Data Science
A Tutorial In Mapping and Exploratory Data Analysis Using R and Python

1 The Dataset

The data we will look at is the set of domestic US flights taken from RITA[1]. We will follow through the steps to reproduce the figure below; a map of the continental US showing the name of each airport and using the font size to illustrate how busy the airport is. The idea is that you can see at a glance which airports are most important, as in a word cloud.

Figure 1: Map showing all mainland airports, with the label in proportion to the traffic in and out (which are virtually equivalent). This figure is purposefully of high-resolution to allow individual names to resolved when zooming in.

The RITA database gives a very detailed record of all the flights taken within the US. The record goes back many years, but I have extracted the data for January, 2013 into a single CSV (comma separated values) file `flights_jan_2013.csv`. The first important point to note is that this file is ≈87 MB in size. This is not *big data* but it also not small data, importantly this is small enough that the data will fit into memory on most machines (your laptop will likely have at least 500MB of RAM). So you can run a script which reads the whole file, stores the content and performs some analysis. (If the file was 100 times bigger a better option would be a solution using a database so that each individual datapoint (or datum) can be accessed efficiently as it is needed.)

The next step is to take a quick look at the file to see what the data looks like, this is where head comes in (your terminal window will probably look different, I have customised mine. See here[2] how to do this or ask me).

We can clearly see that each row corresponds to a single flight and each column to a statistic or identifier. We can also see that some of the values in the rows are empty. Missing or 'dirty' data is very important! It is much easier to

[1] `http://www.transtats.bts.gov/DL_SelectFields.asp?Table_ID=236&DB_Short_Name=On-Time`
[2] `https://wiki.archlinux.org/index.php/Color_Bash_Prompt`

```
[10007][alex.lenovo: TUTORIAL]$ head flights_jan_2013.csv
"MONTH","DAY_OF_MONTH","DAY_OF_WEEK","FL_DATE","CARRIER","TAIL_NUM","FL_NUM","ORIGIN_AIRPORT_ID","ORIGIN_AIRPORT_SEQ_ID","O
RIGIN","DEST_AIRPORT_ID","DEST_AIRPORT_SEQ_ID","DEST","CRS_DEP_TIME","DEP_TIME","DEP_DELAY","CRS_ARR_TIME","ARR_TIME","ARR_
DELAY","DISTANCE","CARRIER_DELAY","WEATHER_DELAY","NAS_DELAY","SECURITY_DELAY","LATE_AIRCRAFT_DELAY",
1,17,4,2013-01-17,"9E","N923XJ","3324",11298,1129803,"DFW",12478,1247802,"JFK","1045","1038",-7.00,"1505","1451",-14.00,139
1.00,,,,,,
1,18,5,2013-01-18,"9E","N907XJ","3324",11298,1129803,"DFW",12478,1247802,"JFK","1045","1037",-8.00,"1505","1459",-6.00,1391
.00,,,,,,
1,19,6,2013-01-19,"9E","N914XJ","3324",11298,1129803,"DFW",12478,1247802,"JFK","1040","1035",-5.00,"1458","1515",17.00,1391
.00,0.00,0.00,17.00,0.00,0.00,
1,20,7,2013-01-20,"9E","N921XJ","3324",11298,1129803,"DFW",12478,1247802,"JFK","1045","1037",-8.00,"1505","1455",-10.00,139
1.00,,,,,,
1,21,1,2013-01-21,"9E","N937XJ","3324",11298,1129803,"DFW",12478,1247802,"JFK","1045","1044",-1.00,"1505","1446",-19.00,139
1.00,,,,,,
1,22,2,2013-01-22,"9E","N601LR","3324",11298,1129803,"DFW",12478,1247802,"JFK","1045","1054",9.00,"1505","1502",-3.00,1391.
00,,,,,,
1,23,3,2013-01-23,"9E","N917XJ","3324",11298,1129803,"DFW",12478,1247802,"JFK","1045","1036",-9.00,"1505","1504",-1.00,1391
.00,,,,,,
1,24,4,2013-01-24,"9E","N936XJ","3324",11298,1129803,"DFW",12478,1247802,"JFK","1045","1036",-9.00,"1505","1520",15.00,1391
.00,0.00,0.00,15.00,0.00,0.00,
1,25,5,2013-01-25,"9E","N907XJ","3324",11298,1129803,"DFW",12478,1247802,"JFK","1045","1042",-3.00,"1505","1525",20.00,1391
.00,0.00,0.00,20.00,0.00,0.00,
```

get an estimate of the *quality* of your data before you start doing serious analysis than it is to discover later on when your calculations don't make any sense. Ignoring missing data is not an option! It never works!

An easier way to see what is going on is to number the columns, we can do this using head and awk as follows;

```
[10010][alex.lenovo: TUTORIAL]$ head -n 1 flights_jan_2013.csv | awk -F ',' '{for(i=1;i<NF;i++) print i,$i}'
1 "MONTH"
2 "DAY_OF_MONTH"
3 "DAY_OF_WEEK"
4 "FL_DATE"
5 "CARRIER"
6 "TAIL_NUM"
7 "FL_NUM"
8 "ORIGIN_AIRPORT_ID"
9 "ORIGIN_AIRPORT_SEQ_ID"
10 "ORIGIN"
11 "DEST_AIRPORT_ID"
12 "DEST_AIRPORT_SEQ_ID"
13 "DEST"
14 "CRS_DEP_TIME"
15 "DEP_TIME"
16 "DEP_DELAY"
17 "CRS_ARR_TIME"
18 "ARR_TIME"
19 "ARR_DELAY"
20 "DISTANCE"
21 "CARRIER_DELAY"
22 "WEATHER_DELAY"
23 "NAS_DELAY"
24 "SECURITY_DELAY"
25 "LATE_AIRCRAFT_DELAY"
```

What this command does is to take the first line of the file i.e. the header with `head -n 1 flights_jan_2012.csv` and then pipes this line into awk. First we tell awk that the fields are separated with a comma (`-F ','`), then we cycle through each field and print the field and its number. `NF` stands for number of fields. Notice that `i` starts from 1 not 0 for `awk`. We have a lot of data available covering the date of each flight, the source and destination, the carrier (which airline) the arrival and departure time as well as any delay and the cause of the delay i.e. weather, security etc.

As an exercise, find out how many different airports had flights leaving from them in January.

```
[10011][alex.lenovo: TUTORIAL]$ awk -F ',' '{print $13}' flights_jan_2013.csv | sort | uniq | wc
    307     307    1843
```

This tells us that there were 307 different airports used in January 2013.

In order to make our map, we need to count how many flights departed each airport, in other words we need to count how many times each airport appears in the list you just created. Thanksfully there is a very useful option in `uniq` which we can use to count the number of appearances of each airport; `-c` for **C**ount.

```
[10014][alex.lenovo: TUTORIAL]$ awk -F ',' '{print $13}' flights_jan_2013.csv | sort | uniq -c | sort -n | tail
  11437 "CLT"
  12819 "DTW"
  13214 "SFO"
  14685 "PHX"
  15004 "IAH"
  17553 "LAX"
  17809 "DEN"
  23477 "DFW"
  23878 "ORD"
  32355 "ATL"
```

This prints all the airports, the number of flights out of each in January and sorts them. We can see that the airports are referred to by their 3 letter code which isn't ideal and that the busiest airport is ATL, Atlanta Georgia and the home of Delta Airways. We will put these values into a file `airport_departures.txt` and we will use it later to make our map. If you are very observant, you will notice that we have piped the output of `sort -n` through `awk '{print $1,$2}'`. The effect of this awk command is to strip out any white space; the first column of the output of `sort -n` is wide enough to contain the largest value it must print, so for small numbers it will print a number of spaces to make the column of a fixed width.

You will also see another file `airports.csv` which contains the full name of each airport, coordinates in space along with its 3 letter code.

```
[10060][alex.lenovo: TUTORIAL]$ awk -F ',' '{print $13}' flights_jan_2013.csv | sort | uniq -c | sort -n | awk '{print $1,$
2}' > airport_departures.txt
```

2 Maps in R

There are several mapping libraries in R and an active community of bloggers (James Cheshire of UCL has a very active blog[3]). As a result there are several different ways to do anything, and I will describe just one.

We will use the maps package along with some other tools. Let's begin by loading the package and drawing a map of the US.

```
> require(maps)
Loading required package: maps
> map('state')
```

This produces the simple map shown below.

So far so good. Now let us read in the data on the airports and put a marker on each airport.

`read.csv` reads a file into a dataframe. It helps to think of a dataframe like a spreadsheet; there are columns and rows. The columns might already have names in the file, in this case we set `header=T` and we can test to see that the columns have the correct names using `names(airports)`. Each row represents an airport and for each one we have the IATA code, the airport name, the city, state, country (redundant in this case) and the coordinates. `read.csv` is smart and normally guesses if a column is float, integer etc.

We can also check the size of this data frame using `dim` and we see that there are 3376 airports in this file, many more than there are in the RITA data. This is likely because some of the airports in `airports` are private or remote airports for which no data is available.

If we want to refer to all the airport codes or the latitude of all the airports as a single list we use the name of the data frame followed by the dollar sign and the name of the column (in R documentation these are referred to as factors). So `airports$lat` will give us the latitude of each airport in order.

Finally we can use a function in R called `head` which behaves exactly like its command line equivalent or we can refer to the same range of rows with `airports[1:10,]`.

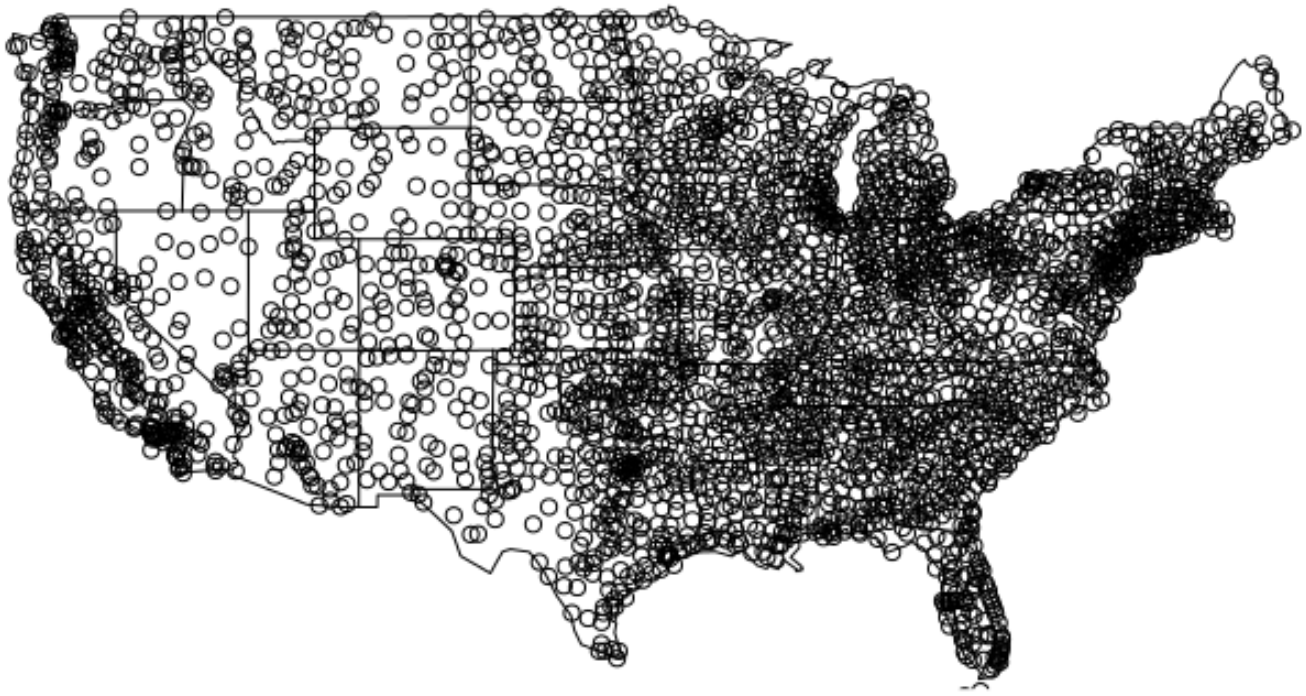
[3] http://spatial.ly/category/r-spatial-data-hints/

```
> airports=read.csv('airports.csv',sep=',',header=T)
> names(airports)
[1] "iata"    "airport" "city"    "state"   "country" "lat"     "long"
> dim(airports)
[1] 3376    7
> head(airports)
  iata             airport            city state country      lat       long
1  00M            Thigpen       Bay Springs    MS     USA 31.95376  -89.23450
2  00R Livingston Municipal       Livingston    TX     USA 30.68586  -95.01793
3  00V          Meadow Lake Colorado Springs    CO     USA 38.94575 -104.56989
4  01G        Perry-Warsaw             Perry    NY     USA 42.74135  -78.05208
5  01J    Hilliard Airpark         Hilliard    FL     USA 30.68801  -81.90594
6  01M   Tishomingo County          Belmont    MS     USA 34.49167  -88.20111
```

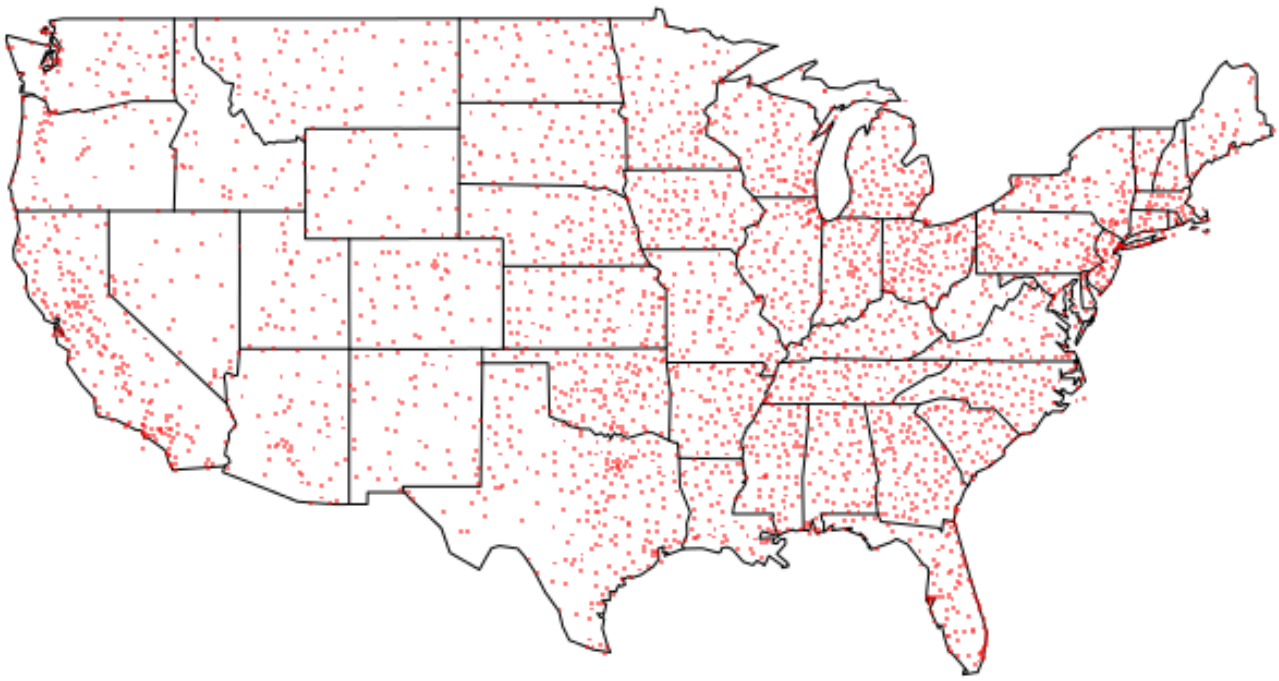
```
> airports[1:10,]
   iata             airport            city state country      lat       long
1   00M            Thigpen       Bay Springs    MS     USA 31.95376  -89.23450
2   00R Livingston Municipal       Livingston    TX     USA 30.68586  -95.01793
3   00V          Meadow Lake Colorado Springs    CO     USA 38.94575 -104.56989
4   01G        Perry-Warsaw             Perry    NY     USA 42.74135  -78.05208
5   01J    Hilliard Airpark         Hilliard    FL     USA 30.68801  -81.90594
6   01M   Tishomingo County          Belmont    MS     USA 34.49167  -88.20111
7   02A          Gragg-Wade           Clanton    AL     USA 32.85049  -86.61145
8   02C             Capitol       Brookfield    WI     USA 43.08751  -88.17787
9   02G   Columbiana County   East Liverpool    OH     USA 40.67331  -80.64141
10  03D    Memphis Memorial          Memphis    MO     USA 40.44726  -92.22696
```

Dataframes are the main data structure in R and can be a bit confusing at first. But they are very powerful and once you start to think of them as just like Excel spreadsheets they are easy.

Now we can put markers on the map for each airport. `points` takes a list of x-coordinates and a list of y-coordinates and puts a marker on each; it can be used for making a scatter plot.



The default markers are a bit ugly, so let's specify a dot as the marker (`pch='.'`), make the dots a bit bigger (`cex=2`) and finally make them a semi-transparent orange colour (`col=rgb(1.0,0.0,0.0,0.5)`).



3 Data Wrangling

Now that we can plot airports, what we want to do is read in the airport traffic from our RITA data, the airport names and coordinates and *only* plot the airports which appear in both files. If we read the 2 files we will have 2 different dataframes which we want to join together. We already have the full list of airports and so we now want the count of departures from the RITA data

```
> departures=read.csv('TUTORIAL/airport_departures.txt',header=F,sep=' ')
> head(departures)
  V1   V2
1  1 DEST
2  9  ADK
3  9  PPG
4 10  RFD
5 12  TTN
6 13  SHD
> tail(departures)
       V1  V2
302 15004 IAH
303 17553 LAX
304 17809 DEN
305 23477 DFW
306 23878 ORD
307 32355 ATL
```

Looking at the top of the file we see a problem, there is apparently an airport called 'DEST'! This is clearly an error from when we parsed the header of the `flights_jan_2012.csv` file, but in this case we can simply ignore it. The end of the file shows the busiest airports as we expect. Since we specified that there is no header, the columns have been relabelled automatically to V1 and V2. So we should give them more useful names that match our other dataframe.

```
> names(departures)=c('traffic','iata')
```

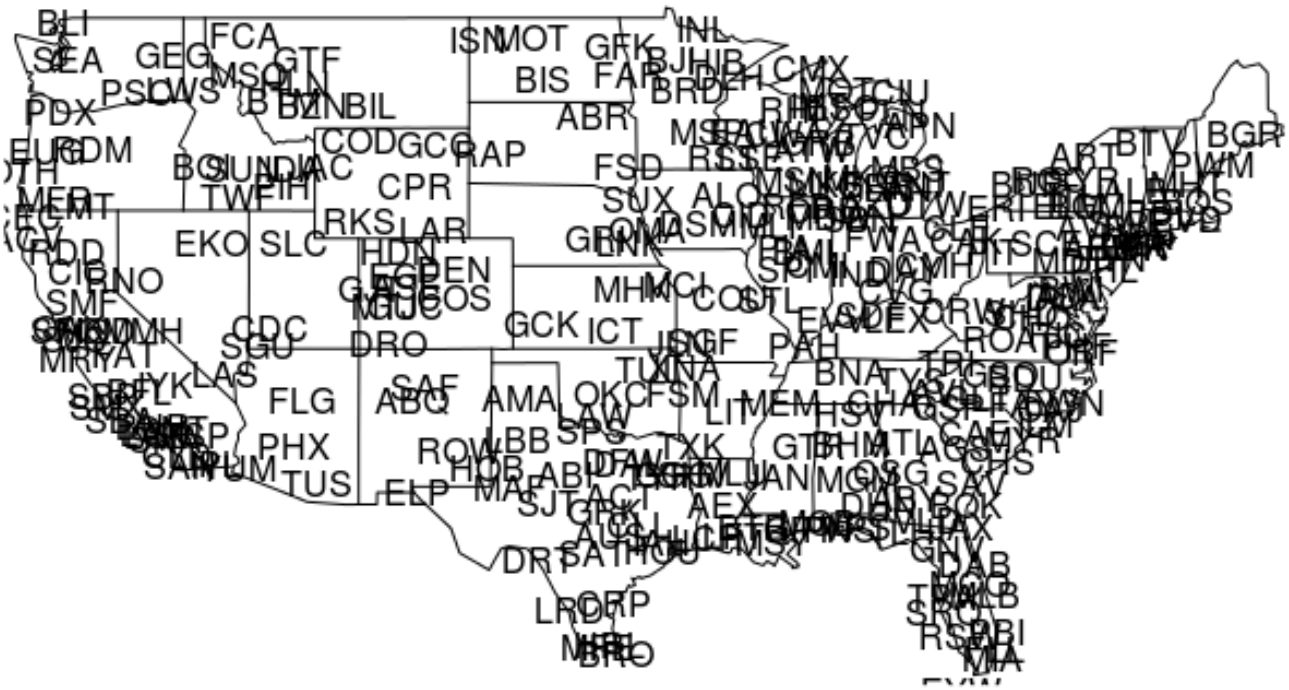
Now we use the `merge` function to combine our data together; the function looks in the column marked 'iata' in both `departures` and `airports` dataframes and when they match, the columns are merged for that row. If you are familiar with an SQL join, the idea is the same. After we do this we see that we have 303 matched airports and for each one; the traffc, the name, city and state along with coordinates. It is very helpful to have all our data in a single dataframe to do the plotting as we will do now.

```
> allData=merge(departures,airports,by='iata')
> dim(allData)
[1] 303   8
> head(allData)
  iata traffic                    airport        city state country      lat
1  ABE     299 Lehigh Valley International   Allentown    PA     USA 40.65236
2  ABI     213            Abilene Regional     Abilene    TX     USA 32.41132
3  ABQ    2074   Albuquerque International Albuquerque    NM     USA 35.04022
4  ABR      63          Aberdeen Regional    Aberdeen    SD     USA 45.44906
5  ABY      87  Southwest Georgia Regional      Albany    GA     USA 31.53552
6  ACT     120              Waco Regional        Waco    TX     USA 31.61129
       long
1  -75.44040
2  -99.68190
3 -106.60919
4  -98.42183
5  -84.19447
6  -97.23052
```

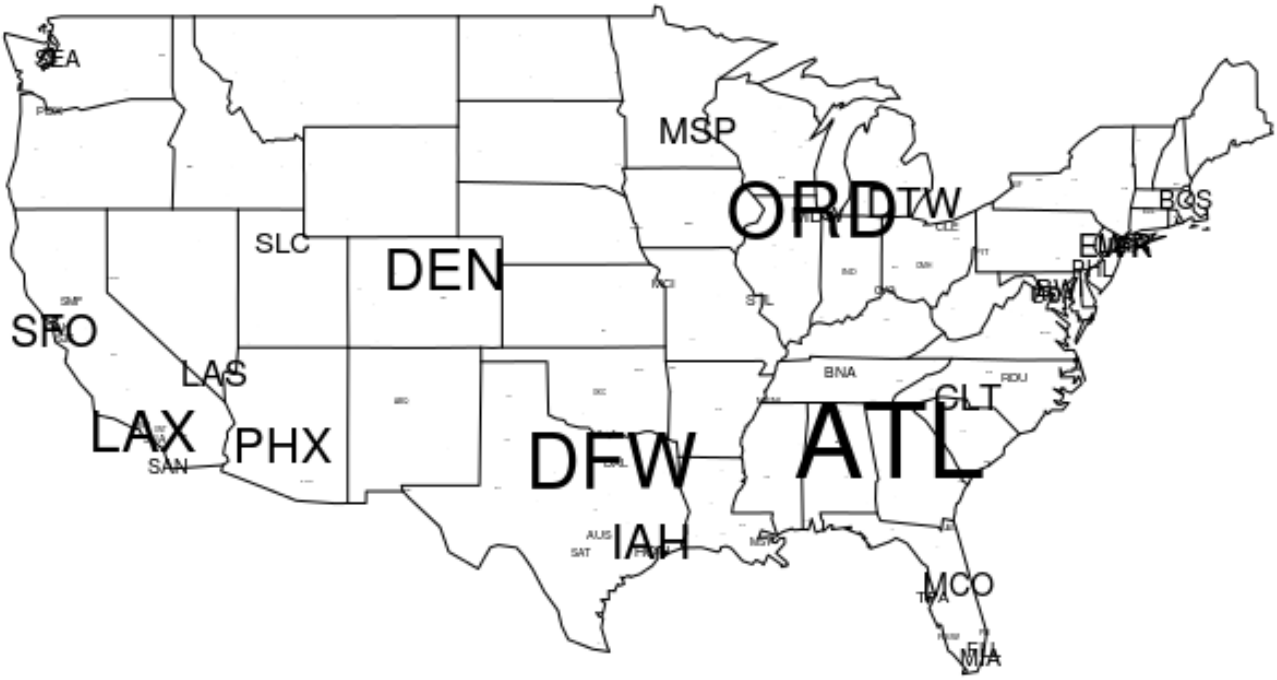
So far so good, the map is starting to take shape! Now we want the busiest airport to have the largest text, so we should pass the `text` function a list of numbers. These numbers should ideally be normalised so that 1 is the busiest and then we can choose the scale that makes the labels big enough to be legible but small enough that they don't take over the figure. I chose a scale of 5 simply by trial and error.

Currently our map looks a little bit plain, so taking our inspiration from the great maps used in the New York times we will try to improve the presentation a bit.

```
> map('state')
> text(allData$long,allData$lat,allData$iata)
```

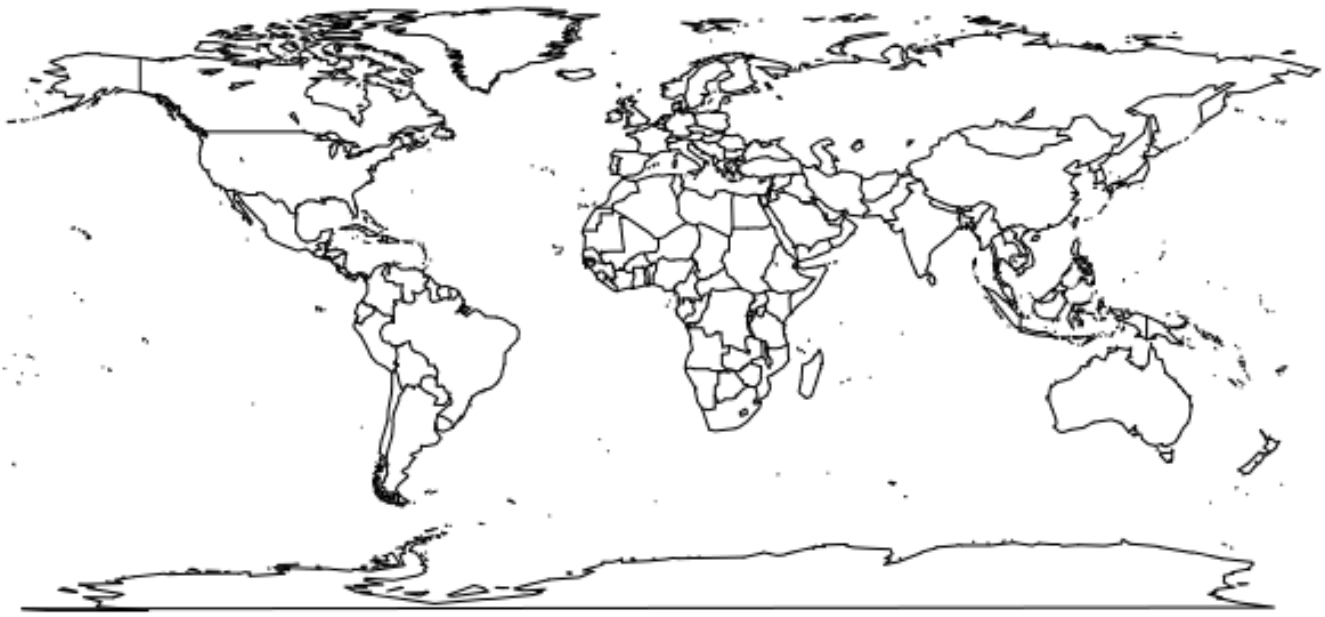


```
> map('state')
> text(allData$long,allData$lat,allData$iata,cex=3*allData$traffic/max(allData$traffic))
```



4 Polishing

We can use the maps package to plot the whole world; `map('world')`.



This returns the entire world, and really we would like to focus on North America so we define a bounding box (this was chosen by trial and error)

In order to make the map cleaner, we do not draw a line around the boundary (`lty=0`) and we fill the land mass with gray (`fill=T,col='gray'`).

As a final touch, it would be nice to show both the country boundaries and the state boundaries. We can do this with these 2 lines, you should play around with them to understand what is happening here.

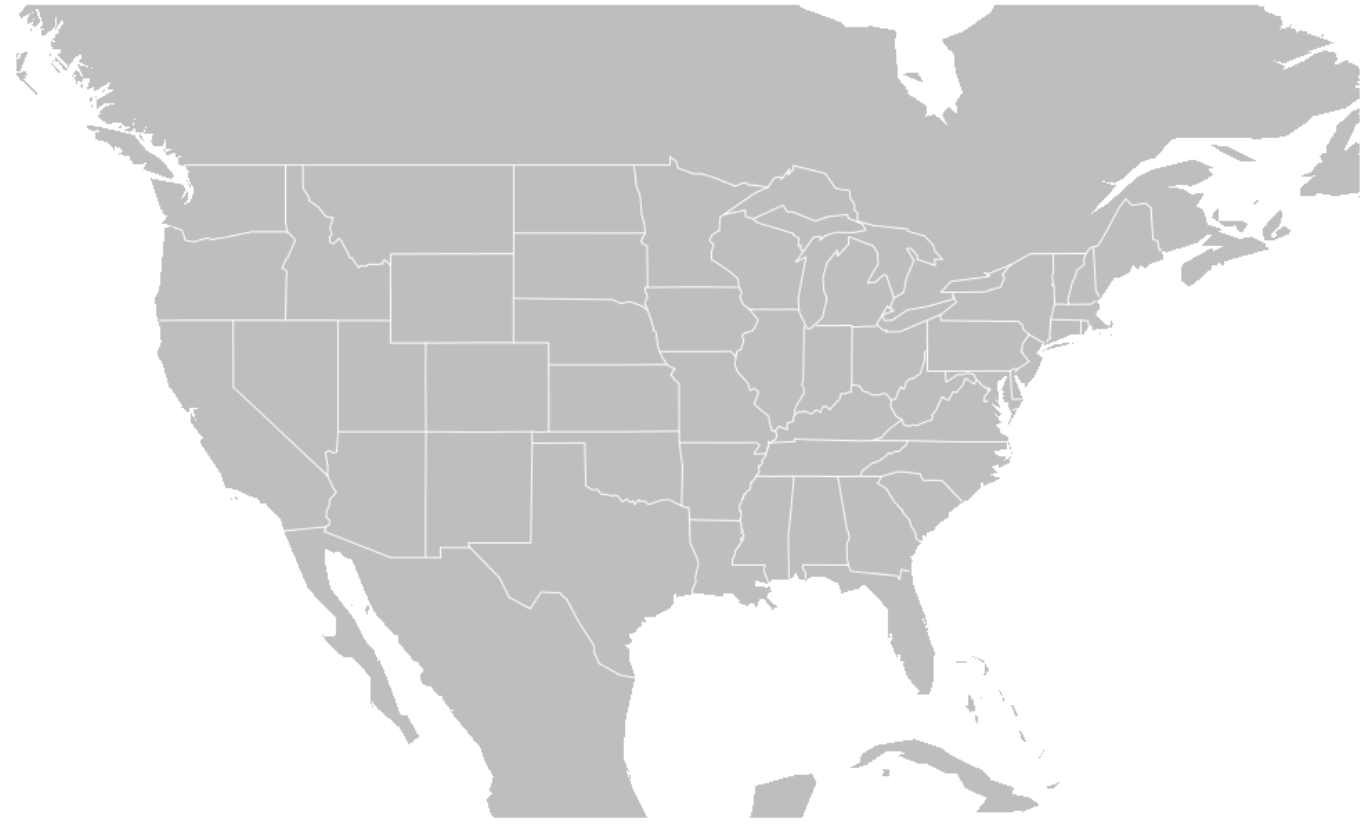
The last thing to do is to overlay the city each airport is at (rather than the IATA code), as text labels. A naive first attempt shows the map getting unclear

Therefore we use an alternative function `pointLabel` from the maptools package, this plots text labels as before but adjusts slightly to minimise overlap of the labels and adjust the scale of the text size.

```
> xlim <- c(-131.738281, -56.601563)
> ylim <- c(20.039321, 55.856229)
> map('world',interior=F,xlim=xlim,ylim=ylim)
```

```
> map('world',interior=F,xlim=xlim,ylim=ylim,fill=T,col='gray',lty=0)
```

```
> map('world',interior=F,xlim=xlim,ylim=ylim,fill=T,col='gray',lty=0)
> map('world',boundary=F,interior=T,col='white',fill=F,xlim=xlim,ylim=ylim,add=T)
> map('state',boundary=F,col='white',xlim=xlim,ylim=ylim,add=T)
```



```
> require(maptools)
> pointLabel(allData$long,allData$lat,labels=allData$city,cex=3*allData$traffic/max(allData$traffic))
```

5 Final Script

```
require(maps)
require(maptools)
airports=read.csv('airports.csv',sep=',',header=T)
names(airports)=c('iata','name','city','state','country','lat','long')
xlim ¡- c(-131.738281, -56.601563)
ylim ¡- c(20.039321, 55.856229)
departures=read.csv('airport_departures.txt',header=F,sep=" ")
names(departures)=c('traffic','iata')
allData=merge(departures,airports,by='iata')
map('world',interior=F,xlim=xlim,ylim=ylim,fill=T,col='gray',lty=0)
map('world',boundary=F,interior=T,xlim=xlim,ylim=ylim,fill=F,col='white',add=T)
map('state',boundary=F,col='white',xlim=xlim,ylim=ylim,add=T)
pointLabel(allData$long,allData$lat,labels=allData$city,cex=3*allData$traffic/max(allData$traffic))
```

6 Further Reading and Extensions

Bradford Cross (a legendary Silicon Valley hacker) set up a flight delay prediction systems that you can read about here[4].

Many nice visualisations have been made of flight routes, here is a nice tutorial from the New York Times data artist Nathan Yau[5]

Another paper noted how flight traffic between cities can predict the amount of communication flow on Twitter between the same cities[6]

There are many interesting angles to analyse this data and probably a few good research projects; how do delays spread through the network? How do *international* flights affect the airport network? What are the main causes of delays?

[4] http://www.datawrangling.com/how-flightcaster-squeezes-predictions-from-flight-data
[5] http://flowingdata.com/2011/05/11/how-to-map-connections-with-great-circles/
[6] http://www.sciencedirect.com/science/article/pii/S0378873311000359