HW 6 Random Number Generators & Rshiny

STAT 5400

Due: Oct 11, 2024 9:30 AM

Problems

Submit your solutions as an .Rmd file and accompanying .pdf file. Include all the **relevant** R code and output. Always interpret your result whenever it is necessary.

Problems

1. Generators of normal distributions.

- Write an R function `BoxMuller` that generates a sample of n values from $N(\mu, \sigma^2)$ using the Box-Muller method. This function should have three arguments:
  - `n`: number of observations,
  - `mu`: the value of mean μ,
  - `sigma`: the value of standard deviation σ. This function should return a vector of n normal variates.
- Write an R function `PolarMethod` that generates a sample of n values from $N(\mu, \sigma^2)$ using the polar method. This function should have the same arguments and return values as the `BoxMuller` function.

```
BoxMuller <- function(n, mu, sigma) {
  if (n %% 2 != 0) {
    stop("n must be an even number.")
  }

  u1 <- runif(n / 2)
  u2 <- runif(n / 2)

  #box muller
  z1 <- sqrt(-2 * log(u1)) * cos(2 * pi * u2)
  z2 <- sqrt(-2 * log(u1)) * sin(2 * pi * u2)


  x <- mu +( sigma * c(z1, z2))

  return(x)
}
```

When the polar method is used, it might be tricky if you want to control `n` when the accept-reject sampling method is used to generate points in the unit disk. You need a different implementation from the one on the slides.

```
PolarMethod <- function(n, mu, sigma) {
z <- numeric(n)
i <- 1
while (i <= n) {
  u1 <- runif(1, -1, 1)
  u2 <- runif(1, -1, 1)
  r <- u1 ^ 2 + u2 ^ 2
  if (r < 1) {
    z[i] <- mu + sigma * u1 * sqrt(-2 * log(r) / r)
    i <- i + 1

    }
  }
return(z)
}
```

- Generate two vectors of 30 standard normal variates, by both `BoxMuller` and `PolarMethod`. Run a test to check if the two vectors are from the same distribution. One solution is to use `ks.test` in R to perform a Kolmogorov-Smirnov test.

```
# Generate two vectors of 30 standard normal variates
set.seed(123)
n <- 30
mu <- 0
sigma <- 1

boxmuller_sample <- BoxMuller(n, mu, sigma)
polarmethod_sample <- PolarMethod(n, mu, sigma)
ks.test(boxmuller_sample, polarmethod_sample)
```

```
##
##  Exact two-sample Kolmogorov-Smirnov test
##
## data:  boxmuller_sample and polarmethod_sample
## D = 0.26667, p-value = 0.2391
## alternative hypothesis: two-sided
```

- Generate n values from the log-normal distributions. Note that if X has a normal distribution, then $Y = \exp(X)$ has a log-normal distribution. Thus you may generate log-normal realizations based on the normal variates that you have generated.

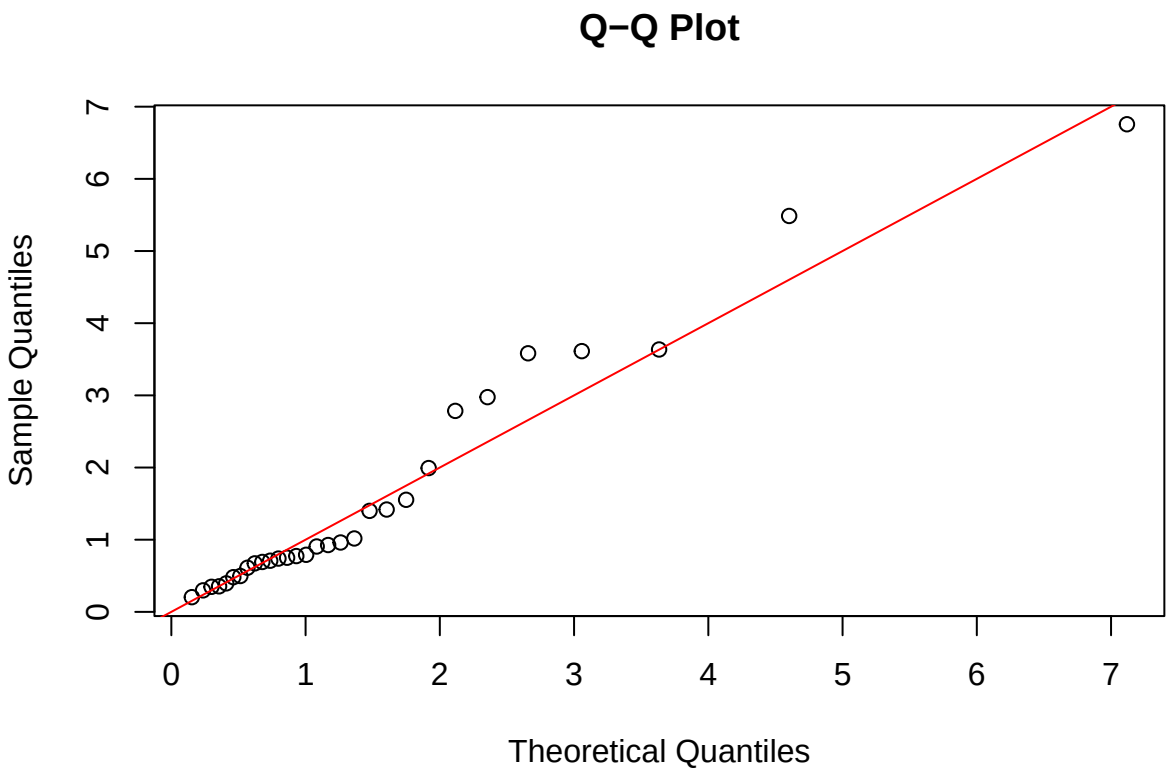
```
box_muller_sample_y <- exp(boxmuller_sample)
print(box_muller_sample_y)
```

```
##  [1] 3.5831176 1.0170976 3.6357302 0.7909054 1.3995468 6.7564683 0.6720560
##  [8] 0.7386342 2.9753929 0.4969816 0.9263228 0.2982066 0.4807490 0.7732140
## [15] 3.6129562 0.3947969 1.9927802 1.4180587 1.5530501 0.9059257 0.2039582
## [22] 0.3472123 0.6917088 0.9614863 0.3539061 0.7504779 0.7090412 0.6114057
## [29] 2.7847062 5.4860401
```

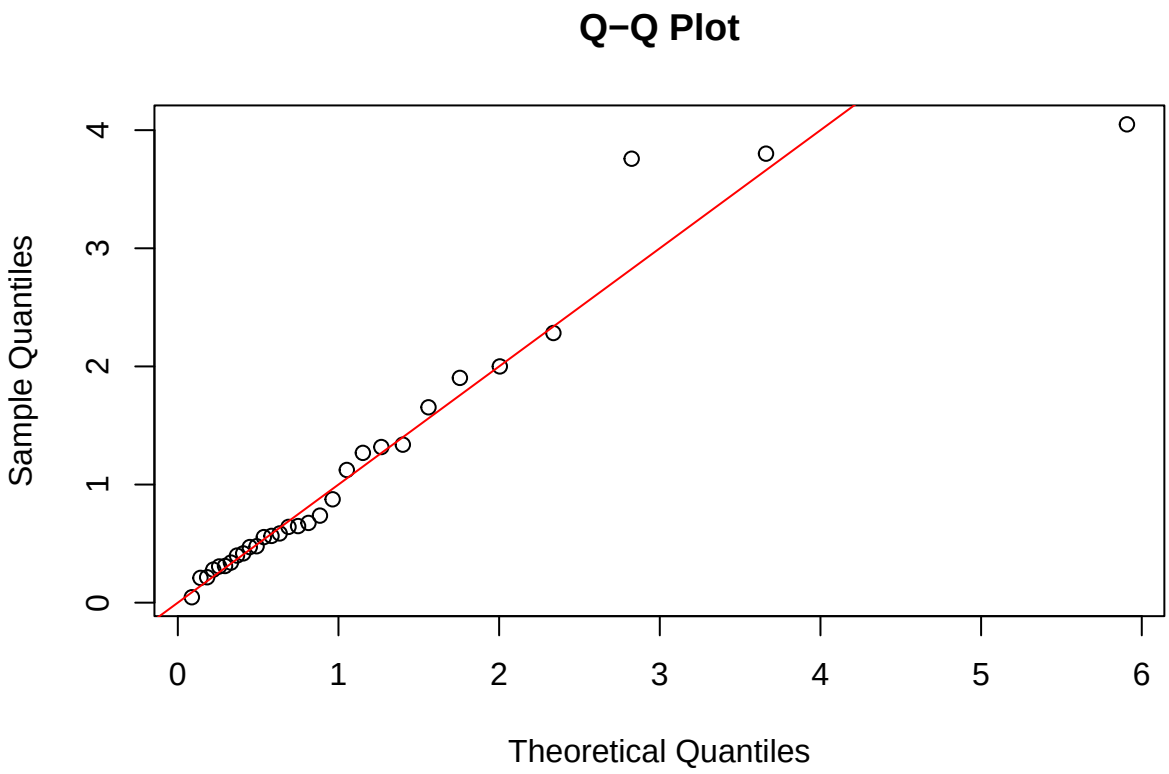
```r
polar_method_sample_y <- exp(polarmethod_sample)
print(polar_method_sample_y)
```

```
##  [1] 1.90295363 0.64146769 2.00006535 0.47785034 0.33947873 0.30635284
##  [7] 0.20983561 0.64851010 0.55493609 1.26814951 1.33765967 0.58607637
## [13] 2.28271603 1.31756584 0.47075996 3.75837261 1.65383310 4.04920276
## [19] 3.80115192 0.87512197 0.21513647 0.73700302 0.31038396 0.67496603
## [25] 0.39870251 0.41736244 0.27943835 0.56505180 1.12337939 0.04599326
```

```r
theoretical_quantiles <- qlnorm(ppoints(length(box_muller_sample_y)), meanlog = mean(log(box_muller_samp
qqplot(theoretical_quantiles, box_muller_sample_y, main = "Q-Q Plot", xlab = "Theoretical Quantiles", yl
abline(0, 1, col = "red")
```




```r
theoretical_quantiles <- qlnorm(ppoints(length(polar_method_sample_y)), meanlog = mean(log(polar_method
qqplot(theoretical_quantiles, polar_method_sample_y, main = "Q-Q Plot", xlab = "Theoretical Quantiles",
abline(0, 1, col = "red")
```



- Have a Q-Q plot to check whether the sample comforts with the log-normal distribution. You may use `qlnorm` function in R to compute the quantiles of log-normal distributions.

2. Generators of multivariate normal distributions.

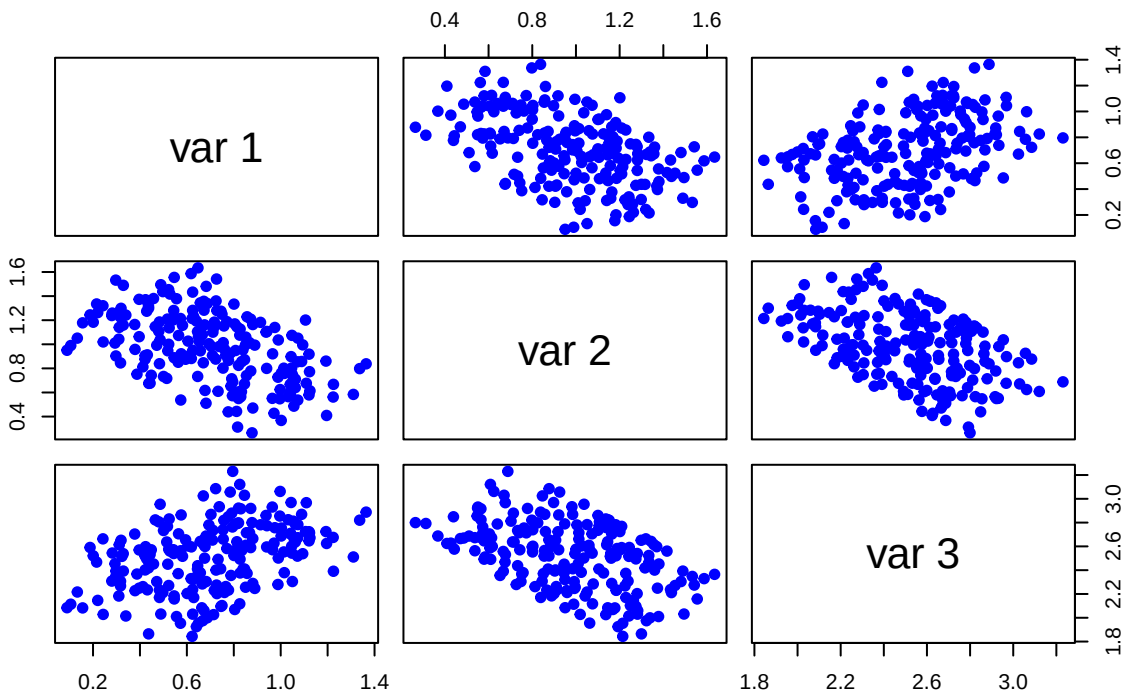
- Write an R function called `mymvrnorm` that generates n random observations from $\mathrm{N}_p(\mu, \Sigma)$. Only calls to R's standard uniform generators `runif` are permitted. This function should have three arguments:
  - `n`, the random sample size,
  - `mu`, the mean vector with p entries,
  - `Sigma`, the variance-covariance matrix, which is symmetric and positive semi-definite.

```
mymvrnorm <- function(n, mu, Sigma) {
p <- length(mu)
A <- eigen(Sigma)$vectors %*% diag(sqrt(eigen(Sigma)$values))
X <- matrix(runif(n * p), ncol = p)
Y <- X %*% t(A) + matrix(mu, nrow = n, ncol = p, byrow = TRUE)
return(Y)
}
# Set parameters
n <- 200
mu <- c(0, 1, 2)  # Mean vector
Sigma <- matrix(c(1.0, -0.5, 0.5,
                  -0.5, 1.0, -0.5,
                  0.5, -0.5, 1.0),
```

```
                nrow = 3, byrow = TRUE)  # Covariance matrix

# Generate random samples
set.seed(123)  # For reproducibility
samples <- mymvrnorm(n, mu, Sigma)

# Print the first few samples to verify
head(samples)
```

```
##           [,1]      [,2]     [,3]
## [1,] 0.8041048 1.1692063 2.069520
## [2,] 0.7227844 0.8770990 3.085260
## [3,] 0.8566090 1.2280950 2.373271
## [4,] 1.0537087 0.7028982 2.812132
## [5,] 0.9962008 0.5475546 2.855019
## [6,] 0.2968903 1.5327733 2.347473
```

```
n <- 200
mu <- c(0, 1, 2)
Sigma <- matrix(c(1.0, -0.5, 0.5,
                  -0.5, 1.0, -0.5,
                  0.5, -0.5, 1.0),
                nrow = 3, byrow = TRUE)

set.seed(123)
samples <- mymvrnorm(n, mu, Sigma)
pairs(samples, main = "Pairs Plot of Generated Multivariate Normal Samples",
      col = "blue", pch = 19)
```




```
cor(samples)
```

```
##            [,1]       [,2]      [,3]
## [1,]  1.0000000 -0.4895213  0.406531
## [2,] -0.4895213  1.0000000 -0.470553
## [3,]  0.4065310 -0.4705530  1.000000
```

- This function should return a matrix with n rows and p columns, where the ith row has the realization of Y_i. The `eigen` function should be called in your definition of `mymvrnorm`.
- Generate 200 random observations from the 3-dimensional multivariate normal distribution having mean vector $\mu = (0, 1, 2)$ and covariance matrix

$$\Sigma = \begin{bmatrix} 1.0 & -0.5 & 0.5 \\ -0.5 & 1.0 & -0.5 \\ 0.5 & -0.5 & 1.0 \end{bmatrix}$$

  using your `mymvrnorm` function.
- Use the R `pairs` plot to graph an array of scatter plots for each pair of variables. For each pair of variables, (visually) check that the location and correlation approximately agree with the theoretical parameters of the corresponding bivariate normal distribution.

3. Truncated normal Here we consider sampling the $\mathrm{N}(0, 1)$ distribution truncated to the interval $[a, \infty)$ where $a > 0$. Inverting the truncated CDF leads to the formula $X = g_1(U) \equiv \Phi^{-1}(\Phi(a) + (1 - \Phi(a))U)$ where $U \sim \mathbf{U}(0, 1)$.

- Now find the smallest integer $a \geqslant 1$ for which $X = g_1(U)$ fails to work. For concreteness we can define failure as delivering at least one NaN or $\pm\infty$ when tested with $U \in \{(i-1/2)/1000 \mid, i = 1, 2, \ldots, 1000\}$.
- Consider $X = g_2(U) \equiv -\Phi^{-1}(\Phi(-a)(1-U))$ that also generates X from the same distribution. Find the smallest integer $a \geqslant 1$ at which g_2 fails using the same criterion as for g_1.

```
#for case 1 at smallest integer
a <- 1
while (TRUE) {
x <- qnorm(pnorm(a) + (1 - pnorm(a)) * runif(1000, (1/2)/1000, (1000 - 1/2)/1000))
if (any(is.nan(x)) || any(is.infinite(x))) {
break
}
a <- a + 1
}
cat("The smallest integer a > 1 for which X = g1(U) fails to work is", a, "\n")
```

```
## The smallest integer a > 1 for which X = g1(U) fails to work is 8
```

```
a <- 1
while (TRUE) {
x <- -qnorm(pnorm(-a) * runif(1000, (1/2)/1000, (1000 - 1/2)/1000))
if (any(is.nan(x)) || any(is.infinite(x))) {
break
}
a <- a + 1
}

cat("The smallest integer a > 1 at which g2 fails is", a, "\n")
```

```
## The smallest integer a > 1 at which g2 fails is 38
```

4. Negative binomial distribution The negative binomial distribution with parameters $r \in \{1, 2, \ldots\}$ and $p \in (0, 1)$, denoted $\text{Negbin}(r, p)$, has probability mass function

$$p_k = \mathbb{P}(X = k) = \binom{k+r-1}{r-1} p^r (1-p)^k, \quad k = 0, 1, \ldots$$

It describes the number of failures before the r' th success in a sequence of independent Bernoulli trials with success probability p. For $r = 1$, it reduces to the geometric distribution. The negative binomial distribution has the following compound representation: $X \sim \text{Poi}(\lambda)$ for $\lambda \sim \text{Gam}(r) \times (1-p)/p$. We don't need r to be an integer. Writing

$$\binom{k+r-1}{r-1} = \frac{(k+r-1)!}{(r-1)!k!} = \frac{\Gamma(k+r)}{\Gamma(r)\Gamma(k+1)}$$

yields a valid distribution

$$p_k = \mathbb{P}(X = k) = \frac{\Gamma(k+r)}{\Gamma(r)\Gamma(k+1)} p^r (1-p)^k$$

for $k \in \{0, 1, 2, \ldots\}$ for any real $r > 0$. The Poisson-gamma mixture representation also holds for $r > 0$. Using the compound representation we find that $\mathbb{E}(X) = r(1-p)/p$ and $\text{Var}(X) = r(1-p)/p^2$.

Generate 1000 variables following negative binomial distribution through the compound representation. Show that the sample mean and sample variance are close to the population mean and variance.

```r
set.seed(123)
r <- 5
p <- 0.4

n <- 1000

lambda <- rgamma(n, r, scale = (1 - p)/p)



X <- rpois(n, lambda)

# Calculate the sample mean and variance
sample_mean <- mean(X)
sample_variance <- var(X)

# Calculate the theoretical mean and variance
theoretical_mean <- r * (1 - p) / p
theoretical_variance <- r * (1 - p) / p^2

# Print the sample and theoretical values
cat("Sample mean:", sample_mean, "\n")
```

```
## Sample mean: 7.376
```

```r
cat("Sample variance:", sample_variance, "\n")
```

```
## Sample variance: 18.41104
```

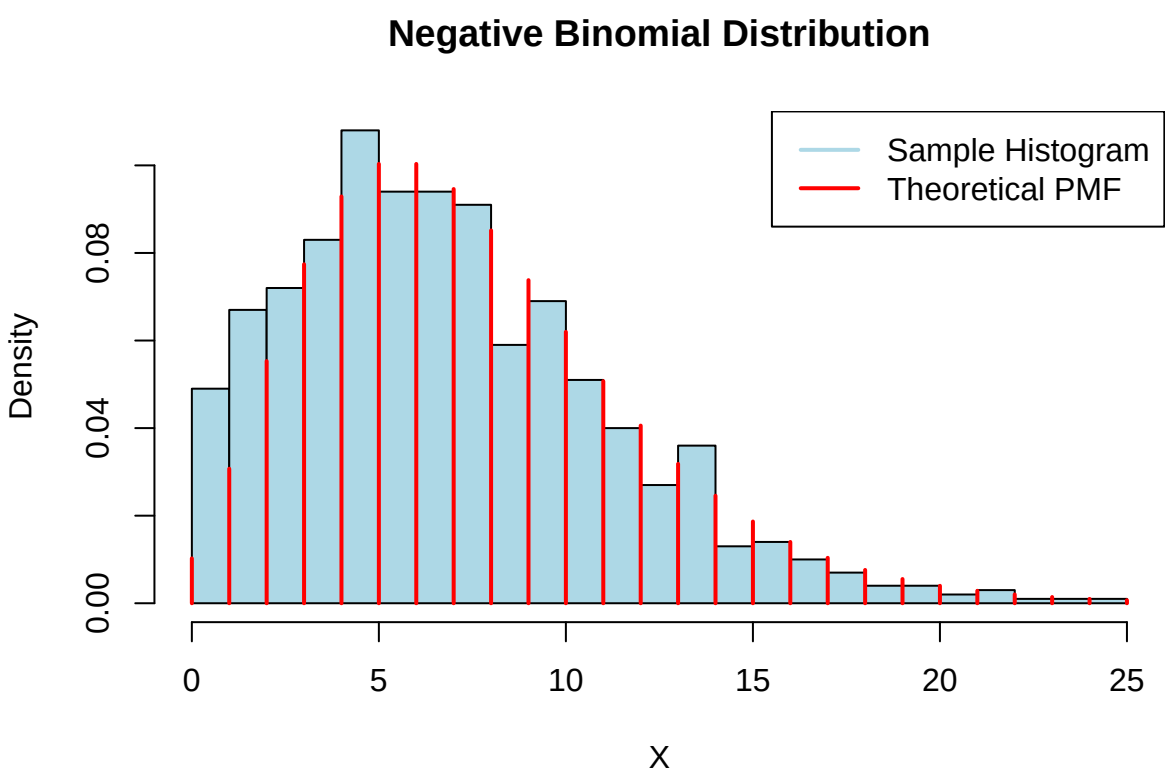
```r
cat("Theoretical mean:", theoretical_mean, "\n")
```

```
## Theoretical mean: 7.5
```

```r
cat("Theoretical variance:", theoretical_variance, "\n")
```

```
## Theoretical variance: 18.75
```

```r
# Plot the histogram of the generated data
hist(X, breaks = 30, col = "lightblue", main = "Negative Binomial Distribution", xlab = "X", freq = FALS

# Overlay the theoretical probability mass function (pmf)
x_vals <- 0:max(X)
pmf_vals <- dnbinom(x_vals, size = r, prob = p)
lines(x_vals, pmf_vals, type = "h", lwd = 2, col = "red")

# Add a legend
legend("topright", legend = c("Sample Histogram", "Theoretical PMF"), col = c("lightblue", "red"), lwd =
```



5. Rshiny The dataset in counties.rds contains the name of each county in the United States, the total population of the county and the percent of residents in the county who are White, Black, Hispanic, or Asian. During the presentation, we use the percent of residents in the county. In the homework, we hope that you could establish the census app by filling the missing code.

Link of counties.rds: https://shiny.rstudio.com/tutorial/written-tutorial/lesson5/census-app/data/counties.rds

Link of helper.R: https://shiny.rstudio.com/tutorial/written-tutorial/lesson5/census-app/helpers.R

Instructional Codes:

```
# Load packages
library(shiny)
library(maps)
library(mapproj)

# Load data
#counties <- readRDS("C:/Users/Wylan Gao/Desktop/counties.rds")


# Source helper functions
#source("https://shiny.posit.co/r/getstarted/shiny-basics/lesson5/census-app/helpers.R")

# User interface
ui <- fluidPage(
  titlePanel("Demographic Map of the United States"),
```

```
  sidebarLayout(
    sidebarPanel(
      helpText("Use the slider to create a demographic maps with
        information from Data provided by the US Census."),

      selectInput("var",
                  label = "Choose a variable to display",
                  choices = list("% White" = "white",
                                 "% Black" = "black",
                                 "% Hispanic" = "hispanic",
                                 "% Asian" = "asian"),
                  selected = "asian"),

      sliderInput("range",
                  label = "Range:",
                  min = 0, max = 100, value = c(0, 100))
    ),

    mainPanel(plotOutput("map"))
  )
)

# Server logic
server <- function(input, output) {
  output$map <- renderPlot({
    data <- switch(input$var,
                   "white" = counties$white,
                   "black" = counties$black,
                   "hispanic" = counties$hispanic,
                   "asian" = counties$asian)

    color <- switch(input$var,
                    "white" = "darkgreen",
                   "black" = "darkred",
                   "hispanic" = "darkorange",
                   "asian" = "darkblue")

    legend <- switch(input$var,
                     "white" = "Percent White",
                    "black" = "Percent Black",
                    "hispanic" = "Percent Hispanic",
                    "asian" = "Percent Asian")

    percent_map(data, color, legend, input$range[1], input$range[2])
  })
}

# Run app
#shinyApp(ui, server)
```