1. **Please write a query matching products that didn't sell very well, being products where the "sold" field has a value of less than 10 (sold < 10)**

```
GET product/_search
{
 "query": {
   "range": {
     "sold": {
       "lte": 10
     }
    }
```

2. **Write a query that matches products that sold okay, meaning less than 30 and greater than or equal to 10 (sold < 30 && sold >= 10).**

```
GET product/_search
{
  "query": {
    "range": {
      "sold": {
        "gte": 10,
        "lte": 30
      }
   }
  }
}
```

3. **Write a query that matches documents containing the term "Meat" within the "tags" field.**

```
GET product/_search
{
 "query":{
   "match":{
     "tags":"meat"
   }
 }
}
```

4. **Write a query matching documents containing one of the terms "Tomato" and "Paste" within the "name" field.**

```
GET /product/_search{
 "query":{
 "match": {
   "name":{
     "query": "Tomato Paste",
      "lenient": "true",
      "operator": "and"
    }
  }
}

}
```

5. **Write a query that matches products with a "name" field including “pasta”, “paste”, or similar. The query should be dynamic and not use the "terms" query clause.**

```
GET /product/_search
{
  "query":{
    "match_phrase_prefix": {
      "name":{
        "query":"pasta",
        "max_expansions":1
      }
    }
  }
}
```

6. **Check and mention how many documents have non-empty tags field in product index.**

```
GET /product/_search
{
  "query": {
    "bool": {
     "must": {
       "exists":{
         "field":"tags"
       }
```

```
      }
     }
    }
   }
```

7. **Find which documents of shakespeare index contain the following terms:**
   men, holy, fight, prince, blessed, conquest, war, knife
   The number of above terms matched in a document must be at least half of number of words in the document. Refer to terms_set query taught in Week II.

```
GET /shakespeare/_search
{
  "query":{
    "terms_set":{
      "text_entry":{
        "terms":
          ["men", "holy", "fight", "prince", "blessed", "conquest", "war", "knife"],
          "minimum_should_match_script":{
            "source":"params.num_terms/4"
          }

      }
    }
  }
}
```

8. Look for document having the phrase **some night-tripping fairy.**

```
GET /shakespeare/_search
{
"query":{
    "match_phrase": {
          "text_entry": "some night-tripping fairy"
          }
}
}
```

9. **Find documents containing the words that are fuzzily similar to “sape of likehood”. Try using a fuzziness of 2. Also, make sure that all three words are available in the document that is returned.**

```
GET /shakespeare/_search
{
   "query": {
      "match": {
         "text_entry": {
            "query": "sape of likehood",
            "fuzziness": 2,
            "operator":"and"
         }
      }
   }
}
```

10. Find all documents that had **Henry IV** in fields **play_name** and **speaker**.

```
GET /shakespeare/_search
{
  "query": {
    "multi_match": {
      "query": "Henry IV",
      "fields": ["play_name","speaker"]
      , "operator": "and"
    }
  }
}
```

11. **Practice using cut-off frequency to handle domain specific stop-words in match query and common-terms query. You can consider shakespeare index’s text-entry field for this purpose.**

```
GET /shakespeare/_search
{
"query": {
```

```
          "match": {
           "text_entry": {
           "query": "my lady",
           "cutoff_frequency": 0.001

          }
    }
  }
}
```

12.Find documents from product index that match the following criteria. Fit in all these criteria in a single query.

- That is currently active, has in_stock of at least 10 and has either wine or meat or both in tags.
- Filter the documents having price of at least 150.
- May or may not have 300 or more items sold.

```
GET product/_search
{
  "query": {
     "bool": {
       "must": [{
         "term": {
           "is_active": {
              "value": true
                }
              }
          }, {
              "range": {
                 "in_stock": {
                    "gte": 10
                 }
              }
          }, {
              "query_string": {
                 "default_field": "tags",
                 "query": "(wine) or (meat)"
              }
          }],
```

```
        "filter": {
          "range": {
            "price": {
              "gte": 150
            }
          }
        },
        "should": [{
          "range": {
            "sold": {
              "gte": 300
            }
          }
        }]
      }
    }
}
```

13.Refer to **Nested** query in [Week III Notes] to do the following question:

1. **Create an index college having following fields:**
   - batch (integer type): example values, 2017, 2018
   - students (nested type, i.e. array of inner objects): each inner object can have three properties **id**, **name** and **age**.

```
PUT college2
{
  "mappings": {
    "_doc": {
      "dynamic": "false",
       "properties": {
          "batch": {
            "type": "integer"
            },
            "students": {
              "type": "nested",
              "properties": {
                "id": {
                  "type": "integer"
                },
```

```
                    "name": {
                        "type": "text"
                    },
                    "age": {
                        "type": "integer"
                    }
                }
            }
        }
    }
}
```

2. **Insert a document with certain id (example, 1), your batch (example, 2017), and an array of 3 students in index college.**

PUT college2/_doc/1

```
{
  "batch": 2019,
  "students": [
    {
      "id": 521,
      "name": "Rachana Banjade",
      "age": 21
    },
    {
      "id": 544,
      "name": "Sweekriti Gautam",
      "age": 21
    },
    {
      "id": 547,
      "name": "Shreeya Pandey",
      "age": 22
    }
  ]
}
```

3. **Use nested query to find parent documents having any student with age greater than 10. Along with parent documents, the inner hits should also be shown.**

```
GET college2/_search
{
  "query": {
    "nested": {
      "path": "students",
      "ignore_unmapped": true,
      "score_mode": "sum",
      "inner_hits": {},
      "query": {
        "bool": {
          "should": [
            {
              "range": {
                "students.age": {
                  "gte": 10

                }
              }
            }
          ]
        }
      }
    }
  }
}
```

14. Create a **filtered alias** of documents of **products** index having **is_active: true**.

```
POST /_aliases
{
  "actions" : [
    {
     "add" : {
       "indices":["product"],
        "alias" : "active_products",
```

```
        "filter" : {
          "term" : {
            "is_active": "true"
            }
            }
          }
      }
   ]
}
```