Today's section will quickly debrief Problem Set 2 and introduce spatial data analysis in `R`.

Problem Set Debrief

- Many people didn't realize that the correlation coefficient between $\ln(Q)$ and the residuals is zero by construction in OLS.
- J = 8 in the F-test as each extra group coefficient must be set to zero individually in the restricted model. 4 more groups than in Q2 so 4 slopes + 4 intercepts = 8.
- Some people calculated the F-test the hard way. See the solutions.
- You can restrict the number of digits and add commas to numbers in knitr using the `prettyNum()` function.
- Lack of comments on your submissions indicates that things look mostly correct.

Spatial Data Analysis

`R` is the first (and mostly only) language with which I have performed spatial data analysis. It's not a nice, coherent, well formed, piece of software for this purpose. Of the packages in my package install script (which you should all have for when you need to install/reinstall `R`), I counted 18 items with functions related to spatial analysis or mapping. It's a mess. Before the `sp` package, apparently it used to be worse! When I'm a tenured professor who has run out of ideas, or an unemployed bum living at my mother's house, I plan to start a project that collects the best of all these functions into a single package with a text book. In the meantime, the status quo will have to do. A decent resource is Applied Spatial Data Analysis with `R` (second edition) by Bivand, Pebesma, and Gómez-Rubio.

Why should I use `R` for spatial analysis then? Two reasons: 1) the value of keeping your work in one program for ease of reproducibility and 2) active development and voluntary support. People are keenly working on developing `R`'s spatial capabilities and people answer questions. The R-sig-geo mailing list is a good resource, as is stackoverflow.

Spatial Data Types

The first step in understanding spatial analysis in `R` is understanding the various data types. This is going to be quick. For each of these classes, there is a corresponding class that adds a `data.frame` with a row for each element of the class, so that data on each element can be nicely kept with the spatial object. For `SpatialPoints`, for example, the corresponding object that also contains a `data.frame` is the `SpatialPointsDataFrame`.

SpatialPoints

A `SpatialPoints` object is a collection of points represented by coordinates, usually latitude/longitude pairs. Each element is a single point.

SpatialLines

A `SpatialLines` object is a collection of lines represented by collections of coordinates in an order. Each element is a collection of points, ordered to represent the line.

SpatialPolygons

A `SpatialPolygons` object is simply a collection of closed lines. Individual "polygons" can have holes, and comprise more than one actual `Polygon` object. Mostly, you won't have to concern yourself with the fine details of how these objects work.

SpatialGrid/SpatialPixels

`SpatialGrid/SpatialPixels` are basically the same object and represent a something close to a rectangular grid. These are used to represent, for example, satellite imagery and gridded weather.

Reading spatial data

There are many different file types that represent spatial data and I have not yet come across a file type that can't be read, though finding a method that works is not always easy. The GDAL driver is pretty flexible and can handle a lot of formats. We'll look at a couple:

ESRI Shapefiles

Being a popular format, this one is particularly easy to get into `R`. Let's download a USA states shapefile and read it in. You will need to install the `sp` and `rgdal` packages if you haven't already.

```
library(sp)
library(rgdal)
```

```
## Loading required package: methods
## rgdal: version: 0.9-1, (SVN revision 518)
## Geospatial Data Abstraction Library extensions to R successfully loaded
## Loaded GDAL runtime: GDAL 1.11.0, released 2014/04/16
## Path to GDAL shared files: C:/Users/Kenny/R/win-library/3.1/rgdal/gdal
## GDAL does not use iconv for recoding strings.
## Loaded PROJ.4 runtime: Rel. 4.8.0, 6 March 2012, [PJ_VERSION: 480]
## Path to PROJ.4 shared files: C:/Users/Kenny/R/win-library/3.1/rgdal/proj
```

```
# Download a USA shapefile
tmpFile <- tempfile()
download.file("http://www2.census.gov/geo/tiger/GENZ2013/cb_2013_us_state_20m.zip",
              destfile = tmpFile)
unzipped <- unzip(zipfile = tmpFile, exdir = getwd())

# This is the actual reading stage
```

```
states <- readOGR(dsn = getwd(), layer = "cb_2013_us_state_20m")
```

```
## OGR data source with driver: ESRI Shapefile
## Source: "C:/Users/Kenny/Dropbox/PhD/ARE212GSI/Github/ARE212/section-11", layer: "cb_2013_us_s
## with 52 features and 9 fields
## Feature type: wkbPolygon with 2 dimensions
```

```
# Remove all the files on disk
file.remove(tmpFile)
```

```
## [1] TRUE
```

```
file.remove(unzipped)
```

```
## [1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE
```

Now let's take a quick look at what we've downloaded!

```
plot(states)
```

Could look better - but we'll worry about that more next time. We can certainly confirm that we've downloaded a `SpatialPolygons` object for the USA.

To look closer at what we have, lets check out the slots[1] that belong to the `SpatialPolygonsDataFrame` class.

```
slotNames(states)
```

```
## [1] "data"        "polygons"    "plotOrder"   "bbox"        "proj4string"
```

The `data` slot simply contains a `data.frame` with a row for each `Polygons` object, here one for each

[1] Think of slots as attributes for a given class that are defined for all objects that belong to that class.

state. The `polygons` slot contains a list of `Polygons` objects. You may interact with the rest at some stage, but I wouldn't worry about them for now. Let's look at the `data.frame` part.

```
head(states@data)
```

```
##   STATEFP  STATENS    AFFGEOID GEOID STUSPS        NAME LSAD        ALAND
## 0      01 01779775 0400000US01    01     AL     Alabama   00 131172434095
## 1      05 00068085 0400000US05    05     AR    Arkansas   00 134772954601
## 2      06 01779778 0400000US06    06     CA  California   00 403482685922
## 3      09 01779780 0400000US09    09     CT Connecticut   00  12541965607
## 4      12 00294478 0400000US12    12     FL     Florida   00 138897453172
## 5      13 01705317 0400000US13    13     GA     Georgia   00 148962779995
##        AWATER
## 0  4594920201
## 1  2958815561
## 2 20484304865
## 3  1815409624
## 4 31413676956
## 5  4947803555
```

.bil pixel files

Now, let's download some temperature data as a `SpatialGridDataFrame`. I'll download the maximum temperature for the contiguous USA for my birthday last year. You can see that now we use the `readGDAL()` function, rather than the `readOGR()` function. Google is your friend for working out when to use which reading function to use to get the data in. Always favor solutions that use the `rgdal` package, as these functions are able to read the map projections, while others do not (for example, the `maptools` package).

```
tmpFile <- tempfile()

url <- paste0("http://www.prism.oregonstate.edu/fetchData.php?type",
              "=bil&kind=recent&elem=tmax&range=daily&temporal=",
              "20140125")

# Download the file to fileName
download.file(url = url, mode = "wb", destfile = tmpFile)

# unzip the file to the working directory.
unzippedFiles <- unzip(zipfile = tmpFile, overwrite = TRUE)

# read into R
tMaxGrid <- readGDAL(fname = unzippedFiles[1])
```

```
## ./PRISM_tmax_stable_4kmD1_20140125_bil.bil has GDAL driver EHdr
## and has 621 rows and 1405 columns
```

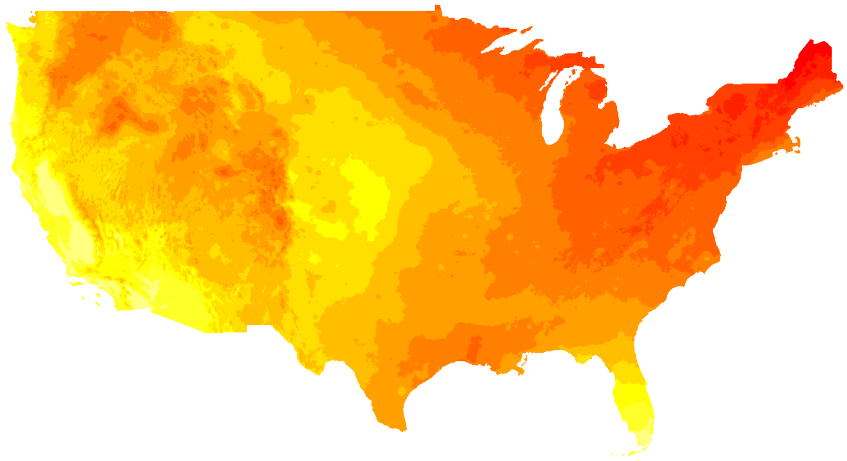
```
# delete the file
file.remove(tmpFile)
```

```
## [1] TRUE
```

```
file.remove(unzippedFiles)
```

```
## [1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
```

```
image(tMaxGrid)
```



Great! This seems to have worked.

Google maps distances

Now, let's use R's interface with Google maps to get some distances.

```
library(ggmap)
distances <- mapdist(from = "Giannini Hall, UC Berkeley, CA 94720",
                     to = as.character(states@data$NAME),
                     mode = "driving") #$
states$distanceToGiannini <- distances$km
```

Spatial Data Analysis, finally

Now, let's quickly do something silly and see if there is a significant correlation between driving distance to Giannini and the temperature grid you downloaded.

```
# This gets the simple average of the grid cells whose centres
# are in each state. NOT a good solution for smaller polygons
# like ZIPs. Stored in a SpatialPolygonsDataFrame.
tMax <- aggregate(tMaxGrid, by = states, FUN = mean, na.rm=TRUE)

# Access the temperature and store it in the states object.
states$tMax <- tMax@data$band1
```

Now, we're ready to run a regression!

```
summary(lm(distanceToGiannini ~ tMax, data = states@data))
```

```
##
## Call:
## lm(formula = distanceToGiannini ~ tMax, data = states@data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1645.48  -446.80   -31.31   358.87  2967.96
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  3552.67     112.97  31.448  < 2e-16 ***
## tMax         -148.09      15.91  -9.307 3.09e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 780.2 on 47 degrees of freedom
##   (3 observations deleted due to missingness)
## Multiple R-squared:  0.6482,Adjusted R-squared:  0.6408
## F-statistic: 86.61 on 1 and 47 DF,  p-value: 3.09e-12
```

Obviously, there is a lot more to spatial analysis in R than calculating attribute values and running regressions. We can calculate spatial correllograms, calculate network statistics, all sorts. Mostly you will discover these things along the way - I'm always available for conversations about these things to get you on the right track.