**Universidade de São Paulo**

**Faculdade de Filosofia, Letras e Ciências Humanas**

**Departamento de Ciência Política**

**FLP0406 – Métodos e Técnicas de Pesquisa em Ciência Política**

**1º semestre / 2018**

**Prof. Glauco Peres da Silva**

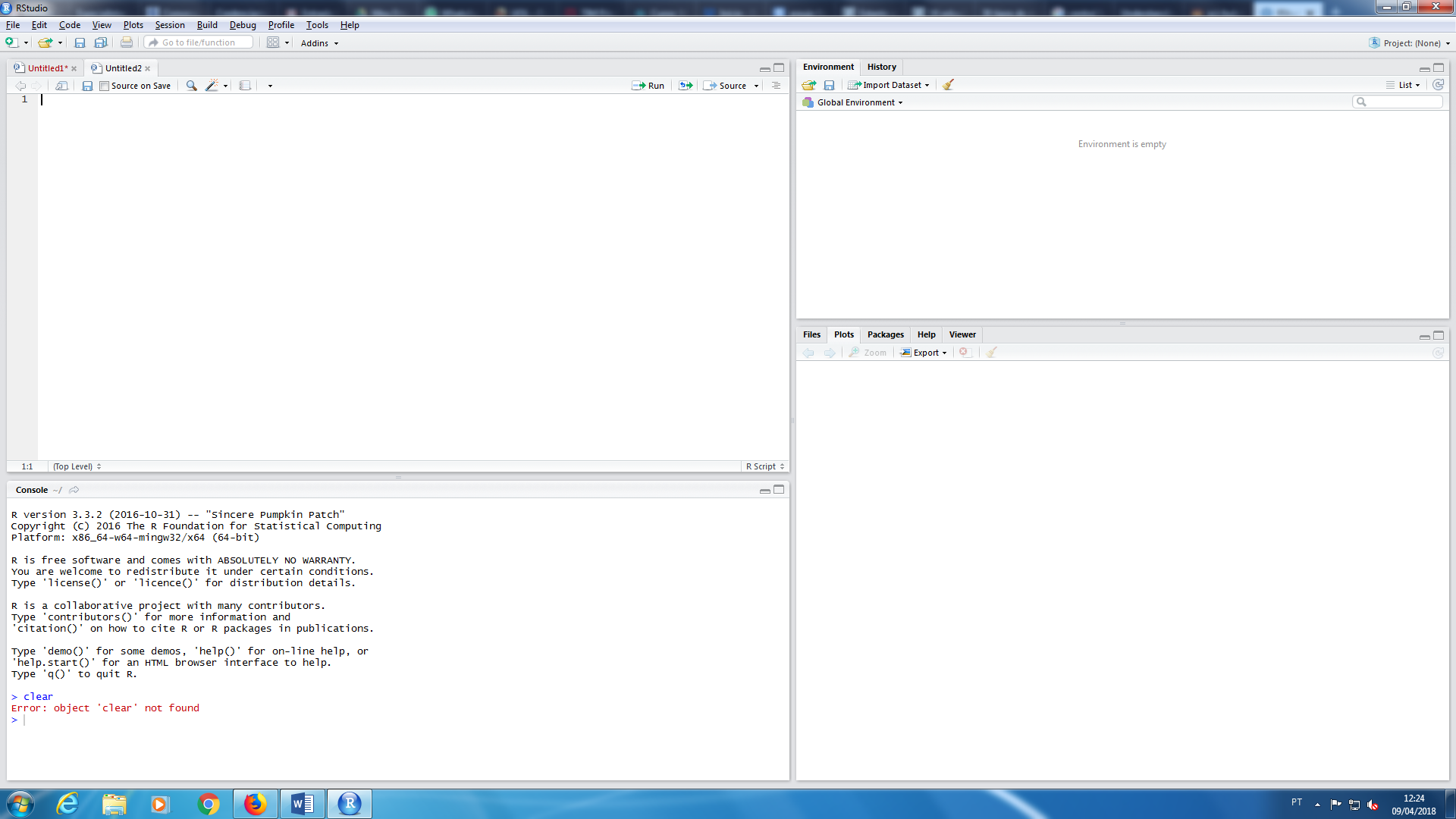
**Lista de Exercícios 5 - Laboratório sobre Teorema do Limite Central**

No laboratório de hoje, tentaremos entender os conceitos relacionados ao Teorema do Limite Central.

Para isso, vamos aplicar simulações utilizando o software R.

Em seu computador, abra o RStudio. O símbolo do software é similar a este aqui: RStudio Logo

Abrirá uma tela como esta abaixo.



A seta vermelha sinaliza o campo onde você escreverá os comandos que serão apresentados a seguir.

Primeiro, serão instalados dois pacotes. Os comandos são os seguintes:

install.packages("ggplot2")

install.packages("knitr")

Depois de instalados, você deve se certificar que o R está configurado corretamente. Para isso, digite:

require(ggplot2)

require(knitr)

Para que as simulações comecem, é preciso dar uma informação original (a “semente”) que será utilizada pelo software para fazer seus cálculos. No comando a seguir, substitua o símbolo # por um número qualquer.

set.seed(#)

Como o teorema do limite central é válido para qualquer distribuição original, mesmo as muito estranhas, vamos criar uma bastante diferente. Vamos juntar duas séries de dados a partir de famílias de distribuição distintas. Para isto, note nos comandos abaixo, as expressões que se referem às distribuições e aos parâmetros delas:

pop1<-rbeta(#, α, β)

pop1 é a variável que você cria; rbeta é um vetor aleatório (random) a partir de uma distribuição beta; # é o número de observações; α, β são os parâmetros da função. Neste link (http://stat.ethz.ch/R-manual/R-devel/library/stats/html/Distributions.html), você pode ver as distribuições existentes no R.

Escolha uma segunda distribuição e crie uma nova série de dados, dessa vez, chamada de pop2. No exemplo abaixo, foi escolhida uma distribuição Poisson:

pop2<-ppois(#, λ)

Os comandos rodados no exemplo que está sendo desenvolvido foram:

pop1<-rbeta(5000, 0.5, 0.1)

pop2<-ppois(5000, 6)

Em seguida, vamos juntar as duas séries em uma série única, chamada pop.

pop <- c(pop1, pop2)

Podemos calcular as estatísticas descritivas da nossa série. Os comandos para calcular a média e o desvio padrão são, já criando variáveis com essas informações, pode digitar algo assim:

mi <- mean(pop)

sigma <- sd(pop)

Faça ambos, pois utilizaremos essas informações depois.

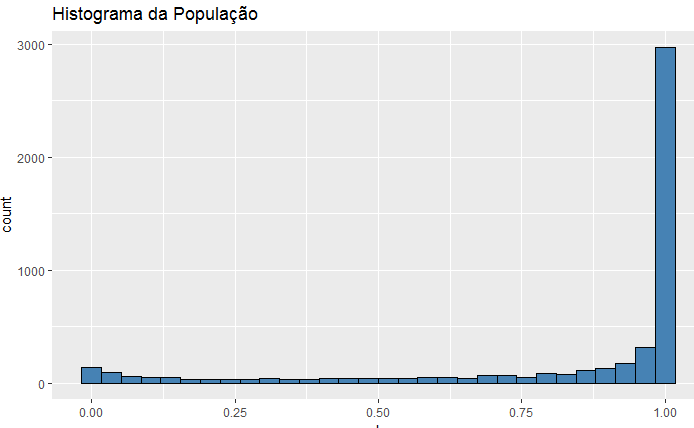
Vamos fazer um histograma de nossa série para identificar o seu comportamento. No caso, os comandos para isso são:

popdf <- as.data.frame(pop)

hg <- ggplot(popdf, aes(x = pop)) + geom\_histogram(colour = "black", fill = "steelblue") + ggtitle("Histograma da População") + xlab("value")

hg

No caso, o histograma ficou assim:



Aqui a média é 0.83 e o desvio padrão, 0.29

Vamos à simulação então.

Inicialmente, vamos criar um vetor em que armazenamos os números de amostras. O comando será esse:

n <- c(1, 5, 10, 30, 50, 100)

Ou seja, faremos simulações com amostras de tamanho 1, 5, 10, etc.

Agora vamos criar um vetor com o número de observações em cada amostra. O comando será esse:

t <- c(10, 100, 1000, 10000)

Criamos um data frame para incluir os resultados que serão simulados com o seguinte comando:

df <- data.frame()

Os comandos para a simulação especificamente são estes aqui[[1]](#footnote-1):

for(i in n) { #para cada n

col <- c()

for(j in t){ #fazemos o loop para cada valor de t

trial <- 1:j

counter <- j #cria um contador para qualquer valor em t que estejamos

value <- c()

while(counter > 0) { #extrai n amostras da população

bucket <- sample(pop, i, replace = TRUE)

xbar <- mean(bucket) #calcular a sua média

value <- c(value, xbar) #e a adiciona a um vetor

counter <- counter - 1 #o contador reduz em 1 até atingir 0

}

sbar <- sd(value) #calcula o desvio padrão da amostra

col <- cbind(trial, value, sbar, i, j) #junta todas as informações

df <- rbind(df, col) #e as adiciona em nossa data frame

} #refaz tudo até terminarmos

}

Um comando para apagar as variáveis criadas e que não serão mais usadas (as chamada de variáveis temporárias) é este:

rm(col, bucket, value, counter, i, j, n, sbar, t, xbar, trial) #apagamos

Como os dados são muito grandes, podemos ver as primeiras 25 linhas. Faríamos o seguinte:

head(df, n = 25)

Estes dados não nos permites ver muita coisa. Podemos criar gráficos mais interessantes com estes dados. Vamos fazer assim:

names(df) <- c("num obs", "valor", "desvio padrão", "obs", "amostras")

g <- ggplot(df, aes(x = valor)) + geom\_density(fill = "steelblue") + facet\_grid(obs ~ amostras, labeller = label\_both) + ggtitle("Demonstrando o Teorema do Limite Central com Simulação") + geom\_vline(xintercept = mi, linetype = "dashed")

O que seu gráfico mostra? Dê uma interpretação para o que você está lendo nele.

Vamos discutir agora o erro padrão do modelo. Vamos inicialmente criar os desvios padrão para em seguida encontrar os erros padrão e verificar, em termos percentuais, o quanto estamos distantes do resultado “correto”. Os comandos são os seguintes:

# criar um data frame dos desvios padrão amostrais simulados

m <- matrix(unique(df$`desvio padrão`), nrow = 4, ncol = 6)

sdf <- as.data.frame(m, row.names = c("t10", "t100", "t1000", "t10000"))

names(sdf) <- c("s1", "s5", "s10", "s30", "s50", "s100")

sdf <- t(sdf) # transpor os resultados para se assemelhar a como organizamos os dados

kable(sdf)

exvals <- sigma/sqrt(c(1, 5, 10, 30, 50, 100)) # Calcula o erro padrão do desvio padrão populacional

dfex <- as.data.frame(exvals, row.names = c("s1", "s5", "s10", "s30", "s50", "s100"))

pexval <- sdf/exvals # Expressa a proximidade em termos percentuais

kable(round(pexval - 1, 3))# Faz a diferença para 1 – daí, quanto mais próximo de zero, maior a evidência de que a simulação forneceu o erro padrão correto.

Qual o resultado do erro padrão estimado? O que você pode dizer sobre isso?

Refaça o exercício começando com outras distribuições. Veja o que acontece com as estimativas finais que serão obtidas. Comente.

1. Na programação no R, o símbolo # permite que você escreva anotações que não são lidas como código. Assim, no código apresentado aqui, o texto que vem logo após serve para explicar o que o comando faz. [↑](#footnote-ref-1)