Code

Learning R Programming (Codecademy)

This is an R Markdown (http://rmarkdown.rstudio.com) Notebook.

R Coding Commands Overview:

Lesson 1: Introduction to R Syntax

Basic Calculations (- + * /) can be used in R programming.

Hide

```
25 * 4 + 9 / 3
```

```
[1] 103
```

Data types include: Numeric, Character, Logical and NA.

Hide

```
print('Tahmid') # '__' is needed to class this as a character data type
```

```
[1] "Tahmid"
```

Hide

```
print(24) # this is classed as a numeric data type
```

```
[1] 24
```

Hide

```
print('24') # this is classed as a character data type
```

```
[1] "24"
```

Hide

```
class (FALSE) # this is a logical data type
```

```
[1] "logical"
```

Variables are used to store information and associate it with a name.

Hide

```
name = 'Tahmid'
print(name)
```

```
[1] "Tahmid"
```

Hide

```
age = 24
print(age)
```

```
[1] 24
```

Vectors are list like structures containing items of the same data type.

Hide

```
spring_months = c("March", "April", "May", "June")
print(spring_months)
```

```
[1] "March" "April" "May"   "June"
```

Hide

```
phone = c(44, 123, 6789)
print(phone)
```

```
[1]   44  123 6789
```

Conditionals can be added to the code to specify and isolate data to be used in the coded task.

Hide

```
message = 'I change based on a condition.'
if (TRUE) {message = 'I execute this when true!'} else {message = 'I execute this when false!'}

print(message)
```

```
[1] "I execute this when true!"
```

Comparison operators (< > <= >= == !=) can also be used in R programming

Hide

```
56 >= 129 # will equate to FALSE
```

```
[1] FALSE
```

Hide

```
56 != 129 # will equate to TRUE
```

```
[1] TRUE
```

Logical operators AND (&), OR (|) and NOT (!)

Hide

```
message = 'Should I pack an umbrella?'
weather = 'cloudy'
high_chance_of_rain = TRUE

if (weather == 'cloudy' & high_chance_of_rain ) {message = 'Pack umbrella!'} else {message = 'No need for umbrella!'}

print(message)
```

```
[1] "Pack umbrella!"
```

Various functions that can be used in R.

Hide

```
data <- c(120,22,22,31,15,120)

unique_vals = unique(data)
print(unique_vals)
```

```
[1] 120  22  31  15
```

Hide

```
solution = sqrt(49)
print(solution)
```

```
[1] 7
```

Hide

```
round_down = floor(3.14)
print(round_down)
```

```
[1] 3
```

Hide

```
round_up = ceiling(3.14)
print(round_up)
```

```
[1] 4
```

Lesson 2: Introduction to Data Frames in R.

Data frames are similar to data in tables that can be found in excel and SQL. The data is formed into columns and rows.

Hide

```
# load libraries
library(dplyr)
library(readr)
```

Hide

```
# load data frame
artists <- read_csv('artists.csv')
```

Hide

```
# inspect data frame
artists
head(artists)
summary(artists)
```

Hide

```
# select columns, filter and arrange rows of artists
artists_manipulated <- artists %>%
  select(-country,-year_founded,-albums) %>%
  filter(spotify_monthly_listeners > 20000000, genre != 'Hip Hop') %>%
  arrange(desc(youtube_subscribers))
artists_manipulated
```

Inspecting a Data Frame.

Hide

```
# load libraries
library(readr)
library(dplyr)
```

Hide

```
# load data frame
artists <- read_csv('artists.csv')
```

Hide

```
# inspect data frame

artists
head(artists)
summary(artists)
```

Piping (%>%) helps increase the readability of data frame code by piping the value on its left into the first argument of the function that follows it.

Hide

```
# inspect data frame with pipe
artists %>%
head()
```

Selecting columns in a data frame.

Hide

```
# select one column
artist_groups = artists %>%
select(group)
```

Hide

```
# select multiple columns
group_info = artists %>% select(group, spotify_monthly_listeners, year_founded)
```

Excluding columns in a data frame.

Hide

```
# select all columns except one
no_albums = artists %>% select(-albums)
no_albums
```

Hide

```
# select all columns except a set
df_cols_removed = artists %>% select(-genre,-spotify_monthly_listeners,-year_founded)
df_cols_removed
```

Filtering rows with logic

Hide

```
# filter rows one condition
rock_groups = artists %>% filter(genre == 'Rock')
rock_groups
```

Hide

```
# filter rows multiple conditions
popular_rock_groups = artists %>% filter(genre == 'Rock', spotify_monthly_listeners > 20000000)
popular_rock_groups
```

Hide

```
# filter rows with or
korea_or_before_2000 = artists %>%
filter(country == 'South Korea' | year_founded < 2000)
korea_or_before_2000
```

Hide

```
# filter rows with not !
not_rock_groups = artists %>%
filter(!(genre == 'Rock'))
not_rock_groups
```

Arranging Rows

Hide

```
# arrange rows in ascending order
group_asc = artists %>%
arrange(group)
group_asc
```

Hide

```
# arrange rows in descending order
youtube_desc = artists %>%
arrange(desc(youtube_subscribers))
youtube_desc
```

Review of shortening code using pipes.

Hide

```
# select columns
chosen_cols = artists %>%
  select(-country,-year_founded,-albums)
head(chosen_cols)
```

Hide

```
# filter rows
popular_not_hip_hop = chosen_cols %>%
  filter(spotify_monthly_listeners > 20000000, genre != 'Hip Hop')
head(popular_not_hip_hop)
```

Hide

```
# arrange rows
youtube_desc = popular_not_hip_hop %>%
  arrange(desc(youtube_subscribers))
head(youtube_desc)
```

Hide

```
# select columns, filter and arrange rows
artists = artists %>% select (-country, -year_founded, -albums) %>%
filter(spotify_monthly_listeners > 20000000 & !(genre == 'Hip Hop')) %>%
arrange(desc(youtube_subscribers))
head(artists)
```

Lesson 3: Modifying Data Frames in R.

Adding Columns to a data frame.

Hide

```
# inspect data frame
head(dogs)
```

Hide

```
# add average height column
dogs = dogs %>%
mutate(avg_height = (height_low_inches + height_high_inches) / 2)
head(dogs)
```

Hide

```
# add average height, average weight and rank change columns
dogs <- dogs %>%
  mutate(avg_height = (height_low_inches + height_high_inches)/2, avg_weight = (weight_low_lbs +
weight_high_lbs)/2, rank_change_13_to_16 = rank_2016 - rank_2013)
  head(dogs)
```

Transmute Columns. # update the function call to drop all existing columns

Hide

```
# update the function call to drop all existing columns
dogs <- dogs %>%
  transmute(breed = breed,avg_height = (height_low_inches + height_high_inches)/2,
         avg_weight = (weight_low_lbs + weight_high_lbs)/2,
        rank_change_13_to_16 = rank_2016 - rank_2013)
head(dogs)
```

Rename Columns in a data frame.

Hide

```
# add columns and remove existing columns
dogs <- dogs %>%
  transmute(breed = breed,
            avg_height = (height_low_inches + height_high_inches)/2,
         avg_weight = (weight_low_lbs + weight_high_lbs)/2,
        rank_change_13_to_16 = rank_2016 - rank_2013)
```

Hide

```
# check column names
original_col_names = colnames(dogs)
original_col_names
```

Hide

```
# rename data frame columns
dogs = dogs %>% rename(avg_height_inches = avg_height, avg_weight_lbs = avg_weight, popularity_c
hange_13_to_16 = rank_change_13_to_16)
new_col_names = colnames(dogs)
```

Hide

```
# add new columns, drop existing columns and arrange
dogs = dogs %>% transmute(breed = breed, height_average_feet = (height_low_inches + height_high_
inches)/2/12, popularity_change_15_to_16 = rank_2016 - rank_2015) %>% arrange(desc(popularity_ch
ange_15_to_16))

head(dogs)
```

Lesson 4: Data Cleaning in R

Dealing with multiple files

Hide

```
# list files
student_files = list.files(pattern = 'exams_.*csv')
```

Hide

```
# read files
df_list = lapply(student_files, read_csv)
```

Hide

```
# concatenate data frames
students = bind_rows(df_list)
```

Hide

```
# number of rows in students
nrow_students = 1000
```

Reshaping Data

Hide

```
# original column names
original_col_names = colnames(students)
original_col_names
```

Hide

```
# gather columns
students = students %>% gather('fractions','probability', key = exam, value = 'score')
```

Hide

```
# updated column names
gathered_col_names = colnames(students)
gathered_col_names
```

Hide

```
# unique value counts of exam
exam_counts = students %>% count(exam)
exam_counts
```

Dealing with duplicates

Hide

```
# drop id column
students = students %>% select(-id)
head(students)
```

Hide

```
# find and count duplicated rows
duplicates = students %>% duplicated()
duplicates = duplicates %>% table()
duplicates
```

Hide

```
# remove duplicated rows, keep only unique rows
students = students %>% distinct()
```

Hide

```
# find and count duplicated rows in updated data frame
updated_duplicates = students %>% duplicated %>% table()
updated_duplicates
```

Splitting by index

Hide

```
# print columns of students
print(colnames(students))
```

Hide

```
# view head of students
head(students)
```

Hide

```
# add gender and age columns
students = students %>% mutate(gender = str_sub(gender_age,1,1), age = str_sub(gender_age,2))
head(students)
```

Hide

```
# drop gender_age column
students = students %>% select(-gender_age)
head(students)
```

Splitting by character

Hide

```
# view the head of students
head(students)
```

Hide

```
# separate the full_name column
students <- students %>%
  separate(full_name,c('first_name','last_name'),' ',extra = 'merge')
head(students)
```

Looking at data types

Hide

```
# print structure of students
print(str(students))
```

String parsing

Hide

```
# view head of students
head(students)
```

Hide

```
# remove % from score column
students = students %>% mutate(score = gsub('\\%', '', score))
head(students)
```

Hide

```
# change score column to numeric
students = students %>% mutate(score = as.numeric(score))
head(students)
```

Lesson 5: Intro Into Visualisation With R

Codecademy Example

Hide

```
# Observe layers being added with the + sign
viz = ggplot(data=movies, aes(x=imdbRating, y=nrOfWins)) +
       geom_point(aes(color=nrOfGenre), alpha=0.5) +
       labs(title="Movie Ratings Vs Award Wins", subtitle="From IMDB dataset", y="Number of Awar
d Wins", x="Move Rating", color = "Number of Genre")

# Prints the plot
viz
```

Associating the data with the plot

Hide

```
#Define variable and print it
viz = ggplot(data = movies)
print(viz)
```

Aesthetic Mapping of the plot

Hide

```
#Create aesthetic mappings at the canvas level
viz = ggplot(data = movies, aes(x= imdbRating, y = nrOfWins))
print(viz)
```

Adding Geoms

Hide

```
# Add a geom point layer
viz <- ggplot(data=movies, aes(x=imdbRating, y=nrOfWins)) +
geom_point()

# Prints the plot
viz
```

Adding geom aesthetics

Hide

```
# Add manual alpha aesthetic mapping
viz <- ggplot(data=movies, aes(x=imdbRating, y=nrOfWins)) +
geom_point(aes(color = nrOfGenre))

# Prints the plot
viz
```

Manual Aesthetics

Hide

```
# Add manual alpha aesthetic mapping
viz <- ggplot(data=movies, aes(x=imdbRating, y=nrOfWins)) +
       geom_point(aes(color=nrOfGenre), alpha = 0.5)

# Prints the plot
viz
```

Labelling the Plot

Hide

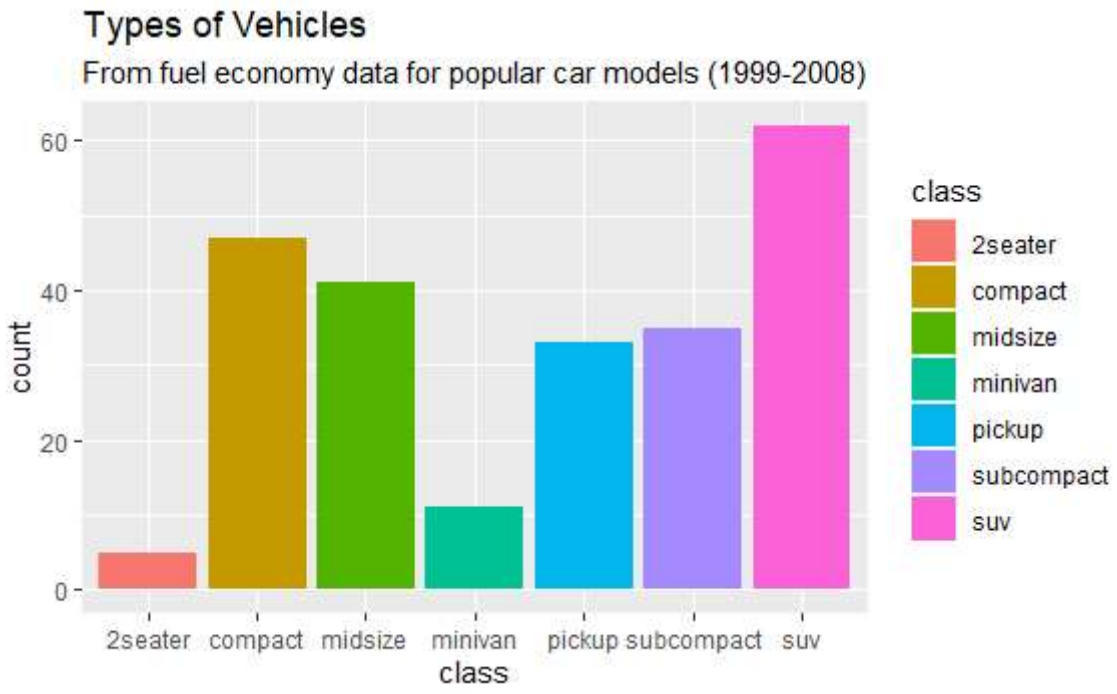
```
# Add labels as specified
viz <- ggplot(data=movies, aes(x=imdbRating, y=nrOfWins)) +
       geom_point(aes(color=nrOfGenre), alpha=0.5) +
       labs(title = 'Movie Ratings Vs Award Wins', subtitle = 'From IMDB dataset', x = 'Movie Ra
ting', y = 'Number of Award Wins', color = 'Number of Genre')

# Prints the plot
viz
```

manufacturer <chr>	model <chr>	displ <dbl>	year <int>	cyl <int>	trans <chr>	drv <chr>	cty <int>	hwy <int>	fl <chr>
audi	a4	1.8	1999	4	auto(l5)	f	18	29	p

manufacturer <chr>	model <chr>	displ <dbl>	year <int>	cyl <int>	trans <chr>	drv <chr>	cty <int>	hwy <int>	fl <chr>
audi	a4	1.8	1999	4	manual(m5)	f	21	29	p
audi	a4	2.0	2008	4	manual(m6)	f	20	31	p
audi	a4	2.0	2008	4	auto(av)	f	21	30	p
audi	a4	2.8	1999	6	auto(l5)	f	16	26	p
audi	a4	2.8	1999	6	manual(m5)	f	18	26	p

6 rows | 1-10 of 11 columns



Lesson 6: Aggregates in R

Calculating column statistics

Hide

```
# define most_expensive here:
most_expensive = orders %>% summarise(max(price, na.rm = TRUE))
print(most_expensive)
```

Hide

```
# define num_colors here:
num_colors = orders %>% summarise(n_distinct_shoe_color = n_distinct(shoe_color))
print(num_colors)
```

Calculating aggregate functions

Hide

```
# define pricey_shoes here:
pricey_shoes = orders %>% group_by(shoe_type) %>% summarize(max_price = max(price, na.rm = TRUE
))
pricey_shoes
```

Hide

```
# define shoes_sold here:
shoes_sold = orders %>% group_by(shoe_type) %>% summarize(count = n())
shoes_sold
```

Hide

```
# define shoe_counts here:
shoe_counts = orders %>% group_by(shoe_type, shoe_color) %>% summarise(count = n())
shoe_counts
```

Hide

```
# define shoe_prices here:
shoe_prices = orders %>% group_by(shoe_type, shoe_material) %>% summarise(mean_price = mean(pric
e, na.rm = TRUE))
shoe_prices
```

Combining grouping with filter

Hide

```
# define most_pop_orders here:
most_pop_orders = orders %>% group_by(shoe_type) %>% filter(n() > 16)
```

Combining grouping with mutate

Hide

```
# define diff_from_mean here:
diff_from_mean = orders %>% group_by(shoe_type) %>% mutate(diff_from_shoe_type_mean = price - me
an(price, na.rm = TRUE))
diff_from_mean
```

Review Task

Hide

```
# inspect data frames
head(orders)
head(page_visits)
```

Hide

```
# define average_price here:
average_price = orders %>% summarise(average_price = mean(price, na.rm = TRUE))
average_price
```

Hide

```
# define click_source here:
click_source = page_visits %>% group_by(utm_source) %>% summarise(count = n(), na.rm = TRUE)
click_source
```

Hide

```
# define click_source_by_month here:
click_source_by_month = page_visits %>% group_by(utm_source, month) %>% summarise(count = n(), n
a.rm = TRUE)
click_source_by_month
```

Lesson 7: Joining Tables in R

Inner Joining

Hide

```
# inspect orders, customers and products
head(sales)
head(targets)
```

Hide

```
# define sales_vs_targets here:
sales_vs_targets = inner_join(sales, targets)
print(sales_vs_targets)
```

Hide

```
# define crushing_it here:
crushing_it = sales_vs_targets %>% filter(revenue > target)
print(crushing_it)
```

Hide

```
# load men_women data here:
men_women = read_csv('men_women_sales.csv')

# inspect men_women here:
head(men_women)
```

Hide

```
# define all_data here:
all_data = sales %>% inner_join(targets) %>% inner_join(men_women)
print(all_data)
```

Hide

```
# define results here:
results = all_data %>% filter(revenue > target, women > men)
print(results)
```

Join on Specific Columns

Hide

```
# rename the id column of products here:
products = products %>% rename('product_id' = 'id')
print(products)
```

Hide

```
# define orders_products here:
orders_products = orders %>% inner_join(products)
print(orders_products)
```

Hide

```
# inspect orders and products
head(orders)
head(products)
```

Hide

```
# define orders_products here:
orders_products = orders %>% inner_join(products, by = c('product_id' = 'id'))
print(orders_products)
```

Hide

```
# define products_orders here:
products_orders = products %>% inner_join(orders, by = c('id' = 'product_id'), suffix = c('_prod
uct', '_order'))
head (products_orders)
```

Using full joins for mismatched joins

Hide

```
# define store_a_b_full here:
store_a_b_full = store_a %>% full_join(store_b)
head(store_a_b_full)
```

Left and right joins

Hide

```
# define left_a_b here:
left_a_b = store_a %>% left_join(store_b)
head(left_a_b)
```

Hide

```
# define left_b_a here:
left_b_a = store_b %>% left_join(store_a)
head(left_a_b)
```

Concatenate data frames

Hide

```
# define menu here:
menu = bakery %>% bind_rows(ice_cream)
head(menu)
```

Review Task

Hide

```
# inspect visits and checkouts here:
head(visits)
head(checkouts)
```

Hide

```
# define v_to_c here:
v_to_c = visits %>% inner_join(checkouts)

v_to_c = v_to_c %>% mutate(time = checkout_time - visit_time)
```

Hide

```
# define avg_time_to_check here:
avg_time_to_check <- v_to_c %>%
  summarize(mean_time = mean(time))
avg_time_to_check
```

Lesson 8: Mean in R

Codecademy Example

Hide

```
# load data frame
greatest_books <- read_csv('top-hundred-books.csv')

#plot data
hist <- qplot(greatest_books$Ages,
      geom='histogram',
      binwidth = 3,
      main = 'Age of Top 100 Novel Authors at Publication',
      xlab = 'Publication Age',
      ylab = 'Count',
      fill=I("blue"),
      col=I("red"),
      alpha=I(.2))

hist
```

Calculating mean

Hide

```
# Set author ages to a vector
author_ages <- greatest_books$Ages

# Use R to calculate mean
average_age = author_ages %>% mean()
print(average_age)
```

Lesson 9: Median in R

Codecademy Example

Hide

```
# load data frame
greatest_books <- read_csv('top-hundred-books.csv')

#plot data
hist <- qplot(greatest_books$Ages,
      geom='histogram',
      binwidth = 3,
      main = 'Age of Top 100 Novel Authors at Publication',
      xlab = 'Publication Age',
      ylab = 'Count',
      fill=I("blue"),
      col=I("red"),
      alpha=I(.2)) +
  geom_vline(aes(xintercept=mean(greatest_books$Ages),
                 color="mean"), linetype="solid",
             size=1) +
  scale_color_manual(name = "statistics", values = c(mean = "red"))

hist
```

Calculating median

Hide

```
# Save author ages to author_ages
author_ages <- greatest_books$Ages

# Use R to calculate the median age of the top 100 authors
median_age = author_ages %>% median()
```

Lesson 10: Mode in R

Mode with DescTools

Hide

```
# Set author ages to
author_ages <- greatest_books$Ages

mode_age <- Mode(author_ages)
```

Lesson 11: Variance in R

Describing the spread of the data

Hide

```
grades <- c(88, 82, 85, 84, 90)
mean <- mean(grades)

#Change these five variables
difference_one <- (88 - mean) ^ 2
difference_two <- (82 - mean) ^ 2
difference_three <- (85 - mean) ^ 2
difference_four <- (84 - mean) ^ 2
difference_five <- (90 - mean) ^ 2

#Part 1: Sum the differences
difference_sum <- difference_one + difference_two + difference_three + difference_four + differe
nce_five

#Part 2: Average the differences
variance <- difference_sum / 5
```

Calculating Variance

Hide

```
#defining the variance of the population mean
variance <- function(x) mean((x-mean(x))^2)

teacher_one_grades <- c(80.24, 81.15, 81.29, 82.12, 82.52, 82.54, 82.76, 83.37, 83.42, 83.45, 8
3.47, 83.79, 83.91, 83.98, 84.03, 84.69, 84.74, 84.89, 84.95, 84.95, 85.02, 85.18, 85.53, 86.29,
86.83, 87.29, 87.47, 87.62, 88.04, 88.5)
teacher_two_grades <- c(65.82, 70.77, 71.46, 73.63, 74.62, 76.53, 76.86, 77.06, 78.46, 79.81, 8
0.64, 81.61, 81.84, 83.67, 84.44, 84.73, 84.74, 85.15, 86.55, 88.06, 88.53, 90.12, 91.27, 91.62,
92.86, 94.37, 95.64, 95.99, 97.69, 104.4)

#Set these two variables equal to the variance of each dataset using NumPy
teacher_one_variance <- variance(teacher_one_grades)
teacher_two_variance <- variance(teacher_two_grades)
```

Lesson 12: Standard Deviation in R

Getting standard deviation from variance

Hide

```
variance <- function(x) mean((x-mean(x))^2)

nba_variance <- variance(nba_data)
okcupid_variance <- variance(okcupid_data)

# Change these variables to be the standard deviation of each dataset.
nba_standard_deviation <- nba_variance ^ 0.5
okcupid_standard_deviation <- okcupid_variance ^ 0.5
```

Hide

```
# Change these variables to be the standard deviation of each dataset.
nba_standard_deviation <- sd(nba_data)
okcupid_standard_deviation <- sd(okcupid_data)
```

Using Standard Deviation

Hide

```
nba_mean <- mean(nba_data)
okcupid_mean <- mean(okcupid_data)
nba_standard_deviation <- sd(nba_data)
okcupid_standard_deviation <- sd(okcupid_data)

#Step 1: Calculate the difference between the player's height and the means
nba_difference <- 80 - nba_mean
okcupid_difference <- 80 - okcupid_mean

#Step 2: Use the difference between the point and the mean to find how many standard deviations
 the player is away from the mean.
num_nba_deviations <- nba_difference / nba_standard_deviation
num_okcupid_deviations <- okcupid_difference / okcupid_standard_deviation
```

Lesson 13: Quartiles

The second quartile = median

Q1 and Q3

Hide

```
# create the variables songs_q1, songs_q2, and songs_q3 here:
songs_q1 = quantile(songs, 0.25)
songs_q2 = quantile(songs, 0.5)
songs_q3 = quantile(songs, 0.75)
```

Lesson 14: Quantiles

Quantiles in R

Hide

```
# define twenty_third_percentile here:
twenty_third_percentile = quantile(songs, 0.23)
```

Multiple quantiles

Hide

```
# define quartiles and deciles here:
quartiles = quantile(songs, c(0.25, 0.50, 0.75))
print(quartiles)
```

Hide

```
# define tenth here:
deciles = quantile(songs, c(0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9))
print(deciles)
```

Lesson 15: Interqaurtile Range

Hide

```
# find the first quartile
q1 <- quantile(songs,0.25)
```

Hide

```
# calculate the third quartile here:
q3 = quantile(songs, 0.75)
```

Hide

```
# calculate the interquartile range here:
interquartile_range = q3 - q1
```

Hide

```
# create the variable interquartile_range here
interquartile_range = IQR(songs)
```

Lesson 16: Hypothesis Testing With R

Sample Mean and Population Mean

A sample is a subset of the entire population. The mean of each sample is a sample mean and it is an estimate of the population mean.
For a population, the mean is a constant value no matter how many times it's recalculated.

Hypothesis Formulation

Null hypothesis required.

"The average length of Golden Retrievers is the same as the average length of Goldendoodles."

Type I and II Errors

A Type I error occurs when a hypothesis test finds a correlation between things that are not related. This error is sometimes called a "false positive" and occurs when the null hypothesis is rejected even though it is true.
The second kind of error, a Type II error, is failing to find a correlation between things that are actually related. This error is referred to as a "false negative" and occurs when the null hypothesis is not rejected even though it is false.

Hide

```
# the true positives and negatives:
actual_positive <- c(2, 5, 6, 7, 8, 10, 18, 21, 24, 25, 29, 30, 32, 33, 38, 39, 42, 44, 45, 47)
actual_negative <- c(1, 3, 4, 9, 11, 12, 13, 14, 15, 16, 17, 19, 20, 22, 23, 26, 27, 28, 31, 34,
35, 36, 37, 40, 41, 43, 46, 48, 49)

# the positives and negatives we determine by running the experiment:
experimental_positive <- c(2, 4, 5, 7, 8, 9, 10, 11, 13, 15, 16, 17, 18, 19, 20, 21, 22, 24, 26,
27, 28, 32, 35, 36, 38, 39, 40, 45, 46, 49)
experimental_negative <- c(1, 3, 6, 12, 14, 23, 25, 29, 30, 31, 33, 34, 37, 41, 42, 43, 44, 47,
48)
```

Hide

```
# define type_i_errors and type_ii_errors here:
type_i_errors = intersect(actual_negative, experimental_positive)
type_i_errors
type_ii_errors = intersect(actual_positive, experimental_negative)
type_ii_errors
```

P-Values

P-values help determine how confident you can be in validating the null hypothesis. In this context, a p-value is the probability that, assuming the null hypothesis is true, you would see at least such a difference in the sample means of your data.

A hypothesis test on the experiment data that returns a p-value of 0.04 would indicate that, assuming the null hypothesis is true and there is no difference in preference for volleyball between all history and chemistry majors, you would see at least such a difference in sample mean (39% - 34% = 5%) only 4% of the time due to sampling error.

Significance Level

While a hypothesis test will return a p-value indicating a level of confidence in the null hypothesis, it does not definitively claim whether you should reject the null hypothesis. To make this decision, you need to determine a threshold p-value for which all p-values below it will result in rejecting the null hypothesis. This threshold is known as the significance level.

It is an industry standard to set a significance level of 0.05 or less, meaning that there is a 5% or less chance that your result is due to sampling error.

One Sample T_Test

A One Sample T-Test compares a sample mean to a hypothetical population mean. It answers the question “What is the probability that the sample came from a distribution with the desired mean?”

The first step is formulating a null hypothesis, which again is the hypothesis that there is no difference between the populations you are comparing. The second population in a One Sample T-Test is the hypothetical population you choose. The null hypothesis that this test examines can be phrased as follows: “The set of samples belongs to a population with the target mean”.

One result of a One Sample T-Test will be a p-value, which tells you whether or not you can reject this null hypothesis. If the p-value you receive is less than your significance level, normally 0.05, you can reject the null hypothesis and state that there is a significant difference.

R has a function called t.test() in the stats package which can perform a One Sample T-Test for you.

Hide

```
# calculate ages_mean here:
ages_mean = mean(ages)
```

Hide

```
# perform t-test here:
results = t.test(ages, mu = 30)
```

Two Sample T_Tests

A Two Sample T-Test compares two sets of data, which are both approximately normally distributed.

The null hypothesis, in this case, is that the two distributions have the same mean.

You can use R’s t.test() function to perform a Two Sample T-Test.

When performing a Two Sample T-Test, t.test() takes two distributions as arguments and returns, among other information, a p-value. Remember, the p-value let's you know the probability that the difference in the means happened by chance (sampling error).

Hide

```
# run two sample t-test here:
results = t.test(week_1, week_2)
print(results)
```

ANOVA

When comparing more than two numerical datasets, the best way to preserve a Type I error probability of 0.05 is to use ANOVA. ANOVA (Analysis of Variance) tests the null hypothesis that all of the datasets you are considering have the same mean. If you reject the null hypothesis with ANOVA, you're saying that at least one of the sets has a different mean; however, it does not tell you which datasets are different.

You can use the stats package function aov() to perform ANOVA on multiple datasets. aov() takes the different datasets combined into a data frame as an argument.

Assumptions of Numerical Hypothesis Testing

The samples should each be normally distributed…ish

Data analysts in the real world often still perform hypothesis tests on datasets that aren't exactly normally distributed. What is more important is to recognize if there is some reason to believe that a normal distribution is especially unlikely. If your dataset is definitively not normal, the numerical hypothesis tests won't work as intended.

The population standard deviations of the groups should be equal

For ANOVA and Two Sample T-Tests, using datasets with standard deviations that are significantly different from each other will often obscure the differences in group means.

To check for similarity between the standard deviations, it is normally sufficient to divide the two standard deviations and see if the ratio is "close enough" to 1. "Close enough" may differ in different contexts, but generally staying within 10% should suffice.

The samples must be independent

When comparing two or more datasets, the values in one distribution should not affect the values in another distribution. In other words, knowing more about one distribution should not give you any information about any other distribution.

It is important to understand your datasets before you begin conducting hypothesis tests on them so that you know you are choosing the right test.