R Markdown

Files	
Notes	
Status	
Topics	Data Camp

Benefits

- Ensures reproducible results

Overview of DataCamp Course

1. Elements of an R Markdown document
2. Add analyses and plots
3. Organize and improve a report
4. Customize fonts and colors

R Markdown Elements

1. **Code**

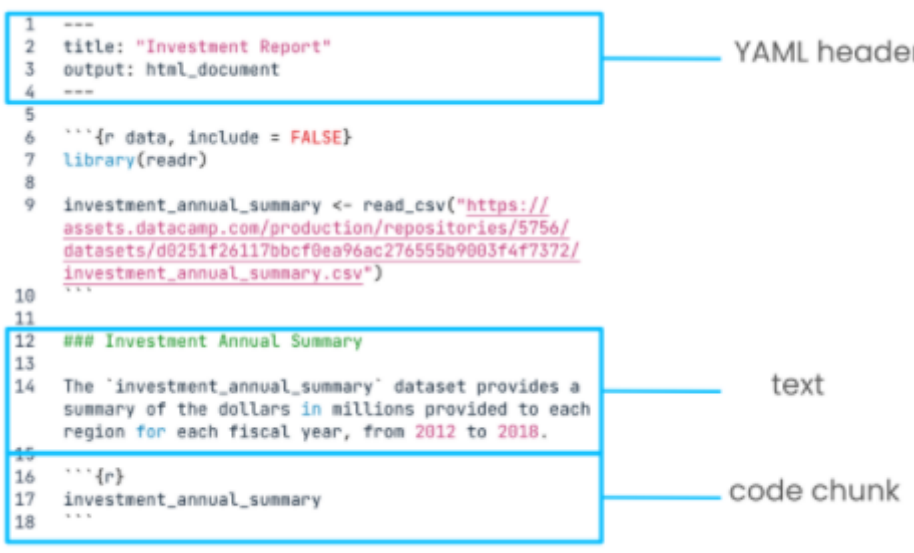
    ▼ **Adding code chunks**

    ```
    ```{r}
    # Code
    ```
    ```

    - Keyboard Shortcut : `Ctrl Alt I`

2. **Text**
3. **Metadata**

▼ **Structure in the file**



▼ **Help**

- Go under `Help` → `Markdown Quick Reference`

Knit

- Helps to convert our R Markdown file into a HTML file

````
---
title: "Investment Report"
output: html_document
---

```{r data, include = FALSE}
library(readr)

investment_annual_summary <- read_csv("https://
assets.datacamp.com/production/repositories/5756/
datasets/d0251f26117bbcf0ea96ac276555b9003f4f7372/
investment_annual_summary.csv")
```

### Investment Annual Summary

The `investment_annual_summary` dataset provides a
summary of the dollars in millions provided to each
region for each fiscal year, from 2012 to 2018.

```{r}
investment_annual_summary
```
````

Investment Report

Investment Annual Summary

The `investment_annual_summary` dataset provides a summary of the dollars in millions provided to each region for each fiscal year, from 2012 to 2018.

```
investment_annual_summary
```

```
## # A tibble: 42 x 3
##    fiscal_year region                          dollars_in_millions
##          <dbl> <chr>                                       <dbl>
##  1        2012 East Asia and the Pacific                    2548
##  2        2012 Europe and Central Asia                      2915
##  3        2012 Latin America and the Caribbean              3680
##  4        2012 Middle East and North Africa                 2210
##  5        2012 South Asia                                   1312
##  6        2012 Sub-Saharan Africa                           2733
##  7        2013 East Asia and the Pacific                    2873
##  8        2013 Europe and Central Asia                      3261
##  9        2013 Latin America and the Caribbean              4822
## 10        2013 Middle East and North Africa                 2038
## # ... with 32 more rows
```

Example of the output that will be generated when the file is knitted

- Keyboard Shortcut for Knitting: `Ctrl Shift K`

Customizing Fonts and Colors

▼ Adding Titles

```
---
title = "Title"
---
```

- Add the `title` under the YAML header

▼ Adding LATEX

```
$\alpha$
```

▼ Adding Headers

```
# Biggest
## Big
### Smaller
```

- The more `#` we add, the smaller the headers

▼ Adding sentences

````
## Headers

Sentence. We can type anything here

```{r}
code_chunk
```
````

- We can type in any sentences that we need anywhere in the R Markdown file. The text will be displayed as it is when we Knit the file

▼ Code Comments

````
```{r}
# This is a code comment
code_chunk
```
````

- We can add in a code comment under the code chunk using `#`

Formatting Text

```
**bold** or __bold__
*italics* or _italics_
~~strikethrough~~
```

- We can use the various symbols to format the text accordingly

Inline Code

```
# Header
The `in_line_code` is typed in using this symbol

```{r}
# This is a code comment
code_chunk
```
```

- Inline code can be added using `` `code` `` while we are writing text

Including Links

```
# Header
This text has the following link to be displayed [text_to_be_displayed](url_link)
```

- The text that we want to be displayed will be placed under `[ ]` and the url link will be placed in `()` right after it with no spaces between them

Including Images

```
# Header

This image will be added ![](url_of_image)
```

- For images, it is the same as links just that we add a `!` before the `[]`

YAML header

- Contains meta data about the file in a key-value pair format
- Appears at the top of the document

```
---
title: "Title"
author: "Author"
output: html_document
---
```

Output types

- Under the `output` key in the YAML header, we can specify different output types
- In the course we will use:
  - `html_document`: HTML Document
  - `pdf_document` : PDF Document

Dates

Adding Dates Manually

```
---
title: "Title"
author: "Author"
date: "14 August 2021"
```

```
output: html_document
---
```

▼ Adding Dates Automatically

```
---
title: "Title"
author: "Author"
date: "`r Sys.Date()`"
output: html_document
---
```

▼ Formatting the date

```
---
title: "Title"
author: "Author"
date: "`r format(Sys.time(), '%d %B %Y')`"
output: html_document
---
```

Date formatting options

Text		Numeric	
%A or %a	Weekday	%d	Decimal date
%B or %b	Month	%m	Decimal month
		%Y or %y	Year

[1] https://www.rdocumentation.org/packages/base/topics/strptime

Capital letters for the full name and lower case is for the abbreviated one
%Y (Full 4 digit year) %y (2 Digit year)

```
---
title: "Title"
author: "Author"
date: "`r format(Sys.time(), '%d %B,  %Y')`"
output: html_document
---
```

▼ Other date options

```
---
title: "Title"
author: "Author"
date: "Last edited `r format(Sys.time(), '%B, %d, %Y')`"
output: html_document
---
```

Analyzing the Data

▼ Data Manipulation

▼ R Functions

▼ `select()`

- Objective: Select/Exclude the variables
- **Code:**
  - `select(df, A, B, C)` - Select the variables A, B and C
  - `select(df, A:C)` - Select all variables from A to C
  - `select(df, -C)` - Exclude C

▼ `filter()`

- Objective: Filter the df based on one or many conditions
- **Code:**
  - `filter(df, condition 1)` - Selecting based on 1 condition
  - `filter(df, condition 1, condition 2)` - Selection must satisfy both `conditon 1` and `condition 2`
  - `filter(df, condition 1 & condition 2)` - Selection must satisfy both `conditon 1` and `condition 2`
  - `filter(df, condition 1 | condition 2)` - Selection must satisfy **either** `conditon 1` **or** `condition 2`

▼ `dplyr` library

- Contains various verbs that helps in making the arrangement of data much easier

▼ `glimpse`

- Under the `dplyr` library
- Objective: Check the structure of a df
- **Code:**
  - `glimpse(df)` - Identical to `str()`

▼ `arrange()`

- Under the `dplyr` library
- Objective: Sort the dataset with one or many variables
- **Code:**
  - `arrange(A)` - Ascending sort of variable A
  - `arrange(A, B)` - Ascending sort of variable A and B
  - `arrange(desc(A), B)` - Descending sort of variable A and ascending sort of B

▼ `%>%` (Pipeline)

- Objective: Create a pipeline between each step
- **Code:**
  - `step 1 %>% step2 %>% step3` - We can carry out each step one after another
- **Benefits:**
  - We can just create the `data.frame` object first and continue to call various selection criteria of our criteria without stating the data
- **Example**

```
# Create the data frame filter_home_wed.It will be the object return at the end of the pipeline
filter_home_wed <-

#Step 1
read.csv(PATH) % > %

#Step 2
select(GoingTo, DayOfWeek) % > %

#Step 3
filter(GoingTo == "Home",DayOfWeek == "Wednesday")
identical(step_3, filter_home_wed)
```

▼ `group_by()`

- Objective: Groups data based on a column
- **Code:**
  - `group_by(mtcars, cy1)` - Group the `mtcars` data frame based on the `cy1` column
- **Note:**
  - There will not be much changes to the way the data in the data frame is displayed other than it will be ordered in the way it is grouped
  - The effectiveness comes in when it is used with other verbs in `dplyr` like `filter, select, arrange, summarize`

▼ `summarize()`

- Objective: Creates a new data frame. It will have one (or more) rows for each combination of grouping variables; if there are no grouping variables, the output will have a single row summarising all observations in the input. It will contain one column for each grouping variable and one column for each of the summary statistics that you have specified
- **Code:**
  - `summarize(mtcars, avg_hp = mean(hp, na.rm = TRUE))` - Summarizes the `mtcars` dataset we will have a column `avg_hp` which gives a mean of the `hp` of all the cars and we will only have 1 value. `na.rm=TRUE` helps to remove all empty values
  - `summarize(group_by(mtcars, cy1), avg_hp = mean(hp, na.rm = TRUE))` - Summarizes the `mtcars` dataset by the groups that was split in `cy1` and we will have a column `avg_hp` which gives a mean of the `hp` of each of the groups that is under `cy1` and `na.rm=TRUE` helps to remove all empty values
- **Note:**
  - Do note that we can specify multiple variables and conditions `variable = condition` and it each of the variable will be a new column in the data frame
- **Useful Functions**
  - Center: `mean(), median()`
  - Spread: `sd(), IQR(), mad()`
  - Range: `min(), max(), quantile()`
  - Position: `first(), last(), nth(),`
  - Count: `n(), n_distinct()`
  - Logical: `any(), all()`

▼ **Loading Packages**

````
---
title: "Investment Report"
date: "`r format(Sys.time(), '%d %B %Y')`"
output: html_document
---

```{r data, include = FALSE}
library(readr)
library(dplyr)

investment_annual_summary <- read_csv("https://assets.datacamp.com/production/repositories/5756/datasets/d0251f26117bbcf0ea96ac276555b9003f4f7372/investment_annual_summary.csv")
investment_services_projects <- read_csv("https://assets.datacamp.com/production/repositories/5756/datasets/bcb2e39ecbe521f4b414a21e35f7b8b5c50aec64/investment_services_projects.csv")
```
````

When loading packages, we will try to load them all at once at the first code chunk so that we have a cleaner code and also we will not duplicate reading in of libraries

▼ **Including code results in text**

```{r}
indonesia_investment_projects_2012 <- investment_projects %>%
  filter(country == "Indonesia",
        date_disclosed >= "2011-07-01",
        date_disclosed <= "2012-06-30")

indonesia_investment_projects_2012_total <- indonesia_investment_projects_2012 %>%
  summarize(sum_total_investment = sum(total_investment, na.rm = TRUE)) #na.rm = TRUE will mean that we will exclude values whe
re which are empty
```

The total investment amount of all projects in Indonesia in the 2012 fiscal year was`r indonesia_investment_projects_2012_total
`million dollars.

▼ Naming code chunks

```
25  ### Investment Projects in Indonesia
26
27  The `investment_services_projects` dataset provides information about each investme
    project from 2012 to 2018. Information listed includes the project name, company na
    sector, project status, and investment amounts.
28
29  ```{r indonesia-investment-projects}
30  indonesia_investment_projects_2012 <- investment_services_projects %>%
31    filter(country == "Indonesia",
32           date_disclosed >= "2011-07-01",
33           date_disclosed <= "2012-06-30")
34
35  indonesia_investment_projects_2012
36  ```
37
38  ### Investment Projects in Indonesia in 2012
39  ```{r indonesia-investment-projects-2012}
40  indonesia_investment_projects_2012 <- investment_services_projects %>%
41    filter(country == "Indonesia",
42           date_disclosed >= "2011-07-01",
43           date_disclosed <= "2012-06-30")
```

We can name the code chunk so that it is easier to refer to when we are debugging or trying to find a specific section. It can be done by putting the name in `{r name_of_code_chunk}` after the ``` of the start of the code chunk

▼ Adding Plots

- **Libraries**
  - `ggplot2`
- `ggplot()`
  - Creates a new `ggplot` object. It can be used to declare the input data frame for a graphic and to specify the set of plot aesthetics intended to be common throughout all subsequent layers unless specifically overridden

  #### ▼ Usage

  ▼ `ggplot(df, aes())`

  - Create a plot based on the data and the various arguments passed into the `aes()` function

  ▼ `aes()` - Provides the aesthetics to the plot

  - `x = variable` - Provides the variable that should be taken as the x-axis
  - `y = variable` - Provides the variable that should be taken as the y-axis
  - `color = variable` - Provides the variable that should be used to group the different data
    - For different values of the variable, it will provide a different color to the plot

  ▼ `geom_point()` - Creates a scatterplot

  - We can use it after the `ggplot()` function is called
  - For example: `ggplot() + geom_point()`

  ▼ `geom_line()` - Creates a line plot

  - We can use it after the `ggplot()` function is called
  - For example: `ggplot() + geom_line()`

  ▼ `labs()` - Add labels to the `ggplot` object

- `title = "title"` - Labels the title of the plot
- `x = "x_label"` - Labels the x-axis of the plot
- `y = "y_label"` - Labels the y-axis of the plot

▼ **Note**

- There could be warnings when we try to plot the `ggplot` object due to missing data and it will show a warning message when that happens

▼ **Plot Options**

- Requires the library `knitr`

▼ **Figure Dimensions**

- Can be used to alter the dimensions of the output in the code chunk and is in terms of inches whereby we just need to state the numerical value
- `fig.width`
- `fig.height`
- `fig.dim`

▼ **Usage**

````
```{r brazil-investment-projects, fig.width = 5, fig.height = 3}
brazil_investment_projects <- investment_services_projects %>%
  filter(country == "Brazil")

ggplot(brazil_investment_projects, aes(x = date_disclosed, y = total_investment, color = status)) +
  geom_point() +
  labs(
    title = "Investment Services Projects in Brazil",
    x = "Date Disclosed",
    y = "Total IFC Investment in Dollars in Millions"
  )
```
````

````
```{r brazil-investment-projects, fig.dim = c(5, 3)}
brazil_investment_projects <- investment_services_projects %>%
  filter(country == "Brazil")

ggplot(brazil_investment_projects, aes(x = date_disclosed, y = total_investment, color = status)) +
  geom_point() +
  labs(
    title = "Investment Services Projects in Brazil",
    x = "Date Disclosed",
    y = "Total IFC Investment in Dollars in Millions"
  )
```
````

▼ **Output Dimensions**

- Can be used to alter the dimensions of the output in the code chunk and is in terms of percentages `%` and it is placed under quotations `'' or " "`
- `out.width`
- `out.height`

▼ **Usage**

````
```{r brazil-investment-projects, out.width = '95%', out.height = '95%'}
brazil_investment_projects <- investment_services_projects %>%
  filter(country == "Brazil")

ggplot(brazil_investment_projects, aes(x = date_disclosed, y = total_investment, color = status)) +
  geom_point() +
  labs(
    title = "Investment Services Projects in Brazil",
    x = "Date Disclosed",
    y = "Total IFC Investment in Dollars in Millions"
  )
```
````

▼ Figure Alignment

- `fig.align` - Options:
  - `'left'`
  - `'right'`
  - `'center'`

▼ Usage

````
```{r brazil-investment-projects, fig.align = 'center'}
brazil_investment_projects <- investment_services_projects %>%
  filter(country == "Brazil")

ggplot(brazil_investment_projects, aes(x = date_disclosed, y = total_investment, color = status)) +
  geom_point() +
  labs(
    title = "Investment Services Projects in Brazil",
    x = "Date Disclosed",
    y = "Total IFC Investment in Dollars in Millions"
  )
```
````

▼ Local vs. Global Options

- We can either set the options locally under a single code chunk or globally if it is repeated for all code chunks

▼ Usage

````
```{r setup, include = FALSE}
knitr::opts_chunk$set(fig.align = 'center', output.width = '80%', echo = TRUE)
```
````

````
```{r brazil-investment-projects, fig.align = 'center'}
brazil_investment_projects <- investment_services_projects %>%
  filter(country == "Brazil")

ggplot(brazil_investment_projects, aes(x = date_disclosed, y = total_investment, color = status)) +
  geom_point() +
  labs(
    title = "Investment Services Projects in Brazil",
    x = "Date Disclosed",
    y = "Total IFC Investment in Dollars in Millions"
  )
```
````

▼ Adding captions

- We can add the captions that we want for the figure under the `fig.cap` in `' ' or " "`
- `fig.cap`

▼ Usage

````
```{r brazil-investment-projects, fig.cap = 'This is about Brazil Investment Projects'}
brazil_investment_projects <- investment_services_projects %>%
  filter(country == "Brazil")

ggplot(brazil_investment_projects, aes(x = date_disclosed, y = total_investment, color = status)) +
  geom_point() +
  labs(
    title = "Investment Services Projects in Brazil",
    x = "Date Disclosed",
    y = "Total IFC Investment in Dollars in Millions"
  )
```
````

Organizing and Improving the report

▼ Lists and Tables

▼ Bulleted Lists

- Bulleted Items are added using `-` or `*` or `+`

Usage

```
- Region
  * East Asia and the Pacific
  + Europe and Central Asia
```

▼ Numbered Lists

- Numbered Items can be added by writing the number and a period `1.`

Usage

```
Region

1. East Asia and the Pacific
2. Europe and Central Asia
  1. Indented numbered list
```

▼ Tables

- Requires the library `knitr`
- Can be done by using `kable()`

▼ Usage

````
```{r tables}
kable(indonesia_investment_projects_2012_summary)
```
````

▼ Note

- Formatting of the data, e.g. combining cell data should be done before using `kable()`

▼ Modifying table column names

- Can be done by stating the `col.names`

▼ Usage

````
```{r tables}
kable(indonesia_investment_projects_2012_summary, col.names= c("Project Name", "Status", "Total Investment")
```
````

▼ Table Alignment

- Default alignment:
  - Right : `numeric`
  - Left: Everything else
- Can be done by stating `align` and setting the alignment for each of the columns that are in the table
- **Alignments:**
  - `'l'` - left
  - `'r'` - right
  - `'c'` - center

▼ Usage

```
```{r tables}
kable(indonesia_investment_projects_2012_summary, col.names= c("Project Name", "Status", "Total Investment", align
= "ccc")
```
```

▼ Adding Table Caption

- Can be done by stating the `caption`

▼ Usage

```
```{r tables}
kable(indonesia_investment_projects_2012_summary, col.names= c("Project Name", "Status", "Total Investment"), capt
ion = "Text" )
```
```

▼ Code Chunk Options

▼ `include` Option

- Decides whether the code and results appear in the knitted file
- **Default** value: `TRUE`
- If the value is set to `FALSE` , it will not appear in the knitted file
- **Note**: Even though we do not show the code or the results, the code still runs, whereas for `eval`, it does not run and does not show

▼ `echo` Option

- Decides whether the code appears in the knitted file
- **Default** value : `TRUE`
- If the value is set to `FALSE`, the code will not appear in the knitted file and only the results will appear

▼ `eval` Option

- Decides whether we want to evaluate the code in the code chunk
- **Default** value : `TRUE`
- If the value is set to `FALSE`, the code will not run, however the code still appears in the knitted file but since it does not run, there is also no results to show

Code Option Summary

Aa Name	Code is run	Code appears in report	Results appear in report
include = FALSE	Yes	No	No
echo = FALSE	Yes	No	Yes
eval = FALSE	No	Yes	No

▼ `collapse` Option

- Decides whether code and any text are split into multiple blocks or as a single block
- **Default** value : `FALSE`
- If the value is set to `FALSE`, the output will be split into multiple blocks, whereas if the value is set to `TRUE`, the output will be as a single block

▼ Warnings, Messages and Errors

▼ `warning` Option

- Decides whether any warnings show up in the knitted file
  - Warnings are shown when the code exclude data from the final results as the values could be missing
- **Default** value: `TRUE`
- If the value is set to `FALSE` , there will not be any warning messages shown even if there are warnings

▼ `message` Option

- Decides whether any messages show up in the knitted file
  - Messages are shown due to the packages and data files that are loaded
- **Default** value: `TRUE`
- If the value is set to `FALSE` , the messages will not be shown

▼ `error` Option

- Decides how errors are displayed in the knitted file
  - Errors refer to the errors that occur when we run the code chunks
- **Default** value: `FALSE`
- If the value is set to `TRUE`, the code will still knit even if there are errors and will show the error
- If the value is set to `FALSE`, the code will stop running when it encounters an error

Customizing the Report

▼ Adding a table of contents

▼ Location

- If we want to show the Table of contents, we have to specify `toc : true`
- It is under the `output -> html_document` of the YAML header
- Note that all the fields should have no spaces with the colon `field:`

```
---
title: "Investment Report"
output:
  html_document:
    toc: true
date: "`r format(Sys.time(), '%d %B %Y')`"
---
```

▼ TOC Depth

- We can specify the depth and state the level of headings that we want to include under the Table of contents
- **Default:**
  - HTML Documents : `3`
  - PDF Documents : `2`
- Use `toc_depth : 2` , with the desired depth of the headers that we want
  - The depth is determined by the number of `#` that is placed before the header as it goes from largest to smallest
  - If we put `2` , we will take all headers that have 2 or lesser `#` into the TOC

```
---
title: "Investment Report"
output:
  html_document:
    toc: true
    toc_depth: 2
date: "`r format(Sys.time(), '%d %B %Y')`"
---
```

▼ Number sections

- We just have to specify `number_sections : true` to show that the numbering
- **Note:**
  - If we do not have the largest header with only 1 `#` , and only have `##` headers, the TOC numbering will be `0.1 Header` and it will be the same for the different levels of headings

```
---
title: "Investment Report"
output:
  html_document:
    toc: true
    toc_depth: 2
    number_sections: true
date: "`r format(Sys.time(), '%d %B %Y')`"
---
```

▼ TOC Float

- We just have to specify `toc_float: true` to make the TOC on the left side of the screen and stays there while scrolling

```
---
title: "Investment Report"
output:
  html_document:
    toc: true
    toc_depth: 2
    number_sections: true
    toc_float: true
date: "`r format(Sys.time(), '%d %B %Y')`"
---
```

- **Note**:
  - When `collapsed` or `smooth_scroll` is used, we do not need to put `toc_float : true` as it will automatically be deemed as `true`
  - This is only for `HTML` documents only

▼ `collapsed`

- **Default:** `TRUE`
- Whether the TOC displays only the largest headers and it will expand when the person is inside a section or is going into another section
- When `collapsed` is set as `FALSE` the full TOC will be shown

```
---
title: "Investment Report"
output:
  html_document:
    toc: true
    toc_depth: 2
    number_sections: true
    toc_float:
      collapsed: false
date: "`r format(Sys.time(), '%d %B %Y')`"
---
```

▼ `smooth_scroll`

- **Default:** `TRUE`
- Whether when the user clicks on a section of the TOC, there will be an animation that scrolls to the section
- When `collapsed` is set as `FALSE` there will not be any animation and the section will automatically be displayed there

```
---
title: "Investment Report"
output:
  html_document:
    toc: true
    toc_depth: 2
    number_sections: true
    toc_float:
      smooth_scroll: false
date: "`r format(Sys.time(), '%d %B %Y')`"
---
```

▼ Creating a report with a parameter

▼ Parameters

- Enables us to create reports for different people/countries etc.

▼ Adding a Parameter

- Add it under the YAML header with `params` field

```
---
title: "Investment Report"
output:
  html_document:
    toc : true
    toc_depth : 2
    number_sections : true
    toc_float : true
date: "`r format(Sys.time(), '%d %B %Y')`"

params:
  country : Indonesia
---
```

▼ Reviewing the Code

1. We will need to review the code that has any references to the parameters that we have stated and make it more general
2. Change any references to the value of the parameter to `params$name_of_param` where in this case the parameter is `params$country`
    - This is so that we can change the parameter value. This is similar to using a variable so that we do not have to change the value for every instance and we can just change the variable assignment instead
3. Changing the reference under the text is also the same, we just have to specify `` `r params$name_of_param` `` and we should also review to ensure that there is no specific reference to the parameter

▼ Reviewing the YAML Header

- If we need to change anything in the YAML Header with reference to the parameter for example the title, we can use the same way as for text `` `r params$name_of_param` ``

▼ Changing the parameter values for different reports

- We just have to go under the `params` field under the YAML heading to change the value of the parameters so that change is reflected throughout the document

▼ Multiple Parameters

- If we require more parameters to be included in the file, we just need to list it under the `params` field

```
---
title: "Investment Report"
output:
  html_document:
    toc : true
    toc_depth : 2
    number_sections : true
    toc_float : true
```

```
date: "`r format(Sys.time(), '%d %B %Y')`"

params:
  country : Indonesia
  year_start : 2011-07-01
  year_end : 2012-06-30
  fy : 2012
---
```

- We just need to repeat the same checks for single parameters and we can begin to change the values of the parameters

▼ Fonts and Colors Customizing

▼ Specifying Element Style

- We make use of the `<style> </style>` tags
- This section can be added anywhere in the document but we will add it at the top to keep the document organized
- Note that when we are specifying properties that should not be a space between the properties and colon `property:`
- **Common Properties:**
  - `color` - Change the color of text
  - `background-color` - Change the background color of text
  - `font-family` - Change the font type of the text
  - `font-size` - Change the font size of the text (in pixels `px`)

▼ Document Style

- To change the style for the whole text
- It is under the `body{}` field

```
<style>
body{
  color: red;
  font-family: Calibri;
  background-color: red;
}
</style>
```

```
<style>
body{
  color: #708090;
  font-family: Calibri;
  background-color: #F5F5F5;
}
</style>
```

▼ Code Chunks

- To change the style for code chunk portions
- It is under the `pre{}` field

```
<style>
pre{
  color: red;
  background-color: red;
}
</style>
```

```
<style>
pre{
  color: #708090;
  background-color: #F5F5F5;
}
</style>
```

▼ Table of Contents

- To change the style for table of contents
- It is under the `#TOC{}` field

```
<style>
#TOC{
  color: red;
  font-family: Calibri;
  font-size: 16px;
  border-color: red;
}
</style>
```

```
<style>
#TOC{
  color: #708090;
  font-family: Calibri;
  font-size: 16px;
  border-color: #F5F5F5;
}
</style>
```

▼ Headers

- To change the style headers
- It is under the `#header{}` field
- **Note**:
  - `opacity` default value is `1` which means it is totally opaque and if we set it to `0` it will be transparent
  - The header settings also affects the title, author and date

```
<style>
#header{
  color: #708090;
  background-color: #F5F5F5;
  opacity: 0.6; # Determined between 0 and 1 and determines how opaque the background of the headers are
  font-family: Calibri;
  font-size: 20px;
}
</style>
```

▼ Title, Author and Date

- To change the style of the title, author and date
- **Fields**
  - Title : `h1.title`
  - Author : `h4.author`
  - Date : `h4.date`
- **Note**:
  - `opacity` default value is `1` which means it is totally opaque and if we set it to `0` it will be transparent

```
<style>
h1.title{
  color: #708090;
  background-color: #F5F5F5;
  opacity: 0.6; #
  font-family: Calibri;
  font-size: 20px;
}

h4.author{
  color: #708090;
  font-family: Calibri;
}
```

```
h4.date{
  color: #708090;
  font-family: Calibri;
}
</style>
```

▼ CSS File

- We can include all the information of the fields that we require so that we do not need to copy and paste into other Markdown files if they have the same styles

```
#TOC{
  color: #708090;
  font-family: Calibri;
  font-size: 16px;
  border-color : #F5F5F5;
}

h1.title{
  color: #708090;
  background-color: #F5F5F5;
  opacity: 0.6; #
  font-family: Calibri;
  font-size: 20px;
}

h4.author{
  color: #708090;
  font-family: Calibri;
}

h4.date{
  color: #708090;
  font-family: Calibri;
}
```

- **Referencing to a CSS File:**

```
---
title: "Investment Report"
output:
  html_document:
    css: styles.css
    toc: true
    toc_depth: 2
    number_sections: true
    toc_float: true
date: "`r format(Sys.time(), '%d %B %Y')`"

params:
  country: Indonesia
  year_start: 2011-07-01
  year_end: 2012-06-30
  fy: 2012
---
```

  - Ensure that the file is under the same directory if not provide the `file_url`