Rapport
Elasticsearch/ Kibana

elasticsearch

kibana

Table of contents

1. Data import

For data insertion, we reused the previously modified file where we modified incorrect data like dates: [0000,00,00] or coma that closed the query. Then, using a Python script, we added the necessary metadata to insert the data into Elasticsearch. We added an index line with the index "basket" as well as the ID of each player.

asketball_women.json > script.py > ...

```python
import json

def add_index_to_json(input_file, output_file, index_name):
    with open(input_file, 'r') as f:
        data = f.readlines()

    transformed_data = []
    for line in data:
        obj = json.loads(line)
        index_line = {"index": {"_index": index_name, "_id": obj.get("_id", None)}}
        transformed_data.append(index_line)
        del obj["_id"]  # Remove _id from the object itself
        transformed_data.append(obj)

    with open(output_file, 'w') as f:
        for obj in transformed_data:
            json.dump(obj, f)
            f.write('\n')

# Replace 'your_input_file.json', 'transformed_output_file.json' and 'movies' with your actual file paths
add_index_to_json('Basketball_womenClean).json', 'Basketball_women2.json', 'basket')

```

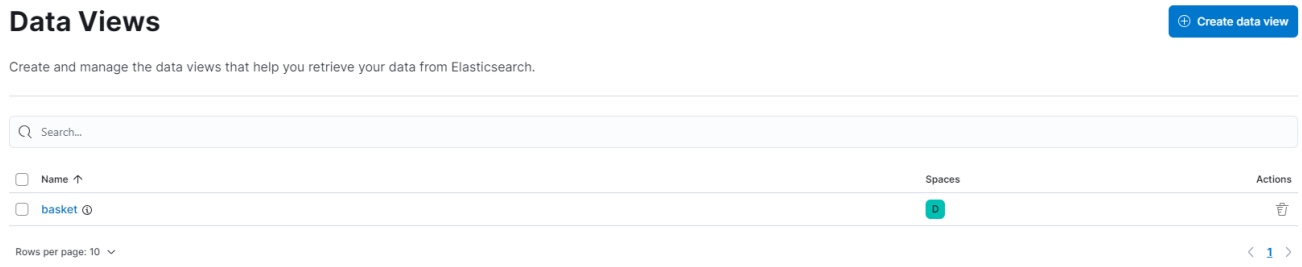
Then with the json that has been created with this script, we have been able to insert the data thanks to the curl command.

```
C:\Users\paull\Downloads\Basketball_women.json>curl --insecure -XPUT -u elastic:ny2TOULZE7i5kSaeVWtT
https://localhost:9200/_bulk -H"Content-Type:application/json" --data-binary @Basketball_women2.json
```

Then we checked and we can see that our index has been well created with the 893 data.

☐	basket	● yellow	open	1	1	893	678.71kb

Then we created a data view with this index.



And we checked that all the data is present with a count.

```
POST /basket/_count
{
  "query": {
    "match_all": {}
  }
}
```

```
{
  "count": 893,
  "_shards": {
    "total": 1,
    "successful": 1,
    "skipped": 0,
    "failed": 0
  }
}
```

We obtained 893 data which are present in our dataset.

2. Simple Queries (6 queries)

1) For the first query let's find all the players that get the award: "Defensive Player of the Year"

```
GET /basket/_search
{
  "_source": ["firstName", "awards_players"],
  "query": {
    "match_phrase": {
      "awards_players.award": "Defensive Player of the Year"
    }
  }
}
```

Let's explain what this query did.

```
GET /basket/_search
{"_source": ["firstName", "awards_players"],
 "query": {
   "match_phrase": {
    "awards_players.award": "Defensive Player of the Year"}
 }
}
```

- The GET method allows you to make an http Get request on the elastic search API, and we specify the basket index that we created during our data import.
- The source parameter specifies the fields that are to be included in the result. This allows us to keep only the name as well as the awards players fields where the interesting information from our request is located.
- Then, we carry out a query or we look for the players where the "awards_players.award" field corresponds exactly to "Defensive Player of the Year".

Here is a part of the result.

```
},
"hits": {
  "total": {
    "value": 8,
    "relation": "eq"
  },
  "max_score": 6.7034864,
  "hits": [
    {
      "_index": "basket",
      "_id": "blackde01w",
      "_score": 6.7034864,
      "_source": {
        "firstName": "Debbie",
        "awards_players": [
          {
            "award": "Defensive Player of the Year",
            "year": 1997,
            "pos": null
          },
          {
            "award": "Defensive Player of the Year",
            "year": 2001,
            "pos": null
          }
        ]
      }
    },
    {
```


Later, in the the hard query, we will use the beginning of this query and add another aggregate function to get more precise information.

2) Let's count the number of distinct teams where Marlies Arkamp played:

```
GET /basket/_search
{
  "query": {
    "match": {
      "fullGivenName": "Marlies Askamp"
    }
  },
  "size": 0,
  "aggs": {
    "unique_teams": {
      "cardinality": {
        "field": "players_teams.tmID.keyword"
      }
    }
  }
}
```


```
GET /basket/_search
{
 "query": {
  "match": {
   "fullGivenName": "Marlies Askamp"
  }
 },
 "size": 0,
```

```
 "aggs": {
  "unique_teams": {
   "cardinality": {
    "field": "players_teams.tmID.keyword"
   }
  }
 }
}
```

- The first part of the query is to get the player whose name is Marlies Askamp using the “match” function to the "fullGivenName” attribute.
- "size": 0 means that we don’t want to display all the information about Marlies Askamp in the result of the query.
- The second part of the query is to use an aggregate function which name is “unique_teams” (we can choose the name we want) in order to create a new output where “cardinality” function is to count every distinct team where she played!
  Then, we applied this query to the attribute “player_teams” which id an array of team she played in. Moreover, we select the attribute “.tmID” in the “player_teams” array and the “.keyword” is used to group by this term (here by tmID).


```
{
  "took": 0,
  "timed_out": false,
  "_shards": {
    "total": 1,
    "successful": 1,
    "skipped": 0,
    "failed": 0
  },
  "hits": {
    "total": {
      "value": 1,
      "relation": "eq"
    },
    "max_score": null,
    "hits": []
  },
  "aggregations": {
    "unique_teams": {
      "value": 3
    }
  }
}
```


Result of the query

Overall, we can conclude that Marlies Askamp played in 3 different teams in her whole career but it can’t tell us if she had played for several years in her career or not, maybe she only played 3 years, so just 3 season is too short for a basketball player, maybe because of an injurie or because she is just a bad player? Let’s verify the hypothesis in our next query!

3) Let’s see how many years Marlies Askamp played:

```
GET /basket/_search
{
  "query": {
    "match_phrase": {
      "fullGivenName": "Marlies Askamp"
    }
  },
  "size": 1,
  "aggs": {
    "unique_years": {
      "cardinality": {
        "field": "players_teams.year"
      }
    }
  }
}
```

GET /basket/_search
{
 "query": {
  "match_phrase": {
   "fullGivenName": "Marlies Askamp"
  }
 },
 "size": 1,
 "aggs": {
  "unique_years": {
   "cardinality": {
    "field": "players_teams.year"
   }
  }
 }
}

- We want to display only the player Marlies Askamp with the match_phrase function.
- Here, the “size”: 1 means that we want to display all the information of this player.
- We use also an aggregate function which name is “unique_years” and we use the function cardinality function in order to count the number of distinct years the player Marlies Askamp played with the attribute “year” located in the “players_team”.

```
{
  "took": 5,
  "timed_out": false,
  "_shards": {
    "total": 1,
    "successful": 1,
    "skipped": 0,
    "failed": 0
  },
  "hits": {
    "total": {
      "value": 1,
      "relation": "eq"
    },
    "max_score": 12.972365,
    "hits": [
      {
        "_index": "basket",
        "_id": "askamma01w",
        "_score": 12.972365,
        "_source": {
          "firstName": "Marlies",
          "middleName": null,
          "lastName": "Askamp",
          "fullGivenName": "Marlies Askamp",
          "nameNick": null,
          "pos": "C",
          "height": 77,
          "weight": 198,
          "college": "none",
          "birthDate": "1970-08-07",
          "birthCity": "Dousten",
          "birthCountry": "GER",
          "highSchool": null,
          "hsCity": null,
          "hsState": null,
          "hsCountry": null,
          "players_teams": [
            {
              "year": 1997,
              "tmID": "PHO",
              "games": 28,
              "minutes": 517,
              "points": 211,
              "steals": 21,
              "blocks": 9
            },
            {
              "year": 1998,
              "tmID": "PHO",
              "games": 26,
              "minutes": 319,
              "points": 139,
              "steals": 12,
              "blocks": 7
            },
            {
              "year": 1999,
              "tmID": "PHO",
              "games": 30,
              "minutes": 781,
              "points": 283,
              "steals": 22,
              "blocks": 18
            },
            {
              "year": 2000,
              "tmID": "MIA",
              "games": 32,
              "minutes": 869,
              "points": 251,
              "steals": 17,
              "blocks": 21
            },
            {
              "year": 2001,
              "tmID": "MIA",
              "games": 30,
              "minutes": 431,
              "points": 74,
              "steals": 12,
              "blocks": 11
            },
            {
              "year": 2002,
              "tmID": "LAS",
              "games": 20,
              "minutes": 215,
              "points": 61,
              "steals": 11,
              "blocks": 4
            },
            {
              "year": 2002,
              "tmID": "MIA",
              "games": 6,
              "minutes": 72,
              "points": 11,
              "steals": 1,
              "blocks": 1
            }
          ],
          "awards_players": []
        }
      }
    ]
  },
  "aggregations": {
    "unique_years": {
      "value": 6
    }
  }
}
```


Result of the query

Overall, we can notice an interesting thing. Indeed, with our aggregate function we can see that the player played 6 years in her whole career BUT we can see that the “players_teams” array there are 7 elements. In fact, we noticed that during her last season she played in 2 different teams: LAS (Las Vegas) and MIA (Miami) that probably means she broke her contract or was transferred…

4) Let’s see how many players are over 220.4623 lbs (100 kgs)

```
GET /basket/_search
{
  "query": {
    "range": {
      "weight":{
        "gte":220.4623,
        "boost":2.0
      }
    }
  }
}
```


```
GET /basket/_search
{
 "query": {
  "range": {
   "weight": {
    "gte":220.4623,
    "boost":2.0
   }
  }
```

```
  }
}
```

GET /basket/_search: This part of the query instructs Elasticsearch to perform a search on the "basket" index. The HTTP method "GET" is used to retrieve the search results.

- "query": This section specifies the search criteria. In this case, it's a "range" query.
- "range": This is a type of query used to filter documents based on a range of values for a given field.
- "weight": This is the field on which the range search is applied. In this case, it's the "weight" field.
- "gte": 220.4623: This signifies "greater than or equal to." The query will search for documents where the value of the "weight" field is greater than or equal to 220.4623. "boost": 2.0: This is a boost parameter, which assigns additional weight to this condition in the relevance calculation. A higher boost means that documents satisfying this condition will have higher relevance in the results.

In summary, this query searches for documents in the "basket" index where the "weight" field value is greater than or equal to 220.4623, and it gives a boost of 2.0 to this condition, potentially influencing the relevance of the results.

5) Let's check the height repartition.

```
GET /basket/_search
{
  "aggs": {
    "height_ranges": {
      "range":{
        "field":"height",
        "ranges":[
          {
            "to":50.0

          },
          {
            "from":50.0,
            "to":60.0
          },
          {
            "from":60.0,
            "to":62.0
          },
          {
            "from":62.0,
            "to":64.0
          },
          {
            "from":64.0,
            "to":66.0
          },
          {
            "from":66.0,
            "to":68.0
          },
          {
            "from":68.0,
            "to":70.0
          },
          {
            "from":70.0,
            "to":72.0
          },
          {
            "from":72.0,
            "to":74.0
          },
          {
            "from":74.0,
            "to":76.0
          },
          {
            "from":76.0,
            "to":78.0
          },
          {
            "from":78.0,
            "to":80.0
          },
          {
            "from":80.0
          }
          ]
      }
    }
  }
}

```

GET /basket/_search
{
 "aggs": {
  "height_ranges": {
   "range": {
    "field":"height",
    "ranges": [
     {
      "to":50.0

     },

```
{
 "from":50.0,
 "to":60.0
},
{
 "from":60.0,
 "to":62.0
},
{
 "from":62.0,
 "to":64.0
},
{
 "from":64.0,
 "to":66.0
},
{
 "from":66.0,
 "to":68.0
},
{
 "from":68.0,
 "to":70.0
},
{
 "from":70.0,
 "to":72.0
},
{
 "from":72.0,
 "to":74.0
},
{
 "from":74.0,
 "to":76.0
},
{
 "from":76.0,
 "to":78.0
},
{
 "from":78.0,
 "to":80.0
},
{
 "from":80.0
}
]
```

```
    }
  }
 }
}
```

We use a range aggregation, compared to a histogram aggregation, it allows us to define specific ranges to get more precision in the ranges where most of the values are and less precision where few people are.

- GET /basket/_search: This part indicates a search operation on the "basket" index.
  "aggs": Stands for aggregations, which are used to compute summary statistics or analytics on the data.
- "height_ranges": The name given to the aggregation. This specific aggregation is intended to define height ranges.
- "range": Specifies a range aggregation, which is used to define specific intervals for a numerical field.
- "field": "height": Indicates that the aggregation is applied to the "height" field.
- "ranges": Defines an array of range intervals. Each interval is specified by a pair of values – "from" and "to".
  The ranges provided in the code segment represent height intervals, such as from less than 50.0, from 50.0 to less than 60.0, and so on.

In summary, this Elasticsearch query performs a range aggregation on the "height" field, creating specific intervals or ranges for height and providing counts of documents falling into each range.

Here, it’s a part important part of the output:

```
"aggregations": {
  "height_ranges": {
    "buckets": [
      {
        "key": "*-50.0",
        "to": 50,
        "doc_count": 81
      },
      {
        "key": "50.0-60.0",
        "from": 50,
        "to": 60,
        "doc_count": 0
      },
      {
        "key": "60.0-62.0",
        "from": 60,
        "to": 62,
        "doc_count": 0
      },
      {
        "key": "62.0-64.0",
        "from": 62,
        "to": 64,
        "doc_count": 7
      },
      {
        "key": "64.0-66.0",
        "from": 64,
        "to": 66,
        "doc_count": 17
      },
      {
        "key": "66.0-68.0",
        "from": 66,
        "to": 68,
        "doc_count": 80
      },
      {
        "key": "68.0-70.0",
        "from": 68,
        "to": 70,
        "doc_count": 108
      },
      {
        "key": "70.0-72.0",
        "from": 70,
        "to": 72,
        "doc_count": 127
      },
      {
        "key": "72.0-74.0",
        "from": 72,
        "to": 74,
        "doc_count": 162
      },
      {
        "key": "74.0-76.0",
        "from": 74,
        "to": 76,
        "doc_count": 177
      },
      {
        "key": "76.0-78.0",
        "from": 76,
        "to": 78,
        "doc_count": 106
      },
      {
        "key": "78.0-80.0",
        "from": 78,
        "to": 80,
        "doc_count": 22
      },
      {
        "key": "80.0-*",
        "from": 80,
        "doc_count": 6
      }
    ]
  }
}
```


We can notice that there are not any players between 50 (127 cm) and 62 (157,42 cm) inches, so we can conclude that the value 81 players who are less than 50 inches are the players who don't have the height in our JSON dataset (it means their height is equal to 0 inch).
Moreover, we can observe that the number of players who have their height increases from 64 inches (162.56 cm) to 76 inches (193,04 cm) and then from 76 inches to 80 inches (203.2 cm) or more the number of players decreases.

Finally, we can see most of the players are between 74 inches (187,96 cm) and 76 inches (193.04 cm), so if a woman wants to play in WNBA, she must be preferably between 74 inches and 76 inches!

6) Let's find the players playing centers position, taller than 74 inches who are not from the USA:

```
GET /basket/_search
{
  "_source": ["firstName", "height", "birthCountry"],
  "query": {
    "bool": {
      "must": [
        {
          "match": {
            "pos": "C"
          }
        },
        {
          "range": {
            "height": {
              "gte": 74
            }
          }
        }
      ],
      "must_not": [
        {
          "match": {
            "birthCountry": "USA"
          }
        }
      ]
    }
  }
}
```

```
GET /basket/_search
{
 "_source": ["firstName", "height", "birthCountry"],
 "query": {
  "bool": {
   "must": [
    {
     "match": {
      "pos": "C"
     }
    },
    {
     "range": {
      "height": {
       "gte": 74
      }
     }
    }
   ],
```

```
    "must_not": [
     {
      "match": {
       "birthCountry": "USA"
      }
     }
    ]
   }
  }
}
```

- The GET method allows you to make an http Get request on the elastic search API, and we specify the basket index that we created during our data import.
- The source parameter specifies the fields that are to be included in the result. This allows us to keep only the name, height, and birth country on players selected by the query.
- Then, we carry out a query by selecting players with the help of a Boolean value:
    - First Boolean condition: players must be Central position (pos: 'C'), and must be taller than 74 inches
    - Second Boolean condition: players must not be born in USA

```
{
  "took": 3,
  "timed_out": false,
  "_shards": {
    "total": 1,
    "successful": 1,
    "skipped": 0,
    "failed": 0
  },
  "hits": {
    "total": {
      "value": 162,
      "relation": "eq"
    },
    "max_score": 2.4497972,
    "hits": [
      {
        "_index": "basket",
        "_id": "abrahta01w",
        "_score": 2.4497972,
        "_source": {
          "firstName": "Tajama",
          "height": 74,
          "birthCountry": "ISV"
        }
      },
      {
        "_index": "basket",
        "_id": "anderch01w",
        "_score": 2.4497972,
        "_source": {
          "firstName": "Chantelle",
          "height": 78,
          "birthCountry": null
        }
      },
      {
        "_index": "basket",
        "_id": "anosini01w",
        "_score": 2.4497972,
        "_source": {
          "firstName": "Nkolika",
          "height": 75,
```


The result shows us 10 players that match these characteristics, including 6 that have not their birth country referenced.

3. Complex Queries (2 queries)

1) In this first complex query, we wanted to check the average and the maximum of blocks and steals of the players that get the award Defensive Player of the year.

```
GET /basket/_search
{
  "size": 0,
  "aggs": {
    "defensive_players": {
      "filter": {
        "match_phrase": {
          "awards_players.award": "Defensive Player of the Year"
        }
      },
      "aggs": {
        "avg_steals_defensive_players": {
          "avg": {
            "field": "players_teams.steals"}},
        "max_steals_defensive_players": {
          "max": {
            "field": "players_teams.steals"}},
        "avg_blocks_defensive_players": {
          "avg": {
            "field": "players_teams.blocks"
          }},
        "max_blocks_defensive_players": {
          "max": {
            "field": "players_teams.blocks"
          }
        }
      }
    }
  }
}
```

Let's explain what we did in this query.

```
GET /basket/_search
{
 "size": 0,
 "aggs": {
  "defensive_players": {
   "filter": {
    "match_phrase": {
     "awards_players.award": "Defensive Player of the Year"
    }
   },
   "aggs": {
    "avg_steals_defensive_players": {
     "avg": {
      "field": "players_teams.steals"}},
    "max_steals_defensive_players": {
     "max": {
      "field": "players_teams.steals"}},
    "avg_blocks_defensive_players": {
     "avg": {
      "field": "players_teams.blocks"
     }},
    "max_blocks_defensive_players": {
     "max": {
      "field": "players_teams.blocks"
     }}}}}}
```

The first argument size: 0 means that the response size should be 0. This means that only aggregations will be returned.

Then the first filter type aggregation allows the document to be filtered in the following way: only the players with the exact mention “Defensive Player of the Year” in the award field will be kept.
Inside this aggregation, we add four other aggregations. The first calculates the average over the players_teams.steals field to obtain the average of steals. We then use the max aggregation to recover the greatest value of the steals. The next two are the average and the max for the blocks this time though.

Here is the result of the query.


```
},
"aggregations": {
  "defensive_players": {
    "doc_count": 8,
    "max_steals_defensive_players": {
      "value": 177
    },
    "avg_steals_defensive_players": {
      "value": 52.73170731707317
    },
    "avg_blocks_defensive_players": {
      "value": 32.09756097560975
    },
    "max_blocks_defensive_players": {
      "value": 98
    }
  }
}
```


To analyze those results, we decided to compare with same query but with all the players, we just removed the first aggregate function.

```
GET /basket/_search
{
  "size": 0,
    "aggs": {
      "avg_steals_all_players": {
        "avg": {
          "field": "players_teams.steals"}},
      "max_steals_all_players": {
        "max": {
          "field": "players_teams.steals"}},
      "avg_blocks_all_players": {
        "avg": {
          "field": "players_teams.blocks"
        }},
      "max_blocks_all_players": {
        "max": {
          "field": "players_teams.blocks"
        }
      }
    }
  }
```

```
},
"aggregations": {
  "avg_steals_all_players": {
    "value": 20.843501326259947
  },
  "max_steals_all_players": {
    "value": 177
  },
  "avg_blocks_all_players": {
    "value": 8.92340848806366
  },
  "max_blocks_all_players": {
    "value": 114
  }
}
```


As we expected, the avg values of blocks and steals in lower than for the players that get the Defensive player of the year title. However, the player the make the maximum number of blocks is not one of the players that got the award.

2) In this second complex query, we retrieve the players who won awards and calculate different values to create a ranking.

```
GET /basket/_search
{
  "runtime_mappings":{
    "award_count":{
      "type":"long",
      "script":"long count = doc['awards_players.year'].size();emit(count);"
    },
    "player_points":{
      "type":"long",
      "script":"long sum = 0; for(points in doc['players_teams.points']){sum += points;}emit(sum);"
    },
    "years_played":{
      "type":"long",
      "script":"long years = doc['players_teams.year'].size();emit(years);"
    },
    "yearly_point_avg":{
      "type":"double",
      "script":"double avg = doc['player_points'].value/doc['years_played'].value;emit(avg);"
    }
  },

  "query":{
    "bool":{
      "filter": [
        {
          "exists":{
            "field":"awards_players"
          }
        },
        {
          "exists":{
            "field":"players_teams"
          }
        }
      ]
    }
  },

  "size":10000,
  "sort": [
    {
      "yearly_point_avg":"desc",
      "award_count":"desc",
      "player_points":"desc"
    }
  ],

  "_source":false,
  "fields": [
    "fullGivenName",
    "award_count",
    "awards_players.award",
    "player_points",
    "years_played",
    "yearly_point_avg"
    ]
}
```

```
GET /basket/_search
{
 "runtime_mappings":{
  "award_count":{
   "type":"long",
   "script":"long count = doc['awards_players.year'].size();emit(count);"
  },
  "player_points":{
   "type":"long",
   "script":"long sum = 0; for(points in doc['players_teams.points']){sum += points;}emit(sum);"
  },
  "years_played":{
   "type":"long",
   "script":"long years = doc['players_teams.year'].size();emit(years);"
  },
  "yearly_point_avg":{
   "type":"double",
   "script":"double avg = doc['player_points'].value/doc['years_played'].value;emit(avg);"
  }
 },

 "query":{
  "bool":{
   "filter": [
    {
     "exists":{
      "field":"awards_players"
     }
    },
    {
     "exists":{
      "field":"players_teams"
     }
    }
   ]
  }
 },

 "size":10000,
 "sort": [
  {
   "yearly_point_avg":"desc",
   "award_count":"desc",
   "player_points":"desc"
  }
 ],

 "_source":false,
```

```
  "fields": [
    "fullGivenName",
    "award_count",
    "awards_players.award",
    "player_points",
    "years_played",
    "yearly_point_avg"
    ]
}
```

In the first part, we define different values using runtime mapping and painless scripts.

- runtime_mappings: This section defines custom fields on the fly during runtime.
  - "award_count": Calculates the count of awards a player has received.
  - "player_points": Computes the total points a player has.
  - "years_played": Calculates the number of years a player has participated.
  - "yearly_point_avg": Computes the average points per year for a player.

In the second part, we check if the player has awards and if he has a team. Checking if the player has a value in the players_teams field protects the painless script that uses players_teams values.

- query: Uses a boolean filter to ensure that documents must have both "awards_players" and "players_teams" fields.

In the third part, we set the size to 10000 to get all the results and sort based on certain values to create a ranking.

- size: Specifies the maximum number of results to return, set to 10,000 in this case.
- sort: Orders the results based on descending values of "yearly_point_avg," "award_count," and "player_points."

Finally, we use “_source”=false and the “fields” property to select only certain fields.

- _source: Set to false to exclude the source document from the search results.
- fields: Specifies the fields to include in the search results, including custom runtime fields and specific fields from the source document.

In summary, this Elasticsearch query performs a search with custom runtime mappings to calculate various statistics for basketball players, filters results based on the existence of certain fields, sorts the results, and specifies the fields to include in the output.

Here, is a part of the output:

```
{
  "_index": "basket",
  "_id": "tauradi01w",
  "_score": null,
  "fields": {
    "awards_players.award": [
      "Most Valuable Player",
      "Rookie of the Year",
      "WNBA Finals Most Valuable Player",
      "WNBA All Decade Team Honorable Mention",
      "Peak Performer: Points",
      "Peak Performer: Points"
    ],
    "player_points": [
      5535
    ],
    "fullGivenName": [
      "Diana Taurasi"
    ],
    "yearly_point_avg": [
      615
    ],
    "years_played": [
      9
    ],
    "award_count": [
      6
    ]
  },
  "sort": [
    615,
    6,
    5535
  ]
},
```


We can see concretely that this query displays every player who has won at least one award, and it shows us all the awards a player won without displaying all their information. Furthermore, it displays the total points of the whole career of the player, the average point she scored in one year, the number of years she played and the total awards she won until the end of her career.

4. Hard Query (1 query)

In the hard query, we will try to see the average points of each team each year and to see which team has the best average points each year.

```
GET /basket/_search
{
  "size": 0,
  "aggs": {
    "Year": {
      "terms": {
        "field": "players_teams.year",
        "size": 100,
        "order": {
          "_key": "desc"
        }
      },
      "aggs": {
        "max_avg_points_per_game": {
          "max_bucket": {
            "buckets_path": "Team>avg_points_per_game"
          }
        },
        "Team": {
          "terms": {
            "field": "players_teams.tmID.keyword",
            "size": 20
          },
          "aggs": {
            "total_points": {
              "sum": {
                "field": "players_teams.points"
              }},
            "total_games": {
              "sum": {
                "field": "players_teams.games"
              }},
            "avg_points_per_game": {
              "bucket_script": {
                "buckets_path": {
                  "totalPoints": "total_points",
                  "totalGames": "total_games"
                },
                "script": "params.totalPoints / params.totalGames"
} } } } } } } }
```

Let’s explain this query

GET /basket/_search

{

"size": 0,

"aggs": {

"Year": {

"terms": {

"field": "players_teams.year",

"size": 100,

"order": {

"_key": "desc"

}

},

"aggs": {

"max_avg_points_per_game": {

"max_bucket": {

"buckets_path": "Team>avg_points_per_game"

}

},

"Team": {

"terms": {

"field": "players_teams.tmID.keyword",

"size": 20

},

"aggs": {

"total_points": {

"sum": {

"field": "players_teams.points"

```
        }},
       "total_games": {
        "sum": {
         "field": "players_teams.games"
        }},
       "avg_points_per_game": {
        "bucket_script": {
         "buckets_path": {
          "totalPoints": "total_points",
          "totalGames": "total_games"
         },
         "script": "params.totalPoints / params.totalGames"
}}}}}}}}
```

The first part is an aggregation that makes a group by year and then order then by descending order.

Then we make two sub aggregations. The first one “Max avg points per game” calculates the maximum value of the average points per game across all the teams for each year. This aggregation uses a max bucket to find the bucket with the highest value. The path of this bucket is specified and is created in the aggregation after.

The second aggregation makes a group by on each team within the specified year.

Then, inside this aggregation, there is also three aggregations.

The first one “total points” makes the sum of all the points by summing the player_teams.points field. This allows us to get the total number of points that each team has each year.

The second one “total games” makes the sum of all the games that has been played by a team. This allows us to get the total number of games for each team each year.

Now, the purpose was to find the average points per game per team per year.

To do that, we created a bucket script aggregation that allows us to perform the division on the results of the other aggregation. It divides the total points by the total games to get the average points.

Here is the result of the query. First these are the results for the year 2012.

```
{
  "key": 2012,
  "doc_count": 146,
  "Team": {
    "doc_count_error_upper_bound": 0,
    "sum_other_doc_count": 4,
    "buckets": [
      {
        "key": "WAS",
        "doc_count": 31,
        "total_games": {
          "value": 6899
        },
        "total_points": {
          "value": 60602
        },
        "avg_points_per_game": {
          "value": 8.784171619075229
        }
      },
      {
        "key": "CHI",
        "doc_count": 25,
        "total_games": {
          "value": 5003
        },
        "total_points": {
          "value": 41108
        },
        "avg_points_per_game": {
          "value": 8.2166699980012
        }
      },
      {
        "key": "PHO",
        "doc_count": 25,
        "total_games": {
          "value": 4475
        },
        "total_points": {
          "value": 41089
        },
        "avg_points_per_game": {
          "value": 9.181899441340782
```

```
    },
    "max_avg_points_per_game": {
      "value": 12.331509846827133,
      "keys": [
        "ORL"
      ]
    }
```


And here is the result for the year 2003

```
"key": 2003,
"doc_count": 175,
"Team": {
  "doc_count_error_upper_bound": 0,
  "sum_other_doc_count": 51,
  "buckets": [
    {
      "key": "HOU",
      "doc_count": 40,
      "total_games": {
        "value": 8407
      },
      "total_points": {
        "value": 67997
      },
      "avg_points_per_game": {
        "value": 8.088140835018438
      }
    },
    {
      "key": "PHO",
      "doc_count": 38,
      "total_games": {
        "value": 6900
      },
      "total_points": {
        "value": 50532
      },
      "avg_points_per_game": {
        "value": 7.323478260869566
      }
    },
    {
      "key": "LAS",
      "doc_count": 36,
      "total_games": {
        "value": 8536
      },
      "total_points": {
```

```
"max_avg_points_per_game": {
  "value": 9.9587002356443,
  "keys": [
    "SEA"
  ]
}
```


We can see that the average points per game is lower for the year 2003 than the year 2012. Also, the team that scores the most points is not the same.