Exercise 1. Data Management Basics

Part 2. *Merging, Categorizing & the Grammar of Graphics*

Summary

The purpose of this second part to lab 1 is to continue introducing some of the basic tools you will be using during this course throughout the semester. You will access public data that is available from the National Center for Education Statistics, obtain that data, and analyze the data to determine rankings of the top American Universities. You will then be introduced to the *grammar of graphics* package, AKA, `ggplot`, and its multitude of useful functions for communicating the results from your work.

Objectives

1. To interactively retrieve data from an open access server via a web browser and import that `.csv` (comma-separated values) file from your William & Mary `H:\\` drive.
2. To subset data according to categorical and continuous variables.
3. To analyze the dataset according to a variety of quantitative methods.
4. To install the library of functions, `ggplot` and use the package in order to communicate the results of your analysis.

Grading This is a two part lab. Both parts are due on Saturday, February 9th by 6:30 PM. This lab is worth 5 points towards your final grade. Your grade will be based on the following criteria.

- Your ability to obtain data from an open access website.
- Your ability to modify, describe and analyze that dataset using primary function from the R base package.
- Your ability to demonstrate basic competence using `ggplot` and save your produced results.

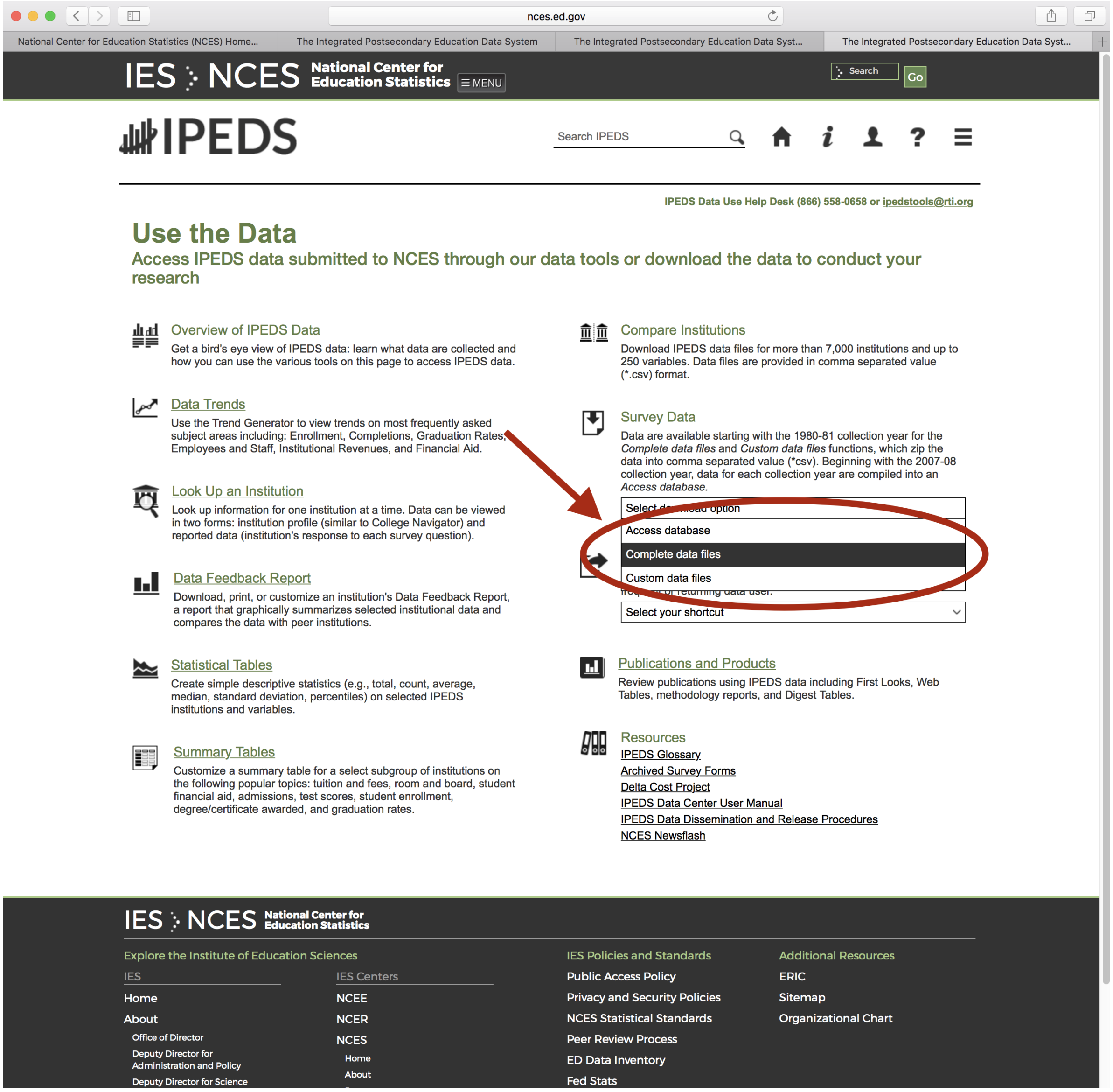
You will need to upload the laboratory deliverables to blackboard prior to the deadline.

Instructions. *Part 2: The Basics Extended - Importing Secondary Data from a Remote Source*

1. Open R Studio. Go to your William & Mary `H:\\` drive, and create a new `.R` script, name it `script_1b` or something similar and save it to your `H:\\lab_1` folder.
2. Just as we have previously done, set your working directory using the `setwd()` command.
3. With your browser go to the National Center for Education Statistics' Data Center website. You should be able to simply click on the following link.

   `https://nces.ed.gov/ipeds/Home/UseTheData`

   Under the survey tab, click on **Complete Data Files** as indicated in the figure below.


nces.ed.gov
National Center for Education Statistics (NCES) Home...
The Integrated Postsecondary Education Data System
The Integrated Postsecondary Education Data Syst...
The Integrated Postsecondary Education Data Syst...
IES NCES National Center for Education Statistics
MENU
Search
Go
IPEDS
Search IPEDS

IPEDS Data Use Help Desk (866) 558-0658 or ipedstools@rti.org

Use the Data

Access IPEDS data submitted to NCES through our data tools or download the data to conduct your research

Overview of IPEDS Data
Get a bird's eye view of IPEDS data: learn what data are collected and how you can use the various tools on this page to access IPEDS data.

Data Trends
Use the Trend Generator to view trends on most frequently asked subject areas including: Enrollment, Completions, Graduation Rates, Employees and Staff, Institutional Revenues, and Financial Aid.

Look Up an Institution
Look up information for one institution at a time. Data can be viewed in two forms: institution profile (similar to College Navigator) and reported data (institution's response to each survey question).

Data Feedback Report
Download, print, or customize an institution's Data Feedback Report, a report that graphically summarizes selected institutional data and compares the data with peer institutions.

Statistical Tables
Create simple descriptive statistics (e.g., total, count, average, median, standard deviation, percentiles) on selected IPEDS institutions and variables.

Summary Tables
Customize a summary table for a select subgroup of institutions on the following popular topics: tuition and fees, room and board, student financial aid, admissions, test scores, student enrollment, degree/certificate awarded, and graduation rates.

Compare Institutions
Download IPEDS data files for more than 7,000 institutions and up to 250 variables. Data files are provided in comma separated value (*.csv) format.

Survey Data
Data are available starting with the 1980-81 collection year for the *Complete data files* and *Custom data files* functions, which zip the data into comma separated value (*csv). Beginning with the 2007-08 collection year, data for each collection year are compiled into an *Access database.*

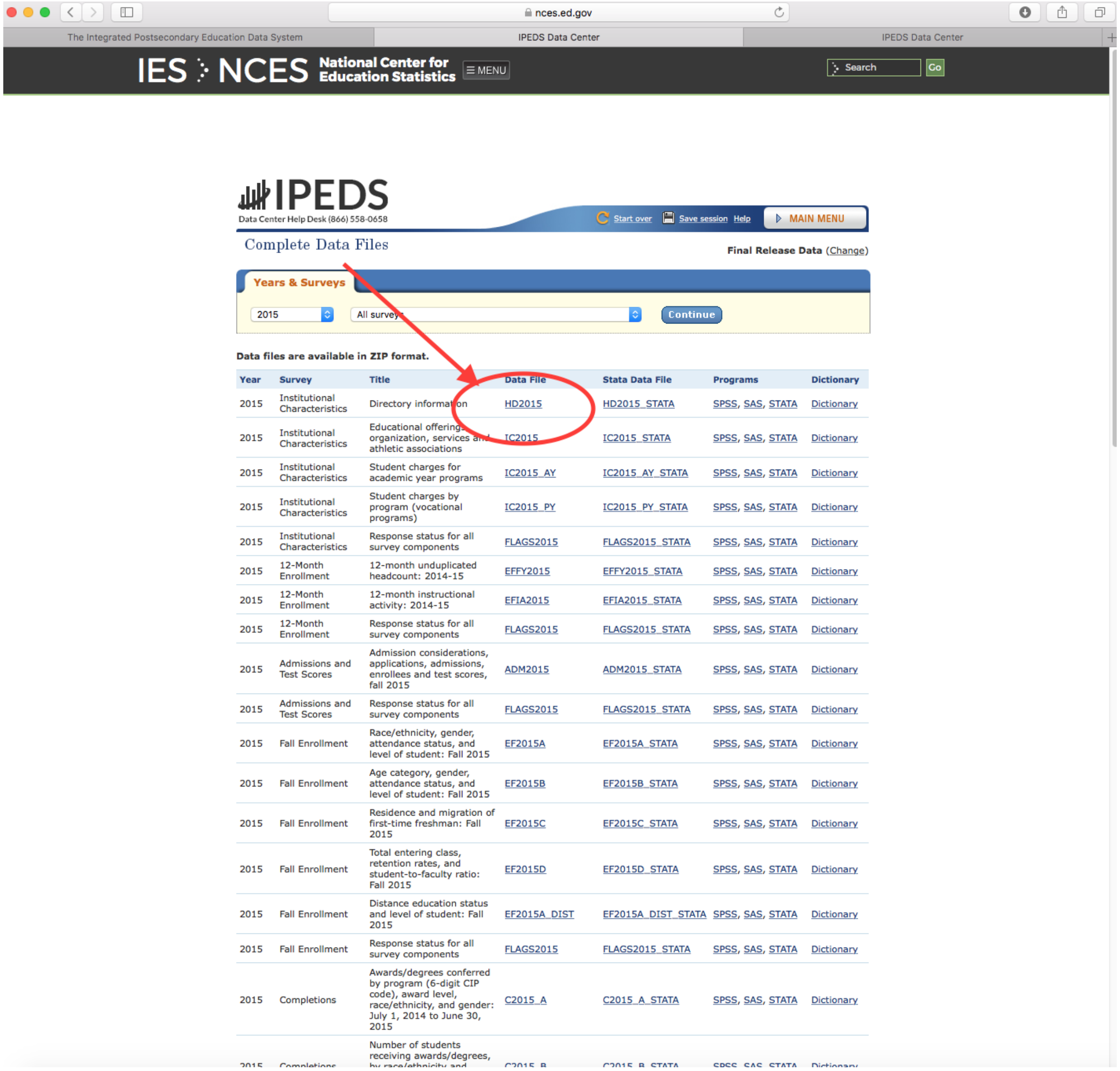
frequent or returning data user.

Publications and Products
Review publications using IPEDS data including First Looks, Web Tables, methodology reports, and Digest Tables.

Resources
IPEDS Glossary
Archived Survey Forms
Delta Cost Project
IPEDS Data Center User Manual
IPEDS Data Dissemination and Release Procedures
NCES Newsflash

IES NCES National Center for Education Statistics
Explore the Institute of Education Sciences
IES
Home
About
Office of Director
Deputy Director for Administration and Policy
Deputy Director for Science
IES Centers
NCEE
NCER
NCES
Home
About
IES Policies and Standards
Public Access Policy
Privacy and Security Policies
NCES Statistical Standards
Peer Review Process
ED Data Inventory
Fed Stats
Additional Resources
ERIC
Sitemap
Organizational Chart

4. After you select **Complete Data Files**, the server should bring you to a new webpage that gives you the option to select from two drop down menus that will generate data for selected years from a number of different possible survey data sets. Go ahead and select **2015**.


nces.ed.gov
The Integrated Postsecondary Education Data System
IPEDS Data Center
IPEDS Data Center
IES NCES National Center for Education Statistics
MENU
Search
Go
IPEDS
Data Center Help Desk (866) 558-0658
Start over
Save session
Help
MAIN MENU
Years & Surveys
2015
All surveys
Continue

Complete Data Files

Final Release Data (Change)

Data files are available in ZIP format.

Year	Survey	Title	Data File	Stata Data File	Programs	Dictionary
2015	Institutional Characteristics	Directory information	HD2015	HD2015_STATA	SPSS, SAS, STATA	Dictionary
2015	Institutional Characteristics	Educational offerings, organization, services and athletic associations	IC2015	IC2015_STATA	SPSS, SAS, STATA	Dictionary
2015	Institutional Characteristics	Student charges for academic year programs	IC2015_AY	IC2015_AY_STATA	SPSS, SAS, STATA	Dictionary
2015	Institutional Characteristics	Student charges by program (vocational programs)	IC2015_PY	IC2015_PY_STATA	SPSS, SAS, STATA	Dictionary
2015	Institutional Characteristics	Response status for all survey components	FLAGS2015	FLAGS2015_STATA	SPSS, SAS, STATA	Dictionary
2015	12-Month Enrollment	12-month unduplicated headcount: 2014-15	EFFY2015	EFFY2015_STATA	SPSS, SAS, STATA	Dictionary
2015	12-Month Enrollment	12-month instructional activity: 2014-15	EFIA2015	EFIA2015_STATA	SPSS, SAS, STATA	Dictionary
2015	12-Month Enrollment	Response status for all survey components	FLAGS2015	FLAGS2015_STATA	SPSS, SAS, STATA	Dictionary
2015	Admissions and Test Scores	Admission considerations, applications, admissions, enrollees and test scores, fall 2015	ADM2015	ADM2015_STATA	SPSS, SAS, STATA	Dictionary
2015	Admissions and Test Scores	Response status for all survey components	FLAGS2015	FLAGS2015_STATA	SPSS, SAS, STATA	Dictionary
2015	Fall Enrollment	Race/ethnicity, gender, attendance status, and level of student: Fall 2015	EF2015A	EF2015A_STATA	SPSS, SAS, STATA	Dictionary
2015	Fall Enrollment	Age category, gender, attendance status, and level of student: Fall 2015	EF2015B	EF2015B_STATA	SPSS, SAS, STATA	Dictionary
2015	Fall Enrollment	Residence and migration of first-time freshman: Fall 2015	EF2015C	EF2015C_STATA	SPSS, SAS, STATA	Dictionary
2015	Fall Enrollment	Total entering class, retention rates, and student-to-faculty ratio: Fall 2015	EF2015D	EF2015D_STATA	SPSS, SAS, STATA	Dictionary
2015	Fall Enrollment	Distance education status and level of student: Fall 2015	EF2015A_DIST	EF2015A_DIST_STATA	SPSS, SAS, STATA	Dictionary
2015	Fall Enrollment	Response status for all survey components	FLAGS2015	FLAGS2015_STATA	SPSS, SAS, STATA	Dictionary
2015	Completions	Awards/degrees conferred by program (6-digit CIP code), award level, race/ethnicity, and gender: July 1, 2014 to June 30, 2015	C2015_A	C2015_A_STATA	SPSS, SAS, STATA	Dictionary
2015	Completions	Number of students receiving awards/degrees, by race/ethnicity and	C2015_B	C2015_B_STATA	SPSS, SAS, STATA	Dictionary

5. This should bring you to a new window where different data files will appear. Select the **Institutional Characteristics** Data File **HD2015** for the 2015 Institutional Characteristics Survey. Once you select the hyperlink to the file, it should begin to download. Place the `hd2015.csv` file in your working directory on your William & Mary `H:\\` drive.

6. Before we leave the IPEDS webpage, go ahead and retrieve the Dictionary for the 2015 Institutional Characteristics Survey. The Dictionary provides information about our data and will help us to understand it better. Go ahead and follow the **2015 Institutional Characteristics Dictionary** link, retrieve the file, in this case named `hd2015.xlsx` and also save it in your William & Mary `H:\\` drive. Once you have saved the data dictionary for our HD2015 dataset, you can open it and have a look at names and descriptions for all of the different variables contained in the data file. You will use the data dictionary to determine the name of individual variables in the dataset that you will need.

7. Now that we have retrieved our data and its corresponding dictionary, we can go ahead and import our `.csv` file into R Studio.

   ```
   inst_raw_data <- function("name_of_file")
   ```

   After creating our object named `inst_raw_data`, you will notice that R Studio adds a new element to the environment tab under data in the top right hand pane. You should now be able to see the data object named `inst_raw_data` in the top right hand corner with a brief description of our data. We can see thatR Studio informs us that `inst_raw_data` has 7647 observations (rows) and 70 variables (columns). If we click on the small blue arrow to the left, our object will expand and each of the 70 variables will be listed, starting with the name of that variable, the class of that variable and a list of the first few outcomes from within that row.

8. Lets consider our dataset of 7647 colleges and the 70 variables used to describe each institutions basic information. First off, create a dataset that includes only schools that offer at minimum a 4 year program. Open your data dictionary and find the variable that classifies each institution as 4-year or higher (4 year), 2-but-less-than 4-year (2 year), or less than 2-year. We will use the `subset()` command to choose only *4-year or higher* colleges from our raw data and write them to a new data frame.

   The `subset()` command is similar to using our subscripting operators `[ ]`, but instead of executing a command by first identifying the `object[row(s), column(s)]`, now we will use a command. When using the `subset()` command, R first asks you to identify an object (such as a data frame) then in the subsequent argument asks for the variable within that object and the specific outcome(s) from that variable that will be kept. In the following command we are asking R to `subset()` all of the observations within our data frame named `inst_raw_data` where the variable `ICLEVEL` is equal to 1.

   ```
   four_yr_colls <- subset(inst_raw_data, ICLEVEL == 1)
   ```

   You will notice that the variable has been set equal to the number that represents the selected outcome. For example, if you had set `ICLEVEL == 2`, then you would have kept all of the observations where Colleges offered 2-but-less-than 4-year (2 year) degrees (i.e. community colleges).

   Now we see that a new data frame object named `four_yr_colls` has been added to our top right pane under our global environment, with 3243 observations (and the same 70 variables) instead of 7647. If you move your cursor to our new object, `four_yr_colls` and click on it, a spreadsheet viewer will open in a new tab adjacent to our script editor in the top left pane. In the top right corner of our data viewer is a small magnifying glass and a field where we can search our data frame object. Test to confirm our University is included in your data frame.

9. In addition to creating a data set that is a subset of all observations that offer a 4-year program, return to the **Data Dictionary** and find the variable that describes all Universities

that offer a Doctoral degree. Again use the `subset()` command to keep all observations where the outcome for the highest level of degree offered is a doctorate.

```
doc_colls  <- subset(inst_raw_data, VARIABLE == OUTCOME)
```

Again, you will notice that the variable is a number that represents the indicated outcome as designated in the **Data Dictionary**. You should notice your new data frame object that represents all Universities that offer doctoral degrees has 1185 observations (while continuing to retain all 70 variables).

10. By keeping your source data external to R and then importing it into your workspace as a new object, you have greatly enhanced the **reproducibility** of your work and advanced one of the fundamental principals of the scientific method. It is good to recognize that your work is in your code, while the data you retrieved from the Department of Education, remains in its original state and does not change. The script you write is applied to the imported object within our workspace. You can literally run your code an infinite number of times, and so long as the data residing on your `H:\\` drive does not change, the output from R will always be the same. A good first step in order to assure **reproducibility** by avoiding confounding vestiges from a previously run lines of script during your current session, is to delete the entire workspace as your very first command. By deleting everything in your workspace you literally have a clean slate before importing your data set again *de novo*.

```
rm(list=ls(all=TRUE))
setwd("H://your_working_directory")
```

By clearing your R workspace with your first line of code, you have literally created a *tabula rasa* and *voilà!* R thinks, there for it is! [1]

Instructions. *Part 2: The Basics Extended - Importing, Joining & Categorizing your Data*

1. With your browser, return to the IPEDS datacenter website at `https://nces.ed.gov/ipeds/Home/UseTheData`. Again choose **Complete Data Files** and then **Provisional Release Data**. For **Years & Surveys** again select **2015** but this time choose **Admissions and Test Scores** from the two drop down menus. Go ahead and retrieve the data file **ADM2015** and the **Data Dictionary** (`adm2015.csv` & `adm2015.xlsx`). Save both to your William & Mary `H:\\` drive.

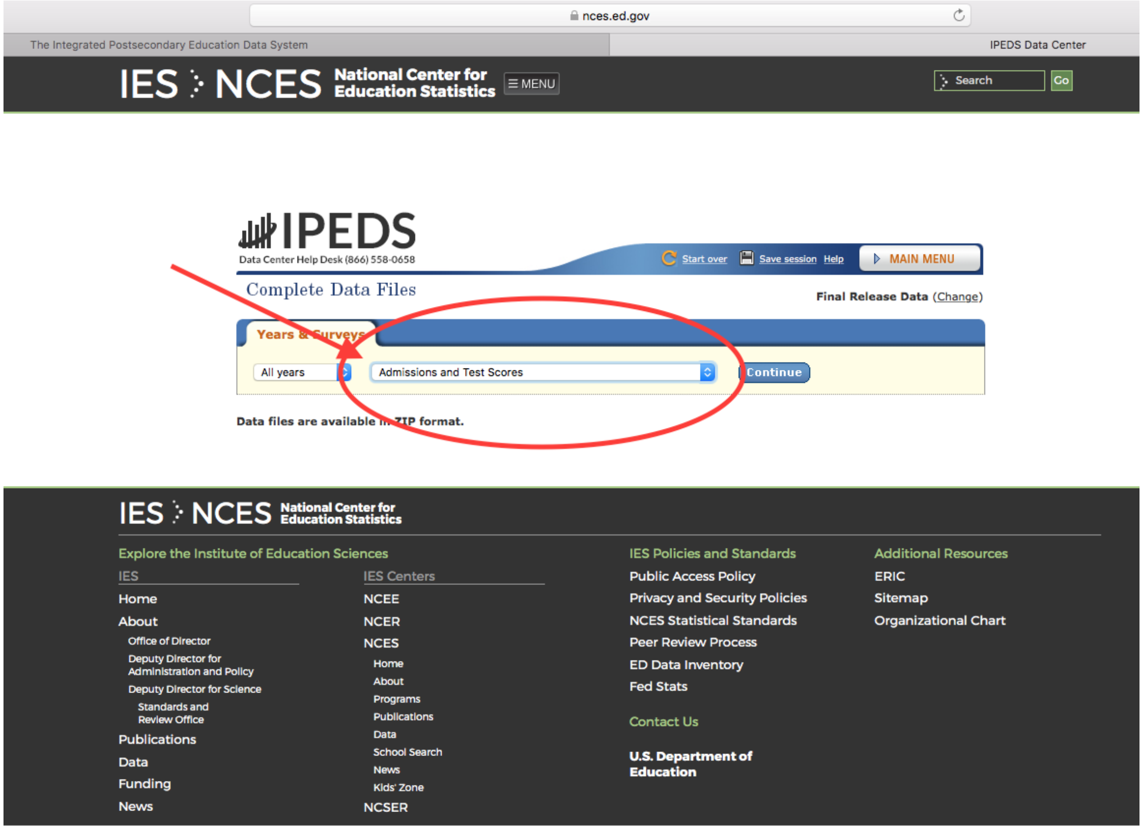
2. Import `adm2015.csv` into your R workspace, but instead of adding it to the last line of your script, go back to where you imported `hd2015.csv` and add this new line of code next.

```
inst_raw_data <- function("name_of_file")
adm_raw_data <- function("name_of_file")
```

You should see that your new data frame has 2210 observations with 76 variables.

---

[1] or if you prefer in Réne Descartes original French phrasing from 1637: *"Je pense, donc je suis!"*


nces.ed.gov
The Integrated Postsecondary Education Data System
IPEDS Data Center
IES NCES National Center for Education Statistics
MENU
Search
Go
IPEDS
Data Center Help Desk (866) 558-0658
Start over
Save session
Help
MAIN MENU
Complete Data Files
Final Release Data (Change)
Years & Surveys
Continue
Data files are available in ZIP format.
IES NCES National Center for Education Statistics
Explore the Institute of Education Sciences
IES
Home
About
Office of Director
Deputy Director for Administration and Policy
Deputy Director for Science
Standards and Review Office
Publications
Data
Funding
News
IES Centers
NCEE
NCER
NCES
Home
About
Programs
Publications
Data
School Search
News
Kids' Zone
NCSER
IES Policies and Standards
Public Access Policy
Privacy and Security Policies
NCES Statistical Standards
Peer Review Process
ED Data Inventory
Fed Stats
Contact Us
U.S. Department of Education
Additional Resources
ERIC
Sitemap
Organizational Chart

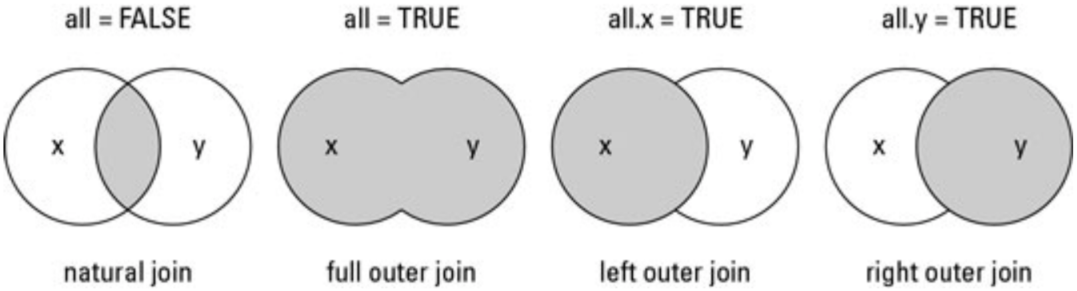
All years

Admissions and Test Scores

3. `View` both of your data frame objects, `inst_raw_data` & `adm_raw_data` and notice that the first variable in both is named `UNITID`. This is the unique identification number that is used to identify each College or University, in order to join different data sets. Joining data by a unique number that identifies an observation will enable us to extend an existing data set for more in-depth analysis by adding more descriptive data. While there are many permutations of combinations we can create when specifying a join, there are four basic types of functions. These types of *joins* borrow their name from SQL, and while they accomplish essentially the exact same result, the name of the function in the base R package is `merge`.

   - **Inner Join or Natural Join** (R argument `all = FALSE`) is the joining of two data sets by using an identifier where only exact matches from both the first data set and the second data set are retained.
   - **Left Join** (R argument `all.x = TRUE`) is the joining of two data sets by using an identifier where all the cases from the first data set are retained while only exact matches from the second data set are kept.
   - **Right Join** (R argument `all.y = TRUE`) is the joining of two data sets by using an identifier where all the cases from the second data set are retained while only exact matches from the first data set are kept.

- **Exclude** (not shown in the illustration below) is creating a new object from the individual observations that did not have a match (can be either from the first or second data set)



Perhaps the most useful join functions are the **inner join**, **left join**, **right join** and the **excludes**. Through using these types of joins, it is possible to greatly increase the probability of getting as close as possible to a perfect match between two or more data sets, even when matching IDs are incomplete or partially incoherent.[2]

We will use the `merge()` function in R to execute our join. Immediately following the import of our two `.csv` files from our working directory into our workspace, add the following command.

```
inst_raw_data <- merge(x = inst_raw_data, y = admin_raw_data,
by = "UNITID", all.x = TRUE)
```

You will notice I have specified the argument as `all.x = TRUE`. This will execute a left join where our first data set `inst_raw_data` is the `x` of left join object. Every observation in the data frame will be retained irregardless of whether it has a matching ID or not. You could modify this part of the argument to get a better understanding of what other specifications will return, for example you could try `all = FALSE` to compare an **inner join** with a **left join**. Do you see what happened with the number of observations in the resulting data frame object? How has it changed? Still, for our purposes, make sure you either comment out your **inner join** command or return your script back to its original state using the **left join**. We want to be certain and retain all observations in our `x` data set.

Following your `merge()` of the two data sets, you should have the two `subset()` commands you wrote earlier. While this may seem like a lot of description up to this point in Part 2 of this lab, in reality, you should only have 7 total lines of code, including: removing all objects from your workspace; setting your working directory; importing two `.csv` files; your `merge()` command; and your two `subset()` commands. After rerunning the `subset()` functions in order to create the object that describes institutions offering at minimum a 4 year degree as well as those that offered doctorates, you will now notice that

---

[2] Many to one, and one to many merges are also possible.

each data frame has 145 variables, instead of 70, since the **admissions and test scores** data has been joined to the original data frame.

4. Before subsetting our raw data into two sets that offer at minimum a 4 year degree as well as all those that offer doctoral degrees, it would be a good idea to simplify and modify it before plotting and communicating our results. Use the `cbind.data.frame()` command to identify which variables you will extract and bind together as a new data frame object. Also use the = argument when specifying each variable so the variable name is clearly assigned in the new object. In your new data frame, go back to your data dictionary and be certain to include the variables that describe each institutions name, the state where the institution is located, its regional identifier, the variable that identifies the classification level for each college or university as well as the variable that identifies the highest degree offered.

```
new_df <- function(name_for_variable = your_data_frame$variable_for_inst_name,
        name_for_variable = your_data_frame$variable_for_state_abbrev,
        name_for_variable = your_data_frame$variable_for_region,
        name_for_variable = your_data_frame$variable_for_level_classifier,
        name_for_variable = your_data_frame$variable_for_highest_level
        )
```

Your new data frame should continue to have 7647 observations, but now instead of 145 variables, you should have just the five.

5. You will notice that most of the variables you created do not provide very helpful classifications in order to interpret their meaning. For example the variable you created to identify the region where an institution is located, just provides a series of numbers from 0 & 9. Enter the `table()` command into the console in order to sum the counts for each observation within the variable you have created to identify region. Your output should look like the following.

```
   0    1    2    3    4    5    6    7    8    9
   7  426 1189 1145  647 1856  821  298 1092  166
```

R is informing us of the counts for each of the possible outcome, but what do these different numbers represent? The data that describes each institutions region is not very informative in its raw format. In order to properly label each category, go back to the **Data Dictionary** for this variable and identify the nominal category (or region) that each number represents. Create a new character, vector object that includes the 10 labels that you will use to identify each region. Since the results from our `table()` command did not include the outcome `-3 - Not available`, just omit it.

```
your_char_vector <- c("US Service schools",
           "New England",
           "Mid East",
           "Great Lakes",
           "Plains",
                "Southeast",
                "Southwest",
                "Rocky Mountains",
                "Far West",
                "Outlying areas"
                )
```

6. After creating your object to label each of the 7647 outcomes in your variable describing the region, now you will assign each one individually. The base R package interprets the `factor()` command by first asking for your `data_frame$variable` and then arguments that interpret the number of total `levels =` within the variable as well as the `labels =` that should be assigned to each category.

   ```
   your_df$variable_for_region <- factor(your_df$variable_for_region,
            levels = min_number:max_number,
            labels = your_char_vector)
   ```

   Now the same `table()` command that you used in step 5 from above should return the following results.

   ```
   US Service schools         New England            Mid East         Great Lakes
                    7                 426                1189                1145
               Plains           Southeast           Southwest     Rocky Mountains
                  647                1856                 821                 298
             Far West      Outlying areas
                 1092                 166
   ```

   Hypothetically, you could execute the same functions in order to `factor()` outcomes in the variables you have created to describe each observations level of classification (all 4 year schools etc...) and highest degree offered (doctoral degree granting institutions) with both `levels =` and `labels =`. Really, this is not needed, since in the subsequent step, you will subset all observations based on your selected outcome from within the variable.

7. Before subsetting and then eventually plotting your data, you will need to add three more variables that describe a continuous measure of each institution in our data set. Go back to your **Data Dictionary** for the **adm2015** data set and find the following three variables.

   - Variable that indicates the *Total number of applicants* that applied to the College of University in the year 2015
   - Variable that indicates the *Total number of admissions* that applied to the College of University in the year 2015
   - Variable that indicates the *Total number of enrolled* that applied to the College of University in the year 2015

   Now add each of these 3 variables to your new data frame.

```
your_new_df$var_ttl_apps <- inst_raw_data$VAR_for_total_apps
your_new_df$var_ttl_admits <- inst_raw_data$VAR_for_total_admitted
your_new_df$var_ttl_enroll <- inst_raw_data$VAR_for_total_enrolled
```

8. Finally, run your `subset()` command that you wrote in step 8 from the first section of this lab (part 2) and create your `four_yr_colls` and `doc_colls` data frames. Make sure you update the name of the data frame to reflect the new one you created, as well as the names of the variable in that data frame that identify level of classification (all 4 year schools etc...) and highest degree offered (doctoral degree granting institutions).

   Since the variables you created in step 4 that designate level of classification (all 4 year schools etc...) and highest degree offered (doctoral degree granting institutions) are the same throughout each of the two data frames, they are no longer needed. Go ahead and delete both by assigning `NULL` to each column using your subscripting operators (you could also use the variable name if you want).

   ```
   four_yr_colls[ ,cols_to_delete] <- NULL
   doc_colls[ ,cols_to_delete] <- NULL
   ```

   Now both data sets should contain 3243 & 1185 observations respectively, each one with 6 variables.

Instructions. *Part 2: The Basics Extended - the Grammar of Graphics*

1. In this last section for Part 2 of Lab 1, you will be using the library of functions called `ggplot2` which stands for *the grammar of graphics*. In order to access the package of functions contained in the `ggplot2` library, you will first need to install and then load the library of functions. The commands you use to installing packages and load libraries into your work session, should be at the beginning of your script. You will also want to use the `dependencies = TRUE` argument so R will also install any other packages that `ggplot2` is dependent upon in order to function. Once you have installed the package into your R library, you then need to inform R to load the `library()` for use during the current session.

   ```
   rm(list=ls(all=TRUE))
   setwd("H://your_working_directory")

   install.packages("ggplot2", dependencies = TRUE)
   library(ggplot2)
   ```

   After running the `install.packages()` command, it will very likely take R some time in order to retrieve the packages from the remote server, unpack each one and install it to your local computer for use. You only need to install each package one time on each computer, unless you want to update the package or if for some reason the R library on the computer has been deleted (which IT does do from time to time). After executing the line `install.packages("ggplot2", dependencies = TRUE)`, comment it out by adding the # at the beginning of the line.

2. The format for plotting with ggplot is a little different than what you have used so far. First you create your ggplot object using the `ggplot()` command. Then once your ggplot object is created, you add layers using aesthetic mapping to variables within your object. First use the `ggplot()` command to create your object using the `four_yr_colls` data frame and the variable `region` as the base layer aesthetic.

   ```
   g <- ggplot(four_yr_colls, aes(region))
   ```

   This function creates your ggplot object named `g`. Then we will add a layer to the base object `g` by mapping aesthetics using `geom_bar()` command. When you run the second line of the script below, you should get a bar plot with that describes the total number of 4 years colleges per region.

   ```
   g <- ggplot(four_yr_colls, aes(region))
   g + geom_bar()
   ```

   Having the names along the bottom of your plot doesn't make too much sense. Flip the axes so the names of each region are on the side and the counts are along the bottom. Also, add a title to your plot. Use the `coord_flip() +` and `labs() +` commands to modify your plot.

   ```
   g + geom_bar() +
     coord_flip() +
     labs(title="Total Number of 4 year Colleges per Region")
   ```

   In each of the previous series of commands, the basic ggplot object `g` remains the same, and in order to generate plots we add different layers of aesthetics to the base layer by using the + function at the end of each line of script.

3. Now instead of creating a plot that describes the total number of 4 year institutions per region, calculate which region receives the most student applications seeking admission. If number of institutions is an indicator, we should also see some similarity between total number of institutions and total number of applications. To calculate the total number of applications, add a `weight = your_variable)` argument to your `geom_bar()` function.

   ```
   g + geom_bar(aes(weight = ttl_apps))
   ```

   Interestingly, the Mid-East has surpassed the Southeast with total number of applications. Sum the variable that describes the total number of applications using `sum(four_yr_colls$ttl_apps` `na.rm = TRUE)`. Compare the calculated results with your plot in order to conduct a basic validation of the data. Now go ahead and flip the axes again, so the names of each region appear on the side by using the `+ coord_flip()` command.

4. In addition to setting the `weight = ttl_apps` include a breakdown of each bar by each state's contribution to its region. Incorporate the `fill =` argument to include your variable that describes the total number of applications per state.

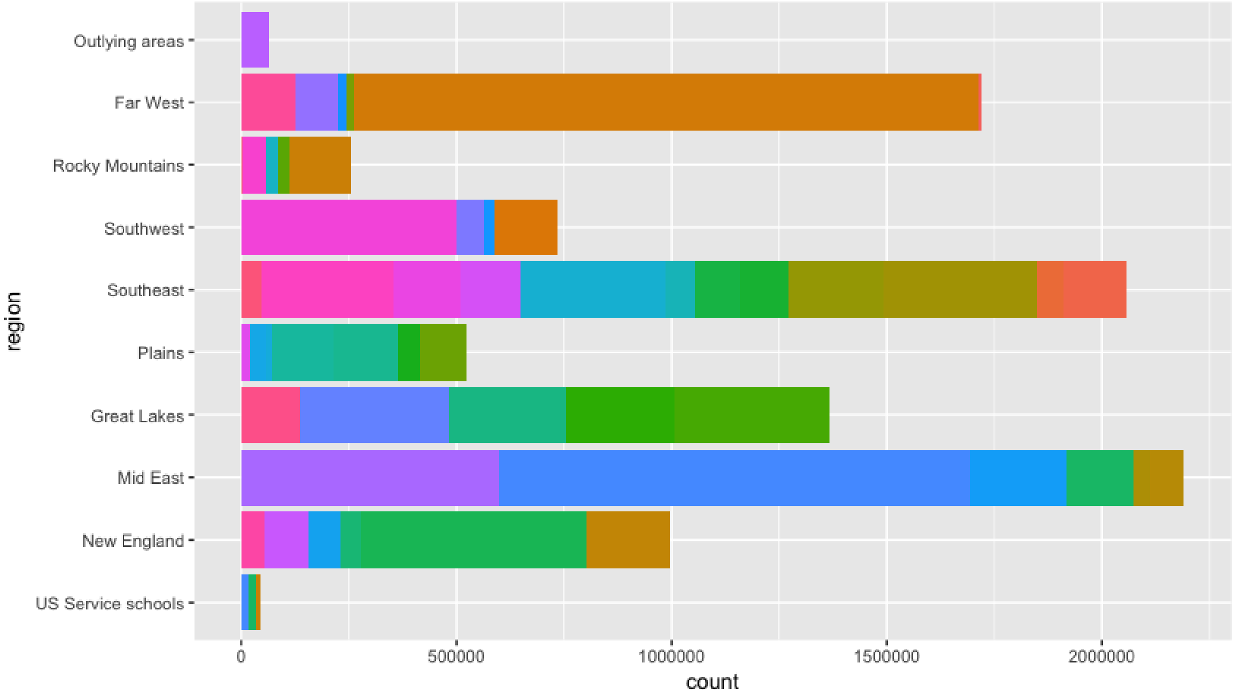
```
g + geom_bar(aes(weight = ttl_apps, fill = state)) +
  coord_flip()
```

This is interesting, but the color coded legend to the side is difficult to interpret. Perhaps we can try moving the legend to the top and see if that will help. Use the `theme()` command to modify the legend by repositioning it at the top of your plot.

```
g + geom_bar(aes(weight = ttl_apps, fill = state)) +
  coord_flip() +
  theme(legend.position = "top")
```

OK, but moving it to the top, doesn't really improve interpretation of the regional results per state. It would be better to label each portion of the bar directly within the barplot. For now, remove the legend by modifying the `legend.position =` argument from `"top"` to `"none"`.

```
g + geom_bar(aes(weight = ttl_apps, fill = state)) +
  coord_flip() +
  theme(legend.position = "none")
```



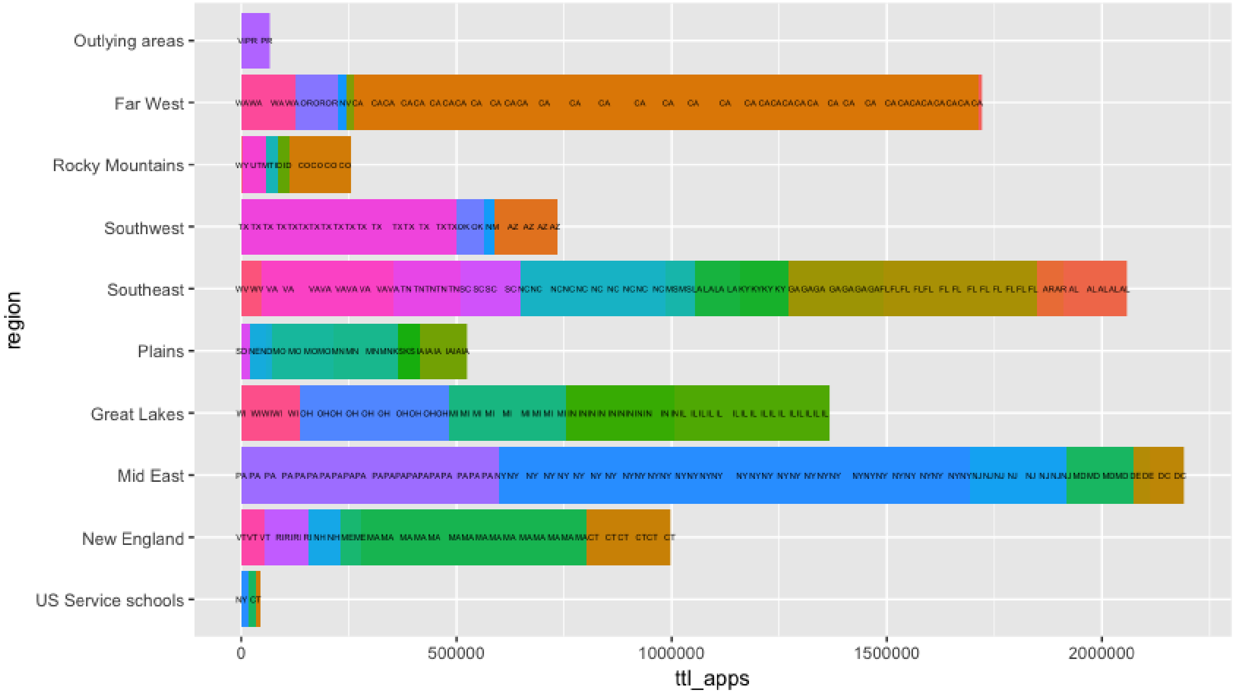
5. In order to assign the state abbreviation as a label to the portion of each bar that represents that states contribution to the total applications, you will need to explicitly respecify your ggplot object. Create a new ggplot object and name it `g2` instead of `g`. Now when assigning the base layer aesthetics, explicitly identify each variable. Also, add the argument

`label = state` in order to inform R which variable will be used in order to label our bar plot.

```
g2 <- ggplot(four_yr_colls,
aes(x = region, y = ttl_apps, fill = state, label = state))
```

6. Since you have mapped specific aesthetics to your base ggplot object, now you can add the `geom_bar()` function again, but this time indicate `stat = "identity"` since your x variable is a discrete set of categories, the regions, while your y variable is a continuous count of total applications. Remember we have flipped the axes. In addition we will add a new command `geom_text()` in order to map labels to our plot. Use the `position = position_stack(vjust = 0.5)` argument to correspond each label to its stacked portion of the bar within each region. The `vjust = .5` argument, specifies moving the label to the vertical mid-point within the that portion of the stacked bar. The `size =` argument sets the font for each label. Add the `check_overlap = TRUE` to resolve conflicts between overlapping labels.

```
g2 + geom_bar(stat = "identity") +
  geom_text(position = position_stack(vjust = 0.5),
    size = 1.5, check_overlap = TRUE) +
  coord_flip() +
  theme(legend.position = "none")
```



You are getting closer, but while each bar is labeled per state, ggplot has added one label for each individual college or university and its portion to the state contribution as well as to its region. While we can readily interpret this plot, the unnecessary repetition of labels degrades the overall presentation quality of the plot. We only need one label per state, but to effectively achieve this result, we will need to modify our data.

7. Fortunately, the ggplot family of functions also includes the `dplyr` package, which is a *grammar of data manipulation* and provides a consistent set of verbs that help you solve the most common data manipulation challenges. Go back and install the package `dplyr`. After you install the package, load the library.

```
rm(list=ls(all=TRUE))
setwd("H://your_working_directory")

#install.packages("ggplot2", dependencies = TRUE)
install.packages("dplyr", dependencies = TRUE)

library(dplyr)
library(ggplot2)
```

8. The package `dplyr` provides for a series of functions that allow you to group and summarize data in order expedite plotting and ultimately communicating your results. It uses the `%>%` operator so R can interpret hierarchical structure, for example variables inside a data frame, or outcomes inside a variable. The following series of commands creates a new data frame where the total number of applications are summed up and grouped by state. It also assigns the region to the state, so that variable is also available for producing our ggplot.

```
grp_state <- four_yr_colls %>%
  group_by(state) %>%
  summarise(
    n = n(),
    ttl_apps = sum(ttl_apps, na.rm = TRUE),
    region = first(region)
  )
```

9. Now go back and recreate your ggplot object (call it g3 this time). Since you have aggregated all of the total applications per state, now you will only have one label per section of each bar, within your barplot.

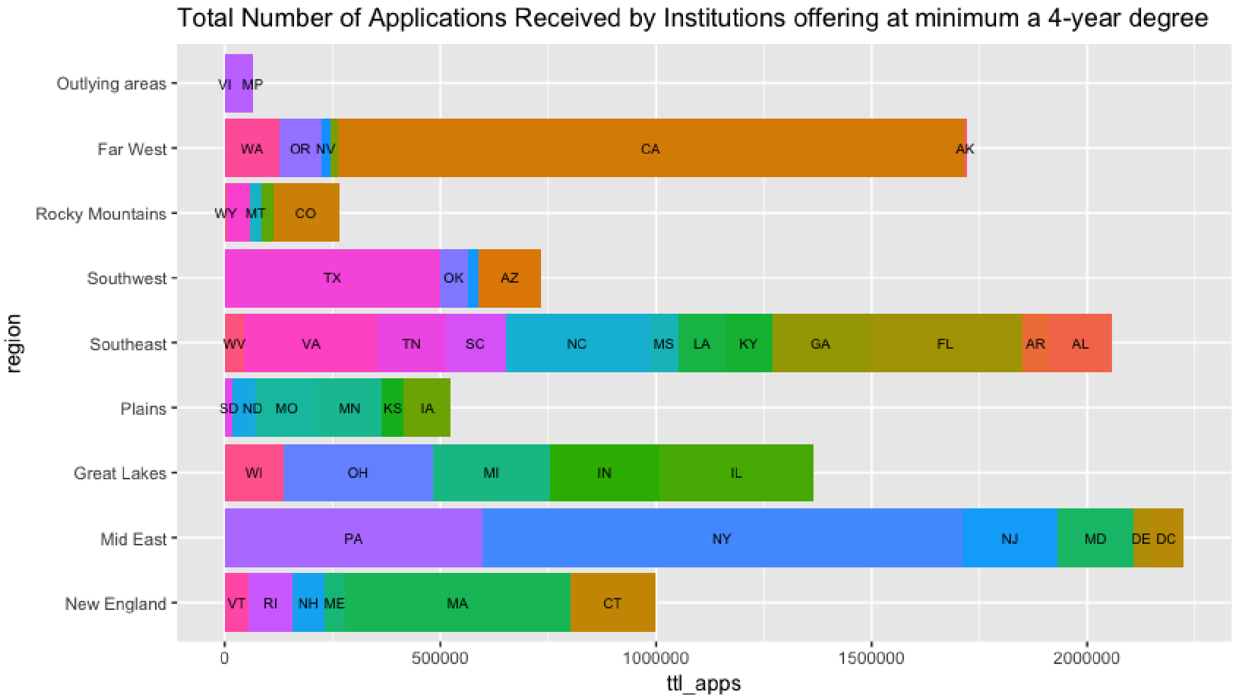
```
g3 <- ggplot(grp_state, aes(x = region, y = ttl_apps, fill = state,
  label = state))

g3 + geom_bar(stat = "identity") +
  geom_text(position = position_stack(vjust = 0.5), size = 1,
    check_overlap = TRUE) +
  coord_flip() +
  theme(legend.position = "none")
```

Deliverable 3.

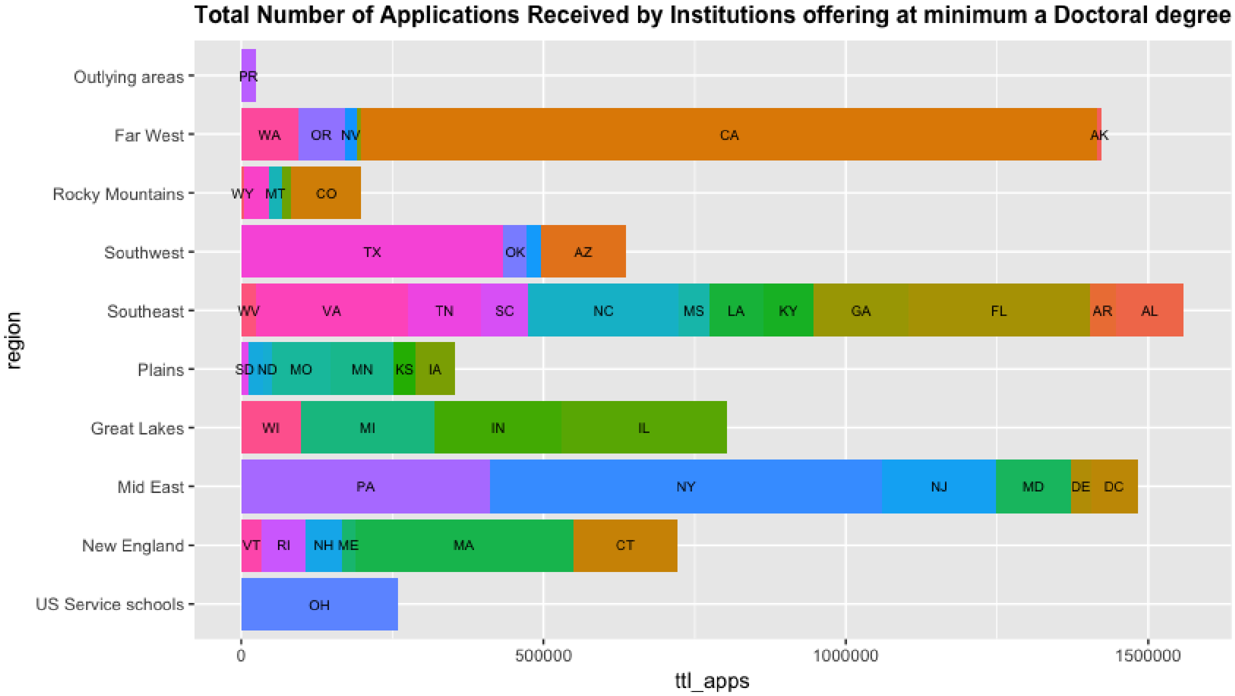
Now go back and do the same steps (1 through 9 in the last section) but instead plot the results for doctoral degree institutions.

Stretch Goals.

1. Generate the same results as above, but instead compare number of students enrolled.
2. Using the variables for number of students applied, number of students admitted and number of students enrolled, generate your list of the top 50 most selective universities in the United States.



Total Number of Applications Received by Institutions offering at minimum a 4-year degree



Total Number of Applications Received by Institutions offering at minimum a Doctoral degree