Observable JS vs Shiny for Interactive Statistical Reports

A Practical Comparison for Data Analysts and Statisticians

Data Science Team

2025-12-08

Table of contents

1 Executive Summary

Interactive data visualization has become essential for modern statistical reporting. Two leading technologies for creating interactive reports within the Quarto ecosystem are **Observable JavaScript (OJS)** and **R Shiny**. This white paper provides a comprehensive comparison to help statisticians and data analysts choose the appropriate technology for their reporting needs.

> **Key Takeaway**
>
> **Observable JS** excels for self-contained, client-side reports that need broad distribution. **Shiny** excels when you need R's statistical ecosystem or server-side computation for sensitive data.

2 Introduction

2.1 The Evolution of Statistical Reporting

Traditional statistical reports were static documents—PDF or Word files with fixed tables and figures. Modern stakeholders increasingly expect interactive elements that allow them to:

- Filter data by subgroups
- Explore different parameter values
- Drill down into specific observations
- Export customized views

Quarto, the next-generation publishing system from Posit, supports both Observable JS and Shiny as interactivity backends, giving statisticians flexibility in their choice of technology.

2.2 Scope of This Comparison

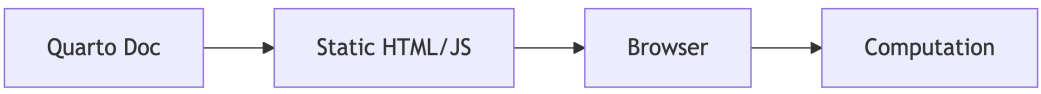
This paper compares Observable JS and Shiny across dimensions critical to statistical reporting:

1. Technical architecture
2. Statistical computing capabilities
3. Deployment and distribution
4. Performance characteristics
5. Learning curve and development experience
6. Security and compliance considerations

3 Technical Architecture

3.1 Observable JS: Client-Side Execution

```
flowchart LR
    A[Quarto Doc] --> B[Static HTML/JS]
    B --> C[Browser]
    C --> D[Computation]
```



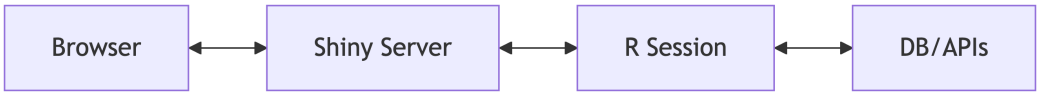
Observable JS Architecture

Observable JS compiles to static HTML and JavaScript files. When a user opens the report:

- **All code executes in the browser**
- Data is embedded in the document or fetched via HTTP
- No server infrastructure required after initial hosting
- Reactivity is handled by Observable's runtime

3.2 Shiny: Server-Side Execution

```
flowchart LR
    A[Browser] <--> B[Shiny Server]
    B <--> C[R Session]
    C <--> D[DB/APIs]
```



Shiny Architecture

Shiny maintains a persistent R session on a server:

- **Code executes on the server**
- Only UI updates are sent to the browser
- Requires running server infrastructure
- Full access to R's computing environment

3.3 Architectural Comparison

Aspect	Observable JS	Shiny
Execution	Client (browser)	Server (R)
State	Browser memory	Server memory
Concurrency	Unlimited	Limited by server
Network	Initial load only	Continuous
Offline	Yes (once loaded)	No

4 Statistical Computing Capabilities

4.1 R/Shiny: The Statistical Powerhouse

Shiny provides direct access to R's comprehensive statistical ecosystem:

Advantages:

- **20,000+ CRAN packages** including specialized methods for:
  - Survival analysis (`survival`, `survminer`)
  - Mixed-effects models (`lme4`, `nlme`)
  - Bayesian inference (`brms`, `rstanarm`)
  - Causal inference (`MatchIt`, `WeightIt`)
  - Meta-analysis (`metafor`, `meta`)

- **Established validation** in regulated industries (pharma, finance)
- **Familiar syntax** for statisticians already using R
- **Heavy computation** can run on powerful servers, not limited by browser

```
# Example: Complex statistical model in Shiny
# This runs on the server with full R capabilities
output$model_summary <- renderPrint({
  model <- lme4::lmer(
    outcome ~ treatment * time + (1 | subject),
    data = filtered_data()
  )
  summary(model)
})
```

4.2 Observable JS: Modern Visualization Focus

Observable provides excellent visualization with growing statistical capabilities:

Advantages:

- **Observable Plot**: Elegant, grammar-of-graphics style plotting
- **D3.js integration**: Unlimited customization potential
- **Arquero**: dplyr-like data transformation in JavaScript
- **Statistical libraries**: `simple-statistics`, `jstat` for basic analyses

Limitations:

- Fewer specialized statistical methods
- Less validation history in regulated environments
- Complex models require custom implementation or API calls

```
//| eval: false
//| echo: true
// Example: Basic statistics in Observable
import { mean, standardDeviation, linearRegression } from "simple-statistics"

stats = ({
  mean: mean(data.map(d => d.value)),
  sd: standardDeviation(data.map(d => d.value)),
  regression: linearRegression(data.map(d => [d.x, d.y]))
})
```

4.3 Statistical Capability Matrix

Capability	Observable JS	Shiny
Descriptive statistics		
Linear regression		
Generalized linear models	Limited	
Mixed-effects models		
Survival analysis		
Bayesian inference		
Time series (ARIMA, etc.)	Limited	
Machine learning	Limited	
Custom visualizations		

5 Deployment and Distribution

5.1 Observable JS: Simplicity and Portability

Deployment options:

1. **Static file hosting**: Any web server, S3, GitHub Pages
2. **Email attachment**: Self-contained HTML file
3. **Shared drives**: No server needed
4. **Quarto Pub**: Free hosting from Posit

Advantages for report distribution:

- Recipients need only a web browser
- No IT infrastructure required
- Works behind firewalls (no external connections)
- Can be archived as single files
- No ongoing server costs

```
# Generate self-contained HTML
quarto render report.qmd --to html

# Result: Single HTML file with embedded data and code
# File size: Typically 1-10 MB depending on data
```

5.2 Shiny: Server Infrastructure Required

Deployment options:

1. **shinyapps.io**: Managed hosting (free tier available)
2. **Posit Connect**: Enterprise server
3. **Shiny Server**: Self-hosted (open source or pro)

4. **Docker/Kubernetes**: Container deployment

Considerations:

- Requires ongoing server maintenance
- Per-user resource consumption
- Session management for concurrent users
- Authentication integration needed for sensitive reports

```
# Deploy to shinyapps.io
rsconnect::deployDoc("report.qmd")

# Or render for Posit Connect
quarto render report.qmd
# Then publish via Connect interface
```

5.3 Deployment Decision Matrix

Scenario	Choice	Rationale
Email to stakeholders	OJS	Self-contained HTML
Dashboard, <10 users	Either	Shiny may be overkill
100+ concurrent users	OJS	No server scaling
Database queries	Shiny	Server-side access
Regulated (21 CFR 11)	Shiny	Audit trail support
Conference presentation	OJS	No network needed
Client deliverable	OJS	No infrastructure

6 Performance Characteristics

6.1 Data Size Considerations

Table 4: Performance by Data Size

Data Size	Observable JS	Shiny
< 10 MB	Excellent	Excellent
10-100 MB	Good	Excellent
100 MB-1 GB	Poor	Good
> 1 GB	Not feasible	Feasible

6.2 Observable JS Performance Profile

Strengths:

- Instant reactivity (no network round-trip)
- Smooth animations and transitions
- Responsive UI updates
- CDN caching for repeat visits

Limitations:

- Browser memory constraints (~1-4 GB practical limit)
- Initial load time proportional to data size
- No background computation
- Single-threaded (UI can freeze during computation)

6.3 Shiny Performance Profile

Strengths:

- Handle arbitrarily large datasets
- Server-side caching across users
- Database connections (query what you need)
- Parallel computation via R packages

Limitations:

- Network latency on each interaction
- Server resources scale with users
- Cold start time for new sessions
- WebSocket connection required

6.4 Optimization Strategies

6.4.1 Observable JS

```
// Pre-aggregate data to reduce browser load
aggregated = data
  .groupby("category")
  .rollup({
    count: d => op.count(),
    mean_value: d => op.mean(d.value)
  })

// Use efficient data formats
data = FileAttachment("data.parquet").parquet()
```

6.4.2 Shiny

```
# Use server-side filtering before sending to UI
output$plot <- renderPlot({
  # Only process visible subset
  data %>%
    filter(date >= input$date_range[1],
           date <= input$date_range[2]) %>%
    ggplot(...)
})

# Cache expensive computations
model_result <- reactive({
  expensive_model(data())
}) %>% bindCache(input$params)
```

7 Learning Curve and Development Experience

7.1 For R Statisticians

Aspect	Observable JS	Shiny
Language familiarity	New (JavaScript)	Familiar (R)
Data manipulation	Learn Arquero	Use dplyr
Plotting	Learn Plot/D3	Use ggplot2
Reactivity model	Automatic	Explicit
Debugging	Browser DevTools	RStudio
Time to productivity	2-4 weeks	1-2 weeks

7.2 Observable JS Learning Path

1. **Week 1**: JavaScript basics, Observable notebook syntax
2. **Week 2**: Observable Inputs and reactivity
3. **Week 3**: Observable Plot for visualization
4. **Week 4**: Arquero for data transformation, integration patterns

Resources:

- Observable tutorials
- Quarto OJS documentation

7.3 Shiny Learning Path

1. **Week 1**: Shiny UI components and layout
2. **Week 2**: Reactive expressions and observers

Resources:

- Mastering Shiny by Hadley Wickham
- Quarto Shiny documentation

7.4 Code Comparison: Same Feature

7.4.1 Observable JS

```
//| eval: false
//| echo: true
// Reactive filter with visualization
viewof species = Inputs.select(
  ["All", "Adelie", "Chinstrap", "Gentoo"],
  {label: "Species"}
)

filtered = species === "All"
  ? data
  : data.filter(d => d.species === species)

Plot.plot({
  marks: [
    Plot.dot(filtered, {x: "bill_length", y: "bill_depth", fill: "species"})
  ]
})
```

7.4.2 Shiny

```
# UI
selectInput("species", "Species",
            choices = c("All", "Adelie", "Chinstrap", "Gentoo"))
plotOutput("scatter")

# Server
filtered <- reactive({
  if (input$species == "All") data
  else data %>% filter(species == input$species)
})
```

```
output$scatter <- renderPlot({
  ggplot(filtered(), aes(bill_length, bill_depth, color = species)) +
    geom_point()
})
```

8 Security and Compliance

8.1 Data Privacy Considerations

Concern	Observable JS	Shiny
Data exposure	In browser (DevTools)	Stays on server
PHI/PII	Needs de-identification	Server-side processing
HIPAA	Challenging	Feasible
Audit logging	Manual	Built-in (Connect)

8.2 Observable JS Security Model

Risks:

- All data sent to the browser is accessible to users
- Cannot truly hide business logic
- API keys should never be embedded

Mitigations:

- Pre-aggregate sensitive data
- Use server APIs for sensitive queries
- Embed only de-identified data

8.3 Shiny Security Model

Advantages:

- Data never leaves the server
- Can integrate with enterprise authentication (LDAP, SAML)
- Audit trails available on Posit Connect
- Row-level security through user context

Requirements:

- Secure server configuration
- HTTPS enforcement
- Session timeout policies
- Regular security updates

8.4 Regulatory Compliance

For FDA-regulated environments (21 CFR Part 11) or similar:

Requirement	Observable JS	Shiny
E-signatures	N/A	Connect supports
Audit trail	Limited	Full support
Access controls	Basic	Role-based
Validation	Challenging	Established

Regulatory Note

For GxP environments, Shiny with Posit Connect provides established validation pathways. Observable JS may require additional validation effort.

9 Use Case Recommendations

9.1 When to Choose Observable JS

Best Fit Scenarios

1. **Self-contained deliverables**: Client reports, manuscripts, presentations
2. **Wide distribution**: Reports going to many recipients
3. **Limited IT support**: No server infrastructure available
4. **Exploratory visualization**: Focus on data exploration over complex statistics
5. **Offline access needed**: Field work, conferences, travel
6. **Simple interactivity**: Filters, tooltips, basic calculations

9.2 When to Choose Shiny

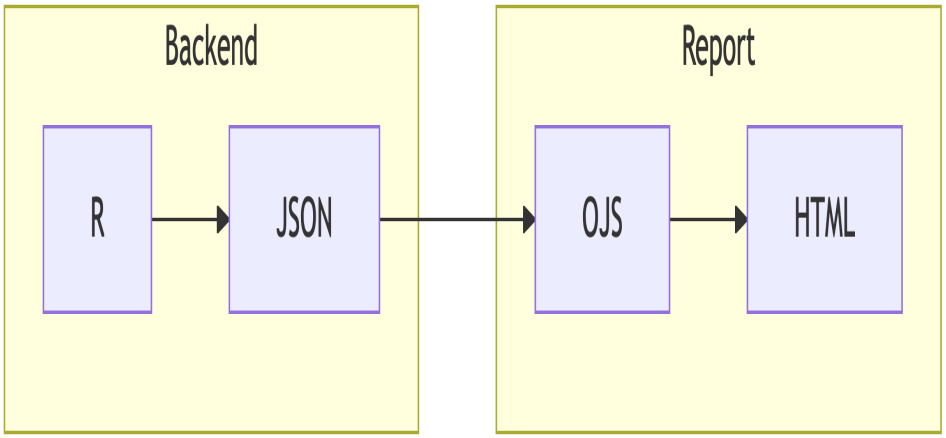
Best Fit Scenarios

1. **Complex statistical methods**: Mixed models, Bayesian analysis, survival curves
2. **Large datasets**: More than ~50 MB of data
3. **Database integration**: Live queries, real-time data
4. **Sensitive data**: PHI, PII, proprietary information
5. **Regulated environments**: Pharma, finance, healthcare
6. **Enterprise deployment**: Integration with existing R infrastructure

9.3 Hybrid Approaches

For some projects, combining both technologies may be optimal:

```
flowchart LR
    subgraph Backend
        A[R] --> B[JSON]
    end
    B --> C
    subgraph Report
        C[OJS] --> D[HTML]
    end
```



Hybrid Architecture

Hybrid workflow:

1. Use R for complex statistical analysis
2. Export results to JSON or Parquet
3. Use Observable JS for interactive visualization
4. Distribute as static HTML

10 Conclusion

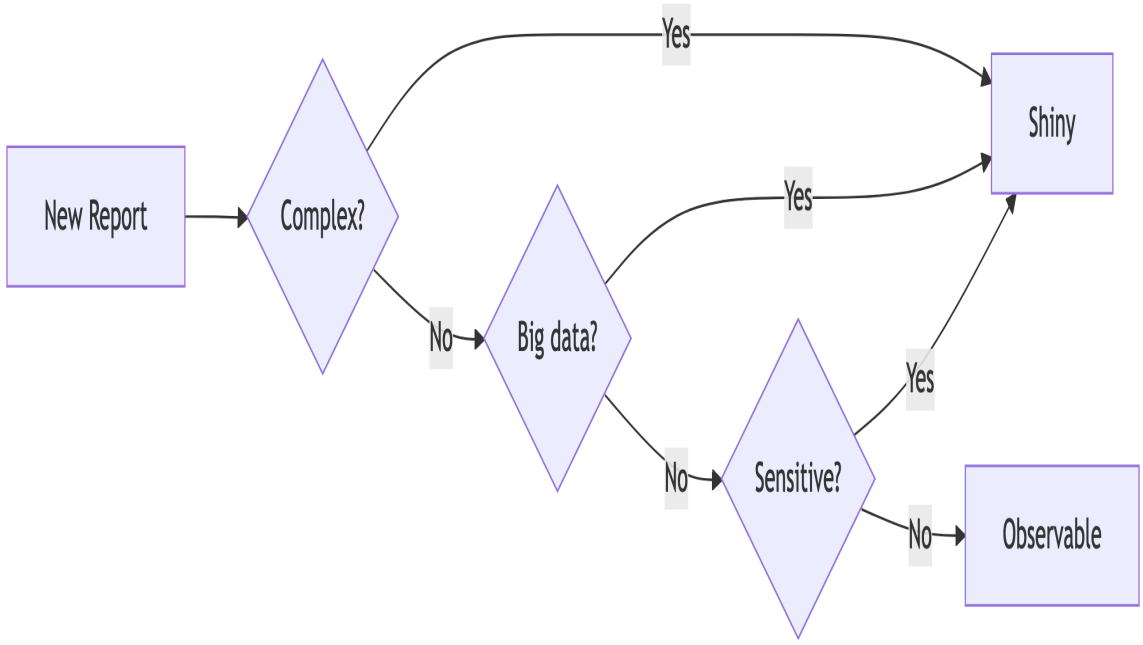
10.1 Summary Comparison

Dimension	Observable JS	Shiny
Best for	Distribution, visualization	Statistics, security
Deployment	Static hosting	Server infrastructure
Data size limit	~50 MB practical	Limited by server
Statistical depth	Basic	Comprehensive

Dimension	Observable JS	Shiny
Learning curve (R users)	Moderate	Low
Offline capability	Yes	No
Enterprise features	Limited	Extensive

10.2 Decision Framework

```
flowchart LR
    A[New Report] --> B{Complex?}
    B -->|Yes| C[Shiny]
    B -->|No| D{Big data?}
    D -->|Yes| C
    D -->|No| E{Sensitive?}
    E -->|Yes| C
    E -->|No| F[Observable]
```



Technology Selection Flowchart

10.3 Final Recommendations

1. **Start with your constraints**: Server availability, data sensitivity, and statistical requirements often dictate the choice.
2. **Consider your audience**: Technical users may appreciate Shiny's depth; broader audiences benefit from Observable's portability.

3. **Invest in learning both**: The skills complement each other, and hybrid approaches are increasingly common.
4. **Prototype quickly**: Both technologies allow rapid prototyping within Quarto—try both before committing.

11 Appendix: Quick Reference

11.1 Observable JS Cheat Sheet

```
// Inputs
viewof x = Inputs.range([0, 100], {label: "Value"})
viewof y = Inputs.select(["A", "B", "C"], {label: "Category"})
viewof z = Inputs.checkbox(["X", "Y", "Z"], {label: "Options"})

// Data
data = FileAttachment("file.csv").csv({typed: true})

// Filtering
filtered = data.filter(d => d.value > x && y.includes(d.category))

// Plotting
Plot.plot({
  marks: [
    Plot.dot(filtered, {x: "xvar", y: "yvar", fill: "group"})
  ]
})
```

11.2 Shiny Cheat Sheet

```
# UI Inputs
sliderInput("x", "Value", 0, 100, 50)
selectInput("y", "Category", c("A", "B", "C"))
checkboxGroupInput("z", "Options", c("X", "Y", "Z"))

# Server
filtered <- reactive({
  data %>% filter(value > input$x, category %in% input$y)
})

output$plot <- renderPlot({
  ggplot(filtered(), aes(xvar, yvar, color = group)) + geom_point()
})
```

12 References

- Wickham, H. (2021). *Mastering Shiny*. O'Reilly Media. https://mastering-shiny.org/
- Quarto Team. (2024). *Quarto Documentation*. Posit. https://quarto.org/
- Observable, Inc. (2024). *Observable Plot Documentation*. https://observablehq.com/plot/
- Posit. (2024). *Shiny Documentation*. https://shiny.posit.co/