Shiny for Python - Complete Guide for Social Sciences

Table of Contents

Introduction to Shiny for Python

What is Shiny for Python?

Shiny for Python brings the powerful reactive programming framework from R to Python. It allows you to create interactive web applications with minimal web development knowledge, focusing on data analysis and visualization.

Key Features

- **Reactive Programming**: Automatic updates when inputs change
- **Rich UI Components**: Professional-looking interfaces with minimal code
- **Seamless Python Integration**: Works with pandas, matplotlib, plotly, and other Python libraries
- **Real-time Interactivity**: Live updates without page refreshes
- **Easy Deployment**: Simple hosting options

When to Use Shiny vs. Other Tools

- **Use Shiny for Python when**: You need reactive programming, complex interactivity, or are familiar with R Shiny
- **Use Streamlit when**: You want rapid prototyping with simpler state management

- **Use Flask/Django when**: You need full web development control

---

Installation and Setup

Installation

```bash
# Install Shiny for Python
pip install shiny

# Additional packages for data science
pip install pandas numpy matplotlib seaborn plotly
pip install scikit-learn wordcloud

# For development
pip install shinylive  # For deployment to web
```

Development Environment Setup

```python
# Create a new directory for your Shiny app
# mkdir my_shiny_app
# cd my_shiny_app

# Create app.py file with basic structure
```

Running Your First App

```bash
# Run the app
shiny run app.py

# Run with hot reload for development
shiny run app.py --reload

# Run on specific port
shiny run app.py --port 8080
```

Basic Shiny App Structure

Minimal App Example

python

```
# app.py
from shiny import App, ui, render

# Define UI
app_ui = ui.page_fluid(
    ui.h1("My First Shiny App"),
    ui.input_text("name", "Enter your name:", "World"),
    ui.output_text("greeting")
)

# Define server logic
def server(input, output, session):
    @output
    @render.text
    def greeting():
        return f"Hello, {input.name()}!"

# Create app
app = App(app_ui, server)
```

App Structure Explained

python

```
from shiny import App, ui, render, reactive

# 1. UI Definition
app_ui = ui.page_fluid(
    # Page layout and components go here
    ui.h1("Page Title"),
    ui.p("Description text"),
    # Input widgets
    ui.input_slider("slider", "Select value:", min=0, max=100, value=50),
    # Output placeholders
    ui.output_plot("my_plot")
)

# 2. Server Function
def server(input, output, session):
    # Reactive functions and output renderers go here

    @output
    @render.plot
    def my_plot():
        # Generate plot based on inputs
        import matplotlib.pyplot as plt
        fig, ax = plt.subplots()
        ax.plot([1, 2, 3], [input.slider(), input.slider()*2, input.slider()*3])
        return fig

# 3. App Creation
app = App(app_ui, server)

# 4. Optional: Custom configuration
if __name__ == "__main__":
    app.run()
```

UI Components

Page Layouts

```
python
```

```python
from shiny import ui

# Fluid page (responsive)
app_ui = ui.page_fluid(
    ui.h1("Fluid Layout"),
    ui.p("Content adapts to screen size")
)

# Fixed page (fixed width)
app_ui = ui.page_fixed(
    ui.h1("Fixed Layout"),
    ui.p("Fixed width layout")
)

# Fillable page (full height)
app_ui = ui.page_fillable(
    ui.h1("Fillable Layout"),
    ui.p("Uses full viewport height")
)

# Navigation bar page
app_ui = ui.page_navbar(
    ui.nav("Tab 1", ui.p("Content for tab 1")),
    ui.nav("Tab 2", ui.p("Content for tab 2")),
    ui.nav("Tab 3", ui.p("Content for tab 3")),
    title="My Dashboard"
)
```

Layout Components

python

```
# Row and column system
app_ui = ui.page_fluid(
    ui.row(
        ui.column(4, ui.p("Column 1 (1/3 width)")),
        ui.column(4, ui.p("Column 2 (1/3 width)")),
        ui.column(4, ui.p("Column 3 (1/3 width)"))
    ),
    ui.row(
        ui.column(6, ui.p("Half width")),
        ui.column(6, ui.p("Half width"))
    )
)

# Sidebar layout
app_ui = ui.page_sidebar(
    ui.sidebar(
        ui.h3("Controls"),
        ui.input_slider("n", "Number of observations:", min=10, max=1000, value=100),
        ui.input_select("dist", "Distribution:", choices=["Normal", "Uniform", "Exponential"])
    ),
    ui.h1("Main Content"),
    ui.output_plot("histogram")
)

# Card layout
app_ui = ui.page_fluid(
    ui.card(
        ui.card_header("Data Summary"),
        ui.card_body(
            ui.p("This card contains summary statistics."),
            ui.output_table("summary_table")
        )
    )
)
```

Text and HTML Elements

python

```
app_ui = ui.page_fluid(
    # Headers
    ui.h1("Main Title"),
    ui.h2("Subtitle"),
    ui.h3("Section Header"),

    # Text elements
    ui.p("Regular paragraph text"),
    ui.strong("Bold text"),
    ui.em("Italic text"),
    ui.code("Code text"),

    # HTML elements
    ui.div("Content in a div", class_="my-class"),
    ui.span("Inline content"),

    # Links and images
    ui.a("Link to Google", href="https://google.com"),
    ui.img(src="path/to/image.jpg", alt="Description"),

    # Lists
    ui.tags.ul(
        ui.tags.li("Item 1"),
        ui.tags.li("Item 2"),
        ui.tags.li("Item 3")
    ),

    # Custom HTML
    ui.HTML("<div class='custom'>Custom HTML content</div>")
)
```

Server Logic and Reactivity

Basic Reactive Programming

python

```python
from shiny import App, ui, render, reactive

app_ui = ui.page_fluid(
    ui.input_slider("n", "Sample size:", min=10, max=1000, value=100),
    ui.input_select("dist", "Distribution:",
                    choices={"norm": "Normal", "unif": "Uniform", "exp": "Exponential"}),
    ui.output_plot("histogram"),
    ui.output_text("summary")
)

def server(input, output, session):
    # Reactive expression - computed once per change
    @reactive.Calc
    def generate_data():
        import numpy as np

        n = input.n()
        dist = input.dist()

        if dist == "norm":
            data = np.random.normal(0, 1, n)
        elif dist == "unif":
            data = np.random.uniform(-2, 2, n)
        else:  # exp
            data = np.random.exponential(1, n)

        return data

    # Output: Histogram
    @output
    @render.plot
    def histogram():
        import matplotlib.pyplot as plt

        data = generate_data()  # Use reactive expression

        fig, ax = plt.subplots(figsize=(10, 6))
        ax.hist(data, bins=30, alpha=0.7, edgecolor='black')
        ax.set_title(f"Histogram of {input.dist()} Distribution (n={input.n()})")
        ax.set_xlabel("Value")
        ax.set_ylabel("Frequency")

        return fig
    # Output: Summary statistics
    @output
    @render.text
    def summary():
        data = generate_data()  # Reuse same reactive expression

        return f"""
        Summary Statistics:
        Mean: {data.mean():.3f}
        Std: {data.std():.3f}
        Min: {data.min():.3f}
        Max: {data.max():.3f}
        """

app = App(app_ui, server)
```

Reactive Values and Effects

python

```
from shiny import reactive

def server(input, output, session):
    # Reactive value - can be modified
    counter = reactive.Value(0)

    # Reactive effect - runs when dependencies change
    @reactive.Effect
    def update_counter():
        # This runs whenever input.button_click() changes
        input.button_click()  # Dependency
        current = counter.get()
        counter.set(current + 1)

    @output
    @render.text
    def counter_text():
        return f"Button clicked {counter()} times"

    # Observe event - run only when specific event occurs
    @reactive.Effect
    @reactive.event(input.reset_button)  # Only when reset button clicked
    def reset_counter():
        counter.set(0)
```

Event Handling

```
python
```

```
app_ui = ui.page_fluid(
    ui.input_action_button("submit", "Submit Data", class_="btn-primary"),
    ui.input_action_button("reset", "Reset", class_="btn-warning"),
    ui.output_text("status"),
    ui.output_table("data_table")
)

def server(input, output, session):
    # Reactive values
    data_processed = reactive.Value(False)
    results = reactive.Value(None)

    # Handle submit button click
    @reactive.Effect
    @reactive.event(input.submit)
    def handle_submit():
        # Simulate data processing
        import pandas as pd
        import time

        # Show processing status
        data_processed.set("processing")

        # Simulate work
        time.sleep(1)

        # Generate sample results
        df = pd.DataFrame({
            'ID': range(1, 11),
            'Value': np.random.randn(10),
            'Category': np.random.choice(['A', 'B', 'C'], 10)
        })

        results.set(df)
        data_processed.set("completed")

    # Handle reset button
    @reactive.Effect
    @reactive.event(input.reset)
    def handle_reset():
        data_processed.set(False)
        results.set(None)
    @output
    @render.text
    def status():
        status = data_processed()
        if status == "processing":
            return "Processing data..."
        elif status == "completed":
            return "Data processing completed!"
        else:
            return "Click Submit to process data"

    @output
    @render.table
    def data_table():
        df = results()
        if df is not None:
            return df
        else:
            return pd.DataFrame()  # Empty table
```

Data Visualization

Matplotlib Integration

```
python
```

```
from shiny import App, ui, render
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

app_ui = ui.page_fluid(
    ui.input_slider("n_points", "Number of points:", min=50, max=500, value=200),
    ui.input_select("plot_type", "Plot type:",
                    choices=["scatter", "line", "histogram", "box"]),
    ui.output_plot("matplotlib_plot")
)

def server(input, output, session):
    @output
    @render.plot
    def matplotlib_plot():
        n = input.n_points()
        plot_type = input.plot_type()

        # Generate sample data
        np.random.seed(42)
        x = np.linspace(0, 4*np.pi, n)
        y = np.sin(x) + np.random.normal(0, 0.3, n)

        fig, ax = plt.subplots(figsize=(10, 6))

        if plot_type == "scatter":
            ax.scatter(x, y, alpha=0.6)
            ax.set_title("Scatter Plot")
        elif plot_type == "line":
            ax.plot(x, y)
            ax.set_title("Line Plot")
        elif plot_type == "histogram":
            ax.hist(y, bins=20, alpha=0.7, edgecolor='black')
            ax.set_title("Histogram")
        elif plot_type == "box":
            # Create grouped data for box plot
            groups = ['A', 'B', 'C']
            data_groups = [y[i::3] for i in range(3)]
            ax.boxplot(data_groups, labels=groups)
            ax.set_title("Box Plot")

        ax.grid(True, alpha=0.3)
        plt.tight_layout()

        return fig

app = App(app_ui, server)
```

Plotly Integration

python

```
from shiny import App, ui, render
import plotly.express as px
import plotly.graph_objects as go
import pandas as pd
import numpy as np

app_ui = ui.page_fluid(
    ui.input_selectize("x_var", "X Variable:",
                      choices=["sepal_length", "sepal_width", "petal_length", "petal_width"]),
    ui.input_selectize("y_var", "Y Variable:",
                      choices=["sepal_length", "sepal_width", "petal_length", "petal_width"]),
    ui.input_checkbox("show_trendline", "Show trendline", value=False),
    ui.output_plot("plotly_scatter")
)

def server(input, output, session):
    @output
    @render.plot
    def plotly_scatter():
        # Load sample data (iris dataset)
        iris = px.data.iris()

        # Create scatter plot
        if input.show_trendline():
            fig = px.scatter(iris, x=input.x_var(), y=input.y_var(),
                           color="species", trendline="ols",
                           title=f"{input.y_var().title()} vs {input.x_var().title()}")
        else:
            fig = px.scatter(iris, x=input.x_var(), y=input.y_var(),
                           color="species",
                           title=f"{input.y_var().title()} vs {input.x_var().title()}")

        fig.update_layout(height=600)

        return fig

app = App(app_ui, server)
```

Seaborn Integration

```
python
```

```
from shiny import App, ui, render
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

app_ui = ui.page_fluid(
    ui.input_select("dataset", "Dataset:",
                    choices=["tips", "flights", "titanic"]),
    ui.input_select("plot_type", "Plot Type:",
                    choices=["correlation", "distribution", "categorical"]),
    ui.output_plot("seaborn_plot")
)

def server(input, output, session):
    @output
    @render.plot
    def seaborn_plot():
        # Load selected dataset
        if input.dataset() == "tips":
            data = sns.load_dataset("tips")
        elif input.dataset() == "flights":
            data = sns.load_dataset("flights")
        else:
            data = sns.load_dataset("titanic")

        fig, ax = plt.subplots(figsize=(12, 8))

        if input.plot_type() == "correlation":
            # Correlation heatmap
            numeric_cols = data.select_dtypes(include=[np.number]).columns
            if len(numeric_cols) > 1:
                corr = data[numeric_cols].corr()
                sns.heatmap(corr, annot=True, cmap='coolwarm', center=0, ax=ax)
                ax.set_title("Correlation Matrix")
            else:
                ax.text(0.5, 0.5, "No numeric columns for correlation",
                        ha='center', va='center', transform=ax.transAxes)

        elif input.plot_type() == "distribution":
            # Distribution plots
            numeric_cols = data.select_dtypes(include=[np.number]).columns
            if len(numeric_cols) > 0:
                # Plot first numeric column
                col = numeric_cols[0]
                sns.histplot(data[col], kde=True, ax=ax)
                ax.set_title(f"Distribution of {col}")

        else:  # categorical
            # Categorical plot
            categorical_cols = data.select_dtypes(include=['object', 'category']).columns
            if len(categorical_cols) > 0:
                col = categorical_cols[0]
                value_counts = data[col].value_counts()
                sns.barplot(x=value_counts.values, y=value_counts.index, ax=ax)
                ax.set_title(f"Distribution of {col}")

        plt.tight_layout()
        return fig

app = App(app_ui, server)
```

Interactive Widgets

Input Widgets Collection

```
python
```

```
app_ui = ui.page_fluid(
    ui.h2("Input Widgets Demo"),

    # Basic inputs
    ui.row(
        ui.column(6,
            ui.card(
                ui.card_header("Text Inputs"),
                ui.input_text("text_input", "Text Input:", value="Hello"),
                ui.input_password("password_input", "Password:"),
                ui.input_text_area("textarea_input", "Text Area:", rows=3),
                ui.input_numeric("numeric_input", "Numeric Input:", value=42)
            )
        ),
        ui.column(6,
            ui.card(
                ui.card_header("Selection Inputs"),
                ui.input_select("select_input", "Select:",
                                choices={"opt1": "Option 1", "opt2": "Option 2"}),
                ui.input_selectize("selectize_input", "Selectize:",
                                   choices=["A", "B", "C"], multiple=True),
                ui.input_radio_buttons("radio_input", "Radio:",
                                       choices={"r1": "Radio 1", "r2": "Radio 2"}),
                ui.input_checkbox_group("checkbox_input", "Checkboxes:",
                                        choices={"c1": "Check 1", "c2": "Check 2"})
            )
        )
    ),

    # Range inputs
    ui.row(
        ui.column(6,
            ui.card(
                ui.card_header("Range Inputs"),
                ui.input_slider("slider_input", "Slider:", min=0, max=100, value=50),
                ui.input_date("date_input", "Date:"),
                ui.input_date_range("date_range_input", "Date Range:"),
                ui.input_file("file_input", "File Upload:", multiple=True)
            )
        ),
        ui.column(6,
            ui.card(
                ui.card_header("Action Inputs"),
                ui.input_action_button("action_button", "Action Button",
                                       class_="btn-primary"),
                ui.input_action_link("action_link", "Action Link"),
                ui.input_checkbox("checkbox_single", "Single Checkbox", value=True),
                ui.input_switch("switch_input", "Switch Input")
            )
        )
    ),

    # Output display
    ui.card(
        ui.card_header("Current Input Values"),
        ui.output_text("input_summary")
    )
)

def server(input, output, session):
    @output
    @render.text
    def input_summary():
        summary = f"""
        Text Input: {input.text_input()}
        Numeric Input: {input.numeric_input()}
        Select Input: {input.select_input()}
        Slider Input: {input.slider_input()}
        Checkbox Single: {input.checkbox_single()}
        Date Input: {input.date_input()}
        """
        return summary

app = App(app_ui, server)
```

Dynamic UI Updates

python

```
from shiny import App, ui, render, reactive

app_ui = ui.page_fluid(
    ui.input_select("dataset_type", "Dataset Type:",
                    choices={"survey": "Survey Data", "economic": "Economic Data", "social": "Social Media"}),
    ui.output_ui("dynamic_inputs"),
    ui.output_plot("dynamic_plot")
)

def server(input, output, session):
    @output
    @render.ui
    def dynamic_inputs():
        dataset_type = input.dataset_type()

        if dataset_type == "survey":
            return ui.div(
                ui.input_slider("sample_size", "Sample Size:", min=100, max=10000, value=1000),
                ui.input_select("age_group", "Age Group:",
                                choices={"all": "All Ages", "young": "18-30", "middle": "31-50", "senior": "50+"})
            )
        elif dataset_type == "economic":
            return ui.div(
                ui.input_date_range("date_range", "Date Range:"),
                ui.input_selectize("indicators", "Economic Indicators:",
                                   choices=["GDP", "Unemployment", "Inflation", "Consumer Confidence"],
                                   multiple=True)
            )
        else:  # social media
            return ui.div(
                ui.input_text("hashtag", "Hashtag:", value="#python"),
                ui.input_numeric("tweet_count", "Number of Tweets:", value=500, min=100, max=5000)
            )

    @output
    @render.plot
    def dynamic_plot():
        dataset_type = input.dataset_type()

        import matplotlib.pyplot as plt
        import numpy as np

        fig, ax = plt.subplots(figsize=(10, 6))
        if dataset_type == "survey" and hasattr(input, 'sample_size'):
            n = input.sample_size()
            data = np.random.normal(50, 15, n)
            ax.hist(data, bins=30, alpha=0.7, edgecolor='black')
            ax.set_title(f"Survey Data Distribution (n={n})")

        elif dataset_type == "economic" and hasattr(input, 'indicators'):
            # Simulate economic data
            dates = pd.date_range('2020-01-01', periods=48, freq='M')
            for indicator in input.indicators():
                values = np.cumsum(np.random.randn(48) * 0.1) + 100
                ax.plot(dates, values, label=indicator, marker='o')
            ax.set_title("Economic Indicators Over Time")
            ax.legend()
            ax.tick_params(axis='x', rotation=45)

        elif dataset_type == "social" and hasattr(input, 'tweet_count'):
            # Simulate social media data
            hours = range(24)
            tweet_counts = np.random.poisson(input.tweet_count() / 24, 24)
            ax.bar(hours, tweet_counts, alpha=0.7)
            ax.set_title(f"Tweet Activity by Hour ({input.hashtag()})")
            ax.set_xlabel("Hour of Day")
            ax.set_ylabel("Tweet Count")

        plt.tight_layout()
        return fig

app = App(app_ui, server)
```

Advanced Features

File Upload and Processing

python

```
from shiny import App, ui, render, reactive
import pandas as pd
import io

app_ui = ui.page_fluid(
    ui.h2("Data Upload and Analysis"),
    ui.input_file("upload", "Choose CSV File:", accept=[".csv"], multiple=False),
    ui.br(),

    # Conditional UI - only show if file uploaded
    ui.panel_conditional(
        "output.file_uploaded",
        ui.row(
            ui.column(4,
                ui.card(
                    ui.card_header("Data Controls"),
                    ui.output_ui("column_selector"),
                    ui.input_numeric("n_rows", "Rows to display:", value=10, min=5, max=100)
                )
            ),
            ui.column(8,
                ui.card(
                    ui.card_header("Data Preview"),
                    ui.output_table("data_preview")
                )
            )
        ),
        ui.row(
            ui.column(6,
                ui.card(
                    ui.card_header("Summary Statistics"),
                    ui.output_table("summary_stats")
                )
            ),
            ui.column(6,
                ui.card(
                    ui.card_header("Data Visualization"),
                    ui.output_plot("data_plot")
                )
            )
        )
    )
)
```

```
def server(input, output, session):
    # Reactive value to store uploaded data
    uploaded_data = reactive.Value(None)

    # Process uploaded file
    @reactive.Effect
    def process_upload():
        file_info = input.upload()
        if file_info is None:
            uploaded_data.set(None)
            return

        # Read the uploaded file
        file_path = file_info[0]["datapath"]
        try:
            df = pd.read_csv(file_path)
            uploaded_data.set(df)
        except Exception as e:
            uploaded_data.set(None)
            print(f"Error reading file: {e}")

    @output
    @render.text
    def file_uploaded():
        return str(uploaded_data() is not None).lower()

    @output
    @render.ui
    def column_selector():
        df = uploaded_data()
        if df is None:
            return ui.div()

        numeric_cols = df.select_dtypes(include=['number']).columns.tolist()

        return ui.div(
            ui.input_select("x_column", "X Column:", choices=numeric_cols),
            ui.input_select("y_column", "Y Column:", choices=numeric_cols),
            ui.input_select("plot_type", "Plot Type:",
                           choices={"scatter": "Scatter", "line": "Line", "hist": "Histogram"})
        )

    @output
```

```
@render.table
def data_preview():
    df = uploaded_data()
    if df is not None:
        return df.head(input.n_rows())
    return pd.DataFrame()

@output
@render.table
def summary_stats():
    df = uploaded_data()
    if df is not None:
        return df.describe()
    return pd.DataFrame()

@output
@render.plot
def data_plot():
    df = uploaded_data()
    if df is None or not hasattr(input, 'x_column'):
        return None

    import matplotlib.pyplot as plt

    fig, ax = plt.subplots(figsize=(10, 6))

    plot_type = input.plot_type()
    x_col = input.x_column()
    y_col = input.y_column()

    if plot_type == "scatter":
        ax.scatter(df[x_col], df[y_col], alpha=0.6)
        ax.set_xlabel(x_col)
        ax.set_ylabel(y_col)
        ax.set_title(f"Scatter Plot: {y_col} vs {x_col}")
    elif plot_type == "line":
        ax.plot(df[x_col], df[y_col])
        ax.set_xlabel(x_col)
        ax.set_ylabel(y_col)
        ax.set_title(f"Line Plot: {y_col} vs {x_col}")
    else:  # histogram
        ax.hist(df[x_col], bins=30, alpha=0.7, edgecolor='black')
        ax.set_xlabel(x_col)
        ax.set_ylabel("Frequency")
```

```
            ax.set_title(f"Histogram of {x_col}")

        plt.tight_layout()
        return fig

app = App(app_ui, server)
```

Progress Indicators and Async Operations

```python
```

```
from shiny import App, ui, render, reactive
import asyncio
import time

app_ui = ui.page_fluid(
    ui.h2("Long-Running Operations with Progress"),

    ui.input_numeric("n_simulations", "Number of Simulations:", value=10, min=1, max=100),
    ui.input_action_button("start_analysis", "Start Analysis", class_="btn-primary"),
    ui.br(), ui.br(),

    ui.output_ui("progress_ui"),
    ui.output_text("status_text"),
    ui.output_plot("results_plot")
)

def server(input, output, session):
    # Reactive values for tracking progress
    analysis_running = reactive.Value(False)
    current_progress = reactive.Value(0)
    results_data = reactive.Value(None)

    @reactive.Effect
    @reactive.event(input.start_analysis)
    def start_long_analysis():
        analysis_running.set(True)
        current_progress.set(0)
        results_data.set(None)

        # Run analysis in background
        asyncio.create_task(run_analysis())

    async def run_analysis():
        """Simulate long-running analysis with progress updates."""
        n_sims = input.n_simulations()
        results = []

        for i in range(n_sims):
            # Simulate work
            await asyncio.sleep(0.5)  # Non-blocking sleep

            # Generate some results
            import numpy as np
            result = {
                'simulation': i + 1,
                'value': np.random.normal(0, 1),
                'category': np.random.choice(['A', 'B', 'C'])
            }
            results.append(result)

            # Update progress
            progress = ((i + 1) / n_sims) * 100
            current_progress.set(progress)

        # Analysis complete
        results_data.set(pd.DataFrame(results))
        analysis_running.set(False)

    @output
    @render.ui
    def progress_ui():
        if analysis_running():
            progress = current_progress()
            return ui.div(
                ui.h4("Analysis in Progress..."),
                ui.tags.div(
                    ui.tags.div(
                        f"{progress:.1f}%",
                        style=f"width: {progress}%; background-color: #007bff; color: white; text-align: center; line-height:
                        class_="progress-bar"
                    ),
                    class_="progress",
                    style="height: 30px; background-color: #e9ecef;"
                )
            )
        return ui.div()

    @output
    @render.text
    def status_text():
        if analysis_running():
            return f"Running simulation {int(current_progress() * input.n_simulations() / 100)}/{input.n_simulations()}"
        elif results_data() is not None:
            return "Analysis completed successfully!"
        else:
            return "Click 'Start Analysis' to begin"
    @output
    @render.plot
    def results_plot():
        df = results_data()
        if df is None:
            return None

        import matplotlib.pyplot as plt

        fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 5))

        # Plot 1: Distribution of values
        ax1.hist(df['value'], bins=15, alpha=0.7, edgecolor='black')
        ax1.set_title("Distribution of Simulation Results")
        ax1.set_xlabel("Value")
        ax1.set_ylabel("Frequency")

        # Plot 2: Category counts
        category_counts = df['category'].value_counts()
        ax2.bar(category_counts.index, category_counts.values, alpha=0.7)
        ax2.set_title("Results by Category")
        ax2.set_xlabel("Category")
        ax2.set_ylabel("Count")

        plt.tight_layout()
        return fig

app = App(app_ui, server)
```

Modal Dialogs and Notifications

python

```python
from shiny import App, ui, render, reactive

app_ui = ui.page_fluid(
    ui.h2("Modal Dialogs and Notifications"),

    ui.row(
        ui.column(4,
            ui.card(
                ui.card_header("Notifications"),
                ui.input_action_button("show_success", "Success Message", class_="btn-success"),
                ui.br(), ui.br(),
                ui.input_action_button("show_warning", "Warning Message", class_="btn-warning"),
                ui.br(), ui.br(),
                ui.input_action_button("show_error", "Error Message", class_="btn-danger")
            )
        ),
        ui.column(4,
            ui.card(
                ui.card_header("Modal Dialogs"),
                ui.input_action_button("show_info_modal", "Info Modal", class_="btn-info"),
                ui.br(), ui.br(),
                ui.input_action_button("show_confirm_modal", "Confirm Modal", class_="btn-primary")
            )
        ),
        ui.column(4,
            ui.card(
                ui.card_header("Results"),
                ui.output_text("modal_result"),
                ui.output_text("notification_log")
            )
        )
    )
)

def server(input, output, session):
    # Track modal results and notifications
    modal_response = reactive.Value("")
    notification_count = reactive.Value(0)

    # Success notification
    @reactive.Effect
    @reactive.event(input.show_success)
    def show_success_notification():
        ui.notification_show("Operation completed successfully!", type="success", duration=3)
        notification_count.set(notification_count() + 1)

    # Warning notification
    @reactive.Effect
    @reactive.event(input.show_warning)
    def show_warning_notification():
        ui.notification_show("This is a warning message.", type="warning", duration=5)
        notification_count.set(notification_count() + 1)

    # Error notification
    @reactive.Effect
    @reactive.event(input.show_error)
    def show_error_notification():
        ui.notification_show("An error occurred!", type="error", duration=7)
        notification_count.set(notification_count() + 1)

    # Info modal
    @reactive.Effect
    @reactive.event(input.show_info_modal)
    def show_info_modal():
        ui.modal_show(
            ui.modal(
                ui.h3("Information"),
                ui.p("This is an informational modal dialog."),
                ui.p("It provides additional details about the application or current operation."),
                ui.tags.ul(
                    ui.tags.li("Feature 1: Data visualization"),
                    ui.tags.li("Feature 2: Interactive analysis"),
                    ui.tags.li("Feature 3: Report generation")
                ),
                title="App Information",
                easy_close=True,
                footer=ui.modal_button("Close", class_="btn-secondary")
            )
        )

    # Confirmation modal
    @reactive.Effect
    @reactive.event(input.show_confirm_modal)
    def show_confirm_modal():
        ui.modal_show(
            ui.modal(
                ui.h3("Confirm Action"),
                ui.p("Are you sure you want to delete all data?"),
                ui.p("This action cannot be undone."),
                title="Confirm Deletion",
                easy_close=False,
                footer=[
                    ui.input_action_button("confirm_yes", "Yes, Delete", class_="btn-danger"),
                    ui.input_action_button("confirm_no", "Cancel", class_="btn-secondary")
                ]
            )
        )

    # Handle confirmation responses
    @reactive.Effect
    @reactive.event(input.confirm_yes)
    def handle_confirm_yes():
        modal_response.set("User confirmed deletion")
        ui.modal_remove()
        ui.notification_show("Data deleted successfully!", type="success")

    @reactive.Effect
    @reactive.event(input.confirm_no)
    def handle_confirm_no():
        modal_response.set("User cancelled deletion")
        ui.modal_remove()

    @output
    @render.text
    def modal_result():
        return f"Last modal response: {modal_response()}" if modal_response() else "No modal responses yet"

    @output
    @render.text
    def notification_log():
        count = notification_count()
        return f"Notifications shown: {count}" if count > 0 else "No notifications shown yet"

app = App(app_ui, server)
```

Social Science Applications

Survey Data Analysis Dashboard

python

```
from shiny import App, ui, render, reactive
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats

# Generate sample survey data
def generate_survey_data(n=1000):
    np.random.seed(42)

    data = {
        'respondent_id': range(1, n+1),
        'age': np.random.normal(40, 15, n).clip(18, 80).astype(int),
        'income': np.random.lognormal(10.8, 0.8, n).clip(20000, 300000).astype(int),
        'education': np.random.choice(['High School', 'Some College', 'Bachelor', 'Master', 'PhD'],
                                     n, p=[0.25, 0.2, 0.3, 0.2, 0.05]),
        'region': np.random.choice(['Northeast', 'Southeast', 'Midwest', 'West'], n),
        'political_affiliation': np.random.choice(['Democrat', 'Republican', 'Independent'],
                                                  n, p=[0.4, 0.35, 0.25]),
        'life_satisfaction': np.random.normal(7, 2, n).clip(1, 10),
        'trust_government': np.random.normal(5, 2, n).clip(1, 10),
        'social_media_hours': np.random.gamma(2, 2, n).clip(0, 12)
    }

    # Add some correlations
    df = pd.DataFrame(data)

    # Education affects income
    education_multiplier = {'High School': 0.8, 'Some College': 0.9, 'Bachelor': 1.1,
                            'Master': 1.3, 'PhD': 1.6}
    df['income'] *= df['education'].map(education_multiplier)
    df['income'] = df['income'].astype(int)

    # Age affects trust in government
    df['trust_government'] += (df['age'] - 40) * 0.02
    df['trust_government'] = df['trust_government'].clip(1, 10)

    return df

app_ui = ui.page_navbar(
    ui.nav("📊 Data Overview",
        ui.row(
        ui.column(4,
            ui.card(
                ui.card_header("Dataset Controls"),
                ui.input_numeric("sample_size", "Sample Size:", value=1000, min=100, max=5000, step=100),
                ui.input_action_button("regenerate_data", "Regenerate Data", class_="btn-primary"),
                ui.br(), ui.br(),
                ui.h5("Dataset Summary"),
                ui.output_text("dataset_summary")
            )
        ),
        ui.column(8,
            ui.card(
                ui.card_header("Sample Data"),
                ui.output_table("sample_data")
            )
        )
    )
),

ui.nav("📈 Demographic Analysis",
    ui.row(
        ui.column(3,
            ui.card(
                ui.card_header("Analysis Controls"),
                ui.input_select("demographic_var", "Demographic Variable:",
                               choices=["age", "income", "education", "region", "political_affiliation"]),
                ui.input_select("outcome_var", "Outcome Variable:",
                               choices=["life_satisfaction", "trust_government", "social_media_hours"]),
                ui.input_checkbox("show_statistics", "Show Statistical Tests", value=True)
            )
        ),
        ui.column(9,
            ui.card(
                ui.card_header("Demographic Analysis"),
                ui.output_plot("demographic_plot"),
                ui.output_text("statistical_results")
            )
        )
    )
),

ui.nav("🔍 Correlation Analysis",
    ui.card(
        ui.card_header("Correlation Matrix"),
            ui.output_plot("correlation_matrix")
        ),
        ui.row(
            ui.column(6,
                ui.card(
                    ui.card_header("Variable Relationships"),
                    ui.input_select("corr_x", "X Variable:", choices=[]),
                    ui.input_select("corr_y", "Y Variable:", choices=[]),
                    ui.output_plot("scatter_plot")
                )
            ),
            ui.column(6,
                ui.card(
                    ui.card_header("Regression Analysis"),
                    ui.output_text("regression_results")
                )
            )
        )
    ),

    ui.nav("📋 Report",
        ui.card(
            ui.card_header("Analysis Report"),
            ui.input_action_button("generate_report", "Generate Report", class_="btn-success"),
            ui.br(), ui.br(),
            ui.output_ui("report_content")
        )
    ),

    title="Social Science Survey Analysis",
    id="main_navbar"
)

def server(input, output, session):
    # Reactive data
    survey_data = reactive.Value(generate_survey_data())

    # Regenerate data when button clicked
    @reactive.Effect
    @reactive.event(input.regenerate_data)
    def regenerate_data():
        new_data = generate_survey_data(input.sample_size())
        survey_data.set(new_data)
    # Update choices for correlation analysis
    numeric_vars = new_data.select_dtypes(include=[np.number]).columns.tolist()
    numeric_vars = [col for col in numeric_vars if col != 'respondent_id']

    ui.update_select("corr_x", choices=numeric_vars, selected=numeric_vars[0] if numeric_vars else None)
    ui.update_select("corr_y", choices=numeric_vars, selected=numeric_vars[1] if len(numeric_vars) > 1 else No

# Initialize correlation variable choices
@reactive.Effect
def initialize_correlation_vars():
    df = survey_data()
    numeric_vars = df.select_dtypes(include=[np.number]).columns.tolist()
    numeric_vars = [col for col in numeric_vars if col != 'respondent_id']

    ui.update_select("corr_x", choices=numeric_vars, selected=numeric_vars[0] if numeric_vars else None)
    ui.update_select("corr_y", choices=numeric_vars, selected=numeric_vars[1] if len(numeric_vars) > 1 else No

@output
@render.text
def dataset_summary():
    df = survey_data()
    return f"""
    Observations: {len(df):,}
    Variables: {len(df.columns)}
    Missing Values: {df.isnull().sum().sum()}
    Date Generated: {pd.Timestamp.now().strftime('%Y-%m-%d %H:%M')}
    """

@output
@render.table
def sample_data():
    return survey_data().head(10)

@output
@render.plot
def demographic_plot():
    df = survey_data()
    demo_var = input.demographic_var()
    outcome_var = input.outcome_var()

    fig, ax = plt.subplots(figsize=(12, 6))

    if df[demo_var].dtype in ['object', 'category']:
        # Categorical demographic variable
        if df[outcome_var].dtype in ['int64', 'float64']:
            # Box plot for categorical x numeric
            df.boxplot(column=outcome_var, by=demo_var, ax=ax)
            ax.set_title(f'{outcome_var.title()} by {demo_var.title()}')
            plt.suptitle('')  # Remove default title
        else:
            # Cross-tabulation for categorical x categorical
            crosstab = pd.crosstab(df[demo_var], df[outcome_var])
            crosstab.plot(kind='bar', ax=ax)
            ax.set_title(f'{outcome_var.title()} by {demo_var.title()}')
            ax.legend(title=outcome_var.title())
    else:
        # Numeric demographic variable
        if df[outcome_var].dtype in ['int64', 'float64']:
            # Scatter plot for numeric x numeric
            ax.scatter(df[demo_var], df[outcome_var], alpha=0.6)

            # Add trend line
            z = np.polyfit(df[demo_var], df[outcome_var], 1)
            p = np.poly1d(z)
            ax.plot(df[demo_var].sort_values(), p(df[demo_var].sort_values()), "r--", alpha=0.8)

            ax.set_xlabel(demo_var.title())
            ax.set_ylabel(outcome_var.title())
            ax.set_title(f'{outcome_var.title()} vs {demo_var.title()}')

    plt.xticks(rotation=45)
    plt.tight_layout()
    return fig

@output
@render.text
def statistical_results():
    if not input.show_statistics():
        return ""

    df = survey_data()
    demo_var = input.demographic_var()
    outcome_var = input.outcome_var()

    try:
        if df[demo_var].dtype in ['object', 'category']:
            # ANOVA for categorical predictor
            groups = [group[outcome_var].values for name, group in df.groupby(demo_var)]
```

```
                f_stat, p_value = stats.f_oneway(*groups)

                result = f"""
                Statistical Test: One-way ANOVA
                F-statistic: {f_stat:.4f}
                p-value: {p_value:.6f}
                Significance: {'Significant' if p_value < 0.05 else 'Not significant'} (α = 0.05)

                Interpretation: {'There are significant differences between groups' if p_value < 0.05 else 'No significant d
                """
            else:
                # Correlation for numeric predictor
                corr, p_value = stats.pearsonr(df[demo_var], df[outcome_var])

                result = f"""
                Statistical Test: Pearson Correlation
                Correlation coefficient: {corr:.4f}
                p-value: {p_value:.6f}
                Significance: {'Significant' if p_value < 0.05 else 'Not significant'} (α = 0.05)

                Interpretation: {'Significant' if p_value < 0.05 else 'No significant'} {'positive' if corr > 0 else 'negative'} c
                Effect size: {'Strong' if abs(corr) > 0.5 else 'Moderate' if abs(corr) > 0.3 else 'Weak'} relationship
                """

            return result

        except Exception as e:
            return f"Error in statistical analysis: {str(e)}"

    @output
    @render.plot
    def correlation_matrix():
        df = survey_data()
        numeric_cols = df.select_dtypes(include=[np.number]).columns
        numeric_cols = [col for col in numeric_cols if col != 'respondent_id']

        corr_matrix = df[numeric_cols].corr()

        fig, ax = plt.subplots(figsize=(10, 8))
        sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', center=0,
                   square=True, fmt='.2f', ax=ax)
        ax.set_title('Correlation Matrix of Numeric Variables')

        plt.tight_layout()
        return fig

    @output
    @render.plot
    def scatter_plot():
        df = survey_data()

        if not hasattr(input, 'corr_x') or not hasattr(input, 'corr_y'):
            return None

        x_var = input.corr_x()
        y_var = input.corr_y()

        if not x_var or not y_var:
            return None

        fig, ax = plt.subplots(figsize=(10, 6))

        ax.scatter(df[x_var], df[y_var], alpha=0.6)

        # Add regression line
        z = np.polyfit(df[x_var], df[y_var], 1)
        p = np.poly1d(z)
        ax.plot(df[x_var].sort_values(), p(df[x_var].sort_values()), "r--", alpha=0.8)

        ax.set_xlabel(x_var.title())
        ax.set_ylabel(y_var.title())
        ax.set_title(f'{y_var.title()} vs {x_var.title()}')
        ax.grid(True, alpha=0.3)

        plt.tight_layout()
        return fig

    @output
    @render.text
    def regression_results():
        df = survey_data()

        if not hasattr(input, 'corr_x') or not hasattr(input, 'corr_y'):
            return "Select variables for analysis"

        x_var = input.corr_x()
        y_var = input.corr_y()
        if not x_var or not y_var:
            return "Select variables for analysis"

        try:
            from sklearn.linear_model import LinearRegression
            from sklearn.metrics import r2_score

            X = df[[x_var]].values
            y = df[y_var].values

            model = LinearRegression()
            model.fit(X, y)
            y_pred = model.predict(X)

            r2 = r2_score(y, y_pred)

            # Statistical significance
            corr, p_value = stats.pearsonr(df[x_var], df[y_var])

            result = f"""
            Linear Regression Results:

            Equation: {y_var} = {model.intercept_:.3f} + {model.coef_[0]:.3f} * {x_var}
            R-squared: {r2:.4f}
            Correlation: {corr:.4f}
            p-value: {p_value:.6f}

            Interpretation:
            - {r2*100:.1f}% of variance in {y_var} is explained by {x_var}
            - For each unit increase in {x_var}, {y_var} {'increases' if model.coef_[0] > 0 else 'decreases'} by {abs(mod
            - Relationship is {'statistically significant' if p_value < 0.05 else 'not statistically significant'}
            """

            return result

        except Exception as e:
            return f"Error in regression analysis: {str(e)}"

    # Report generation
    @reactive.Value
    def report_generated():
        return False

    @reactive.Effect
    @reactive.event(input.generate_report)
    def generate_analysis_report():
        report_generated.set(True)

    @output
    @render.ui
    def report_content():
        if not report_generated():
            return ui.div("Click 'Generate Report' to create analysis summary")

        df = survey_data()

        # Calculate key statistics
        numeric_vars = df.select_dtypes(include=[np.number]).columns
        numeric_vars = [col for col in numeric_vars if col != 'respondent_id']

        # Strongest correlations
        corr_matrix = df[numeric_vars].corr()
        correlations = []

        for i in range(len(corr_matrix.columns)):
            for j in range(i+1, len(corr_matrix.columns)):
                var1 = corr_matrix.columns[i]
                var2 = corr_matrix.columns[j]
                corr_val = corr_matrix.iloc[i, j]
                correlations.append((abs(corr_val), var1, var2, corr_val))

        correlations.sort(reverse=True)
        top_correlations = correlations[:3]

        report_html = f"""
        <div class="report-content">
            <h3>Survey Analysis Report</h3>
            <p><strong>Generated:</strong> {pd.Timestamp.now().strftime('%B %d, %Y at %I:%M %p')}</p>

            <h4>Dataset Overview</h4>
            <ul>
                <li>Sample size: {len(df):,} respondents</li>
                <li>Variables: {len(df.columns)} total</li>
                <li>Missing values: {df.isnull().sum().sum()}</li>
            </ul>

            <h4>Key Demographics</h4>
            <ul>
        <li>Average age: {df['age'].mean():.1f} years (SD = {df['age'].std():.1f})</li>
        <li>Median income: ${df['income'].median():,}</li>
        <li>Average life satisfaction: {df['life_satisfaction'].mean():.1f}/10</li>
        <li>Average trust in government: {df['trust_government'].mean():.1f}/10</li>
    </ul>

    <h4>Education Distribution</h4>
    <ul>
"""

for education, count in df['education'].value_counts().items():
    pct = (count / len(df)) * 100
    report_html += f"<li>{education}: {count} ({pct:.1f}%)</li>"

report_html += "</ul><h4>Strongest Correlations</h4><ul>"

for _, var1, var2, corr_val in top_correlations:
    direction = "positive" if corr_val > 0 else "negative"
    strength = "strong" if abs(corr_val) > 0.5 else "moderate" if abs(corr_val) > 0.3 else "weak"
    report_html += f"<li>{var1.replace('_', ' ').title()} & {var2.replace('_', ' ').title()}: {strength} {direction} correla

report_html += """
    </ul>

    <h4>Key Findings</h4>
    <ul>
        <li>Survey data shows typical patterns expected in social science research</li>
        <li>Education levels show expected correlation with income levels</li>
        <li>Age demographics suggest a representative adult sample</li>
        <li>Life satisfaction and trust metrics provide insights into social attitudes</li>
    </ul>

    <h4>Recommendations</h4>
    <ul>
        <li>Consider stratified analysis by demographic groups</li>
        <li>Investigate regional differences in key outcomes</li>
        <li>Examine potential mediating variables in strong correlations</li>
        <li>Consider longitudinal follow-up studies</li>
    </ul>
</div>

<style>
.report-content {
    font-family: Arial, sans-serif;
```

```
            line-height: 1.6;
            max-width: 800px;
        }
        .report-content h3 {
            color: #2c3e50;
            border-bottom: 2px solid #3498db;
            padding-bottom: 5px;
        }
        .report-content h4 {
            color: #34495e;
            margin-top: 20px;
        }
        .report-content ul {
            padding-left: 20px;
        }
        .report-content li {
            margin-bottom: 5px;
        }
        </style>
        """

        return ui.HTML(report_html)

app = App(app_ui, server)
```

Text Analysis Dashboard

python

```python
from shiny import App, ui, render, reactive
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from collections import Counter
import re

# Sample social science text data
def generate_text_data():
    """Generate sample text data for analysis."""

    topics = {
        'education': [
            "The education system needs significant reform to better prepare students for the future.",
            "Funding for public schools has been declining over the past decade.",
            "Online learning has transformed the educational landscape permanently.",
            "Teachers deserve better compensation for their critical work.",
            "Student loan debt is creating barriers to higher education access."
        ],
        'healthcare': [
            "Healthcare costs continue to rise faster than inflation rates.",
            "Mental health services need to be more accessible and affordable.",
            "The pandemic highlighted weaknesses in our healthcare infrastructure.",
            "Preventive care can reduce long-term healthcare expenses significantly.",
            "Healthcare workers face burnout at unprecedented levels."
        ],
        'economy': [
            "Income inequality has reached levels not seen since the 1920s.",
            "The gig economy offers flexibility but lacks traditional benefits.",
            "Automation threatens many traditional manufacturing jobs.",
            "Small businesses struggle to compete with large corporations.",
            "Economic recovery varies significantly across different regions."
        ]
    }

    data = []
    for topic, texts in topics.items():
        for i, text in enumerate(texts):
            data.append({
                'id': len(data) + 1,
                'topic': topic,
                'text': text,
                'length': len(text),
                'word_count': len(text.split()),
                'sentiment_score': np.random.uniform(-0.5, 0.5)  # Simplified sentiment
            })

    return pd.DataFrame(data)

app_ui = ui.page_navbar(
    ui.nav("📝 Text Data",
        ui.row(
            ui.column(4,
                ui.card(
                    ui.card_header("Data Controls"),
                    ui.input_file("text_upload", "Upload Text File (.csv, .txt):",
                                accept=[".csv", ".txt"]),
                    ui.br(),
                    ui.input_action_button("use_sample", "Use Sample Data", class_="btn-primary"),
                    ui.br(), ui.br(),
                    ui.output_text("data_summary")
                )
            ),
            ui.column(8,
                ui.card(
                    ui.card_header("Text Preview"),
                    ui.output_table("text_preview")
                )
            )
        )
    ),

    ui.nav("📊 Text Statistics",
        ui.row(
            ui.column(6,
                ui.card(
                    ui.card_header("Length Distribution"),
                    ui.output_plot("length_distribution")
                )
            ),
            ui.column(6,
                ui.card(
                    ui.card_header("Word Frequency"),
                    ui.input_numeric("top_words", "Top N words:", value=20, min=5, max=50),
                    ui.output_plot("word_frequency")
                )
            )
        ),
        ui.row(
            ui.column(12,
                ui.card(
                    ui.card_header("Text Statistics by Category"),
                    ui.output_plot("category_analysis")
                )
            )
        )
    ),

    ui.nav("🔍 Word Analysis",
        ui.row(
            ui.column(4,
                ui.card(
                    ui.card_header("Search Controls"),
                    ui.input_text("search_word", "Search for word:", value="education"),
                    ui.input_select("analysis_type", "Analysis Type:",
                                   choices={"frequency": "Word Frequency",
                                          "context": "Word Context",
                                          "cooccurrence": "Co-occurrence"})
                )
            ),
            ui.column(8,
                ui.card(
                    ui.card_header("Word Analysis Results"),
                    ui.output_plot("word_analysis"),
                    ui.output_text("word_context")
                )
            )
        )
    ),

    title="Text Analysis Dashboard"
)

def server(input, output, session):
    # Reactive data storage
    text_data = reactive.Value(generate_text_data())

    # Use sample data
    @reactive.Effect
    @reactive.event(input.use_sample)
    def load_sample_data():
```

```
    text_data.set(generate_text_data())

# Handle file upload
@reactive.Effect
```