# CS 1132 Fall 2021 Assignment 3 due 10/18 at 11:59 pm

***Adhere to the Code of Academic Integrity.*** You may discuss background issues and general strategies with others and seek help from course staff, but the implementations that you submit must be your own. In particular, you may discuss general ideas with others but you may not work out the detailed solutions together. It is never OK for you to see or hear another student's code and it is never OK to copy code from published/Internet sources. If you feel that you cannot complete the assignment on you own, seek help from the course staff.

Do not use the `break`, `continue`, or `return` statements in any homework or test in CS 1132.

# 1 Data exploration: public transport utility data worldwide

In this assignment you will work through real data from a partial list of public transport utilities (airports, train stations, ferry terminals, etc.) around the world. The given data file has been pre-processed only minimally, so you will get to work with a set of data that is not perfectly simplified—a realistic scenario! You need to process the file—import data, manipulate, and compute—and then answer a set of questions and produce a couple of visualizations. These answers and visualizations must be achieved *programmatically* and not hard-coded; i.e., if we run your script against a different dataset (with the same format), its results must be correct for that new data.

Completing this assignment will help reinforce your knowledge on manipulating cell arrays and `char` arrays and will further give you the opportunity to explore some of MATLAB's graphics tools. You will also practice using MATLAB documentation to look up functions and their use.

## 1.1 Data file

The data file named `TransportData.csv`, containing information on public transport utilities, will be used for this assignment. Each row of text in the file lists the information of one public transport utility; there are ten data fields for each utility (i.e., there are ten data items on each row):

1. Public transport utility name - name of the airport, train station, ferry terminal, etc.
2. City - city where the utility is located.
3. Country - country where the utility is located.
4. Latitude - latitude of the utility's location in **degrees**.
5. Longitude - longitude of the utility's location in **degrees**.
6. Altitude - altitude of the utility's location in ft.
7. Timezone - nominal hours offset from UTC.
8. DST - daylight savings time. Can be one of E (Europe), A (US/Canada), S (South America), O (Australia), Z (New Zealand), N (None) or U (Unknown).
9. Tz database time - Timezone in "tz" (Olson) format, e.g., "America/Los_Angeles".
10. Type - type of the public transport utility. Can be one of "airport" for air terminals, "station" for train stations, "port" for ferry terminals and "unknown" if not known.

Take a look at the data file using any text editor, e.g., the MATLAB editor (double-click on the filename within the MATLAB desktop). Don't be intimidated by the amount of information contained in this data set! We will not be using all the information in `TransportData.csv`; we have left some of them for you to explore in case you're interested.

## 1.2 Required, allowed, and forbidden functions

You will submit five function files: `findMatch.m`, `readTransportData.m`, `splitTimeZones.m`, `indexAirports.m`, `tabulateDistances.m` and the script `assignment3.m`. The given file `cellstr2str.m` is a completed function that you can download and use.

*Use only the built-in functions listed below for handling strings (*`char` *arrays) and files;* you may not need all of them. You can of course still use general built-in functions not related specifically to strings and files, such as `length`, `zeros`, `cell`, etc. You can use:

- `fopen`, `fclose`, `feof`, `fgetl` for file handling,
- `textscan` (explained below) for parsing (separating into parts) a string, and `strcmpi` or `strcmp` for comparing strings.

| You must *not* use built-in functions `find`, `strfind`, `findstr`, and `contains` in this assignment. |
|---|

### 1.2.1 Matching strings

Implement the following function:

```
function idx= findMatch(C, s)
% C is a 1-d cell array of strings with unique entries.
% Find the first cell in C that matches string s completely, ignoring case.
% idx is a vector of the first index of C where a match occurs.
% E.g., if  C is {'Matt'; 'uses'; 'Matlab'; 'on'; 'a'; 'mAt'}
%       and  s is 'Mat', then idx is [6].
% If there is no match then idx is [].
```

Look for opportunities to be efficient—do you always need to scan the whole array? As a reminder, built-in function `strcmp` is case-sensitive while `strcmpi` is case-insensitive; in this assignment we will use case-insensitive comparisons.

For this and all other functions that you write where a function specification—function header comment—is given, be sure to copy *exactly both* the function header *and* the function comment into your function file. I.e., the function specification should be complete in your file.

Be sure to work on and test your functions one at a time! The example in this function comment is a test that you can use; you should create other tests as well. Make effective use of this function in the remaining parts of the assignment.

### 1.2.2 Reading the data set

We will now write a function to transfer the data that is stored in `TransportData.csv` into a MATLAB array to enable us to manipulate the data later on. Since the information contained in `TransportData.csv` is a mixture of strings and numbers, a cell array is ideal for storing the data. Implement the following function[1] as specified:

[1] CSV is a very common format for tabular data that you will likely encounter in the future; when you do, use one of Matlab's built-in functions, like `readmatrix`, to parse it—they will be faster and properly handle corner cases we are ignoring here. But for now, we need to practice the lower-level file routines.

```
function transportData = readTransportData(filename)
% This function reads the contents of filename into cell array transportData.
% filename is a text file in which the first line stores the headers of the
% dataset while the remaining rows contain the data to be stored in
% transportData, with an exception explained below.  Each column of data in
% filename is delimited by ','.
% Exception:  if the 9th column of a row of data contains the value '\N' or
% '' (empty char array), that row of data should not be stored in
% transportData.
%
% transportData is a 2D cell array where each row corresponds to a row in
% filename excluding the header and each column corresponds to a column of
% data in filename.  I.e., transportData{r,c} stores the data in the rth
% row cth column of filename, not counting the header row and any row whose
% 9th column has the value '\N' or ''.
```

Before implementing this function, inspect the data set first to see how it looks. As you can observe, the first line of the data set contains the header names of the columns but we are only storing the information contained from line 2 onwards. Therefore, your function should read the file line by line (but discard/ignore the first line read). Each line of this data set contains data items delimited by ','. You will convert each line of the data set into a 1-d cell array where each cell stores one data item. However, we exclude the rows in `TransportData.csv` for which the Tz database time (9th column) contains the value '\N'[2] or '' (empty char array) as it disrupts the data analysis later on.

The built-in function `textscan` is useful for parsing strings that represent both numeric and text information and that use a known delimiter. Here is an example for parsing a string delimited by commas:

```
str= 'abc,5,-3,h,,defg';  % note that the 5th data item is missing
data= textscan(str, '%s %f %f %s %f %s', 'delimiter', ',', 'EmptyValue', -Inf);
```

In the call to `textscan` above ...

- The first argument is the string (variable) to parse, `str`.
- The second argument indicates the expected format of the individual data items in the string: `'%s %f %f %s %f %s'` indicates *string number number string number string*, just like the format sequences used in `fprintf` statements.
- the third and fourth arguments above say that the delimiter is a comma.
- The fifth and sixth arguments above say that any missing value is to be replaced by `-Inf`. We will not be making use of this feature in this assignment, however.

`textscan` returns a 1-d cell array of the data items, but it deals with strings in a peculiar way.[3] Continuing with the above example, `data` would store the *nested* cell array `{ {'abc'}, 5, -3, {'h'}, -Inf, {'defg'} }`, i.e. each string data item is stored as a cell array within a cell array. To circumvent this problem, we have provided the function `cellstr2str` which removes the nesting. Using `cellstr2str` on `data` above would return the non-nested 1-d cell array `{ 'abc', 5, -3, 'h', -Inf, 'defg' }`. Read the function header of `cellstr2str` to learn how to use it.

Finally, how do you build a 2-d cell array given two 1-d cell arrays? Use *concatenation*! Here's an example:

```
data1= { 'abc', 5, -3, 'h', -Inf, 'defg' };
data2= { 'aaa', 5,  6, 'h',    2, 'xyzzz' };
arr= [data1; data2];
```

Note the use of *square brackets*, not braces, for the concatenation operation. If you use braces, then you would end up with a nested cell array instead of the requested *2-d* cell array.

[2] Unlike other languages, Matlab does *not* use backslash as an "escape character," except in the context of `fprintf`, so '\N' can be entered as-is.

[3] `textscan` is a useful function for handling delimited data but has many peculiarities. For future use (outside of CS 1132) you would want to read through MATLAB's documentation to learn additional options and do some experiments!

Make sure that this function works correctly before proceeding with the other functions. The output of this function will be used as an input for the remaining functions you have to write and it would be important for you to be able to test those functions using the correct output from `readTransportData`. Note that it may take your code as long as a minute to run with the given data set. *Program development hint: open* `TransportData.csv` *using any text editor and copy just the first few rows of data into another file. Then do your initial testing by reading the smaller data file in order to reduce the function execution time.*

## 1.3 Splitting time zone entries

As you can observe from our data set, the Tz database time is written in the format 'area/location' where "area" is the name of a continent or an ocean while "location" typically refers to a major city in the region. This level of granularity is a bit too fine-grained for our purposes; grouping by area alone produces more interesting statistics. Therefore, we need to write code to separate the information contained before and after the '/' in each entry in the 9th column. Implement the following function as specified:

```
function [areas,locations] = splitTimeZones(transportData)
% This function segregates the data value in one of the columns in
% transportData for further data processing.
%
% transportData is a 2D cell array where each row represents a hub of
% public transport. The 9th column contains information about the timezone
% where the public transport utility is located in the tz (Olson) format.
% Each entry in this column takes on the form 'area/location'.
%
% areas and locations are 1D cell arrays such that areas{k} and locations{k}
% store the timezone information separated into area and location, respectively,
% for the kth row in transportData.
```

For full credit, you must design and make use of a non-trivial subfunction. Make sure that the function header in your subfunction has a brief description about the subfunction and its inputs and outputs.

## 1.4 Indexing airport information

As you may have noticed, the majority of the transport utilities in the data set are airports. Hence, we will write code to further segregate the airport data by determining the indices of the rows in the data set corresponding to airports. We will also implement code to obtain a unique list of countries and time zones represented by these airports. Additionally, we will index the rows containing airport information according to the country and time zone area the airport is located in. This would facilitate analyzing statistics such as the number of airports per country or per time zone area as we will investigate later on.

Implement the following function as specified:

```
function [countries,countryIDs,uniqueAreas,areaIDs,airportIDs] = ...
    indexAirports(transportData,areas)
% This function indexes the data in transportData by identifying the
% countries and time zone areas of all airports contained in the data set.
% The data corresponding to public transport utilities that are not
% airports will not be considered.
%
% Inputs:
%
% transportData is a 2D cell array where each row represents a mode of
% public transport. The 3rd column represents the country where the the
% public transport utility is located while the 10th column represents the
% type of the utility.
%
% areas is a 1D cell array such that areas{k} stores the timezone area
% for the kth row in transportData.  (See function splitTimeZones.)
%
% Outputs:
%
% countries is a 1D cell array that stores all the unique countries
% referenced in transportData that have at least one public transport
% utility whose type is 'airport'.
%
% countryIDs is a 1D cell array such that countryID{k} is a numeric array
% storing the row numbers of transportData whose country is countries{k}
% and whose public transport utility type is 'airport'.
%
% uniqueAreas is a 1D cell array that stores all the unique timezone areas
% for which there is a public transport utility of type 'airport'.
%
% areaIDs is a 1D cell array such that areaIDs{k} is a numeric array
% storing the row numbers of transportData whose timezone area is uniqueAreas{k}
% and for which the public transport utility type is 'airport'.
%
% airportIDs is a 1D numeric array that stores the row numbers of
% transportData whose public transport utility
% type is 'airport'.
```

For full credit, make effective use of the function `findMatch` that you implemented. In addition, you should only traverse the rows of `transportData` once. Since data is always growing over time, it's important that we're not too wasteful when processing it, or one day our code could be too slow to effectively do it's job.

## 1.5 Distances between transport utilities

The latitude and longitude data allow us to find the distance between each pair of transport utilities (not only airports!). We use the Haversine formula to compute the distance between two points on a sphere given their latitude and longitude. Consider two points $P_1$ and $P_2$ with (latitude,longitude) measurements in **radians** denoted by $(\phi_1, \lambda_1)$ and $(\phi_2, \lambda_2)$, respectively. The Haversine distance is given by

$$2r \arcsin \sqrt{\sin^2\left(\frac{\phi_1 - \phi_2}{2}\right) + \cos(\phi_1)\cos(\phi_2)\sin^2\left(\frac{\lambda_1 - \lambda_2}{2}\right)}$$

where $r$ is the Earth's radius. Implement the following function as specified:

```
function distances = tabulateDistances(transportData, rEarth)
% Computes the distance between all the public transport utilities.
%
% transportData is a 2D cell array where each row represents a mode of
% public transport. The 4th and the 5th column are the latitude and
% longitude, respectively, in degrees, of the public transport utility.
%
% rEarth is the radius of the earth
%
% distances is a 2D numeric array such that Distance(j,k) is the Haversine
% distance between the public transport utility corresponding to rows j
% and k of transportData.  Distance is reported in the same units as rEarth.
```

For full credit, design and make use of a subfunction. In addition, try to avoid doing any redundant calculations—we're processing a lot of data here.

## 1.6 Answer questions on and make visualizations from the data

Add code to `assignment3.m` to answer the following questions and make the requested visualizations (draw figures). Your code should print a descriptive answer to the Command Window so that it is clear what question is being answered. Include the question number in the screen output. Observe that question 1 has been answered for you in the provided code as an example.

*Make effective use of the functions we have developed thus far.*

1. How many airports are recorded in this data set?
2. Determine the number of countries that have at least one airport recorded in this data set.
3. Consider the top 30 countries with the most number of airports recorded in the data set. Draw a bar graph to show the number of airports recorded per country for those that are in the top 30. You are allowed to use built-in function `sort`. The x-axis should show the country names without overlap when the figure is enlarged sufficiently (e.g., full screen). The example code below will get you started; search the MATLAB documentation for additional details as necessary. In addition, do not forget to label the y-axis and add a title.

   Here's a bar graph example:

   ```
   xdat= 1:4;  ydat=[7 11 9 5];
   bar(xdat, ydat)  % 4 bars with heights specified by ydat
                    % The bars are located at x coordinates xdat.
   ```

   The bar graph in MATLAB uses numeric axes. If you want to have text labels (country names) on the horizontal axis, then you need to add the following code:

   ```
   barNames= {'Name1' 'Name2' 'Name3' 'Name4'};  % cell array containing the bar names
   set(gca, 'XTick', xdat, 'XTickLabel', barNames);
            % set and gca are built-in functions
            % XTick and XTickLabel are built-in property names of the graph
   ```

4. Draw a piechart to illustrate the proportion of airports in each time zone area (i.e., each slice of the "pie" represents the airports in one area). Use MATLAB's documentation to learn how to draw a pie chart. Begin your graphics code with the command `figure` to start a new figure window (so that the previous bar graph is not replaced). Be sure to label the portions of the pie chart appropriately and include a title for the figure.

   Notice that because some time zone areas only have a very small number of airports, their slices are small and the labels in the pie chart for these areas may overlap even if the figure is sufficiently enlarged. To prevent this, search "offset pie slice" in the MATLAB documentation. This gives you the option to emphasize specific slices (e.g., the slices that occupy a very small portion of the pie) and allows more space for showing their labels. For example, try offsetting the the slices that occupy less than 3% of the pie.

5. Determine which pair of airports are closest to each other and which pair are farthest from each other. Print the answers neatly with their corresponding distance and corresponding countries.

   When formatting information like this, ask yourself: would somebody unfamiliar with this assignment be able to understand the text I just printed? Writing complete sentences helps with this.

6. Determine the airport with the largest altitude for each time zone area. Print your results neatly with the corresponding altitude (in ft) and corresponding country. If your solution involves cell array slicing, you may find the built-in function `cell2mat` helpful (look it up in the MATLAB documentation).

## 2 Self-check list

The following is a list of the minimum *necessary* criteria that your assignment must meet in order to be considered *satisfactory*. Failure to satisfy any of these conditions will result in an immediate request to resubmit your assignment. Save yourself and the graders time and effort by going over it before submitting your assignment for the first time.

Note that, although all of these are necessary, meeting all of them might still not be *sufficient* to consider your submission satisfactory. We cannot list everything that could be possibly wrong with any particular assignment!

- Δ Comment your code! Every function must be properly commented, regarding function purpose and input/output arguments.
- Δ Suppress all unnecessary output by placing semicolons (;) appropriately. At the same time, make sure that all output that your program intentionally produces is formatted in a user-friendly way.
- Δ Make sure your functions' names are *exactly* the ones we have specified, *including* case.
- Δ Check that the number and order of input and output arguments for each of the functions matches exactly the specifications we have given.
- Δ Check that the units of any physical quantities (including angles) match specifications, and label the units of such quantities in output.
- Δ Test each one of your functions independently, whenever possible, or write short scripts to test them.
- Δ Check that your scripts do not crash (i.e., end unexpectedly with an error message) or run into infinite loops. Check this by running each script several times in a row. Before each test run, you should type the commands `clear all; close all;` to delete all variables in the workspace and close all figure windows.

## 3 Submission instructions

Upload files `findMatch.m`, `readTransportData.m`, `splitTimeZones.m`, `indexAirports.m`, `tabulateDistances.m` and the script `assignment3.m` to CMS in the submission area corresponding to Assignment 3 before the deadline. Although we will accept an assignment that is up to 24 hours late with a late penalty, *submit your files on time in the first round because you can re-submit at the next deadline (first resubmission) without any penalty*! Submit *all* the required files on time—even if only one file is submitted late the entire assignment will be marked late.