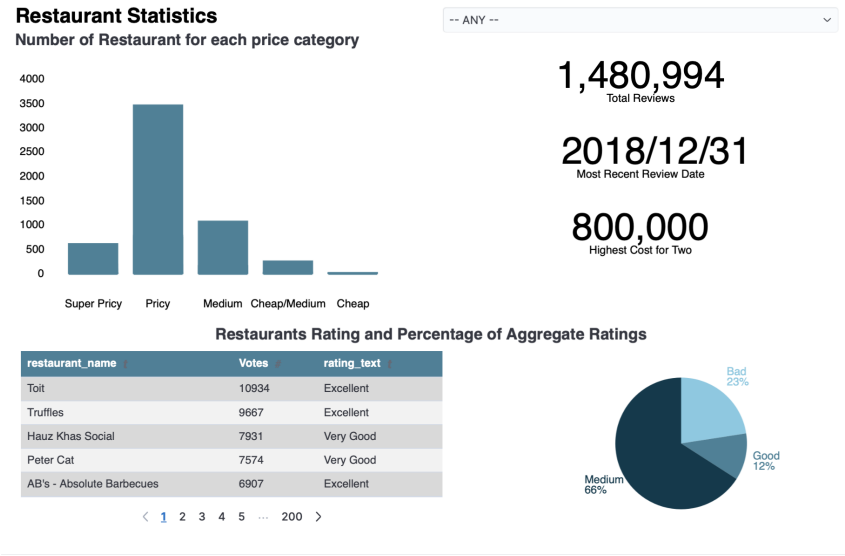

Figure 8: Visual representation of some metrics about restaurants

Filters

Dropdown filter is used to update the data in the Canvas. In the Figure 5, there is no city selected, so each of the component that are used to build the Canvas used all data in the Elasticsearch index. When I select a city all the Canvas update to show the data linked to that city. This is how it looks when I am choosing a city:

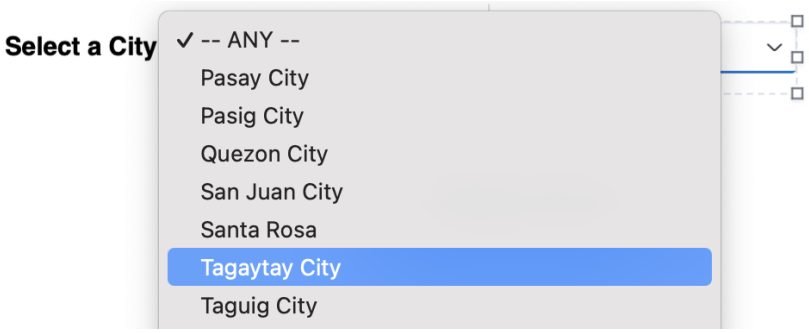

Figure 9: Visual representation of how filters appear when choosing a city

This is how I set the filter to display cities using a dropdown control in Kibana Canvas. The control filters the dashboard based on the selected city from the `City.keyword` field.

```
esdocs index="restaurants*" fields="City.keyword"
| dropdownControl valueColumn="City.keyword" filterColumn="City.keyword
   "
| render css=".canvasRenderEl{font-family:'Arial',sans-serif;padding:5
   px;
}"
```

Listing 15: Kibana Canvas expression for City filter

Metrix

The three metrix that in Figure 5 are displayed in the top right corner, are useful because they give a quick and valuable overview of the dataset's scale, pricing range, and recency:

- Total number of reviews: Indicates how much user feedback is available. A higher number suggests strong engagement and helps assess the reliability of other metrics (like average ratings).
- Highest cost for 2 people: Gives insight into the upper end of restaurant pricing. It can reveal luxury or premium options in the dataset and helps understand cost diversity across restaurants.
- Date of the most recent review: Shows how up-to-date the data is. A recent date means the dataset is current and trends or insights derived from it are more relevant

Data here are computed by doing a SQL query using the canvas query language, over the Elasticsearch index:

```
kibana
| selectFilter
| essql
query="SELECT␣MAX(Date)␣AS␣most_recent_review_date␣FROM␣restaurants"
| mapColumn "most_recent_review_date" fn={getCell "
   most_recent_review_date" | formatdate "MMM␣DD␣YYYY"}
| math "most_recent_review_date"
| metric "Most␣Recent␣Review␣Date"
| render
```

Listing 16: Kibana Canvas expression for Most recent Date of reviews metrix

Here we are using the `selectFilter` to apply the filter to the query. Inside the query, we are selecting the maximum date using `Max(date)`, saved as `most_recent_reviews_date`. Then, we format the data to show the first three letters of the month, the day, and the year, and finally render the metric.

Pie chart and Bar chart

Pie chart is great to show percentage also when we have only three division of the original pie. The main insight that we can get from this chart is to know (for a given city) how many restaurants have a good quality so are Good, and how many are considered bad. So this metrix help the user to understant the general reputation of restaurant in a certain area. Instead the bar chart explain well the average price for two for city. By categorizing restaurants from cheap to super pricy, the user can easily identify whether the market is dominated by budget-friendly places or expensive venues. Both diagram has been built by doing a query (as the one for Data metrix) over the restaurant index. Below I show the one I use to compute the bar chart, but the one for the pie chart for some aspect is very similar.

```
kibana
| selectFilter
| essql
query="
SELECT
CASE
WHEN␣AverageCostForTwo<=50␣AND␣AverageCostForTwo>0␣THEN␣'Cheap'
WHEN␣AverageCostForTwo>50␣AND␣AverageCostForTwo<=100␣THEN␣'Cheap/Medium
   '
WHEN␣AverageCostForTwo>100␣AND␣AverageCostForTwo<=250␣THEN␣'Medium'
WHEN␣AverageCostForTwo>250␣AND␣AverageCostForTwo<=1000␣THEN␣'Pricy'
WHEN␣AverageCostForTwo>1000␣THEN␣'Super␣Pricy'
END␣AS␣price_category,City.keyword,COUNT(RestaurantName.keyword)␣as␣
   n_restaurants
FROM␣restaurants
WHERE␣price_category␣IS␣NOT␣NULL
GROUP␣BY␣price_category,␣City.keyword
ORDER␣BY␣price_category␣DESC
"
| pointseries x="price_category" y="n_restaurants"
| plot defaultStyle={seriesStyle bars=0.75 color="#f2bc33"}
| render
```

Listing 17: Kibana Canvas expression bar chart

In this Canvas query, I create custom price categories (or bins) based on the `AverageCostForTwo` field. Each restaurant is assigned to a category such as Cheap, Cheap/Medium, Medium, Pricy, or Super Pricy. I then count the number of restaurants in each price category and city using the `COUNT(RestaurantName.keyword)` function. Restaurants without a valid price category (i.e., null or zero values) are excluded. The data is grouped by `price_category` and `City.keyword`, and the results are ordered from the most expensive to the cheapest category. Finally, the data is visualized as a bar chart.

4 Conclusion

This project demonstrates the power of Kibana in transforming raw data into actionable insights through well-structured queries, meaningful aggregations, and visually engaging dashboards. By leveraging Kibana Canvas, we built an interactive and user-friendly view of restaurant ratings across cities, enabling dynamic filtering, price segmentation, and rating classification. The use of Elasticsearch aggregations allowed for precise binning and categorization, while the visualizations, including heatmaps, metrics, and tables, offered a comprehensive snapshot of the restaurant landscape. Together, these tools provided a flexible and efficient way to analyze and communicate key insights, showcasing the full potential of the Elastic Stack in real-world data exploration.