Setting Up Multi-Language Quarto Documents on macOS

A Complete Guide to R, Python, Julia, and Observable JS Integration

[Your Name]

2025-02-12

Table of contents

0.1 Introduction

Creating multi-language documents in Quarto can be powerful but requires careful setup. This guide walks through setting up a macOS environment to run Quarto documents that combine R, Python, Julia, and Observable JS. We'll use the Palmer Penguins dataset as an example, creating visualizations in each language.

0.2 System Requirements

- macOS (tested on Monterey and later)
- At least 8GB RAM
- ~5GB free disk space for all installations
- Admin privileges for installations
- Internet connection for package downloads

0.3 Basic Setup

0.3.1 1. R and RStudio

1. Install R from CRAN
2. Install RStudio
3. Install Quarto

Install required R packages:

```
install.packages(c(
  "palmerpenguins",  # for example data
  "dplyr",           # for data manipulation
  "ggplot2",         # for R plotting
  "scales",          # for plot formatting
  "reticulate",      # for Python integration
```

```
  "JuliaCall"        # for Julia integration
))
```

0.3.2 2. Python Setup

1. Install Anaconda (recommended) or Miniconda
2. Install required Python packages:

```
conda install pandas seaborn matplotlib
```

3. Verify Python configuration in R:

```
library(reticulate)
py_config()  # Should show your Anaconda Python path
```

> **!** Common Python Issues
>
> - If `py_config()` doesn't show your Anaconda installation, you may need to specify it:
>
>   ```
>   use_python("/path/to/anaconda/bin/python")
>   ```
>
> - Seaborn styling requires specific syntax:
>
>   ```
>   sns.set_theme(style="whitegrid")  # Don't use plt.style.use()
>   ```

0.3.3 3. Julia Setup

1. Install Julia from julialang.org
2. Add Julia to PATH (usually automatic with installer)
3. Install required Julia packages:

```
using Pkg
Pkg.add([
  "UnicodePlots",
  "DataFrames",
  "CSV",
  "Statistics"
])
```

4. Setup Julia-R connection:

```
library(JuliaCall)
julia_setup()
```

> ⚠ Julia Gotchas
>
> - Some Julia plotting backends have system dependencies
> - Use UnicodePlots for most reliable results
> - Ensure Julia working directory has write permissions

0.3.4 4. Observable JS Setup

No separate installation needed, but your Quarto document must include:

```
dependencies:
  - name: "@observablehq/plot"
    version: latest
```

0.4 Verifying Your Setup

Run these commands to verify each language integration:

0.4.1 R Environment

```
sessionInfo()
```

0.4.2 Python Environment

```
library(reticulate)
py_config()
```

0.4.3 Julia Environment

```r
library(JuliaCall)
julia_eval("versioninfo()")
```

0.5 Testing a Minimal Example

Here's a minimal example combining all languages:

```r
# R code
print("Hello from R!")
```

```
[1] "Hello from R!"
```

```r
# Python code
print("Hello from Python!")
```

```
Hello from Python!
```

```r
# Julia code
println("Hello from Julia!")
```

```
Hello from Julia!
```

```
// Observable JS code
md`Hello from Observable JS!`
```

0.6 Common Issues and Solutions

0.6.1 Working Directory Problems

All languages need to access the same files. Check working directories:

```r
# R
getwd()

# Python
import os
os.getcwd()
```

```
# Julia
pwd()
```

0.6.2 File Permissions

Ensure your working directory has write permissions:

```
chmod 755 /path/to/working/directory
```

0.6.3 Memory Issues

Monitor memory usage:

```
# R memory usage
gc()

# Python memory usage
import psutil
psutil.Process().memory_info().rss / 1024 / 1024  # MB
```

0.7 Full Example: Palmer Penguins Visualization

Here's a complete example creating the same visualization in all four languages. See the companion repository at [GitHub Link] for the full code.

0.8 Best Practices

1. **Test Incrementally**
   - Start with R chunks
   - Add Python integration
   - Add Julia integration
   - Finally, add Observable JS
2. **Use Version Control**
   - Track your environment specifications
   - Document package versions
   - Share reproducible setups

3. **Document Dependencies**
    - Create a requirements.txt for Python
    - Use renv for R
    - Document Julia package versions

0.9 Resources

- Quarto Documentation
- R Markdown Cookbook
- Reticulate Documentation
- JuliaCall Documentation
- Observable Documentation

0.10 Conclusion

Setting up a multi-language Quarto environment requires attention to detail but offers powerful capabilities for data analysis and visualization. Keep your system updated, use version control, and document your setup for reproducibility.

0.11 Session Info

```
sessionInfo()
```

```
R version 4.4.2 (2024-10-31)
Platform: aarch64-apple-darwin20
Running under: macOS Sequoia 15.3

Matrix products: default
BLAS:   /Library/Frameworks/R.framework/Versions/4.4-arm64/Resources/lib/libRblas.0.dylib
LAPACK: /Library/Frameworks/R.framework/Versions/4.4-arm64/Resources/lib/libRlapack.dylib;  L

locale:
[1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8

time zone: America/Los_Angeles
tzcode source: internal

attached base packages:
```

```
[1] stats     graphics  grDevices datasets  utils     methods   base

other attached packages:
 [1] shiny_1.10.0       styler_1.10.3      quarto_1.4.4       pacman_0.5.1       readxl_1.4.3
[11] purrr_1.0.4        readr_2.1.5        tidyr_1.3.1        tibble_3.2.1       ggplot2_3.5
[21] datapasta_3.1.0    ggthemes_5.1.0     conflicted_1.2.0   DT_0.33            naniar_1.1.0

loaded via a namespace (and not attached):
 [1] tidyselect_1.2.1  viridisLite_0.4.2 R.utils_2.12.3    fastmap_1.2.0     promises_1.3.2
[11] processx_3.8.5    magrittr_2.0.3    compiler_4.4.2    rlang_1.1.5       tools_4.4.2
[21] xml2_1.3.6        pkgload_1.4.0     miniUI_0.1.1.1    R.cache_0.16.0    withr_3.0.2
[31] colorspace_2.1-1  scales_1.3.0      cli_3.6.3         generics_0.1.3    remotes_2.5.0
[41] vctrs_0.6.5       Matrix_1.7-2      jsonlite_1.8.9    hms_1.1.3         JuliaCall_0.17.6
[51] gtable_0.3.6      later_1.4.1       munsell_0.5.1     pillar_1.10.1     htmltools_0.5.8
[61] R.methodsS3_1.8.2 memoise_2.0.1     snakecase_0.11.1  httpuv_1.6.15     Rcpp_1.0.14
```

0.12 Appendix: Fully Documented multi3.qmd

Here's a complete, documented version of the multi-language visualization example. Each section includes detailed explanations of the code and important considerations.

```
---
title: "Palmer Penguins: Multi-Language Visualization Comparison"
author: "[Your Name]"
date: "`r Sys.Date()`"
format:
  html:
    # Enable code folding for cleaner output
    code-fold: true
    # Bundle all resources into single HTML
    embed-resources: true
    theme: default
    execute:
      echo: true
# Required for Observable Plot
dependencies:
  - name: "@observablehq/plot"
    version: latest
---

## Introduction
This document demonstrates how to create the same visualization - a scatterplot of
```

```
bill depth versus bill length by species - using four different programming
languages: R, Python, Julia, and Observable JS. Each implementation showcases
the unique strengths of its ecosystem.

## Data Preparation in R
# First, we load required libraries and prepare our data
```

```r
# Load required libraries
library(palmerpenguins)  # Contains our dataset
library(dplyr)           # For data manipulation
library(ggplot2)         # For visualization
library(scales)          # For plot scaling

# Prepare and save data for other languages
# We remove NA values to ensure consistency across languages
data <- penguins %>% na.omit()

# Save to CSV for other languages to read
# Note: row.names=FALSE prevents adding an index column
write.csv(data, "penguins.csv", row.names = FALSE)

# Display data structure for verification
glimpse(data)
```

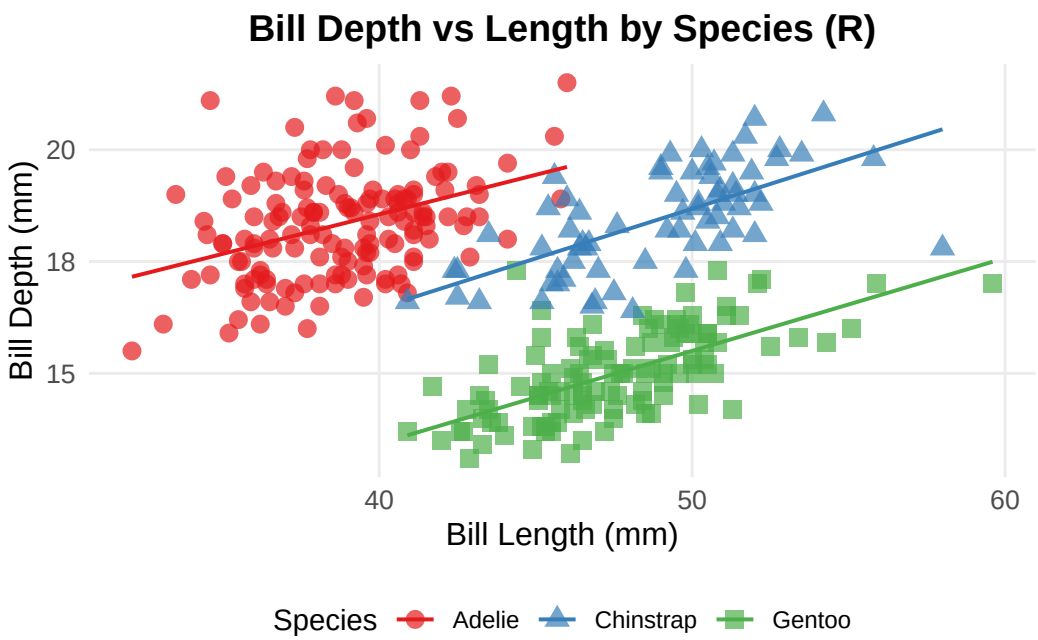
```
Rows: 333
Columns: 8
$ species           <fct> Adelie, Adelie, Adelie, Adelie, Adelie, Ad~
$ island            <fct> Torgersen, Torgersen, Torgersen, Torgersen~
$ bill_length_mm    <dbl> 39, 40, 40, 37, 39, 39, 39, 41, 39, 35, 37~
$ bill_depth_mm     <dbl> 19, 17, 18, 19, 21, 18, 20, 18, 21, 21, 18~
$ flipper_length_mm <int> 181, 186, 195, 193, 190, 181, 195, 182, 19~
$ body_mass_g       <int> 3750, 3800, 3250, 3450, 3650, 3625, 4675, ~
$ sex               <fct> male, female, female, female, male, female~
$ year              <int> 2007, 2007, 2007, 2007, 2007, 2007, 2007, ~
```

0.13 Visualization in R (ggplot2)

1 Create a sophisticated scatter plot with trend lines

```
ggplot(data, aes(x = bill_length_mm,
                 y = bill_depth_mm,
                 color = species,
                 shape = species)) +
  # Add points with some transparency
  geom_point(size = 3, alpha = 0.7) +
  # Add trend lines without confidence intervals
  geom_smooth(method = "lm", se = FALSE, linewidth = 0.7, alpha = 0.5) +
  # Use colorblind-friendly palette
  scale_color_brewer(palette = "Set1") +
  # Add clear labels
  labs(title = "Bill Depth vs Length by Species (R)",
       x = "Bill Length (mm)",
       y = "Bill Depth (mm)",
       color = "Species",
       shape = "Species") +
  # Use minimal theme for clean look
  theme_minimal() +
  # Customize theme elements
  theme(
    plot.title = element_text(hjust = 0.5, size = 14, face = "bold"),
    legend.position = "bottom",
    panel.grid.minor = element_blank(),
    axis.text = element_text(size = 10),
    axis.title = element_text(size = 12)
  )
```

```
`geom_smooth()` using formula = 'y ~ x'
```

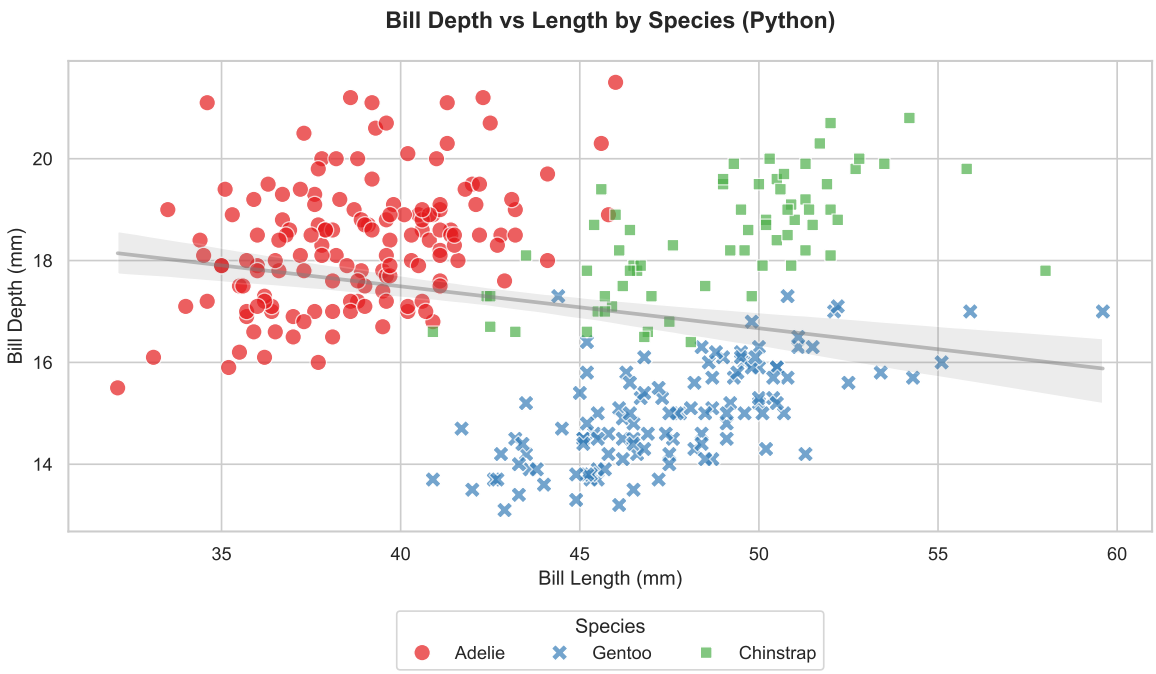


1.1 Visualization in Python (Seaborn)

2 Create equivalent visualization using seaborn

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Set style - use whitegrid for consistency with R plot
sns.set_theme(style="whitegrid")
sns.set_palette("Set1")  # Match R's color scheme

# Read the CSV file created by R
penguins = pd.read_csv("penguins.csv").dropna()

# Create figure with specific size
plt.figure(figsize=(10, 6))

# Create scatterplot with regression lines
sns.scatterplot(data=penguins,
                x='bill_length_mm',
                y='bill_depth_mm',
```

```
                hue='species',
                style='species',
                s=100,  # Point size
                alpha=0.7)  # Match R's transparency

# Add overall trend line
sns.regplot(data=penguins,
            x='bill_length_mm',
            y='bill_depth_mm',
            scatter=False,
            color='gray',
            line_kws={'alpha': 0.5})

# Customize plot
plt.title('Bill Depth vs Length by Species (Python)',
          pad=20,
          size=14,
          weight='bold')
plt.xlabel('Bill Length (mm)', size=12)
plt.ylabel('Bill Depth (mm)', size=12)

# Adjust legend position to match R plot
plt.legend(title='Species', bbox_to_anchor=(0.5, -0.15),
          loc='upper center', ncol=3)

# Adjust layout to prevent cutoff
plt.tight_layout()
plt.show()
```



2.1 Visualization in Julia

3 Create a text-based scatter plot using UnicodePlots

```julia
using UnicodePlots  # For reliable plotting in any environment
using DataFrames    # For data manipulation
using CSV           # For reading CSV
using Statistics    # For basic stats

# Read data
penguins = CSV.read("penguins.csv", DataFrame)
```

```
333×8 DataFrame
 Row   species    island     bill_length_mm  bill_depth_mm  flipper_length_mm
       String15   String15   Float64         Float64        Int64

   1   Adelie     Torgersen            39.1           18.7                 181
   2   Adelie     Torgersen            39.5           17.4                 186
   3   Adelie     Torgersen            40.3           18.0                 195
```

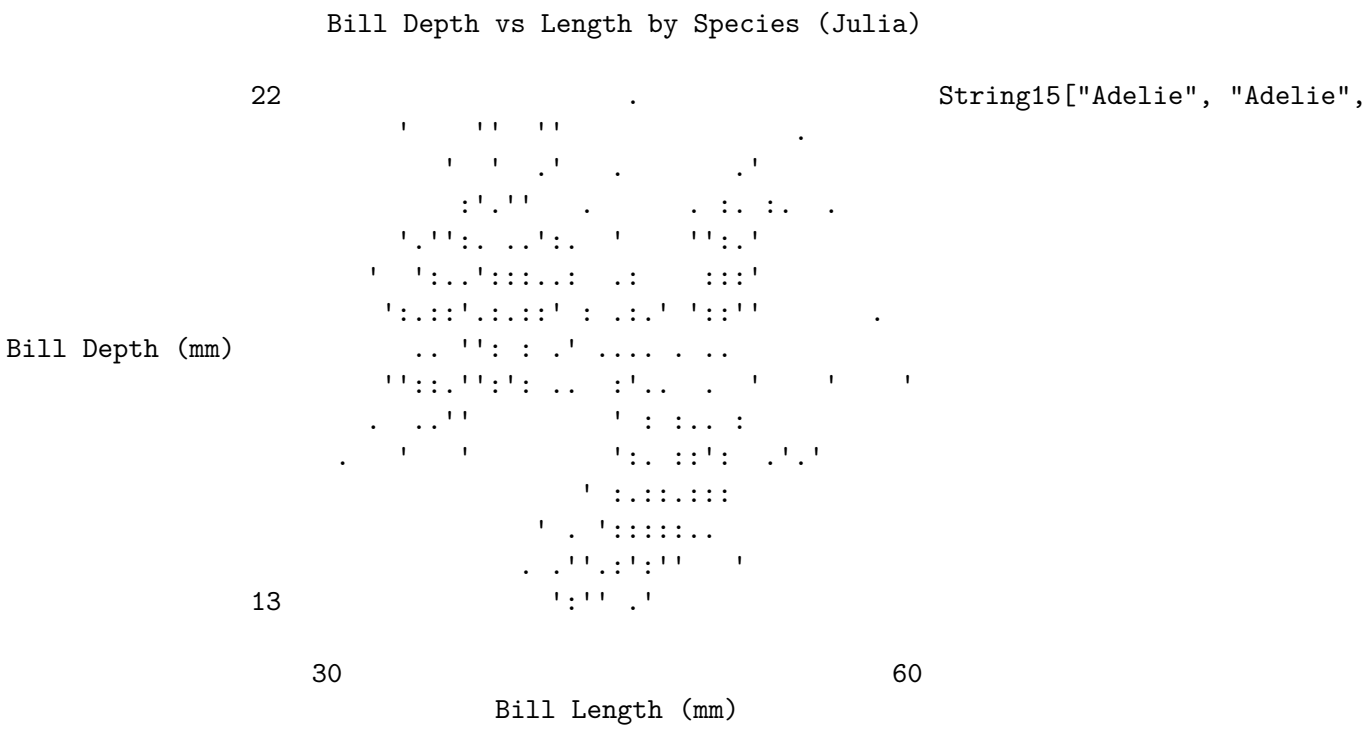
```
  4   Adelie     Torgersen           36.7          19.3               193
  5   Adelie     Torgersen           39.3          20.6               190
  6   Adelie     Torgersen           38.9          17.8               181
  7   Adelie     Torgersen           39.2          19.6               195
  8   Adelie     Torgersen           41.1          17.6               182

327   Chinstrap  Dream               46.8          16.5               189
328   Chinstrap  Dream               45.7          17.0               195
329   Chinstrap  Dream               55.8          19.8               207
330   Chinstrap  Dream               43.5          18.1               202
331   Chinstrap  Dream               49.6          18.2               193
332   Chinstrap  Dream               50.8          19.0               210
333   Chinstrap  Dream               50.2          18.7               198
                                                  3 columns and 318 rows omitted
```

```
dropmissing!(penguins)  # Remove any NA values
```

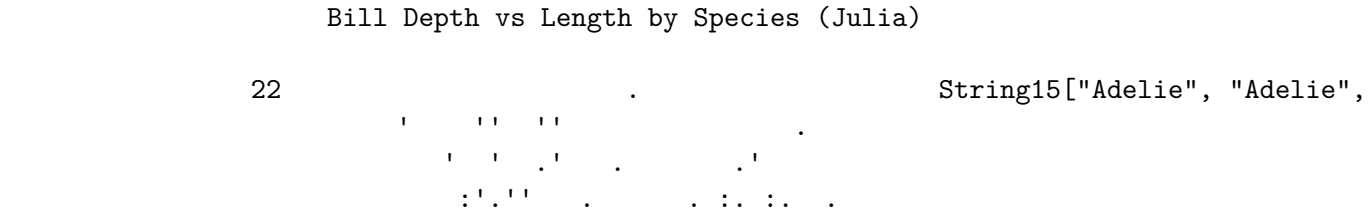
```
333×8 DataFrame
 Row   species    island     bill_length_mm  bill_depth_mm  flipper_length_mm
       String15   String15   Float64         Float64        Int64

   1   Adelie     Torgersen           39.1          18.7               181
   2   Adelie     Torgersen           39.5          17.4               186
   3   Adelie     Torgersen           40.3          18.0               195
   4   Adelie     Torgersen           36.7          19.3               193
   5   Adelie     Torgersen           39.3          20.6               190
   6   Adelie     Torgersen           38.9          17.8               181
   7   Adelie     Torgersen           39.2          19.6               195
   8   Adelie     Torgersen           41.1          17.6               182

 327   Chinstrap  Dream               46.8          16.5               189
 328   Chinstrap  Dream               45.7          17.0               195
 329   Chinstrap  Dream               55.8          19.8               207
 330   Chinstrap  Dream               43.5          18.1               202
 331   Chinstrap  Dream               49.6          18.2               193
 332   Chinstrap  Dream               50.8          19.0               210
 333   Chinstrap  Dream               50.2          18.7               198
                                                   3 columns and 318 rows omitted
```

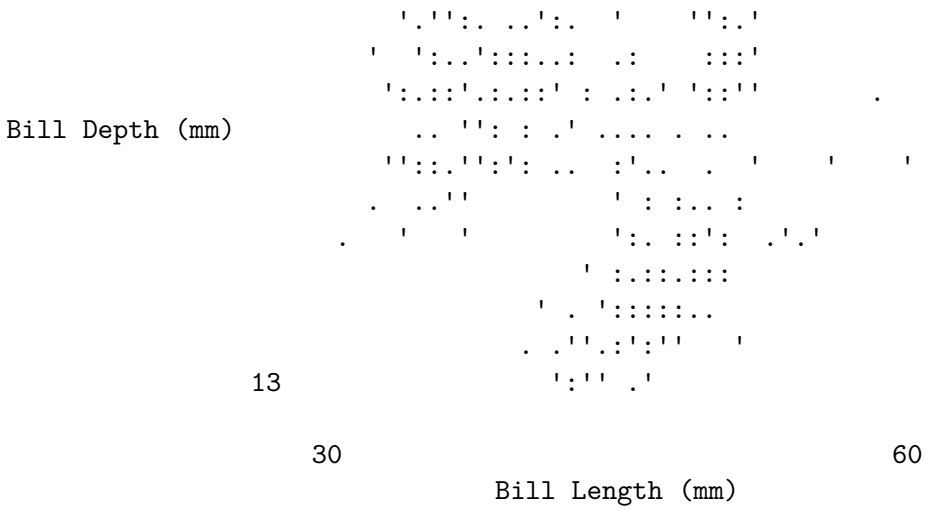
```
# Create scatter plot
# Note: Using UnicodePlots for reliability across environments
```

```
plt = scatterplot(
    penguins.bill_length_mm,
    penguins.bill_depth_mm,
    name = string.(penguins.species),  # Convert species to strings
    title = "Bill Depth vs Length by Species (Julia)",
    xlabel = "Bill Length (mm)",
    ylabel = "Bill Depth (mm)",
    canvas = DotCanvas  # Use dots for points
)
```




```
# Show plot
plt
```





3.1 Visualization in Observable JS

4 Create an interactive visualization using Observable Plot

```
// Import Observable Plot library
import { Plot } from "@observablehq/plot"

// Load and prepare data
penguins = FileAttachment("penguins.csv").csv()

// Create plot with Observable Plot
Plot.plot({
  // Use consistent color scheme
  color: {scheme: "category10"},
  // Add scatter plot points
  marks: [
    Plot.dot(penguins, {
      x: "bill_length_mm",
      y: "bill_depth_mm",
      stroke: "species",
      fill: "species"
    })
  ],
  // Add axis labels
  x: {label: "Bill Length (mm)"},
```

```
  y: {label: "Bill Depth (mm)"},
  // Add title
  title: "Bill Depth vs Length by Species (Observable JS)"
})
```

4.1 Conclusion

Each language offers unique advantages for visualization:

- **R (ggplot2)**: Excellent defaults and highly customizable aesthetic options
- **Python (Seaborn)**: Clean integration with statistical functions and good handling of categorical variables
- **Julia (UnicodePlots)**: Fast performance and text-based visualization capabilities
- **Observable JS**: Interactive capabilities and web-native visualization

The core visualization remains consistent across all implementations, showing the relationship between bill depth and length across different penguin species, while each implementation showcases the strengths of its respective ecosystem. “‘

4.1.1 Key Implementation Notes

1. **YAML Header**
    - Includes necessary dependencies for Observable Plot
    - Sets up HTML output with code folding
    - Ensures resources are embedded for portability
2. **Data Preparation**
    - Uses R for initial data loading and cleaning
    - Exports clean CSV for other languages
    - Maintains consistent data across all visualizations
3. **Language-Specific Considerations**
    - R: Uses ggplot2 for publication-quality static plots
    - Python: Matches R’s aesthetic choices for consistency
    - Julia: Uses UnicodePlots for reliability
    - Observable JS: Provides interactive web-native visualization
4. **Common Patterns**
    - Consistent color schemes
    - Similar axis labels and titles

- Comparable point sizes and transparencies
- Aligned legend positions where possible

5. **Error Prevention**

   - Explicit NA handling
   - Consistent working directory usage
   - Clear data export/import chain
   - Reliable plotting backends

This implementation provides a robust template for multi-language visualization that can be adapted for other datasets and analysis needs.