**Primary components of R data structures**:

- Named Values 🡪 A variable given a name, akin to a variable that has been given a **SINGLE VALUE**

- Assigning a named value follows the pattern of:

x <- #

where x is the value’s name and the assignment command is “<-“ to give the named value the numerical value #.

***Note:*** all environment objects are mutable – meaning they can be assigned and reassigned multiple times throughout a script or file.

- Numeric Vectors 🡪 R’s version of arrays, a list of numbers assigned to a location and stored as a **SINGLE DATA STRUCTURE**

**-** To create a numeric vector, the line follows a pattern of:

numlist <- c(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)

where “c” stands for concatenate and links together all the values in the parentheses in a comma-separated format

*Note:* Objects that are not a named value will have their data type and dimensions listed in the environment pane.

- The example “numlist” above would have the environment display as

num [1:10]

for numeric object data type and dimensions of 1 row to 10 columns

- Matrix 🡪 Similar to a vector of vectors, with each value in the matrix required to have the **SAME DATA TYPE**, used more by older scripts and files than more recent ones – which use Data Frames and Tibbles more frequently

- Data Frame 🡪 Similar to a PANDAS DataFrame, where each column can be a **DIFFERENT DATA TYPE**

- Tibble 🡪 Introduced by the tidyverse package in R, the most recent data object structure, tibbles are an optimized data frame with extra metadata and features compared to basic data frames.

**Creating functions in R**:

- Similar to Python functions, they are denoted by a parentheses pair ( ) and follow a basic syntax pattern

function\_name <- function(arg1, arg2=T, …){

<FUNCTION BODY>

return <RETURN VALUE>

}

With these components, the basic pattern includes

a function’s name, the assignment operator, a declaration of function creation as “function”, the arguments listed in parentheses, the body of the function’s instructions, and the return command with the desired return information listed.

- The argument “arg1” is required to be input into the function’s operations

- Argument “arg2” is an optional such as declaring “exclaim=TRUE” for conditional argument

*Note:* Similar to Javascript, the function uses special brackets { } to enclose the function instead of colons as in Python syntax.

**Key R Commands and Shortcuts**:

- Function Help 🡪 Enter “?name\_of\_function\_in\_question” into the console for the documentation to open in the help pane

- Read CSV File 🡪 read.csv(file, header = TRUE, sep = “,”, quote = “\””, dec = “.”, fill = TRUE, comment.char = “”, …)

- Module’s indicated way to read CSV files:

named\_value <- read.csv(file=’file\_path/file\_name.csv’, check.names=F, stringsAsFactors = F)

- The function requires inputs of **file name**, **confirmation of header**, **separating symbol type¸** **quote symbol type**, **decimal symbol type**, **fill blanks for lacking row entries**, **comment character type**.

- Other Parameter Meanings: **Check.names=F** means separations in the header will be changed to a period instead of spaces, punctuation or other characters, **stringsAsFactors = F** means if a column is a string data type, it will be cast as a factor.

- Reading Other File Types 🡪 read.delim( ) [for tab-delimited files], read.table( ) [to manually indicate what the delimiter symbol is for the file to be used].

- Import JSONlite Package Into File 🡪 library(jsonlite)

- Read JSON File 🡪 fromJSON(txt=’file\_name.json’)

- Import TidyVerse Package Into File 🡪library(tidyverse)

- Determine If Data Frame Values Are All Equal 🡪 all.equal( )

- Retrieve Column Names In Data Frame 🡪 colnames( )

- View Headers In Dataset 🡪 head( dataset\_name )

**Selecting And Indexing In R**:

- Select Data: Easiset with the bracket notation such as in

> x <- c(3, 3, 2, 2, 5, 5)

> x[3]

[1] 2

- The selecting the third entry in the array gives the returned value of [1] 2, because it is an object of dimensions of “1” and the selected value is equal to “2”

*Note:* R’s index is base **1** *NOT* base **0**, unlike Python and Javascript, so indexing for position 3 means entering x[3]

- Selecting Data From Two-Dimensional Data Structures: Such as Matrices, Dataframes, Tibbles

- Using bracket notation, selecting data follows the **FORMAT**

object\_name[row, “column\_name”] ***or*** object\_name[row#, column#]

- Alternatively, use “$” operator on a two-dimensional object such as in:

data\_frame\_name$”Column\_Name”[#]

Where the column is named in string format after the “$” operator and bracket notation is used to index the vector for a single value.

*Note:* The vector does not need to be indexed – meaning you can select and retrieve the entire column/vector’s values or add further actions for that data.

**Filter A Dataset**:

- Use bracket notation and logic operators to select only data with matching criteria, such as:

filtered\_table <- table\_name[table\_name$numerical\_column > 1000,]

This code line filters the table for the column named “numerical\_column” and returns only rows that have data greater than 1,000 in the “numerical\_column” column.

- *Note*: A comma “,” follows the statement because it indicates the subset is created by rows. Adding columns to the statement after the comma specifies the columns to select from that subset.

(\* Best if used for simple logical statements)

- Alternatively, use the subset function for more complex logical statements in this pattern:

subset(x, subset, select, drop = FALSE, …)

This function uses arguments with 🡪- **x** indicates the data object to be referenced

- **subset** indicates the logical statements for which rows to keep

- **select** indicates the logical statements for which columns to keep

Module’s example function for demo data table:

# filter by price and drivetrain

filter\_table2 <- subset(demo\_table2, price > 10000 & drive == “4wd” & “clean” %in% title\_status)

Where, logic operators “&” and “==” are key parts of indicating which parameters to meet for data to be included in the filtered subset.

*Note*: Column names are not included as strings in quotes “ “ but can be referenced as simply their names. The value entries in those columns/vectors must be referenced as strings with quotes “ “, however.

**Extra:** Example of the same statement using brackets:

filter\_table2 <- demo\_table2[(“clean” %in% demo\_table2$title\_status) & (demo\_table2$price > 10000) & (demo\_table2$drive == “4wd”),]

- Sample function to create a sampled vector from a larger vector:

sample(x, size, replace = FALSE, prob = NULL)

With this function, the parameters are: **x** is the larger vector to select from, **size** indicates the number of data points to select from x, and **replace** is a flag for whether to allow or disallow the same values to be selected

- Sampling two-dimensional data structures require supplying the index of each row to sample, accomplished through 3 steps:

1. Create a numerical vector with the same length as the number of rows in the dataframe with the colon “:” operator

num\_rows <- 1:nrow(table\_name)

2. Use the sample( ) function to sample a list of indices from the first vector

sample\_rows <- sample(num\_rows, 3) 🡪 ex. to sample 3 rows

3. Use bracket notation to retrieve the data frame rows from the sampled list

table\_name[sample\_rows,]

*Combine these steps into a single statement in the form:*

table\_name[sample(1:nrow(table\_name), 3),]

**Data Transformation**:

- Mutate Data To Add New Variables and Preserve Existing Ones:

mutate(data) with module example:

demo\_table <- demo\_table %>% mutate(Mileage\_per\_Year=Total\_Miles/(2020-Year), IsActive=TRUE)

This line adds 2 columns to the “demo\_table” data table, with the parameters defined

- Transmute Data To Add New Variables and Drop Existing Ones:

transmute(data, …)

**Grouping Data**:

- Group\_by Function: similar to PANDAS’ groupby function, but needs a second step for creating a summary data frame. Using the format:

table\_name %>% group\_by(column\_name)

- Summarize Based on Groups: Second step in creating a table comparison for groupings in data frame. Uses format:

summarize(condition/value desired for comparison)

🡪 Combined, the module example function in one full line is:

summarize\_demo <- demo\_table2 %>% group\_by(condition) %>% summarize(Mean\_Mileage=mean(odometer), .groups=’keep’)

- *Note*: .groups has 4 possible parameters to be applied to the summarize( ) function:

- .groups = “drop\_last” – drops the last grouping level (default)

- .groups = “drop” – drops all grouping levels and returns a tibble

- .groups = “keep” – preserves the grouping of the input

- .groups = “rowwise” – turns each row into its own group

**Reshaping Data**:

- Gather Wide Dataset Into a Long Dataset: Compresses hard-to-read datasets into a more manageable long-form set. Uses format:

gather(data, key=”key\_entry”, value=”value\_entry”, …)

Where: **data** is the data frame to be reshaped, **key** is the name of the variable column that the wider data frame will collapse into, **value** is the name of the new value column derived from the original data and **“…”** indicates a list of columns to collapse into the key column

- Spread Out A Variable Column Into Columns For Each Variable:

spread(data, key, value, fill = NA, convert = FALSE, drop = TRUE, sep = NULL)

Where: **data** is the data frame to be reshaped, **key** is the name of the variable column to be spread out, **value** is the column to be filled with the new variable columns, **fill** is an optional argument that will set empty rows to the declared fill value.

**Using GGPlot2**:

- All figures use three main components in ggplot2:

1. ggplot function – to indicate what variables to use

ggplot(data = NULL, mapping = aes( ) )

Where: **data** is the input data frame and **mapping** uses the aesthetic aes( ) function to assign variables as dependent and independent

2. geom function – to indicate what plots to generate

The geom function has a very large number of arguments that can be passed into it, but often it can be left to successfully plot data using ONLY the function itself.

- *NOTE*: Functionally, both geom\_bar( ) and geom\_col( ) create bar charts, but they expect different inputs. For geom\_bar( ) – it expects **ONE** variable and generates frequency data. For geom\_col( ) – it expects **TWO** variables where the size of each category’s bar is also expected to be provided.

3. formatting or theme functions – to customize the plot

- Examples Of Functions For Creating Bar Plots:

- Module’s first example plot:

plt <- ggplot(mpg, aes(x=class))

🡪 # import dataset (mpg) into ggplot2

plt + geom\_bar()

🡪 # plot a bar plot

- A more complete list of functions for creating a plot:

mpg\_summary <- mpg %>% group\_by(manufacturer) %>% summarize(Vehicle\_Count=n(), .groups = ‘keep’)

🡪 # Create the summary table

plt <- ggplot(mpg\_summary, aes(x=manufacturer, y=Vehicle\_Count))

🡪 # import dataset into ggplot2

plt + geom\_col( )

🡪 # plot a bar plot

- Formatting Functions:

- To label the axes, the xlab( ) and ylab( ) functions can be applied with the label provided as a string argument when providing the line for generating the plot. For example:

plt + geom\_col( ) + xlab(“Manufacturing Company”) + ylab(“Number of Vehicles in Dataset”)

- To adjust the text in the labels, additional formatting can be applied by adding a “+” symbol after the above example and in the subsequent entry prompt line in the console add:

theme(axis.text.x=element\_text(angle=45, hjust=1))

- Using this line, the text element is selected for the x axis’ labels and the angle is adjusted by 45 degrees, and the horizontal justification is changed to 1 point. The same method can be for changing the **y** axis’ labels and adding the argument into the theme( ) function.

- Creating Line and Scatter Plots:

- A Comparison of a **Categorical** Variable and a **Continuous** Numerical Variable: a continuous variable is needed to show the progression of the data compared to the categorical variable in a line format.

- Plot with the geom function:

plt + geom\_line( )

- Adjust The Scale of the Plot With Scaling Functions: scale\_x\_discrete( ) and scale\_y\_continuous( ). For example use:

plt + geom\_line( ) + scale\_x\_discrete(limits=c(4,6,8)) + scale\_y\_continuous(breaks = c(15:30))

🡪 The **limits** argument for scale\_x\_discrete is necessary and creates tick marks for each value in the list. The **breaks** argument for scale\_y\_continuous is used to bound the y-axis to a defined range, in this case between 15 and 30, to provide easier-to-read formatting for the plot.

- Scatter Plot Creation: similar to line plot generation, with the **independent** variable set to the x values and the **dependent** variable set to the y values within the aes( ) function. For example, for a plot comparing engine size to fuel efficiency in cities, use:

plt <- ggplot(mpg,aes(x=displ,y=cty))

plt + geom\_point( ) + xlab(“Engine Size (L)”) + ylab(“City Fuel-Efficiency (MPG)”)

🡪 This plot visually compares two variables but scatter plots can be used to compare **MORE** than two variables.

- The aes( ) function can be adjusted for for several customizable arguments, such as: **alpha** for transparency of the data points, **color** to change the color of the data points, **shape** to change the shape for the point’s representation, and **size** which will adjust the size for each data point. Example from module:

plt <- ggplot(mpg,aes(x=displ, y=cty, color=class, shape=drv))

plt + geom\_point( ) + labs(x=”Engine Size (L)”, y=”City Fuel-Efficiency (MPG)”, color=”Vehicle Class”, shape=”Type of Drive”)

🡪 A key thing to *NOTE* is a column/vector can be used for reference in the customizing arguments and will be designated by the function for the differing logical groupings it identifies. The only thing needed is to declare the variable to the column/vector and to label the customization argument as such. Also, note labels can be applied in a more general format by parentheses ( ) around the designated variables that have been assigned in the ggplot function.

- Creating Boxplots: Use geom\_boxplot( ), module example for generating a boxplot:

plt <- ggplot(mpg,aes(y=hwy))

plt + geom\_boxplot( )

*NOTE*: The geom\_boxplot( ) function expects a **NUMERIC** vector for the y-value assignment due to ggplot accounting for multiple boxplots in a single figure. If a categorical grouping factor is assigned to x, a boxplot will be created to compare measurements from a variety of groups.

- For example, if the example functions used are:

plt <- ggplot(mpg, aes(x=manufacturer, y=hwy))

plt + geom\_boxplot( ) + theme(axis.text.x=element\_text(angle=45, hjust=1))

Then, the resulting plot would be a horizontally spread collection of boxplots comparing the highway fuel efficiency across the y-axis for each manufacturer represented by a tick on the x-axis.

- Creating A Heatmap Plot: Use function geom\_tile( ), with the module example using:

mpg\_summary <- mpg %>% group\_by(model, year) %>% summarize(Mean\_Hwy=mean(hwy), .groups = ‘keep’)

plt <- ggplot(mpg\_summary, aes(x=model, y=factor(year), fill=Mean\_Hwy))

plt + geom\_tile( ) + labs(x=”Model”, y=”Vehicle Year”, fill=”Mean Highway (MPG)”) +

> theme(axis.text.x = element\_text(angle=90, hjust=1, vjust=.5))

- Adding Layering To Plots: There are two types of layers – using the **same** variables and input data as the original plot, or using **different** **but complimentary** data compared to the original plot.

🡪 *Type 1*: For example, a boxplot can have a scatter plot added on top of it to give visual points that represent the key sections of the boxplot. For the boxplot example from the module above, simply add a plus “+” symbol after the theme adjustment section and add a geom ( ) function as:

geom\_point( )

- With this addition, the plot is generated and then a scatter plot is overlayed on top of the first plot with the same data as the first plot, since the line of code is effectively continuous with the appended sections included by the plus “+” symbols.

🡪 *Type 2*: Adding new data and variables into the geom function uses the **mapping** argument mentioned earlier that can be optional for generating plots. For example, adding error bars to a scatter plot for reference of potential inaccuracy would be done through:

mpg\_summary <- mpg %>% group\_by(class) %>% summarize(Mean\_Engine=mean(displ), SD\_Engine=sd(displ), .groups = ‘keep’)

plt <- ggplot(mpg\_summary, aes(x=class, y=Mean\_Engine))

plt + geom\_point(size=4) + labs(x=”Vehicle Class”, y=”Mean Engine Size”) +

geom\_errorbar(aes(ymin=Mean\_Engine-SD\_Engine, ymax=Mean\_Engine + SD\_Engine))

- These functions generate a summary that generates a scatter plot for the mean engine sizes vs. vehicle classes and adds an errorbar for the variation from one standard deviation on either side of the calculated mean value. This is effectively a simplified boxplot, using two different plot types.

*NOTE*: The argument fed into the geom\_point( ) function for “size=4” indicates the points mapped are given a **size** of 4, as opposed to the **default size** of 1 when the argument is not added.

- *Faceting Plots By Separating Out Plots For Each Level*: Using the ggplot2 function at the end of the plotting statement - facet( ) or facet\_wrap( ). For example, use:

plt + geom\_boxplot( ) + facet\_wrap(vars(MPG\_Type)) + theme(…, legend.position = “none”) + xlab(“Manufacturer”)

- This *shortened* (for simplicity’s sake, the plt function is omitted) section of code applies the facet wrapping function to compare the data based on MPG Type and gives two colors to compare the clearest two sections of performance in this area for the resulting plot:

(red for lower rating, blue for higher rating in the module’s case of color assignment)

The resulting plot has two layers with boxplots and coloring to indicate rating on the y-axis compared to the average.

*NOTE*: The xlab section is added at the end of the code line instead of after the plot generating function, showing the sections can be moved around based on priority.

**Statistical Analysis In R Scripts:**

- Qualitative testing For Normality: Plotting the distribution of a dataset can be done with the function:

geom\_density( ) 🡪 Added onto the ggplot function, density represents the y-axis values

- This function plots a numerical vector and the ggplot function would need the x-axis to represent the weights “wt” values for intervals of spread in the data (normalized data would have the ***highest density*** concentrated near the center of the ***weights*** on the x-axis.)

- Quantitative Testing For Normality: Testinmg for the p-value (probability-value) and W determination of a numeric vector, the function is applied as:

shapiro.test(x) 🡪 X represents a numeric vector of data values with at least 3 values

- The p-value result follows the trend: 0.05+ is considered normally distributed, and less than 0.05 indicates the distribution is abnormal.

- Handling Skewed Data: Two forms of skewed data exist – “left skew” and “right skew” for the trend of the center of distribution being shifted left or right, if the concentrated section in the distribution is to the right of the mean/median – it is “left skewed” and if it is to the left of the mean/median – it is “right skewed”. This indicates the larger section (and likely, less intense slope of the curve’s line) of the visible curve is more populated on the left or right side of the mean/median, meaning some extreme values are probably drawing attention to its side of the data distribution.

- *Large Dataset or Subtle Signs of Skewed Data*: The option is to simply point out there are signs of skewed data distribution when presenting the data findings. The expectation is the impact on the analysis should be minimal and can be presented while bringing attention to the possibility of slightly skewed data.

- *Smaller Dataset or Clear Impact From Skewed Data*: Action is needed for a usable analysis, and three options follow:

1.) Add more data points, if possible, to balance out the skew, though this is not a greatly reliable option and potentially might not even be possible to add data points or improve the distribution.

2.) Resample or regenerate the data if there is a potential of an error when drawing from the original conditions or dataset relating to the data acquisition.

3.) Transform the data using ***another numerical variable*** (making the data dependent on a different variable) or by using an ***operator*** (such as using a logarithmic transformation or exponentially adjusting the data). One of the easier methods is by using ln(dataset) [base e] or log10(dataset) [base 10], where the data is adjusted to have a less extreme impact to distribution from individual data values.

- Hypothesis Testing: To determine the probability of an event under certain conditions. Two types exist – ***null hypothesis*** and ***alternate hypothesis*** – where the potential outcome is explained as random chance, or is influenced by non-random events, respectively.

- *NOTE***:** Both are used to explain a certain outcome, where H0 indicates the event’s cause is **unsubstantial** and the outcome is ***insignificant*** and Ha indicates a **different** *and* **discernable** cause exists and the outcome is ***significant***. In either case, evidence is necessary and the key part of the conclusion.

- The five steps in Hypothesis Testing:

1.) Generate a null hypothesis, a corresponding alternate hypothesis and the significance level.

2.) Identify statistical analysis to assess the truth of the null hypothesis.

3.) Compute the p-value using statistical analysis.

4.) Compare p-value to the significance level from step 1.

5.) Reject the null hypothesis (or fail to reject it) and generate a conclusion.

- Significance Value Representation: The smaller a significance value is, the more important the findings and less likely the findings are insignificant. The module’s claims are “in most cases, a significance level of 0.05 is sufficient, but if our hypotheses are…for critical decision-making, we might want…smaller cutoffs such as 0.01 or 0.001.”

- Significance level of 0.1 = 1 in 10 and 0.001 is 1 in 1,000. Meaning 1.0 indicates a conclusion is wrong 1 in 1 times or entirely null.

- Sampling Functions: To gather a sample of the population data available for testing, two functions exist in the dplyr package – sample( ) and sample\_n( ). These functions follow the pattern:

sample\_n(tbl, size, replace = FALSE, weight = NULL)

🡪 With **tbl** being the input table (data frame’s name) and **size** is the number of rows to return.

*NOTE*: If the data frame input was grouped using group\_by( ), the size argument

is the number of groups to be returned.

One example from the module:

sample\_table <- population\_table %>% sample\_n(50)

plt <- ggplot(sample\_table, aes(x=log10(Miles\_Driven)))

plt + geom\_density( )

- The t.test Function: A test to determine key values of probability, the function and an example of its use:

t.test(x, y = NULL, alternative, mu)

- One Sample t.test Example:

t.test(log10(sample\_table$Miles\_Driven), mu=mean(log10(population\_table$Miles\_Driven)))

- This test returns several specific values: a **t-value**, **df-value**, **p-value**, the **alternative hypothesis**, a **confidence level** and **sample estimates**.

***Two Sample t.test Example:***

t.test(log10(sample\_table$Miles\_Driven), log10(sample\_table2$Miles\_Driven))

🡪 The key difference being the user feeds in 2 samples as arguments)

***Using Two Sample t.test To Compare Samples:***

t.test(mpg\_1999$hwy, mpg\_2008$hwy, paired = T)

🡪 The key difference here is the two samples are indicated as “paired” with the argument being given “T” for true and informing the function the two samples are paired.

- The ANOVA Test Function: A test for analysis of variance (AN. O. VA.) to compare the means of a continuous numerical variable across a number of groups (factors, in R).

- Two *different* types of test exist, one-way ANOVA (single dependent variable and single independent variable ) and two-way ANOVA (two different independent variables). The test is conducted with the aov function in the format:

aov(formula, data, projections, qr, contrasts, …)

🡪 Where, **formula** indicates how the function should interpret different variables and factors (common cases are Y ~ A or Y ~ A + B, with Y being the dependent variable’s column name and A and B are the independent variable column names and **data** is the input data frame’s name.

- Example from the module:

aov(hp ~ cyl, data=mtcars\_filt)

🡪 This function uses previously defined variables and declares the data frame and the returned table includes **sum of squares**, **degree of freedom**, and the values for the “cyl” variable and residuals for residual error for the ANOVA model based on the data frame. The final returned data value is the **residual standard error**.

🡪 Applying the summary( ) function to the aov( ) function above returns a Pr(>F) value, which is the same as the p-value statistic

- Correlation Of Data Variables: The correlation coefficient “r” can be determined in R by combining the geom\_point( ) function with the cor( ) function. The functions follow the format:

plt + geom\_point( ) 🡪 Creates a scatter plot

cor(data$Vector\_name\_1, data$Vector\_name\_2) 🡪 Calculates the coefficient

- The resulting value is returned in the console as a [1] dimensional value. The closer to 1.0, the stronger the correlation between the variables and the correlation coefficient can be taken as an absolute value for correlated strength.

- ***Correlation Matrices***: Instead of computing each pairwise correlation in a dataset, a matrix can be created and the cor( ) function is able to be applied to return the values in a matrix. The module example follows the following format:

used\_matrix <- as.matrix(used\_cars[,c(“Selling\_Price”, “Present\_Price”, “Miles\_Driven”)])

cor(used\_matrix)

- The returned matrix will return a 3 x 3 matrix, with the 3 different dimensional variables being compared to each other for an overall presented comparison of each vector of values.

- Linear Regression Equation: The linear regression equation yields a r squared value that shows a prediction based on the trend of data available currently. It uses the correlation coefficient to approximate the following point with accuracy more-or-less related to the square of the correlation coefficient. The function follows the format:

lm(formula, data, subset, weights, …)

- This analysis requires similar conditions to those needed for a t-test, namely:

1.) the input data is numerical and continuous

2.) the input data follows a linear pattern

3.) there is variability in the independent (x) variable, meaning some gradient is observed

4.) the residual error should be normally distributed (the distance from each data point to the line)

Example from the module:

lm(qsec ~ hp, mtcars)

- This example applies the formula of similarity between “qsec” and “hp” in the dataset “mtcars” provided in the module, this is just an example shown and the actual data is not needed for conceptualization of linear models.

- To determine the p-value and r-squared values for a simple linear regression model, the summary function can be used in the form:

summary(lm(qsec~hp, mtcars))

- The returned analysis from this function provides the **residual standard error**, **multiple R-squared** value, **adjusted R-squared** value and **p-value** as the key analytical results. The underlined values are considered most useful for the current level of analysis being conducted in this module.

- An example of combining all aspects of the linear regression model into a plot generation:

model <- lm(qsec ~ hp, mtcars)

yvals <- model$coefficients[‘hp’]\*mtcars$hp + model$coefficients[‘(Intercept)’]

- Then, combine the plot generation functions:

plt <- ggplot(mtcars, aes(x=hp, y=qsec))

plt + geom\_point( ) + geom\_line(aes(y=yvals), color = “red”)

🡪 The resulting plot combines a scatter plot of data points and provides a line representing the linear regression (line of best fit) for the dataset in red. The yvals variable indicates the values found from the linear model function and shows the trends in the data.

- Multiple Linear Regression: Generating a plot with multiple linear regression is unlikely to yield any useful results, but the returned values from multiple line inputs can be useful for gaining insight into the trends in a dataset. As a result it is ideal to create a summary of the linear model function using the format:

summary(lm(formula, data, subset, weights)

Example from the module:

summary(lm(qsec ~ mpg + disp + drat + wt + hp, data = mtcars))

🡪 The resulting summary of values provides a Pr(>|t|) value in the table, which represents the probability that each coefficient contributes a random amount of variance to the linear model.

- Chi Squared Test: Used to compare categorical distributions, the function uses the format:

chisq.test(x, y, correct, p, rescale.p, …)

🡪 With the values **x** indicating a numeric vector or matrix, **y** indicating a numeric vector (if x is a matrix, it is ignored and if **x** is a factor, y should be a factor of the SAME length), correct indicating whether to apply continuity correcting (when computing for 2 x 2 tables, one half is subtracted from all |O - E| differences), **rescale.p** where if it is true, then p is rescaled to sum to 1 and if false, and p sum does not equal 1, an error is returned.

**R Logic Operators and Python Equivalents**

|  |  |  |  |
| --- | --- | --- | --- |
| **Category** | **R Operator** | **Description** | **Python Equivalent** |
| Arithmetical | + | Addition operator | + |
| - | Subtraction operator | - |
| \* | Multiplication operator | \* |
| / | Division operator | / |
| ^ or \*\* | Exponent operator | \*\* |
| %% | Modulus operator (finds the remainder of the first element divided by the second) | % |
| Relational | < | Each element in the first data structure is less than each element in the second data structure. | < |
| <= | Each element in the first data structure is less than or equal to each element in the second data structure. | <= |
| > | Each element in the first data structure is greater than each element in the second data structure. | > |
| >= | Each element in the first data structure is greater than or equal to each element in the second data structure. | >= |
| == | Each element in the first data structure is equal to each element in the second data structure. | numpy.equal() |
| != | Each element in the first data structure is unequal to each element in the second data structure. | numpy.not\_equal() |
| Logical | x|y | Element-wise OR operator—each element of *x* and *y* structures are combined and returns TRUE if either element is TRUE. | numpy.array(x) | numpy.array(y) |
| x&y | Element-wise AND operator—each element of *x* and *y* structures are combined and returns TRUE if both elements are TRUE. | numpy.array(x) & numpy.array(y) |
| x||y | Logical OR operator—the first element of *x* and *y* structures are combined and returns TRUE if either element is TRUE. | x[0] or y[0] |
| x&&y | Logical AND operator—the first element of *x* and *y* structures are combined and returns TRUE if either element is TRUE. | x[0] and y[0] |
| Miscellany | isTRUE(x) | Checks if the logic *x* is TRUE, otherwise FALSE. | if x: |
| x %in% y | Checks if *x* is contained within *y*. | x in y |
| x:y | Creates a range of integer values from *x* to *y*. | range(x,y) |

**Statistical Test Lookup Table**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Statistical Test** | **Input Variable Type** | | | | **Analytical Question** |
| **Independent** | | **Dependent** | |
| **No. of Variables** | **Data Type** | **No. of Variables** | **Data Type** |
| **One-sample t-test** | 1 | Dichotomous (Population or Sample) | 1 | Continuous | Is there a statistical difference between the mean of the sample distribution and the mean of the population distribution? |
| **Two-sample t-test** | 1 | Dichotomous (Sample A versus Sample B) | 1 | Continuous | Is there a statistical difference between the distribution means from two samples? |
| **ANOVA** | 1+ | Categorical | 1 | Continuous | Is there a statistical difference between the distribution means from multiple samples? |
| **Simple linear regression** | 1 | Continuous | 1 | Continuous | Can we predict values for a dependent variable using a linear model and values from the independent variable? |
| **Multiple linear regression** | 2+ | Continuous | 1 | Continuous | How much variance in the dependent variable is accounted for in a linear combination of independent variables? |
| **Chi-squared** | 1 | Categorical | 1+ | Categorical | Is there a difference in categorical frequencies between groups? |