

Elastic Stack 을 활용한 Data Dashboard 만들기

Week 3 - Elasticsearch API를 활용해보자



Fast Campus

목차

- Dev Tools	3
- Data type	7
- API	
- Indicies API	
- Create Index	22
- Delete Index	23
- Mapping	24
- Document API	
- Create Document	35
- Get Document	37
- Delete Document	38
- Update Document	40
- Reindex Document	44
- Search API (Query DSL)	
- Match All	53
- Term/Terms	55
- Prefix/Wildcard/Fuzzy	57
- Range	60
- Query String	61
- Exists	62
- Bool	66
- 기타	75
- 설치	84

Kibana Dev Tools를 살펴보자

(REST API of Elasticsearch)

Dev Tools

Kibana에 접속해서 Dev Tools 화면으로 가자



Dev Tools

다음과 같이 입력하고 녹색 버튼을 눌러보자  

Dev Tools

Console

1 GET shopping/_search
2 []
3 "query": {
4 "match_all": {}
5 }
6 }

1. 입력

2. 선택

```
1 {  
2   "took": 0,  
3   "timed_out": false,  
4   "_shards": {  
5     "total": 5,  
6     "successful": 5,  
7     "skipped": 0,  
8     "failed": 0  
9   },  
10  "hits": {  
11    "total": 20226,  
12    "max_score": 1,  
13    "hits": [  
14      {  
15        "_index": "shopping",  
16        "_type": "shopping",  
17        "_id": "AV-iDKZcRJy4v-Hns1Sk",  
18        "_score": 1,  
19        "_source": {  
20          "접수번호": "277",  
21          "주문시간": "2016-04-11T04:28:14",  
22          "수령시간": "2016-04-11T07:18:14",  
23          "예약여부": "예약",  
24          "배송메모": "상품 이상",  
25          "고객ip": "130.152.206.29",  
26          "고객성별": "남성",  
27          "고객나이": 40,  
28          "물건좌표": "36.56404885993116, 129.87454556889554",  
29          "고객주소_시도": "서울특별시",  
30          "구매사이트": "g마켓",  
31          "판매자평점": 3,  
32          "상품분류": "스웨터",  
33          "상품가격": 10000,  
34          "상품개수": 1,  
35          "결제카드": "시티"  
36        }  
37      },  
38    ]  
39  },  
40}
```

Dev Tools

과거에 작성한 API 이력 조회

cURL 명령어로 복사

The screenshot shows the Elasticsearch Dev Tools interface. On the left is the **Console** pane, which contains a code editor with the following Elasticsearch search query:

```
1 | GET shopping/_search
2 | {"query": {"match_all": {}}}
3 |
4 | GET shopping/_search
5 | {
6 |     "query": {
7 |         "match_all": {}
8 |     }
9 | }
```

A context menu is open over the last line of the query, with options **Copy as cURL** and **Auto indent**. An arrow points from the text above this section to the **Copy as cURL** option.

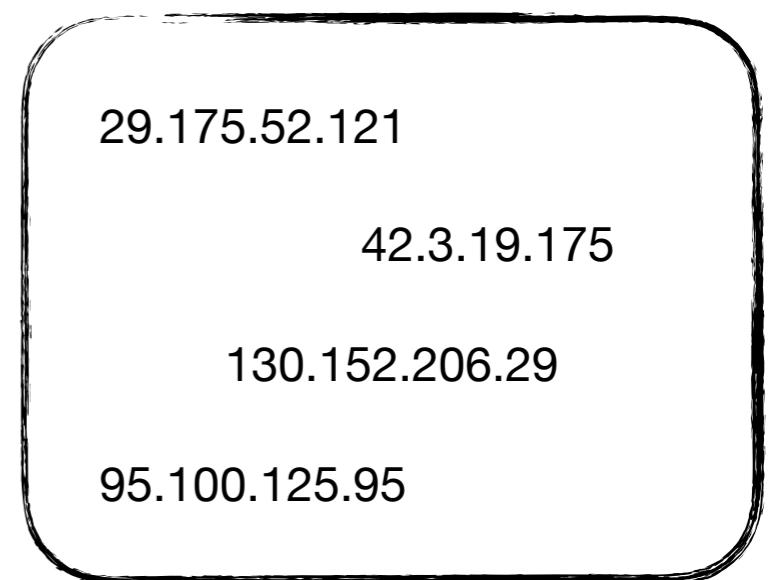
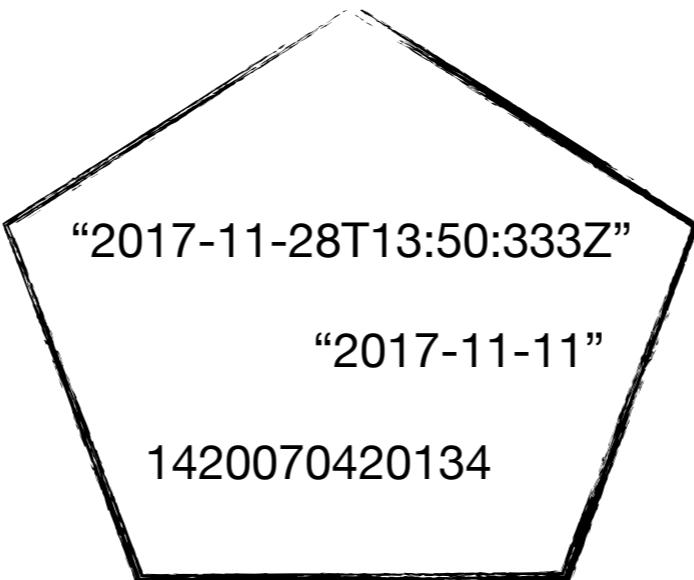
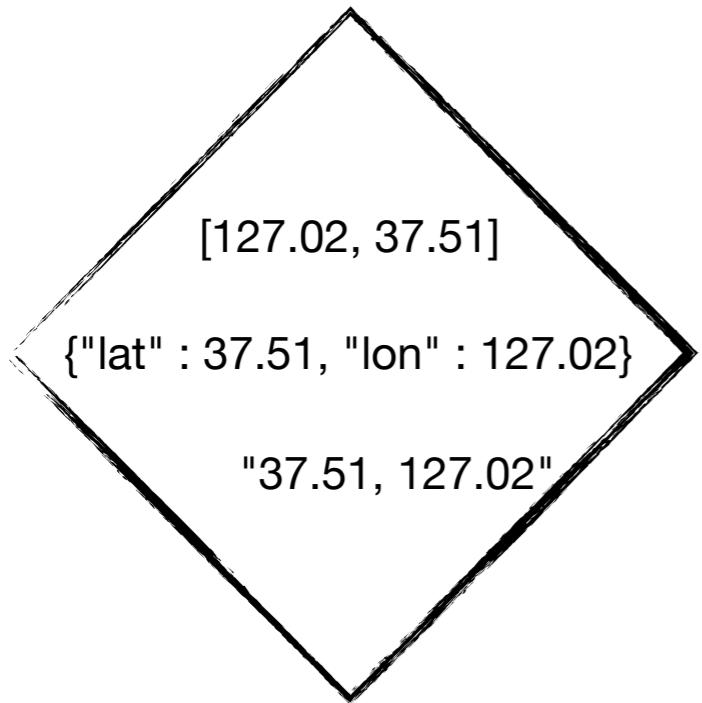
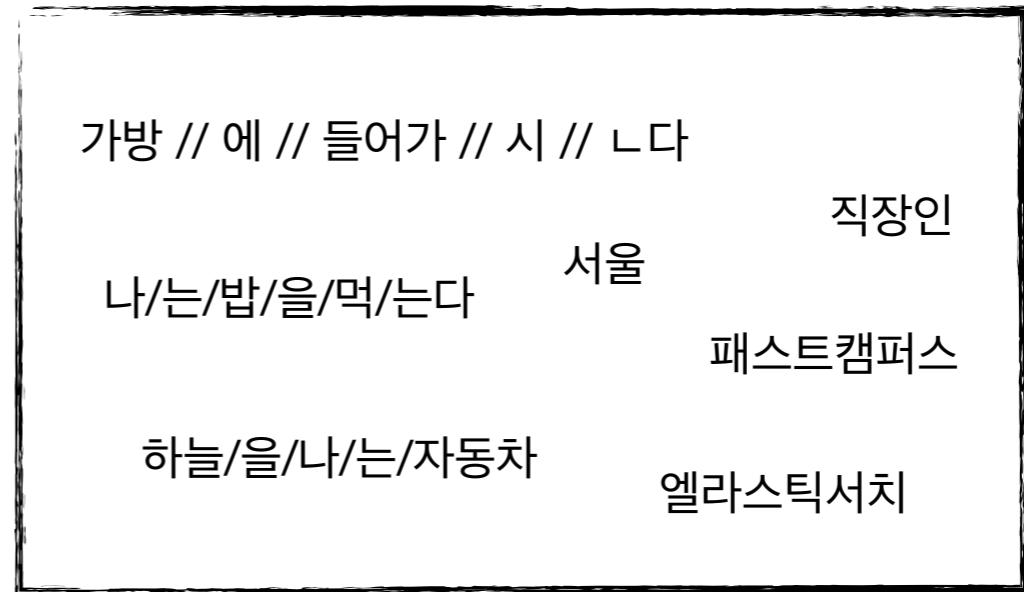
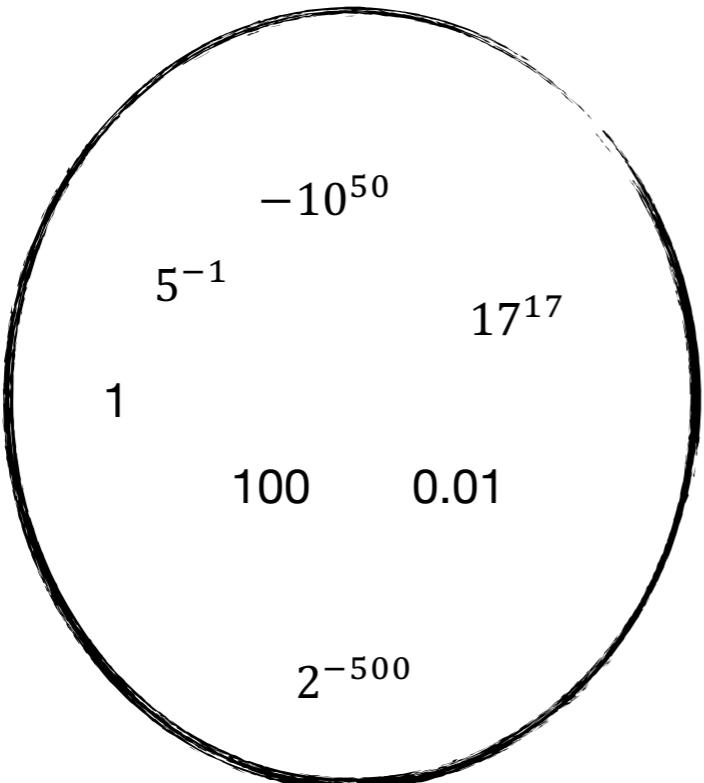
To the right is the **Output Pane**, which displays the results of the Elasticsearch search query. The results are paginated, with page 1 of 5 shown. The results are as follows:

```
1 | {
2 |     "took": 0,
3 |     "timed_out": false,
4 |     "_shards": {
5 |         "total": 5,
6 |         "successful": 5,
7 |         "skipped": 0,
8 |         "failed": 0
9 |     },
10 |     "hits": {
11 |         "total": 20222,
12 |         "max_score": 1,
13 |         "hits": [
14 |             {
15 |                 "_index": "shopping",
16 |                 "_type": "shopping",
17 |                 "_id": "AV-iDKZcRJy4v-Hns1Sk",
18 |                 "_score": 1,
19 |                 "_source": {
20 |                     "접수 번호": "277",
21 |                     "주문 시간": "2016-04-11T04:28:14",
22 |                     "수령 시간": "2016-04-11T07:18:14",
23 |                     "예약 여부": "예약",
24 |                     "배송 메모": "상품 이상",
25 |                     "고객 ip": "130.152.206.29",
26 |                     "고객 성별": "남성",
27 |                     "고객 나이": 40,
28 |                     "물건 좌표": "36.56404885993116, 129.87454556889554",
29 |                     "고객 주소_시도": "서울특별시",
30 |                     "구매 사이트": "g마켓",
31 |                     "판매자 평점": 3,
32 |                     "상품 분류": "스웨터",
33 |                     "상품 가격": 10000,
34 |                     "상품 개수": 1,
35 |                     "결제 카드": "시티"
36 |                 }
37 |             },
38 |             {
39 |                 "_index": "shopping",
40 |                 "_type": "shopping",
41 |                 "_id": "AV-iDKahRJy4v-Hns1Sp",
42 |                 "_score": 1,
```

Below the output pane, there is a navigation bar with icons for History, Settings, and Help, and a large double-headed vertical arrow icon between the panes.

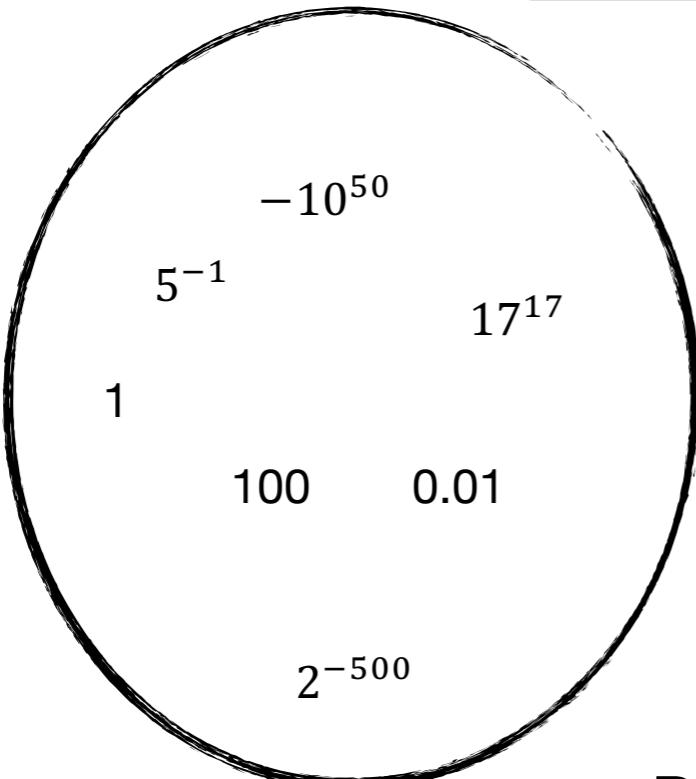
**Dashboard에 데이터를 넣을 때
알면 좋은 (최소한의) Datatype을 살펴보자**

Data Type

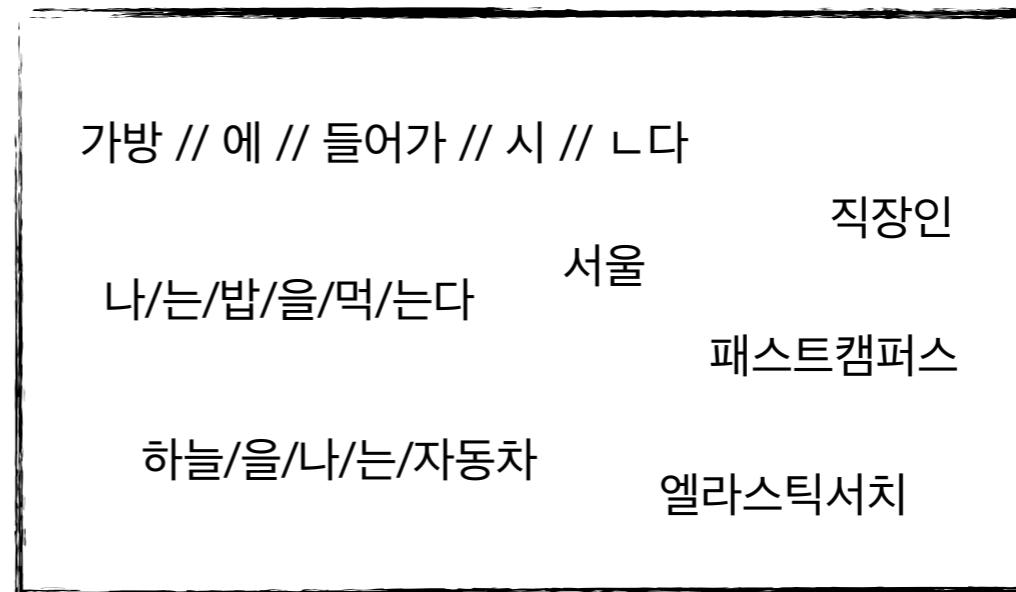


Data Type

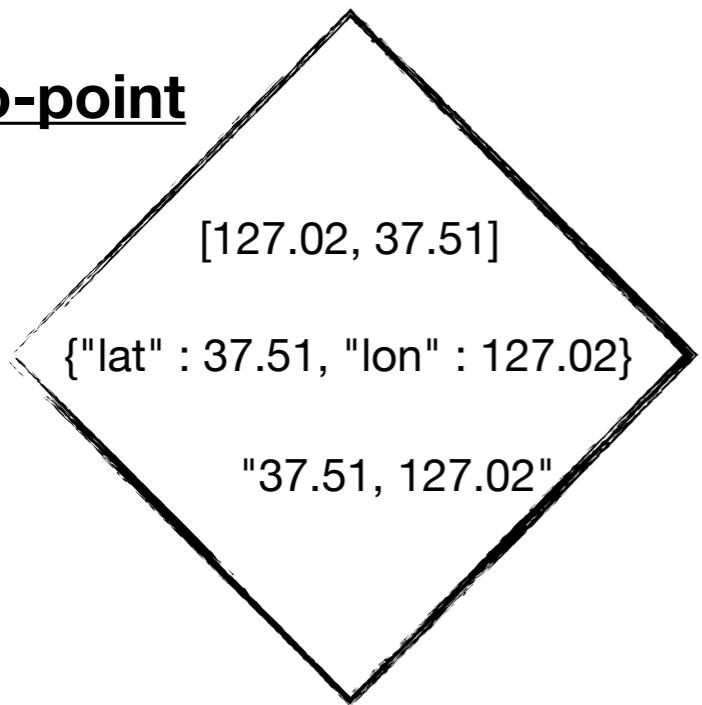
Numeric



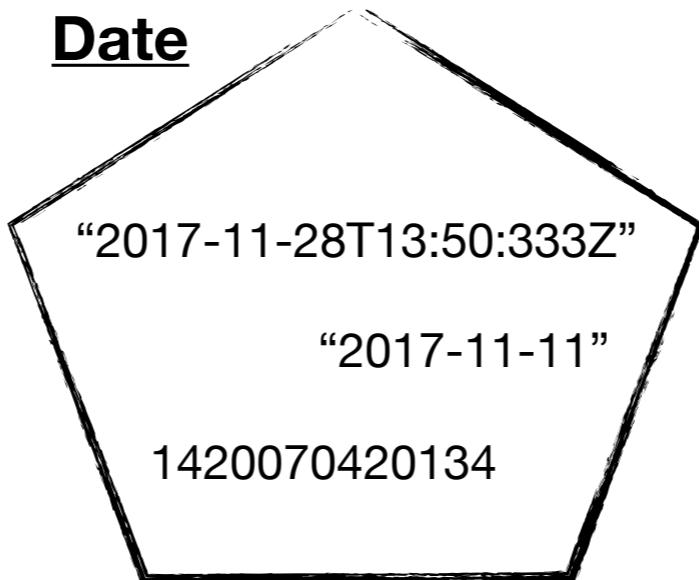
String



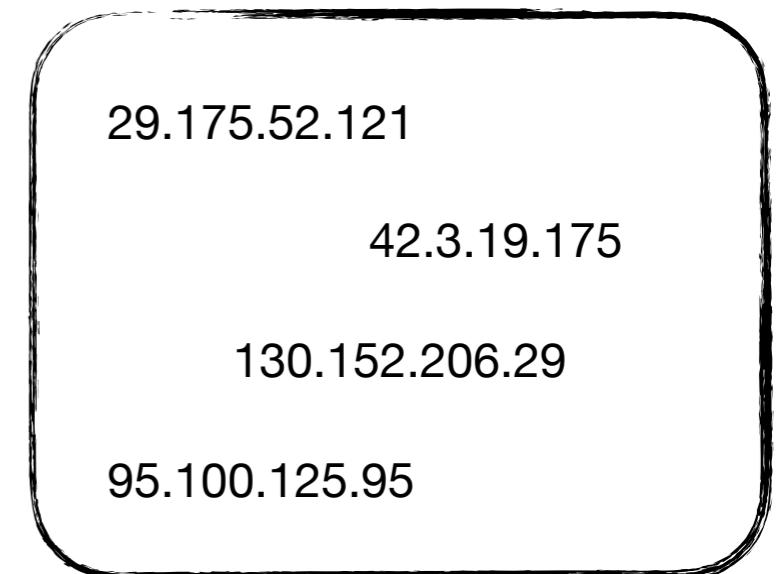
Geo-point



Date



IP



Data Type

이 때 주의할 Type은 **Numeric**과 **String**이다.

Geo-point, Date, IP는 Format이 다양할 뿐 Type 자체는 1개다



Numeric datatypes

The following numeric types are supported:

정수



`long`

A signed 64-bit integer with a minimum value of `-263` and a maximum value of `263-1`.

`integer`

A signed 32-bit integer with a minimum value of `-231` and a maximum value of `231-1`.

`short`

A signed 16-bit integer with a minimum value of `-32,768` and a maximum value of `32,767`.

`byte`

A signed 8-bit integer with a minimum value of `-128` and a maximum value of `127`.

`double`

A double-precision 64-bit IEEE 754 floating point.

`float`

A single-precision 32-bit IEEE 754 floating point.

`half_float`

A half-precision 16-bit IEEE 754 floating point.

`scaled_float`

A floating point that is backed by a `long` and a fixed scaling factor.

부동 소수점





Numeric datatypes

The following numeric types are supported:

`long` A signed 64-bit integer with a minimum value of -2^{63} and a maximum value of $2^{63}-1$.

`integer` A signed 32-bit integer with a minimum value of -2^{31} and a maximum value of $2^{31}-1$.

`short` A signed 16-bit integer with a minimum value of -2^{15} and a maximum value of $2^{15}-1$.

어떤 Type을 사용해야 될까?

`byte` A signed 8-bit integer with a minimum value of -128 and a maximum value of 127 .

`double` A double-precision 64-bit IEEE 754 floating point.

`float` A single-precision 32-bit IEEE 754 floating point.

`half_float` A half-precision 16-bit IEEE 754 floating point.

`scaled_float` A floating point that is backed by a `long` and a fixed scaling factor.

Data Type - Numeric

- 정수형 : 값의 크기에 따라 분류
- 부동소수점 : Precision 정도에 따라 분류



가지고 있는 **Numeric Data**의 성격을 잘 모른다면,
최적화에 집중하기보다 안정적으로 사용할 수 있도록
가장 큰 type을 사용하자

- 정수형 : Long
- 부동소수점 : Double

- 정수형 : 값의 크기에 따라 분류
- 부동소수점 : Precision 정도에 따라 분류



가지고 있는 Numeric Data의 성격을 잘 안다면,
최적화에 집중하자

정수형

- 검색과 색인 성능 향상을 위해 데이터를 담을 수 있는 Smallest Type 고르자
- 다만 실제 저장된 값의 크기에 따라 용량이 정해지므로 어떤 Type을 고르던 영향은 없다
- 범위에 어긋나는 데이터는 저장할 수 없다 (p15)

부동소수점

- 데이터 왜곡을 허용할 수 있는 범위 내의 Smallest Type 고르자
- 어떤 Type을 사용하는지에 따라서 Disk Space에 많은 영향을 준다
- 데이터를 표현하기 부족한 Precision을 선택하면 예기치 않은 일이 생길 수 있다 (p16)

1. 정수 : mapping에서 설정한 정수형 data type 범위 밖의 값을 넣을 경우

```
PUT my_index
{
  "mappings": {
    "my_type": {
      "properties": {
        "test": {
          "type": "byte"
        }
      }
    }
  }
}

POST my_index/my_type
{
  "test" : 129
```

```
{
  "error": {
    "root_cause": [
      {
        "type": "mapper_parsing_exception",
        "reason": "failed to parse [test]"
      }
    ],
    "type": "mapper_parsing_exception",
    "reason": "failed to parse [test]",
    "caused_by": {
      "type": "illegal_argument_exception",
      "reason": "Value [129] is out of range for a byte"
    }
  },
  "status": 400
}
```



2. 부동소수점 : mapping에서 설정한 정수형 data type 범위 밖의 값을 넣을 경우

1) Field 생성

- test-double : double
- test-half-float : half-float

2) 데이터 입력

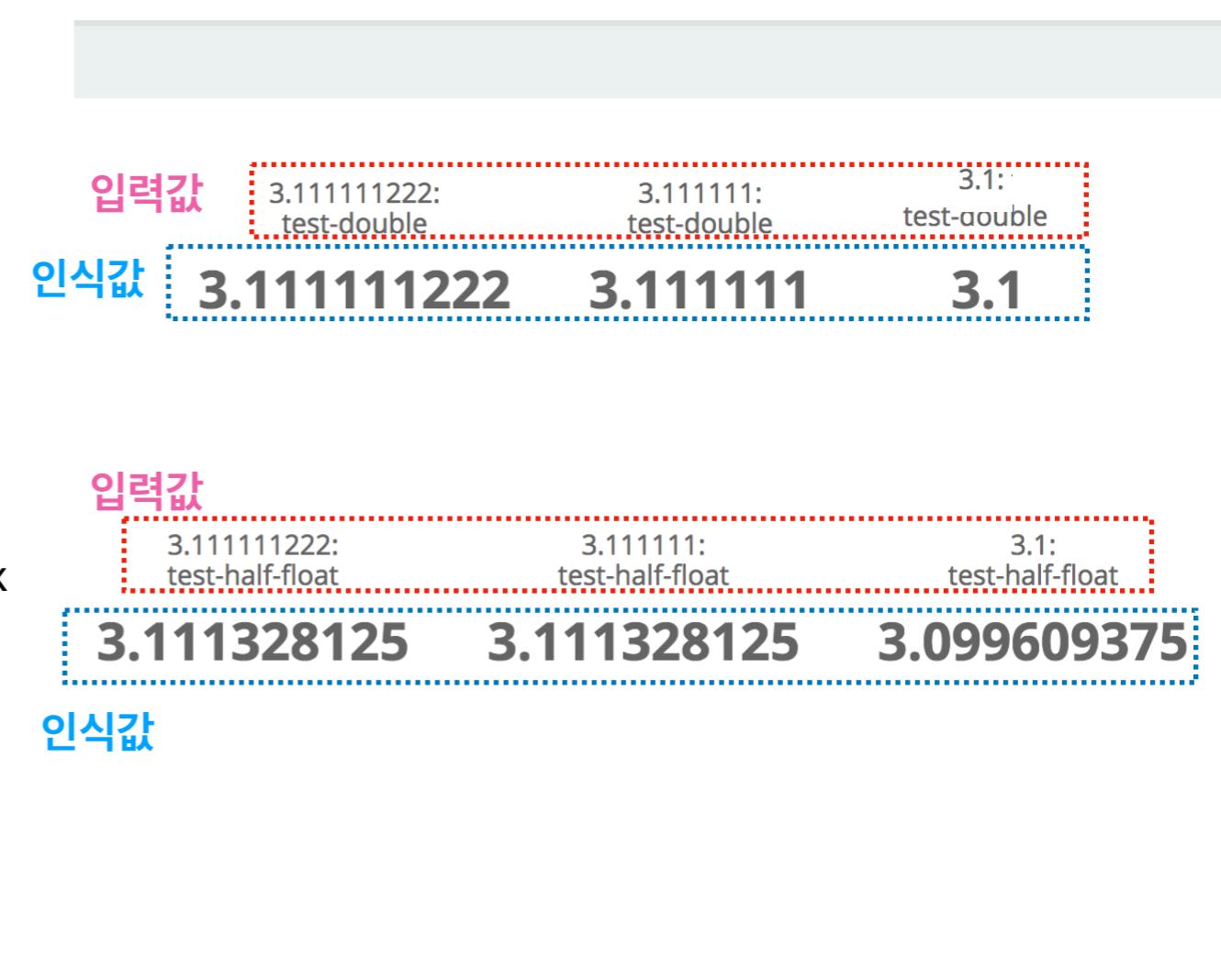
- test-double : 3.1, 3.111111, 3.11111122222
- test-half-float : 3.1, 3.111111, 3.11111122222

3) 데이터 조회

- test-double : 3.111111 ~ 3.1111112 값 조회 => 존재
- test-half-float : 3.111111 ~ 3.1111112 값 조회 => 존재 x

4) 데이터 확인

- test-double : 표현할 수 있는 Precision 내 존재
- test-half-float : 표현할 수 있는 Precision 내 존재 x



**Elasticsearch가 검색엔진인 만큼,
String Field 선택은 매우 중요하다!**

Data Type - String

Keyword : 입력 String Field의 값을 **하나의 단위**로 보고 싶은 경우

Text : 입력 String Field를 **더 작은 단위**로 분석하고 싶은 경우

Keyword로 설정

가방에 들어가신다

가방에 들어가신다

나는 밥을 먹는다

나는 밥을 먹는다

패스트캠퍼스 엘라스틱서치

패스트캠퍼스 엘라스틱서치

Text로 설정 (분석기에 따라 상이)

가방 // 에 // 들어가 // 시 // 냈다

나 // 는 // 밥 // 을 // 먹 // 는다

패스트캠퍼스 // 엘라스틱서치



“패스트캠퍼스 엘라스틱서치” => text field와 keyword field 비교

1) Field 생성

- test-text : text
- test-keyword : keyword

2) 데이터 입력

- test-text : “패스트캠퍼스 엘라스틱서치”
- test-keyword : “패스트캠퍼스 엘라스틱서치”

3) 데이터 조회

- test-text : “패스트캠퍼스” 조회
- test-keyword : “패스트캠퍼스” 조회

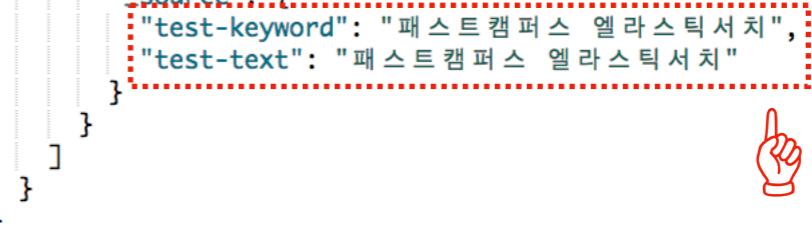
4) 데이터 확인

- test-text : 검색 o
- test-keyword : 검색 x

test-text (검색됨)

```

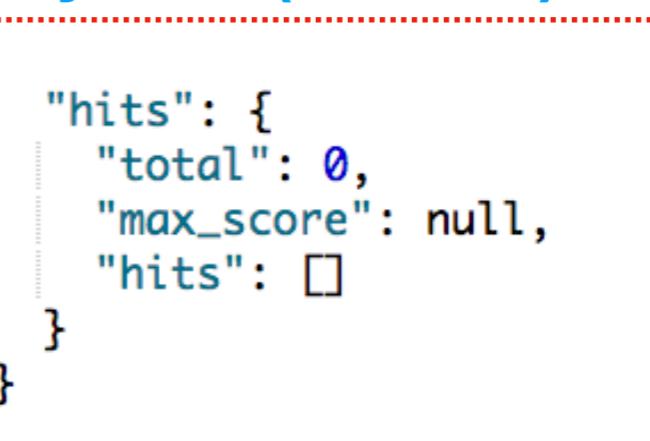
"hits": {
  "total": 1,
  "max_score": 0.25811607,
  "hits": [
    {
      "_index": "my_index",
      "_type": "my_type",
      "_id": "AWE8BViSPloSIAlpN8ut",
      "_score": 0.25811607,
      "source": {
        "test-keyword": "패스트캠퍼스 엘라스틱서치",
        "test-text": "패스트캠퍼스 엘라스틱서치"
      }
    }
  ]
}
  
```



test-keyword (검색 안됨)

```

"hits": {
  "total": 0,
  "max_score": null,
  "hits": []
}
  
```



다양한 Elasticsearch API 중에서

Indices, Document, Search API를 알아보자

API - Indices

Indices API로 무얼 할 수 있을까?



API - Indices

Index 생성

PUT {Index 이름}



PUT week3_higee

API - Indices

Index 삭제

DELETE {Index 이름}



DELETE week3_higee

Mapping 설정은 꼭 필요한가?

Index의 Field들이 어떤 Type으로 저장되는지가 중요한가?

yes no

필요하다

필요 없다

- Mapping이 없어도 에러가 발생하지는 않는다
- 다만 사용자가 원하는 Data Type으로 데이터가 저장된다는 보장이 없다. 예) “2017-01-01 13:00:00”
- 그러므로 (데이터 입력 전에) 가급적 Mapping을 설정하는 걸 권장한다

그렇다면 Mapping은 어느 시점에 어떻게 설정하는가?

- Mapping을 통해 Data Type을 정의하려는 Field에 데이터가 색인되기 전까지는 아무 때나 가능하다
- Mapping을 설정 하기 전에 Data가 색인되면 Elasticsearch가 적당한 Data Type을 부여한다
 - 단, 한 번 설정된 Data Type은 변경이 불가능하다
 - 단, 데이터가 색인된 후에도 새로운 Field에 대한 Mapping은 추가할 수 있다
- 그러므로 일반적으로는 Index 생성하는 시점에 같이 설정하는 걸 권장한다

Index 생성 후 Mapping 추가

```
PUT {Index 이름}
```

1.

```
PUT week3_higee
```



```
PUT {Index 이름}/_mapping/{Type 이름}
{
  "properties": {
    "{Field 이름}" : {
      "type" : "{Field Type}"
    }
  }
}
```

2.

```
PUT week3_higee/_mapping/week3_higee
{
  "properties": {
    "price" : {
      "type" : "integer"
    }
  }
}
```

Index 생성하면서 Mapping 추가

```
PUT {Index 이름}
{
  "mappings": {
    "{Type 이름)": {
      "properties": {
        "{Field1 이름)": {
          "type": "{Field1 Type}"
        },
        "{Field2 이름)": {
          "type": "{Field2 Type}"
        }
      }
    }
  }
}
```



```
PUT week3_higee_mapping
{
  "mappings": {
    "week3_higee_mapping": {
      "properties": {
        "price": {
          "type": "integer"
        },
        "time": {
          "type": "date"
        }
      }
    }
  }
}
```

기존 Mapping에 새로운 Field Mapping 추가하기

```
PUT {Index 이름}/_mapping/{Type 이름}
{
  "properties": {
    "{Field 이름}" : {
      "type" : "{Field Type}"
    }
  }
}
```



```
PUT week3_higee/_mapping/week3_higee
{
  "properties": {
    "age" : {
      "type" : "integer"
    }
  }
}
```

API - Indices

- Template을 생성한다고 **Index가 생성되지는 않는다.**
- Template에서 정의한 Index Pattern에 걸리는 **Index가 생성될 때 Mapping도 생성**

Template 활용해서 자동으로 Mapping 추가

```
PUT _template/{Template 이름}
{
  "template": "{Index Pattern}",
  "mappings": {
    "{Type 이름)": {
      "properties": {
        "Field1 이름": {
          "type": "Field1 Type"
        },
        "Field2 이름": {
          "type": "Field2 Type"
        }
      }
    }
  }
}
```



```
PUT _template/template_higee
{
  "template": "higee-log-*",
  "mappings": {
    "template_higee": {
      "properties": {
        "price": {
          "type": "integer"
        },
        "time": {
          "type": "date"
        }
      }
    }
  }
}
```

그렇다면 Template은 언제 유용할까?

비슷한 이름의 Index가 정기적으로 생성되는 Log Data 등

(higee-log-2017.01.01 higee-log-2017.01.02 higee-log-2017.01.03 ...)

Template을 사용하지 않으면

- 1) 자동으로 생성되는 Mapping을 사용하거나
- 2) 모든 Index마다 직접 Mapping을 추가해야 한다

API - Indices

Mapping 확인

GET {Index 이름}/_mapping



GET **week3_higee**/_mapping

Template Mapping 확인

```
PUT higee-log-2017.01.01
```

```
GET higee-log-2017.01.01/_mapping
```

Document API로 무얼 할 수 있을까?

Document API로 무얼 할 수 있을까?

- **Document 생성 ©**
- **Document 조회 ®**
- **Document 수정 ℗**
- **Document 삭제 ®**
- **Document 복사**

Document 추가 (지정 ID)

```
PUT {Index 이름}/{Type 이름}/{ID}  
{  
    "{Field 이름}" : {Value}  
}
```

```
PUT week3_higee/week3_higee/1  
{  
    "price" : 10000,  
    "age" : 17  
}
```

```
PUT week3_higee/week3_higee/2  
{  
    "price" : 2000,  
    "age" : 20  
}
```

```
PUT week3_higee/week3_higee/3  
{  
    "price" : 1000,  
    "age" : 25  
}
```

```
PUT week3_higee/week3_higee/4  
{  
    "price" : 7000,  
    "age" : 33  
}
```

Document 추가 (임의 ID)

```
POST {Index 이름}/{Type 이름}  
{  
  "{Field 이름}" : {Value}  
}
```



```
POST week3_higee/week3_higee  
{  
  "price" : 5000,  
  "age" : 19  
}
```

ID로 Document 조회

* Query로 Document 조회하는 건 Search API

```
GET {Index 이름}/{Type 이름}/{ID} → GET week3_higee/week3_higee/1
```

ID로 Document 삭제

```
DELETE {Index 이름}/{Type 이름}/{ID} → DELETE week3_higee/week3_higee/1
```

Query로 Document 삭제

```
POST {Index 이름}/_delete_by_query
{
  "query": {
    "match": {
      "{Field 이름)": "{Value}"
    }
  }
}
```



```
POST week3_higee/_delete_by_query
{
  "query": {
    "match": {
      "age": 20
    }
  }
}
```

ID로 Document 부분 수정

```
POST {Index 이름}/{Type 이름}/{ID}/_update  
{  
  "doc": {  
    "{Field}" : {Value}  
  }  
}
```



```
POST week3_higee/week3_higee/3/_update  
{  
  "doc": {  
    "age" : 50  
  }  
}
```

ID로 Document 전체 수정

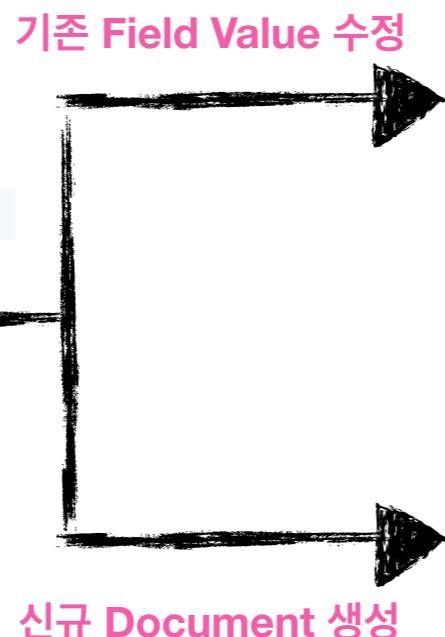
```
PUT {Index 이름}/{Type 이름}/{ID}  
{  
  "{Field}" : {Value}  
}
```



```
PUT week3_higee/week3_higee/3  
{  
  "warning" : "해당 Document 전체 변경"  
}
```

ID로 Document 수정 (Upsert)

```
POST {Index 이름}/{Type 이름}/{ID}/_update
{
  "doc" : {
    "{Field 이름}" : "{Value}"
  },
  "doc_as_upsert" : true
}
```



```
POST week3_higee/week3_higee/4/_update
{
  "doc" : {
    "price" : 50000
  },
  "doc_as_upsert" : true
}
```

```
POST week3_higee/week3_higee/777/_update
{
  "doc" : {
    "price" : 50000
  },
  "doc_as_upsert" : true
}
```

Query로 Document 수정

```
POST {Index 이름}/{Type 이름}/_update_by_query
{
  "script": {
    "source": "ctx._source[{Field 이름}] = Value"
  },
  "query": {
    "term": {
      "{Field 이름)": "Value"
    }
  }
}
```

1. POST `week3_higee/week3_higee/_update_by_query`

```
{
  "script": {
    "source": "ctx._source['age'] = 50"
  },
  "query": {
    "term": {
      "age": 33
    }
  }
}
```



```
POST {Index 이름}/{Type 이름}/_update_by_query
{
  "script": {
    "source": "ctx._source.{Field 이름} = Value"
  },
  "query": {
    "term": {
      "{Field 이름)": "Value"
    }
  }
}
```

2. POST `week3_higee/week3_higee/_update_by_query`

```
{
  "script": {
    "source": "ctx._source.age = 70"
  },
  "query": {
    "term": {
      "age": 50
    }
  }
}
```

Index 내 모든 Document 복사

```
POST _reindex
{
  "source": {
    "index": "{복사하려는 원본 Index 이름}"
  },
  "dest": {
    "index": "{복사본을 저장할 Index 이름}"
  }
}
```



```
POST _reindex
{
  "source": {
    "index": "week3_higee"
  },
  "dest": {
    "index": "week3_higee_reindex"
  }
}
```

Reindex 사용 시 주의할 점

- Reindex 하는 순간 Destination Index는 생성된다
- Reindex는 순전히 Documents만 복사된다
- 그 외 Index 설정은 복사가 되지 않으므로 Destination Index 설정을 끝낸 후에 Reindex 사용 권장한다
- 즉, 가장 중요한 설정 중 하나인 **Mapping은 Reindex 전에 꼭 하기를 권장**한다

Index 내 일부 Document 복사

```
POST _reindex
```

```
{  
  "source": {  
    "index": "{복사하려는 Index 이름}",  
    "type" : "{복사하려는 Type 이름}",  
    "query": {  
      "term": {  
        "{Field 이름}": "{Value}"  
      }  
    }  
  },  
  "dest": {  
    "index": "{복사본을 저장할 Index 이름}"  
  }  
}
```



```
POST _reindex
```

```
{  
  "source": {  
    "index": "week3_higee",  
    "type" : "week3_higee",  
    "query": {  
      "term": {  
        "age": 19  
      }  
    }  
  },  
  "dest": {  
    "index": "week3_higee_reindex2"  
  }  
}
```



Request Body Search에서 사용되는 Domain Specific Language

Search API (특히 **Query DSL**)로 무얼 할 수 있을까?

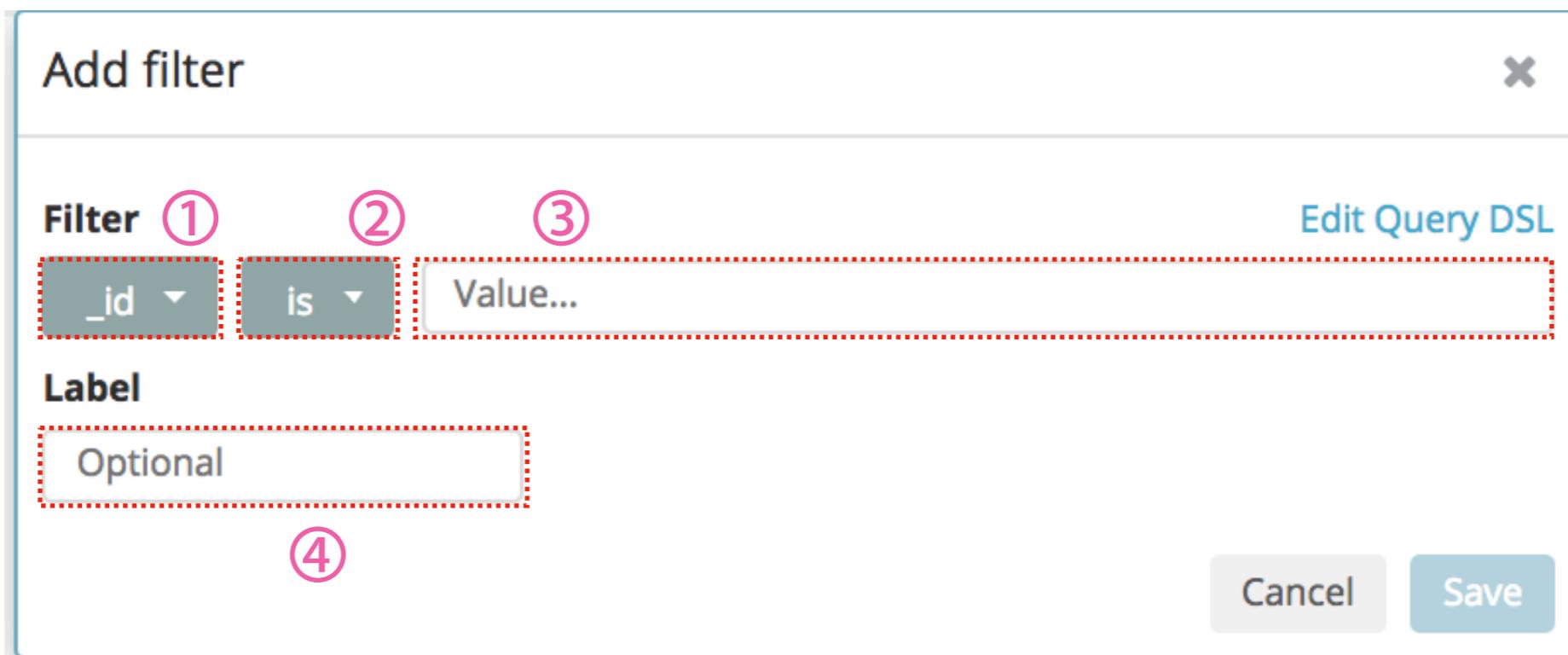
Match All Query	Full Text Queries	Term Level Queries	Specialized Queries	Compound Queries
match-all	query-string ⋮	exists fuzzy prefix range term terms wildcard ⋮	script ⋮	bool ⋮

간략하나마 뭘 위한건지는 알겠는데

Dashboard를 구축/운영하는데 왜 필요하지?

Filter 기능 강화를 위해서

Filter를 다시 보자



- ① Filter 적용할 Field 선택
- ② 적용할 Operator 선택 (다음 페이지 참조)
- ③ Filter에 적용하려는 Value 입력
- ④ (여러 Filter 구분하기 위한) 이름 입력

Filter를 다시 보자



Add filter

Filter ① ② ③

_id ▾ is ▾ Value...

Label

Optional

④

Cancel Save

A screenshot of a 'Add filter' dialog box. At the top left is a 'Help' icon (a question mark) and a close button (an 'X'). Below the title 'Add filter' is a section labeled 'Filter' with three numbered circles (1, 2, 3) above them. Circle 1 points to a dropdown menu with '_id' selected. Circle 2 points to another dropdown menu with 'is' selected. Circle 3 points to a text input field labeled 'Value...'. To the right of this section is a red-bordered button labeled 'Edit Query DSL'. Below this is a 'Label' section with a red-bordered input field labeled 'Optional'. At the bottom right are 'Cancel' and 'Save' buttons. A large red circle labeled '④' is positioned at the bottom center of the dialog.

☞ 클릭

- ① Filter 적용할 Field 선택
- ② 적용할 Operator 선택 (다음 페이지 참조)
- ③ Filter에 적용하려는 Value 입력
- ④ (여러 Filter 구분하기 위한) 이름 입력

Edit Query DSL

Add filter ×

Filter

Search filter values

1 { }

이 부분에 **Query DSL**을 활용해서
Filter를 생성할 수 있다.

Filters are built using the [Elasticsearch Query DSL](#).

Label

Optional

Cancel

Save

Filter + Query DSL을 이용하면

		Filter + Query DSL	Filter	Search
“고객성별”이 여성인 Data		✓	✓	✓
“결제카드”가 우리 또는 국민인 Data		✓	✓	✓
“고객성별”이 남성이면서 “연령대”가 20대	SCRIPTED FIELD	✓	✓	
“구매사이트”가 쿠팡이거나 “상품개수”가 1~3인 Data	OR 연산	✓		✓
“결제카드”가 “우”로 시작하는 모든 Data	WILDCARD 검색	✓		✓
“구매사이트”가 22번가(오타 아니에요)와 유사한 Data	FUZZY / PROXIMITY 검색	✓		✓



- 지난 수업 때 얘기한 Filter는 Query DSL 제외
- 수업 진도상 표기일 뿐 실제로는 둘 다 Filter로 본다

모든 Documents 조회

```
GET {Index 이름}/{Type 이름}/_search
{
  "query" : {
    "match_all" : {}
  }
}
```



```
GET shopping/shopping/_search
{
  "query" : {
    "match_all" : {}
  }
}
```

API - Search

```
{  
① "took": 0,          ②  
  "timed_out": false,  
  "_shards": {  
    "total": 5,  
    "successful": 5,  
    "skipped": 0,  
    "failed": 0  
  },                ③  
  "hits": {  
    ④ "total": 20222,  
      "max_score": 1,  
      "hits": [  
        {  
          ⑤ "_index": "shopping",  
            "_type": "shopping",  
            "_id": "AV-iDKZcRJy4v-Hns1Sk",  
            "_score": 1,  
            "_source": {  
              "접수번호": "277",  
              "주문시간": "2016-04-11T04:28:14",  
              "고객ip": "130.152.206.29",  
              "물건좌표": "36.56, 129.87",  
              "판매자평점": 3,  
              "상품분류": "스웨터",  
              "상품가격": 10000,  
            }  
          }  
        ]  
      }  
    }  
}
```



- ① Elasticsearch 검색 소요시간 (millisecond)
- ② 검색결과가 time out에 걸렸는지 표시
- ③ 몇 개의 shards가 검색되었는지 표시
- ④ 검색된 Documents의 개수
- ⑤ 실제 Documents 내용

검색어와 정확히 일치하는 Document 조회

```
GET {Index 이름}/{Type 이름}/_search
{
  "query" : {
    "term" : {
      "{Field 이름}" : "{Value}"
    }
  }
}
```



```
GET shopping/shopping/_search
{
  "query" : {
    "term" : {
      "상품분류" : "셔츠"
    }
  }
}
```

검색어 중 적어도 1개와 정확히 일치하는 Document 조회

```
GET {Index 이름}/{Type 이름}/_search
{
  "query" : {
    "terms" : {
      "{Field 이름}" : ["{Value}", "{Value}"]
    }
  }
}
```



```
GET shopping/shopping/_search
{
  "query" : {
    "terms" : {
      "상품분류" : ["셔츠", "스웨터"]
    }
  }
}
```

특정 접두어로 시작하는 Document 조회

```
GET {Index 이름}/{Type 이름}/_search
{
  "query": {
    "prefix" : {
      "{Field 이름}" : "{Value}"
    }
  }
}
```



```
GET shopping/_search
{
  "query": {
    "prefix" : {
      "고객주소_시도" : "경상"
    }
  }
}
```

Wildcard Expression 만족하는 Documents 조회

```
GET {Index 이름}/{Type 이름}/_search
{
  "query": {
    "wildcard" : {
      "{Field 이름}" : "{Value}"
    }
  }
}
```



```
GET shopping/shopping/_search
{
  "query": {
    "wildcard" : {
      "고객주소_시도" : "경*도"
    }
  }
}
```

```
GET shopping/shopping/_search
{
  "query": {
    "wildcard" : {
      "고객주소_시도" : "경?도"
    }
  }
}
```

검색어와 유사한 Documents 조회

```
GET {Index 이름}/{Type 이름}/_search
{
  "query": {
    "fuzzy" : {
      "{Field 이름}" : "{Value}"
    }
  }
}
```



```
GET shopping/_search
{
  "query": {
    "fuzzy" : {
      "고객주소_시도" : {
        "value" : "경상북남"
      }
    }
  }
}
```

특정 Numeric Field가 임의의 범위 내에 있는 Documents 조회

```
GET {Index 이름}/{Type 이름}/_search
{
  "query": {
    "range": {
      "{Field 이름)": {
        "gte": "{Value}",
        "lte": "{Value}",
      }
    }
  }
}
```



```
GET shopping/_search
{
  "query": {
    "range": {
      "주문시간": {
        "gte": "2017-02-15"
      }
    }
  }
}
```

Lucene Query Syntax를 만족하는 Documents 조회

```
GET {Index 이름}/{Type 이름}/_search
{
  "query" : {
    "query_string" : {
      "query" : "{LUCENE QUERY}"
    }
  }
}
```



```
GET shopping/shopping/_search
{
  "query" : {
    "query_string" : {
      "query": "고객나이 : [10 TO 25]"
    }
  }
}
```



Query String Syntax

non-null value가 존재하는 Documents 조회

```
GET {Index 이름}/{Type 이름}/_search
{
  "query": {
    "exists" : {
      "field" : "{Field 이름}"
    }
  }
}
```



```
GET shopping/shopping/_search
{
  "query": {
    "exists" : {
      "field" : "상품분류"
    }
  }
}
```

Scripted Field가 특정 조건을 만족하는 Document 조회

The screenshot shows the Kibana Management interface with the 'Management / Kibana' header. The left sidebar includes options like Discover, Visualize, Dashboard, Timeline, Dev Tools, and Management. The main area displays the 'shopping' index pattern. A green banner at the top indicates a 'Time Filter field name: 주문시간'. Below it, a note states: 'This page lists every field in the **shopping** index and the field's associated core type as recorded by Elasticsearch. While this list allows you to view the core type of each field, changing field types must be done using Elasticsearch's Mapping API %'. The 'Scripted fields' section is highlighted with a red dashed box. It contains 16 entries:

name	lang	script	format	controls
시간대	painless	doc['주문시간'].date.hourOfDay		
배송소요시	painless	(doc['수령시간'].value-doc['주문시간'].value)/1000/60/60		
간				
요일	painless	(doc['주문시간'].date.dayOfWeek == 1 ? '월' : (doc['주문시간'].date.dayOfWeek == 2 ? '화' : ((doc['주문시간'].date.dayOfWeek == 3 ? '수' : ((doc['주문시간'].date.dayOfWeek == 4 ? '목' : ((doc['주문시간'].date.dayOfWeek == 5 ? '금' : ((doc['주문시간'].date.dayOfWeek == 6 ? '토' : '일'))))))))))		
요일_sort	painless	doc['주문시간'].date.dayOfWeek		
연령대	painless	if (doc['고객나이'].value < 20) { return "10대" } else if (doc['고객나이'].value < 30) { return "20대" } else if (doc['고객나이'].value < 40) { return "30대" } else if (doc['고객나이'].value < 50) { return "40대" } return "50대 이상"		
배송소요일	painless	if ((doc['수령시간'].value-doc['주문시간'].value)/1000/60/60 < 24) { return "당일" } else if ((doc['수령시간'].value-doc['주문시간'].value)/1000/60/60 < 48) { return "다음날" } else if ((doc['수령시간'].value-doc['주문시간'].value)/1000/60/60 < 72) { return "모레" } return "나흘 이상"		
수				
배송소요일	painless	if ((doc['수령시간'].value-doc['주문시간'].value)/1000/60/60 < 24) { return 0 } else if ((doc['수령시간'].value-doc['주문시간'].value)/1000/60/60 < 48) { return 1 } else if ((doc['수령시간'].value-doc['주문시간'].value)/1000/60/60 < 72) { return 2 } return 3		
수_sort				
성별-카드	painless	doc['고객성별'].value + '-' + doc['결제카드'].value	String	
결제카드-	painless	doc['결제카드'].value + '카드'	String	
배송소요시	painless	(doc['수령시간'].value-doc['주문시간'].value)/1000/60/60	Number	
간-				
소비행태	painless	if (doc['상품개수'].value < 3) { return "저소비" } else if (doc['고객나이'].value < 6) { return "평균" } return "과소비"	String	
요일-	painless	doc['주문시간'].date.dayOfWeek		
성별카드가	painless	doc['고객성별'].value + '-' + doc['결제카드'].value	String	
자				
결제카드풀	painless	doc['결제카드'].value + "카드"	String	
네이				



위에서 생성했던 Scripted Field를 직접 사용하지는 못한다

Scripted Field가 특정 조건을 만족하는 Document 조회

```
GET {Index 이름}/{Type 이름}/_search
{
  "query": {
    "script": {
      "script": {
        "source": "{Script Field 및 조건}",
        "lang": "painless"
      }
    }
  }
}
```

즉, Scripted Field 생성하기 위해 사용했던 코드를 직접 입력하라는 것이다

Script Query가 특정 조건을 만족하는 Document 조회



```
GET shopping/_search
{
  "query": {
    "script": {
      "script": {
        "source": "doc['주문시간'].date.hourOfDay > 15",
        "lang": "painless"
      }
    }
  }
}
```

조건

👉 Scripted Field 생성 Source

여러가지 Query를 복합적으로 사용할 수 있을까?

- A : 고객주소_시도 = 서울특별시
- B : 구매사이트 = 11로 시작
- C : 고객나이 < 30
- D : 주문날짜 = 일요일

Term Query
Wildcard Query
Range Query
Script Query



위의 Query를 아래와 같은 조건으로 검색 가능

- A AND B
- A AND NOT B
- A OR B
- A AND (B OR C)
- A AND (B OR C OR D 중 2개 이상 만족)

:

Bool Query의 Occurrence Type을 알아보자

(정확히 일치하지는 않는다)

Bool Query Occurrence

Logical Statement

must

AND

must_not

NOT

should

OR

기본 구조는 다음과 같다 (Term Query 예시)

```
GET {Index 이름}/{Type 이름}/_search
```

```
{  
  "query": {  
    "bool": {  
      "must": [  
        {  
          "term": {  
            "고객주소_시도": "서울특별시"  
          }  
        }  
      ],  
      "must_not": [  
        {  
          "term": {  
            "상품분류": "셔츠"  
          }  
        }  
      ],  
      "should": [  
        {  
          "term": {  
            "결제카드": "시티"  
          }  
        }  
      ],  
      "minimum_should_match": 1  
    }  
  }  
}
```

👉 반드시 만족해야 한다

👉 반드시 만족하면 안된다

👉 {minimum_should_match}개 이상 만족해야 한다

👉 should clause 내의 query가 n개 이상 참이어야 한다

1. A AND B를 구해보자

A : 고객주소_시도 = 서울특별시
B : 구매사이트 = 11로 시작

```
GET shopping/_search
{
  "query": {
    "bool": {
      "must": [
        { "term": { "고객주소_시도": "서울특별시" } },
        { "wildcard": { "구매사이트": "11*" } }
      ]
    }
  }
}
```

2. A AND NOT B를 구해보자

A : 고객주소_시도 = 서울특별시
B : 구매사이트 = 11로 시작

```
GET shopping/_search
{
  "query": {
    "bool": {
      "must": [
        { "term": { "고객주소_시도": "서울특별시" } }
      ],
      "must_not": [
        { "wildcard": { "구매사이트" : "11*" } }
      ]
    }
  }
}
```

3. A OR B를 구해보자

A : 고객주소_시도 = 서울특별시
B : 구매사이트 = 11로 시작

```
GET shopping/_search
{
  "query": {
    "bool": {
      "should": [
        { "term": { "고객주소_시도": "서울특별시" } },
        { "wildcard": { "구매사이트" : "11*" } }
      ],
      "minimum_should_match": 1
    }
  }
}
```

4. A AND (B OR C)를 구해보자

```
GET shopping/_search
{
  "query": {
    "bool": {
      "must": [
        { "term": { "고객주소_시도": "서울특별시" } }
      ],
      "should": [
        { "wildcard": { "구매사이트" : "11*" } },
        { "range": { "고객나이": { "lt": 30 } } }
      ],
      "minimum_should_match": 1
    }
  }
}
```

A : 고객주소_시도 = 서울특별시
B : 구매사이트 = 11로 시작
C : 고객나이 < 30

5. A AND (B OR C OR D 중 2개 이상) 를 구해보자

```
GET shopping/_search
```

```
{  
  "query": {  
    "bool": {  
      "must": [  
        { "term": { "고객주소_시도": "서울특별시" }}  
      ],  
      "should": [  
        { "wildcard": { "구매사이트" : "11*"}},  
        { "range": { "고객나이": { "lt": 30}}},  
        { "script": { "script" : { "source" : "doc['주문시간'].date.dayOfWeek == 7"}}}  
      ],  
      "minimum_should_match": 2  
    }  
  }  
}
```

A : 고객주소_시도 = 서울특별시
B : 구매사이트 = 11로 시작
C : 고객나이 < 30
D : 주문날짜 = 일요일

아직 Bool Query가 익숙하지 않으면 우선 query string으로 시작하자  

(다만 기능이 제한적이다)

```
GET shopping/shopping/_search
{
  "query": {
    "query_string": {
      "query": "고객나이 : [10 TO 25] OR 구매사이트: 쿠팡",
      "analyze_wildcard": true
    }
  }
}
```

모든 Query를 외워야 되나?

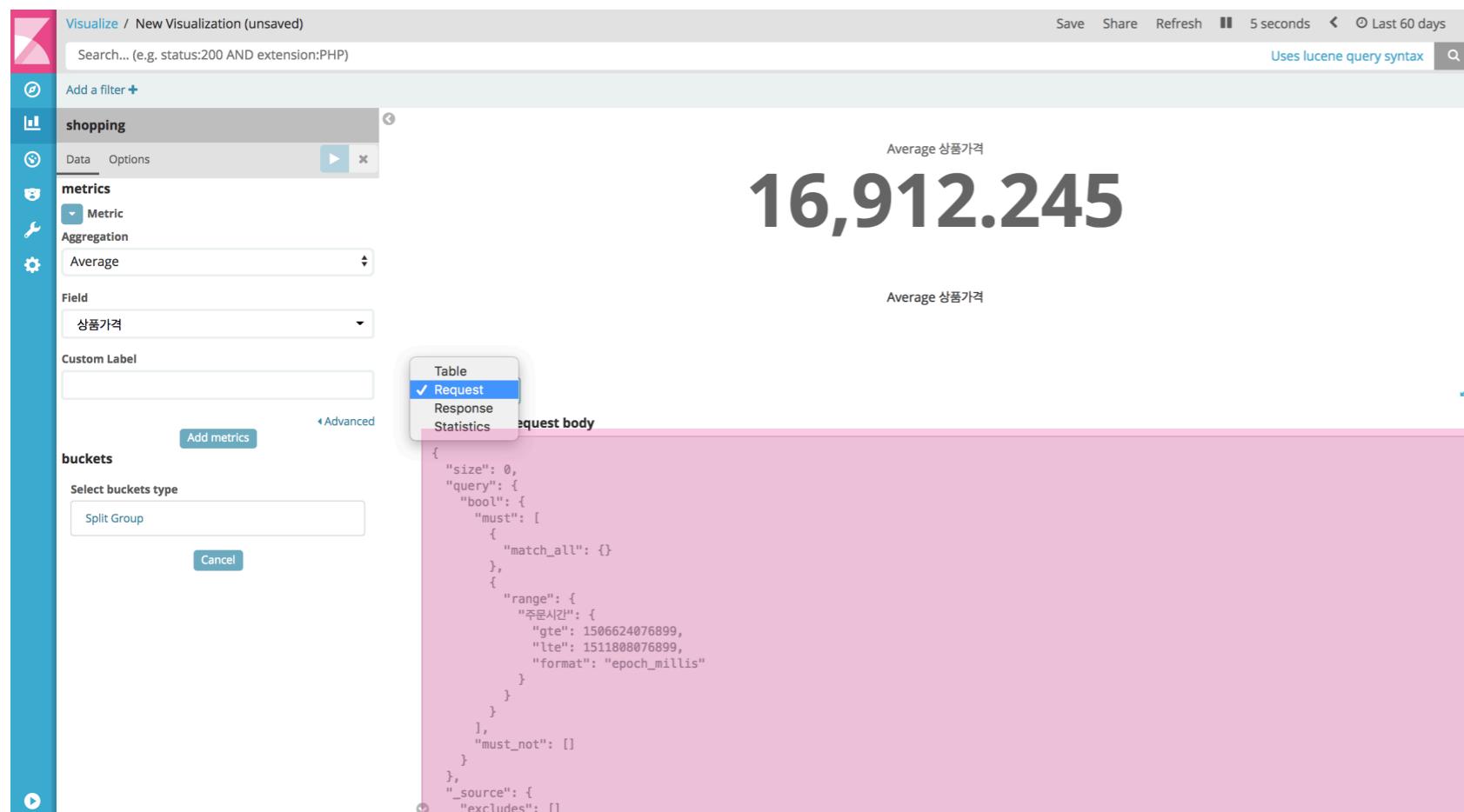
(Elasticsearch Query 결과를 시각화하는)

Kibana를 적극 활용하자 

최근 60일 동안의 상품가격 Field의 평균을 구한다고 하자 

문제는 Aggregation Query는 아직 접하지도 않았다는 것이다.

이 때 **Visualization Spy**를 보면 복사/붙여넣기가 가능한 코드를 볼 수 있다



Average 상품가격
16,912.245

Average 상품가격

Request

```
{
  "size": 0,
  "query": {
    "bool": {
      "must": [
        {
          "match_all": {}
        },
        {
          "range": {
            "주문시간": {
              "gte": 1506624076899,
              "lte": 1511808076899,
              "format": "epoch_millis"
            }
          }
        }
      ],
      "must_not": []
    },
    "_source": {
      "excludes": []
    }
  }
}
```

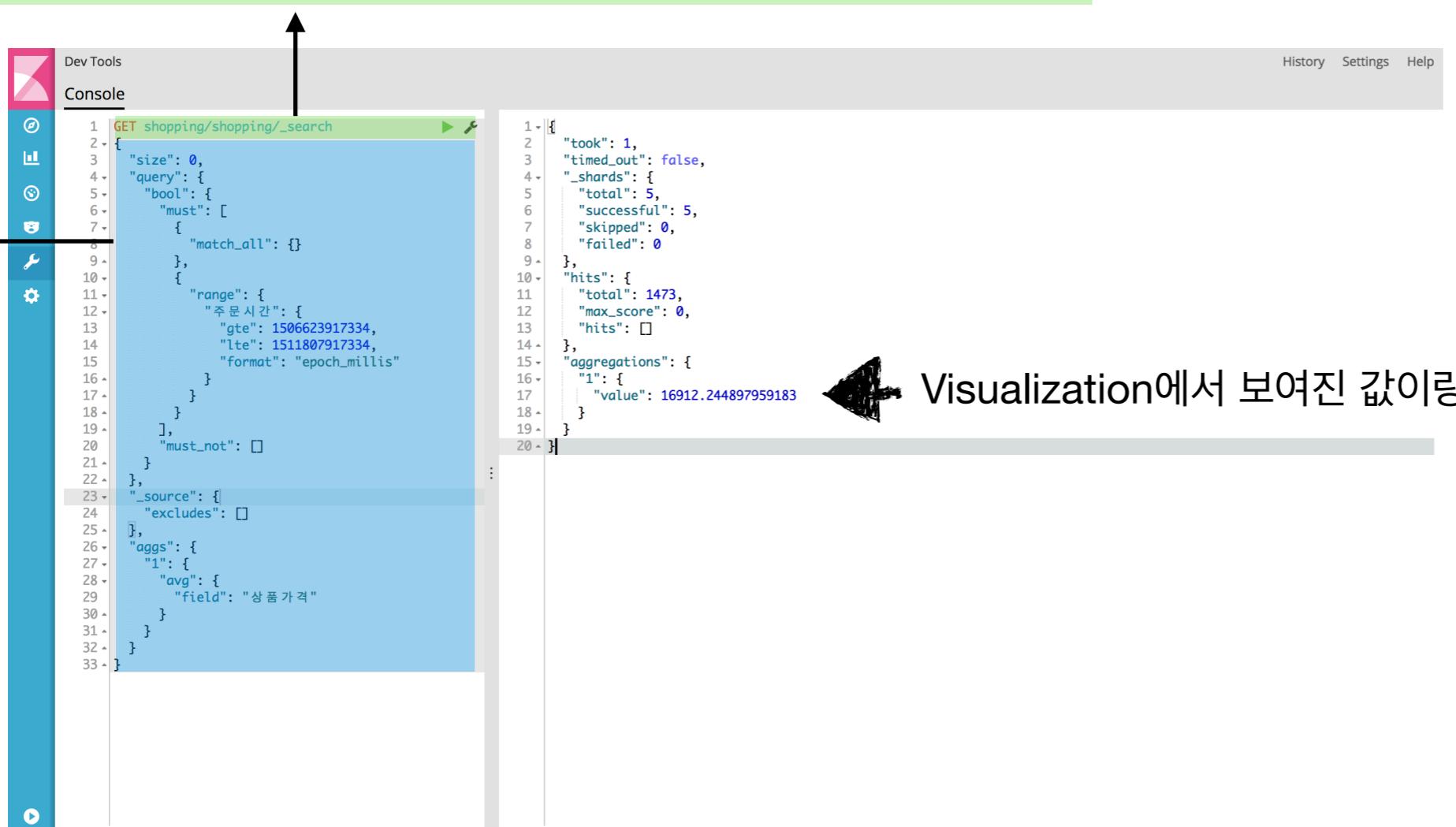
Elasticsearch Request Body!!!

앞장에서 봤던 Elasticsearch Request Body 전체를 복사하자 (Ctrl + C)

그리고 Dev Tools로 이동하자

→ 그 다음 복사한 Elasticsearch Request Body를 붙여넣자

마지막으로 가장 윗줄에 GET method를 붙여주자



The screenshot shows the Elasticsearch Dev Tools interface. On the left, there's a sidebar with various icons. The main area is titled 'Console' and contains a code editor with the following JSON query:

```
1 GET shopping/shopping/_search
2 {
3   "size": 0,
4   "query": {
5     "bool": {
6       "must": [
7         {
8           "match_all": {}
9         },
10        {
11          "range": {
12            "주문 시간": {
13              "gte": 1506623917334,
14              "lte": 1511807917334,
15              "format": "epoch_millis"
16            }
17          }
18        ],
19        "must_not": []
20      }
21    },
22    "_source": {
23      "excludes": []
24    },
25    "aggs": {
26      "1": {
27        "avg": {
28          "field": "상품 가격"
29        }
30      }
31    }
32  }
```

The results of the search are displayed on the right, showing the following JSON response:

```
1 {
2   "took": 1,
3   "timed_out": false,
4   "_shards": {
5     "total": 5,
6     "successful": 5,
7     "skipped": 0,
8     "failed": 0
9   },
10  "hits": {
11    "total": 1473,
12    "max_score": 0,
13    "hits": []
14  },
15  "aggregations": {
16    "1": {
17      "value": 16912.244897959183
18    }
19  }
20 }
```

A large black arrow points from the text 'Visualization에서 보여진 값이랑 일치한다!' to the 'value' field in the aggregations section of the response.

그렇다면 Sum Aggregation은 어떻게 할까?  

API

```
GET shopping/shopping/_search
```

```
{  
  "size": 0,  
  "query": {  
    "bool": {  
      "must": [  
        {  
          "match_all": {}  
        },  
        {  
          "range": {  
            "주문시간": {  
              "gte": 1506623917334,  
              "lte": 1511807917334,  
              "format": "epoch_millis"  
            }  
          }  
        }  
      ],  
      "must_not": []  
    }  
  },  
  "_source": {  
    "excludes": []  
  },  
  "aggs": {  
    "1": {  
      "avg": {  
        "field": "상품가격"  
      }  
    }  
  }  
}
```

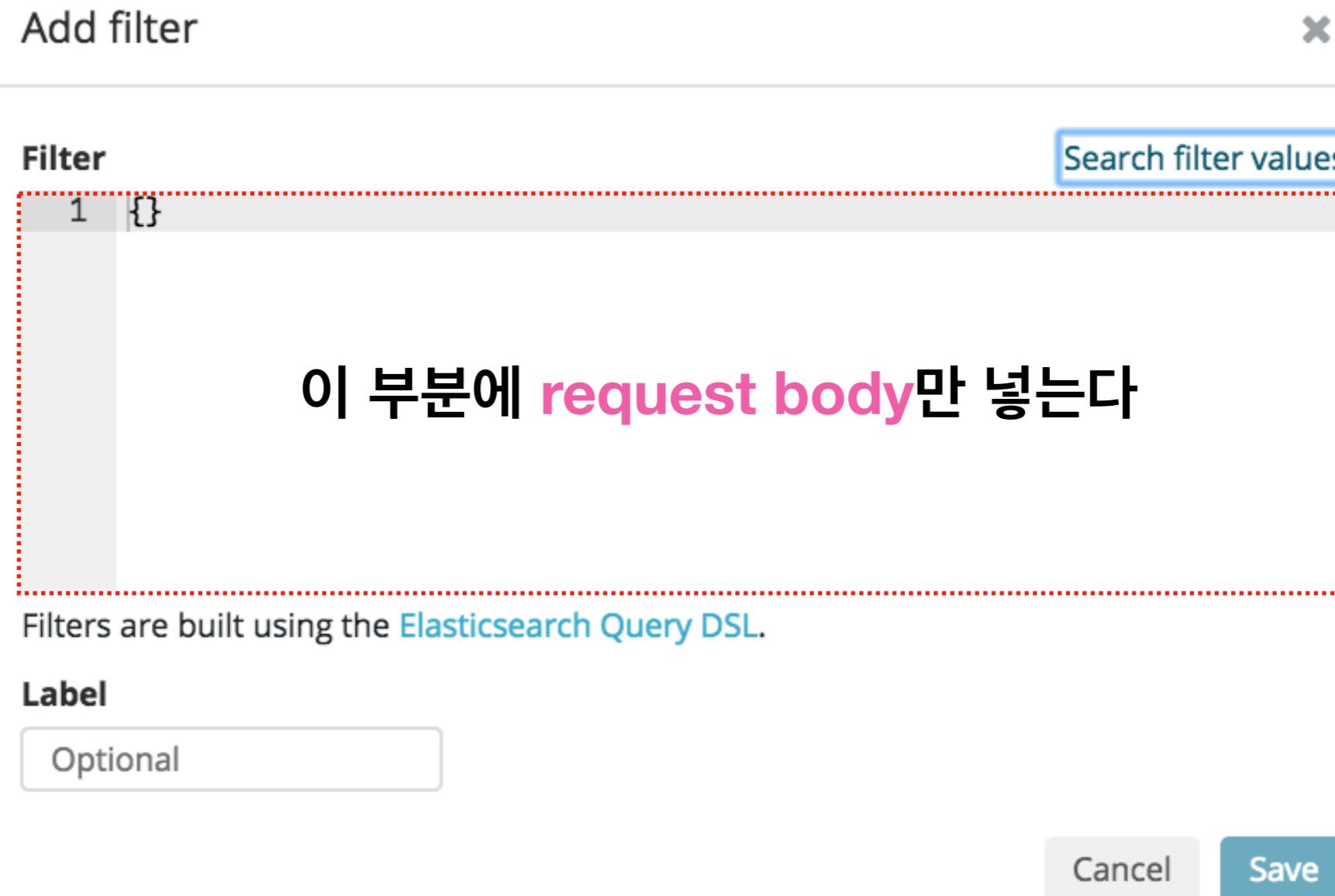
이처럼

- Visualize
- Visualization Spy
- Request
- Dev Tools

방식으로 접근하면
빠르게 원하는 query를 작성할 수 있다.

→ avg라고 되어 있는 부분을 sum으로 바꾸면 된다!

지금까지 배운 Query를 Filter에 어떻게 적용할까?



지금까지 배운 Query를 Filter에 어떻게 적용할까?

Add filter



Filter

```
1 {  
2   "query": {  
3     "term": {  
4       "상품분류": "셔츠"  
5     }  
6   }  
7 }
```

Search filter values

☞ 1. 입력

Filters are built using the [Elasticsearch Query DSL](#).

Label

셔츠

☞ 2. 입력

Cancel

Save

☞ 3. 선택

지금까지 배운 Query를 Filter에 어떻게 적용할까?

적용완료

데이터 확인

The screenshot shows the Kibana Discover interface with the following details:

- Top Bar:** 135 hits, Search... (e.g. status:200 AND extension:PHP), New, Save, Open, Share, Last 90 days.
- Sidebar (Discover):** Selected Fields: _source, Available Fields: shopping, Popular: 결제카드, 고객주소_시도, 배송메모, 배송소요시간, 상품분류, 수령시간, 시간대, 연령대, 요일, 주문시간, _id, _index, _score, _type, 결제카드-, 결제카드풀네이, 고객ip, 고객나이, 고객성별, 구매사이트, 나이를알려주겠-, 물건좌표, 배송소요시간-. A red box highlights the '셔츠' button under Selected Fields.
- Chart:** A histogram titled '주문시간 per day' showing the count of orders over time. The x-axis ranges from October 31st 2017 to January 29th 2018. The y-axis ranges from 0 to 15.
- Table (Results):** A list of search results for the query '셔츠'. Each result includes a timestamp, date, and a detailed log entry. A red box highlights the '_source' column, and a red hand icon points to the first entry.

Result Examples:

- 12월31일 17시59분: 상품분류: 셔츠 접수번호: 759 주문시간: 12월31일 17시59분 수령시간: 01월04일 18시24분 예약여부: 일반 배송메모: 부재증 고객ip: 198.69.83.206 고객성별: 여성 고객나이: 26 물건좌표: 37.82120780792847, 128.65164454811415 고객주소_시도: 서울특별시 구매사이트: 위메프 판매자평점: 1 상품가격: 28,000 상품개수: 7 결제카드: 우리 _id: AV-iDM6XRJy4v-Hns1aG _type: shopping _index: shopping _score: - 배송이얼마나걸리나: 96 요일--: 7 성별카드가자: 여성-우리 배송소요일수: 나흘 이상 배송소요일수_sort: 3 나이를알려주겠: 20 age 시간대: 8 연령대: 20대 소비행태: 과소비 배송소요시간: 96 요일_sort: 7 결제카드풀네이: 우리카드 요일: 일 성별-카드: 여성-우리 배송소요시간--: 96 결제카드--: 우리카드
- 12월30일 11시34분: 상품분류: 셔츠 접수번호: 5413 주문시간: 12월30일 11시34분 수령시간: 01월01일 21시40분 예약여부: 일반 배송메모: 관리실에 맡김 고객ip: 80.220.204.239 고객성별: 남성 고객나이: 21 물건좌표: 35.04875031549871, 128.07056814953543 고객주소_시도: 대전광역시 구매사이트: GS샵 판매자평점: 1 상품가격: 18,000 상품개수: 1 결제카드: 국민 _id: AV-iDkN5RJy4v-Hns2i0 _type: shopping _index: shopping _score: - 배송이얼마나걸리나: 58 요일--: 6 성별카드가자: 남성-국민 배송소요일수: 모레 배송소요일수_sort: 2 나이를알려주겠: 20 age 시간대: 2 연령대: 20대 소비행태: 저소비 배송소요시간: 58 요일_sort: 6 결제카드풀네이: 국민카드 요일: 토 성별-카드: 남성-국민 배송소요시간--: 58 결제카드--: 국민카드
- 12월30일 07시07분: 상품분류: 셔츠 접수번호: 3451 주문시간: 12월30일 07시07분 수령시간: 01월02일 14시37분 예약여부: 일반 배송메모: 부재증 고객ip: 167.190.90.160 고객성별: 여성 고객나이: 24 물건좌표: 37.346744741125214, 128.5509722697623 고객주소_시도: 충청남도 구매사이트: 쿠팡 판매자평점: 1 상품가격: 20,000 상품개수: 1 결제카드: 우리 _id: AV-iDaTSRJy4v-Hns2EK _type: shopping _index: shopping _score: - 배송이얼마나걸리나: 79 요일--: 5 성별카드가자: 여성-우리 배송소요일수: 나흘 이상 배송소요일수_sort: 3 나이를알려주겠: 20 age 시간대: 22 연령대: 20대 소비행태: 저소비 배송소요시간: 79 요일_sort: 5 결제카드풀네이: 우리카드 요일: 금 성별-카드: 여성-우리 배송소요시간--: 79 결제카드--: 우리카드
- 12월29일 19시11분: 상품분류: 셔츠 접수번호: 8534 주문시간: 12월29일 19시11분 수령시간: 01월02일 12시59분 예약여부: 일반 배송메모: 관리실에 맡김 고객ip: 25.182.190.33 고객성별: 여성 고객나이: 17 물건좌표: 36.22963636912366, 127.40979543882128 고객주소_시도: 전라남도 구매사이트: 쿠팡 판매자평점: 1 상품가격: 22,000 상품개수: 1 결제카드: 하나 _id: AV-iD0VvRJy4v-Hns3Tl _type: shopping _index: shopping _score: - 배송이얼마나걸리나: 89 요일--: 5 성별카드가자: 여성-하나 배송소요일수: 나흘 이상 배송소요일수_sort: 3 나이를알려주겠: 10 age 시간대: 10 연령대: 10대 소비행태: 저소비 배송소요시간: 89 요일_sort: 5 결제카드풀네이: 하나카드 요일: 금 성별-카드: 여성-하나 배송소요시간--: 89 결제카드--: 하나카드
- 12월29일 11시48분: 상품분류: 셔츠 접수번호: 13004 주문시간: 12월29일 11시48분 수령시간: 01월02일 07시44분 예약여부: 일반 배송메모: 주소 오류 고객ip: 92.198.64.230 고객성별: 여성 고객나이: 20 물건좌표: 37.297368471075586, 126.27321074940016 고객주소_시도: 제주특별자치도 구매사이트: GS샵 판매자평점: 4 상품가격: 17,000 상품개수: 1 결제카드: 우리 _id: AV-iE5_-RJy4v-Hns4Zc _type: shopping _index: shopping _score: - 배송이얼마나걸리나: 91 요일--: 5 성별카드가자: 여성-우리 배송소요일수: 나흘 이상 배송소요일수_sort: 3 나이를알려주겠: 20 age 시간대: 2 연령대: 20대 소비행태: 저소비 배송소요시간: 91 요일_sort: 5 결제카드풀네이: 우리카드 요일: 금 성별-카드: 여성-우리 배송소요시간--: 91 결제카드--: 우리카드

설치 - AWS EC2 Instance 생성

우선 AWS EC2 Instance를 생성해보자  

Elastic Stack을 설치해보자  

보너스 문제

다음과 같은 조건을 만족하는 Visualization을 만들고 공유하자 

Index : shopping

Date Range : 2017.01.01 ~ 2017.12.31

Visualization Type : Area Chart

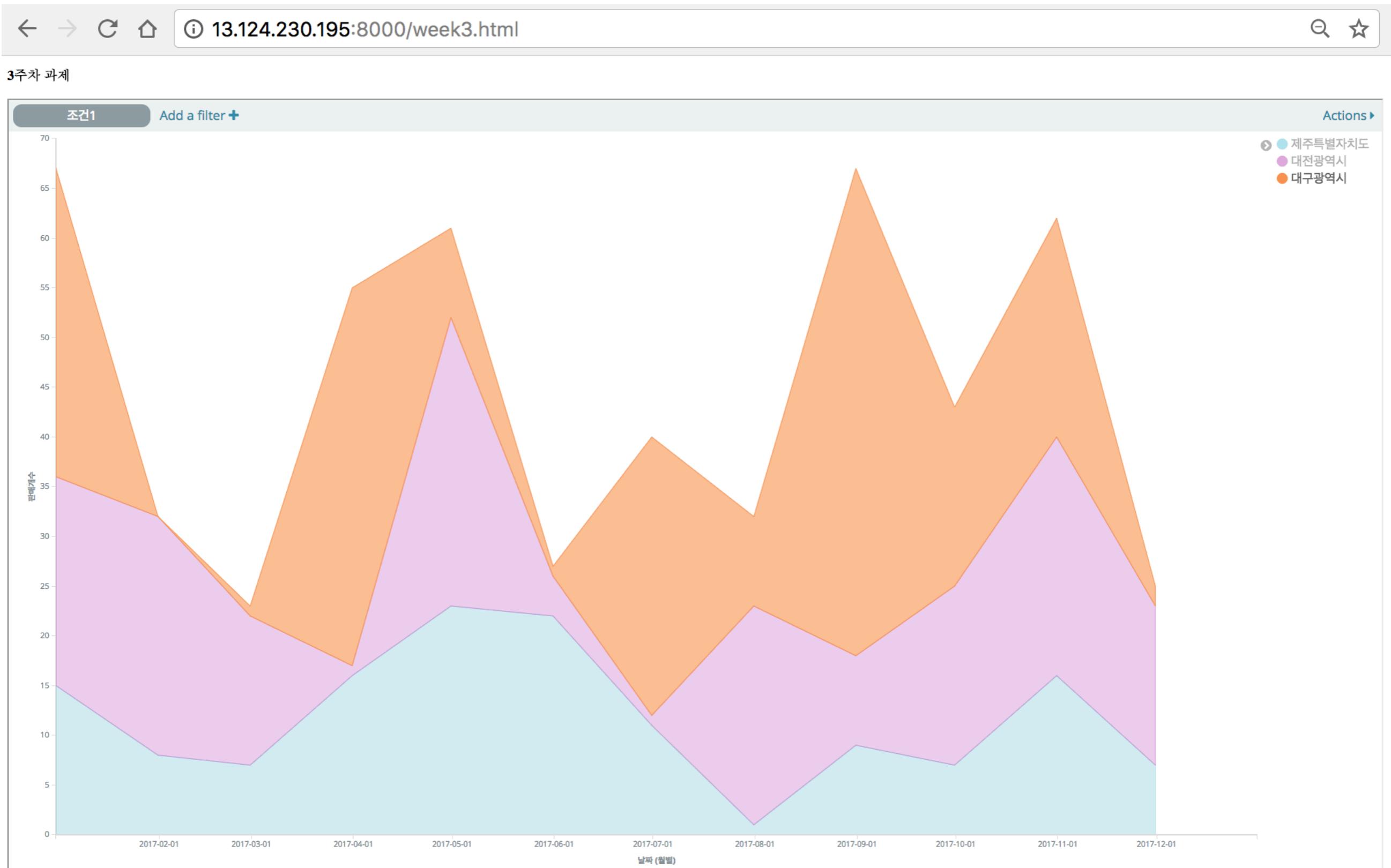
Visualization 조건

- “상품가격”의 평균이 큰 “고객주소_시도” 3개를 선정해서
- “주문시간”을 기준으로 월별로
- “상품개수”의 합계를 표시하자

Filter 조건

- 반드시 만족 : 고객나이 > 25
- 다음 중 최소 1개 만족
 - 서울주소_시도 : “경”으로 시작하고 “도”로 끝나는 3글자
 - 결제카드 : 우리
 - 주문시간 기준으로 시간대가 18시보다 이후

보너스 문제



질문 및 Feedback은

gshock94@gmail.com로 주세요