

## UNIT IV

### DATA EXPLORATION AND DATA VISUALIZATION IN R

Introduction to R and RStudio - The Basics of Data Exploration - Loading Data into R - Transforming Data - Creating Tidy Data

\*\*\*\*\*

#### I INTRODUCTION TO R AND R STUDIO

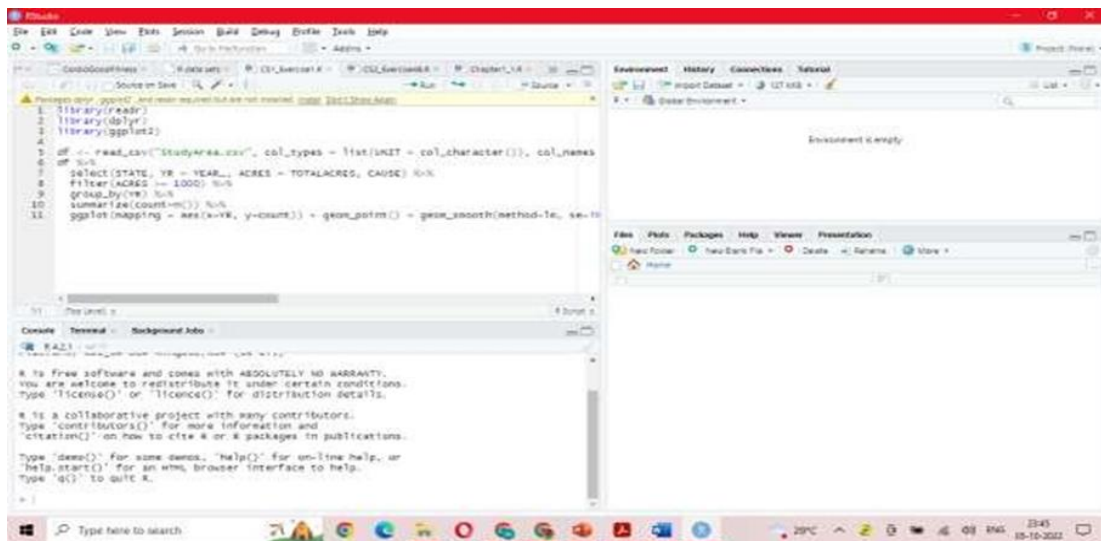
- The R Project for Statistical Computing, or simply named R, is a free software environment for statistical computing and graphics..

#### INTRODUCTION TO RSTUDIO

- RStudio is a free, open source IDE for R. It includes a console, syntax-highlighting editor that supports direct code execution, as well as tools for plotting, history, debugging and workspace management.

<https://www.rstudio.com/products/rstudio/>

#### THE RSTUDIO INTERFACE



- To simplify the overview of RStudio we'll break the IDE into quadrants to make it easier to reference each component of the interface.
- The screenshot above illustrates each of the quadrants.

#### ❖ Files Pane – (Q1)

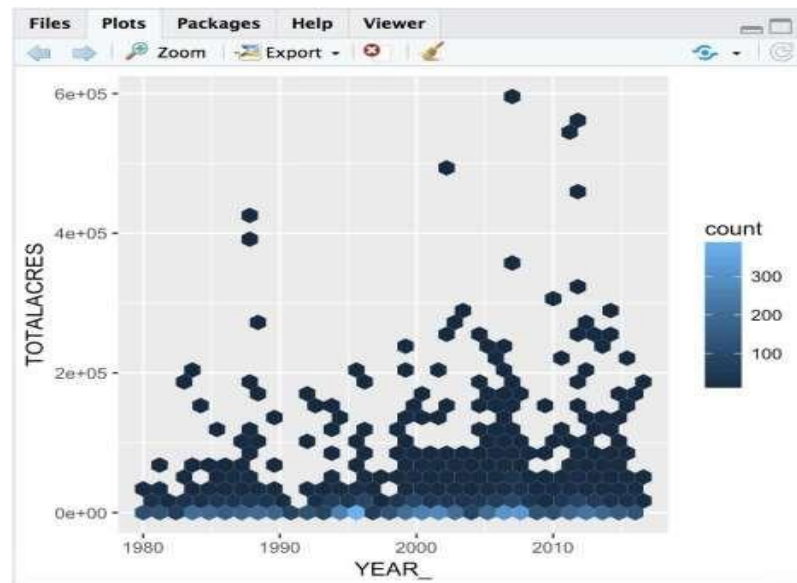
- The Files pane functions like a file explorer similar to Windows Explorer on a

Windows operating system or Finder on a Mac.

- This tab, displayed in the screenshot below, provides the following functionality:
  - i. Delete files and folders
  - ii. Create new folders
  - iii. Rename folders
  - iv. Folder navigation
  - v. Copy or move files
  - vi. Set working directory or go to working directory
  - vii. View files
  - viii. Import datasets

#### ❖ Plots Pane – (Q1)

- The Plotspane, displayed in the screenshot above, is used to view output visualizations produced when typing code into the Console window or running a script.
- Plots can be created using a variety of different packages, but we'll primarily be using the ggplot2 package.
- Once produced, it can be zoomed in, export as an image, or PDF, copy to the clipboard, and remove plots. It can also be navigated to previous and next plots.



#### ❖ Packages Pane – (Q1)

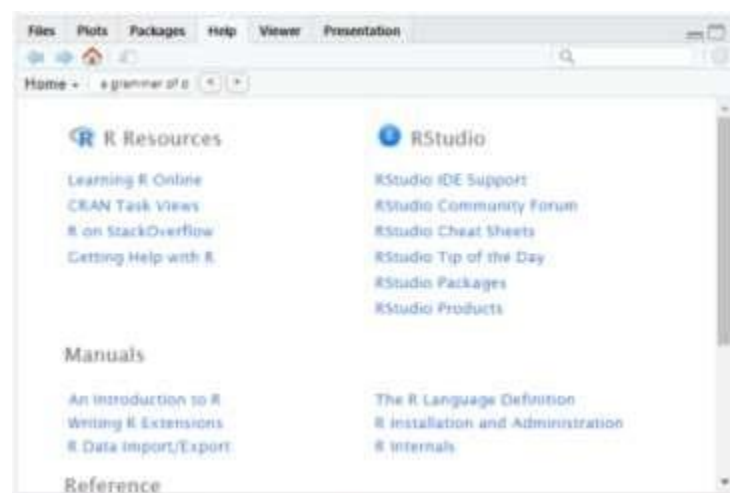
- The Packages pane, shown in the screenshot below, displays all currently installed packages along with a brief description and version number for the package.
- Packages can also be removed using the x icon to the right of the version number for the package.

- Clicking on the package name will display the help file for the package in the Help tab.
- Clicking on the checkbox to the left of the package name loads the library so that it can be used when writing code in the Console window

Files		Plots		Packages		Help		Viewer	
Install		Update		Packrat					
Name		Description				Version			
System Library									
<input type="checkbox"/>	acepack	ACE and AVAS for Selecting Multiple Regression Transformations				1.4.1			
<input type="checkbox"/>	assertthat	Easy Pre and Post Assertions				0.2.0			
<input type="checkbox"/>	backports	Reimplementations of Functions Introduced Since R-3.0.0				1.1.2			
<input type="checkbox"/>	base64enc	Tools for base64 encoding				0.1-3			
<input type="checkbox"/>	BH	Boost C++ Header Files				1.66.0-1			
<input type="checkbox"/>	bindr	Parametrized Active Bindings				0.1.1			
<input checked="" type="checkbox"/>	bindrcpp	An 'Rcpp' Interface to Active Bindings				0.2			
<input checked="" type="checkbox"/>	bitops	Bitwise Operations				1.0-6			
<input type="checkbox"/>	boot	Bootstrap Functions (Originally by Angelo Canty for S)				1.3-20			
<input type="checkbox"/>	broom	Convert Statistical Analysis Objects into Tidy Data Frames				0.4.3			
<input type="checkbox"/>	callr	Call R from R				2.0.2			
<input type="checkbox"/>	caTools	Tools: moving window statistics, GIF, Base64, ROC AUC, etc.				1.17.1			
<input type="checkbox"/>	cellranger	Translate Spreadsheet Cell Ranges to Rows and Columns				1.1.0			
<input type="checkbox"/>	checkmate	Fast and Versatile Argument Checks				1.8.5			

### ❖ Help Pane – (Q1)

- The Help pane, shown in the screenshot below, displays linked help documentation for any packages that you have installed.

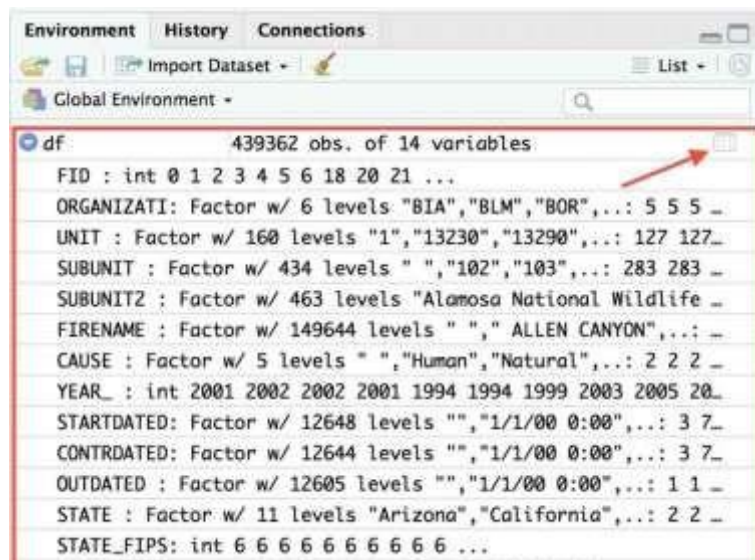


### ❖ Viewer Pane – (Q1)

- RStudio includes a Viewerpane that can be used to view local web content.
- For example, web graphics generated using packages like googleVis, htmlwidgets, and RCharts, or even a local web application created with Shiny.
- However, keep in mind that the Viewer pane can only be used for local web content in the form of static HTML pages written in the session's temporary directory or a locally run web application.
- The Viewer pane can't be used to view online content.

### ❖ Environment Pane – (Q2)

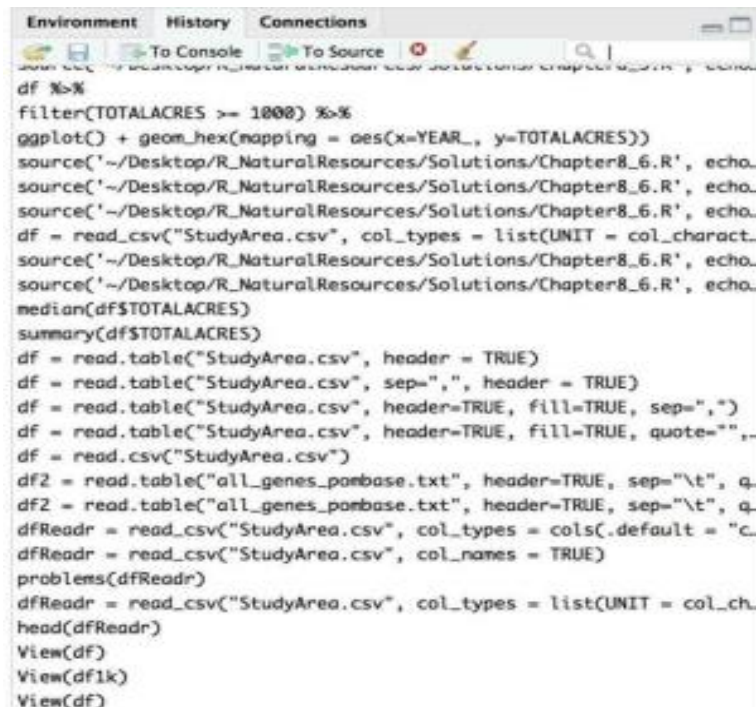
- The Environment pane contains a listing of variables that you have created for the current session.
- Each variable is listed in the tab and can be expanded to view the contents of the variable.
- You can see an example of this in the screenshot below by taking a look at the df variable.
- The rectangle surrounding the df variable displays the columns for the variable



### ❖ History Pane – (Q2)

- The History pane displays a list of all commands that have been executed in the current session.
- This tab includes a number of useful functions including the ability to save these commands to a file or load historical commands from an existing file.
- You can also select specific commands from the History tab and send them directly to the console or an open script.

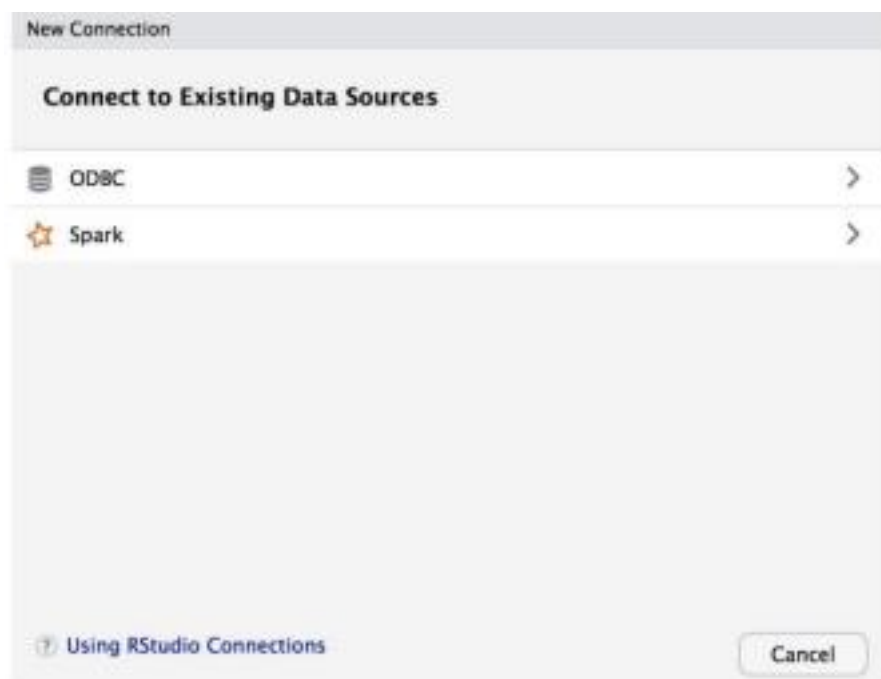
- You can also remove items from the History pane.



```
df %>%
  filter(TOTALACRES >= 1000) %>%
  ggplot() + geom_hex(mapping = aes(x=YEAR_, y=TOTALACRES))
source('~/.Desktop/R_NaturalResources/Solutions/Chapter8_6.R', echo=
source('~/.Desktop/R_NaturalResources/Solutions/Chapter8_6.R', echo=
source('~/.Desktop/R_NaturalResources/Solutions/Chapter8_6.R', echo=
df = read_csv("StudyArea.csv", col_types = list(UNIT = col_charact_
source('~/.Desktop/R_NaturalResources/Solutions/Chapter8_6.R', echo=
source('~/.Desktop/R_NaturalResources/Solutions/Chapter8_6.R', echo=
median(df$TOTALACRES)
summary(df$TOTALACRES)
df = read.table("StudyArea.csv", header = TRUE)
df = read.table("StudyArea.csv", sep="," , header = TRUE)
df = read.table("StudyArea.csv", header=TRUE, fill=TRUE, sep="," )
df = read.table("StudyArea.csv", header=TRUE, fill=TRUE, quote="," ,
df = read.csv("StudyArea.csv")
df2 = read.table("all_genes_pombase.txt", header=TRUE, sep="\t", q_
df2 = read.table("all_genes_pombase.txt", header=TRUE, sep="\t", q_
dfReadr = read_csv("StudyArea.csv", col_types = cols(.default = "c_
dfReadr = read_csv("StudyArea.csv", col_names = TRUE)
problems(dfReadr)
dfReadr = read_csv("StudyArea.csv", col_types = list(UNIT = col_ch_
head(dfReadr)
View(df)
View(df1k)
View(df)
```

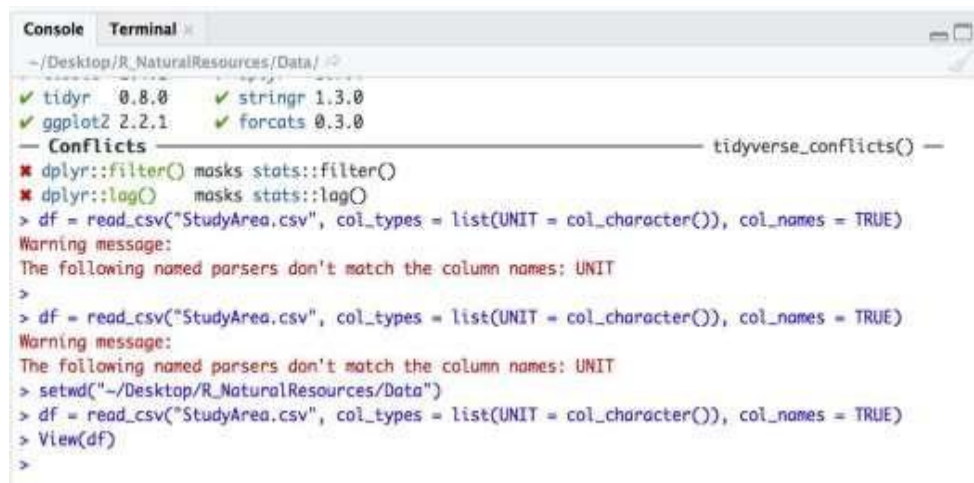
## ❖ Connections Pane – (Q2)

- The Connections tab can be used to access existing or create new connections to ODBC and Spark data sources.



### ❖ Source Pane – (Q3)

- The Source pane in RStudio is used to create scripts, and display datasets.
- An R script is simply a text file containing a series of commands that are executed together.
- Commands can also be written line by line from the Console pane as well.
- When written from the Console pane, each line of code is executed when you click the Enter (Return) key. However, scripts are executed as a group.
- Multiple scripts can be open at the same time with each script occupying a separate tab as seen in the screenshot.
- RStudio provides the ability to execute the entire script, only the current line, or a highlighted group of lines.
- This gives you a lot of control over the execution the code in a script.



```
Console Terminal x
~/Desktop/R_NaturalResources/Data/
✓ tidy 0.8.0 ✓ stringr 1.3.0
✓ ggplot2 2.2.1 ✓ forcats 0.3.0
— Conflicts — tidyverse_conflicts() —
✗ dplyr::filter() masks stats::filter()
✗ dplyr::log() masks stats::log()
> df = read_csv("StudyArea.csv", col_types = list(UNIT = col_character()), col_names = TRUE)
Warning message:
The following named parsers don't match the column names: UNIT
>
> df = read_csv("StudyArea.csv", col_types = list(UNIT = col_character()), col_names = TRUE)
Warning message:
The following named parsers don't match the column names: UNIT
> setwd("~/Desktop/R_NaturalResources/Data")
> df = read_csv("StudyArea.csv", col_types = list(UNIT = col_character()), col_names = TRUE)
> View(df)
>
```

### ❖ Console Pane – (Q4)

- The Console pane in RStudio is used to interactively write and run lines of code.
- Each time we enter a line of code and click Enter(Return) it will execute that line of code.
- Any warning or error messages will be displayed in the Console window as well as output from print() statements.





```
1 df = read_csv("StudyArea.csv", col_types = list(UNIT = col_character()), col_names = TRUE)
2 #steps 1-3
3 df %>%
4   select(ORGANIZATI, STATE, YEAR_, TOTALACRES, CAUSE) %>%
5   filter(TOTALACRES >= 1000) %>%
6   ggplot() + geom_histogram(mapping = aes(x=TOTALACRES), binwidth=500)
7
8 #step 4
9 df %>%
10  count(cut_width(TOTALACRES, 500))
```

#### ❖ Terminal Pane – (Q4)

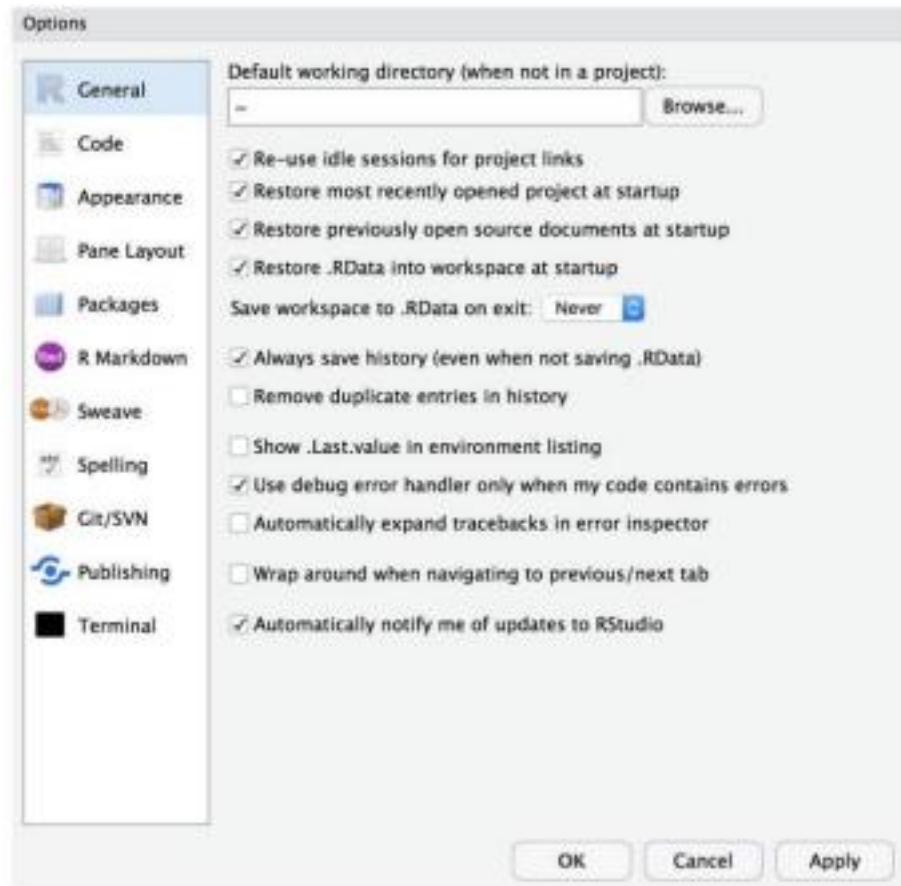
- The RStudio Terminal pane provides access to the system shell from within the RStudio IDE.
- It supports xterm emulation, enabling use of full-screen terminal applications (e.g. text editors, terminal multiplexers) as well as regular command-line operations with line editing and shell history.
- There are many potential uses of the shell including advanced source control operations, execution of long-running jobs, remote logins, and system administration of RStudio.
- The Terminal pane is unlike most of the other features found in RStudio in that it's capabilities are platform specific.
- In general, these differences can be categorized as either Windows capabilities or other (Mac, Linux, Rstudio Server).



```
Terminal 1 ~ /Desktop/WildfireApp
Eric's-MacBook-Pro:WildfireApp ericpimpler$ pwd
/Users/ericpimpler/Desktop/WildfireApp
Eric's-MacBook-Pro:WildfireApp ericpimpler$
```

#### CUSTOMIZING THE INTERFACE

- If we don't like the default RStudio interface, we can customize the appearance.
- To do so, go to Tool | Options (RStudio | Preferences on a Mac).
- The dialog seen in the screenshot below will be displayed.



- The Pane Layout tab is used to change the locations of console, source editor, and tab panes, and set which tabs are included in each pane.

## INSTALLING RSTUDIO

- Go to <https://www.rstudio.com/products/rstudio/download/> find RStudio for Desktop, the Open Source License version, and follow in the instructions to download and install the software.

### Exercise 1: Creating variables and assigning data

- In the R programming language, like other languages, variables are given a name and assigned data.
- Each variable has a name that represents its area in memory.
- In R, variables are case sensitive so use care in naming your variable and referring to them later in your code.
- There are two ways that variables can be assigned in R.
- In the first code example below, a variable named x is created.
- The use of a less than sign immediately followed by a dash then precedes the variable name.

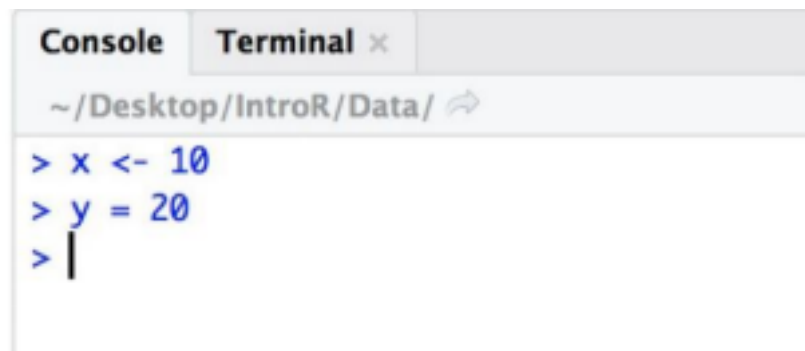


- This is the operator used to assign data to a variable in R. On the right-hand side of this operator is the value being assigned to the variable.
- In this case, the value 10 has been assigned to the variable x.
- To print the value of a variable in R you can simply type the variable name and then click the Enter key on your keyboard.

```
x <- 10
x
[1] 10
```

- The other way of creating and assigning data to a variable is to use the equal sign.
- In the second code example we create a variable called y and assign the value 10 to the variable.
- This second method of creating and assigning data to a variable is probably more familiar to you if you've used other languages like Python or JavaScript

```
y = 10
y
[1] 10
```



The screenshot shows an R console window with two tabs: 'Console' and 'Terminal'. The 'Console' tab is active, and the path '~/Desktop/IntroR/Data/' is displayed at the top. The console shows the following commands and output:

```
> x <- 10
> y = 20
> |
```

- Variables in R are case sensitive.
- To illustrate this, create a new variable called myName and assign it the value of your name as I have done in the screenshot below.
- In this case, since we've enclosed the value with quotes, R will assign it as a character (string) data type.
- Any sequence of characters, whether they be letters, numbers, or special characters, will be defined as a character data type if surrounded by quotes.



```
Console Terminal x
~/Desktop/IntroR/Data/
> myName = "Eric"
> myName
[1] "Eric"
> myname
Error: object 'myname' not found
```

- To see a list of all variables in your current workspace you can type the ls() function.



```
Source Console Terminal
~/Desktop/IntroR/Data/
> ls()
[1] "myName" "x" "y" "z"
>
```

Values	
myName	"Eric"
x	10
y	20
z	30

## Exercise 2: Using vectors and factors

- In R, a vector is a sequence of data elements that have the same data type.
  - Vectors are used primarily as container style variables used to hold multiple values that can then be manipulated or extracted as needed.
  - For example, all the values must be numeric, character, or Boolean.
  - You can't include any sort of combination of data types.
  - **To create a vector in R you call the c() function and pass in a list of values of the same type.**
  - After creating a vector there are a number of ways that you can examine, manipulate, and extract data.
1. Open RStudio and find the Console pane. It should be on the left-hand side of your screen at the bottom.
  2. In the R Console pane create a new vector as seen in the code example below.
  3. The c() function is used to create the vector object. This vector is composed of character data types. Remember that all values in the vector must be of the same data type.

```
layers<-c('parcels','streets','railroads','streams','buildings')
```

4. Get the length of the vector using the length() function. This should return a value of 5.

```
length(layers)
```

**[1] 5**

5. You can retrieve individual items from a vector by passing in an index number. Retrieve the Railroads value by passing in an index number of 3, which corresponds to the positional order of this value. R is a 1 based language so the first item in the list occupies position 1.

```
layers[3] [1]  
[1] "railroads"
```

6. We can extract a contiguous sequence of values by passing in two index numbers as seen below.

```
layers[3:5]  
[1] "Railroads" "Streams" "Buildings"
```

7. Values can be removed from a vector by passing in a negative integer as seen below. This will remove Streams from the vector.

```
> layers  
[1] "parcels" "streets" "railroads" "streams" "buildings"  
> layers[-4]  
[1] "parcels" "streets" "railroads" "buildings"
```

8. Create a second vector containing numbers as seen below.

```
layerIds <- c(1,2,3,4)
```

9. In this next step we're going to combine the layers and layerIds vectors into a single vector.

```
layerIds <- c(1,2,3,4)  
combinedVector <- c(layers, layerIds)  
combinedVector  
[1] "Parcels" "Streets" "Railroads" "Streams" "Buildings" [6] "1" "2" "3" "4"
```

10. Now let's create two new sets of vectors to see how vector arithmetic works. Add the following lines of code.

```
x <- c(10,20,30,40,50)  
y <- c(100,200,300,400,500)
```

Now add the values of the vectors.

```
x + y  
[1] 110 220 330 440 550
```

11. Subtract the values.

```
y - x
[1] 90 180 270 360 450
```

12. Multiply the values.

```
10 * x
[1] 100 200 300 400 500
20 * y
[1] 2000 4000 6000 8000 10000
```

13. We can also use the built in R function against the values of a vector. Enter the follow lines of codes to see how the built-in functions work.

```
> sum(x)
[1] 150
> mean(x)
[1] 30
> mean(y)
[1] 300
> median(y)
[1] 300
> max(x)
[1] 50
> min(y)
[1] 100
```

14. Now let's talk about ordering of factors. There may be times when you want to order the output of the factor. For example, you may want to order the results by month. Enter the following code:

**The two steps to creating a factor are:**

- a) Creating a vector
- b) Converting the vector created into a factor using function factor()

**Example:**

```
# Creating a vector
x <- c("female", "male", "male", "female")
print(x)
# Converting the vector x into a factor
gender <- factor(x)
print(gender)
Output:
```

**[1] female male male female**  
**Levels: female male**

### **Exercise 3: Using lists**

- A list is an ordered collection of elements, in many ways very similar to vectors. However, there are some important differences between a list and a vector.
- With lists we can include any combination of data types.
- This differs from other data structures like vectors, matrices, and factors which must contain the same data type.
- Lists are highly versatile and useful data types.
- A list in R acts as a container style object in that it can hold many values that you store temporarily and pull out as needed.
- Lists can be created through the use of the list() function.

#### **Example:**

```
mylist<-list("abc",2000,5000,TRUE,FALSE)  
> mylist  
[[1]]  
[1] "abc"  
  
[[2]]  
[1] 2000  
  
[[3]]  
[1] 5000  
  
[[4]]  
[1] TRUE  
  
[[5]]  
[1] FALSE
```

**We can get the number of items in a list by calling the length() function. length(mylist)**

**[1] 5**

- Finally, there may be times when you are uncertain if a variable is stored as a vector or a list.
- We can use the is.list() function, which will return a TRUE or FALSE value that indicates whether the variable is a list object.

```
> is.list(mylist)
```

**[1] TRUE**

#### **Exercise 4: Using data classes**

- In this, we'll take a look at matrices and data frames.
- A matrix in R is a structure very similar to a table in that it has columns and rows. This type of structure is commonly used in statistical operations.
- A matrix is created using the `matrix()` function. The number of columns and rows can be passed in as arguments to the function to define the attributes and data values of the matrix.
- A matrix might be created from the values found in the attribute table of a feature class. However, keep in mind that all the values in the matrix must be of the same data type.
- Data frames in R are very similar to tables in that they have columns and rows.
- This makes them very similar to matrix objects as well. In statistics, a dataset will often contain multiple variables.
- In the R Console create a new matrix as seen in the code example below.
- The `c()` function is used to define the data for the object. This matrix is composed of numeric data types. Remember that all values in the matrix must be of the same data type.

```
> matrx <- matrix(c(2,4,3,1,5,7), nrow=2, ncol=3, byrow=TRUE)
```

```
> matrx
```

```
[,1] [,2] [,3]
```

```
[1,] 2 4 3
```

```
[2,] 1 5 7
```

```
> matrx <- matrix(c(2,4,3,1,5,7), nrow=2, ncol=3, byrow=FALSE)
```

```
> matrx
```

```
[,1] [,2] [,3]
```

```
[1,] 2 3 5
```

```
[2,] 4 1 7
```

- Now let's retrieve a value from the matrix with the code you see below:

```
>matrx[2,3]
```

```
[1] 7
```

- We can use the `colSums()`, `colMeans()` or `rowSums()` functions against the data as well.

```
> colSums(matrx)
```

```
[1] 6 4 12
```

```
> colMeans(matrx)
```

```
[1] 3 2 6
```



```
> rowSums(matrx)
[1] 10 12
```

```
> rowMeans(matrx)
[1] 3.333333 4.000000
```

- Now we'll turn our attention to Data Frames. Clear the R console and execute the `data()` function as seen below. This displays a list of all the sample datasets that are part of R. You can use any of these datasets.
- We'll use the `USArrests` data frame. Add the code you see below to display the contents of the `USArrests` data frame.

```
> data()
> data(USArrests)
> USArrests
```

```
Murder Assault UrbanPop Rape
Alabama 13.2 236 58 21.2
Alaska 10.0 263 48 44.5
Arizona 8.1 294 80 31.0
Arkansas 8.8 190 50 19.5
California 9.0 276 91 40.6
Colorado 7.9 204 78 38.7
Connecticut 3.3 110 77 11.1
Delaware 5.9 238 72 15.8
Florida 15.4 335 80 31.9
Georgia 17.4 211 60 25.8
Hawaii 5.3 46 83 20.2
Idaho 2.6 120 54 14.2
Illinois 10.4 249 83 24.0
```

- Next, we'll pull out the data for all rows from the `Assault` column.

```
> USArrests$Assault

[1] 236 263 294 190 276 204 110 238 335 211 46 120 249 113 56 115 109
[18] 249 83 300 149 255 72 259 178 109 102 252 57 159 285 254 337 45
[35] 120 151 159 106 174 279 86 188 201 120 48 156 145 81 53 161
```

- A value from a specific row, column combination can be extracted using the code seen below where the row is specified as the first offset and the column is the second.

```
> USArrests[50,2]
[1] 161
```

```
>USArrests[50,]
```

**Murder Assault UrbanPop Rape**  
**Wyoming 6.8 161 60 15.6**

- In RStudio go to the **File menu and select Import Dataset | From Text (readr)** .
- This will display the dialog seen in the screenshot below.
- This package is used to efficiently read external data into a data frame.
- Use the Browse button to browse to the StudyArea.csv file found in the Data folder where you installed the exercise data for this book.
- Click Import from this Import Test Data dialog. This will load the data into a data frame (technically called a Tibble in tidyverse) called StudyArea.
- It will also use the View() function to display the results in a tabular view displayed in the screenshot below:

### Exercise 5: Looping statements

#### ❖ For loops

- For loops are used when you know exactly how many times to repeat a block of code.
- This includes the use of data frame objects that have a specific number of rows.
- For loops are typically used with vector and data frame structures.

**Add the following lines of code:**

```
for (fire in 1:nrow(StudyArea))  
{  
  print(StudyArea[fire, "TOTALACRES"])  
}
```

- Run the code by selecting **Code | Run Region | Run All** from the RStudio menu or by clicking the Source button on the script tab.
- This will produce a stream of data that looks similar to what you see below.

```
# A tibble: 1 x 1 TOTALACRES  
<dbl>  
1 0.100  
# A tibble: 1 x 1  
TOTALACRES  
<dbl>  
1 3.  
# A tibble: 1 x 1
```

```
TOTALACRES
<dbl>
1 0.500
# A tibble: 1 x 1
```

Another simple example:

```
week<-c('Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday',
'Saturday')
for (day in week)
{
  # displaying each string in the vector
  print(day)
}
```

#### Exercise 6: Decision support statements – if | else

- Decision support statements enable to write code that branches based upon specific conditions.
- The basic if | else statement in R is used for decision support.
- Basically, if statements are used to branch code based on a test expression.
- If the test expression evaluates to TRUE, then a block of code is executed.
- If the test evaluates to FALSE then the processing skips down to the first else if statement or an else statement if you don't include any else if statements.
- Add the if | else if block. This script loops through all the rows in the StudyArea data frame and prints out messages that indicate when a fire has burned more than the specified number of acres for each category.

```
for (fire in 1:nrow(StudyArea))
{
  if(StudyArea[fire, "TOTALACRES"] > 100000)
  {
    print(paste("100K Fire: ", StudyArea[fire, "FIRENAME"], sep = ""))
  }
  else if (StudyArea[fire, "TOTALACRES"] > 75000)
  {
    print(paste("75K Fire: ", StudyArea[fire, "FIRENAME"], sep = ""))
  }
  else if (StudyArea[fire, "TOTALACRES"] > 50000)
  {
    print(paste("50K Fire: ", StudyArea[fire, "FIRENAME"], sep = ""))
  }
}
```

}

- Run the code by selecting Code | Run Region | Run All from the RStudio menu or by clicking the Source button on the script tab.
- The script should start producing output in the Console pane similar to what you see below.

```
[1] "50K Fire: PIRU"  
[1] "100K Fire: CEDAR"  
[1] "50K Fire: MINE"  
[1] "100K Fire: 24 COMMAND"  
[1] "50K Fire: RANCH"  
[1] "75K Fire: HARRIS"  
[1] "50K Fire: SUNNYSIDE TURN OFF" [1] "100K Fire: Range 12"
```

### Exercise 7: Using functions

- Functions are a group of statements that execute as a group and are action oriented structures in that they accomplish some sort of task.
- Input variables can be passed into functions through what are known as parameters.
- Another name for parameters is arguments.
- These parameters become variables inside the function to which they are passed.

```
function_name <- function(arg_1, arg_2, ...)  
{  
  Function body  
}
```

### Exercise 8: Introduction to tidyverse

- While the base R package includes many useful functions and data structures that we can use to accomplish a wide variety of data science tasks, the thirdparty tidyverse package supports a comprehensive data science workflow as illustrated in the diagram below.
- The tidyverse ecosystem includes many subpackages designed to address specific components of the workflow.
- Tidyverse is a coherent system of packages for importing, tidying, transforming, exploring, and visualizing data.
- The packages of the tidyverse ecosystem were mostly developed by Hadley Wickham, but they are now being expanded by several contributors.
- Tidyverse packages are intended to make statisticians and data scientists more productive by guiding them through workflows that facilitate communication, and result in reproducible work products.
- Fundamentally, the tidyverse is about the connections between the tools that make the workflow possible.

Let's briefly discuss the core packages that are part of tidyverse:

### **readr**

- The goal of readr is to facilitate the import of file-based data into a structured data format.
- The readr package includes seven functions for importing file-based datasets including csv, tsv, delimited, fixed width, white space separated, and web log files.
- Data is imported into a data structure called a tibble.
- Tibbles are the tidyverse implementation of a data frame.
- They are quite similar to data frames, but are basically a newer, more advanced version.
- However, there are some important differences between tibbles and data frames.
- Tibbles never convert data types of variables.
- They never change the names of variables or create row names.
- Tibbles also have a refined print method that shows only the first 10 rows, and all columns that will fit on the screen. Tibbles also print the column type along with the name.

### **tidyr**

- Data tidying is a consistent way of organizing data in R, and can be facilitated through the tidyr package.
- There are three rules that we can follow to make a dataset tidy.
- First, each variable must have its own column.
- Second, each observation must have its own row, and finally, each value must have its own cell.

### **dplyr**

- The dplyr package is a very important part of tidyverse.
- It includes five key functions for transforming your data in various ways.
- These functions include filter(), arrange(), select(), mutate(), and summarize().
- In addition, these functions all work very closely with the groupby() function.
- All five functions work in a very similar manner where the first argument is the data frame you're operating on, and the next N number of arguments are the variables to include.
- The result of calling all five functions is the creation of a new data frame that is a transformed version of the data frame passed to the function.

### **ggplot2**

- The ggplot2 package is a data visualization package for R, created by Hadley Wickham in 2005 and is an implementation of Leland Wilkinson's Grammar of Graphics.
- Grammar of Graphics is a term used to express the idea of creating individual blocks that are combined into a graphical display.
- The building blocks used in ggplot2 to implement the Grammar of Graphics include data,

aesthetic mapping, geometric objects, statistical transformations, scales, coordinate systems, position adjustments, and faceting.

- Using ggplot2, we can create many different kinds of charts and graphs including bar charts, box plots, violin plots, scatterplots, regression lines, and more.
- There are a number of advantages to using ggplot2 versus other visualization techniques available in R.
- These advantages include a consistent style for defining the graphics, a high level of abstraction for specifying plots, flexibility, a built-in theming system for plot appearance, mature and complete graphics system, and access to many other ggplot2 users for support.

### **Other tidyverse packages**

- The tidyverse ecosystem includes a number of other supporting packages including stringr, purr, forcats, and others.

## **II THE BASICS OF DATA EXPLORATION**

In this heading we'll cover the following topics:

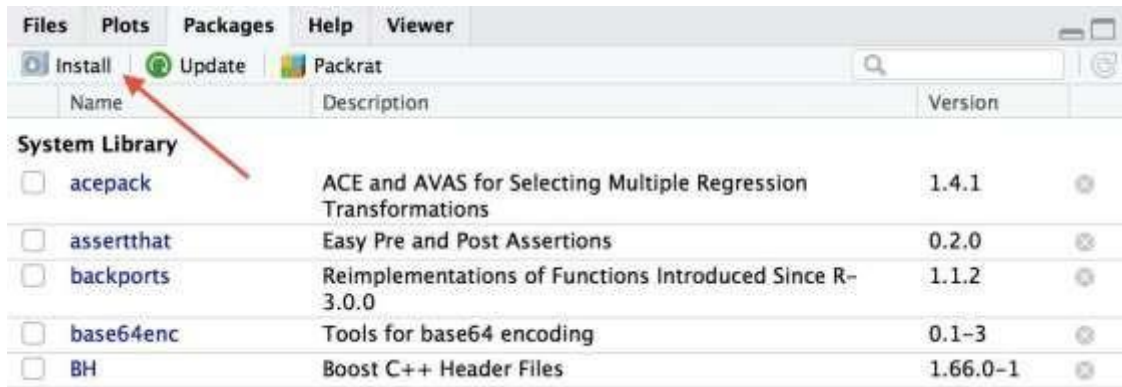
- Installing and loading tidyverse
- Loading and examining a dataset
- Filtering a dataset
- Grouping and summarizing a dataset
- Plotting a dataset

### **EXERCISE 1: INSTALLING AND LOADING TIDYVERSE**

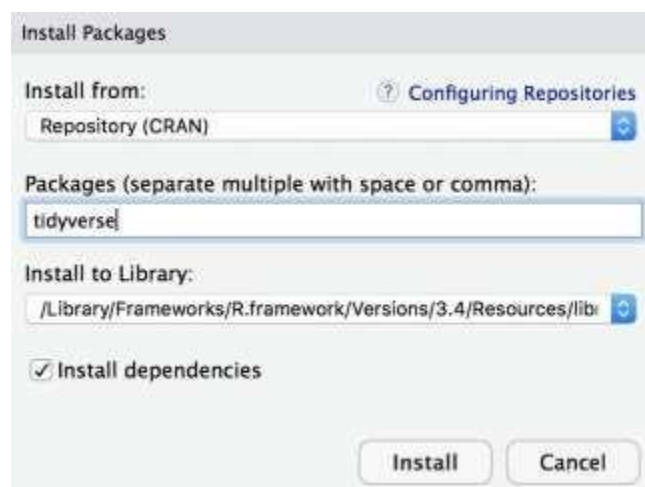
- Open RStudio.
- The tidyverse package is really more an ecosystem of packages that can be used to carry out various data science tasks.
- When you install tidyverse it installs all of the packages that are part of tidyverse, many of which we discussed in the last topic. Alternatively, you can install them individually as well. There are a couple ways that you can install packages in RStudio.
- Locate the Packages pane in the lower right portion of the RStudio window

To install a new package using this pane, click the Install button shown in the screenshot below





- In the Packages textbox, type tidyverse.
- Alternatively, you can load the packages individually so instead of typing tidyverse you would type readr or ggplot2 or whatever package you want to install.
- We're going to use the readr, dplyr, and ggplot2 packages in this chapter and in many others so you can either install the entire tidyverse package, which includes the packages



- The other way of installing packages is to use the `install.packages()` function as seen below. This function should be types from the Console pane.

**`install.packages(<package>)`**

For example, if you wanted to install the dplyr package you would type:

**`install.packages("dplyr")`**

- To use the functionality provided by a package it also needs to be loaded either into an individual script that will use the package, or it can also be loaded from the Packages pane.

- To load a package from the Packages pane, simply click the checkbox next to the package as seen in the screenshot below

	Name	Description	Version
<input type="checkbox"/>	RColorBrewer	ColorBrewer Palettes	1.1-2
<input type="checkbox"/>	Rcpp	Seamless R and C++ Integration	0.12.16
<input type="checkbox"/>	RCurl	General Network (HTTP/FTP/...) Client Interface for R	1.95-4.10
<input checked="" type="checkbox"/>	readr	Read Rectangular Text Data	1.1.1
<input type="checkbox"/>	readxl	Read Excel Files	1.0.0
<input type="checkbox"/>	rematch	Match Regular Expressions with a Nicer 'API'	1.0.1
<input type="checkbox"/>	reprex	Prepare Reproducible Example Code for Sharing	0.1.2
<input type="checkbox"/>	reshape	Flexibly Reshape Data	0.8.7
<input type="checkbox"/>	reshape2	Flexibly Reshape Data: A Reboot of the Reshape Package	1.4.3
<input checked="" type="checkbox"/>	rgdal	Bindings for the 'Geospatial' Data Abstraction Library	1.3-1
<input type="checkbox"/>	rgeos	Interface to Geometry Engine - Open Source ('GEOS')	0.3-26

- You can also load a package from either a script or the Console pane by typing **library(<package>)**.
- For example, to load the readr package you would type the following:

**library(readr)**

## EXERCISE 2: LOADING AND EXAMINING A DATASET

- The tidyverse package is designed to work with data stored in an object called a Tibble.
  - Tibbles are the tidyverse implementation of a data frame. They are quite similar to data frames, but are basically a newer, more advanced version.
  - There are some important differences between tibbles and data frames.
  - Tibbles never convert the data types of variables. Also, they never change the names of variables or create row names.
  - Tibbles also have a refined print method that shows only the first 10 rows, and all columns that will fit on the screen.
  - Tibbles also print the column type along with the name.
  - Getting data into a tibble object for manipulation, analysis, and visualization is normally accomplished through the use of one of the read functions found in the readr package.
  - In this exercise you'll learn how to read the contents of a CSV file into R using the read\_csv() function found in the readr package.
1. Open R Studio.
  2. In the Packages pane scroll down until you see the readr package and check the box just to the left as seen below as seen in the screenshot from the last exercise in this chapter.

3. You will also need to set the working directory for the RStudio session. The easiest way to do this is to go to Session | Set Working Directory | Choose Directory and then navigate to the IntroR\Data folder where you installed the exercise data for this book.
4. The `read_csv()` function is going to be used to read the contents of a file called `Crime_Data.csv`. This file contains approximately 481,000 crime reports from Seattle, WA covering a span of approximately 10 years. If you have Microsoft Excel or some other spreadsheet type software take a few moments to examine the contents of this file.

For each crime offense this file includes date and time information, crime categories and description, police department information including sector, beat, and precinct, and neighborhood name.

5. Find the RStudio Console pane and add the code you see below. This will read the data stored in the `Crime_Data.csv` file into a data frame (actually a tibble as discussed in the introduction) called `dfCrime`.

```
dfCrime = read_csv("Crime_Data.csv", col_names = TRUE)
```

6. You'll see some messages indicating the column names and data types for each as seen below.

Parsed with column specification:

```
cols( `Report Number` = col_double(),  
      `Occurred Date` = col_character(),  
      `Occurred Time` = col_integer(),  
      `Reported Date` = col_character(),  
      `Reported Time` = col_integer(),  
      `Crime Subcategory` = col_character(),  
      `Primary Offense Description` = col_character(),  
      Precinct = col_character(),  
      Sector = col_character(),  
      Beat = col_character(),  
      Neighborhood = col_character())
```

7. You can get a count of the number of records with the `nrow()` function.

```
nrow(dfCrime) [1] 481376
```

8. The `View()` function can be used to view the data in a tabular format as seen in the

screenshot below.

### View(dfCrime)

	Report Number	Occurred Date	Occurred Time	Reported Date	Reported Time	Crime Subcategory	Primary Offense Description
1	2.008e+13	12/13/1908	2114	12/13/2008	2114	DUI	DUI-LIQUOR
2	2.010e+13	06/15/1964	0	06/15/2010	1031	FAMILY OFFENSE-NONVIOLENT	CHILD-OTHER
3	2.012e+12	01/01/1973	0	01/25/2012	1048	SEX OFFENSE-OTHER	SEXOFF-OTHER
4	2.013e+13	06/01/1974	0	09/09/2013	1117	SEX OFFENSE-OTHER	SEXOFF-OTHER
5	2.016e+13	01/01/1975	0	08/11/2016	1054	SEX OFFENSE-OTHER	SEXOFF-OTHER
6	1.975e+12	12/16/1975	900	12/16/1975	1500	BURGLARY-RESIDENTIAL	BURGLARY-FORCE-R

9. It will often be the case that you don't need all the columns in the data that you import. The dplyr package includes a select() function that can be used to limit the fields in the data frame. In the Packages pane, load the dplyr library. Again, if you don't see the dplyr library then it (or the entire tidyverse) will need to be installed.

10. In this case we'll limit the columns to the following:

Reported Date, Crime Subcategory, Primary Offense Description, Precinct, Sector, Beat, and Neighborhood. Add the code you see below to accomplish this.

```
dfCrime=select (dfCrime, 'Reported Date', 'CrimeSubcategory', 'Primary Offense Description', 'Precinct', 'Sector', 'Beat', 'Neighborhood')
```

11. View the results.

### View(dfCrime)

12. You may also want to rename columns to make them more reader friendly or perhaps simplify the names. The select() function can be used to do this as well. Add the code you see below to see how this works. You simply pass in the new name of the column followed by an equal sign and then the old column name.

```
dfCrime = select(dfCrime, 'CrimeDate' = 'Reported Date', 'Category' = 'Crime Subcategory', 'Description' = 'Primary Offense Description', 'Precinct', 'Sector', 'Beat', 'Neighborhood')
```

## EXERCISE 3: FILTERING A DATASET

- In addition to limiting the columns that are part of a data frame, it's also common to subset or filter the rows using a where clause.
- Filtering the dataset enables you to focus on a subset of the rows instead of the entire dataset. The dplyr package includes a filter() function that supports this capability.
- In this exercise you'll filter the dataset so that only rows from a specific neighborhood are

included.

- ❖ In the RStudio Console pane add the following code. This will ensure that only crimes from the QUEEN ANNE neighborhood are included.

```
dfCrime2 = filter(dfCrime, Neighborhood == 'QUEEN ANNE')
```

- ❖ Get the number of rows and view the data if you'd like with the View() function.

```
nrow(dfCrime2) [1] 25172
```

- ❖ You can also include multiple expressions in a filter() function.
- ❖ For example, the line of code below would filter the data frame to include only residential burglaries that occurred in the Queen Anne neighborhood.
- ❖ There is no need to add the line of code below. It's just meant as an example.

```
dfCrime3 = filter(dfCrime, Neighborhood == 'QUEEN ANNE', Category ==  
'BURGLARY- RESIDENTIAL')
```

#### **EXERCISE 4: GROUPING AND SUMMARIZING A DATASET**

- The group\_by() function, found in the dplyr package, is commonly used to group data by one or more variables.
- Once grouped, summary statistics can then be generated for the group or you can visualize the data in various ways.
- For example, the crime dataset we're using in this chapter could be grouped by offense, neighborhood and year and then summary statistics including the count, mean, and median number of burglaries by year generated.
- It's also very common to visualize these grouped datasets in different ways.
- Barcharts, scatterplots, or other graphs could be produced for the grouped dataset. In this exercise you'll learn how to group data and produce summary statistics.

- ❖ In the RStudio console window add the code you see below to group the crimes by police beat.

```
dfCrime2 = group_by(dfCrime2, Beat)
```

- ❖ The n() function is used to get a count of the number of records for each group. Add the code you see below.

```
dfCrime2 = summarise(dfCrime2, n = n())
```

- ❖ Use the head() function to examine the results.head(dfCrime2)

```
# A tibble: 4 x 2 Beat n
```

```
<chr> <int>\
```

**1 D2 4373**

**2 Q1 88**

**3 Q2 10851**

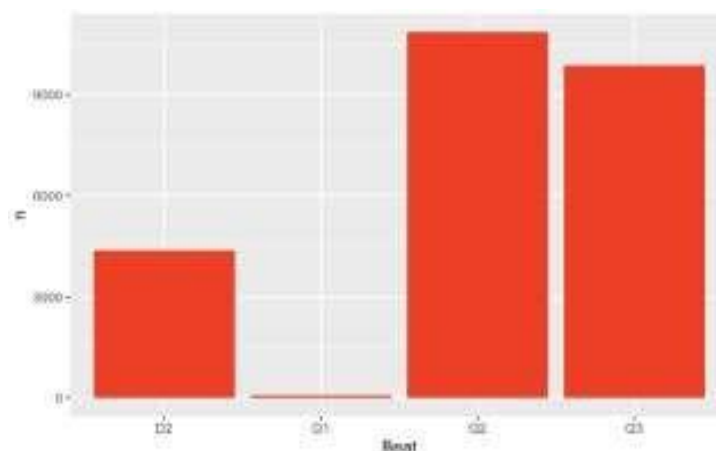
**4 Q3 9860**

## **EXERCISE 5: PLOTTING A DATASET**

- ❖ The ggplot2 package can be used to create various types of charts and graphs from a data frame.
- ❖ The ggplot() function is used to define plots, and can be passed a number of parameters and joined with other functions to ultimately produce an output chart.
- ❖ The first parameter passed to ggplot() will be the data frame you want to plot.
- ❖ Typically this will be a data frame object, but it can also be a subset of a data frame defined with the subset() function.
- ❖ The first code example on this slide passes a variable called housing, which contains a data frame. In the second code example, the subset() function is passed as the parameter.
- ❖ This subset function defines a filter that will include only rows where the State variable is equal to MA or TX.
  - In the RStudio console add the code you see below.
  - The ggplot() function in this case is passed the dfCrime data frame created in a previous exercises.
  - The geom\_col() function is used to define the geometry of the graph (bar chart) and is passed a mapping parameter which is defined by calling the aes() function and passing in the columns for the x axis (Beat), and the y axis (n = count).

**ggplot(data=dfCrime2) + geom\_col(mapping = aes(x=Beat, y=n), fill="red")**

- ✓ This will produce the chart you see below in the Plots pane.





## EXERCISE 6: GRAPHING BURGLARIES BY MONTH AND YEAR

- ✓ Create a couple barcharts that display the number of burglaries by year and by month for the Queen Anne neighborhood.
- ✓ In addition to the dplyr and ggplot2 packages we used previously, we'll also use the lubridate package to manipulate date information.
- ✓ In the RStudio Packages pane, load the lubridate package.
- ✓ The lubridate package is part of tidyverse and is used to work with dates and times. Also, make sure the readr, dplyr and ggplot2 packages are loaded.
  - ❖ Load the crime data from the Crime\_Data.csv file.

```
dfCrime = read_csv("Crime_Data.csv", col_names = TRUE)
```

- ❖ Specify the columns and column names.

```
dfCrime = select(dfCrime, 'CrimeDate' = 'Reported Date', 'Category' =  
'Crime Subcategory', 'Description' = 'Primary Offense Description',  
'Precinct', 'Sector', 'Beat', 'Neighborhood')
```

- ❖ Filter the records so that only residential burglaries in the Queen Anne neighborhood are retained.

```
dfCrime2 = filter(dfCrime, Neighborhood == 'QUEEN ANNE', Category ==  
'BURGLARY-RESIDENTIAL')
```

- ❖ The dplyr package includes the ability to dynamically create new columns in a data frame through the manipulation of data from existing columns in the data frame.
- ❖ The mutate() function is used to create the new columns.
- ❖ Here the mutate() function will be used to extract the year from the CrimeDate column.
- ❖ Add the following code to see this in action.
- ❖ The second parameter creates a new column called YEAR and populates it by using the year() function from the lubridate package.
- ❖ Inside the year() function the CrimeDate column, which is a character column, is converted to a date and the format of the date

```
dfCrime3 = mutate(dfCrime2, YEAR = year(as.Date(dfCrime2$CrimeDate,  
format='%m/%d/%Y')))
```

- ❖ View the result. Notice the YEAR column at the end of the data frame. The mutate() function always adds new columns to the end of the data frame.

```
View(dfCrime3)
```

meDate	Category	Description	Precinct	Sector	Beat	Neighborhood	YEAR
22/2008	BURGLARY-RESIDENTIAL	BURGLARY-NOFORCE-RES	WEST	Q	Q3	QUEEN ANNE	2008
02/2008	BURGLARY-RESIDENTIAL	BURGLARY-FORCE-RES	WEST	Q	Q3	QUEEN ANNE	2008
02/2008	BURGLARY-RESIDENTIAL	BURGLARY-FORCE-RES	WEST	D	D2	QUEEN ANNE	2008
07/2008	BURGLARY-RESIDENTIAL	BURGLARY-FORCE-RES	WEST	Q	Q3	QUEEN ANNE	2008
09/2008	BURGLARY-RESIDENTIAL	BURGLARY-NOFORCE-RES	WEST	Q	Q2	QUEEN ANNE	2008
10/2008	BURGLARY-RESIDENTIAL	BURGLARY-FORCE-RES	WEST	Q	Q3	QUEEN ANNE	2008
19/2008	BURGLARY-RESIDENTIAL	BURGLARY-NOFORCE-RES	WEST	Q	Q2	QUEEN ANNE	2008
14/2008	BURGLARY-RESIDENTIAL	BURGLARY-NOFORCE-RES	WEST	Q	Q3	QUEEN ANNE	2008
17/2008	BURGLARY-RESIDENTIAL	BURGLARY-FORCE-RES	WEST	Q	Q3	QUEEN ANNE	2008
17/2008	BURGLARY-RESIDENTIAL	BURGLARY-NOFORCE-RES	WEST	Q	Q2	QUEEN ANNE	2008
18/2008	BURGLARY-RESIDENTIAL	BURGLARY-FORCE-RES	WEST	Q	Q2	QUEEN ANNE	2008
31/2008	BURGLARY-RESIDENTIAL	BURGLARY-NOFORCE-RES	WEST	D	D2	QUEEN ANNE	2008
30/2008	BURGLARY-RESIDENTIAL	BURGLARY-FORCE-RES	WEST	Q	Q3	QUEEN ANNE	2008

- ❖ Now we'll group the data by year and summarize by getting a count of the number of crimes per year. Add the following lines of code.

```
dfCrime4 = group_by(dfCrime3, YEAR)
dfCrime4 = summarise(dfCrime4, n = n())
```

- ❖ View the result.

```
View(dfCrime4)
```

	YEAR	n
1	2008	205
2	2009	194
3	2010	181
4	2011	203
5	2012	170
6	2013	209
7	2014	216
8	2015	258
9	2016	220
10	2017	218
11	2018	119

- Create a bar chart by calling the ggplot() and geom\_col() functions as seen below.
- Define YEAR as the column for the x axis and the number of crimes for the y axis.

- This should produce the chart you see below in the Plots pane.

```
ggplot(data=dfCrime4) + geom_col(mapping = aes(x=YEAR, y=n), fill="red")
```

- Now we'll create another bar chart that displays the number of crimes by month instead of year.
- First, create a MONTH column using the mutate() function.

```
dfCrime3 = mutate(dfCrime2, MONTH = month(as.Date(dfCrime2$CrimeDate,format='%m/%d/%Y')))
```

- Group and summarize the data by month.

```
dfCrime4 = group_by(dfCrime3, MONTH) dfCrime4 = summarise(dfCrime4, n = n())
```

- View the result.

```
View(dfCrime4)
```

- Create the bar chart.

```
ggplot(data=dfCrime4) + geom_col(mapping = aes(x=MONTH, y=n), fill="red")
```

