# Major assignment 2: Your submission

This is your assignment template for [CompX Major assignment 2](https://courses.edx.org/courses/course-v1:AdelaideX+CompX+3T2018/courseware/79355773219d46ce8241e39d7cdce010/a97f27d67b97489aacee42f78ee03357/1?activate_block_id=block-v1%3AAdelaideX%2BCompX%2B3T2018%2Btype%40vertical%2Bblock%40a84c6f7dfc2148ca9535b6520818daa1). Save this document on our local machine and include all of your work within the relevant sections. Once you’ve completed all five parts of the assignment, upload the document via the submission area on the “[Submit your assignment](https://courses.edx.org/courses/course-v1:AdelaideX+CompX+3T2018/courseware/79355773219d46ce8241e39d7cdce010/a97f27d67b97489aacee42f78ee03357/14?activate_block_id=block-v1%3AAdelaideX%2BCompX%2B3T2018%2Btype%40vertical%2Bblock%40022fb40d796042f294a4a9c9cff714ed)” page at the end of Major assignment 2.

# You will need to:

# Include both the answer to the question, and The code as required.

# Your answer will assist the University of Adelaide academic staff member assess your code submission. The point(s) value at the end of each question is only for your code as you will have received points for correctly answering the question in the course (on edX).

**Quick links:**

[Major assignment 2: Part 1](#_Major_assignment_2:_1)

[Major assignment 2: Part 2](#_Major_assignment_1:_1)

[Major assignment 2: Part 3](#_Major_assignment_1:_2)

[Major assignment 2: Part 4](#_Major_assignment_1:_3)

[Major assignment 2: Part 5](#_Major_assignment_2:)

# Major assignment 2: Part 1

1. Read in the flights dataset from the NYC flights package [1 point]

Your relevant code goes here:

    /\*\*

     \* Populate the with data read from a CSV file.

     \* @param path The path to the CSV to read the data from.

     \*/

    public void loadData(String path) {

        // initialize the columns and rows

        this.columns = new ArrayList<String>();

        this.rows = new ArrayList<TableRow>();

        // open the file for reading

        File f = new File(path);

        try {

            Scanner sc = new Scanner(f);

            if (sc.hasNextLine()) {

                // process the header row

                String line = sc.nextLine();

                // yes this is a total cheat, but not worth the effort when this is a bad way to read csv files anyway

                String columns[] = line.replace("\"", "").split(",");

                for (String column : columns) {

                    this.addColumn(column);

                }

                // load the table rows

                while( sc.hasNextLine() ) {

                    line = sc.nextLine();

                    String values[] = line.replace("\"", "").split(",");

                    // check for NA values when the row should be exluded

                    boolean include = true;

                    for (String value : values) {

                        if (value.equals("NA")) {

                            include = false;

                            break;

                        }

                    }

                    // add the row to the dataset

                    if (include) {

                        TableRow row = this.addRow();

                        for (int i = 0; i < values.length; i++) {

                            row.setString(i, values[i]);

                        }

                    }

                }

            }

        } catch (FileNotFoundException ex) {

            System.out.println("File " + f + " not found.");

        }

    }

1. Provide evidence of how you were able to count the number of flights leaving EWR, JFK and LGA [1 point]

Your answers to number of flights leaving EWR, JFK and LGA go here:

|  |
| --- |
| Flights leaving EWR: 117127  Flights leaving JFK: 109079  Flights leaving LGA: 10114 |

Your relevant code goes here:

    /\*\*

     \*  Finds the rows in the Table that contain the value provided, and returns references to those rows.

     \* @param value The value to match.

     \* @param columnName Title of the column to search.

     \* @return Returns an ArrayList of rows matching the criteria.

     \*/

    public ArrayList<TableRow> findRows(String value, String columnName) {

        ArrayList<TableRow> results = new ArrayList<TableRow>();

        for (TableRow row : this.rows) {

            if (row.getString(columnName).equals(value)) {

                results.add(row);

            }

        }

        return results;

    }

}

        // find the number of flights leaving airports

        String[] origins = {"EWR", "JFK", "LGA"};

        for (String origin : origins) {

            System.out.println(String.format(

                "Flights leaving %s: %s",

                origin,

                flights.findRows(origin, "origin").size()));

        }

1. Provide evidence of how you were able to calculate the minimum distance, median, mean and maximum distance of flights leaving LGA [1 points]

Your answers to minimum distance, median, mean and maximum distance go here:

|  |
| --- |
| LGA min : 96.0  LGA median : 762.0  LGA mean : 784.7561103421001  LGA max : 1620.0 |

Your relevant code goes here:

    /\*\*

     \* Determine the max value from a set of data points.

     \*

     \* @param data The float array of values to determine the maximum value for.

     \* @return Return the max value.

     \*/

    public static double getMax(double data[]) {

        double max = data[0];

        for (int i = 0; i < data.length; i++) {

            if (data[i] > max) {

                max = data[i];

            }

        }

        return max;

    }

    /\*

     \* Determine the max value from a set of data points.

     \*/

    public static double getMin(double data[]) {

        double min = data[0];

        for (int i = 0; i < data.length; i++) {

            if (data[i] < min) {

                min = data[i];

            }

        }

        return min;

    }

    /\*

     \* Calculate the mean for the data points.

     \*/

    public static double getMean(double[] data) {

        double total = 0;

        // calculate the sum of the data points

        for (int i = 0; i < data.length; i++) {

            total += data[i];

        }

        return total / data.length;

    }

    /\*

     \* Determine the data point median

     \*/

    public static double getMedian(double[] data) {

        Arrays.sort(data);

        int middle = data.length / 2;

        if (data.length % 2 == 0) {

            return (data[middle - 1] + data[middle]) / 2.0;

        } else {

            return data[middle];

        }

    }

        System.out.println("\n--- Part 1: Question 4-7");

        // get the LGA rows

        Table lgaTable = new Table(flights.findRows("LGA", "origin"));

        // get distance as a double array for processing

        double[] lgaDistance = lgaTable.toDoubleArray("distance");

        System.out.println("LGA min    : " + Flights.getMin(lgaDistance));

        System.out.println("LGA median : " + Flights.getMedian(lgaDistance));

        System.out.println("LGA mean   : " + Flights.getMean(lgaDistance));

        System.out.println("LGA max    : " + Flights.getMax(lgaDistance));

# Major assignment 2: Part 2

1. Analyse the different aircraft, paying attention to the tailnum field, in order to work out how many different aircraft operated for three airlines: UA, HA and B6 [1 point]

Your answers to how many aircraft operated for UA, HA and B6 go here:

|  |
| --- |
| Unique Tailnums for UA: 620  Unique Tailnums for HA: 14  Unique Tailnums for B6: 193 |

Upload the entire code for this part, with specific reference to the output for each airline:

    /\*\*

     \* Get the unqiue values a column contains.

     \* @param columnName The name to retrieve the list of unique values for.

     \* @return A String array containing unique values.

     \*/

    String[] getUnique(String columnName) {

        // get the array of values for the column

        String[] values = this.toArray(columnName);

        List<String> valuesList = Arrays.asList(values);

        // use a hash set to get the unique values

        HashSet<String> valueSet = new HashSet<String>();

        valueSet.addAll(valuesList);

        return valueSet.toArray(new String[0]);

    }

}

        System.out.println("\n--- Part 2: Question 1-3");

        String[] carriers = {"UA", "HA", "B6"};

        for (String carrier : carriers) {

            // get the flights operated by the carrier

            Table carrierTable = new Table(flights.findRows(carrier, "carrier"));

            System.out.println(String.format(

                "Unique Tailnums for %s: %s",

                carrier,

                carrierTable.getUnique("tailnum").length));

        }

2. Analyse B6, paying attention to how many different destinations B6 flew to in December [1 point]

Your answer to how many different destinations B6 flew to goes here:

|  |
| --- |
| B6 unique Destinations (December): 39 |

Your relevant code goes here:

        System.out.println("\n--- Part 2: Question 3");

        Table b6Table = new Table(flights.findRows("B6", "carrier"));

        Table b6December = new Table(b6Table.findRows("12", "month"));

        System.out.println("B6 unique Destinations (December): " + b6December.getUnique("dest").length);

3. Analyse LGA, paying attention to the airports listed in the dataset that cannot be reached directly from LGA by taking a single flight [1 point]

Your answer to how many of the airports cannot be reached directly from LGA by taking a single flight goes here:

|  |
| --- |
| Total destinations with no direct flight from LGA: 36 |

Your code goes here:

        System.out.println("\n--- Part 2: Question 5");

        // get the total number of destinations

        int destTotal = flights.getUnique("dest").length;

        // get the destinations of LGA

        int destLGA = new Table(

            flights.findRows("LGA", "origin"))

            .getUnique("dest")

            .length;

        // show the unreachable destinations count

        System.out.println("Total destinations with no direct flight from LGA: " + (destTotal - destLGA));

# Major assignment 2: Part 3

1. Identify how many existing flights will get re-allocated to ECI. Note: Round the number of flights [2 points]

Your answer to how many existing flights will get re-allocated to ECI goes here:

*This requirement is very ambiguous, I would actually go back to the product owner and the team for clarification since there are multiple ways to interpret this. I went for a simple mathematical approach to at least show one approach to estimate the number: 4/13 flights will be re-allocated to ECI.*

|  |
| --- |
| Re-allocations: 100722 |

Your code goes here:

    /\*\*

     \* Estimate how many flights will be re-scheduled when ECI is added.

     \* This is based on the simple realization that given the number of runways about 30% of flights will be send to ECI.

     \* @return The estimated number of flight changes.

     \*/

    public long estimateFlightChanges() {

        // calculate the ratio of flights ECI should receive

        int totalRunways = 3 + 2 + 4 + 4;

        double eciRatio = 4.0 / totalRunways;

        // get the total number of flights

        int totalFlights = flights.getRowCount();

        // calculate the number of flight changes

        double flightChanges = eciRatio \* totalFlights;

        // return the result

        return Math.round(flightChanges);

    }

        FlightScheduler scheduler = new FlightScheduler();

        scheduler.loadData("section10/Flights.csv");

        // calculate flight re-allocation

        System.out.println("Re-allocations: " + scheduler.estimateFlightChanges());

2. Using HashMaps or HashSets, write a constructor, FlightScheduler(), for the class FlightScheduler [2 points]

Your code goes here:

    protected Flights flights;

    protected Map<String, TableRow> clusteredFlightIndex;

    /\*\*

     \* Initialize a default instance of the class.

     \*/

    public FlightScheduler() {

        this.flights = new Flights();

        this.clusteredFlightIndex = new HashMap<String, TableRow>();

    }

3. Write a function to load the data for the class FlightScheduler with the following signature:  
void loadData(String flightDataFile) [2 points]

Your code goes here:

    /\*\*

     \* Read flight data from a CSV file.

     \* @param flightDataFile The path to the CSV file to load.

     \*/

    public void loadData(String flightDataFile) {

        // initilize this variable again in case of a second load

        this.clusteredFlightIndex = new HashMap<String, TableRow>();

        // load the data from CSV

        Table sourceData = new Flights(flightDataFile);

        // create the data table with the required additional column

        this.flights = new Flights();

        for (String columnName : sourceData.columns) {

            this.flights.addColumn(columnName);

        }

        // add the new column

        this.flights.addColumn("orignal\_origin");

        // add the source rows and initialize the orignal\_origin column

        for (TableRow row : sourceData.rows) {

            TableRow newRow = this.flights.addRow();

            // add the original data

            for (String columnName : row.columns) {

                newRow.setString(columnName, row.getString(columnName));

            }

            // initialize the orignal\_origin column

            newRow.setString("orignal\_origin", null);

            // add the row to the index

            this.addToIndex(newRow);

        }

    }

    /\*\*

     \* Get the clustered key for a fligh in the index.

     \* @param day The day of the flight.

     \* @param month The flight month.

     \* @param year The year of the flight.

     \* @param flightCode The flight code.

     \* @return Return the clustered key as a string.

     \*/

    protected String getClusterKey(String day, String month, String year, String flightCode) {

        return String.format("%s\_%s\_%s\_%s",

            year,

            month,

            day,

            flightCode

        );

    }

    /\*\*

     \* Add a row to the clustered index.

     \* @param row The row to add to the index.

     \*/

    protected void addToIndex(TableRow row) {

        String clusteredKey = this.getClusterKey(

            row.getString("day"),

            row.getString("month"),

            row.getString("year"),

            row.getString("flight"));

        // add the row to the index

        this.clusteredFlightIndex.put(clusteredKey, row);

    }

4. Write a function to reallocate flights for the class FlightScheduler with the following signature:  
void reallocate (int day, int month, int year, String flightCode) [2 points]

Your code goes here:

    /\*\*

     \* Re-allocate a flight to ECI.

     \* @param day The day of the flight.

     \* @param month The flight month.

     \* @param year The year of the flight.

     \* @param flightCode The flight code.

     \*/

    public void reallocate(int day, int month, int year, String flightCode) {

        // get the clustered key

        String clusteredKey = this.getClusterKey(

            String.valueOf(day),

            String.valueOf(month),

            String.valueOf(year),

            flightCode);

        // get the row to update

        TableRow row = this.clusteredFlightIndex.get(clusteredKey);

        // set the new origin

        row.setString("orignal\_origin", row.getString("origin"));

        row.setString("origin", "ECI");

    }

5. Write a function to check if a flight is moved for the class FlightScheduler with the following signature:  
boolean check(int day, int month, int year, String flightCode)  
[2 points]

Your code goes here:

    /\*\*

     \* Check if a flight is moved to ECI

     \* @param day The day of the flight.

     \* @param month The flight month.

     \* @param year The year of the flight.

     \* @param flightCode The flight code.

     \* @return Returns true if the flight has been moved, false otherwise.

     \*/

    public boolean check(int day, int month, int year, String flightCode) {

        // get the clustered key

        String clusteredKey = this.getClusterKey(

            String.valueOf(day),

            String.valueOf(month),

            String.valueOf(year),

            flightCode);

        // get the row to update

        TableRow row = this.clusteredFlightIndex.get(clusteredKey);

        return row.getString("orignal\_origin") != null;

    }

# Major assignment 2: Part 4

1. Identify how many different airports you can reach from EWR by taking two (2) flights. Note: This means that you need to take one flight from EWR to airport X, and then take a flight from airport X to somewhere else   
[2 points]

Your answer to how many different airports can be reached from EWR by taking two (2) flights goes here:

Your code goes here:

2. Identify how many different airports you can reach from EWR by taking three (3) flights   
[2 points]

Your answer to how many different airports can be reached from EWR by taking three (3) flights goes here:

Your code goes here:

# Major assignment 2: Part 5

1. Based on the assumptions outlined within the scenario, identify the maximum number of flights that you could have taken in 2013 [6 points]

Your answer to the maximum number of flights you can take in 2013 goes here:

Your code goes here: