Week 10 – Aggregating Enterprise Risks

We can. . .

1. Experiment with different degrees of freedom to sensitive ourselves to the random numbers generated.
2. Parameterize correlations. This means assign correlations to a variable and place that variable into the `sigma` matrix. This might get into trouble with an error. It would mean we would have to reassign the correlation. The mathematical problem is finding a **positive definite** variance-covariance matrix.
3. How different are the value at risk and expected shortfall measures between the use of the Gaussian (normal) copula and the t-copula? Why should a decision maker care?

All of that experimentation begs for an interactive decision tool.

Let's build an app . . .

The application (the "app") will be housed in an R script that contain four architectural layers.

Four architectural layers

1. Analytics
2. User Interface (UI)
3. Server
4. Application generator

Analytics

1. Libraries used in app processes
2. Function that wraps analytical script
3. Inputs from UI layer to server layer
4. Outputs from server layer to UI layer

UI

1. Slide bars for user to input range of parameters
2. Plots to display results
3. Text to report results

Server

1. Run analytics with inputs from the UI and from a simulation function
2. Generate outputs for UI

Application generator

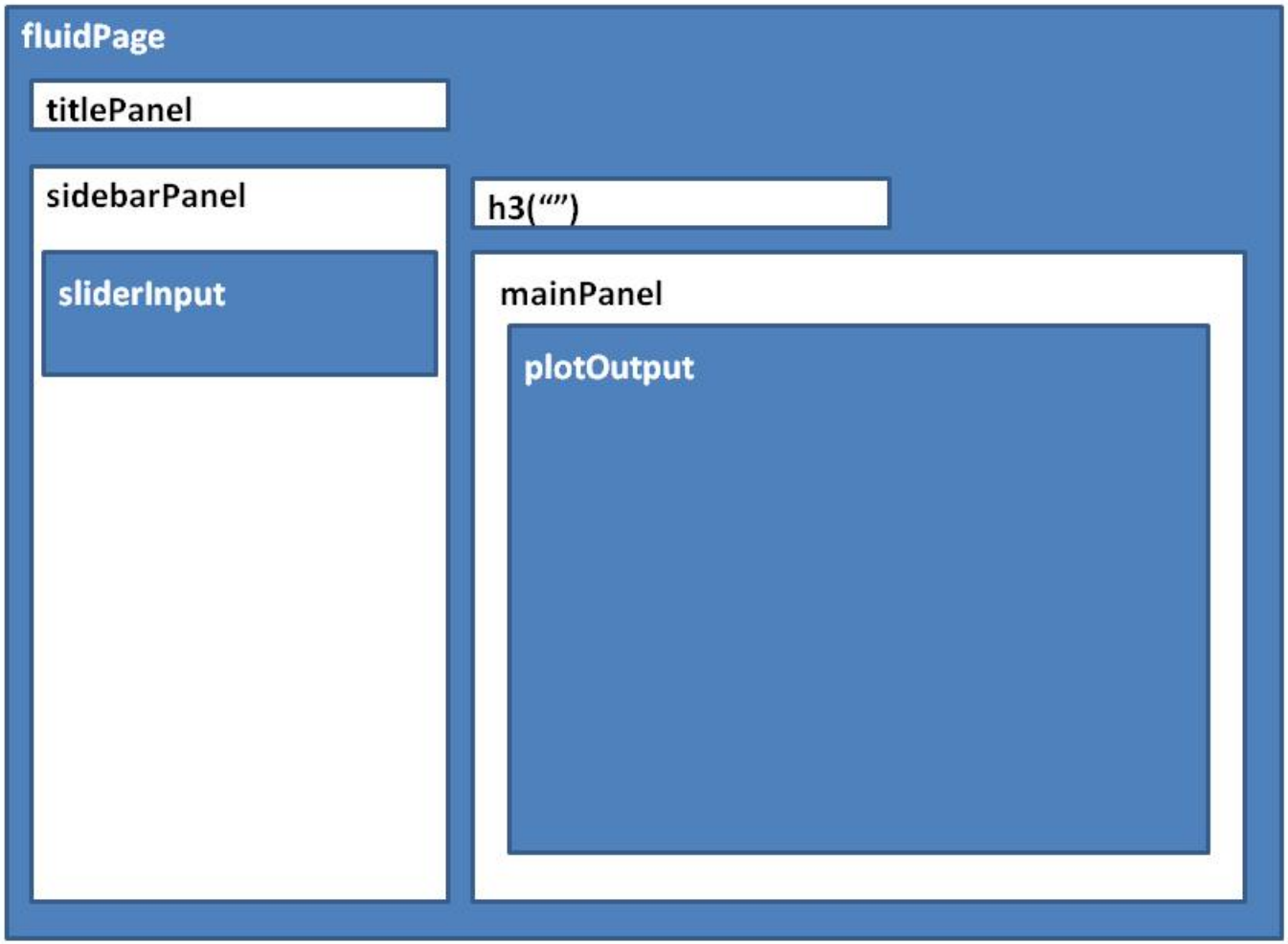
- Run application function with UI and Server inputs

The simulation function

```
library(shiny)
require(mvtnorm)
require(psych)
risk.sim <- function(input) {
    # Begin enterprise risk simulation
    set.seed(1016)  # Freezes the random seed to reproduce results exactly
    n.risks <- 3  # Number of risk factors
    m <- n.risks
    n.sim <- 1000  # pull slider settings into the sigma correlation matrix
    sigma <- matrix(c(1, input[1], input[2], input[1], 1, input[3], input[2],
        input[3], 1), nrow = m)
    z <- rmvt(n.sim, delta = rep(0, nrow(sigma)), sigma = sigma, df = 6, type = "shifted")
    u <- pt(z, df = 6)
    x1 <- qgamma(u[, 1], shape = 2, scale = 1)
    x2 <- qbeta(u[, 2], 2, 2)
    x3 <- qt(u[, 3], df = 6)
    factors.df <- cbind(x1/10, x2, x3/10)
    colnames(factors.df) <- c("Revenue", "Variable Cost", "Fixed Cost")
    revenue <- 1000 * (1 + factors.df[, 1])
    variable.cost <- revenue * factors.df[, 2]
    fixed.cost <- revenue * factors.df[, 3]
    total.cost <- variable.cost + fixed.cost
    operating.margin <- revenue - variable.cost - fixed.cost
    analysis.t <- cbind(revenue, total.cost, operating.margin)
    colnames(analysis.t) <- c("Revenue", "Cost", "Margin")
    return(analysis.t)
}
```

The UI

Here is a mock-up of the screen we will implement in Shiny.



UI Design

Here is what the Shiny UI code looks like:

```
ui <- fluidPage(titlePanel("Enterprise Risk Analytics"), sidebarLayout(sidebarPanel(sliderInput(inputId = "cor.1",
    label = "Set the Revenue - Variable Cost Correlation", value = 0.5, min = 0.1,
    max = 0.9), sliderInput(inputId = "cor.2", label = "Set the Revenue - Variable Cost Correlation",
    value = 0.5, min = 0.1, max = 0.9), sliderInput(inputId = "cor.3", label = "Set the Variable - Fixed Cost Correlation",
    value = 0.5, min = 0.1, max = 0.9)), mainPanel(plotOutput("pairs.1"))))
```

The server

- The Shiny server is a function
- The function gets inputs from the UI
- Generates outputs that are sent back to the UI

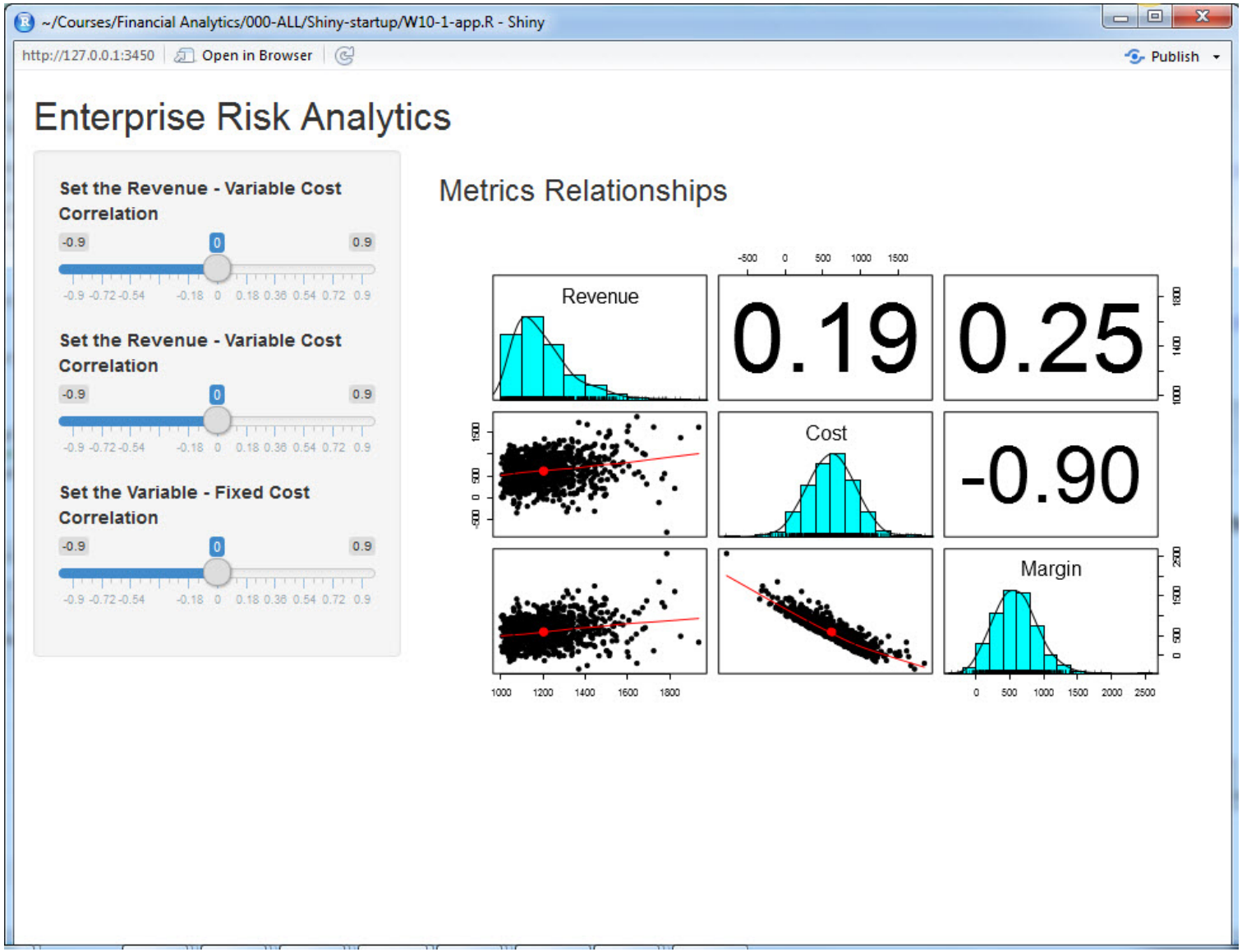
```
server <- function(input, output) {
    output$pairs.1 <- renderPlot({
        analysis.t <- risk.sim(c(input$cor.1, input$cor.2, input$cor.3))
        pairs.panels(analysis.t)
    })
}
```

Run the app

This function call the Shiny application process with inputs `ui` and `server`.

```
shinyApp(ui = ui, server = server)
```

Here is what you see when you run the app in the script window of Rstudio.



ERM Application Screenshot

What else could we do?

- Build tabs for various components of the analysis
- Use tables to summarize metrics (e.g., VaR, ES)
- Whatever else the consumer of this analysis would need

Whitman
SCHOOL of MANAGEMENT

SYRACUSE UNIVERSITY

MBA@SYRACUSE