

# Unit-4 : Documentation & Deployment

## Topic : R markdown

### 1.1 What is R markdown?

**R Markdown** is a file format for making dynamic documents with R. It allows you to combine plain text with R code, which can be executed to produce output such as tables, plots, and reports.

### 1.2 Why use R markdown?

Here are some key features of R Markdown:

#### 1. Integration of Code and Text:

- R Markdown files can contain both narrative text (written in Markdown) and chunks of R code. This allows you to explain your analysis while showing the code that generated your results.

#### 2. Output Formats:

- R Markdown supports multiple output formats, including:
  - **HTML**: Great for online sharing.
  - **PDF**: Ideal for print-ready documents, using LaTeX.
  - **Word**: Suitable for Microsoft Word documents.
  - **Presentation Slides**: Can create slideshows using formats like Beamer or Slidy.

#### 3. Dynamic Documents:

- R Markdown documents are dynamic, meaning they can automatically update outputs (like plots or summaries) if the underlying data or code changes.

#### 4. Knitr Engine:

- R Markdown uses the **knitr** package, which processes the document. It executes the R code, captures the output, and combines it with the Markdown text to create the final document.

## 5. Chunk Options:

- Code chunks can have options to control behavior, such as whether to show the code, cache results, or suppress messages. This helps customize the output based on the audience or purpose of the document.

## 6. User-Friendly:

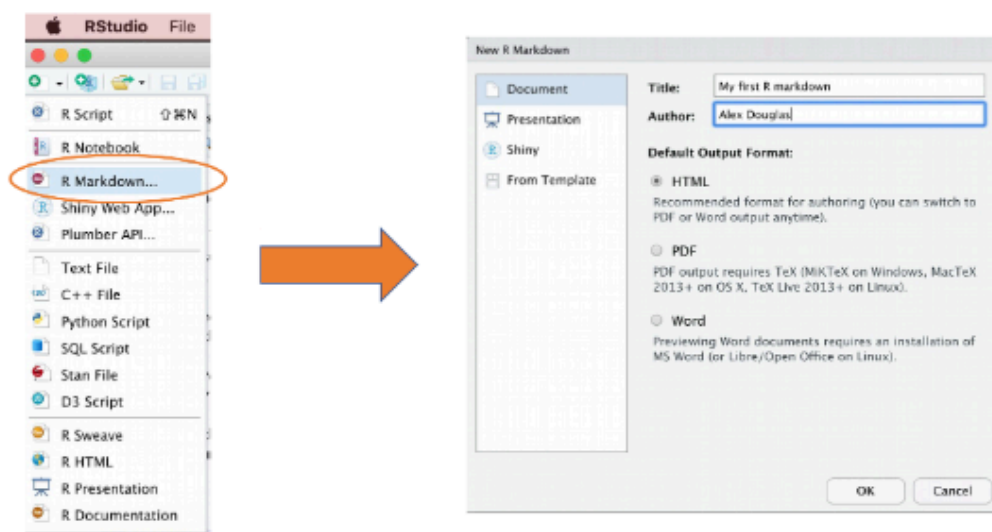
- The syntax is straightforward, making it accessible to users who may not be familiar with programming. Markdown is easy to read and write, which helps in creating well-documented analyses.

## 7. Interactivity:

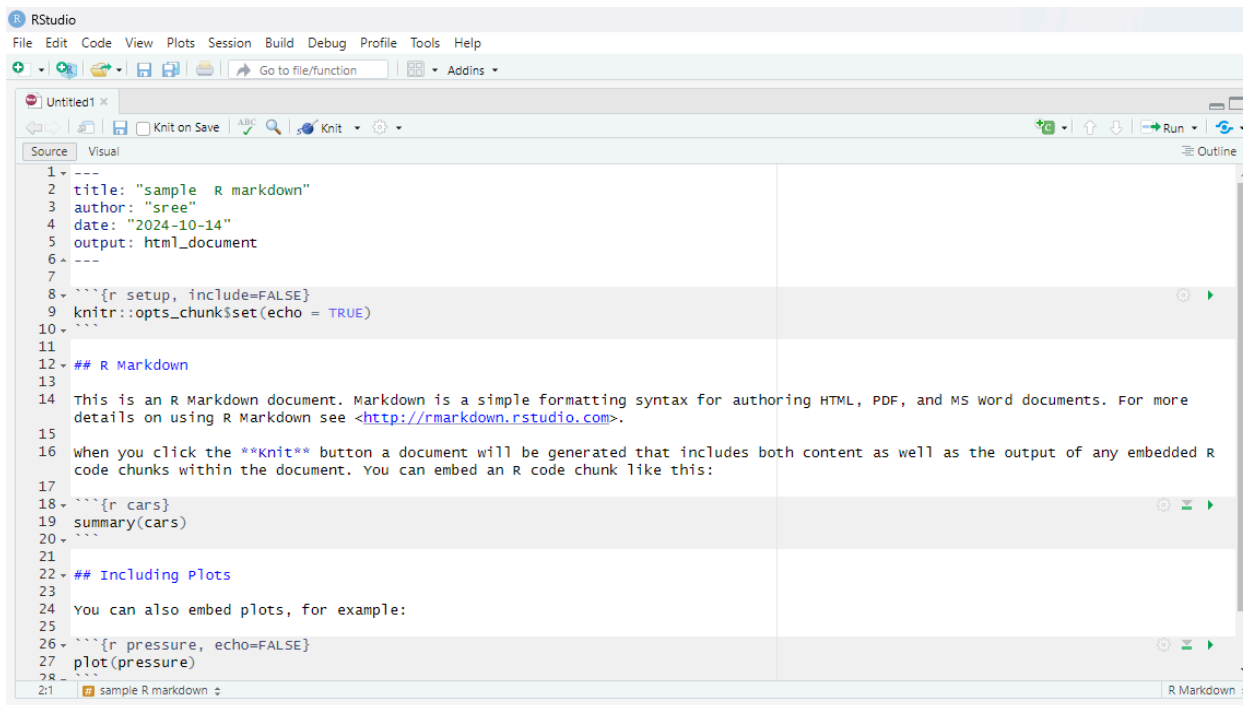
- R Markdown documents can include interactive elements (e.g., plots created with libraries like **plotly** or **shiny**) that enhance the presentation of data.

## 1.3 Create an R markdown document

- Within RStudio, click on the menu `File -> New File -> R Markdown....`
- In the pop up window, give the document a 'Title' and enter the 'Author' information (your name) and select HTML as the default output



You will notice that when your new R markdown document is created it includes some example R markdown code. Normally you would just highlight and delete everything in the document except the information at the top between the `---` delimiters

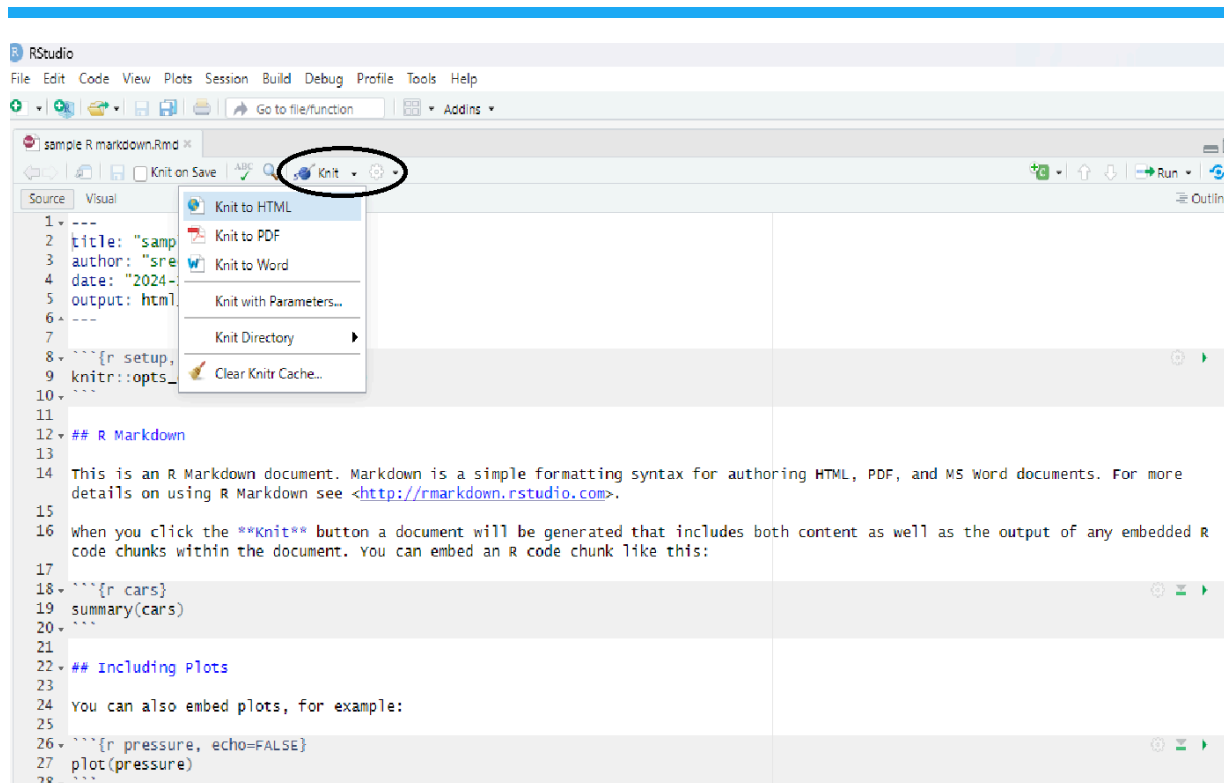


```

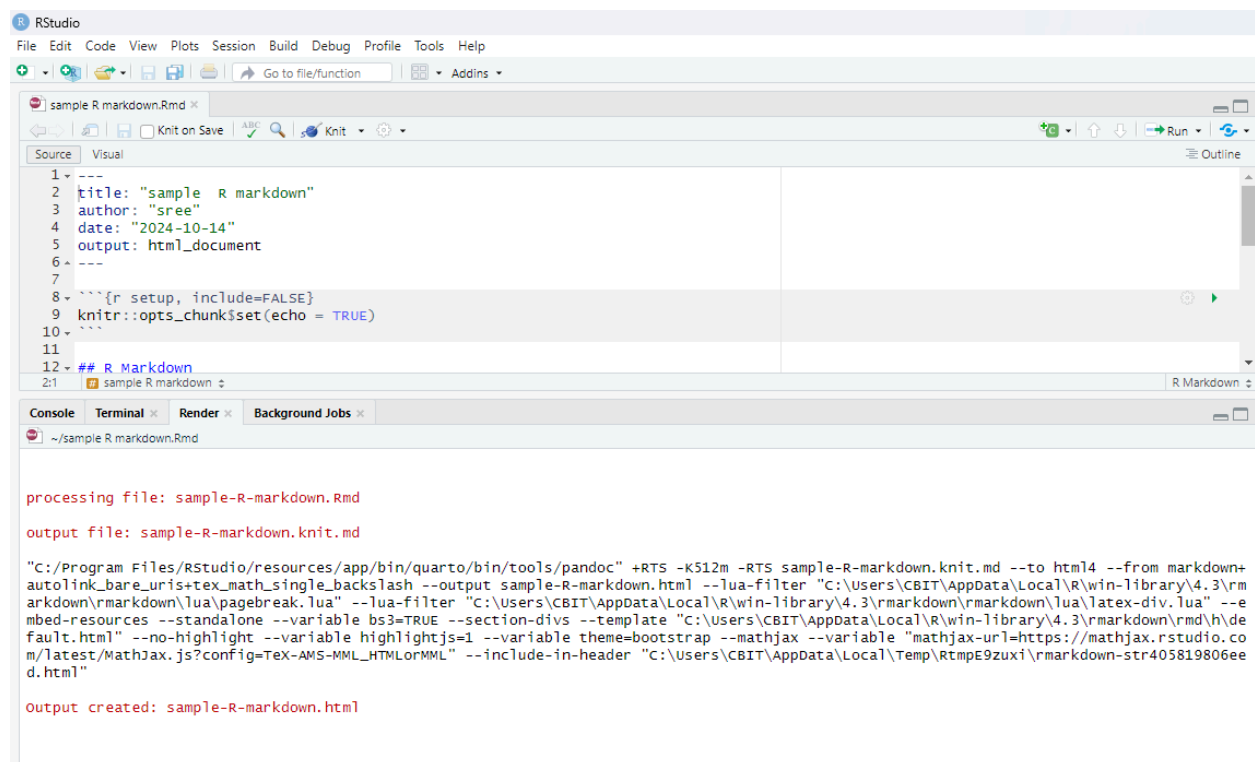
1 ---
2 title: "sample R markdown"
3 author: "sree"
4 date: "2024-10-14"
5 output: html_document
6 ---
7
8 ```{r setup, include=FALSE}
9 knitr::opts_chunk$set(echo = TRUE)
10 ```
11
12 ## R Markdown
13
14 This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS word documents. For more
15 details on using R Markdown see <http://rmarkdown.rstudio.com>.
16
17 when you click the knit button a document will be generated that includes both content as well as the output of any embedded R
18 code chunks within the document. You can embed an R code chunk like this:
19
20 ```{r cars}
21 summary(cars)
22 ```
23
24 ## Including Plots
25
26 You can also embed plots, for example:
27
28 ```{r pressure, echo=FALSE}
29 plot(pressure)
30 ```
31
32 sample R markdown
  
```

Save the file in a convenient location. The file will have the **.Rmd** extension, indicating that it is an R Markdown file.

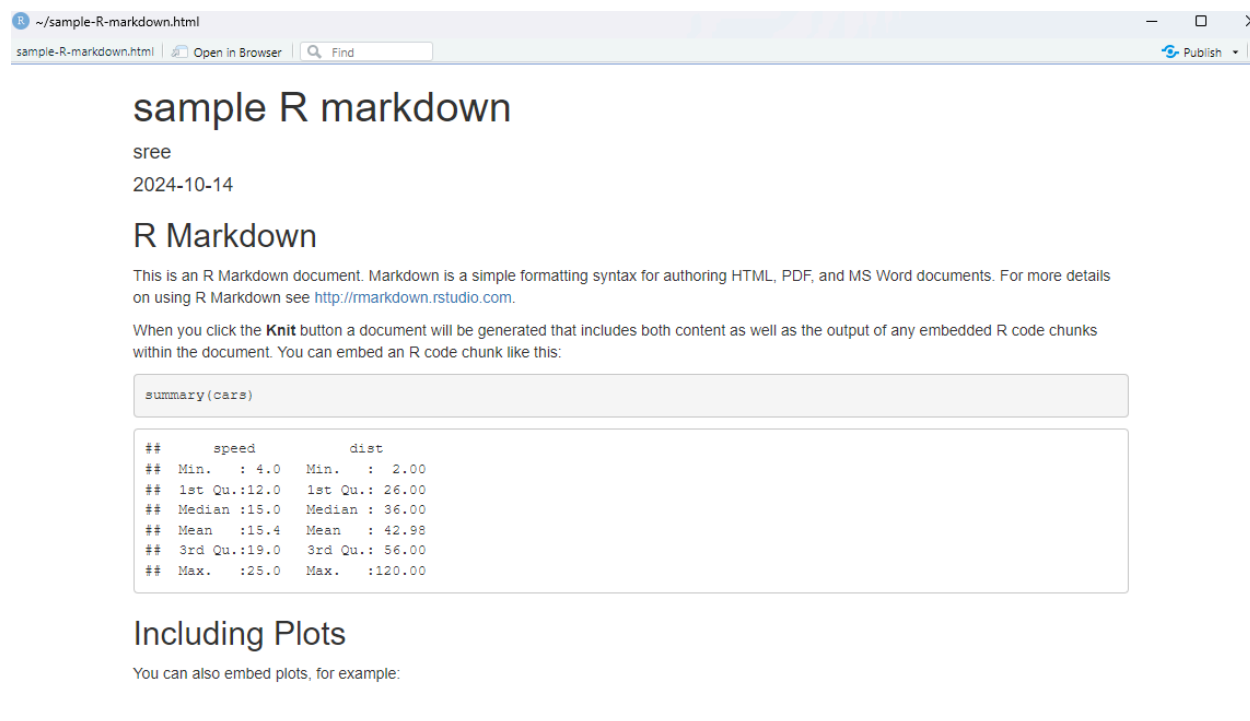
Now, to convert your **.Rmd** file to a HTML document click on the little black triangle next to the **Knit** icon at the top of the source window and select **knit to HTML**.



RStudio will now 'knit' (or render) your `.Rmd` file into a HTML file. Notice that there is a new `render` tab in your console window which provides you with information on the rendering process and will also display any errors if something goes wrong.



If everything went smoothly a new HTML file will have been created and saved in the same directory as your `.Rmd` file. To view this document simply double click on the file to open in a browser (like Chrome or Firefox) to display the rendered content.



```
library(rmarkdown)
```

```
render('my_first_rmarkdown.Rmd', output_format = 'html_document')
```

```
# alternatively if you don't want to load the rmarkdown package
```

```
rmarkdown::render('my_first_rmarkdown.Rmd', output_format = 'html_document')
```

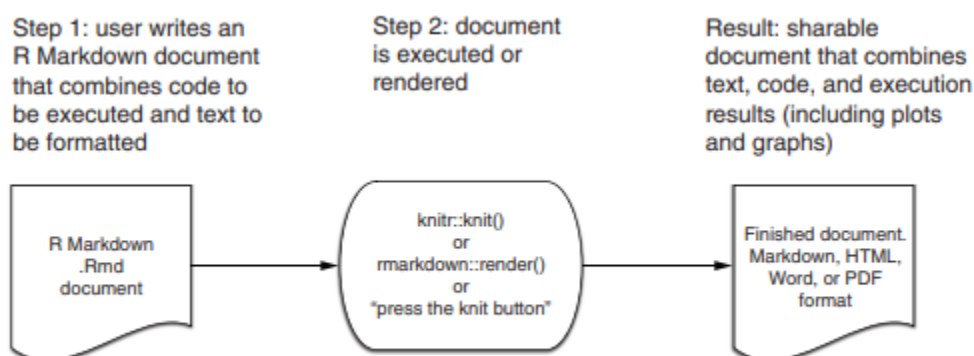
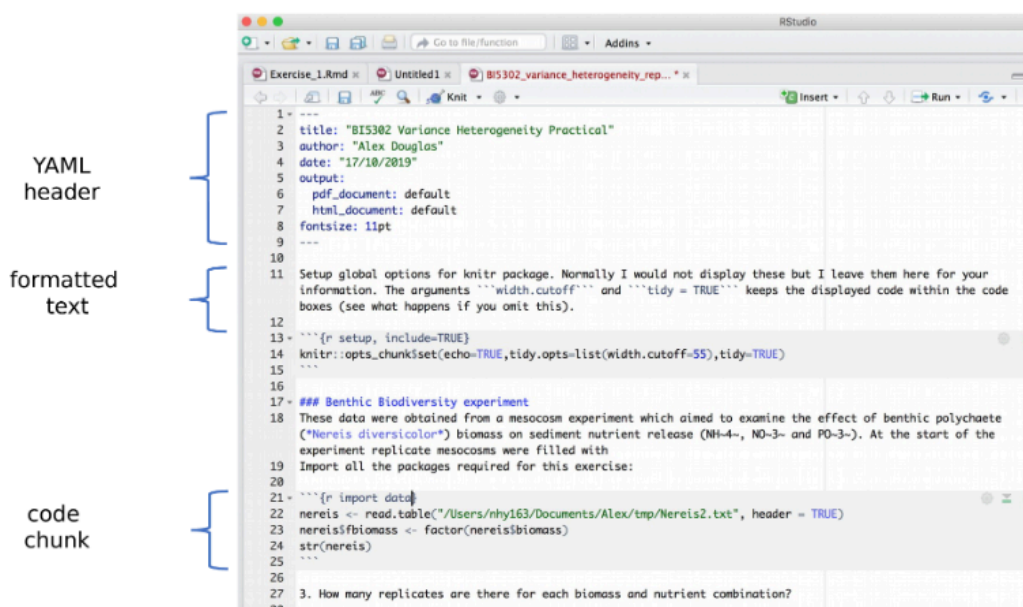


Figure 11.3 The R markdown process

## 1.4 Different components of a typical R markdown document.

Normally each R markdown document is composed of 3 main components: 1) a YAML header, 2) formatted text and 3) one or more code chunks.



### YAML header

The YAML header is surrounded before and after by a `---` on its own line. In RStudio a minimal YAML header is automatically created for you when you create a new R markdown document

```
---
title: My first R markdown document
author: Jane Doe
date: March 01, 2020
output: html_document
---
```

## Formatted text

Goal	R markdown	output
bold text	<code>**mytext**</code>	<b>mytext</b>
italic text	<code>*mytext*</code>	<i>mytext</i>
strikethrough	<code>~~mytext~~</code>	<del>mytext</del>
superscript	<code>mytext^2^</code>	mytext <sup>2</sup>
subscript	<code>mytext~2~</code>	mytext <sub>2</sub>

## Headings

```
# Benthic Biodiversity experiment
## Benthic Biodiversity experiment
### Benthic Biodiversity experiment
#### Benthic Biodiversity experiment
##### Benthic Biodiversity experiment
##### Benthic Biodiversity experiment
```

Benthic Biodiversity experiment

Benthic Biodiversity experiment #

Benthic Biodiversity experiment

Benthic Biodiversity experiment

Benthic Biodiversity experiment

Benthic Biodiversity experiment

## Lists

```
- item 1
- item 2
  + sub-item 2
  + sub-item 3
- item 3
- item 4
```

```
1. item 1
2. item 2
  + sub-item 2
  + sub-item 3
3. item 3
4. item 4
```

```
• item 1
• item 2
  ◦ sub-item 2
  ◦ sub-item 3
• item 3
• item 4
```

```
1. item 1
2. item 2
  ◦ sub-item 2
  ◦ sub-item 3
3. item 3
4. item 4
```

## Images

![Cute grey kitten](images/Cute\_grey\_kitten.jpg)

## Links

You can include a text for your clickable [link](<https://www.worldwildlife.org>)

## Code chunks

All code chunks start and end with three backticks `````

```
```{r}
Any valid R code goes here
```
```

You can insert a code chunk by either typing the chunk delimiters ````{r}` and ````` manually or use the RStudio toolbar (the Insert button) or by clicking on the menu

Code -> Insert Chunk

You can also specify an optional code chunk name (or label) which can be useful when trying to debug problems and when performing advanced document rendering.



```
``{r, summary-stats}
```

| Chunk option | default value    | Function   |
|--------------|------------------|--|
| echo         | echo=TRUE        | If FALSE, will not display the code in the final document  |
| results      | results='markup' | If 'hide', will not display the code's results in the final document. If 'hold', will delay displaying all output pieces until the end of the chunk. If 'asis', will pass through results without reformatting them. |
| include      | include=TRUE     | If FALSE, will run the chunk but not include the chunk in the final document.  |
| eval         | eval=TRUE        | If FALSE, will not run the code in the code chunk.   |
| message      | message=TRUE     | If FALSE, will not display any messages generated by the code.   |
| warning      | warning=TRUE     | If FALSE, will not display any warning messages generated by the code.   |

## Adding figures

```
``{r, simple-plot, fig.width=4, fig.height=3, fig.align='center'}
x <- 1:10    # create an x variable
y <- 10:1    # create a y variable
dataf <- data.frame(x = x, y = y)
plot(dataf$x, dataf$y, xlab = "x axis", ylab = "y axis")
``
```

```
``{r, echo=FALSE, fig.align='center', out.width='50%'}
library(knitr)
include_graphics("images/Cute_grey_kitten.jpg")
``
```

## R markdown example for Linear regression:

```

1 ---
2 title: "LR Model"
3 author: "sree"
4 date: "2024-10-14"
5 output: html_document
6 ---
7 ```{r setup, include=FALSE}
8 fig.dim <- 5
9 library(knitr)
10 library(colorspace) # for adjustcolor
11 opts_chunk$set(
12   fig.height=fig.dim,
13   fig.width=fig.dim,
14   fig.align='center'
15 )
16 ```
17 simulated data:
18 ```{r sim_first}
19 n <- 10
20 slope <- 0.2
21 noise.sd <- 0.25
22 xy <- data.frame( x=rnorm(n))
23 xy$y <- xy$x * slope + noise.sd*rnorm(n)
24 summary(xy)
25 ```
26 Now, fit a linear model:
27
28 ```{r fit_model}
29 xy.lm <- lm( y ~ x, data=xy )
30 summary(xy.lm)
31 ```
32 Look at the results:
33
34 ```{r first_plot}
35 plot( y ~ x, data=xy )
36 abline(coef(xy.lm),col='red')
37 ```
38
39
40

```

## LR Model

sree

2024-10-14

Simulated data:

```
n <- 10
slope <- 0.2
noise.sd <- 0.25
xy <- data.frame( x=rnorm(n))
xy$y <- xy$x * slope + noise.sd*rnorm(n)
summary(xy)
```

```
##           x           y
## Min.      :-1.0702  Min.      :-0.58637
## 1st Qu.: -0.6230  1st Qu.: -0.32460
## Median :  0.2559  Median :  0.06175
## Mean     :  0.0269  Mean      :-0.02655
## 3rd Qu.:  0.3559  3rd Qu.:  0.26498
## Max.     :  1.1071  Max.       :  0.38623
```

Now, fit a linear model:

```
xy.lm <- lm( y ~ x, data=xy )
summary(xy.lm)
```

```
##
## Call:
## lm(formula = y ~ x, data = xy)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.16139 -0.13665 -0.05112  0.05609  0.48624
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.03676    0.06629   -0.555  0.59435
## x           0.37954    0.09012    4.212  0.00295 **
## ---
```

### 1.5 What is predicting buzz

- "Predicting buzz" refers to forecasting or estimating the level of attention, interest, or excitement a particular event, product, or piece of content will generate.
- In marketing, social media, and product launches, "buzz" often refers to the viral spread of information and the overall hype or public engagement surrounding something.
- Predicting buzz is valuable for businesses, influencers, and media companies, as it can help guide decision-making, resource allocation, and marketing strategies.

---

**Common methods to predict buzz include:**

1. **Sentiment analysis:** Analyzing the tone and emotion behind social media posts, reviews, or comments to gauge excitement or interest.
2. **Trend analysis:** Monitoring trending topics, hashtags, or keywords to identify what people are currently talking about and how fast it's spreading.
3. **Historical data:** Using past data (e.g., previous product launches, event announcements) to model future interest or engagement.
4. **Social media engagement:** Tracking likes, shares, comments, and followers to anticipate potential growth or virality.

## Unit-4

# Version Control

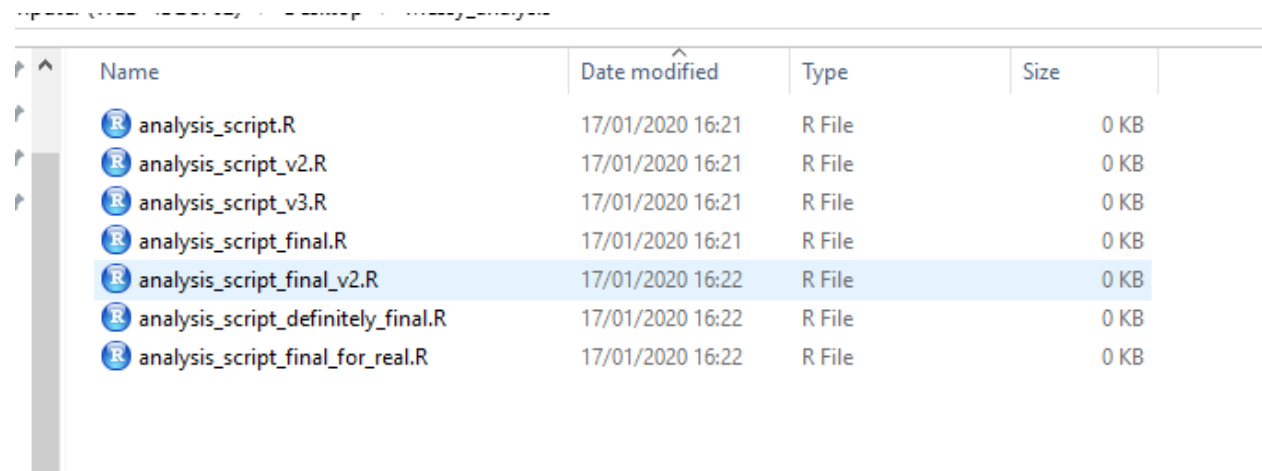
## What is a Version control

A [Version Control System](#) (VCS) keeps a record of all the changes you make to your files that make up a particular project and allows you to revert to previous versions of files if you need to.

Git-Version control system, it's free and open source and it integrates nicely with RStudio.

## Why use version control?

helps avoid this (familiar?) situation when you're working on a project



| Name                               | Date modified    | Type   | Size |
|------------------------------------|------------------|--------|------|
| analysis_script.R                  | 17/01/2020 16:21 | R File | 0 KB |
| analysis_script_v2.R               | 17/01/2020 16:21 | R File | 0 KB |
| analysis_script_v3.R               | 17/01/2020 16:21 | R File | 0 KB |
| analysis_script_final.R            | 17/01/2020 16:21 | R File | 0 KB |
| analysis_script_final_v2.R         | 17/01/2020 16:22 | R File | 0 KB |
| analysis_script_definitely_final.R | 17/01/2020 16:22 | R File | 0 KB |
| analysis_script_final_for_real.R   | 17/01/2020 16:22 | R File | 0 KB |



Benefits of version control, particularly with Git and GitHub

1. **Automatic Version Tracking:** Keeps a comprehensive record of all file versions automatically.
2. **Easy Reversion:** Allows users to easily revert to previous versions of files when needed.
3. **Centralized Management:** Organizes all files in a single location for easy access.
4. **Collaboration Facilitation:** Enables collaborators to review, contribute to, and reuse work effectively.
5. **Accessibility:** Files can be accessed from anywhere with an internet connection on any device.

## What is Git and GitHub?

### Git :

- Developed by Linus Torvalds as a version control system.
- Tracks changes to various file types (e.g., .pdf, .Rmd, .docx, .txt, .jpg), with plain text files being optimal.
- **Sufficient for tracking files and versions locally on your computer.**

### Repository Definition:

- A collection of all files in a project is called a repository (or repo).

### GitHub :

- A web-based hosting service for Git repositories.

- Facilitates remote copies for backup and collaborative work.

The process for starting a project with Git and GitHub:

- 1. Create a Remote Repository:**

- Begin by creating a remote repository on GitHub (optional but common).

- 2. Clone the Repository:**

- Clone the remote repository to your local computer (think of this as copying).
- Note: This cloning is usually a one-time event.

- 3. Work Locally:**

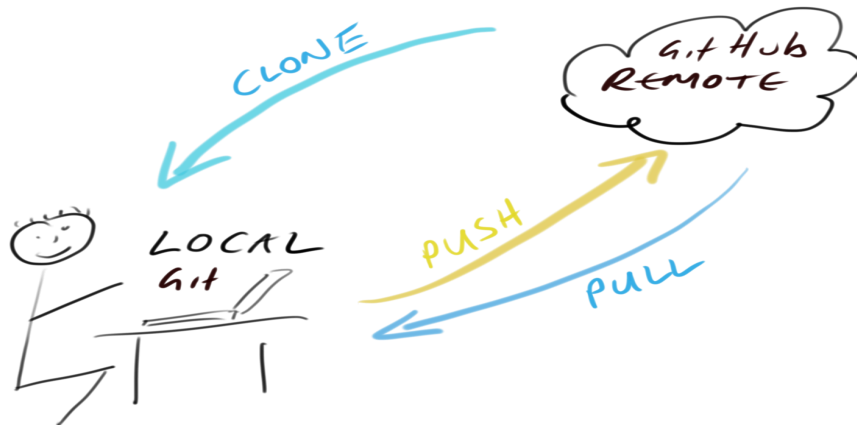
- Work on your project locally by creating and saving files (scripts, R Markdown documents, figures, etc.).

- 4. Take Snapshots of Changes:**

- After making important changes, take snapshots of your files by creating commits.

- 5. Push Changes to GitHub:**

- Push your commits to the remote GitHub repository to create a backup or share with collaborators.



## Getting started

### Install Git

To install Git on a Windows computer we recommend you download and install Git for Windows (also known as 'Git Bash'). You can find the download file and installation instructions - <https://git-scm.com/downloads>


### Configure Git

After installing Git, you need to configure it so you can use it. Click on the Terminal tab in the Console window again and type the following:

```
git config --global user.email 'you@youremail.com'
```

```
git config --global user.name 'Your Name'
```





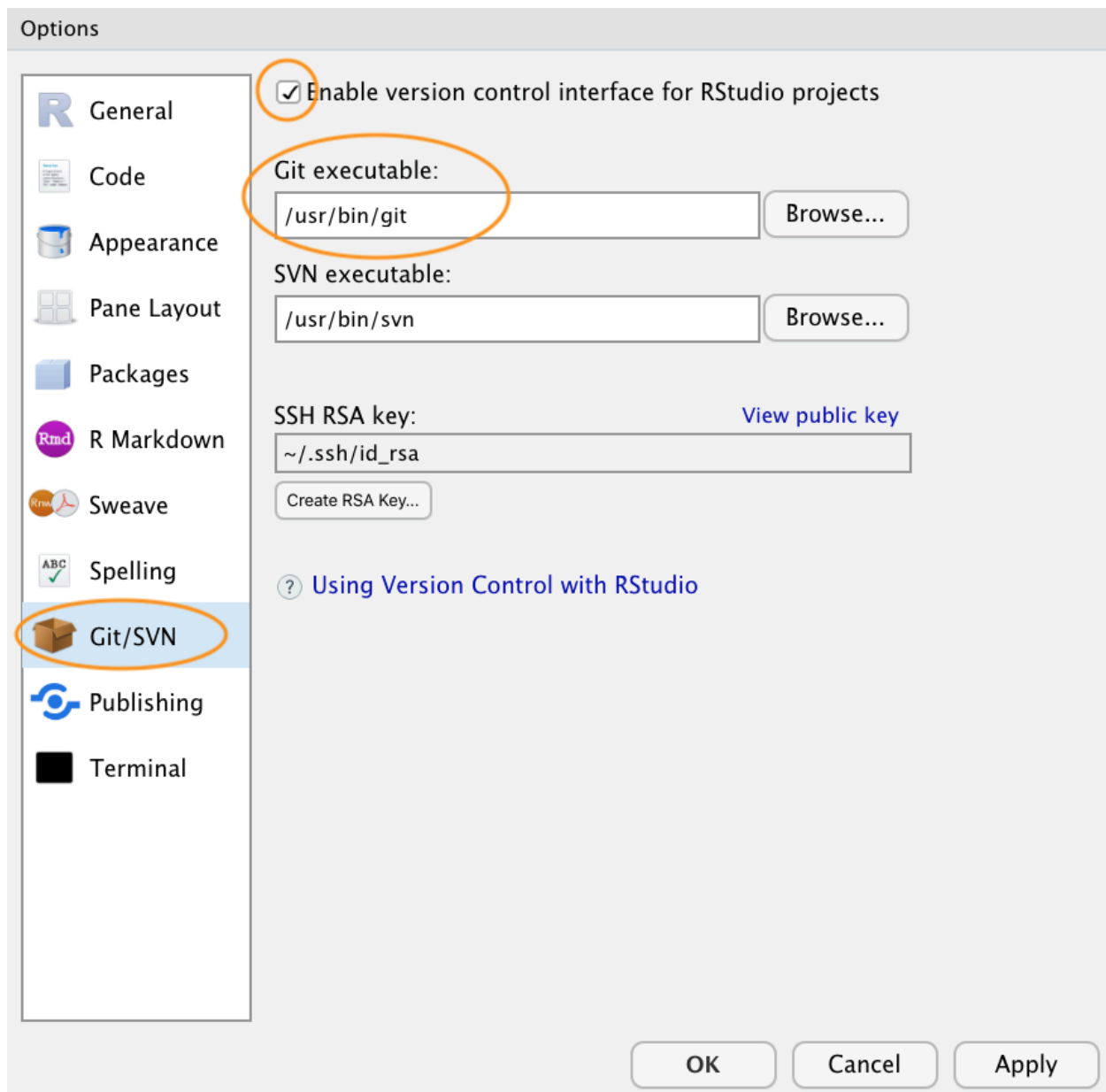
To verify that you have successfully configured Git type the following into the Terminal

**git config --global --list**

You should see both your `user.name` and `user.email` configured.

## Configure RStudio

In RStudio, go to the menu **Tools -> Global Options -> Git/SVN** and make sure that 'Enable version control interface for RStudio projects' is ticked and that the 'Git executable:' path is correct for your installation.



## Register a GitHub account

### 1. Creating a GitHub Account:

- Sign up for a free account on GitHub <https://github.com/>
- Provide a username, email address, and strong password.

- Use your University email address for potential benefits (e.g., educator or researcher account).

## **2. Choosing a Username:**

- Keep it short and lowercase.
- Use hyphens to separate words if needed.
- Incorporate your actual name for professional visibility.

## **3. Selecting a Plan:**

- Click on 'Select a plan' and choose the 'Free Plan' option.
- Complete any CAPTCHA or verification steps.

## **4. Email Verification:**

- Check your email to verify your address.

# Setting up a project in RStudio

**Let's create your first version controlled RStudio project.**

## **Option 1: Setup a Remote GitHub Repository First**

1. Create a Remote Repository:
  - Set up a new repository on GitHub.
2. **Connect RStudio Project:**
  - Link your RStudio project to the newly created remote repository.

## **Option 2: Setup a Local Repository First**

1. Create a Local Repository:
  - Initialize a local Git repository on your machine.



## 2. Link to Remote GitHub Repository:

- Connect the local repository to a remote GitHub repository after it has been set up.

## Option 1 - GitHub first

### **Sign In:**

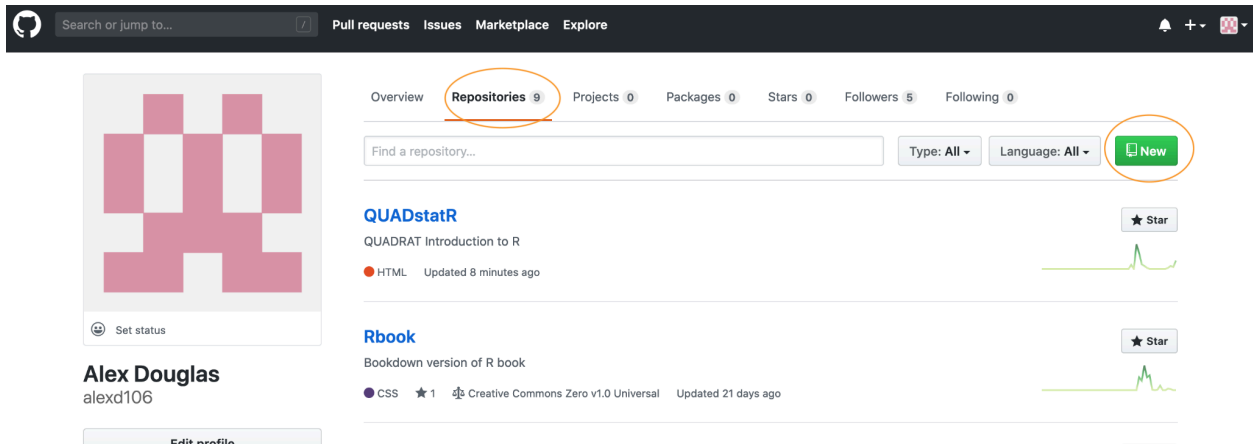
- Go to your GitHub page and sign in to your account if you haven't already.

### **Navigate to Repositories:**

- Click on the 'Repositories' tab at the top of the page.

### **Create New Repository:**

- Click on the green 'New' button on the right side of the Repositories page.



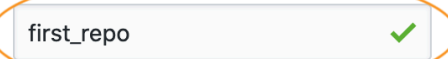
## Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository](#).

Owner



Repository name \*



Great repository names are short and memorable. Need inspiration? How about **fuzzy-fiesta**?

Description (optional)

☒ **Public**

Anyone can see this repository. You choose who can commit.

☐ **Private**

You choose who can see and commit to this repository.

Skip this step if you're importing an existing repository.

☒ **Initialize this repository with a README**

This will let you immediately clone the repository to your computer.

Add .gitignore: **None** ▼

Add a license: **None** ▼



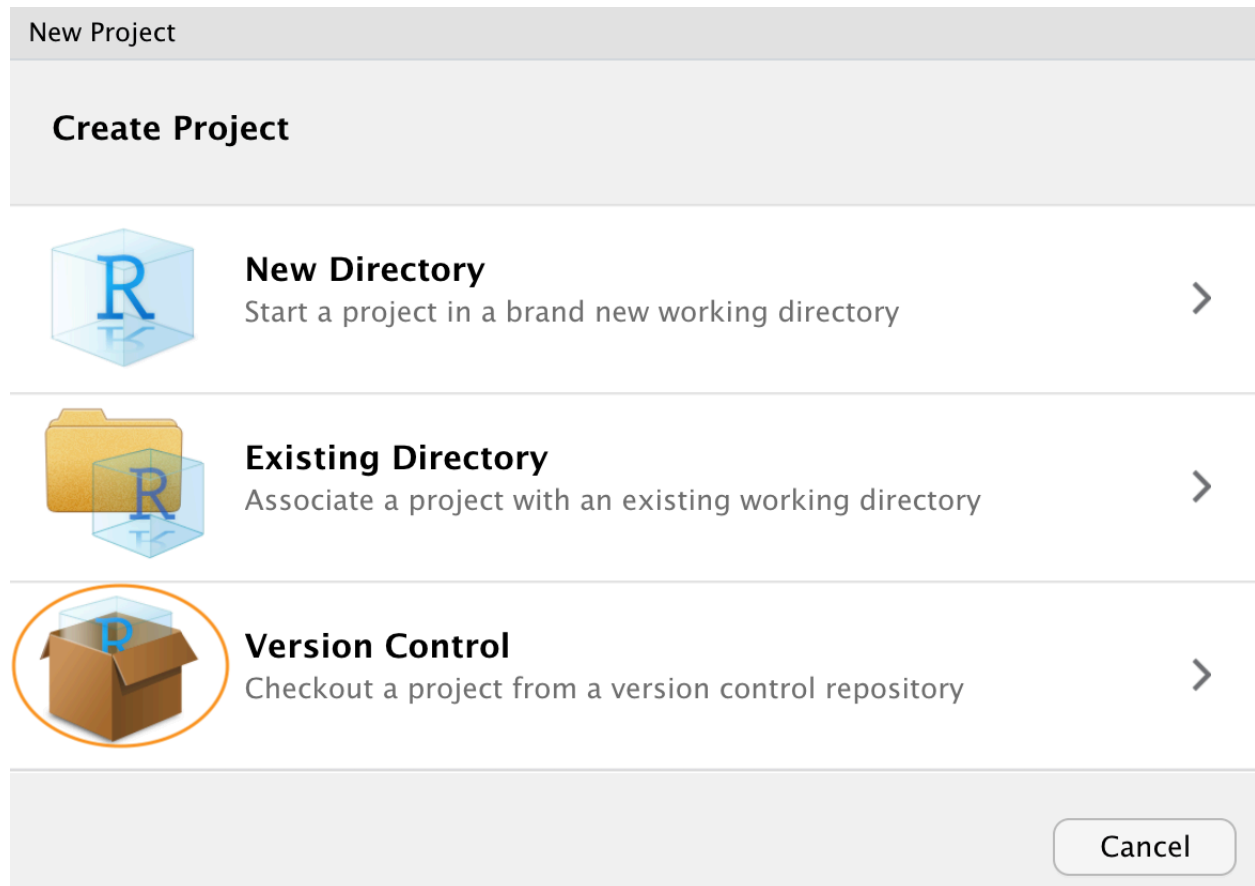
**Create repository**

Next click on the green 'Clone or Download or code' button and copy the `https://...` URL that pops up for later (either highlight it all and copy or click on the copy to clipboard icon to the right).

The first screenshot shows a GitHub repository named 'first\_repo' by user 'Pulipatisr'. The repository is public and has one branch, 'main'. A dropdown menu is open from the 'Code' button, showing options to clone the repository using HTTPS, SSH, or GitHub CLI. The HTTPS URL is highlighted: `https://github.com/Pulipatisr/first_repo.git`. Below the URL, there are options to 'Open with GitHub Desktop' and 'Download ZIP'.

The second screenshot shows a GitHub repository named 'first\_repo' by user 'alex106'. The repository has one commit, one branch, and one contributor. A dropdown menu is open from the 'Clone or download' button, showing options to clone the repository using HTTPS or SSH. The HTTPS URL is highlighted: `https://github.com/alex106/first_repo.git`. Below the URL, there are options to 'Open in Desktop' and 'Download ZIP'.


In RStudio click on the **File -> New Project** menu. In the pop up window select **Version Control**.



New Project

Back

Clone Git Repository



Repository URL:

https://github.com/alex106/first\_repo.git

Project directory name:

first\_repo

Create project as subdirectory of:

~/Documents/Alex/Teaching

Browse...

☒ Open in new session

Create Project

Cancel

### New Directory Creation:

- RStudio creates a new directory on your local computer named after your repository.


### Cloning the Remote Repository:

- The remote repository is cloned into this new directory.

### New Files Generated:

- The directory will contain the following three new files:
  - first\_repo.Rproj (or the name you chose for your repository)



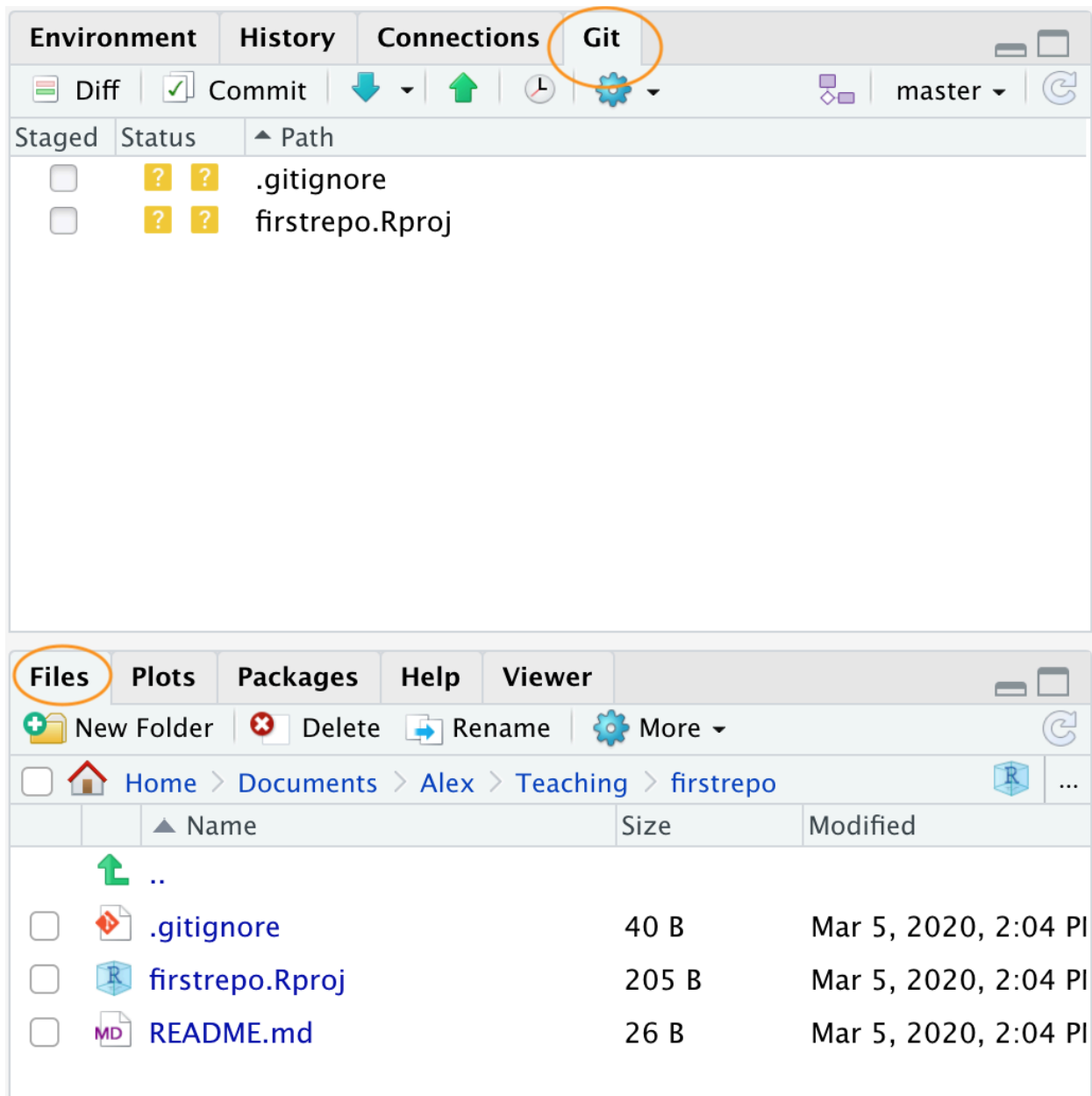
- 
- README.md
  - .gitignore

**Files Tab:**

- You can view these files in the Files tab, usually located in the bottom right pane of RStudio.

**Git Tab:**

- A Git tab will appear in the top right pane, listing the two files (usually the README.md and .gitignore) ready for version control actions.

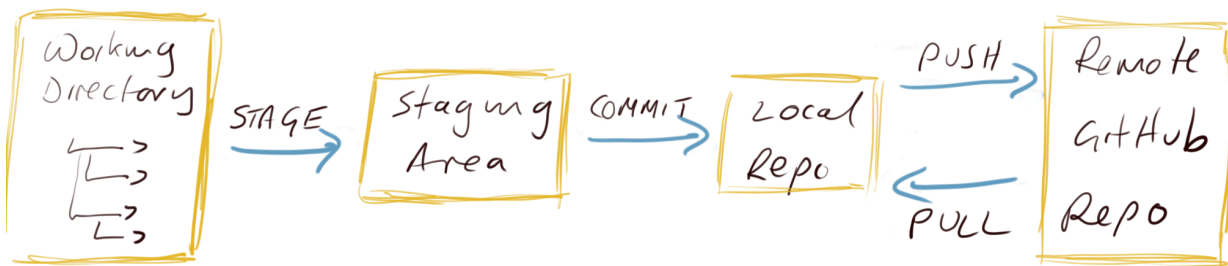


## Using Git

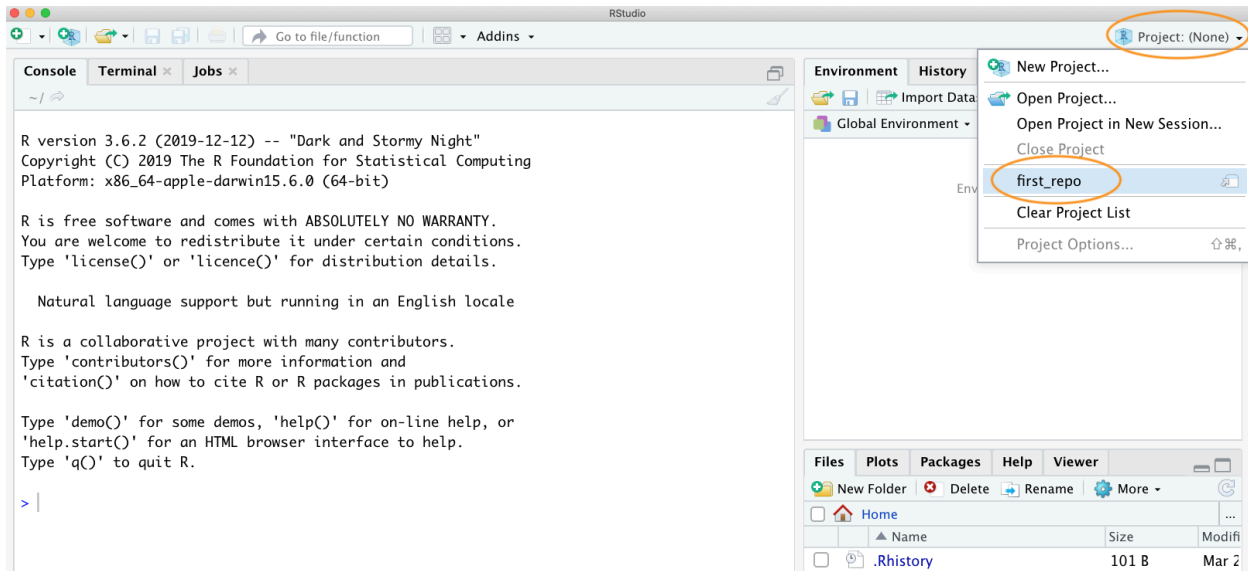
It's finally time to learn how to use Git in RStudio!

Typically, when using Git your workflow will go something like this:

1. You create/delete and edit files in your project directory on your computer as usual (saving these changes as you go)
2. Once you've reached a natural 'break point' in your progress (i.e. you'd be sad if you lost this progress) you stage these files
3. You then commit the changes you made to these staged files (along with a useful commit message) which creates a permanent snapshot of these changes
4. You keep on with this cycle until you get to a point when you would like to push these changes to GitHub
5. If you're working with other people on the same project you may also need to pull their changes to your local computer

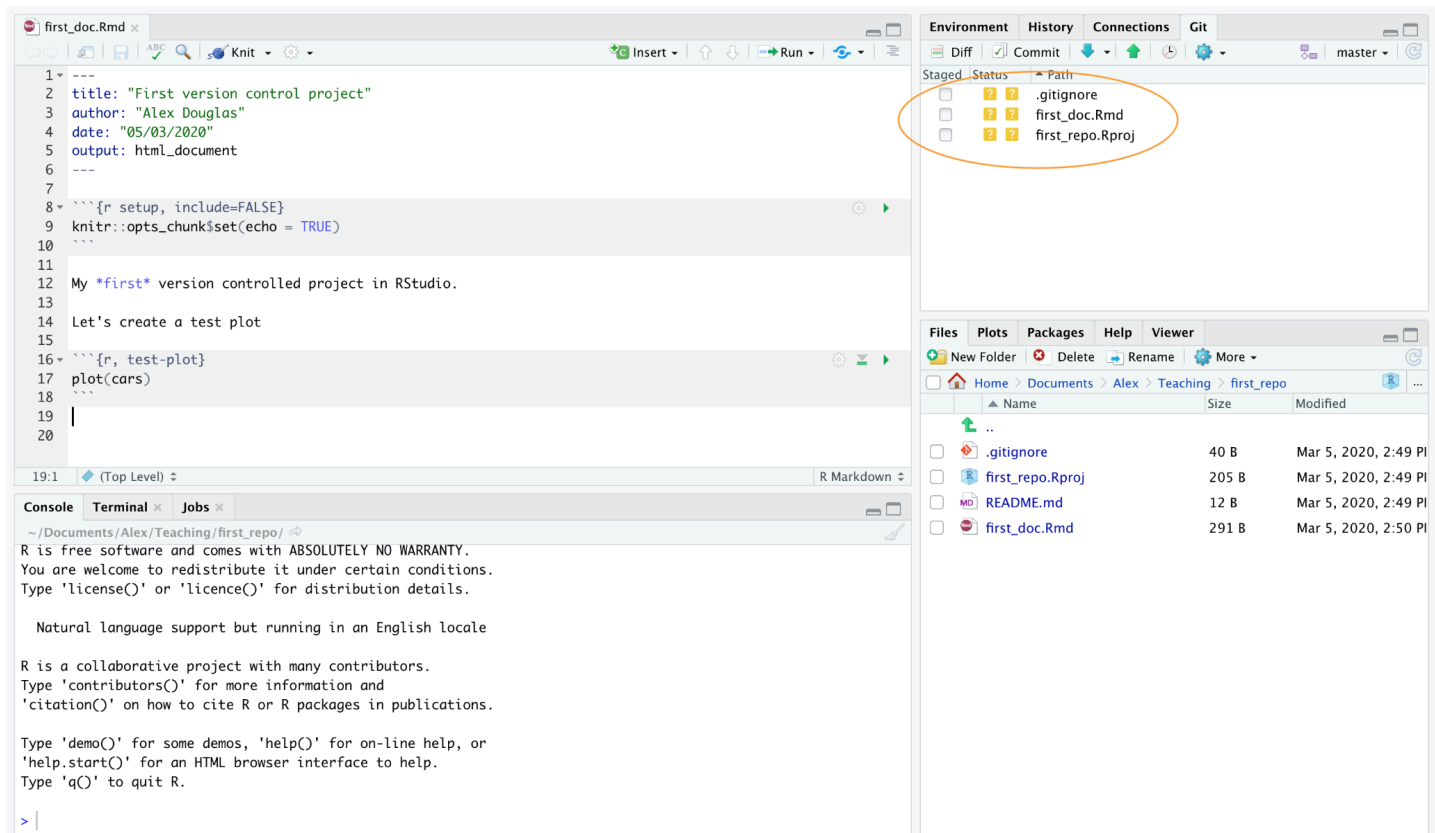


In RStudio open up the `first_repo.Rproj` you created previously during Option 1. Either use the `File -> Open Project` menu or click on the top right project icon and select the appropriate project.

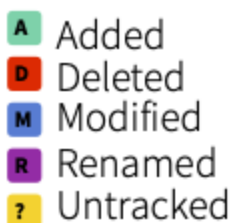


Create an R markdown document inside this project by clicking on the **File -> New File -> R markdown**

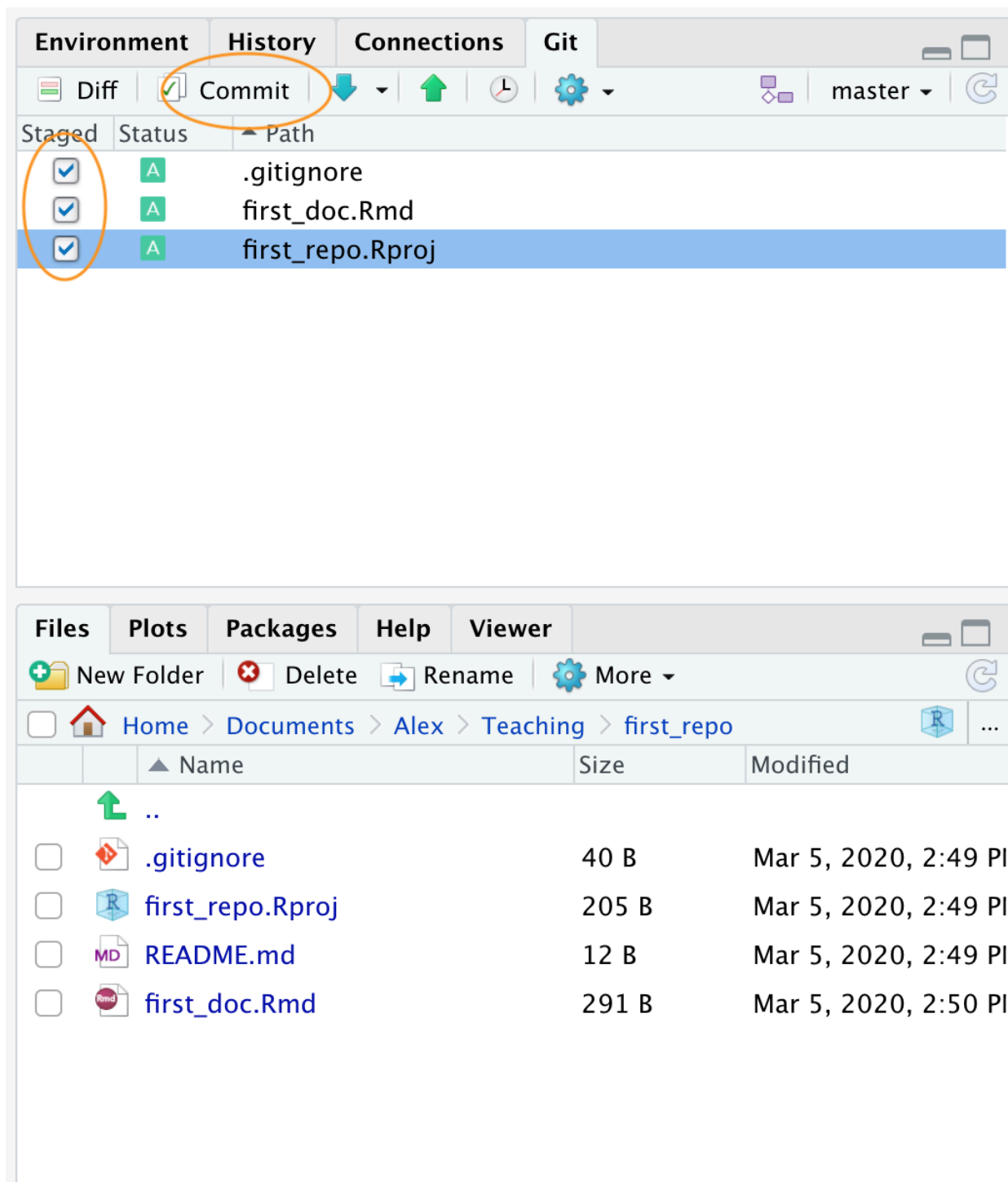
Take a look at the 'Git' tab which should list your new R markdown document (`first_doc.Rmd` in this example) along with `first_repo.Rproj`, and `.gitignore` (you created these files previously when following Option 1).

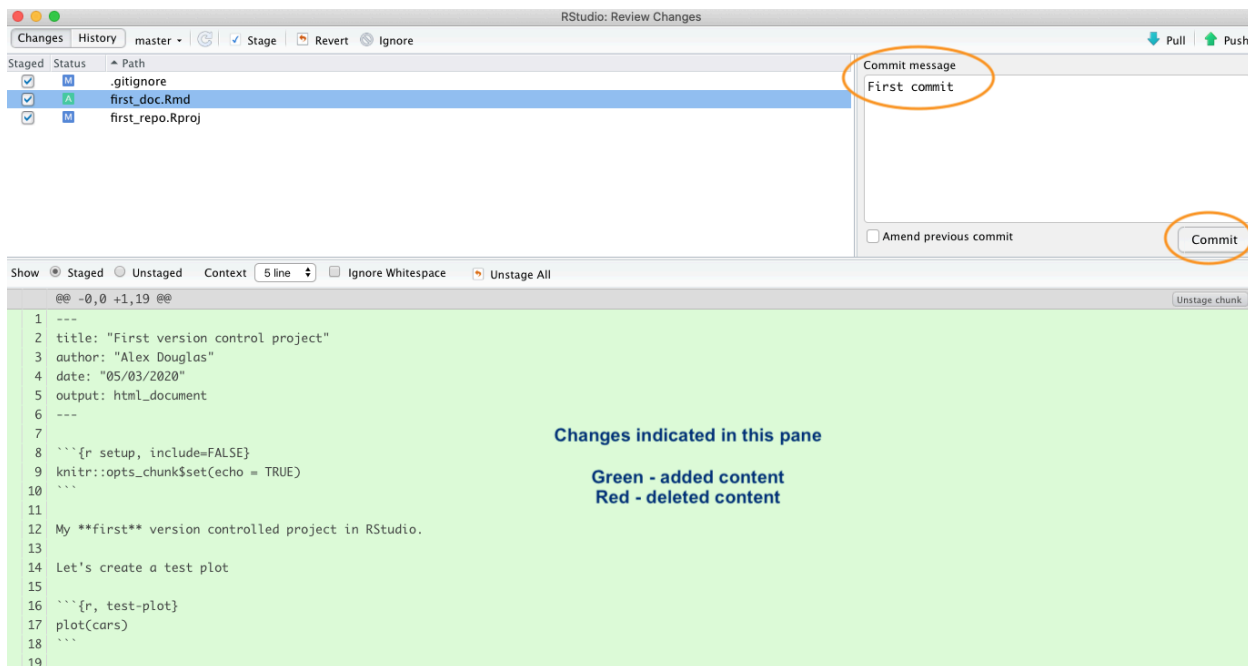


Following our workflow, we now need to stage these files. To do this tick the boxes under the 'Staged' column for all files. Notice that there is a status icon next to the box which gives you an indication of how the files were changed. In our case all of the files are to be added (capital A) as we have just created them.

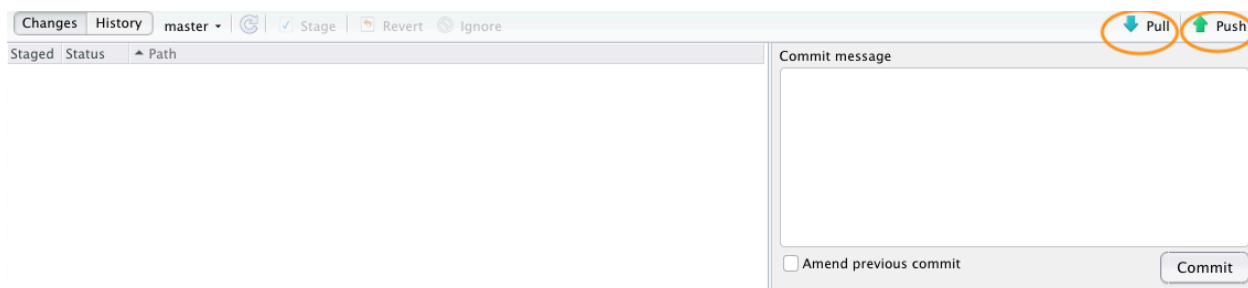


After you have staged the files the next step is to commit the files. This is done by clicking on the 'Commit' button.

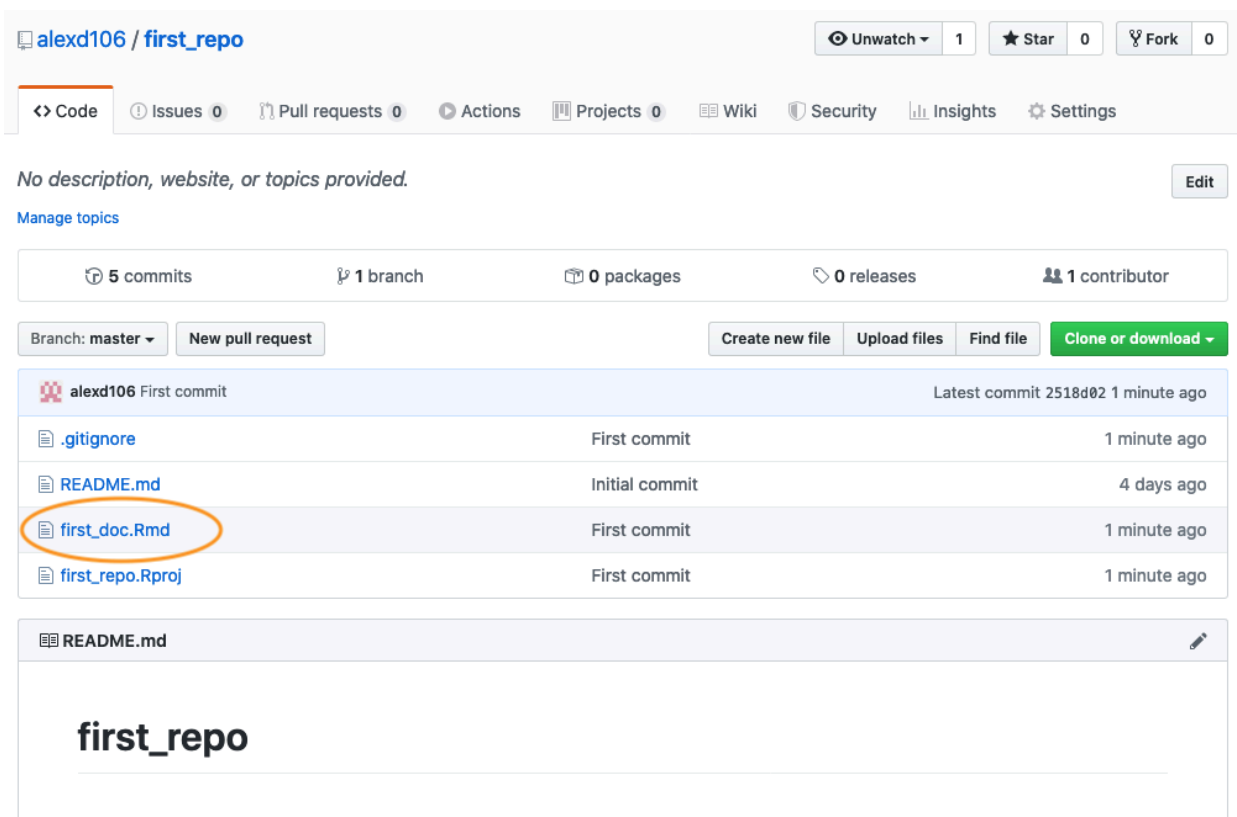




Now that you have committed your changes the next step is to push these changes to GitHub. Before you push your changes it's good practice to first pull any changes from GitHub.



To confirm the changes you made to the project have been pushed to GitHub, open your GitHub page, click on the Repositories link and then click on the `first_repo` repository.



alex106 / first\_repo

Unwatch 1 Star 0 Fork 0

<> Code Issues 0 Pull requests 0 Actions Projects 0 Wiki Security Insights Settings

No description, website, or topics provided. [Edit](#)

[Manage topics](#)

5 commits 1 branch 0 packages 0 releases 1 contributor

Branch: master New pull request Create new file Upload files Find file Clone or download

| alex106 First commit             |                | Latest commit 2518d02 1 minute ago |
|----------------------------------|----------------|------------------------------------|
| <a href="#">.gitignore</a>       | First commit   | 1 minute ago                       |
| <a href="#">README.md</a>        | Initial commit | 4 days ago                         |
| <a href="#">first_doc.Rmd</a>    | First commit   | 1 minute ago                       |
| <a href="#">first_repo.Rproj</a> | First commit   | 1 minute ago                       |

README.md

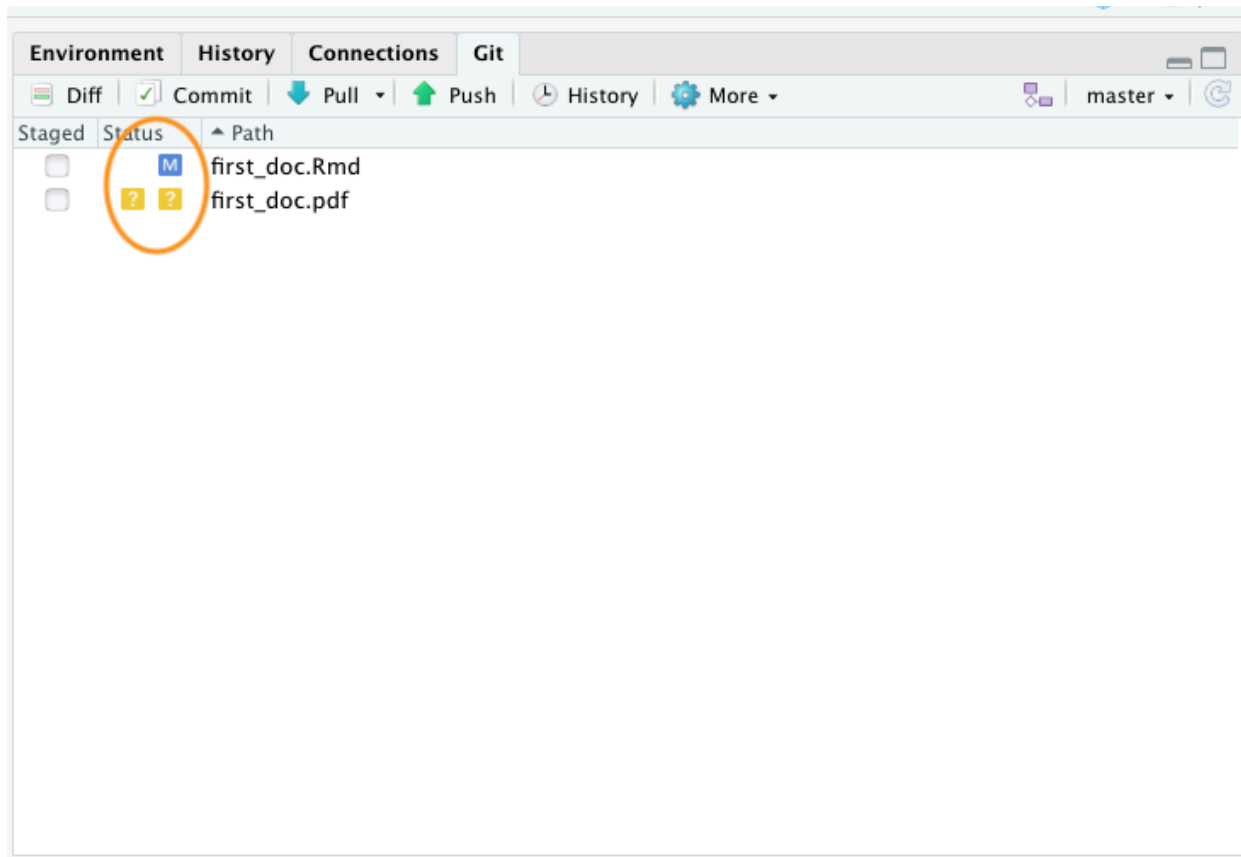
first\_repo

## Tracking changes

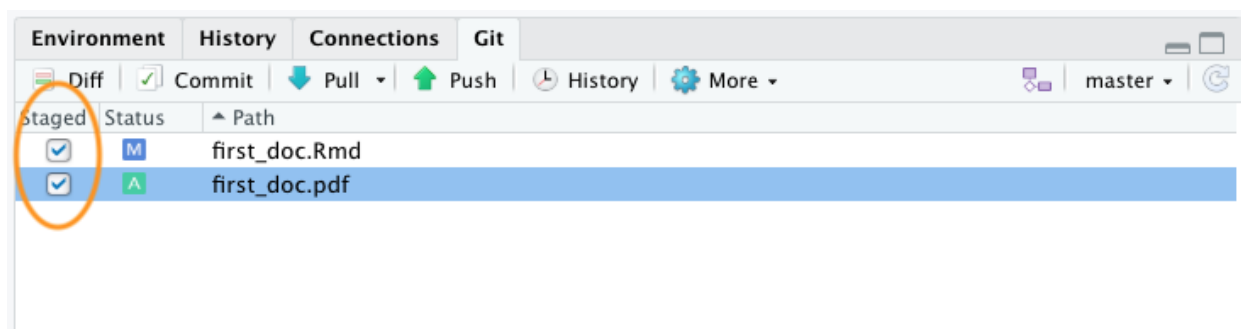
Now let's make some further changes to your R markdown file and follow the workflow once again but this time we'll take a look at how to identify changes made to files, examine the commit history and how to restore to a previous version of the document.



Notice that these two files have been added to the `Git` tab in RStudio. The status icons indicate that the `first_doc.Rmd` file has been modified (capital M) and the `first_doc.pdf` file is currently untracked (question mark).



To stage these files tick the 'Staged' box for each file and click on the 'Commit' button to take you to the 'Review Changes' window.



RStudio: Review Changes

Changes History master Stage Revert Ignore Pull Push

Staged Status Path

first\_doc.Rmd  
first\_doc.pdf

Commit message  
improved plot and added summary table of dataframe

Amend previous commit Commit

Show Staged Unstaged Context 5 line Ignore Whitespace Unstage All

@@ -1,19 +1,28 @@

```
1 1 ---
2 2 title: "First version control project"
3 3 author: "Alex Douglas"
4 4 date: "05/03/2020"
5  output: html_document
5  output: pdf_document
6 6 ---
7 7
8 8 ```{r setup, include=FALSE}
9 9 knitr::opts_chunk$set(echo = TRUE)
10 10 ```
11 11
12 12 My first version controlled project in RStudio.
12 12 This report documents my first attempts of using Git and Github to version control an RStudio project. I will be
13 13 modifying this report, staging and committing changes and then pushing to GitHub.
14 14 Let's create a test plot
14 14 Let's create a test plot of distance (miles) and speed (mph).
15 15
16 16 ```{r, test-plot}
17 17 plot(cars)
17 17 plot(cars, col = "red", xlab = "speed (mph)", ylab = "distance (miles)")
18 18 ```
19 19
20 20 A summary of the data frame is given below
21 21
22 22 ```{r, cars-summary}
23 23 library(knitr)
24 24 kable(summary(cars))
25 25 ```
26 26
27 27
28 28
```

line numbers

deleted lines

added lines

Go to your online GitHub repository and you will see your new commits

## Commit history

To view your commit history in RStudio click on the 'History' button (the one that looks like a clock) in the Git pane to bring up the history view in the 'Review Changes' window. You can also click on the 'Commit' or 'Diff' buttons which takes you to the same window (you just need to additionally click on the 'History' button in the 'Review Changes' window).



### Top Pane:

- Displays the list of all commits in the repository.
- Commits are ordered with the most recent at the top and oldest at the bottom.
- Each commit is associated with a message summarizing the changes.

### Bottom Pane:

- Shows the details of the selected commit when clicked.
- Provides a summary of:
  - **Date:** When the commit was made.
  - **Author:** Who made the commit.

- **Commit Message (Subject):** A brief message about the commit's purpose.

## Commit Identifiers:

- **SHA (Secure Hash Algorithm):** A unique identifier for each commit.
- **Parent SHA:** Identifies the previous commit in the history, establishing the commit order.

## Functionality:

- **View and Revert:** SHA identifiers allow you to view or revert to previous file versions.
- **File View:** You can click on "View file @ SHA key" (e.g., "View file @ 2b4693d1") to see the content of each file at that specific commit.

The screenshot shows the RStudio 'Review Changes' window. The 'History' tab is selected, showing a list of commits. The first commit is highlighted, showing its details: SHA 2b4693d1, Author alexd106 <alex106@gmail.com>, Date 2020-03-24 14:18, Subject improved plot and added summary table of dataframe, and Parent d27e79f1. A box labeled 'summary information' points to this commit details. Below the commit list, the file 'first\_doc.Rmd' is selected, and its content is displayed. A box labeled 'view the file' points to a link 'View file @ 2b4693d1' in the top right corner of the file view area.

| Subject                   | Author                           | Date       | SHA      |
|---------------------------|----------------------------------|------------|----------|
| HEAD -> refs/heads/master | alex106 <alex106@gmail.com>      | 2020-03-24 | 2b4693d1 |
| First commit              | alex106 <alex106@gmail.com>      | 2020-03-24 | d27e79f1 |
| Initial commit            | Alex Douglas <alex106@gmail.com> | 2020-03-24 | c80b0c75 |

```

SHA 2b4693d1
Author alexd106 <alex106@gmail.com>
Date 2020-03-24 14:18
Subject improved plot and added summary table of dataframe
Parent d27e79f1
first_doc.Rmd
first_doc.Rmd
@@ -2,17 +2,27 @@
2 2 title: "First version control project"
3 3 author: "Alex Douglas"
4 4 date: "05/03/2020"
5 output: html_document
5 output: pdf_document
6 ---
7 
8 {r setup, include=FALSE}
9 knitr::opts_chunk$set(echo = TRUE)
10 
11 
12 My **first** version controlled project in RStudio.
12 This report documents my first attempts at using Git and GitHub to version control an RStudio project. I will be modifying this report, staging and committing changes and then pushing to GitHub.
  
```

You can also view your commit history on GitHub website but this will be limited to only those commits you have already pushed to GitHub. To view the commit history navigate to the repository and click on the 'commits' link (in our case the link will be labelled '3 commits' as we have made 3 commits).

alex106 / first\_repo

Watch 0 Star 0 Fork 0

Code Issues 0 Pull requests 0 Actions Projects 0 Wiki Security Insights Settings

No description, website, or topics provided. [Edit](#)

[Manage topics](#)

3 commits 1 branch 0 packages 0 releases 1 contributor

Branch: master New pull request Create new file Upload files Find file Clone or download

| alex106 improved plot and added summary table of dataframe |  | Latest commit 2b4693d yesterday |
|--|--|---------------------------------|
| <a href="#">.gitignore</a>                                 | First commit                                       | yesterday                       |
| <a href="#">README.md</a>                                  | Initial commit                                     | yesterday                       |
| <a href="#">first_doc.Rmd</a>                              | improved plot and added summary table of dataframe | yesterday                       |
| <a href="#">first_doc.pdf</a>                              | improved plot and added summary table of dataframe | yesterday                       |
| <a href="#">first_repo.Rproj</a>                           | First commit                                       | yesterday                       |

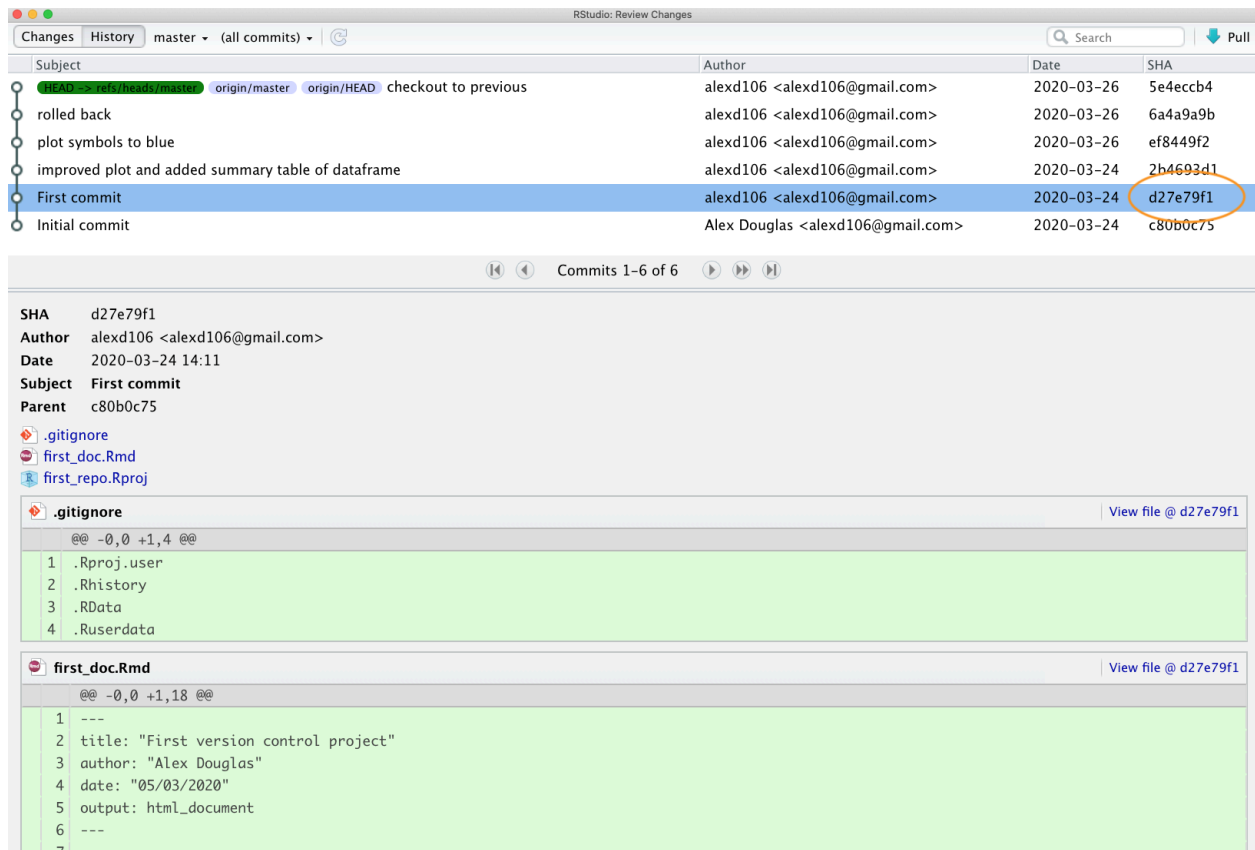
[README.md](#)

# first\_repo

## Reverting changes

Staged, committed and pushed

The `revert` command in Git essentially creates a new commit based on a previous commit and therefore preserves all of your commit history. To rollback to a previous state (commit) you first need to identify the SHA for the commit you wish to go back to (as we did above) and then use the `revert` command in the Terminal



We can use the `revert` command as shown below in the Terminal. The `--no-commit` option is used to prevent us from having to deal with each intermediate commit.

```
git revert --no-commit d27e79f1..HEAD
```

## Deployment of R Models:

1. Web Application
2. Http Service
3. Export

## 1. Web Application using Shiny

**Shiny** is an R package that enables the creation of interactive web applications directly from R. It is particularly useful for data scientists and statisticians, as it allows them to build applications for data analysis, visualization, and model interaction without extensive web development experience.

### Key Features of Shiny

- **Interactivity:** Shiny applications can include input controls (sliders, text boxes, dropdowns) that users interact with to dynamically update outputs.
- **Live Updating:** Outputs like plots and tables update in real time based on user inputs.
- **User Interface (UI) Customization:** Shiny provides functions for creating UI layouts, but it can also integrate HTML, CSS, and JavaScript for enhanced customization.
- **Server-side Processing:** The server function in a Shiny app allows R code to respond to user inputs and update outputs accordingly.

## Creating a Shiny app in RStudio

### 1. Build Your Model

Start by training your model in R. For example, you might train a machine learning model using packages like **caret**, **randomForest**, or **xgboost**.

#### # Example: Train a simple random forest model

```
library(randomForest)
data(iris)
model <- randomForest(Species ~ ., data=iris)
```

### 2. Set Up a New Shiny Project

1. Open RStudio.
2. Go to **File > New Project**.
3. Select **New Directory > Shiny Web Application**.
4. Name your project and choose the directory where you want it to be saved.
5. You'll get a template with **app.R** containing both **UI** and **Server** sections.

A basic Shiny app has two key components: **UI** (user interface) and **Server**.

1. **UI:** Defines the layout and appearance of the app.
2. **Server:** Contains the server-side logic, including model inference.

Here's a template for a Shiny app:

```
library(shiny)

# UI
ui <- fluidPage(
  titlePanel("Model Deployment with Shiny"),

  sidebarLayout(
    sidebarPanel(
      # Input fields for model features
      numericInput("sepal_length", "Sepal Length:", value = 5.0),
```



```

        numericInput("sepal_width", "Sepal Width:", value = 3.0),
        numericInput("petal_length", "Petal Length:", value = 4.0),
        numericInput("petal_width", "Petal Width:", value = 1.0),
        actionButton("predict", "Predict")
    ),

    mainPanel(
        # Output field for model prediction
        textOutput("result")
    )
)
)

# Server logic
server <- function(input, output) {
  observeEvent(input$predict, {
    # Extract inputs
    new_data <- data.frame(
      Sepal.Length = input$sepal_length,
      Sepal.Width = input$sepal_width,
      Petal.Length = input$petal_length,
      Petal.Width = input$petal_width
    )

    # Generate prediction
    prediction <- predict(model, new_data)

    # Output the prediction
    output$result <- renderText({
      paste("Predicted Species:", prediction)
    })
  })
}

# Run the app
shinyApp(ui = ui, server = server)

```

### 3. Run the Shiny App in RStudio

1. In RStudio, click the **Run App** button, usually located in the top-right corner of the script editor.
2. The app will open in a new window or in the Viewer pane.
3. Test the inputs and ensure that the output appears as expected.

To test the app locally, save it as an `app.R` file, open an R session, and run:

```
shiny::runApp("path/to/your/app.R")
```

This will open the Shiny app in your web browser.

## 4. Deploy the Shiny App Online

Deploying a Shiny app online is straightforward, especially with **Shinyapps.io**—an **RStudio-hosted service designed specifically for Shiny apps**.

Here's a step-by-step guide to deploying your app using Shinyapps.io, as well as a few alternative methods.

### Deploying with Shinyapps.io

#### Step 1: Set Up Shinyapps.io

1. Go to [Shinyapps.io](https://shinyapps.io) and create an account if you haven't already.
2. Once you're logged in, you'll receive a **token** and **secret key**. You'll use these credentials to deploy the app directly from RStudio.

#### Step 2: Install and Configure the **rsconnect** Package

In RStudio, install the **rsconnect** package if it's not already installed.

```
install.packages("rsconnect")  
library(rsconnect)
```

#### Step 3: Authenticate with Shinyapps.io

Authenticate your Shinyapps.io account by entering your token and secret key in RStudio. Use the credentials provided in your Shinyapps.io account.

```
rsconnect::setAccountInfo(name = "your_shinyapps_username",  
                           token = "your_token_here",  
                           secret = "your_secret_here")
```

#### Step 4: Deploy the Shiny App

Make sure your working directory is set to the location of your **app.R** file, or specify the full path to the app. Then, deploy the app:

```
rsconnect::deployApp("path/to/your/app")
```

This will upload your app to Shinyapps.io, and RStudio will provide you with a URL where your app is hosted.

#### Step 5: Access Your Deployed App

Once deployment is complete, Shinyapps.io will generate a URL (e.g., [https://your\\_username.shinyapps.io/your\\_app](https://your_username.shinyapps.io/your_app)). You can share this link with others, and they'll be able to access your app online.

## What is an HTTP Service?

An **HTTP service** (also known as a web API or RESTful service) allows applications to communicate over the web using the HTTP protocol. HTTP services expose functionalities or data as endpoints, which other applications can interact with by making requests (e.g., GET, POST) to these endpoints.

## Key Concepts of HTTP Services

- **Endpoints:** Specific URLs where resources or functions are accessible.
- **HTTP Methods:** Methods like GET, POST, PUT, and DELETE specify the type of request being made.
- **JSON:** Common data format used in HTTP services for data exchange.

HTTP services allow flexible integration with other systems, where data can be sent to an endpoint, processed, and then returned as a response.

### Example 1:

1. **Create a New Plumber Script (e.g., `simple_plumber.R`):**

```
# simple_plumber.R

library(plumber)

# Define a simple GET endpoint that returns a message
#* @get /hello
function() {
  list(message = "Hello, Plumber API is working!")
}
```

To run this Plumber script, open an R console and run the following:

```
library(plumber)

r <- plumb("path/to/simple_plumber.R")

r$run(port = 8000) # Runs the API on port 8000
```

## 3. Testing the Endpoint

Once the Plumber API is running, you can test it using:

**Using a Browser:** Open your browser and go to <http://127.0.0.1:8000/hello>. You should see the following JSON response:

```
{"message": "Hello, Plumber API is working!"}
```

**Using `httr` in R:** You can test the API using the `httr` package in R like this:

```
library(httr)

# Send a GET request to the /hello endpoint
response <- GET("http://127.0.0.1:8000/hello")

# Print the response content
content(response)
```

```
$message  
[1] "Hello, Plumber API is working!"
```

## Example2: Deploying a Model as an HTTP Service Using Plumber

The **Plumber** package in R makes it easy to turn R functions into API endpoints, allowing you to deploy models as an HTTP service.

### Step-by-Step Guide to Deploying an HTTP Service with Plumber

Install Plumber:

```
install.packages("plumber")
```

Create a Model and Save It (if you don't already have one):

```
library(randomForest)  
  
# Train a simple random forest model on iris data  
iris_model <- randomForest(Species ~ ., data = iris)  
saveRDS(iris_model, "iris_model.rds")
```

Write a Plumber API Script: Create a file called `plumber.R`, to define the HTTP endpoint:

```
# plumber.R  
  
library(plumber)  
  
# Load the saved model  
model <- readRDS("iris_model.rds")  
  
# Define an API endpoint for scoring new data  
## @post /score  
## @param data A JSON object containing data to score  
function(data) {  
  new_data <- as.data.frame(data)  
  prediction <- predict(model, new_data)  
  list(prediction = as.character(prediction))  
}
```

Run the Plumber API Locally: To start the API server locally, open a new R session and use the following command:

```
library(plumber)  
  
# Point to the plumber.R file  
r <- plumb("path/to/plumber.R")  
r$run(port = 8000)
```

Once running, you can access the endpoint at <http://localhost:8000/score>.

**Test the Endpoint:** You can use `curl`, `Postman`, or an R script to test your API.

```
library(httr)
library(jsonlite)

# Define new data for prediction
new_data <- data.frame(Sepal.Length = 5.1, Sepal.Width = 3.5,
  Petal.Length = 1.4, Petal.Width = 0.2)

# Send POST request to the API
response <- POST(
  "http://localhost:8000/score",
  body = toJSON(list(data = new_data)),
  encode = "json"
)

# View the response
content(response)
```

## Export:

Export Using `tidypredict`, you can convert certain R models into SQL queries that can be directly run on a database. `tidypredict` supports several model types, including `lm` (linear models) and `randomForest`, among others.

Here's a simple example using `lm` with `tidypredict` to create an SQL query for a linear regression model. This allows you to deploy a model directly in SQL for predictions on a database.

## Step-by-Step Example with `lm` Model

### Step 1: Install and Load Required Libraries

Make sure you have the required libraries installed:

```
install.packages("tidypredict")
install.packages("dplyr")
```

Load the libraries:

```
library(tidypredict)
library(dplyr)
```

### Step 2: Create a Simple Linear Model in R

For this example, let's create a linear regression model using the `mtcars` dataset. We'll predict the `mpg` (miles per gallon) based on `wt` (weight) and `hp` (horsepower).

```
# Fit a linear model
model <- lm(mpg ~ wt + hp, data = mtcars)
```

### Step 3: Generate SQL Code Using `tidypredict`

`tidypredict` can convert the `lm` model into SQL code. This SQL code can then be exported to run directly in a SQL database.

```
# Generate SQL code for the model
sql_code <- tidypredict_to_sql(model, dbplyr::simulate_mssql())
cat(sql_code)
```

The `tidypredict_to_sql` function generates SQL code compatible with a variety of databases. Here, `dbplyr::simulate_mssql()` is used to simulate SQL syntax for Microsoft SQL Server, but you can substitute with `dbplyr::simulate_postgres()`, `dbplyr::simulate_mysql()`, or others, depending on your database.

The resulting `sql_code` will look something like this:

#### Sql Code:

```
CASE
  WHEN (0.000000 + (-3.877830 * "wt") + (-0.031773 * "hp")) IS NULL
THEN NULL
  ELSE (0.000000 + (-3.877830 * "wt") + (-0.031773 * "hp"))
END
```

### Step 4: Export or Use the SQL Code

You can now use the generated SQL code in your SQL database to make predictions.

For example, if you have a `car_data` table in your SQL database with columns `wt` and `hp`, you can use the SQL code to predict `mpg` directly in the database:

Sql code

```
SELECT
  (0.000000 + (-3.877830 * wt) + (-0.031773 * hp)) AS predicted_mpg
FROM
  car_data;
```

### Step 5: Deploy to SQL Database

You can use this SQL query in your database for predictions without needing to run the model in R. This is particularly useful for deploying models in production environments where predictions need to be made directly in the database.