EECS E6893 Big Data Analytic HW4

Chong Hu ch3467

November 15, 2019

Problem 1. Bar Chart

1. Answer these questions in simple words.

   1.1 SVG Coordinate Space basically works in the way of Mathematical / Graph Coordinate Space, except for:

   SVG Coordinate space has x=0 and y=0 coordinates fall on the top left, while Mathematical / Graph Coordinate Space has x=0 and y=0 fall on the bottom left.

   SVG Coordinate space has the Y coordinate growing from top to bottom.

   1.2 `.enter` identifies any DOM elements that need to be added when the joined array is longer than the selection. `.exit` returns an exit selection which consists of the elements that need to be removed from the DOM.

   1.3 `transform` in SVG is used to transform a single SVG shape element or group of SVG elements. SVG transform supports `translate, scale, rotate` and `skew`. `translation` moves all the points of an element in the same direction and by the same amount. It preserves parallelism, angles and distances.

   1.4 Anonymous function in d3.js is used to easily manipulate data and data could be applied in DOM. In this case, the answer is

```
[5,6,7,8,9]
```

2. bar-chart

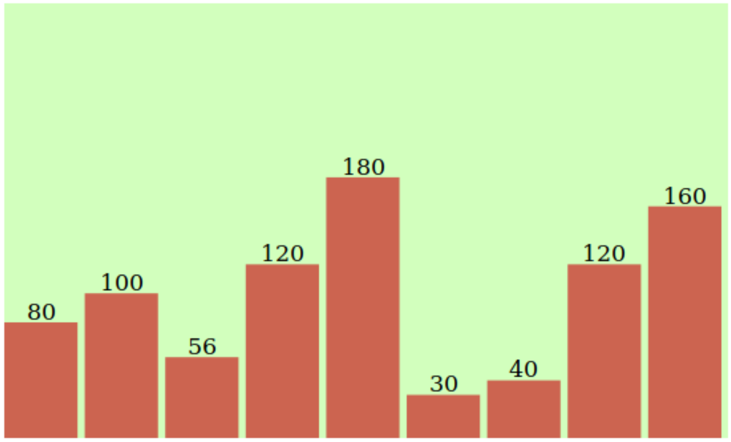


Figure 1: Bar Chart plot using SVG.

Code: Here I write the javascript in a single file (`part1.js`), separated with the structure file `part1.html`

`part1.html`

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Homework 4 Question 1</title>
</head>

<body>
<style>body {
}

.bar-chart {
    background-color: #D2FFBD;
}</style>
<svg class="bar-chart"></svg>

<script src="https://d3js.org/d3.v5.min.js"></script>
<script src="part1.js"></script>

</body>
</html>
```

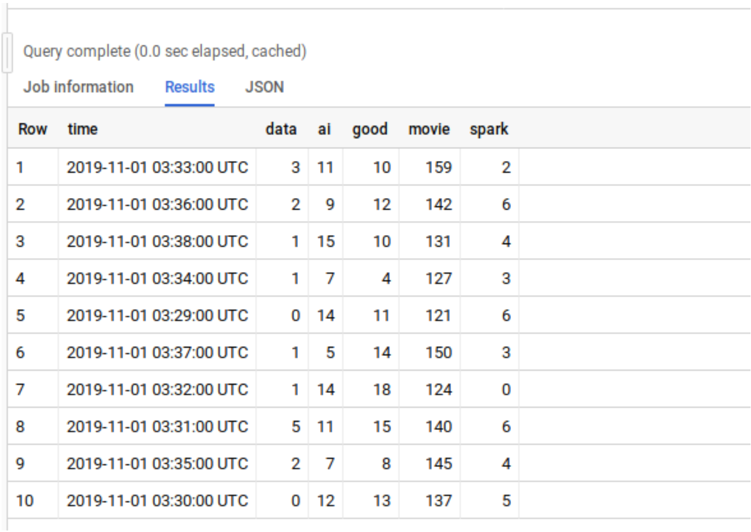
part1.js

```javascript
var data = [80, 100, 56, 120, 180, 30, 40, 120, 160];
var svgWidth = 500, svgHeight = 300;
// The required padding between bars is 5px.
// The label must locate 2px above the middle of each bar.

var svg = d3.select('svg')
    .attr("width", svgWidth)
    .attr("height", svgHeight);

const barPadding = 5;
const barWidth = svgWidth / data.length - barPadding;

function translateBarHelper(d, i) {
    return "translate(" + (barWidth + barPadding) * i + ","
        + (svgHeight - d) + ")";
}

function translateTextHelper(d, i) {
    return "translate(" + ((barWidth + barPadding) * i + barWidth / 2) + ","
        + (svgHeight - d - 2) + ")";
}

var barChart = svg.selectAll("rect")
    .data(data)
    .enter();

barChart.append("rect")
    .attr("class", "bar")
    .attr("height", function (d) {
        return d;
    })
    .attr("width", barWidth)
    .attr("transform", translateBarHelper)
    .attr("fill", "#CC6450");

barChart.append("text")
    .text(function (d) {
        return d;
    }).attr("transform", translateTextHelper)
    .style("text-anchor", "middle");
```

Problem 2. Dashboard

1. Data processing.

   Result preview:



Query complete (0.0 sec elapsed, cached)

Row	time	data	ai	good	movie	spark
1	2019-11-01 03:33:00 UTC	3	11	10	159	2
2	2019-11-01 03:36:00 UTC	2	9	12	142	6
3	2019-11-01 03:38:00 UTC	1	15	10	131	4
4	2019-11-01 03:34:00 UTC	1	7	4	127	3
5	2019-11-01 03:29:00 UTC	0	14	11	121	6
6	2019-11-01 03:37:00 UTC	1	5	14	150	3
7	2019-11-01 03:32:00 UTC	1	14	18	124	0
8	2019-11-01 03:31:00 UTC	5	11	15	140	6
9	2019-11-01 03:35:00 UTC	2	7	8	145	4
10	2019-11-01 03:30:00 UTC	0	12	13	137	5

Figure 2: Preview of the table in the BigQuery

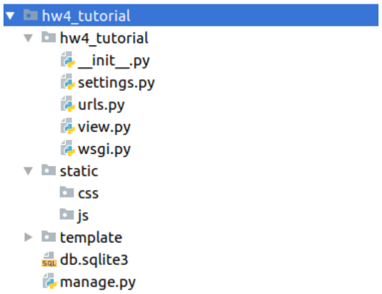
Code of data processing: Here I directly use SQL in BigQuery to process the data. Here is the SQL query.

```sql
CREATE VIEW  IF NOT EXISTS twitter_analysis.data as
SELECT time, count as data
FROM `hardy-symbol-252200.twitter_analysis.wordcount`
WHERE word="data";

CREATE VIEW IF NOT EXISTS twitter_analysis.ai as
SELECT time, count as ai
FROM `hardy-symbol-252200.twitter_analysis.wordcount`
WHERE word="ai";

CREATE VIEW  IF NOT EXISTS twitter_analysis.good as
SELECT time, count as good
FROM `hardy-symbol-252200.twitter_analysis.wordcount`
WHERE word="good";

CREATE VIEW IF NOT EXISTS twitter_analysis.movie as
SELECT time, count as movie
FROM `hardy-symbol-252200.twitter_analysis.wordcount`
WHERE word="movie";
```

```
CREATE VIEW  IF NOT EXISTS twitter_analysis.spark as
SELECT time, count as spark
FROM `hardy-symbol-252200.twitter_analysis.wordcount`
WHERE word="spark";

CREATE TABLE IF NOT EXISTS twitter_analysis.rstcnt AS
(SELECT COALESCE(t1.time, t2.time) as time, IFNULL(data, 0) as data,
IFNULL(ai, 0) as ai, IFNULL(good, 0) as good, IFNULL(movie, 0) as movie,
IFNULL(spark, 0) as spark
FROM
(SELECT COALESCE(t1.time, t2.time) as time, IFNULL(data, 0) as data,
IFNULL(ai, 0) as ai, IFNULL(good, 0) as good, IFNULL(movie, 0) as movie
FROM
(SELECT COALESCE(t1.time, t2.time) as time, IFNULL(data, 0) as data,
IFNULL(ai, 0) as ai
FROM twitter_analysis.data t1
FULL OUTER JOIN
twitter_analysis.ai t2
ON t1.time = t2.time)  t1
FULL OUTER JOIN
(SELECT COALESCE(t1.time, t2.time) as time, IFNULL(good, 0) as good,
IFNULL(movie, 0) as movie
FROM twitter_analysis.good t1
FULL OUTER JOIN
twitter_analysis.movie t2
ON t1.time = t2.time) t2
ON t1.time = t2.time) t1
FULL OUTER JOIN
twitter_analysis.spark t2
ON t1.time = t2.time);
```

2. Create Django project.

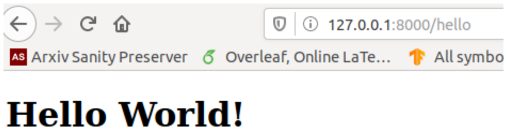
   Screenshot of Directory Structure:



```
▼ hw4_tutorial
  ▼ hw4_tutorial
      __init__.py
      settings.py
      urls.py
      view.py
      wsgi.py
  ▼ static
      css
      js
  ▶ template
    db.sqlite3
    manage.py
```

Figure 3: Screenshot of Directory Structure

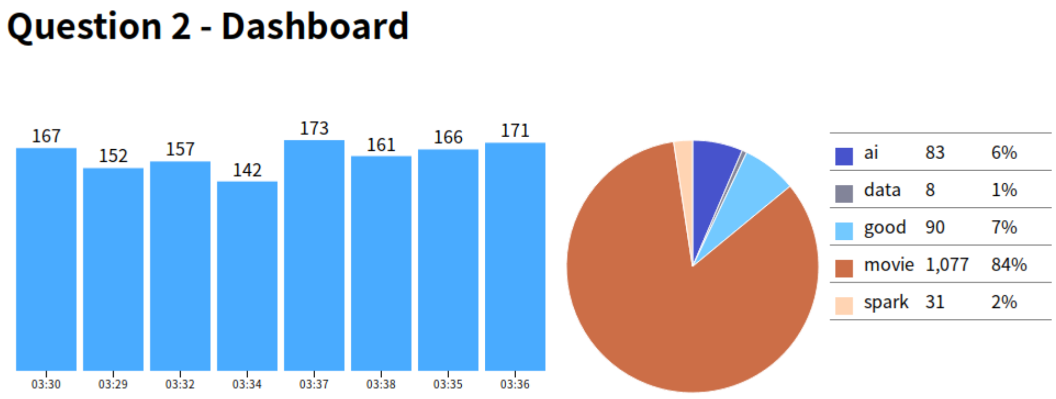
Screenshot to show `helloworld` page:



Hello World!

Figure 4: Screenshot of `helloworld` page

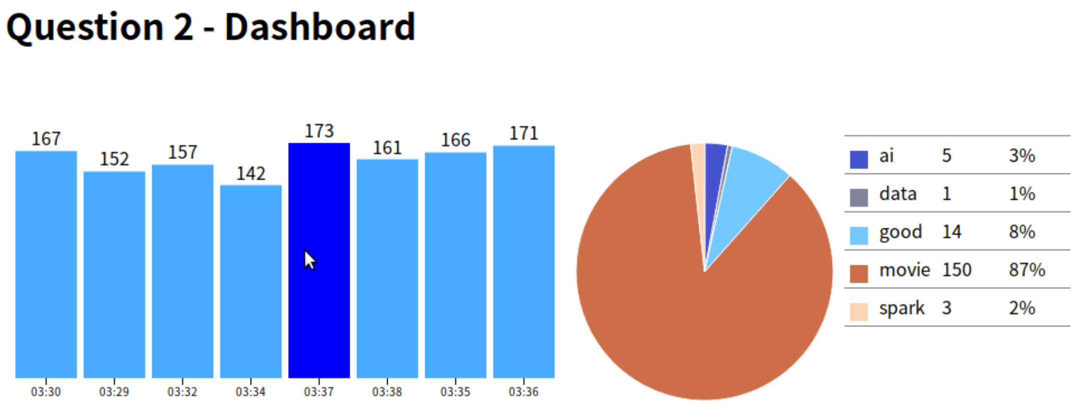
3. Finish the code. Output result:



Question 2 - Dashboard

■	ai	83	6%
■	data	8	1%
■	good	90	7%
■	movie	1,077	84%
■	spark	31	2%

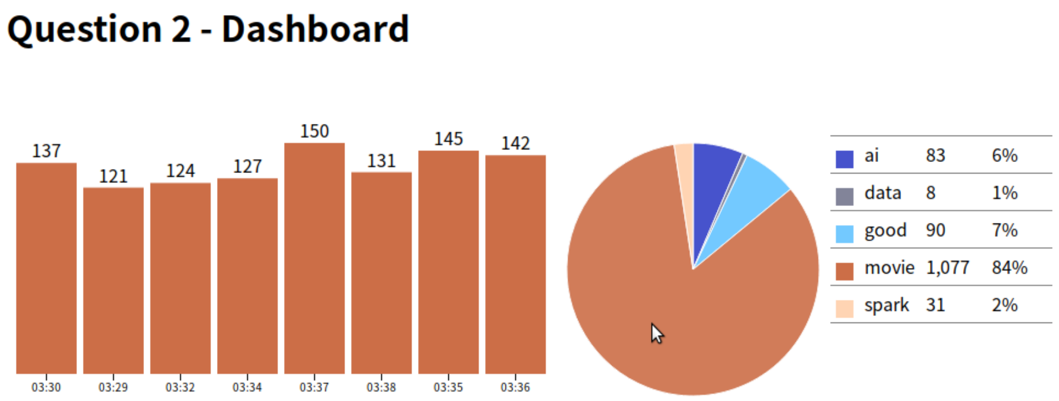
Figure 5: Screenshot of Dashboard.



Question 2 - Dashboard

■	ai	5	3%
■	data	1	1%
■	good	14	8%
■	movie	150	87%
■	spark	3	2%

Figure 6: Screenshot of Dashboard when mouse on the bar.



🟦 ai	83	6%
⬜ data	8	1%
🟦 good	90	7%
🟧 movie	1,077	84%
🟧 spark	31	2%

Figure 7: Screenshot of Dashboard when mouse on the Pie chart.

Code:

Inside `view.py`:

```
credentials = service_account.Credentials.from_service_account_file(
    '/home/huchong/Downloads/hardy-symbol-252200-25dbece318bf.json')


def dashboard(request):
    pandas_gbq.context.credentials = credentials
    pandas_gbq.context.project = "hardy-symbol-252200"

    SQL = "SELECT time, ai, data, good, movie, spark " \
          "FROM `hardy-symbol-252200.twitter_analysis.rstcnt` " \
          "LIMIT 8"
    df = pandas_gbq.read_gbq(SQL)
    df_list = df.to_dict('records')

    data_list = []
    for df_row in df_list:
        data_row = dict()
        data_row["Time"] = df_row["time"].strftime(format="%H:%M")
        df_row = dict(df_row)
        df_row.pop("time")
        data_row["count"] = df_row
        data_list.append(data_row)

    data = dict()
    data["data"] = data_list

```

```
'''
    TODO: Finish the SQL to query the data, it should be limited to 8 rows.
    Then process them to format below:
    Format of data:
    {'data': [{'Time': hour:min, 'count': {'ai': xxx, 'data': xxx, 'good': xxx, 'movie': xxx,
              {'Time': hour:min, 'count': {'ai': xxx, 'data': xxx, 'good': xxx, 'movie': xxx,
              ...
              ]
    }
'''

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

Inside dashboard.js

```
// Define the color to change if your mouse move on the bar
var barColor = '#49abff';

// Choose color for each word:
function segColor(c) {
    cmap = {
        ai: "#4753CC",
        data: "#828499",
        good: "#73C9FF",
        movie: "#CC6E47",
        spark: '#FFD4B3'
    };
    /* TO FINISH */
    return cmap[c];
}
```

```
// compute total for each state.
fData.forEach(function (d) {
    d.total = NaN;
    /* TO FINISH */
    d.total = Object.keys(d.count)
        .reduce((sum, key) => sum + d.count[key], 0);
});
```

```
bars.append("rect")
/* TO FINISH */
```

```
    .attr("x", function (d, i) {
        return x(d[0]);
    })
    /* TO FINISH */
    .attr("y", function (d) {
        return y(d[1]);
    })
    .attr("width", x.rangeBand())
    .attr("height", function (d) {
        return hGDim.h - y(d[1]);
    })
    ...
```

```
//Create the frequency labels ABOVE the rectangles.
bars.append("text").text(function (d) {
    return d3.format(",")(d[1])
})
    .attr("x", function (d) {
        /* TO FINISH */
        return x(d[0]) + x.rangeBand() / 2;
    })
    /* TO FINISH */
    .attr("y", function (d) {
        return y(d[1]) - 5;
    })
    .attr("text-anchor", "middle");
```

```
// transition the height and color of rectangles.
bars.select("rect").transition().duration(500)
/* TO FINISH */
    .attr("y", function (d) {
        return y(d[1]);
    })
    ...
```

```
// calculate total count by segment for all state.
var tF = ['ai', 'data', 'good', 'movie', 'spark'].map(function (d) {
    return {
        type: d, count: d3.sum(fData.map(function (t) {
            /* TO FINISH */
            return t.count[d];
            }))
        };
    });
```

Problem 3. Connection

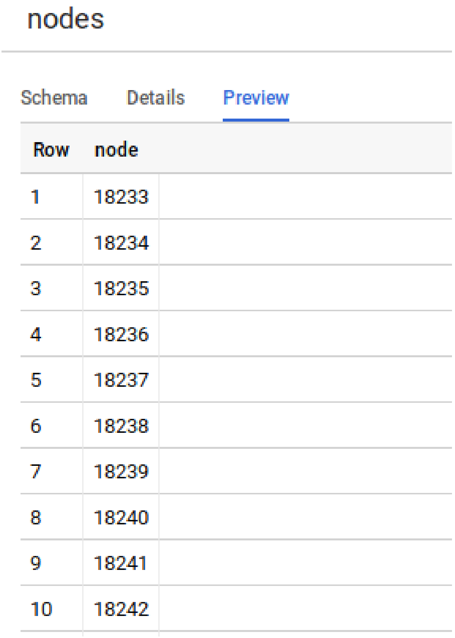
1. Data processing.

   Here I process the data in local and save required nodes and edges into `nodes.csv` and `edges.csv` separately, and then upload to `BigQuery`. Here is the python script to deal with data, including both nodes and edges data.

```python
import csv
import os

from graphframes import *
from pyspark import SQLContext
from pyspark import SparkConf, SparkContext


def getData(sc, filename):
    """
    Load data from raw text file into RDD and transform.
    Hint: transfromation you will use: map(<lambda function>).
    Args:
        sc (SparkContext): spark context.
        filename (string): hw2.txt cloud storage URI.
    Returns:
        RDD: RDD list of tuple of (<User>, [friend1, friend2, ... ]),
        each user and a list of user's friends
    """
    # read text file into RDD
    data = sc.textFile(filename)
    data = data.map(lambda line: line.split("\t")).map(
        lambda line: (int(line[0]), [int(x) for x in line[1].split(",")] if len(
            line[1]) else []))
    return data


def get_vertices(data, sqlcontext):
    """
    get vertices
    :param data: RDD list of tuple of (<User>, [friend1, friend2, ... ]),
        each user and a list of user's friends
    :param sqlcontext: SQLContext
    :return: dataframe
    """
    vertices = data.map(lambda line: (line[0],))

    return sqlcontext.createDataFrame(vertices, schema=["id"])


def get_edges(data, sqlcontext):
    """
    get edges
    :param data: RDD list of tuple of (<User>, [friend1, friend2, ... ]),
        each user and a list of user's friends
    :param sqlcontext: SQLContext
    :return:
    """

    def map_friends(line):
        """
        map function to construct edge between friends
        construct a pair of ((friend1, friend2) -> common friends list)
        if two friends are already direct friends, then common friends list
        is empty.
        :param line: tuple of (<User>, [friend1, friend2, ... ]),
                    each user and a list of user's friends
        :return: friend pair
        """
        user = line[0]
        friends = line[1]
        for i in range(len(friends)):
            yield (user, friends[i])

    edges = data.flatMap(map_friends)
    return sqlcontext.createDataFrame(edges, schema=["src", "dst"])


def save_nodes(nodes):
    """
    save node list to csv
    :param nodes: list that contain nodes in the cluster of 25 users
    :return:
    """
    with open("nodes.csv", "w") as csv_file:
        csv_writer = csv.writer(csv_file)
        csv_writer.writerow(["node"])
        csv_writer.writerows(nodes)
```

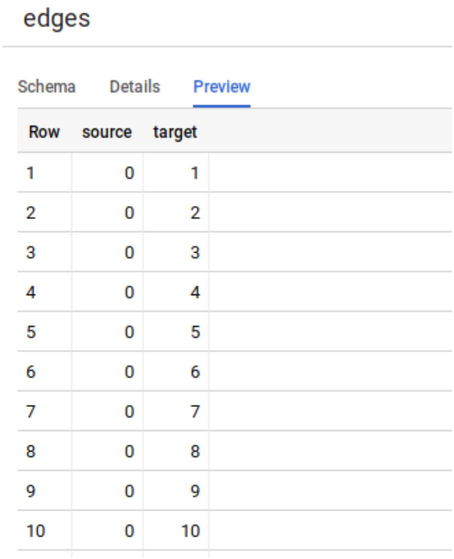
```python


def connected_components(graph):
    """
    run connected components on graph
    :param graph: Graph contains vertices and edges
    :return:
    """
    print("connected components")
    result = graph.connectedComponents(algorithm="graphx")

    # How many clusters / connected components in total for this dataset
    cluster_num = result.select("component").distinct().count()
    print("clusters amount: ", cluster_num)
    print()

    # How many users in the top 10 clusters?
    print("number of users in top 10 cluster")
    res1 = result.groupBy("component").count().orderBy('count',
                                                       ascending=False)
    res2 = res1.head(10)
    total = 0
    for row in res2:
        total += row["count"]
        print("cluster id:\t%d\tnumber of users:\t%d" % (
            row["component"], row["count"]))
    print("Total number of users in top 10 cluster:\t", total)
    print()

    # What are the user ids for the cluster which has 25 users?
    print("user ids for the cluster which has 25 users")
    cluster_id = res1.where(res1["count"] == 25).select("component").collect()
    cluster_id = [row["component"] for row in cluster_id]
    user_list = result.where(result["component"].isin(cluster_id)).select(
        "id").collect()
    user_ls = [row["id"] for row in user_list]
    user_ls.sort()
    save_nodes([[node] for node in user_ls])
    print(user_ls)
    print()

    # get edges for 25 nodes
    df_edges = graph.edges.filter(
        graph.edges.dst.isin(user_ls) & graph.edges.src.isin(user_ls))
    df_edges = df_edges.rdd.map(
        lambda x: (user_ls.index(x[0]), user_ls.index(x[1]))).toDF(
        ["source", "target"])
    # write edges to csv
    df_edges.toPandas().to_csv("edges.csv", header=True, index=False)
    return


def main():
    # Configure Spark
    if not os.path.isdir("checkpoints"):
        os.mkdir("checkpoints")
    conf = SparkConf().setMaster('local').setAppName('connected components')
    sc = SparkContext(conf=conf)
    sqlcontext = SQLContext(sc)
    SparkContext.setCheckpointDir(sc, "checkpoints")

    # The directory for the file
    filename = "q1.txt"

    # Get data in proper format
    data = getData(sc, filename)
    edges = get_edges(data, sqlcontext)
    vertices = get_vertices(data, sqlcontext)
    graph = GraphFrame(vertices, edges)
    connected_components(graph=graph)


if __name__ == '__main__':
    main()
```

Previews in the `BigQuery`.



Row	node
1	18233
2	18234
3	18235
4	18236
5	18237
6	18238
7	18239
8	18240
9	18241
10	18242

Figure 9: Screenshot of preview of `nodes` in `BigQuery`



Row	source	target
1	0	1
2	0	2
3	0	3
4	0	4
5	0	5
6	0	6
7	0	7
8	0	8
9	0	9
10	0	10

Figure 10: Screenshot of preview of `edges` in `BigQuery`

2. Finish the code.

   Inside `view.py`: Notice that possible duplicates of edges' data are already removed in the data processing step and the amount of edges is correct.

```
credentials = service_account.Credentials.from_service_account_file(
    '/home/huchong/Downloads/hardy-symbol-252200-25dbece318bf.json')


def connection(request):
    pandas_gbq.context.credentials = credentials
    pandas_gbq.context.project = "hardy-symbol-252200"
    SQL1 = 'SELECT node ' \
           'FROM `hardy-symbol-252200.graph.nodes`'
    df1 = pandas_gbq.read_gbq(SQL1)

    SQL2 = 'SELECT source, target ' \
           'FROM `hardy-symbol-252200.graph.edges`'
    df2 = pandas_gbq.read_gbq(SQL2)

    data = {
        'n': list(df1.T.to_dict().values()),
        'e': list(df2.T.to_dict().values())
    }

    '''
        TODO: Finish the SQL to query the data, it should be limited to 8 rows.
        Then process them to format below:
        Format of data:
        {
        ‘n’: [{‘node’: 18233},{‘node’: 18234},...]
        'e': [{'source':0, 'target':0},{'source':0, 'target':1},... ]
        }
    '''

    return render(request, 'connection.html', data)
```

Inside `connection.js`:

```
var svg = d3.select("body")
    .append("svg")
    /* TO FINISH */
    .attr("height", height)
    /* TO FINISH */
    .attr("width", width);
```

```
var svg_edges = svg.selectAll("line")
/* TO FINISH */
    .data(edges)
    .enter()
    /* TO FINISH */
    .append("line")
    .style("stroke", "#ccc")
    .style("stroke-width", 1);
```

```
var svg_nodes = svg.selectAll("circle")
/* TO FINISH */
    .data(nodes)
    .enter()
    /* TO FINISH */
    .append("circle")
    .attr("r", 20)
    /* TO FINISH */
    .style("fill", function (d) {
        return color(d.index);
    })
    .call(force.drag);
```

```
var svg_texts = svg.selectAll("text")
    ...
    /* TO FINISH */
    .text(function (d) {
        return d.node;
    });
```

```
force.on("tick", function () {
/* TO FINISH */
svg_edges.attr("x1", function (d) {
    return d.source.x;
})
/* TO FINISH */
    .attr("y1", function (d) {
        return d.source.y;
    })
    /* TO FINISH */
    .attr("x2", function (d) {
        return d.target.x;
    })
    /* TO FINISH */
    .attr("y2", function (d) {
        return d.target.y;
        });
```

Result:

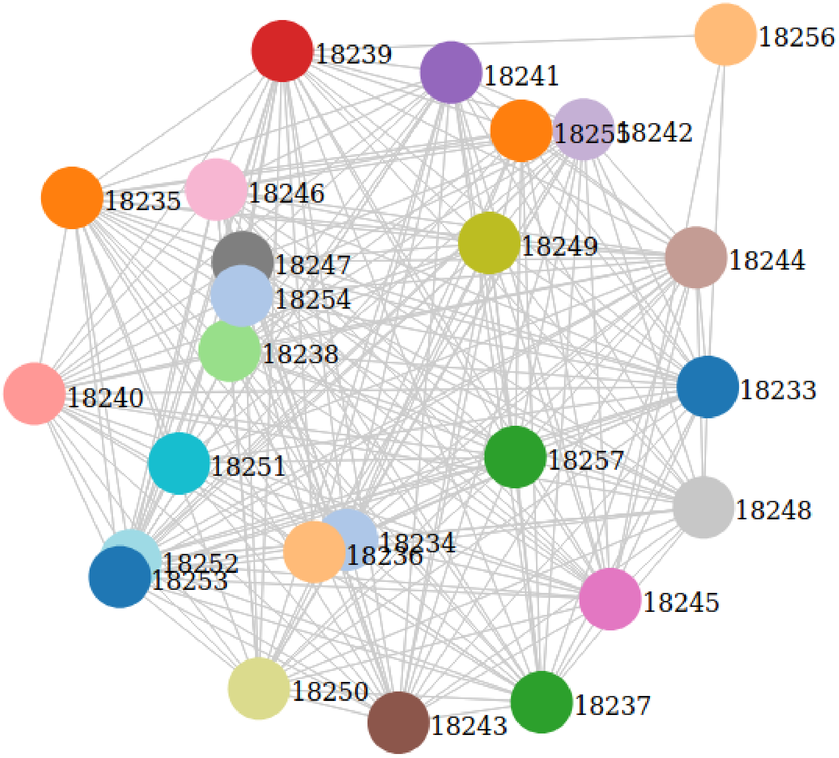


Figure 10: Screenshot of Connection output result.