Homework4

Part 1.

Problem 1

1.

1.1 SVG Coordinate Space works in the same way that mathematical graph coordinate space works except for two important features:

1) SVG Coordinate space has x=0 and y=0 coordinates fall on the top left.

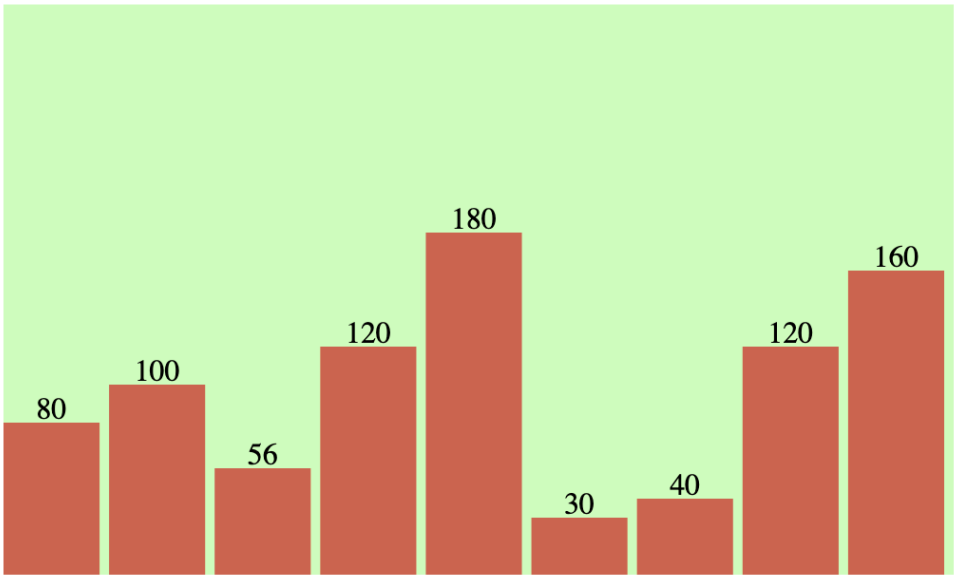
2) SVG Coordinate space has the **Y** coordinate growing from *top to bottom*.

1.2 Using D3's *enter()* and *exit()* selections, you can create new nodes for incoming data and remove outgoing nodes that are no longer needed.

1.3 The **transform** attribute defines a list of transform definitions that are applied to an element and the element's children. The translate (<x> [<y>]) transform function moves the object by x and y (i.e. $x_{new} = x_{old} + <x>$, $y_{new} = y_{old} + <y>$).

1.4 It should return [5, 6, 7, 8, 9]

2. output bar-chart



Code:

In html:

big_data_hw4_part1_samplecode.html > html

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

<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="hw4_part1.js"></script>
<script>
    barPlot()
</script>

</body>
</html>
```

In js:

hw4_part1.js > barPlot

```javascript
function barPlot() {
    var data = [80, 100, 56, 120, 180, 30, 40, 120, 160];
    var svgWidth = 500, svgHeight = 300;
    var barPadding = 5, textPadding = 2;
    var barWidth = (svgWidth / data.length);
    // 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);

    var barChart = svg.selectAll("rect")
            .data(data)
            .enter()
            .append("rect")
            .attr("class", "bar")
            .attr("transform", function (d, i) {
                var xCoordinate = barWidth * i;
                var yCoordinate = svgHeight - d;
                return "translate(" + xCoordinate + "," + yCoordinate + ")";
            })
            .attr('width', barWidth - barPadding)
            .attr('height', function(d) {return d})
            .attr("fill", "#CC6450");

    barText = svg.selectAll("text")
            .data(data)
            .enter()
            .append("text")
            .text(function(d) {return d;})
            .attr("transform", function (d, i) {
                var xCoordinate = barWidth * (i + 1/2) - barPadding / 2;
                var yCoordinate = svgHeight - d - textPadding;
                return "translate(" + xCoordinate + "," + yCoordinate + ")";
            })
            .attr("text-anchor", "middle");
}
```

Part 2.

Problem 2

Step 1: data processing

SQL code:

Query editor

```
CREATE TABLE `bigdata_sparkStreaming.wordcountwide` AS
SELECT
  time,
  MAX(CASE WHEN word = 'ai' THEN count ELSE 0 END) AS ai,
  MAX(CASE WHEN word = 'data' THEN count ELSE 0 END) AS data,
  MAX(CASE WHEN word = 'good' THEN count ELSE 0 END) AS good,
  MAX(CASE WHEN word = 'movie' THEN count ELSE 0 END) AS movie,
  MAX(CASE WHEN word = 'spark' THEN count ELSE 0 END) AS spark
FROM
  `bigdata_sparkStreaming.wordcount`
GROUP BY
  time
ORDER BY
  time
```

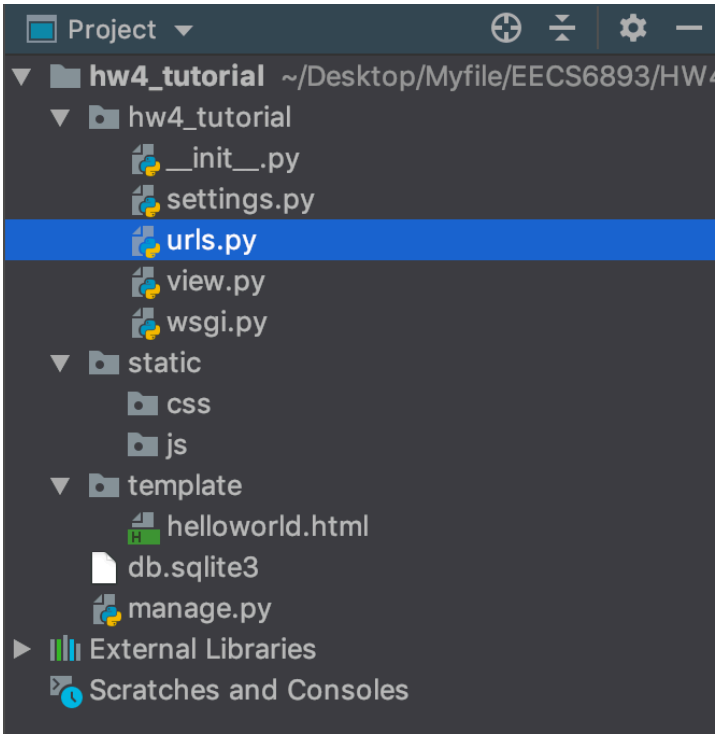
Result preview:

wordcountwide

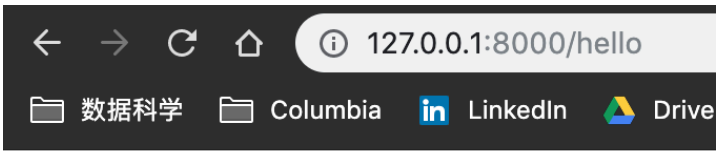
Schema Details Preview

Row	time	ai	data	good	movie	spark
1	2019-11-13 17:00:25 UTC	25	9	8	279	11
2	2019-11-13 17:01:25 UTC	23	14	12	282	11
3	2019-11-13 17:02:25 UTC	28	7	5	290	7
4	2019-11-13 17:03:25 UTC	17	8	10	285	8
5	2019-11-13 17:04:25 UTC	24	3	8	281	8
6	2019-11-13 17:05:25 UTC	24	5	12	280	5

Step 2: Django



Output:



hello world!

Step 3: code

In view.py:

```
def dashboard(request):
    pandas_gbq.context.credentials = credentials
    pandas_gbq.context.project = "test-project-251000"

    SQL = "SELECT * " \
          "FROM `bigdata_sparkStreaming.wordcountwide` " \
          "LIMIT 8"
    df = pandas_gbq.read_gbq(SQL)

    data = {}
    data['data'] = []
    for i in range(df.shape[0]):
        data['data'].append({'Time': df.iloc[i, 0].strftime('%H:%M'),
                             'count': {'ai': df.iloc[i, 1],
                                       'data': df.iloc[i, 2],
                                       'good': df.iloc[i, 3],
                                       'movie': df.iloc[i, 4],
                                       'spark': df.iloc[i, 5]}})
```

In dashboard.js:

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

```
// compute total count for each state.
fData.forEach(function (d) {
    d.total = d3.sum(Object.values(d['count'])); /* TO FINISH */
});
```

```
    //create the rectangles.
    bars.append("rect")
        .attr("x", function(d, i) {return  i * hGDim.w / 8} /* TO FINISH */)
        .attr("y", function(d) {return y(d[1])} /* TO FINISH */)
        .attr("width", x.rangeBand())
```

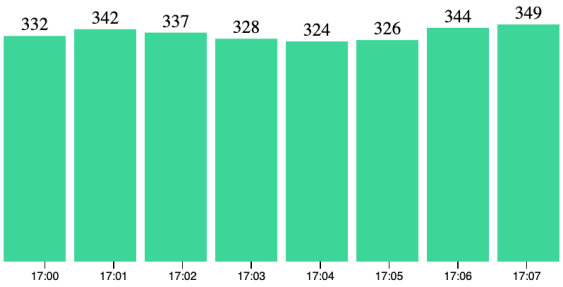
```
    //Create the frequency labels ABOVE the rectangles.
    bars.append("text").text(function (d) {
        return d3.format(",")(d[1])
    })
        .attr("x", function (d, i) {
            return i * hGDim.w / 8 + x.rangeBand() / 2 /* TO FINISH */;
        })
        .attr("y", function(d) {return y(d[1]) - 2} /* TO FINISH */)
        .attr("text-anchor", "middle");
```

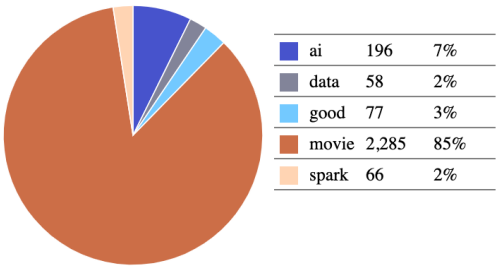
```
        // transition the height and color of rectangles.
        bars.select("rect").transition().duration(500)
            .attr("y", function(d) {return y(d[1])} /* TO FINISH */)
            .attr("height", function (d) {
                return hGDim.h - y(d[1]);
            })
```

```
// 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) {
            return t['count'][d]; /* TO FINISH */
        }))
    };
});
```

Output:

Question 2 - Dashboard





Problem 3

Step 1: data processing

Code:

```
from graphframes import *  # for graph analysis
from pyspark import SparkConf, SparkContext
import pyspark
import sys

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).map(lambda line: line.split('\t'))

    # TODO: implement your logic here
    data = data.map(lambda tmp: (tmp[0], [num for num in tmp[1].split(',')]))

    return data
```

```
def getEdges(line):
    """
    get edges from input data

    Args:
        line (tuple): a tuple of (<User1>, [(<User2>, 0), (<User3>, 1)....])
    Returns:
        RDD of tuple (line[0], connected friend)
    """
    friends = line[1]
    for i in range(len(friends)):
        # Direct friend
        yield (line[0], friends[i])

conf = SparkConf()
sc = pyspark.SparkContext.getOrCreate(conf=conf)
sc.setCheckpointDir('/checkpoints')
# The directory for the file
filename = "gs://big_data_hw/hw2/q1.txt"

# Get data in proper format
data = getData(sc, filename)
vertices = data.map(lambda x: (x[0],))
edges = data.flatMap(getEdges)
V = spark.createDataFrame(vertices, ["id"])
E = spark.createDataFrame(edges, ["src", "dst"])
G = GraphFrame(V, E)
compo = G.connectedComponents()
```

```
# nodes
qNodes = compo.filter(compo["component"] == 103079215141).select('id')
qNodes.write.save('gs://big_data_hw/hw4/ndoes', format="json", mode="overwrite")

# edges
tmp = [int(q.id) for q in qNodes.collect()]
# filter query text
qtext = "("
for i in range(len(tmp)-1):
    qtext += "src = {} or ".format(tmp[i])
qtext += "src = {}) and (".format(tmp[-1])
for i in range(len(tmp)-1):
    qtext += "dst = {} or ".format(tmp[i])
qtext += "dst = {})".format(tmp[-1])

qEdges = G.filterEdges(qtext).edges
x = qEdges.toPandas().replace([str(i) for i in tmp], [str(i) for i in range(len(tmp))])
qEdges = spark.createDataFrame(x)
qEdges.write.save('gs://big_data_hw/hw4/edges', format="json", mode="overwrite")
```

```
import sys
import requests
import subprocess
from google.cloud import bigquery

# nodes
files = 'gs://big_data_hw/hw4/ndoes' + '/part-*'
subprocess.check_call(
    'bq load --source_format NEWLINE_DELIMITED_JSON '
    '--replace '
    '--autodetect '
    '{dataset}.{table} {files}'.format(
        dataset='my_dataset', table='nodes', files=files
    ).split())
output_path = sc._jvm.org.apache.hadoop.fs.Path('gs://big_data_hw/hw4/ndoes')
output_path.getFileSystem(sc._jsc.hadoopConfiguration()).delete(
    output_path, True)
```

```
# edges
files = 'gs://big_data_hw/hw4/edges' + '/part-*'
subprocess.check_call(
    'bq load --source_format NEWLINE_DELIMITED_JSON '
    '--replace '
    '--autodetect '
    '{dataset}.{table} {files}'.format(
        dataset='my_dataset', table='edges', files=files
    ).split())
output_path = sc._jvm.org.apache.hadoop.fs.Path('gs://big_data_hw/hw4/edges')
output_path.getFileSystem(sc._jsc.hadoopConfiguration()).delete(
    output_path, True)
```

BigQuery Table:

nodes

Schema Details Preview

Row	id
1	18233
2	18234
3	18235
4	18236
5	18237

edges

QUERY TABLE COPY

Schema Details Preview

Row	dst	src
1	0	3
2	0	4
3	0	5
4	0	6
5	0	7

Rows per page: 100 1 - 100 of 452

Step 2: Finish the code

In view.py:

```
def connection(request):
    pandas_gbq.context.credentials = credentials
    pandas_gbq.context.project = "test-project-251000"
    SQL1 = "SELECT id " \
           "FROM `my_dataset.nodes`"
    df1 = pandas_gbq.read_gbq(SQL1)

    SQL2 = "SELECT src, dst " \
           "FROM `my_dataset.edges` "
    df2 = pandas_gbq.read_gbq(SQL2)

    data = {}
    data['n'] = []
    data['e'] = []

    for i in range(df1.shape[0]):
        data['n'].append({'node':df1.iloc[i,0]})

    for j in range(df2.shape[0]):
        data['e'].append({'source':df2.iloc[j,0], 'target':df2.iloc[j,1]})

    '''
        TODO: Finish the SQL to query the data
        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)
```

In connection.js:

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

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

```
    var svg_nodes = svg.selectAll("circle")
                       .data(nodes /* TO FINISH */)
                       .enter()
                       .append("circle" /* TO FINISH */)
                       .attr("r",20)
                       .style("fill", function(d) {return "#"+((1<<24)*Math.random()|0).toString(16)} /*
                       .call(force.drag);
```

```
var svg_texts = svg.selectAll("text")
                   .data(nodes)
                   .enter()
                   .append("text")
                   .style("fill", "black")
                   .attr("dx", 20)
                   .attr("dy", 8)
                   .text( function(d) {return d['node']} /* TO FINISH */ );
```

```
force.on("tick", function(){
    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} /* TO FINISH */);
```

Output:

Question 3 - Connection