

Entrega 1

Rubio Cobeta, Juan

11 de octubre de 2025

Ejercicio 1:

1. Código de la función

```
puntos <- function(n, m, p) {  
  
  urna_elegida <- sample(c("dados", "naipes"), size = 1)  
  
  if (urna_elegida == "dados") {  
  
    num_dados_a_sacar <- ceiling(n * p / 100)  
  
    lanzamientos <- sample(1:6, size = num_dados_a_sacar, replace = TRUE)  
  
    resultado <- trunc(mean(lanzamientos))  
  
  } else {  
  
    urna_naipes <- rep(c(10, 11, 12), times = m)  
  
    num_naipes_a_sacar <- ceiling(length(urna_naipes) * p / 100)  
  
    naipes_extraidos <- sample(urna_naipes, size = num_naipes_a_sacar)  
  
    resultado <- sum(naipes_extraidos)  
  
  }  
  
  return(resultado)  
}
```

2. Explicación razonada del código empleado

El código define una función llamada `puntos` que acepta tres argumentos: `n` (número de dados en la primera urna), `m` (número de grupos de figuras en la segunda) y `p` (el porcentaje de elementos a extraer).

La lógica de la función es la siguiente:

1. **Elección de la Urna:** La primera acción es seleccionar una de las dos urnas. Esto se logra con `sample(c("dados", "naipes"), size = 1)`, que devuelve aleatoriamente una de las dos cadenas de texto con una probabilidad de 50% para cada una. El resultado se almacena en la variable `urna_elegida`.
2. **Estructura Condicional `if/else`:** El flujo del programa se divide en dos ramas principales basadas en el valor de `urna_elegida`.

3. Bloque if (Si se elige la urna “dados”):

- Se calcula cuántos dados extraer. La fórmula `ceiling(n * p / 100)` obtiene el `p` por ciento de `n`. Se utiliza `ceiling()` para redondear el resultado al siguiente entero superior, garantizando que si, por ejemplo, el resultado es 4.1, se extraigan 5 dados, lo cual es una interpretación lógica del requerimiento.
- La simulación de los lanzamientos se realiza de forma vectorizada con `sample(1:6, size = num_dados_a_sacar, replace = TRUE)`. Esta única línea de código genera todos los resultados de los dados a la vez, cumpliendo la restricción de no usar bucles. `replace = TRUE` es fundamental, ya que el resultado de un dado no influye en los demás.
- Finalmente, se calcula la media de las puntuaciones con `mean()` y se elimina la parte decimal con `trunc()`.

4. Bloque else (Si se elige la urna “naipes”):

- Primero, se establece el contenido de la urna. Asumiendo las puntuaciones de la baraja española (`Sota=10, Caballo=11, Rey=12`), se crea un vector numérico con `m` copias de cada valor usando la función `rep()`.
- Se calcula el número de naipes a extraer de forma análoga al caso de los dados, pero esta vez sobre el tamaño total de la urna de naipes (`3 * m`).
- Se extraen las cartas con `sample(urna_naipes, size = num_naipes_a_sacar)`. Por defecto, `sample()` opera sin reemplazo, lo que simula correctamente la extracción de cartas únicas de una baraja.
- El resultado final es la suma de los valores de las cartas extraídas, calculado con `sum()`.

5. Valor de Retorno:

La función concluye devolviendo el valor almacenado en la variable `resultado`, que contendrá la puntuación calculada según la urna elegida.

3. Interpretación de los resultados obtenidos

Para demostrar el funcionamiento de la función y poder interpretar sus resultados, la ejecutaremos en tres escenarios distintos. Cada caso está diseñado para probar un aspecto específico: su exactitud matemática con valores simples, su comportamiento en un caso general y su capacidad para manejar grandes volúmenes de datos.

Caso 1: Comprobación con valores sencillos

Primero, realizamos una comprobación con `m=1` grupo de naipes (3 cartas en total) y extraemos el `p=100%`. Esto nos permite verificar la lógica de forma manual.

```
set.seed(1200)
resultado_caso_1 <- puntos(n = 60, m = 1, p = 100)
print(resultado_caso_1)
```

```
## [1] 33
```

Interpretación:

Si usamos esos parámetros, se consigue un grupo de figuras, y el resultado es 33, acorde a lo que debería ser: $10+11+12$. Vemos que el código funciona como debería.

Caso 2: Un ejemplo general

A continuación, se ejecuta un caso general con valores intermedios: `n=60` dados, `m=25` grupos de naipes, y un porcentaje de extracción del `p=20%`.

```
set.seed(123)
resultado_caso_2 <- puntos(n = 60, m = 25, p = 20)
print(resultado_caso_2)
```

```
## [1] 3
```

Interpretación:

Al ejecutar la función puntos(n = 60, m = 25, p = 20) con la semilla 123, el resultado obtenido es 3.

Este resultado se ha producido de la siguiente manera:

Con la semilla 123, esta vez la función sample() ha elegido la urna de los dados.

La urna contenía n=60 dados.

Se ha extraído el p=20% de esos dados, lo que corresponde a ceiling($60 * 0.20$) = 12 dados.

La función ha simulado 12 lanzamientos, ha calculado su valor medio y lo ha truncado a su parte entera, dando como resultado final 3.

Este valor representa el resultado de un único experimento aleatorio. Una ejecución diferente podría elegir otra urna o producir un resultado distinto.

Caso 3: Comportamiento con valores grandes

Finalmente, se prueba la función con un volumen de datos muy alto (m=1,000,000 de grupos de naipes) para verificar que escala correctamente.

```
set.seed(1200)
resultado_caso_3 <- puntos(n = 60, m = 1e6, p = 100)
print(resultado_caso_3)

## [1] 3.3e+07
```

Interpretación:

Al ejecutar la función puntos(n = 60, m = 1e6, p = 100) con la semilla 1200, el resultado obtenido es 3.3e+07.

Este resultado se ha producido de la siguiente manera:

Al igual que en el primer caso, la semilla 1200 ha elegido la urna de los naipes.

La urna contenía m = 1,000,000 de grupos, es decir, un total de 3 millones de naipes.

Se ha extraído el p=100%, por lo que se han seleccionado los tres millones de naipes.

El resultado 3.3e+07 es la suma de las puntuaciones de todas las cartas; $(10+11+12) * 1e6 = 33 * 1e6$

Este último caso demuestra que la función es robusta y capaz de manejar grandes volúmenes de datos de forma eficiente.

Ejercicio 1.2:

Para este ejercicio, hemos modificado levemente la función puntos, para que nos devuelva también el número de cada urna:

Código de la función:

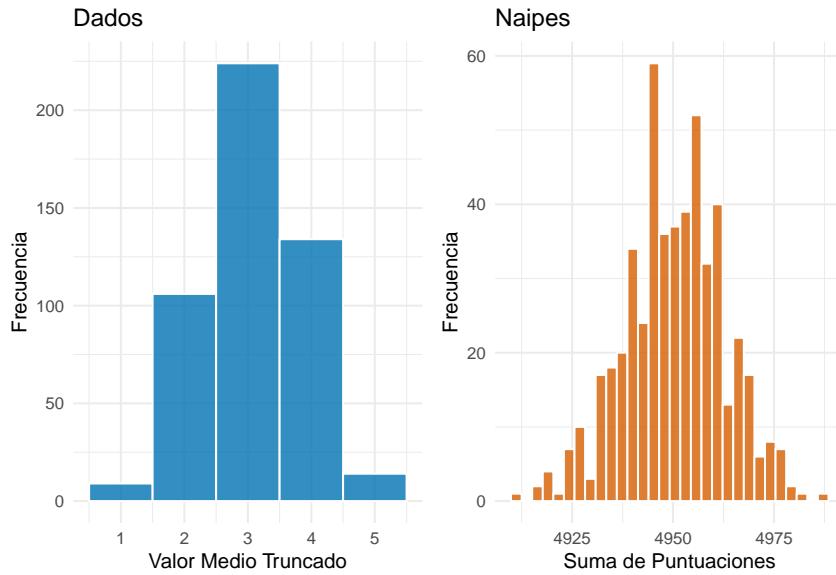
```
library(patchwork)
puntos <- function(n, m, p) {
  urna_elegida <- sample(c("dados", "naipes"), size = 1)

  if (urna_elegida == "dados") {
    num_dados_a_sacar <- ceiling(n * p / 100)
    lanzamientos <- sample(1:6, size = num_dados_a_sacar, replace = TRUE)
    resultado <- trunc(mean(lanzamientos))
  } else {
    urna_naipes <- rep(c(10, 11, 12), times = m)
    num_naipes_a_sacar <- ceiling(length(urna_naipes) * p / 100)
    naipes_extraidos <- sample(urna_naipes, size = num_naipes_a_sacar)
    resultado <- sum(naipes_extraidos)
  }
  # El cambio clave: devolver una lista con ambos datos.
  return(list(urna = urna_elegida, resultado = resultado))
}

# 2. Nueva función 'grafica'
grafica <- function(k, n, m, p) {

  if (k <= 10) {
    stop("El valor de k debe ser mayor que 10.")
  }
  lista_resultados <- replicate(k, puntos(n, m, p), simplify = FALSE)
  df_resultados <- dplyr::bind_rows(lista_resultados)
  resultados_dados <- df_resultados[df_resultados$urna == "dados", ]
  resultados_naipes <- df_resultados[df_resultados$urna == "naipes", ]
  plot_dados <- ggplot(resultados_dados, aes(x = resultado)) +
    geom_histogram(binwidth = 1, fill = "#0072B2", color = "white", alpha = 0.8) +
    labs(title = "Datos",
         x = "Valor Medio Truncado", y = "Frecuencia") +
    theme_minimal()
  plot_naipes <- ggplot(resultados_naipes, aes(x = resultado)) +
    geom_histogram(fill = "#D55E00", color = "white", alpha = 0.8) +
    labs(title = "Naipes",
         x = "Suma de Puntuaciones", y = "Frecuencia") +
    theme_minimal()
  plot_final <- plot_dados + plot_naipes

  print(plot_final)
}
set.seed(999)
grafica(k = 1000, n = 10, m = 300, p = 50)
```



2. Explicación razonada del código empleado

- **Modificación de la función puntos:** El primer paso fue alterar la función del Ejercicio 1. La versión original devolvía únicamente el valor numérico del resultado. Para poder clasificar las `k` simulaciones, la función se modificó para que devuelva una `lista` de R con dos componentes: `urna` (una cadena de texto, “dados” o “naipes”) y `resultado` (el valor numérico). Esto es fundamental para el resto del proceso.
- **Nueva función grafica y sus parámetros:** Se define la función `grafica` que acepta `k` (el número de simulaciones) y los parámetros `n`, `m` y `p` para pasárselos a la función `puntos`. Incluye una validación inicial para asegurar que `k > 10`.
- **Ejecución sin bucles con `replicate()`:** Para ejecutar la función `puntos` `k` veces sin usar un ciclo `for` o `while`, se emplea la función `replicate()`. Esta función está diseñada específicamente para repetir una expresión en R un número determinado de veces. El argumento `simplify = FALSE` es importante para que el resultado sea una lista, donde cada elemento es la lista devuelta por `puntos`.
- **Transformación de datos con `dplyr::bind_rows()`:** El resultado de `replicate()` es una lista de listas, un formato poco práctico para la visualización. Se transforma esta estructura en una tabla de datos (o `data.frame`) única y ordenada usando `dplyr::bind_rows()`. Esta función toma la lista y apila cada elemento de forma inteligente, creando dos columnas: `urna` y `resultado`.
- **Elección de los gráficos:** Se eligió el **histograma** como el tipo de gráfico apropiado para ambos casos, ya que es ideal para visualizar la distribución de una variable numérica.
 - Para los **dados**, donde los resultados son enteros, se usa `geom_histogram(binwidth = 1)`. Esto hace que cada barra represente la frecuencia de un único valor entero (2, 3, 4, etc.), funcionando de manera muy similar a un gráfico de barras.
 - Para los **naipes**, se deja que `ggplot2` determine el ancho de las barras automáticamente, lo cual es adecuado para una variable con un rango de valores más amplio.
- **Combinación de gráficos con `patchwork`:** Para mostrar ambos gráficos en una misma ventana, se utiliza el paquete `patchwork`. Este paquete sobrecarga el operador `+`, permitiendo “sumar” objetos de `ggplot` para componerlos en una sola visualización de forma muy intuitiva. El código `plot_dados + plot_naipes` coloca ambos gráficos uno al lado del otro.

3. Interpretación

Al ejecutar la función, se generan y muestran dos histogramas, uno para cada urna, combinados en una sola figura.

1. **Gráfico de Dados (izquierda):** Este gráfico muestra la distribución de las medias truncadas de los lanzamientos de dados. Como la media teórica del lanzamiento de un dado es 3.5, es lógico que los resultados de la simulación se concentren mayoritariamente en los valores **3 y 4**, que son los más frecuentes. La distribución es aproximadamente simétrica en torno a estos valores centrales.
2. **Gráfico de Naipes (derecha):** Este gráfico muestra la distribución de la suma de las puntuaciones de las cartas extraídas. Se puede observar una distribución que se asemeja a una **curva normal (campana de Gauss)**. Esto es un resultado esperado y un ejemplo práctico del **Teorema del Límite Central**.

En conjunto, la función **grafica** nos permite visualizar y entender de forma muy clara el comportamiento probabilístico de los dos procesos aleatorios a lo largo de muchas repeticiones.

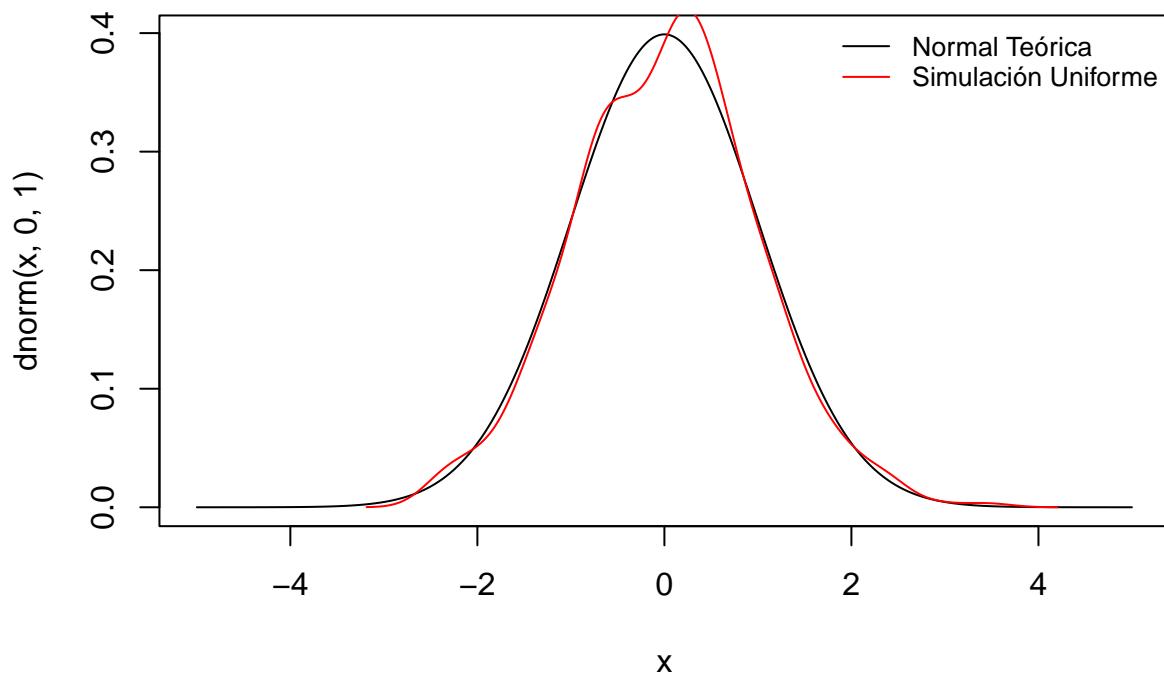
EJERCICIO 2:

Realiza una simulación del TCL utilizando una distribución Uniforme(0,1) ¿Cuando $X_i \sim \text{Uniforme}(0, 1)$ son necesarios los mismos valores de n para obtener una buena aproximación Normal de \bar{X}_n ? ¿Por qué?

Código de la función:

```
media <- 0.5
x <- seq(-5, 5, length.out = 1000)
aproxnormal <- function(n, m)
{
  datos <- matrix(runif(n * m, 0, 1), nrow = n)
  Xns <- colMeans(datos)
  simplifico <- (Xns - media) * sqrt(n) * sqrt(12)
  plot(x, dnorm(x, 0, 1), type = "l", main = paste("Aproximación con n =", n))
  lines(density(simplifico), col = "red")
  legend("topright", legend=c("Normal Teórica", "Simulación Uniforme"),
         col=c("black", "red"), lty=1, cex=0.8, bty="n")
}
aproxnormal(1000, 500)
```

Aproximación con n = 1000



2. Explicación razonada del código empleado

El código está diseñado para simular y visualizar el Teorema del Límite Central (TCL). El objetivo es comprobar si la distribución de las medias muestrales estandarizadas de una población Uniforme se aproxima a una distribución Normal estándar.

- Función `aproxnormal(n, m)`:

- **Parámetros:** La función toma dos argumentos: `n`, que es el tamaño de cada muestra, y `m`, que es el número de muestras (o repeticiones del experimento) que se van a generar.
- **Generación de Datos:** La línea `datos <- matrix(runif(n * m, 0, 1), nrow = n)` es el núcleo de la simulación. Genera $n * m$ números aleatorios de una distribución Uniforme(0,1) y los organiza en una matriz de `n` filas y `m` columnas. De esta forma, cada una de las `m` columnas representa una muestra aleatoria de tamaño `n`. Este método es vectorizado y mucho más eficiente que un bucle.
- **Cálculo de Medias:** `Xns <- colMeans(datos)` calcula la media de cada columna, generando un vector de `m` medias muestrales (\bar{X}_n).
- **Estandarización:** La línea `simplifico <- (Xns - media) * sqrt(n) * sqrt(12)` aplica la fórmula de estandarización del TCL, que es $Z = \frac{\bar{X}_n - \mu}{\sigma/\sqrt{n}}$. Para una distribución Uniforme(0,1), la media teórica μ es 0.5 y la varianza σ^2 es 1/12. La fórmula del código es una reorganización algebraica de la fórmula estándar.
- **Visualización:** El código primero dibuja la curva de densidad de una Normal estándar teórica (`dnorm(x, 0, 1)`, línea negra) y luego superpone la densidad estimada de las medias muestrales estandarizadas (`density(simplifico)`, línea roja) para comparar el ajuste.
- **Ejecución:**
 - `aproxnormal(1000, 500)` ejecuta la simulación con un tamaño de muestra grande (`n=1000`) y 500 repeticiones para mostrar un caso donde la convergencia es clara.

3. Interpretación

Al ejecutar el código con `n=1000`, el gráfico muestra que la línea roja (densidad de las medias simuladas) y la línea negra (Normal estándar teórica) son prácticamente indistinguibles. Esto confirma visualmente que el Teorema del Límite Central se cumple para la distribución Uniforme.

Respuesta a la pregunta del enunciado:

No, no son necesarios los mismos valores de `n`. Cuando las muestras provienen de una distribución Uniforme, se necesita un tamaño de muestra (`n`) significativamente menor para obtener una buena aproximación a la Normalidad en comparación con distribuciones que son muy asimétricas (como, por ejemplo, una distribución Exponencial).

¿Por qué?

La razón fundamental es la simetría de la distribución de partida.

1. **Caso de la Distribución Uniforme:** La distribución Uniforme(0,1) es perfectamente simétrica alrededor de su media (0.5). El TCL describe cómo la distribución de las medias muestrales converge hacia la distribución Normal, que es el arquetipo de una distribución simétrica. Si partimos de una distribución que ya es simétrica, el TCL tiene “menos trabajo que hacer”. La distribución de \bar{X}_n será simétrica desde el principio, incluso para `n` muy pequeños. Solo necesita que su forma se vuelva más acampanada, lo cual ocurre muy rápidamente. En la práctica, con `n` tan bajo como 5 o 10, la aproximación ya es bastante razonable.
2. **Caso de una Distribución Asimétrica:** Por el contrario, si partiéramos de una distribución muy asimétrica (como la Exponencial), las medias de muestras pequeñas también tendrían una distribución asimétrica. Se necesitaría un tamaño de muestra `n` mucho mayor (la regla empírica suele citar `n > 30`) para que el efecto de promediar muchas observaciones logre “contrarrestar” esa asimetría inicial y producir una distribución de medias que sea lo suficientemente simétrica y con forma de campana.

En conclusión, la velocidad de convergencia a la Normalidad en el TCL depende fuertemente de la forma de la distribución original de los datos. Para distribuciones simétricas como la Uniforme, la convergencia es muy rápida.

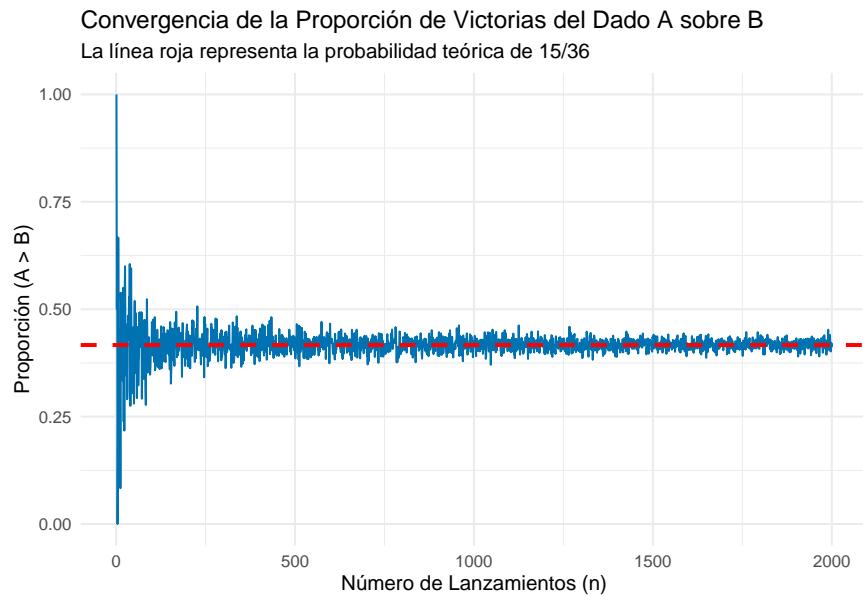
Ejercicio 3:

1. Código empleado

```
library(ggplot2)
proporcion_A_mayor <- function(n) {
  lanzamientos_A <- sample(1:6, size = n, replace = TRUE)
  lanzamientos_B <- sample(1:6, size = n, replace = TRUE)
  A_gana <- lanzamientos_A > lanzamientos_B
  return(sum(A_gana) / n)
}

n_max_lanzamientos <- 2000
vector_n <- 1:n_max_lanzamientos
set.seed(2025)
proporciones_calculadas <- sapply(vector_n, proporcion_A_mayor)

datos_grafico <- data.frame(
  Numero_de_Lanzamientos = vector_n,
  Proporcion = proporciones_calculadas
)
prob_teorica <- 15/36
grafico_convergencia <- ggplot(datos_grafico, aes(x = Numero_de_Lanzamientos, y = Proporcion)) +
  geom_line(color = "#0072B2") +
  geom_hline(aes(yintercept = prob_teorica), color = "red", linetype = "dashed", size = 1) +
  labs(
    title = "Convergencia de la Proporción de Victorias del Dado A sobre B",
    subtitle = paste("La línea roja representa la probabilidad teórica de 15/36"),
    x = "Número de Lanzamientos (n)",
    y = "Proporción (A > B)"
  ) +
  theme_minimal()
print(grafico_convergencia)
```



2. Explicación razonada del código empleado

El objetivo es visualizar cómo la proporción de un evento aleatorio converge a su probabilidad teórica a medida que aumenta el número de repeticiones.

- **Función Vectorizada `proporcion_A_mayor(n)`:** Esta función auxiliar encapsula la lógica del experimento para un número `n` de lanzamientos.
 - **Simulación de Lanzamientos:** En lugar de simular un lanzamiento cada vez, se generan los `n` resultados de ambos dados de una sola vez con `sample(1:6, size = n, replace = TRUE)`. Esto crea dos vectores (`lanzamientos_A` y `lanzamientos_B`) de longitud `n`.
 - **Comparación Vectorial:** La comparación `lanzamientos_A > lanzamientos_B` se realiza elemento a elemento entre los dos vectores, devolviendo un vector lógico (TRUE/FALSE) que indica en qué lanzamientos el dado A tuvo un resultado superior.
 - **Cálculo de la Proporción:** R interpreta TRUE como 1 y FALSE como 0. Por tanto, `sum()` sobre el vector lógico cuenta eficientemente el número de veces que A fue superior. Al dividir esta suma entre `n`, obtenemos la proporción de victorias para esa simulación.
- **Repetición del Experimento con `sapply()`:**
 - Para observar la convergencia, no basta con ejecutar el experimento para un único valor de `n`. Necesitamos ver cómo evoluciona la proporción para una secuencia de `n` (ej. desde 1 hasta 2000), cuantas más veces se haga más exacto será.
 - La función `sapply()` es la herramienta vectorizada ideal para este propósito. La línea `sapply(vector_n, proporcion_A_mayor)` aplica la función `proporcion_A_mayor` a cada elemento del `vector_n` y devuelve los resultados en un nuevo vector, evitando así un bucle `for`.
- **Visualización con `ggplot2`:**
 - Se crea un `data.frame` que servirá como fuente de datos para el gráfico, con el número de lanzamientos en el eje X y la proporción calculada en el eje Y.
 - Se utiliza un gráfico de líneas (`geom_line`), ya que es el más adecuado para mostrar una tendencia a lo largo de una secuencia continua como es el número de lanzamientos.
 - Se añade una línea horizontal (`geom_hline`) que representa la probabilidad teórica de que `A > B` ($15/36$). Esta línea sirve como referencia visual para comprobar la convergencia del experimento.

3. Interpretación de los resultados obtenidos

El gráfico muestra la evolución de la proporción de veces que el resultado del dado A es superior al del dado B (línea azul) a medida que aumenta el número de lanzamientos del experimento (`n`, en el eje X).

- **Volatilidad con `n` pequeños:** Se observa claramente que cuando el número de lanzamientos es bajo (a la izquierda del gráfico), la proporción es muy inestable y puede tomar valores extremos (muy altos o muy bajos), alejados del valor real.
- **Convergencia con `n` grandes:** A medida que el número de lanzamientos `n` aumenta (moviéndonos hacia la derecha), la línea azul se estabiliza progresivamente y se acerca cada vez más a la línea roja discontinua.
- **Probabilidad Teórica:** La línea roja representa la probabilidad teórica del suceso ($15/36 = 0.417$).

Este gráfico es una demostración visual perfecta de la **Ley de los Grandes Números**: a medida que el número de repeticiones de un experimento aleatorio aumenta, la frecuencia relativa (proporción) de un suceso tiende a converger a su probabilidad teórica.

Ejercicio 4: Resumen y Valoración del Curso

A continuación, se presenta una valoración personal sobre el desarrollo del curso hasta la fecha, estructurada en los cuatro puntos solicitados.

- **Contenidos teóricos ofrecidos en el curso:** La valoración de los contenidos teóricos es muy positiva. Los temas presentados resultan interesantes y relevantes, destacando especialmente la orientación práctica de los apuntes. La combinación de una base teórica sólida con ejemplos de código en R directamente aplicables facilita enormemente la asimilación de conceptos y su posterior implementación, creando un puente efectivo entre la teoría y la práctica.
- **Desarrollo temporal:** El desarrollo temporal del curso me ha parecido adecuado y bien planificado. La secuenciación de los temas sigue un orden lógico y coherente, permitiendo construir el conocimiento de manera progresiva. El ritmo ha sido constante, sin que se perciban saltos abruptos o una sobrecarga de información en momentos puntuales, lo que ha facilitado el seguimiento continuo de la materia.
- **Nivel de dificultad:** Personalmente, he percibido el nivel de dificultad como moderado-bajo (de momento, ya que llevamos pocos temas). Esta percepción está, sin duda, influenciada por mi formación previa en el Grado en Matemáticas, la cual me ha proporcionado una base sólida en los fundamentos estadísticos y de probabilidad que sustentan la materia. No tengo ninguna duda de que se me va a seguir exigiendo durante todo el curso, incluso con un incremento de esfuerzo necesario.
- **Trabajo personal:** El trabajo personal requerido ha sido constante pero manejable. La naturaleza eminentemente práctica del curso, centrada en la resolución de ejercicios en R, exige una dedicación regular para interiorizar no solo la sintaxis del lenguaje, sino, sobre todo, la capacidad de plantear y resolver problemas de forma autónoma. Esta metodología fomenta un aprendizaje activo que considero muy valioso y efectivo para consolidar las competencias de la asignatura.