Recitation 1: R Studio, Data Frames, and Flow Control

Statistical Computing, 36-350

Monday September 21, 2015

R Studio

In R Studio there are (by default) four "panes" to work in.

- Upper Left: Source Pane
- Lower Left: Console
- Upper Right: "Workspace"
- Lower Right: "Plots/Help"

These can all be configured, but we'll stick with the default configuration for now.

Source Pane

The Source Pane provides several helpful features:

- Highlighting
- Line Numbering
- Indentation and Parentheses

Console Pane

The console "is" R. You feed commands into it, and it produces output. It also keeps track of variables, functions, etc. that you define, and the previous commands you've given.

The console pane is where you directly interact with the console. You can do this by typing commands in directly, or by running them from the source pane.

The output from the console is often given in the console. Plots appear separately (by default in the "Plots/Help" pane).

Workspace

You can view a few different things in the upper right. By default you'll see the Environment, where R shows you what things it's keeping track of (Data frames, variables, functions). Many of these things, e.g. data frames and functions, can be opened in the source pane for viewing.

You can also view the commands the console has run, in the order it ran them.

Plots/Help

Plots is self-explanatory. Note that you can flip back and forth between various plots you've made.

The Help browser allows you to search for documentation on functions you are using, or might use. You can also access these from the console. Great when you know roughly what you want to do, but need to check syntax. Also, check the "See Also" at the bottom if you're not sure what function you're looking for.

Data Frames

A data frame is a fancy list. Let's load the data frame from the lecture:

```
library(datasets)
states = data.frame(state.x77, Abbr=state.abb,
                    Region=state.region, Division=state.division)
```

```
class(states)
```

```
## [1] "data.frame"
```

```
typeof(states)
```

```
## [1] "list"
```

Basic Access: Row

```
states['Alabama',]
```

```
##         Population Income Illiteracy Life.Exp Murder HS.Grad Frost  Area
## Alabama       3615   3624        2.1    69.05   15.1    41.3    20 50708
##         Abbr Region           Division
## Alabama   AL  South East South Central
```

Basic Access: Column

```
head( states['Population'] ,10)
```

```
##             Population
## Alabama           3615
## Alaska             365
## Arizona           2212
## Arkansas          2110
## California       21198
## Colorado          2541
## Connecticut       3100
## Delaware           579
## Florida           8277
## Georgia           4931
```

Also `states$Population`

Basic Access: Single Entry

```
states['Alabama','Population']
```

```
## [1] 3615
```

Fancy Access: Slice

"Slicing" a data frame returns a new data frame with the desired rows and/or columns

```
states[1:4,c('Population','Income')]
```

```
##          Population Income
## Alabama        3615   3624
## Alaska          365   6315
## Arizona        2212   4530
## Arkansas       2110   3378
```

Fancy Access: Conditions

Let's look at the big states:

```
states[states$Population>10000,c('Area','Region')]
```

```
##                Area        Region
## California   156361          West
## Illinois      55748 North Central
## New York      47831     Northeast
## Ohio          40975 North Central
## Pennsylvania  44966     Northeast
## Texas        262134         South
```

Can also save this condition in a vector

```
bigstates = states$Population>10000
states[bigstates,'Abbr']
```

```
## [1] CA IL NY OH PA TX
## 50 Levels: AK AL AR AZ CA CO CT DE FL GA HI IA ID IL IN KS KY LA MA ... WY
```

Flow Control: For Loops

Generate the first 10 elements of the Fibonacci sequence

```
fibonacci = c(1,1)
for (i in 3:10){
  fibonacci[i] = fibonacci[i-1]+fibonacci[i-2]
}
fibonacci
```

```
##  [1]  1  1  2  3  5  8 13 21 34 55
```

Flow Control: While Loops

A random walk

```
position = 0
total_steps = 0
while( abs(position) < 10){
  my.rand = runif(1)
  if (my.rand>0.5){
    step = 1
  } else {
    step = -1
  }
  position = position + step
  total_steps = total_steps + 1
}
```

```
total_steps
```

```
## [1] 18
```

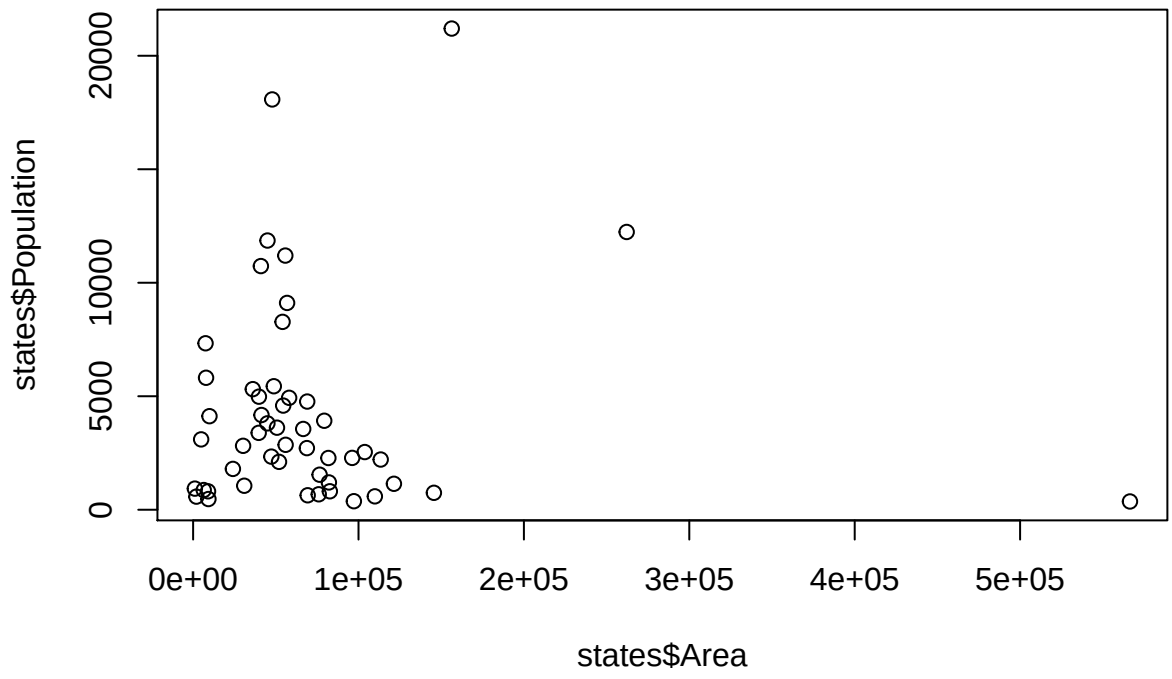
```
position
```

```
## [1] 10
```

Bonus: Plotting

Basic Plotting: provide x and y data:

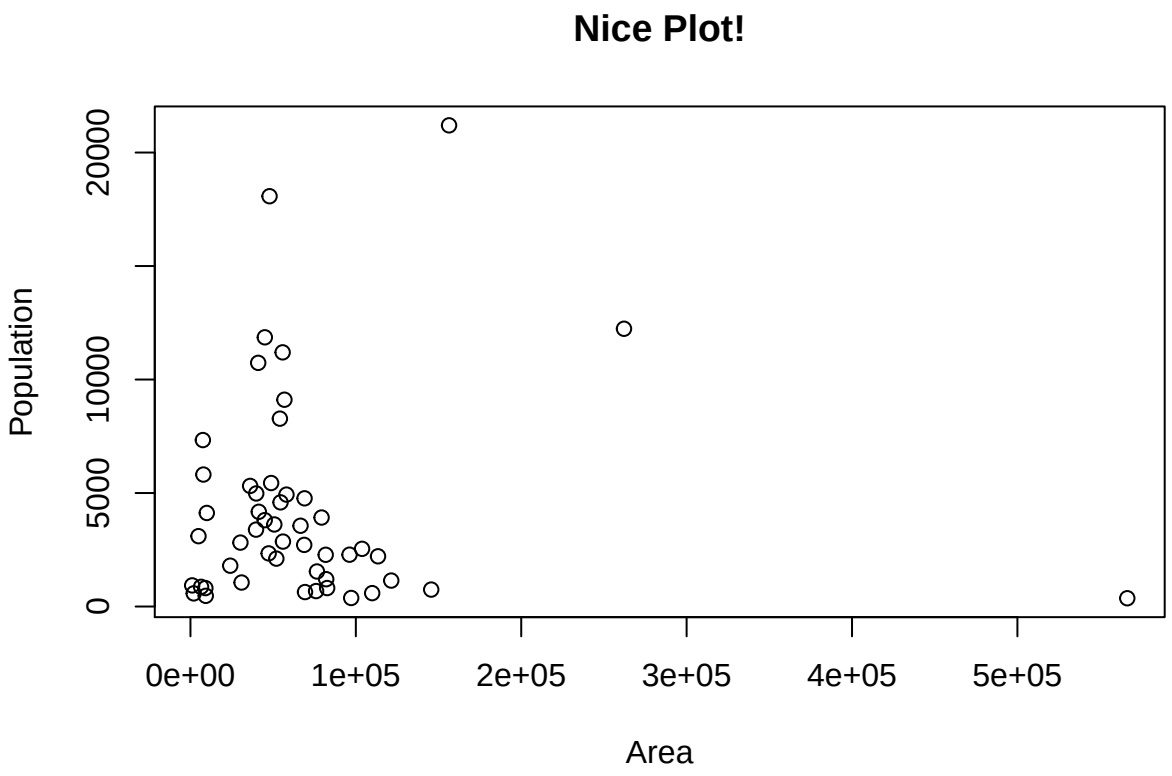
```
plot(states$Area,states$Population)
```



Bonus: Pretty Plots

We can pass additional parameters to `plot` that tell it what the plot should look like.

```
plot(states$Area,states$Population,xlab='Area',ylab='Population',main='Nice Plot!')
```



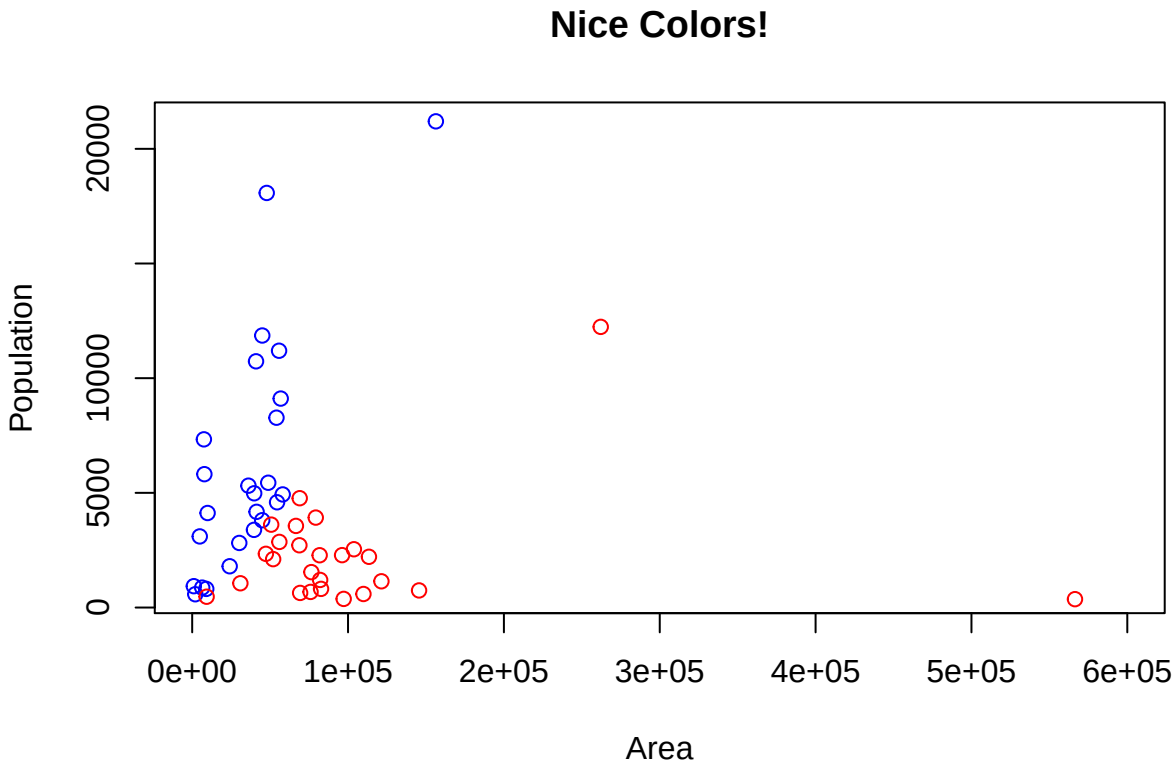
Bonus: More Points on One Plot

Let's split the states up by population density

```
popdensity = states$Population/states$Area
dense  = (popdensity>median(popdensity))
sparse = (popdensity<=median(popdensity))
```

Then make a plot with both

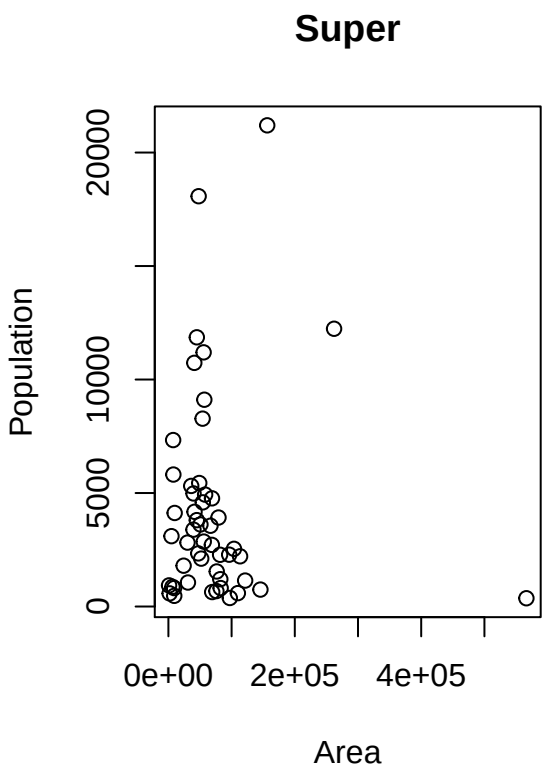
```
plot(states$Area[dense],states$Population[dense],col='blue',xlab='Area',ylab='Population',main='Nice Co
points(states$Area[sparse],states$Population[sparse],col='red',xlim=c(0,6e5))
```

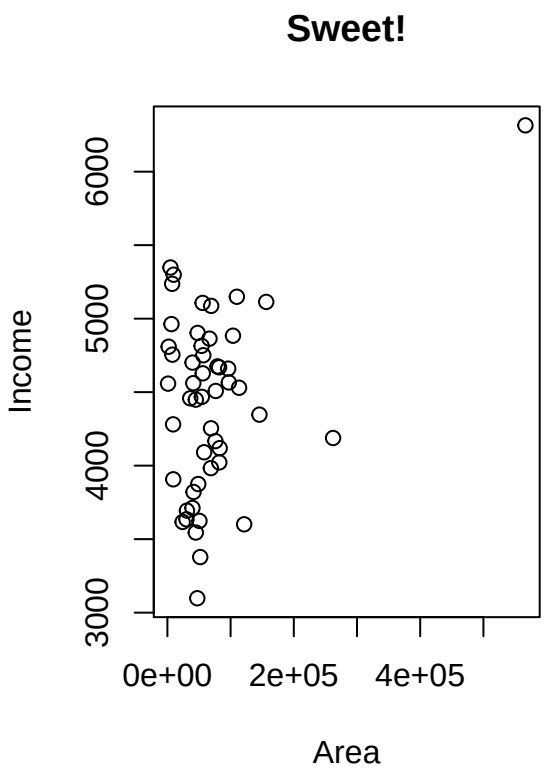


Bonus: More Plots

You can produce side-by-side plots by setting a global parameter

```
par(mfrow=c(1,2))
plot(states$Area,states$Population,xlab='Area',ylab='Population',main='Super')
plot(states$Area,states$Income,xlab='Area',ylab='Income',main='Sweet!')
```


Super
Population
0
5000
10000
20000
0e+00
2e+05
4e+05
Area


Sweet!
Income
3000
4000
5000
6000
0e+00
2e+05
4e+05
Area