POL 345: Quantitative Analysis and Politics

Precept Handout 2

Week 3 (Verzani Chapter 2: 2.1-2.3)

Remember to complete the entire handout and submit the precept questions to the Blackboard DropBox 24 hours before precept. In this handout, we cover the following new materials:

- Using logical operators: `<`, `<=`, `>`, `>=`, `==`, `!=`, `&`, and `|`
- Subsetting data with `[]` and `subset()` using logical expressions
- Using `ifelse()` for conditional statements
- More functions for summary statistics: `var()` (variance), `sd()` (standard deviation), `weighted.mean()`, `quantile(X, P)`, and `IQR()` (Inter-quartile Range)
- Applying functions by indexes using `tapply()`
- Common arguments for graphs: `main` (main title), `xlab` and `ylab` (axis labels), `xlim` and `ylim` (axis limits), `pch` (point symbol), `lty` (line type), `lwd` (line width), and `col` (color)
- Adding features to graphs with `lines()` and `abline()` (lines), `points()` (points), and `text()` (text)
- Using `hist()` to generate histograms.
- Calculating a smooth density via `density()`
- Adding a legend to an existing graph by `legend()`
- Printing and saving graphs

1 Logical Operators and Values

- Logical operators (`<`, `<=`, `>`, `>=`, `==` and `!=`) allow for data manipulation and subsetting by determining whether a specified condition is `TRUE` or `FALSE`, both of which must be uppercased and are special values in **R** just like `NA`. The operators correspond to standard use. For instance, `<=` evaluates whether a number is greater than or equal to a specified value. The symbol `!=` corresponds to "not equal". The output of a logical statement is of the class `logical`.

```
> 4 > 3

[1] TRUE

> "Hello" == "hello"

[1] FALSE

> y <- 4 > 3
> y

[1] TRUE

> class(y)

[1] "logical"
```

- Logical operators may be applied to individual data entries or entire vectors (or even a dataframe!). When applied to a vector, logical operators evaluate each element of the vector.

```
> x <- c(3, 2, 1, -2, -1)
> x >= 2

[1]  TRUE  TRUE FALSE FALSE FALSE

> x != 1

[1]  TRUE  TRUE FALSE  TRUE  TRUE
```

- Combine logical statements and operations with `&` (and) and `|` (or).

```
> x > 2 | x <= -1

[1]  TRUE FALSE FALSE  TRUE  TRUE

> x > 0 & x <= 2

[1] FALSE  TRUE  TRUE FALSE FALSE
```

- We can add up the number of `TRUE` statements using `sum()`. That is, `TRUE` and `FALSE` are equal to 1 and 0, respectively, when forced to be numeric.

```
> sum(x > 0 & x <= 2)  #Adds up the number of TRUE statements

[1] 2
```

2 Subsetting with Logical Expressions

For the remainder of this handout, we will use the following data, which is a collection of county-level data used by D. Matthews and J. Prothro in *Negroes and the New Southern Politics*. We can answer some interesting questions using this data set. Does the state-wide mean value of black voter registration depend on the existence of polltax? What about the literacy requirement? Finally, what do you find when considering the four combinations of these two? The variables of the data are:

Variable	Description
`state`	state name
`county`	county code
`polltax`	the existence of polltax (1 = Yes, 0 = No)
`litreq`	the existence of literacy requirement (1 = Yes, 0 = No)
`blackPop`	1960 % black of state population (100s)
`pBlackReg`	% black voting age population registered in 1964 (black registration rate)
`fedex66`	federal examiner present in county in 1966
`pIncreaseReg`	% increase in black registration rate from 1964 to 1968

- In the previous week, we learned that vectors and data frames can be subsetted by using brackets (`[ ]`). For example, a subset of a data frame can be obtained by specifying row numbers (or row names) and column numbers (or column names) in brackets. It turns out that logical expressions can also be used within brackets for subsetting.

```
> reg <- read.table("registration.txt", header=TRUE)
> ## black registration is lower where polltax is present
> mean(reg$pBlackReg[reg$polltax == 1])

[1] 17.9589

> mean(reg$pBlackReg[reg$polltax == 0])

[1] 38.2488

> ## black registration is lower where literacy requirement is imposed
> mean(reg$pBlackReg[reg$litreq == 1])

[1] 30.3551

> mean(reg$pBlackReg[reg$litreq == 0])

[1] 53.0716

> ## black registration is lowest where both requirements are present
> mean(reg$pBlackReg[(reg$polltax == 1) & (reg$litreq == 1)])

[1] 17.9589

> ## no observations returns NaN
> mean(reg$pBlackReg[(reg$polltax == 1) & (reg$litreq == 0)])
```

```
[1] NaN
```

```
> mean(reg$pBlackReg[(reg$polltax == 0) & (reg$litreq == 1)])
```

```
[1] 34.6375
```

```
> mean(reg$pBlackReg[(reg$polltax == 0) & (reg$litreq == 0)])
```

```
[1] 53.0716
```

- In addition to `[ ]`, `subset()` may be used to subset data, which takes vectors and data frames as the first argument. Then, users can specify `subset` and/or `select` as arguments. The former should be a logical vector indicating elements or rows to keep while the latter should specify the variables to keep (either by a vector of variable names or by a numeric vector indicating column numbers)

```
> ## counties with a higher than average black population but lower than
> ## average registration rate
> lowreg <- subset(reg, subset = ((reg$blackPop >= mean(reg$blackPop))
+                                 & (reg$pBlackReg <= mean(reg$pBlackReg))),
+                  select = c("blackPop", "pBlackReg", "polltax", "litreq"))
> ## How many impose both polltax and literacy requirement
> nrow(lowreg[(lowreg$polltax == 1) & (lowreg$litreq == 1), ])
```

```
[1] 34
```

```
> ## Another way
> sum((lowreg$polltax == 1) & (lowreg$litreq == 1))
```

```
[1] 34
```

3 Using Conditional Statements via `ifelse()`

Conditional Statements evaluate a logical statement, then perform different actions depending on whether the statement is true or false. The function `ifelse(X, Y, Z)` performs an action `Y` and returns the result of this action as the output if the statement `X` is true and performs `Z` and returns the output if `X` is false.

```
> ## Creating a new variable indicating counties with higher than average
> ## black population and polltax
> reg$highpoptax <- ifelse((reg$blackPop >= mean(reg$blackPop) & reg$litreq == 1),
+                          "Yes", "No")
> ## a more complex example creating region variable
> reg$region <- ifelse(reg$state=="Alabama" | reg$state=="Georgia" |
+         reg$state=="Louisiana" | reg$state=="Mississippi" |
+         reg$state=="South Carolina", "Deep South", "Peripheral South")
> reg$region <- as.factor(reg$region)
> table(reg$region)

      Deep South Peripheral South
             361              76
```

4 More Functions for Summarizing Data

In addition to the functions we learned last week (i.e., `mean()`, `median()`, `min()`, `max()`, and `range()`), we have the following new functions that are useful for summarizing data.

- `var()` (variance) and `sd()` (standard deviation) summarize numeric data.

```
> ## two ways of calculating standard deviation
> sd(reg$pBlackReg)

[1] 25.3474

> sqrt(var(reg$pBlackReg))

[1] 25.3474
```

- Weighted mean can be computed using `weighted.mean(X, Y)`, where the output is the mean of `X` weighted by `Y`.

```
> ## overall registration rate should be weighted by county population
> weighted.mean(reg$pBlackReg, reg$blackPop)

[1] 32.0482
```

- The function `quantile(X, P)` provides the sample quantiles of a numeric vector `X` for each element of another numeric vector `P`.

```
> quantile(reg$pBlackReg) # the default is quartiles plus min and max

  0%  25%  50%  75% 100%
 0.0 12.3 29.3 52.9 99.9

> quantile(reg$pBlackReg, seq(from = 0.2, to = 0.8, by = 0.2))  # quintiles

  20%   40%   60%   80%
 8.70 23.10 36.00 58.18
```

- The function `IQR()` returns the interquartile range

```
> IQR(reg$pBlackReg)

[1] 40.6
```

5 Applying Functions by Indexes

In many situations, we want to apply the same function repeatedly for different parts of the data. For example, in the black registration data, we may want to compute the registration rate within each state. Doing this manually is a pain especially if the number of states is large; you have to subset the data for one state and then use `mean()` to compute the registration for that state, and this has to be repeated for each state.

The function `tapply()` (`t` is a short hand for table) enables you to do such computation in one line. Specifically, `tapply(X, INDEX, FUN)` applies the function `FUN` to `X` for each of the groups defined by a vector `INDEX`. Replace `FUN` with `mean, median, sd`, etc. to generate desired quantity.

```
> ## Calculate the mean of % black registration rates by state
> tapply(reg$pBlackReg, reg$state, mean)

      Alabama        Florida        Georgia      Louisiana
     24.14242       53.07164       31.70701       37.99048
Mississippi North Carolina South Carolina
      3.88621       47.97778       37.43696
```

6 Graphs for Univariate Data: Histograms

Graphs are critical tools for summarizing data in a straightforward and easy to understand manner. Great graphics strengthen projects and report by illustrating central features of the data without much additional explanation. Bad graphics are inefficient (leaving out critical information such as labels), potentially misleading, or too complicated.

- There are several common graphing arguments that specify basic features of the graph, including the number of figures included on a graph, titles, axis labels, data range, etc. The following table summarizes these arguments:

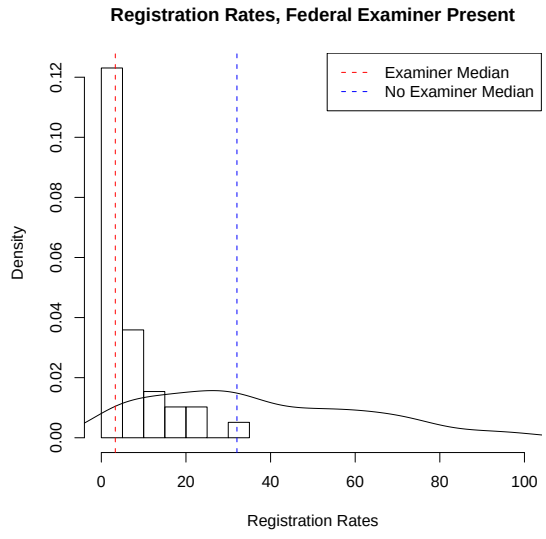
`main`	Main title of the graph.
`xlab`, `ylab`	Labels for the x-axis and y-axis.
`xlim`, `ylim`	Specifies the x-limits and y-limits, as in `xlim = c(0, 10)`, for the interval $[0, 10]$.
`col`	Specifies the color to use , e.g., `"blue"` or `"red"`.

- The second class of graphing commands adds additional features to an existing graph. These functions include `points()` for adding points, `lines()` for lines, and `text()` for texts.
- The function `hist()` will produce a histogram to summarize the distribution of data. Setting `freq = FALSE` within `hist()` will produce a histogram rather than a frequency plot. If you specify a single number as the argument `breaks`, you will be able to set the number of equally spaced bins. If you give a numeric vector instead, it will specify the breakpoints between histogram cells.
- The function `density()` will calculate the smooth density of a `numeric` object as an output, which then in turn can be an input to the `plot()` function to draw the smooth histogram (use the `lines()` function to add it to the existing graph).

`lines()`	Adds a plot-line to figure
	e.g. `lines(x, y)` where `x` and `y` define coordinates
`abline()`	Adds a straight line
	e.g. `abline(h = x)` to place a horizontal line at height `x`
	e.g. `abline(v = x)` to place a vertical line at point `x`
	e.g. `abline(a = x, b = y)` to place a line with intercept `x` and slope `y`
`points()`	Add points
	e.g. `points(x, y)` to place dots with x and y as the coordinates
	e.g. `points(x, y, line = TRUE)` connects the dots as a line
`text()`	Adds additional text
	e.g. `text(x, y, z)` to display `z` as a text centered at coordinates (`x`, `y`)

- To add a legend to an existing graph, use `legend()`. The syntax `legend(x, y, z)` adds legend with text `z` at coordinates (`x`, `y`), which can also be substituted with `"topleft"`, `"bottomright"`, etc.

```
> ## begin by subsetting the data
> examiner <- reg[reg$fedex66 == 1, ]
> noexaminer <- reg[reg$fedex66 == 0, ]
> ## histogram for counties with examiner
> hist(examiner$pBlackReg, freq = FALSE, breaks = 10, xlim = c(0, 100),
+      main = "Registration Rates, Federal Examiner Present",
+      xlab = "Registration Rates")
> ## add counties with no-examiner as smooth density
> lines(density(noexaminer$pBlackReg))
> ## add lines to compare median of counties with/without examiner
> abline(v = median(examiner$pBlackReg), col = "red", lty = 2)
> abline(v = median(noexaminer$pBlackReg), col = "blue", lty = 2)
> ## add legend
> legend("topright", c("Examiner Median", "No Examiner Median"),
+        lty = c(2, 2), col = c("red", "blue"))
```



7 Printing and Saving Graphs

There are a few ways to print and save the graphs you create in **R**.

- In the window of your graph (if you are a Mac user, make sure your graphic window rather than the R console is selected), you can click File: Save as: PDF... or File: Print....
- You can also right-click on a figure in **R** and copy the image (if you are a Mac user, you need to highlight the graph and type Apple+C to copy it). Then paste that image into Microsoft Word or any other document.
- You can also do it via a command by using `pdf()` before your plotting commands and then `dev.off()` afterwards.

```
> pdf(file = "myplot.pdf", height = 3, width = 5) # height and width are in inches
> dev.off() ## This creates a pdf file in the working directory
```

8 Precept Questions

In preparation for precept, please answer the following questions based on the `justices.RData` file (available at Blackboard) and submit your code following the instruction given in the syllabus. In a 2002 article, Andrew Martin and Kevin Quinn explored the extent to which the ideal points (i.e., policy preferences) of Supreme Court Justices change throughout their tenure on the Court.The data set contains the following:

- `term` – Supreme Court Term
- `justice` – Justice's Last Name
- `idealpt` – Justice's Estimated Ideal Point, where negative values indicate liberal leanings and positive values indicate conservative leanings
- `pparty` – President's Political Party

1. Using the `tapply()` function, create a variable for the median ideal point of court justices for each term of the court. (Hint: You may need to change the class of a variable to do this. See the section on "Object Class" from Precept 1).
2. Generate a new variable *in the justices data set* to indicate whether each justice falls on the Conservative or Liberal end of the ideal point spectrum. Using `ifelse()`, generate a new variable that takes a value of `Liberal` if the justice's ideal point is less than 0 and a value of `Conservative` if the justice's ideal point is greater than 0. Using `table()`, determine how many justices in the data set were Conservative and how many were liberal.
3. Create a histogram of justice's ideal points. Using `tapply()`, calculate *each justice's* median ideal point. Generate a histogram of the justice's ideal points. Be sure to add an informative title and labels. Create a red, vertical dashed line indicating the median. Additionally, add the density line to the plot. Save the graph you created as a pdf file using the file name `xxx.pdf` where `xxx` is your netID. Submit it to Blackboard along with your R script file `xxx.R` (Do not turn in your R console print out).