Elasticsearch Aggregation 기법을 활용한 django 웹서비스에 대쉬보드 구현

- Aggregation 쿼리문을 장고에서 실행, 웹서비스에서 시각화 구현 (아래 순서에 따라 작동방식 이해)

1. Kibana에서 aggregation 쿼리 작성.
2. Django의 views.py에서 aggregation 쿼리문 작성.
3. Django 템플릿에 데이터 렌더링.

- Aggregation관련 다양한 기능들
  (참조: https://esbook.kimjmin.net/08-aggregations)

1. Metrics Aggregations
2. Bucket Aggregations
3. Sub-Aggregatoins
4. Pipeline Aggregations

- Kibana에서 aggregation쿼리 예시 (Dev-ops 메뉴에서 실행)

```
GET /logs-index/_search
{
  "size": 0,
  "aggs": {
    "top_source_ips": {
      "terms": {
        "field": "src_ip.keyword",
        "size": 10
      }
    }
  }
}
```

- `logs-index` 인덱스에서 **source IP별로 상위 10개**를 집계하는 **terms aggregation**을 수행하는 쿼리

- django 에서 elasticsearch와의 연동을 진행한 후, views.py에서 이와 같은 쿼리문을 정의할 수 있다
(다음 슬라이드 참조)

- Django의 views.py에서 aggregation 쿼리문 작성.

```python
from django.shortcuts import render
from elasticsearch import Elasticsearch

# Elasticsearch 클라이언트 설정
es = Elasticsearch(['http://localhost:9200'])

def dashboard_view(request):
    # Aggregation 쿼리 작성
    query = {
        "size": 0,
        "aggs": {
            "top_source_ips": {
                "terms": {
                    "field": "src_ip.keyword",
                    "size": 10
                }
            }
        }
    }

    # Elasticsearch 쿼리 전송
    result = es.search(index="logs-index", body=query)

    # Aggregation 결과 추출
    top_ips = result['aggregations']['top_source_ips']['buckets']

    # 결과를 Django 템플릿으로 전달
    context = {
        'top_ips': top_ips,
    }

    return render(request, 'dashboard.html', context)
```

- Django의 **views.py**에서 Elasticsearch 클라이언트를 사용하여 **aggregation 쿼리**를 Elasticsearch에 전송

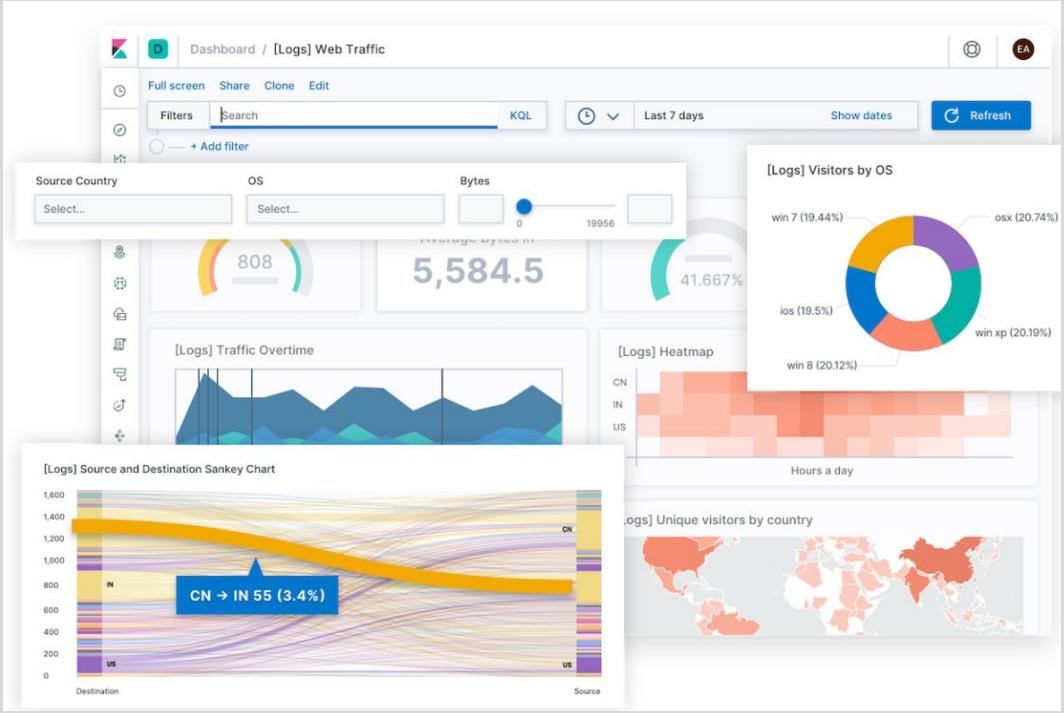
- 쿼리의 결과를 받아서 Django 템플릿에 전달

– Django 템플릿에 데이터 렌더링

```
<!DOCTYPE html>
<html>
<head>
    <title>Aggregation Dashboard</title>
</head>
<body>
    <h1>Top Source IPs</h1>
    <ul>
        {% for ip in top_ips %}
            <li>{{ ip.key }}: {{ ip.doc_count }} hits</li>
        {% endfor %}
    </ul>
</body>
</html>
```

- Html 페이지에서 views.py에서 정의한 context를 지정하여 자료 출력
- source IP와 해당 IP에서 발생한 이벤트 수를 리스트 형식으로 표시

- Aggregation관련 다양한 기능들



- Aggregation 기능을 이용하여 단순 검색이 아닌 여러가지 연산을 통해 Elasticsearch 사용경험 증대
- Kibana에서 바 차트, 파이 차트 등으로 데이터를 시각화 업무를 수행 시 Aggregation 개념이 적용
  - Metrics Aggregations: 최소, 최대, 합계, 평균을 집계
  - Bucket Aggregations: 주어진 조건으로 분류된 버킷들을 만들고, 각 버킷에 소속되는 도큐먼트들을 모아 그룹으로 구분
  - Sub-aggregations: aggregation 내부에 다시 aggregation 선언
  - Pipeline Aggregations: metrics aggregation의 결과를 추가 연산하여, 이동평균 및 미분값 등등을 도출하는데 사용

- Aggregation 쿼리 테스트를 위한 데이터 사전 입력

테스트를 위한 my_stations 인덱스에 데이터 입력

```
PUT my_stations/_bulk
{"index": {"_id": "1"}}
{"date": "2019-06-01", "line": "1호선", "station": "종각", "passangers": 2314}
{"index": {"_id": "2"}}
{"date": "2019-06-01", "line": "2호선", "station": "강남", "passangers": 5412}
{"index": {"_id": "3"}}
{"date": "2019-07-10", "line": "2호선", "station": "강남", "passangers": 6221}
{"index": {"_id": "4"}}
{"date": "2019-07-15", "line": "2호선", "station": "강남", "passangers": 6478}
{"index": {"_id": "5"}}
{"date": "2019-08-07", "line": "2호선", "station": "강남", "passangers": 5821}
{"index": {"_id": "6"}}
{"date": "2019-08-18", "line": "2호선", "station": "강남", "passangers": 5724}
{"index": {"_id": "7"}}
{"date": "2019-09-02", "line": "2호선", "station": "신촌", "passangers": 3912}
{"index": {"_id": "8"}}
{"date": "2019-09-11", "line": "3호선", "station": "양재", "passangers": 4121}
{"index": {"_id": "9"}}
{"date": "2019-09-20", "line": "3호선", "station": "홍제", "passangers": 1021}
{"index": {"_id": "10"}}
{"date": "2019-10-01", "line": "3호선", "station": "불광", "passangers": 971}
```

- Metrics Aggregations: sum

my_stations 인덱스의 passangers 필드 합 (sum) 을 가져오는 aggs

```
GET my_stations/_search
{
  "size": 0,
  "aggs": {
    "all_passangers": {
      "sum": {
        "field": "passangers"
      }
    }
  }
}
```

my_stations 인덱스의 passangers 필드 합 (sum) 을 가져오는 aggs 결과

```
{
  "took" : 1,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 10,
      "relation" : "eq"
    },
    "max_score" : null,
    "hits" : [ ]
  },
  "aggregations" : {
    "all_passangers" : {
      "value" : 41995.0
    }
  }
}
```

쿼리에 대한 상태

1)시간제한 안에 응답 완료
timed_out : false

2)shard 갯 수 : 1

3)쿼리 성공
successful: 1

쿼리에 대한 결과

1)총 10개의 자료를 검색
Hits -> value : 10

2)총 passenger
All_passengers ->
value: 41995.0

- Metrics Aggregations: sum (강남 only)

stations 값이 "강남" 인 도큐먼트들의 passangers 필드 합 (sum) 을 가져오는 aggs

```
GET my_stations/_search
{
  "query": {
    "match": {
      "station": "강남"
    }
  },
  "size": 0,
  "aggs": {
    "gangnam_passangers": {
      "sum": {
        "field": "passangers"
      }
    }
  }
}
```

stations 값이 "강남" 인 도큐먼트들의 passangers 필드 합 (sum) 을 가져오는 aggs 결과


```
{
  "took" : 0,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 5,
      "relation" : "eq"
    },
    "max_score" : null,
    "hits" : [ ]
  },
  "aggregations" : {
    "gangnam_passangers" : {
      "value" : 29656.0
    }
  }
}
```


쿼리에 대한 결과

1)총 5개의 자료가 매칭
Hits -> value : 54

2)총 passenger가 줄어든 것을 확인 가능
All_passengers -> value: 41995.0 -> 29656.0

- Metrics Aggregations: stats (min, max, avg, sum)

stats 로 passangers 필드의 min, max, sum, avg

```
GET my_stations/_search
{
  "size": 0,
  "aggs": {
    "passangers_stats": {
      "stats": {
        "field": "passangers"
      }
    }
  }
}
```

stats 로 passangers 필드의 min, max, su

```
{
  "took" : 2,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 10,
      "relation" : "eq"
    },
    "max_score" : null,
    "hits" : [ ]
  },
  "aggregations" : {
    "passangers_stats" : {
      "count" : 10,
      "min" : 971.0,
      "max" : 6478.0,
      "avg" : 4199.5,
      "sum" : 41995.0
    }
  }
}
```

쿼리에 대한 결과

Min, max, avg, sum 의 모든 자료들이 검색

- Metrics Aggregations: cardinality

line 필드의 값이 몇 종류인지를 가져오는 aggs

```
GET my_stations/_search
{
  "size": 0,
  "aggs": {
    "uniq_lines ": {
      "cardinality": {
        "field": "line.keyword"
      }
    }
  }
}
```

```
  },
  "aggregations" : {
    "uniq_lines " : {
      "value" : 3
    }
  }
}
```

쿼리에 대한 결과

`"uniq_lines " : { "value" : 3 }` 처럼 실제로 line 필드에는 "1호선", "2호선", "3호선" 총 3 종류의 값들이 있음

- Metrics Aggregations: percentiles

passangers 필드의 백분위를 가져오는 aggs

```
GET my_stations/_search
{
  "size": 0,
  "aggs": {
    "pass_percentiles": {
      "percentiles": {
        "field": "passangers"
      }
    }
  }
}
```

```
"aggregations" : {
  "pass_percentiles" : {
    "values" : {
      "1.0" : 971.0000000000001,
      "5.0" : 971.0,
      "25.0" : 2314.0,
      "50.0" : 4766.5,
      "75.0" : 5821.0,
      "95.0" : 6478.0,
      "99.0" : 6478.0
    }
  }
}
```

쿼리에 대한 결과

디폴트로 **1%**, **5%**, **25%**, **50%**, **75%**, **95%**, **99%** 구간에 위치 해 있는 값들을 표시

*아래와 같이 구간 지정 가능

```
"percentiles": {
  "field": "passangers",
  "percents": [ 20, 60, 80 ]
}
```

```
"pass_percentiles" : {
  "values" : {
    "20.0" : 1667.5,
    "60.0" : 5568.0,
    "80.0" : 6021.0
  }
}
```

- Metrics Aggregations: percentile_ranks

```
GET my_stations/_search
{
  "size": 0,
  "aggs": {
    "pass_percentile_ranks": {
      "percentile_ranks": {
        "field": "passangers",
        "values": [ 1000, 3000, 6000 ]
      }
    }
  }
}
```

```
"aggregations" : {
  "pass_percentile_ranks" : {
    "values" : {
      "1000.0" : 10.059568131049886,
      "3000.0" : 29.218263576617087,
      "6000.0" : 79.1549295774648
    }
  }
```

쿼리에 대한 결과

반대로 값을 입력해서 그 값이 위치 해 있는 백분위를 확인 가능

*전체 수험생의 성적 중에서 특정 점수가 상위 몇% 에 있는지 등을 파악하기 위한 과업을 수행하기에 적절

- Bucket Aggregations: range

*숫자 필드 값으로 범위를 지정하고 각 범위에 해당하는 버킷 생성

range aggs 를 이용해서 passangers 필드의 값을 버킷

```
GET my_stations/_search
{
  "size": 0,
  "aggs": {
    "passangers_range": {
      "range": {
        "field": "passangers",
        "ranges": [
          {
            "to": 1000
          },
          {
            "from": 1000,
            "to": 4000
          },
          {
            "from": 4000
          }
        ]
      }
    }
  }
}
```

```
"aggregations" : {
  "passangers_range" : {
    "buckets" : [
      {
        "key" : "*-1000.0",
        "to" : 1000.0,
        "doc_count" : 1
      },
      {
        "key" : "1000.0-4000.0",
        "from" : 1000.0,
        "to" : 4000.0,
        "doc_count" : 3
      },
      {
        "key" : "4000.0-*",
        "from" : 4000.0,
        "doc_count" : 6
      }
    ]
  }
}
```

쿼리에 대한 결과

각 버킷을 구분하는 **key** 값은 `"from-to"` 형태로 생성

`"*-1000.0"`,
`"1000.0-4000.0"`,
`"4000.0-*"`
각 버킷에 속한 도큐먼트의 개수가 **1**, **3**, **6**개 인 것을 확인

- Bucket Aggregations: histogram

*histogram 은 from, to 대신 interval 옵션을 이용해서 주어진 간격 크기대로 버킷을 구분

histogram aggs 를 이용해서 passangers 필드의

```
GET my_stations/_search
{
  "size": 0,
  "aggs": {
    "passangers_his": {
      "histogram": {
        "field": "passangers",
        "interval": 2000
      }
    }
  }
}
```

```
"aggregations" : {
  "passangers_his" : {
    "buckets" : [
      {
        "key" : 0.0,
        "doc_count" : 2
      },
      {
        "key" : 2000.0,
        "doc_count" : 2
      },
      {
        "key" : 4000.0,
        "doc_count" : 4
      },
      {
        "key" : 6000.0,
        "doc_count" : 2
      }
    ]
  }
}
```

쿼리에 대한 결과

0, 2000, 4000, 6000 인
버킷들이 생성되었고 각
버킷들에 포함되는 도큐먼트
개수가 표시된 것을 확인

- Bucket Aggregations: date_range, date_histogram

*날짜 필드를 이용해서 범위별로 버킷의 생성

```
GET my_stations/_search
{
  "size": 0,
  "aggs": {
    "date_his": {
      "date_histogram": {
        "field": "date",
        "interval": "month"
      }
    }
  }
}
```


```
"aggregations" : {
  "date_his" : {
    "buckets" : [
      {
        "key_as_string" : "2019-06-01T00:00:00.000Z",
        "key" : 1559347200000,
        "doc_count" : 2
      },
      {
        "key_as_string" : "2019-07-01T00:00:00.000Z",
        "key" : 1561939200000,
        "doc_count" : 2
      },
      {
        "key_as_string" : "2019-08-01T00:00:00.000Z",
        "key" : 1564617600000,
        "doc_count" : 2
      },
      {
        "key_as_string" : "2019-09-01T00:00:00.000Z",
        "key" : 1567296000000,
        "doc_count" : 3
      },
      {
        "key_as_string" : "2019-10-01T00:00:00.000Z",
        "key" : 1569888000000,
        "doc_count" : 1
      }
```


date_range ranges 옵션:
`{"from": "2019-06-01", "to": "2016-07-01"}`

date_histogram interval 옵션:
day, month, week와 같은 값들을 활용, 날짜 간격 지정

****7.2 버전부터** interval 옵션이 사용 종료 권장됨에 따라 **fixed_interval** 과 **calendar_interval** 으로 분리. year, quarter, month, week, 같은 달력 기준의 값은 `"calendar_interval" : "month"` 로 입력,
30일 처럼 정확히 구분되는 날짜들은 `"fixed_interval" : "30d"` 로 지정

쿼리에 대한 결과

date_histogram 을 이용해서 date 값을 1개월 간격의 버킷으로 구분한 결과

- Bucket Aggregations: terms

*__keyword__ 필드의 문자열 별로 버킷을 나누어 집계가 가능
*다음은 my_stations 인덱스에서 __station.keyword__ 필드를 기준으로 버킷들을 만드는 예제

terms 를 이용해서 station 값을 별로 버킷 생성

```
GET my_stations/_search
{
  "size": 0,
  "aggs": {
    "stations": {
      "terms": {
        "field": "station.keyword"
      }
    }
  }
}
```

```
"aggregations" : {
  "stations" : {
    "doc_count_error_upper_bound" : 0,
    "sum_other_doc_count" : 0,
    "buckets" : [
      {
        "key" : "강남",
        "doc_count" : 5
      },
      {
        "key" : "불광",
        "doc_count" : 1
      },
      {
        "key" : "신촌",
        "doc_count" : 1
      },
      {
        "key" : "양재",
        "doc_count" : 1
      },
      {
        "key" : "종각",
        "doc_count" : 1
      },
      {
        "key" : "홍제",
        "doc_count" : 1
      }
```

쿼리에 대한 결과

총 10개의 데이터 중에서
강남역에 관련한 데이터 5개
및 나머지 역들에 대한
데이터를 각 1개씩 집계한
결과를 표현

- Bucket Aggregations: sub-aggregations (1)

*Bucket Aggregation 으로 만든 버킷들 내부에 다시 "aggs" : { } 를 선언, 또 다른 버킷을 만들거나 Metrics Aggregation 을 생성

terms aggs 아래에 하위 terms aggs 사용

```
GET my_stations/_search
{
  "size": 0,
  "aggs": {
    "lines": {
      "terms": {
        "field": "line.keyword"
      },
      "aggs": {
        "stations_per_lines": {
          "terms": {
            "field": "station.keyword"
          }
        }
      }
    }
  }
}
```

```
"aggregations" : {
  "stations" : {
    "doc_count_error_upper_bound" : 0,
    "sum_other_doc_count" : 0,
    "buckets" : [
      {
        "key" : "강남",
        "doc_count" : 5,
        "avg_psg_per_st" : {
          "value" : 5931.2
        }
      },
      {
        "key" : "불광",
        "doc_count" : 1,
        "avg_psg_per_st" : {
          "value" : 971.0
        }
      },
      {
        "key" : "신촌",
        "doc_count" : 1,
        "avg_psg_per_st" : {
          "value" : 3912.0
        }
      },
      {
        "key" : "양재",
        "doc_count" : 1,
        "avg_psg_per_st" : {
          "value" : 4121.0
        }
      },
      {
        "key" : "종각",
        "doc_count" : 1,
        "avg_psg_per_st" : {
          "value" : 2314.0
        }
      },
      {
        "key" : "홍제",
        "doc_count" : 1,
        "avg_psg_per_st" : {
          "value" : 1021.0
```

쿼리에 대한 결과

각 역들(station.keyword)에 관한 평균 이용객 수 를 표현

- Bucket Aggregations: sub-aggregations (2)

*Bucket Aggregation 으로 만든 버킷들 내부에 다시 "aggs" : { } 를 선언, 또 다른 버킷을 만들거나 Metrics Aggregation 을 생성

terms aggs 아래에 하위 terms aggs 사용

```
GET my_stations/_search
{
  "size": 0,
  "aggs": {
    "lines": {
      "terms": {
        "field": "line.keyword"
      },
      "aggs": {
        "stations_per_lines": {
          "terms": {
            "field": "station.keyword"
          }
        }
      }
    }
  }
}
```

```
      "key" : "2호선",
      "doc_count" : 6,
      "stations_per_lines" : {
        "doc_count_error_upper_bound" : 0,
        "sum_other_doc_count" : 0,
        "buckets" : [
          {
            "key" : "강남",
            "doc_count" : 5
          },
          {
            "key" : "신촌",
            "doc_count" : 1
          }
        ]
      }
    },
    {
      "key" : "3호선",
      "doc_count" : 3,
      "stations_per_lines" : {
        "doc_count_error_upper_bound" : 0,
        "sum_other_doc_count" : 0,
        "buckets" : [
          {
            "key" : "불광",
            "doc_count" : 1
          },
          {
            "key" : "양재",
            "doc_count" : 1
          },
          {
            "key" : "홍제",
            "doc_count" : 1
          }
        ]
      }
    },
    {
      "key" : "1호선",
      "doc_count" : 1,
      "stations_per_lines" : {
        "doc_count_error_upper_bound" : 0,
        "sum_other_doc_count" : 0,
        "buckets" : [
          {
            "key" : "종각",
            "doc_count" : 1
          }
```

쿼리에 대한 결과

각 호선들에 상응하는
역명들에 대한 정보를 표기

- Bucket Aggregations: Pipeline Aggregations (1)

*pipeline을 통해 다른 버킷의 결과들을 다시 연산, min_bucket, max_bucket, avg_bucket, sum_bucket, stats_bucket, 이동 평균을 구하는 moving_avg, 미분값을 구하는 derivative, 값의 누적 합을 구하는 cumulative_sum 등등의 연산이 가능

passangers 의 값을 입력으로 받는 cumulative_sum aggs 실행

```
GET my_stations/_search
{
  "size": 0,
  "aggs": {
    "months": {
      "date_histogram": {
        "field": "date",
        "interval": "month"
      },
      "aggs": {
        "sum_psg": {
          "sum": {
            "field": "passangers"
          }
        },
        "accum_sum_psg": {
          "cumulative_sum": {
            "buckets_path": "sum_psg"
          }
        }
      }
    }
  }
}
```

```
"aggregations" : {
  "months" : {
    "buckets" : [
      {
        "key_as_string" : "2019-06-01T00:00:00.000Z",
        "key" : 1559347200000,
        "doc_count" : 2,
        "sum_psg" : {
          "value" : 7726.0
        },
        "accum_sum_psg" : {
          "value" : 7726.0
        }
      },
      {
        "key_as_string" : "2019-07-01T00:00:00.000Z",
        "key" : 1561939200000,
        "doc_count" : 2,
        "sum_psg" : {
          "value" : 12699.0
        },
        "accum_sum_psg" : {
          "value" : 20425.0
        }
      },
```

```
.
"value" : 31970.0
.
.
"value" : 41024.0
.
.
"value" : 41995.0
```

쿼리에 대한 결과

accum_sum_psg 결과에 **sum_psg** 값이 다음과 같이 계속 누적되어 더해지고 있는 것을 확인

`"accum_sum_psg" : { "value" : 7726.0}` = 7726.0

`"accum_sum_psg" : { "value" : 20425.0}` = 7726.0 + 12699.0

`"accum_sum_psg" : { "value" : 31970.0}` = 7726.0 + 12699.0 + 11545.0

...

- Bucket Aggregations: Pipeline Aggregations (2)

*서로 다른 버킷에 있는 값들도 bucket_path에 > 기호를 이용해서 "부모>자녀" 형태로 지정이 가능
*sum_bucket 을 이용해서 **mon>sum_psg** 버킷에 있는 passangers 필드값의 합을 구하는 예제

다른 부모의 자녀 버킷에 있는 필드를 입력으로 받는 pipeline aggs

```
GET my_stations/_search
{
  "size": 0,
  "aggs": {
    "mon": {
      "date_histogram": {
        "field": "date",
        "interval": "month"
      },
      "aggs": {
        "sum_psg": {
          "sum": {
            "field": "passangers"
          }
        }
      }
    },
    "bucket_sum_psg": {
      "sum_bucket": {
        "buckets_path": "mon>sum_psg"
      }
    }
  }
}
```


```
"aggregations" : {
  "mon" : {
    "buckets" : [
      {
        "key_as_string" : "2019-06-01T00:00:00.000Z",
        "key" : 1559347200000,
        "doc_count" : 2,
        "sum_psg" : {
          "value" : 7726.0
        }
      },
      {
        "key_as_string" : "2019-07-01T00:00:00.000Z",
        "key" : 1561939200000,
        "doc_count" : 2,
        "sum_psg" : {
          "value" : 12699.0
        }
      },
      {
        "key_as_string" : "2019-08-01T00:00:00.000Z",
        "key" : 1564617600000,
        "doc_count" : 2,
        "sum_psg" : {
          "value" : 11545.0
        }
      },
      {
        "key_as_string" : "2019-09-01T00:00:00.000Z",
        "key" : 1567296000000,
        "doc_count" : 3,
        "sum_psg" : {
          "value" : 9054.0
        }
      },
      {
        "key_as_string" : "2019-10-01T00:00:00.000Z",
        "key" : 1569888000000,
        "doc_count" : 1,
        "sum_psg" : {
          "value" : 971.0
        }
      }
    ]
  },
  "bucket_sum_psg" : {
    "value" : 41995.0
```


쿼리에 대한 결과

2019-06-01: 7,726명.

2019-07-01: 12,699명,

2019-08-01: 11,545명 …

bucket_sum_psg는 모든 월에 대한 승객 수 합계를 계산한 값으로, 총 41,995명