---
title: "Assignment3_DS_2003"
output: pdf_document
date: "2024-02-15"

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
install.packages("tinytex")
```

Dataframes

Built-in Data Frames

R and associated packages have numerous built-in data frames which we will utilize in class.

'Built-in' means we don't have to load the data from a csv/excel/database. The data is already stored as dataframe in R or R package.

Number 1

One of these datasets is `mtcars`. Type mtcars in the code chunk below and run the code.
#Similar to another assignment in DS 2002
```{r}
 mtcars
```

- You should get a pretty nice looking table with cars as observations and variables about each car's features

Number 2

Before going further, it is good practice to know more about the data you are working with.

In the code chunk below use the function `help()` to find out more about `mtcars`.

```{r}
help(mtcars)
```

- The R documentation for the `mtcars` dataset should have shown up in the 'Help' tab of your RStudio.

- Take a quick scan of what is included in the dataset. Look at what variable abbreviation stand for.

Number 3

The number of data rows in the data frame is given by the `nrow()` function.

How many rows are in `mtcars`?

```{r}
nrow(mtcars)
```

Number 4

And the number of columns of a data frame is given by the `ncol()` function.

How many columns are in `mtcars`?

```{r}
ncol(mtcars)
```

Number 5

Dimensions refers to the data frames rows x columns. You can find the dataframe dimensions using `dim()`

What are the dimensions of `mtcars`?
#Dimensions are 32 x 11
```{r}
dim(mtcars)
```

Number 6
Another useful exploratory function for data frames in `str()`.

Try `str()` with `mtcars below. What does this function tell you about the df?
#str tells us that mtcars is a dataframe with 32 observations from 11 variables

```{r}
str(mtcars)
```

Number 7

We can also preview a data frame with `head()`

Instead of printing out the entire data frame, it is often desirable to
preview it with the head function beforehand.

Try this function below with `mtcars`

```{r}
head(mtcars)
```

Number 8

It is also useful to have a larger view of the data. Try the function `view()` with `mtcars` below.

Explore the functionality of the new tab.

#Opens new tab in R with larger view of dataframe
```{r}
View(mtcars)
```

Number 9

- Now we are going to subset the dataset using Base R syntax. We will cover dplyr (TV) syntax next class.
- Base R syntax can be very useful for navigating data frames.

To retrieve data in a cell, we would enter its **row** and **column**
coordinates in the single square bracket `[ ]` operator.

The two coordinates are separated by a comma, e.g. `[row, col]`.

For example, to retrieve the cell value from the first row, second column of mtcars the syntax would be `mtcars[1, 2]`.

Note: R uses 1-based indexing

In the code chunk below retrieve the cell value from the 5th row and 3rd column

```{r}
mtcars[5, 3]
```

Number 10

In data frames, we can use names instead of the numeric coordinates.

Example: if you want to know how many miles per gallon (mpg) the Duster 360 gets, the syntax is `mtcars["Duster 360", "mpg"]`

- Note the use of the `""` when calling variables by name.

How many cylinders does the Mazda RX4 have?

```{r}
mtcars["Mazda RX4", "cyl"]

```

Number 11

We reference a data frame column with the double square bracket `[[]]` operator, just as we do for lists.

For example, to retrieve the ninth column vector of the built-in data set mtcars, we write `mtcars[[9]]`

Below reference the 2nd column vector of `mtcars`

```{r}
mtcars[[2]]
```

Number 12

We can retrieve the same column vector by its name

Example: `mtcars[["am"]]`

Try this for column 2.

```{r}
mtcars[["cyl"]]
```

Number 13

We can also retrieve with the column using the `$` operator in lieu of the double square bracket operator

Example: `mtcars$am`

Try for cylinders below

```{r}
mtcars$cyl
```

Number 14
Yet another way to retrieve the same column vector is to use the single square bracket "[]" operator.

We pre-pend the column name with a comma character, which signals a wildcard match for the row position

Example: `mtcars[, "am"]`

Try this syntax for cylinders below.

```{r}
mtcars[, "cyl"]
```

Data Frame Column Slice

We retrieve a data frame column slice with the single square bracket `[ ]` operator.

Number 15

Numeric Indexing

The following is syntax for a slice containing the fifth column of the built-in data set `mtcars`

`mtcars[5]`

Try a column slice using numeric indexing for column 1 of the `mtcars` data frame.

```{r}
mtcars[, 1]
```

Number 16

Name Indexing

We can retrieve the same column slice by its name.

Example: `mtcars["drat"]`

```{r}
mtcars["drat"]
```

Number 17

To retrieve a data frame slice with the two columns `mpg` and `hp`, we pack the column names in an index vector inside the single square bracket operator

`mtcars[c("mpg", "hp")]`

Try with any other two variables in the data frame.

```{r}
mtcars[c("cyl", "disp")]
```

Data Frame Row Slice

We retrieve rows from a data frame with the single square bracket operator, just like what we did with columns. However, in additional to an index vector of row positions, we append an extra comma character. This is important, as the extra comma signals a wildcard match for the second coordinate for column positions.

Number 18

Numeric Indexing

For example, the following retrieves a row record of the built-in data set mtcars. Please notice the extra comma in the square bracket operator, and it is not a typo. It states that the 1974 Camaro Z28 has a gas mileage of 13.3 miles per gallon, and an eight cylinder 245 horse power engine, ..., etc

```{r}
mtcars[24,]
```

Try this syntax for row 15

```{r}
mtcars[15,]
```

Note: To retrieve more than one rows, we use a numeric index vector
#Need concatention "c" in front of the two rows, separated by comm in parenthesis. Comma exists after parenthesis all enclosed in square brackets
```{r}
mtcars[c(3, 24),]
```

Number 19

Name Indexing

We can retrieve a row by its name.

```{r}
mtcars["Camaro Z28",]
```

Try this syntax for the car you found the numeric indexing above.
#Datsun 710 and Camaro Z28
```{r}
mtcars[c("Datsun 710", "Camaro Z28"),]
```

Note: And we can pack the row names in an index vector in order to retrieve multiple rows.

```{r}
mtcars[c("Datsun 710", "Camaro Z28"),]
```

Number 20

Logical Indexing

Lastly, we can retrieve rows with a logical index vector. In the
following vector L, the member value is TRUE if the car has automatic
transmission, and FALSE if otherwise.

```{r}
L <- mtcars$am == 0
L
```

Here is the list of vehicles with automatic transmission

```{r}
mtcars[L,]
```

And here is the gas mileage data for automatic transmission
#mpg data for automatic transmission in mtcars
```{r}
mtcars[L,]$mpg
```

Duplicate the code above to find the mpg of all 4 cylinder cars. Comment your code as you go (e.g., write what you code is doing)

```{r}
L_cyl <- mtcars$cyl == 4 #In the following vector L_cyl, the member value is TRUE if the car has four cylinders, and FALSE if otherwise
mtcars[L_cyl, ]$mpg #Given the cars that have four cylinders, find the mpg of these cylinder cars
```

Note

Syntax for change the name of a column in base R

`names(df)[names(df) == 'old.var.name'] <- 'new.var.name'`

Dplyr

Import the `tidverse` package
```{r}
#Import packages
library(tidyr)
library(dplyr)
library(ggplot2)
```

Question 21
`View()` the built-in dataset `diamonds`
How many columns are in the dataframe?
```{r}
View(diamonds)


# How many columns are in the dataframe?
# There are 10 columns in the diamonds datframe
```

Question 22
Check the structure of the dataframe.
```{r}
str(diamonds)



# What is the most common data type? (consider int & num to be the same data type)
# The most common data type is int & num
```

Question 23
Subset the dataframe using numerical indices to include the columns `carat`, `clarity`, `color` & `price`

```{r}
subset_numbers <- diamonds[, c("carat", "clarity", "color", "price")]

```

Number 24
Subset the dataframe using column names the dataframe to include the columns `carat`, `clarity`, & `price`
```{r}
```

subset_names <- diamonds[, c("carat", "clarity", "price")]

```

**Number 25**
Drop the column `table` from the dataframe.
#select a column, then use -table
```{r}
diamonds <- subset(diamonds, select = -table)


```

Numbe 26
Select all columns in the dataframe that start with the letter "c". Print the first 5 rows.
```{r}
c_columns <- diamonds[, grep("^c", names(diamonds))]
head(c_columns, 5)
```

Number 27
What is the primary difference between the functions `select()` & `filter()`?
```{r}
# Type answer here
#select() is used to choose between columns while filter() is used to choose rows based on conditions.
```

Number 28
Subset the dataframe to only include observations where `price` is greater than `2000`. Include only the last 5 observations.
Hint: Save to a new dataframe
```{r}
#subset price > 2000
subset_price <- subset(diamonds, price > 2000)
tail(subset_price, 5)

```

Number 29
How many observations have a price > 10000? What percent of the entire dataframe is that?
```{r}
price_gt_10k <- sum(diamonds$price > 10000)

```

```{r}
percent_gt_10k <- (price_gt_10k / nrow(diamonds)) * 100
percent_gt_10k
#9.68% of the entire dataframe have observations with a price > 10000
```

Number 30
Sort the `diamonds` dataframe in reverse alphabetical order by `color`
```{r}
diamonds <- diamonds[order(diamonds$color, decreasing = TRUE),]

```

Number 31
What is the average price by diamond cut?
```{r}
aggregate(price ~ cut, data = diamonds, FUN = mean)
```

Number 32
Calculate a price x depth variable. Then find the average of your new variable by color.
```{r}
diamonds$price_depth <- diamonds$price * diamonds$depth
aggregate(price_depth ~ color, data = diamonds, FUN = mean)
# The trend suggests that as color rating decreases (i.e., becomes worse), the price per depth tends to increase, indicating potentially higher value for diamonds with lower color ratings.
```
```