Assignment 1: Setting Up Workflow

Applied Quantitative Methods for the Social Sciences II

Carlos III–Juan March Institute, Spring 2026

Instructions:

- Deadline: **February 12, before class**
- This assignment walks you through setting up Git and GitHub
- You will use this setup to submit all assignments throughout the course
- Complete all the tasks below and send me the link to your GitHub repository

1 Task 1: Create a GitHub Account

If you don't already have a GitHub account, create one at `https://github.com`.

1. Go to `https://github.com` and click "Sign up"
2. Choose a **professional username**—you may use this for years in your academic career
3. Use your university email or a professional email address
4. Complete the verification process
5. Optionally: Add a profile picture and brief bio

2 Task 2: Create a Repository for This Course

Create a new **public** repository. Name it something like `aqmss2` or `quant-methods-2026`.

2.1 Option A: Using the GitHub Web Interface

1. Click the "+" icon in the top-right corner of GitHub
2. Select "New repository"
3. Fill in the form:
   - **Repository name**: `aqmss2` (or similar)
   - **Description**: "Assignments for AQMSS II, Spring 2026"
   - **Visibility**: Select **Public**
   - Check the box "Add a README file"
4. Click "Create repository"

2.2 Option B: Using the Command Line

First, make sure Git is installed on your computer. Then run:

```
# Create a new directory and initialize Git
mkdir aqmss2
cd aqmss2
git init

# Create a README file
echo "# AQMSS II - Problem Sets" > README.md

# Stage and commit the file
git add README.md
git commit -m "Initial commit"

# Set up the remote repository (create it on GitHub first, without README)
git branch -M main
git remote add origin https://github.com/YOUR-USERNAME/aqmss2.git
git push -u origin main
```

Note: Replace `YOUR-USERNAME` with your actual GitHub username.

2.3 Option C: Using RStudio

1. First, create an empty repository on GitHub (without README)
2. In RStudio: File → New Project → Version Control → Git

3. Paste your repository URL: `https://github.com/YOUR-USERNAME/aqmss2.git`
4. Choose a location on your computer
5. Click "Create Project"

3 Task 3: Edit the README File

Your README is the “front page” of your repository. Edit it to include:

- Your name
- A brief description (e.g., “Assignments for AQMSS II, Spring 2026”)
- Optionally, a list of what will be in the repository

3.1 Option A: On the Web

1. Click on `README.md` in your repository
2. Click the pencil icon (edit) in the top-right of the file view
3. Make your changes using Markdown syntax
4. Scroll down and click “Commit changes”
5. Add a commit message like “Update README with my info”

3.2 Option B: Command Line

```
# Edit README.md with any text editor, then:
git add README.md
git commit -m "Update README with my info"
git push
```

3.3 Option C: RStudio

1. Edit the `README.md` file in the Files pane
2. Go to the Git pane (usually top-right)
3. Check the box next to `README.md` to stage it
4. Click “Commit”
5. Write a commit message and click “Commit”
6. Click “Push” to upload to GitHub

4 Task 4: Create a Folder and R File

Create a folder called `problem_sets` and add your first R file.

4.1 Option A: On the Web

1. Click "Add file" → "Create new file"
2. In the filename box, type: `problem_sets/ps1.R`
   - This creates the folder and file at once
3. Add the following content:
   ```
   # Assignment 1
   # AQMSS II, Spring 2026
   # [Your Name]

   # This file will contain my solutions for Assignment 1
   print("Hello, Git!")
   ```
4. Scroll down and commit with message "Add ps1.R"

4.2 Option B: Command Line

```
# Create the folder
mkdir problem_sets

# Create the R file (use any text editor)
# Then stage, commit, and push:
git add problem_sets/ps1.R
git commit -m "Add ps1.R"
git push
```

4.3 Option C: RStudio

1. Create a new folder `problem_sets` in the Files pane
2. Create a new R script: File → New File → R Script
3. Add the header content and save as `problem_sets/ps1.R`
4. In the Git pane, stage the new file, commit, and push

5 Task 5: Load a Dataset and Make a Plot

In this task, you will download a public dataset, add it to your repository, load it in R, and create a simple plot.

5.1 Download the data

We will use the Gapminder dataset, which contains country-level data on life expectancy, GDP per capita, and population from 1952 to 2007.

1. Create a `data/` folder in your repository
2. In R, run the following to download the data and save it as a CSV file:

```
install.packages("gapminder")
library(gapminder)
write.csv(gapminder, "data/gapminder.csv", row.names = FALSE)
```

3. Verify that the file `data/gapminder.csv` exists in your repository folder

5.2 Load the data and make a plot

Create a new R script called `problem_sets/ps1_plot.R` with the following steps:

1. Load the data from the CSV file:

```
library(ggplot2)
gapminder <- read.csv("data/gapminder.csv")
```

2. Explore the data briefly:

```
head(gapminder)
str(gapminder)
```

3. Create a plot showing how life expectancy has changed over time for a few countries of your choice. For example:

```
countries <- c("Spain", "France", "Germany", "United Kingdom")
df <- gapminder[gapminder$country %in% countries, ]

ggplot(df, aes(x = year, y = lifeExp, color = country)) +
  geom_line() +
  geom_point() +
  labs(x = "Year", y = "Life expectancy",
       title = "Life expectancy over time") +
  theme_minimal()

ggsave("problem_sets/ps1_plot.png", width = 7, height = 5)
```

4. You can choose different countries or a different variable (e.g., `gdpPercap` or `pop`)

5.3 Commit everything

Stage and commit the data file, the R script, and the plot:

```
git add data/gapminder.csv problem_sets/ps1_plot.R problem_sets/ps1_plot.png
git commit -m "Add gapminder data and first plot"
git push
```

6 Task 6: View Your Commit History

Check that your commits were recorded properly.

6.1 Option A: On the Web

1. Go to your repository page on GitHub
2. Click on "Commits" (or the clock icon with a number)
3. You should see your commits listed with messages and timestamps

6.2 Option B: Command Line

```
git log --oneline
```

This shows a compact list of your commits.

6.3 Option C: RStudio

1. In the Git pane, click "History" (clock icon)
2. Browse through your commits

Take a screenshot of your commit history showing at least 3–4 commits.

7 Submission

Send me an email with:

1. The URL of your GitHub repository
   (e.g., `https://github.com/username/aqmss2`)
2. A screenshot of your commit history

I will check that:

- Your repository is **public**
- It contains a README with your name
- It has a `data/` folder with the Gapminder CSV file
- It has a `problem_sets/` folder with your R files and a saved plot
- There are multiple commits in the history

8 Optional: Installing Git Locally

If you want to use Git from the command line, you need to install it first.

8.1 Installation

- **Mac**: Git comes pre-installed. Or install via Homebrew: `brew install git`
- **Windows**: Download from `https://git-scm.com/download/win`
- **Linux**: Use your package manager, e.g., `sudo apt install git`

8.2 First-Time Configuration

After installing, configure your identity (run once):

```
git config --global user.name "Your Name"
git config --global user.email "your@email.com"
```

8.3 Cloning an Existing Repository

If you created the repository on GitHub first, download it to your computer:

```
git clone https://github.com/YOUR-USERNAME/aqmss2.git
cd aqmss2
```

8.4 Configuring RStudio

To use Git in RStudio:

1. Go to Tools → Global Options → Git/SVN
2. Make sure "Git executable" points to your Git installation
3. Restart RStudio

9 Quick Reference: Common Git Commands

Command	**What it does**
`git status`	Show which files have changed
`git add <file>`	Stage a file for commit
`git add .`	Stage all changed files
`git commit -m "msg"`	Save staged changes with a message
`git push`	Upload commits to GitHub
`git pull`	Download changes from GitHub
`git log --oneline`	Show commit history (compact)
`git diff`	Show changes not yet staged

10 Resources

- GitHub's official guides: `https://docs.github.com/en/get-started`
- Happy Git with R: `https://happygitwithr.com`
- Git cheat sheet: `https://education.github.com/git-cheat-sheet-education.pdf`
- Interactive Git exercises: `https://gitexercises.fracz.com`
- Pro Git book (free): `https://git-scm.com/book/en/v2`

Appendix: Setting Up the Command Line on Windows

If you are using Windows and want to follow the command line instructions in this assignment, you need to set up a Unix-like terminal first. The default Windows terminals (Command Prompt and PowerShell) use different commands and syntax from what is shown in this assignment and in most online tutorials.

10.1 Recommended: Use Git Bash

When you install Git for Windows (from `https://git-scm.com/download/win`), it includes **Git Bash**—a terminal emulator that provides a Unix-like command line environment on Windows. This is the easiest way to get started.

1. Download and run the Git for Windows installer
2. During installation, accept the default options. In particular:
    - Select "Use Git from Git Bash only" (or the default option)
    - Select "Use bundled OpenSSH"
    - Select "Checkout Windows-style, commit Unix-style line endings"
3. After installation, you can open Git Bash by:
    - Searching for "Git Bash" in the Start menu
    - Right-clicking in any folder and selecting "Git Bash Here"

Git Bash supports all the commands used in this assignment (`mkdir`, `cd`, `echo`, `git`, etc.) with the same syntax as on Mac and Linux.

10.2 Alternative: Windows Terminal with PowerShell

If you prefer to use PowerShell, be aware of these differences:

- Most Git commands work the same (`git add`, `git commit`, etc.)
- File paths use backslashes (\) instead of forward slashes (/), though Git accepts both
- The `echo` command syntax differs: use `Set-Content` or `Out-File` instead of > redirection
- Some Unix commands (`touch`, `cat`) are not available by default

10.3 Setting up your PATH

After installing Git, make sure it is accessible from your terminal:

```
# In Git Bash or PowerShell, run:
git --version
```

If this returns a version number (e.g., `git version 2.43.0`), you are ready to go. If not, you may need to add Git to your system PATH:

1. Open Settings → System → About → Advanced system settings
2. Click "Environment Variables"
3. Under "System variables", find `Path` and click "Edit"
4. Add the path to your Git installation (usually `C:\Program Files\Git\cmd`)
5. Click OK and restart your terminal

11 Using Positron Instead of RStudio

In this course, we recommend using **Positron** (`https://positron.posit.co`) as your IDE instead of RStudio. Positron is a next-generation data science IDE built by Posit (the same company behind RStudio). It is based on VS Code and designed for R and Python. Here is what differs from RStudio when completing this assignment.

11.1 Installing Positron

1. Download Positron from `https://positron.posit.co`
2. Install it like any other application
3. On first launch, Positron will detect your R installation automatically

11.2 Differences for This Assignment

The following table maps the RStudio instructions in this assignment to their Positron equivalents:

RStudio	**Positron**
File → New Project → Version Control → Git	File → New Folder, then open the folder. Use the built-in terminal (`Ctrl+‘`) to run `git clone`.
Git pane (top-right)	Source Control panel in the left sidebar (branch icon), or `Ctrl+Shift+G`
Check box next to file to stage it	Click the `+` icon next to the file in the Source Control panel
Click “Commit” button	Click the checkmark icon in the Source Control panel, type your message in the text box
Click “Push” button	Click the `...` menu in Source Control → Push, or use the terminal: `git push`
Click “History” (clock icon)	Use the terminal: `git log --oneline`, or install the “Git Graph” extension for a visual history
Files pane (bottom-right)	Explorer panel in the left sidebar (top icon), or `Ctrl+Shift+E`
File → New File → R Script	File → New File, then select “R File”
Tools → Global Options → Git/SVN	Git is detected automatically. If not, open Settings (`Ctrl+,`) and search for “git.path”

11.3 Key Advantages of Positron

- **Integrated terminal**: Positron has a built-in terminal (`Ctrl+‘`) where you can run Git commands directly—no need to switch to a separate application
- **Better Git integration**: The Source Control panel shows diffs, staged changes, and commit history in one place
- **Extensions**: You can install extensions (e.g., “Git Graph” for visual commit history, “R” for enhanced R support) from the Extensions panel
- **Multiple languages**: Positron works equally well with R and Python, which is useful if you work across both

Note: All the command line instructions in this assignment work identically regardless of whether you use RStudio, Positron, or a standalone terminal. The differences only apply when using the graphical interface.

12 Using Sublime Text 4 as a Code Editor

Sublime Text is a fast, lightweight code editor. Unlike RStudio or Positron, it is not an IDE—it does not come with an R console, file browser, or plot viewer out of the box. Instead, you write R code in Sublime Text and send it to a separate R console running alongside it. This minimal setup requires only two packages.

12.1 Step 1: Install Sublime Text 4

Download from `https://www.sublimetext.com` and install it.

12.2 Step 2: Install Package Control

Package Control is Sublime Text's package manager. To install it:

1. Open the Command Palette: `Ctrl+Shift+P` (Windows/Linux) or `Cmd+Shift+P` (Mac)
2. Type "Install Package Control" and press Enter
3. Wait for the confirmation message

12.3 Step 3: Install R-IDE and SendCode

Using Package Control, install two packages:

1. Open the Command Palette (`Ctrl+Shift+P` / `Cmd+Shift+P`)
2. Type "Package Control: Install Package" and press Enter
3. Search for **R-IDE** and install it (provides R syntax highlighting, code completions, and function signatures)
4. Repeat the process and install **SendCode** (sends code from the editor to an external R console)

12.4 Step 4: Configure SendCode

SendCode needs to know where to send your code. Open its settings:

1. Go to Preferences → Package Settings → SendCode → Settings
2. In the right-hand pane (user settings), paste the following:

```
{
    "prog": "Terminal"
}
```

On **Mac**, set `"prog"` to `"Terminal"` to send code to the built-in Terminal (where you will run R), or to `"iTerm"` if you use iTerm2. On **Windows**, set it to `"Cmder"`, `"ConEmu"`, or another terminal emulator. On **Linux**, set it to `"tmux"` or `"linux-terminal"`.

12.5 Step 5: The Workflow

1. Open a terminal window and start R by typing `R` and pressing Enter
2. In Sublime Text, open your `.R` file
3. Place your cursor on a line of code and press `Ctrl+Enter` (Windows/Linux) or `Cmd+Enter` (Mac)—SendCode sends that line to the R console and moves to the next line
4. To send a selection, highlight multiple lines and press the same shortcut

That is the entire setup. You edit in Sublime Text, execute in the R console, and switch between them as needed. For Git operations, use the terminal.