Markdown Syntax (https://stackedit.io/editor)

#(#####) text

Heading

text **or** _text_ — *Italic*

text **or** text — **Bold**

~~text~~

~~Strikethrough~~

- (* or +) text — •

Unordered list

1. text — 1.

Ordered list

[Alt texts](URL **or** path) — Alt text

<URL **or** path> — URL **or** path

![alt text](image URL **or** path) — Inline

Image Display

![alt text][logo] where [logo]: image url **or** path

Reference Image Display

> paragraphs **or** text — Block

Quote

$$text$$ — Center

Display

`text` — Inline

`code`

```python
s = “Hello World!”
“Hello World!“
print (s)
(s)
```

```
Markdown | Less | Pretty
--- | --- | ---
*Still* | `renders` | **nicely**
1 | 2 | 3
```

Markdown	Less	Pretty
Still	`renders`	**nicely**
1	2	3

———

Horizontal Line — Thin

***

Horizontal Line — Thick

You can create footnotes like this[^footnote].

You can create footnotes like this[2].

Git/GitHub Syntax (https://education.github.com/git-cheat-sheet-education.pdf)

SQL (https://www.codecademy.com/articles/sql-commands?r=master)

CREATE TABLE table (col dtype, etc); — Create
a table with table_name and parameters.

SELECT *(or col) **FROM** table; — Fetch
data [col_name or *everything(*)*] from table_name

INSERT INTO table (col) **VALUES** (contents); — Insert
new rows into table_name (col) with values (contents)

UPDATE table **SET** col = new value **WHERE** col = condition; — Update
row(s) in table_name, Set col to new value, Where row condition met

ALTER TABLE table **ADD COLUMN** col dtype; — Alter
Table add **new col** to table_name

DELETE FROM table **WHERE** col **IS NULL**; — Delete
row(s) from table_name Where row condition met

SELECT DISTINCT col **FROM** table; — Fetch
unique values in the result set

WHERE (=, != , >, <, >=, <=, **LIKE**, **BETWEEN**, **%**A**, %**a**, %**xxx**%**); — Query
filtering (LIKE searches similar str, %A, %a search starts with 'A' or 'a', or contain 'xxx',
AND condition1 **OR** condition2, **IS NULL**, **IS NOT NULL**
Between for range filter), combine multiple conditions using AND OR

ORDER BY col **DESC**/**ASC**; — Sort
result set by col_name in descending (DESC) or ascending (ASC) order
DESC LIMIT 3; — Specify
the max number of result sets (like df.head(x))

SELECT COUNT(*) **FROM** table; — Count()
takes col_name and count number of rows

SELECT col, **COUNT**(*) **FROM** table **GROUP BY** col; — Used
with SELECT to arrange identical data into groups.

SELECT SUM(col) **FROM** table; — Sum all
the values within the col_name

SELECT MAX(col) **FROM** table; — Return
largest value from the fetched col_name

SELECT MIN(col) **FROM** table; Return smallest value from the fetched col_name

SELECT AVG(col) **FROM** table; Return average value from the fetched col_name

SELECT ROUND(**AVG**(col), decimal#) **FROM** table; Return rounded(decimal#) average value from the fetched col_name

CREATE TABLE table (id INTEGER PRIMARY KEY, etc.) Create a table_name specifying id col is primary key col, no NULL and unique

SELECT table1.col1, table1.col2, table2.col1 (etc.) **FROM** table1, table2; Fetch data from multiple tables ***(cross join)***

SELECT * **FROM** table1 **JOIN** table2 **ON** table1.col = table2.col; Fetch data from multiple tables while combining rows if join condition is true ***(inner join)***

SELECT * **FROM** table1 **LEFT JOIN** table2 **ON** table1.col = table2.col; Table1 is fully fetched, table2 is joined onto table1 when condition is met ***(left join)***

SELECT table1.col1 **AS** 'xxx', table2.col1 **AS** 'yyy' **FROM** table1 **JOIN** table2 **ON** condition; Fetch table1.col1 data and rename col AS 'xxx'

SELECT * **FROM** table1 **WHERE** col **IN** (**SELECT** col **FROM** table2 **WHERE** condition);
Subquery within a query fetching data (non-correlated subquery)

ex.

SELECT *a.dep_month, a.dep_day_of_week,* ***AVG****(a.flight_distance)* ***AS*** *average_distance* *** The inner query fetched distance, month, day, which is then used by the outer query to
FROM *(*
compute the average total distance flown by day of week and month
SELECT dep_month, dep_day_of_week, dep_date, ***sum****(distance)* ***AS*** *flight_distance* ***(Ex from CodeAcademy SQL:Table Transformation Subqueries 4) ***
FROM flights
GROUP BY *1, 2, 3*
) a
GROUP BY *1, 2*
ORDER BY *1, 2;*

SELECT col **FROM** table **AS** t **WHERE** condition < (A row is processed in the outer query, such that particular row in the outer query, the **SELECT** **AVG**(condition) subquery is executed. To identify correlated query, spot whether inner query is linked to
FROM table outer query using t.col =(<, > or other operators) table.col
WHERE col = t.col);

SELECT col(s) **FROM** table1 **UNION (ALL) SELECT** col(s) **FROM** table2; Merge cols from tables, statement must same # of col and dtype (distinct default unless ALL)

SELECT col(s) **FROM** table1 **INTERSECT SELECT** col(s) **FROM** table2;
Combine and return only common rows between two SELECTs

SELECT col(s) **FROM** table1 **EXCEPT SELECT** col(s) **FROM** table2; Returns

distinct rows from the 1st SELECT that aren't output by the 2nd SELECT

```
SELECT
        CASE                                                                    If, then,
else in SQL is done using CASE,
                WHEN elevation < 500 THEN 'Low'                                 END is
required to terminate the statement, ELSE is optional and return NULL is not included
                WHEN elevation BETWEEN 500 AND 1999 THEN 'Medium'
                WHEN elevation >= 2000 THEN 'High'
                ELSE 'Unknown'
        END AS elevation_tier, count(*)
FROM airports
GROUP BY 1;
```

```
SELECT DATETIME or DATE or TIME(datetime_col. '+5 hours') FROM table;           Output
YYYY-MM-DD hh:mm:ss format for datetime_col, '+5 hours' for time increment.

SELECT str1 ll ' ' ll str2 AS new_str FROM table;
Concatenate two strings into one, space in between ll ' ' ll matters

SELECT REPLACE(col, 'str_want_to_be_replaced', 'new_str') AS alias FROM table;
Replace selected str into new str within the selected col
```

Python

```
re.search('Phd | P.h.D | Ph.D.', the list you want to search within.strip()) != None, Phd +=1     Regular
expression of Python, search similar str within data structure, if yes add count 1
re.search('^Asso.', the list you want to search within.strip()) != None, Asso +=1                  Regular
expression of Python, search str starts with 'Asso' in data structure, if yes count +1
pd.read_csv('file.csv', parse_dates = [col index])
Parse_date tells read_csv to interpret values in col 5 as dates and convert into np datetime64
pyplot.xticks(rotation = 30)
Rotating x-axis label by 30 degree counterclockwise
pd.rolling_mean(series, window_size)                                                               Pandas
rolling mean attribute takes in a series, and an int representing window size
pd.ewma(series, window_size)                                                                       Pandas
exponentially-weighted moving average takes in a series, and an span number (int)
df.series.fillna(ewma, in place = True)
Replaces missing data with corresponding value from ewma, typical handle on missing data
```

R (https://cran.r-project.org/doc/contrib/Short-refcard.pdf)

———Variables

x <- 2L R
method of assigning integer to an variable
x <- 2.5 R
method of assigning double(float) to an variable
x <- 3 + 2i R
method of assigning complex to an variable
x <- “str” R
method of assigning character(str) to an variable
q <- TRUE(T) or FALSE(F) R
method of assigning boolean(T/F) to an variable
10 %% 8 R
method of return only the reminder of the division (In this case it returns 2)
typeof() **OR** class()
Attribute of finding the type of variable
paste(charac1, charac2)
Character(string) addition in R
R1 <- 4 > 5 R2 <- !(5>1) R1
Check the statement on the right and return either T or F into variable, R2 assigns (NOT(TRUE)) == (FALSE) as variable
R1 | R2 Check
either R1 or R2 is TRUE, if one of them is true return TRUE
R1 & R2 Check
both R1 and R2 are TRUE, if both of them are true return TRUE, else FALSE
isTRUE(var) Check
if variable is TRUE
rm(var)
Remove existing variable

———Loops

while(var < value){ **While loop** in R, while(condition){perform this task}, **EX[**counter <- 1, while(counter < 12){print(counter) counter <- counter + 1}**]**
do something **#break** **#break command will break and terminate the loop Python: break**
}

for(condition){ **For loop** in R, for(condition){perform this task}, **EX[** for(i in 1:5){print(“hello”} **] #for i in 1 to 5 (1-5), print hello each time**
do something **#R can be itemize as Python, EX[v <- c(1, 2, 3), for(number in v){code} works in R]**
} **#matrix iteration in forloop is by col (col1 -> col2 -> col3)**

If loop

in R, if(condition){perform this task} else if(condition2){perform this task} else{perform this task}
x <- rnorm(1)
#random normal distribution with n = 1
if(x > 1){ #if loop
in R, if x is less than 1
answer <- "Greater than 1" #then
print statement

} else if(x >= -1){ #else if
x is greater or equal to -1
  answer <- "Beween -1 and 1" #print
else if statement
} else{ #else
  answer <- "Less than -1" #print
else statement
}

———Vector Operations

———

var <- c(x1, x2, x3, x4, x5) Define
numeric vectors in R is done through combine function c(x1, x2, x3, …, xn)
is.numeric(var) Check
if var is a numeric vector
is.integer(var) Check
if var is a integer vector
is.double(var) Check
if var is a double vector **#vector contains doubles, R default stores double**
is.character(var) Check
if var is a character vector **#can't convert character to number but can convert number to character**
as.list() **OR** as.data.frame() **OR** as.double() **OR** as.numeric() **OR** as.integer() Convert
other data structures into as.want_to_be_data_structure **#Python: x.astype(int)**
as.Date("non-standard_format_date", format = "%b-%d-%y") Intake
an non-standard date format and convert it into standard yyyy-mm-dd format
seq(starting_number, ending_number, steps) Create
a numeric sequence from starting number to ending number with steps inclusive
rep(number I vector I character, number of replications) Create
a vector with the input element with specify number of replications
sort(vector) Sort a
vector, default in ascending order, decreasing = TRUE will reverse
rev(vector)
Reverse elements in object
var[x] **OR** var[-x] **OR** var[x:y] Select
the x-index element from vector, **OR** select every element except for the x-index element, **OR** elements from x to y index
var[c(x1, x2, x3)] **OR** var[c(-x1, -x2)] w[c(x1, x2, x3)] **OR** var[c(-x1, -x2)] is the same as w[x1, x2, x3] **OR** x[-x1, -x2] as deletion of the input index elements
var[row, col]
Selecting element in matricies **#var[row,] is the selection of the entire row, var[, col] is the selection of the entire col**
names(vector) <- name_vector Assign
names to a vector **EX[**names(Charlies) <- c("a", "b", "c", "d", "e")**]**
sum(vector) **OR** mean(vector) **OR** prod(vector) Sum /
Average/ Product(multiply) all the elements in the vector
vector[vector < **OR** > **OR** != **OR** == value **&** condition 2 **I** condition 3] Vector
filtering
paste0(vector1, vector2, ….)
Concatenate (add one after another) vectors
append(value **OR** vector, vector **OR** list to be append on) Append
(add) values **OR** vector into a list **OR** vector, **#Python: list.append(value)**

———Matrix Operations

———

——————

rbind(vector1, vector2, …, vectorn) Takes multiple vectors and fill each row starting on the first row

cbind(vector1, vector2, …, vectorn) Takes multiple vectors and fill each col starting on the first col

matrix(vector, # of rows, # of cols, optional[byrow = T/F]) Takes a vector and bends it to create a matrix, default fills 1st col -> 2nd col -> .., byrow = T makes it fill row first

m[“row name”, “col name”] **OR** m[row#, col#] **OR** m[“row name”, col#] **OR** m[row#, “col name”]
Different ways of accessing specific element in the matrix

rownames(matrix) <- row_name_vector Assign row names to a matrix

colnames(matrix) <- col_name_vector Assign col names to a matrix

m[row, col] <- var **#R allows you to select certain element within the matrix and assign new values to it**

m1 / m2 Matrix division across two matrices

t(matrix)
Transpose matrix in R

colsums(matrix) **OR** rowsums(matrix) Takes the sum through the rows **OR** col in the matrix

colmeans(matrix) OR rowmeans(matrix) Takes the average through the rows **OR** col in the matrix

m[x:x, y:y]
Subsetting matrix through index **EX[**A[1:3, 6:10] subsets matrix A to return new matrix with row 1 to 3 with col 6 to 10**]**

m[c(x1, x2),] **OR** m[, c(y1, y2), drop = F]
Subsetting the rows or cols from matrix, can use col or row names instead of index as well

m[row, col] **OR** m[row,] **OR** m[, col] **OR** m[row, col, drop = F] Passing in values into matrix will return vector in default, the optional argument drop = F will return matrix instead of vector

———Function

——

function_name <- function(input1, input2 input3 = default){
Creating function in R
  #code execute
#Return at the end of function serve the same purpose as python return
  return(result)
}

sapply(v, function(num){num * 2})
Anonymous function, apply anonymous function (num*2) on each elements in v (return vector) **#Python: Lambda operator**

sample(x = x1:x2, n)
Generate random *n* values between range of x1 to x2

grepl(“keyword_want_to_search_for”, character_vector) Search for the keyword within the character_vector, return logical statement **(TRUE / FALSE)**

grep(“keyword_want_to_search_for”, character_vector) Search for the keyword within the character_vector, return the index of the found keyword

Sys.Date() Returns today in the format of “yyyy-mm-dd” as Date object

strptime(“hh:mm:ss”, format = “%H:%M:%S”) Intakes character vector and convert into timestamps

filter(df, col_name == condition1, col_name > condition2, col_name < condition3, etc.) Filter df like using subset function, specify col_name wants to be filtered on and the conditions **#Package "dplyr".**

slice(df, row_indices_to_slice) Slice df horizontally on rows **#Package "dplyr". EX[slice(df, 1:10) will return first two rows of df]**

arrange(df, col1, col2, col3, … etc.) Sort df by columns **#Package "dplyr". EX[arrange(df, col1, desc(col2)) returns df sort by col1 then desc(col2) order]**

select(df, col1, col2, col3, … etc.) Select specific col from the df **#Package "dplyr".**

rename(df, new_col_name = original_col_name)
Rename cols in df **#Package "dplyr".**

distinct(df, col_name) Select distinct(unique) values from df cols **#Package "dplyr". often used with distinct(select(df, col))**

mutate(df, new_col = col1 - col2) Quick way to create new_col based on old column operations **#Package "dplyr".**

transmute(df, new_col = col1 - col2) Quick way to create and return only the new_col based on old column operations **#Package "dplyr".**

summarise(df, new_col = mean **OR** max **OR** etc.(col1, na.rm = TRUE)) Perform column-wise calculation (mean, max, min, etc. and remove all NA values) **#Package "dplyr".**

sample_n(df, n)
Randomly select *n* rows from the df **#Package "dplyr"**

sample_frac(df, n)
Randomly select *n fraction (n = 0.1 means 10%)* rows from the df **#Package "dplyr"**

gather(df, key, pair, cols_want_to_gather_into_key-value_pair)
Collapse multiple cols into key-value pairs **#Package "tidyr".**

spread(df, key, pair)
Uncollapse gathered data (key-value pairs) into multiple cols **#Package "tidyr".**

separate(df, col_desired_to_be_separated, c('col1', 'col2'), sep = "separation character")
Separate elements from 1 col to 2+ cols based on separation character, default = non-alphabetical characters **#Package "tidyr".**

unite(df, new_col_name, col1, col2, sep = "separation character") Unite col1 & col2 elements into one col, join with separation character **#Package "tidyr".**

iris %>% R piping operator, takes output of one command and use it as input for the next, connect by %>%
  subset(iris$col_name == value) %>%
#similar to linux pipping command "|"
  tail(n = 5) %>% **#Same as summary(tail(subset(df, df$col_name == value), n = 5))**
  summary()

———Data Frame

———

read.csv(file.choose()) **OR** read.csv("file_name") Read **csv** file in R, file.choose() prompts window to choose file, if same path, can just type in file name **#Python: pd.read_csv()**

write.csv(df, file = "file_name_want_to_be_saved_as.csv") Write a **csv** file in R, #Python: pd.to_csv

excel_sheets("file_name")
Returns vector referencing sheets(tabs) in excel file #Package "readxl" allows read in excel file in R

read_excel("file_name", sheet = "sheet_vector_output_from_above") Read **excel** sheets in R, sheet argument takes excel_sheet("file_name") output

entire.workbook <- lapply(excel_sheets("file_name"), read_exce, path = "file_name") Read

entire excel file with all sheets **#list-apply(get all sheet names, apply read_excel on each, along the excel file)**
lapply(vector, function) **OR** sapply(vector, function) **OR** vapply(vector, function)
List.apply applies the function onto each element in the vector, returns a list **#sapply & vapply returns vector, matrix or array**
getwd() Get
current working directory, same as unix command pwd
setwd("path") Set the
working directory path **EX[**setwd("\Users\IvanC\R\")**] #with working directory**
nrow(df) Returns
the number of rows in the data frame **#python: df.shape()**
ncol(df) Returns
the number of cols in the data frame **#python: df.shape()**
head(df, n = 6) Returns
top 6 rows (default) of the data frame **#python: df.head()**
tails(df, n = 6) Returns
the last 6 rows (default) of the data frame **#python: df.tail()**
str(df)
STRUCTURE() Returns a quick debrief of the data frame **#python: df.describe()**
summary(df) Returns
a summary of the data frame
df[row, col] **==** df[[row, col]] Extract
elements from the df **#remain df for using df[row,], drop dimension into vector uses df[, col, drop = F], or [["col"]]**
df$col_name **==** df[, "col_name"] **==** df[, col#] **==** df[["col_name"]] Select
the col **OR** row from the data frame and output it as vector form **#df["col_name"] OR df[col#] returns df form**
levels(df$col_name)
df$col_name selects the specific col, and levels would return the categorical description of the levels **EX[**high, medium, low**]**
df$col_name * **OR** + **OR** / **OR** - df$col_name
dataframe basic addition, subtraction, division, multiplication operations
df$new_col_name <- vector **OR** col_values Create
new col with values into df **#if input insufficient to # of rows, vector will recycle to fill in (has to multiple of row#)**
df$col_name <- NULL
Remove col from data frame
df[df$col_name < **OR** > **OR** != **OR** == value **&** condition 2 **I** condition 3] Filtering
data frame with input condition(s), df$col_name </>/!=/== value is the condition, this condition can be store as a boolean
subset(df, col_name_want_to_search_in %in% vector_contain_keywords_for_search)
Subsetting a data frame based on keywords vector search in a particular column(T/F) vector first
subset(df, col_name_want_to_search_in < **OR** != **OR** == value **& OR I** condition 2)
Subsetting a data frame based on condition input into the col_want_to_search_in **EX[**subset(df, subset = rain == TRUE)**]**
any(is.na(df **OR** df$col)) <- replacement data(mean or other values) Returns
a TRUE **OR** FALSE statement whether the df **OR** df$col has any missing data, then assign with new values
order(df$col_name) **OR** order(-df$col_name) Returns
vector with index orders of rankings **EX[**[2 1 5] is index 2 value lowest & index 5 highest, -df["col_name"] reverse order**]**
df <- data.frame(col_name 1 = vector1, col_name2 = vector2, col_name3 = vector3…)
Creating data frame from vectors, same as creating matrix with vectors using rbind() or cbind()
merge(df1, df2, by.x = "df1_col", by.y = "df2_col") Merge
two data frames into one by joining on columns, x refers to df1, y refers to df2

df$col_name <- factor(df$col_name, Turn numeric elements in a df col into categorical variables, integer/float into string
ordered = True, levels == c(define order vector))
#ordered = T, levels = c("hot", "med", cold") gives the factor rankings, specify by the levels vector

———List ——————————————————

list(vname = vector, mname = matrix, dname = df) List in R can store vector, matrix, and df into a single var (name it when parse in), index through double bracket [1], [2], [3]
list$element_in_list **OR** list[]'element_name]] OR list['element_name']
Indexing and selecting elements in list, $element **AND** [['element_name]] returns numeric vector, ['element_name'] return list
c(list1, list2)
combine two lists together at one end of the other.

———Data Visuzaliation (https://www.rstudio.com/wp-content/uploads/2015/03/ggplot2-cheatsheet.pdf :::: ggplot2) (http://ropensci.github.io/plotly-test-table/tables/0e3d5ca144d27d8416318824c1b6ec1421a51045/index.html & https://plot.ly/ggplot2/ :::: ggplotly)———

qplot(data = df, x = col1, y = col2, size = I(value), color = I("b/r/g.."), geom = ("boxplot") qplot function from ggplot2, quick visualization for data frame, alpha is transparency in 0-1 scale
shape = I(1-25 different shapes), alpha = I(), main = "plot title") **#color = col3, same as dragging a parameter onto the color filter in Tableau**

ggplot(data = df, ggplot in R, parse in data, define aes(x,y, and optional layer), geom_point() is scatter plot EX[*geom_line(size = 1), define*
aes(x = x_var, y = y_var, color = 3rd_var, size = 4th_var)) + l*ine size*], geom_smooth() is line with CI
geom_point(alpha = 0.1, size = 1) + geom_smooth() + **#aes is the function in ggplot that takes mapping variables on plot [x, y, color, size]** xlab("x_axes_title") + ylab("y_axes_title") **#xlab and ylab defines the x and y axes title**
xlim(start, end) + ylim(start, end) **#xlim and ylim sets x and y axes coordinates**

p <- ggplot(data = df,
aes(x = x_var, y = y_var, color = 3rd_var, size = 4th_var)) Creates data layer with defined x and y and other optional variables
p + geom_point(aes(size = new_variable, color = new_variable2)) **#This creates an incident of using aes() function in geom_points() to override the p variables, and creates a one timed**
+ scale_color_gradient(low = "color1", high = "color2") **plot with the overridden variables. #scale_color_gradient() specifies color gradient**

ggplot(data = df, aes(x = x_var)) + Creates histogram with bin width set as value input, histogram does not require y_variable
geom_histogram(binwidth = value, fill = "color", aes(fill = 2nd_var), color = "color") **#fill = "color" sets the entire graph to a color, aes(color = 2nd_var) maps it on another variable, color sets border colors**
ggplot(data = df, aes(x = categorical_var, y = 2nd_var, color = 3rd_var)) + Creates box plot in R, x needs to be categorical variable
geom_boxplot(size = box_size_value, alpha = value) + geom_jitter() **#size**

inside geom_boxplot() sets box plot border size, geom_jitter() adds random points on each box plot background

ggplot(data = df, aes(x = var1, y = var2)) + Creates **2d visualization heat-map** like graph (based on frequency of occurance) with var1 and var2
    geom_bin2d(binwidth = c(x1, x2)) **OR** geom_hex() **OR** geom_density2d()
#binwidth affects the resolution of the heat map, default is set to be c(1,1)
    scale_fill_gradient(low = "color1", high = "color2")
#scale_fill_gradient() specifies color gradient

ggplot(data = df, aes(x = var_x, y = var_y, color = 3rd_var)) +
Facet_grid() creates row or col separations on each categorical variables **#acting like pages on Tableau**
    geom_point(size = 3) +
#facet_grid(var4~.) facets by row(x), facet_grid(. ~var5) facets by col(y)
    facet_grid(var4~.) **OR** facet_grid(.~var5) **OR** facet_grid(var4~var5)
#facet_grid(var4~var5) will create both row and col separation, ##acting like sns.pairplot()

ggplot(data = df, aes(x = var1, y = var2, size = var3, color = var4)) + Zoom in onto particular c(start, end) vector defined window on the graph
    geom_point() + coord_cartesian(xlim **OR** ylim = c(start, end))
#coord_cartesian(xlim = c(start, end)) is the function that takes a length of 2 vector (start & end) and zoom in on graph
  (+) coord_fixed(ratio = 1/3) **#set aspect ratio of the output graph (1/3 means 3x to 1y ratio)**

ggplot(data = df, aes(x = var1, y = var2) +
    xlab("x_axis_title") + ylab("y_axis_title") +
    ggtitle("plot_title") **#set plot title, can use theme to set title's font and color**
  (+) theme_dark()
#pre_set plot_theme, more pre_set themes in Package "ggthemes"
    theme(axis.title.x = element_text(color = "color", size = value), **#set x title's font and color**
        axis.title.y = element_text(color = "color", size = value), **#set y title's font and color**
        axis.text.x = element_text(size = value), **#set x axis coordinate's font and color**
        axis.text.y = element_text(size = value)) **#set y axis coordinate's font and color**
        legend.title = element_text(size = value), **#set legend title's font and color**
        legend.text = element_text(size = value), **#set legend text's font and color**
        legend.position = c(1, 1), **#set legend position (0,0 is bottom-left, 1,1 is upper-right)**
        legend.justification = c(1, 1), **#justify legend position, must use to accompany legend.position**
        plot.title = element_text(color = "color", size = value, family = "Courier"))

ggplotly(ggplot(#codes)) **Create interactive plots (EXTREMELY USEFUL)**

##add ? OR help("topic") OR help.search("topic") in front of R functions will pull up the help page

---

Statistics

- **Probability mass function (PMF)** - maps from values to its probability, a probability is a frequency expressed as a fraction of the sample size (1, 2, 2, 3, 5) —> ([1:0.2], [2:0.4], [3:0.2], [5:0.2])
- **Cumulative distribution function (CDF)** - function that maps from a value to its percentile rank, (1, 2, 2, 3, 5) —> ([0:0], [1:0.2], [2:0.6], [3:0.8], [5:1])
- **Interquartile range (IQR)** - a measure of the spread of a distribution between 75th to 25th percentiles
- **Complementary CDF (CCDF)** - function(1 - CDF(x)) on the log scale
- **When **p-value is smaller** than alpha (0.05 or 0.1), **events are unlikely to occur by chance (meaning statistically significant).****
- **When **R^2** value for the model **is small**, means that **the variable doesn't account for a substantial part of the variation**.**
- **Time Series Analysis** uses **Moving Averages**, which divides the series into overlapping regions called windows, and computes the average of the values in each window.
- **Time Series Analysis** also applies **Exponentially-Weighted Moving Average (EWMA)**, which computes weighted average where the most recent value has the highest weight and the weights for previous values drop off exponentially.
- **Time Series Analysis** employs **Serial Correlation** with lag as the shift to evaluate corr (Correlation in Python) from one value to its next value.