Assignment 2 Part 1 Report

Ilaria Cattaneo

23rd April 2023

1 Exercise 1

1.1 Point 1 - Indexing

First thing first, we converted the csv file into a json file. So, the step-by-step procedure consists as follows. We deleted the first column of the given csv, because we didn’t need the indices at the beginning of each row, and them we saved data in a new csv file to avoid confusion; the code is written below:

```
import csv
        with open(’restaurants.csv’, ’r’) as csvfile:
            reader = csv.reader(csvfile)
            your_list = list(reader)
            for row in your_list:
                del row[0]
            with open(’restaurants2.csv’, ’w’) as f:
                writer = csv.writer(f)
                writer.writerows(your_list)
```

We wrote the code to convert the csv into a json file, specifying the data types we would like to have in the mapping when importing it into ElasticSearch, using the code below:

```
import json
import ast
import sys
jsonfile = open(’restaurants.json’, ’w’)
with open(’restaurants2.csv’, ’r’) as csvfile:
    reader = csv.DictReader(csvfile)
    for row in reader:
        document = {}
        for column, value in row.items():
            if column == ’RestaurantName’:
                document[column] = value
            elif column == ’City’:
                document[column] = value
            elif column == ’AverageCostForTwo’:
                document[column] = float(value)
            elif column == ’AggregateRating’:
                document[column] = float(value)
            elif column == ’RatingText’:
                document[column] = value
            elif column == ’Votes’:
                document[column] = float(value)
            elif column == ’Date’:
                document[column] = value
            elif column == ’Coordinates’:
                res = ast.literal_eval(value)
                document[column] = res

        jsonfile.write(json.dumps(document))
        jsonfile.write(’\n’)
```

Then, using the terminal, we used the commands the professor provided us on iCorsi, in order to make a bulk import of the json file into ElasticSearch into an index named 'restaurants1'; the code is written below:

```
$ sed -i .bak -E 's/.*/{"index":␣{}}\&/g' restaurants.json
$ split -l 10000 restaurants.json restaurants_split_
$ for f in `ls restaurants_split_*` ; do curl -k --user
"elastic:${ES_PASS}" -X POST https://localhost:9200/restaurants1/
    _bulk
-H 'Content-type:␣application/x-ndjson' --data-binary @$f; done
```

After that, on ElasticSearch we checked the mapping of our imported file, using the following command.

```
    GET restaurants1/_mapping
```

We noticed that, the imported file has a slightly different mapping with respect to the one we've written in our python code; for instance, 'AverageCostForTwo' in our code was written as a float, but it was mapped as an integer, so after looking at the data we decided to leave it as an integer. To be sure everything was working smoothly, we created a new index called 'restaurantsindex' and its mapping, and imported the json again with the new index.

```
PUT restaurantsindex

PUT restaurantsindex/_mapping
{
  "properties": {
    "AggregateRating": {
      "type": "float"
    },
    "AverageCostForTwo": {
      "type": "integer"
    },
    "City": {
      "type": "text",
      "fields": {
            "keyword": {
              "type": "keyword",
              "ignore_above": 256
        }
      }
    },
    "Coordinates": {
      "type": "geo_point"
    },
    "Date": {
      "type": "date"
    },
    "RatingText": {
      "type": "text",
      "fields": {
            "keyword": {
              "type": "keyword",
              "ignore_above": 256
        }
      }
    },
    "RestaurantName": {
      "type": "text",
      "fields": {
            "keyword": {
              "type": "keyword",
              "ignore_above": 256
        }
      }
    },
    "Votes": {
      "type": "float"
    }
  }
}
```

```
$ for f in `ls restaurants_split_*` ; do curl -k --user
"elastic:${ES_PASS}" -X POST https://localhost:9200/restaurantsindex
   /_bulk
-H 'Content-type:␣application/x-ndjson' --data-binary @$f; done
```

1.2 Point 2 - Queries

1.2.1 Question a

In the first task we have to get restaurants that have 'pizza' in the name and not 'pasta', that have reviews greater than or equal to 'Very Good'. The code is as follows.

```
GET restaurantsindex/_search
{
  "size": 52,
  "query": {
    "bool": {
      "must": [
        {
          "match": {
          "RestaurantName": "pizza"
        }},
        {
          "bool": {
            "should": [
              {
                "match": {
                  "RatingText": "Very Good"
                }
              },
              {
                "match": {
                  "RatingText": "Excellent"
                }
              }
            ]
          }
        },
        {
          "bool": {
            "must_not": [
              {
                "match": {
                  "RestaurantName": "pasta"
                }
              }
            ]
          }
        }
      ]
    }
  }
}
```

From the running of this query, we found 52 documents returned; in particular, there are 52 restaurants that have a rating of 'Very Good' or 'Excellent' that have 'pizza' and not 'pasta' in their names.

1.2.2 Question b

In the second task we have to retrieve the 5 most expensive restaurants, which have reviews from 2018 and are located within 20km from Athens, using the following code.

```
GET restaurantsindex/_search
{
  "size": 5,
  "query": {
    "bool": {
      "must": [
        {
          "geo_distance": {
            "distance": "20km",
            "Coordinates": {
              "lat": 33.9259,
              "lon": -83.3389
            }

          }},
          {
            "range": {
              "Date": {
                "gte": "2018||/y",
                "lte": "2018||/y",
                "format": "yyyy"
              }
            }
          }
      ]
    }
  },
  "sort": [
    {
      "AverageCostForTwo": {
        "order": "desc"
      }
    }
  ]
}
```

The query returned 8 documents, this means that 8 restaurants satisfy the conditions written in the code to answer the task.

1.2.3 Question c

In the third task we have to retrieve all restaurants which contain the sub-string ’pizz’ in their name but that do not contain neither ’pizza’ nor ’pizzeria’. To do so, we used the code below.

```
GET restaurantsindex/_search
{
  "query": {
    "bool": {
      "must": [
        {
          "wildcard": {
            "RestaurantName": {
              "value": "*pizz*"
            }
          }
        },
        {
          "bool": {
            "must_not": [
              {
                "wildcard": {
                  "RestaurantName": {
                    "value": "*pizza*"
                  }
                }
              },
              {
                "wildcard": {
                  "RestaurantName": {
                    "value": "*pizzeria*"
                  }
                }
              }
            ]
          }
        }
      ]
    }
  }
}
```

The query returned 1 document and this means that there is only 1 restaurant, whose name contains the sub-string 'pizz' but does not contain neither 'pizza' nor 'pizzeria'; as a matter of fact, the restaurant returned by the query is called 'Pizzoccheri'.

1.3 Point 3 - Aggregations

1.3.1 Question a

In the first activity of this section, we have to aggregate restaurants that have 'Good' as review by number of votes, by creating 4 buckets and for each of them we have to return the maximum and the minimum value of 'AverageCostForTwo'.

```
GET restaurantsindex/_search
{
  "size": 0,
  "query": {
    "bool": {
      "must": [
        {
          "match": {
            "RatingText": "Good"
          }
        }
      ]
    }
  },
  "aggs": {
    "rating_buckets": {
      "range": {
        "field": "Votes",
        "ranges": [
          {
            "from": 0,
            "to": 250
          },
          {
            "from": 250,
            "to": 500
          },
          {
            "from": 500,
            "to": 750
          },
          {
            "from": 750,
            "to": 1000
          }
        ]
      },
      "aggs": {
        "min_cost": {
          "min": {
            "field": "AverageCostForTwo"
          }
        },
        "max_cost":{
          "max": {
            "field": "AverageCostForTwo"
          }
        }
      }
    }
  }
}
```

The aggregation query returned a total of 3162 documents; in this case we are not facing the shard size problem. The result is reported below.

```
{
  "took": 1,
  "timed_out": false,
  "_shards": {
    "total": 1,
    "successful": 1,
    "skipped": 0,
    "failed": 0
  },
  "hits": {
    "total": {
      "value": 3162,
      "relation": "eq"
    },
    "max_score": null,
    "hits": []
  },
  "aggregations": {
    "rating_buckets": {
      "buckets": [
        {
          "key": "0.0-250.0",
          "from": 0,
          "to": 250,
          "doc_count": 2060,
          "max_cost": {
            "value": 350000
          },
          "min_cost": {
            "value": 0
          }
        },
        {
          "key": "250.0-500.0",
          "from": 250,
          "to": 500,
          "doc_count": 583,
          "max_cost": {
            "value": 450000
          },
          "min_cost": {
            "value": 10
          }
        },
        {
          "key": "500.0-750.0",
          "from": 500,
          "to": 750,
          "doc_count": 217,
          "max_cost": {
            "value": 5000
          },
          "min_cost": {
            "value": 10
          }
        },
        {
          "key": "750.0-1000.0",
          "from": 750,
          "to": 1000,
          "doc_count": 99,
          "max_cost": {
            "value": 200000
          },
          "min_cost": {
            "value": 10
          }
        }
      ]
    }
  }
}
```

1.3.2 Question b

The second activity consists of finding cities that have at least 10 restaurants and restaurants that have at least 100 votes, and returning the 7 cities with the highest ’AverageCostForTwo’.

```
GET restaurantsindex/_search
{
  "size": 0,
  "query": {
    "range": {
      "Votes": {
        "gte": 100
      }
    }
  },
  "aggs": {
    "frequent_cities": {
      "terms": {
        "field": "City.keyword",
        "min_doc_count": 10,
        "order": {
          "average": "desc"
        },
        "shard_size": 1000,
        "size": 7
      },
        "aggs": {
        "average": {
          "max": {
            "field": "AverageCostForTwo"
          }
        }
      }
    }
  }
}
```

The aggregation query returned 2799 documents; here we are facing the shard problem, as we find ”doc_count_error_upper_bound”: -1, so to have the right result we must specify the shard size to 1000. The result is reported below.

```
{
  "took": 1,
  "timed_out": false,
  "_shards": {
    "total": 1,
    "successful": 1,
    "skipped": 0,
    "failed": 0
  },
  "hits": {
    "total": {
      "value": 2799,
      "relation": "eq"
    },
    "max_score": null,
    "hits": []
  },
  "aggregations": {
    "frequent_cities": {
      "doc_count_error_upper_bound": 0,
      "sum_other_doc_count": 970,
      "buckets": [
        {
          "key": "Jakarta",
          "doc_count": 16,
          "average": {
            "value": 800000
          }
        },
        {
          "key": "New Delhi",
          "doc_count": 1174,
          "average": {
            "value": 8000
          }
        },
        {
          "key": "Gurgaon",
          "doc_count": 274,
          "average": {
            "value": 5000
          }
        },
        {
          "key": "Colombo",
          "doc_count": 14,
          "average": {
            "value": 4500
          }
        },
        {
          "key": "Noida",
          "doc_count": 192,
          "average": {
            "value": 3200
          }
        },
        {
          "key": "Jaipur",
          "doc_count": 18,
          "average": {
            "value": 2500
          }
        },
        {
          "key": "Bangalore",
          "doc_count": 20,
          "average": {
            "value": 2400
          }
        }
      ]
    }
  }
}
```

1.3.3 Question c

In the third activity, we have to filter for restaurants that are located within 9000km of New Dehli and show the highest number of votes for different rating types in descending order.

```
GET restaurantsindex/_search
{
  "size": 0,
  "query": {
    "bool": {
      "must": [
        {
          "geo_distance": {
            "distance": "9000km",
            "Coordinates": {
              "lat": 28.642449499999998,
              "lon": 77.10684570000001
            }
          }
        }
      ]
    }
  },
  "aggs": {
    "ratings": {
      "terms": {
        "field": "RatingText.keyword",
        "size": 10
      },
      "aggs": {
        "max_rate": {
          "max": {
            "field": "AggregateRating"
          }
        },
        "max_votes": {
          "max": {
            "field": "Votes"
          }
        },
        "sorting_votes": {
         "bucket_sort": {
           "sort": [
             {
               "max_votes": {
                 "order": "desc"
               }
             }
            ]
         }
        }
      }
    }
  }
}
```

The aggregation query returned 8930 documents. Here we are not facing the shard problem. The result is reported below.

```
{
  "took": 0,
  "timed_out": false,
  "_shards": {
    "total": 1,
    "successful": 1,
    "skipped": 0,
    "failed": 0
  },
  "hits": {
    "total": {
      "value": 8930,
      "relation": "eq"
    },
    "max_score": null,
    "hits": []
  },
  "aggregations": {
    "ratings": {
      "doc_count_error_upper_bound": 0,
      "sum_other_doc_count": 0,
      "buckets": [
        {
          "key": "Excellent",
          "doc_count": 196,
          "max_rate": {
            "value": 4.900000095367432
          },
          "max_votes": {
            "value": 10934
          }
        },
        {
          "key": "Very Good",
          "doc_count": 834,
          "max_rate": {
            "value": 4.400000095367432
          },
          "max_votes": {
            "value": 7931
          }
        },
        {
          "key": "Good",
          "doc_count": 1902,
          "max_rate": {
            "value": 3.9000000953674316
          },
          "max_votes": {
            "value": 4914
          }
        },
        {
          "key": "Average",
          "doc_count": 3683,
          "max_rate": {
            "value": 3.4000000953674316
          },
          "max_votes": {
            "value": 2460
          }
        },
        {
          "key": "Poor",
          "doc_count": 180,
          "max_rate": {
            "value": 2.4000000953674316
          },
          "max_votes": {
            "value": 2412
          }
        },
        {
          "key": "Not rated",
          "doc_count": 2135,
          "max_rate": {
            "value": 0
          },
          "max_votes": {
            "value": 3
          }
        }
      ]
    }
  }
}
```