

O USO DO SOFTWARE R COMO FERRAMENTA DE APOIO À PROBABILIDADE, ESTATÍSTICA E PESQUISA OPERACIONAL EM ENGENHARIA DE PRODUÇÃO

Aluno: André Provenzano Naveiro
Orientador: Fernando Luiz Cyrino Oliveira
Co-orientadora: Paula Medina Maçaira

Introdução

Recentes estudos mostram que a popularidade da linguagem de programação R aumentou substancialmente nos últimos anos, tendo sido cada vez mais utilizado no ambiente acadêmico e corporativo. O software R, homônimo da linguagem, tem sido amplamente utilizado por estatísticos e mineradores de dados, pois possui características fundamentais no mundo da programação: é gratuito, de fácil aprendizado e implementação.

A crescente utilização do R está associada a tarefas tão simples quanto o cálculo da média amostral e também a tarefas complexas como o desenvolvimento de funções que incorporem modelos híbridos de previsão. Mesmo usuários iniciantes podem realizar tarefas complexas de forma simples ao utilizar conjuntos prontos de pacotes que possuem as mais variadas funções com comandos para análise estatística e visualização de dados. Tais pacotes têm como objetivo criar um meio-termo entre o conforto de usar soluções do tipo “caixa-preta” e a dificuldade de se criar códigos de programação.

Haja vista as vantagens e possibilidades de utilização da linguagem R, esta apostila busca introduzir ao leitor ao mundo do software R de forma clara e consistente.

A apostila é constituída de dez seções com material de ensino além de Introdução e Conclusão. As seções são: Baixando o software R, O RStudio, Compreendendo o R, Manipulando os objetos do R, Importar e exportar dados do R, Gráficos, Funções, Estatísticas descritivas, Distribuições de probabilidade e análise de Séries temporais.

Para obter maiores informações sobre o software R, visite a [página](#) do The R Project for Statistical Computing.

Além da apostila, em anexo é possível acessar um artigo aceito para apresentação oral no XLVIII Simpósio Brasileiro de Pesquisa Operacional que acontecerá em Vitória (ES) de 27 a 30 de setembro de 2016. Tal artigo é um dos produtos da Iniciação Científica e foi concebido no âmbito da aplicação do conhecimento adquirido do software R.

Baixando o Software R

Nessa seção será apresentado todo o processo de instalação do Software R, detalhando todo o passo a passo para a realização do mesmo.

O software está disponível de forma gratuita no seu site oficial para as plataformas UNIX, Windows e MacOS. Para fazer seu download basta seguir o passo a passo abaixo:

1. Entre no site oficial do R: <http://www.r-project.org/>
2. Clique em CRAN e selecione o servidor
 - O botão CRAN se encontra na barra lateral esquerda, embaixo de Download CRAN (Figura abaixo). Ao apertar o botão aparecerão todos os servidores, devendo a pessoa escolher o local mais próximo de sua cidade. No caso do Brasil temos servidores em São Paulo, Piracicaba, Rio de Janeiro, Paraná e Santa Cruz.
3. Escolha a plataforma utilizada
 - As opções de plataforma se encontram da parte superior da tabela, podendo escolher dentre as opções de *Download R for Windows*, Linux ou MacOS. Os próximos passos dependem da plataforma escolhida, mas em todas o caminho é bem intuitivo. Como a maior parte das pessoas possuem Windows, continuarei o download nessa plataforma.
4. Clique em base
 - Assim terá acesso para a página em que será baixado o R
5. Escolha a versão mais recente disponível
 - A versão está apresentada no canto esquerdo com nomenclatura R-(Número da versão). Assim, clique na opção *Download R-3.3.0(ou a mais recente) for Windows*. Após esta etapa será feito o download do arquivo.
6. Clique em Salvar
 - O Download do R já foi realizado, mas ainda falta instalar o software.
7. Clique no arquivo e selecione a opção Executar
 - Após ter selecionado, basta escolher o idioma e apertar avançar seguidas vezes, fazendo algumas escolhas pessoais. Tome o cuidado de identificar o local que está sendo instalado.
8. Caso tenha alguma dúvida
 - Visite a página de perguntas frequentes - <http://cran.r-project.org/doc/FAQ/R-FAQ.html>

O RStudio

Baixando o RStudio

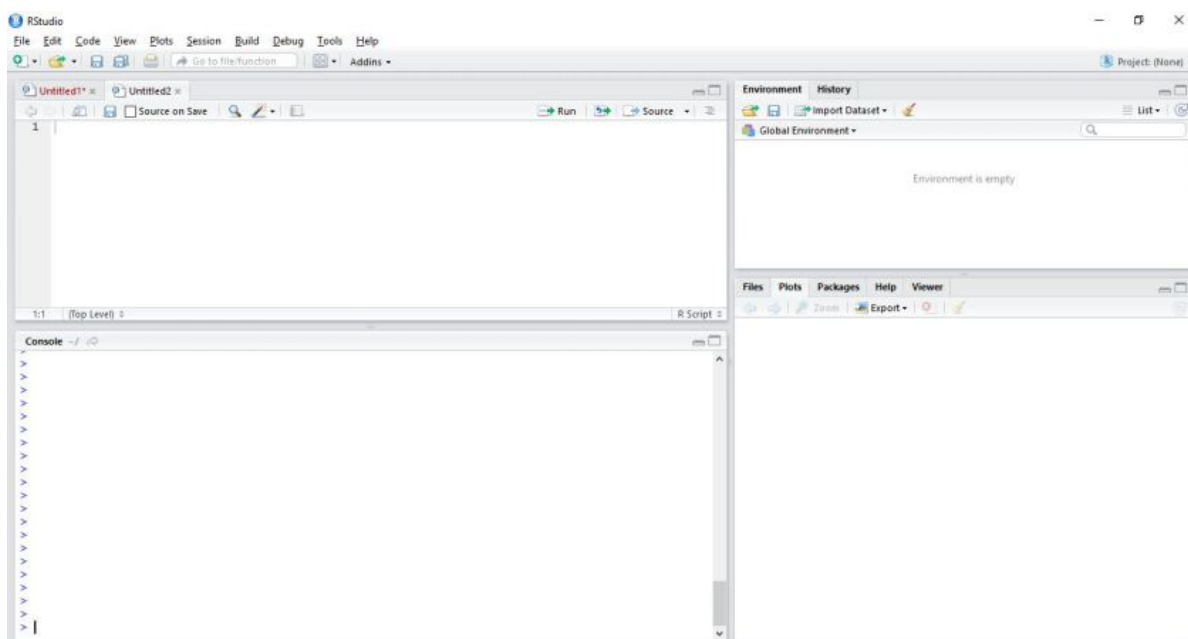
Nessa seção será apresentado todo o processo de instalação do RStudio, detalhando todo o passo a passo para a realização do mesmo. Além disso serão explicados todos os recursos disponíveis na interface do RStudio.

Apesar do R vir com uma interface gráfica, ele pode gerar dificuldades para novos usuários, de modo a ser interessante a instalação do RStudio. O RStudio é um ambiente de desenvolvimento integrado aberto e free para o software R, cujo objetivo é facilitar a integração entre o R e o usuário, sendo uma interface muito mais intuitiva. Dentre suas principais vantagens temos o Highlight - Preto para as funções e objetos, azul para os valores e verde para os textos - o Autocomplete, o match automático de parênteses e chaves, uma interface intuitiva para objetos, gráficos e script, e possibilidade de interação com HTML. Para baixá-lo siga os passos a seguir:

1. Entre no site <https://www.rstudio.com/>
2. Clique na opção Download RStudio localizada no lado esquerdo.
 - O RStudio está disponível no site <https://www.rstudio.com/>. Para fazer seu download clique no menu Products->RStudio; vá em Download RStudio Desktop e selecione seu sistema operacional. É recomendável instalar o RStudio no mesmo diretório em que foi instalado o R e, depois, configurar o RStudio para Executar como Administrador.
3. Clique no ícone Desktop RStudio localizado no canto inferior esquerdo.
4. Clique na opção Download RStudio Desktop também no canto inferior esquerdo.
5. Selecione o link compatível com seu sistema operacional.
6. Instale o arquivo.
 - É interessante que o arquivo seja instalado no mesmo diretório em que foi instalado o Software R.
7. Configure o RStudio para Executar como Administrador.

Interface do RStudio

Após ter completado a instalação de ambos os programas, a interface de trabalho do R será a seguinte :



A interface do RStudio é dividida em 4 janelas básicas:

- No **canto superior esquerdo** está o editor do R, local reservado para digitar os comandos do R.
- No **canto inferior esquerdo** se localiza o console do R, que além de mostrar as saídas numéricas no R, avisa em casos de erro na linguagem, explicando sua causa.
- Já no **canto superior direito** está o workspace e history do documento aberto. Nele temos apresentados os objetos criados e suas configurações, de modo que ao clicar na tabela a direita das variáveis do tipo Data (vetores e matrizes), podemos visualizar seu conteúdo. As demais variáveis já apresentam seus respectivos valores à sua direita.
- Por fim, o **canto inferior direito** apresenta as saídas gráficas do R, possibilitando a exportação das mesmas através do botão *Export*. Nele ainda é possível visualizar os pacotes instalados e obter informações através da função *Help*. Nela podemos tirar dúvidas sobre as funções, visualizando o código do comando, os parâmetros que as funções devem receber, sua utilidade e alguns exemplos.

Criando projeto no RStudio

Para criar um novo Script, selecione no canto superior esquerdo File->New File->R Script ou tecle Ctrl+Shift+N. Após aberta a página e digitado o comando, para executá-lo selecione o botão Run ou tecle Ctrl+R. É possível a criação de vários Scripts, de modo a permitir uma fácil mobilidade para consulta de trabalhos feitos anteriormente.

Compreendendo o R

Funções Básicas do R

O R possui inúmeras funções, dos mais diferentes níveis de complexidade. A tabela a seguir apresenta algumas das funções mais básicas e essenciais para o usuário durante seu trabalho:

Função	Descrição
<code>help('nome da função')</code> ou <code>? 'nome da função'</code>	Pesquisa uma função com nome já conhecido
<code>?? 'nome da função'</code>	Pesquisa uma função que não sabe exatamente o nome
<code>setwd("")</code>	Muda o diretório do trabalho
<code>getwd()</code>	Mostra o diretório do trabalho
<code>ls()</code>	Lista o nome dos objetos criados na sessão atual
<code>dir()</code>	Lista todos os arquivos na pasta de trabalho atual
<code>search()</code>	Lista todos os pacotes carregados
<code>rm('objeto')</code>	Remove o objeto entre parênteses
<code>rm(list=ls(all=TRUE))</code>	Remove todos os objetos, limpando a memória
<code>install.packages('pacote')</code>	Instala um pacote pela primeira vez
<code>library('pacote')</code>	Carrega um pacote ou biblioteca previamente instalado
<code>attach('data frame')</code>	Fixa e reconhece os objetos dentro de um data frame
<code>detach('data frame')</code>	Função que desfaz o comando attach

Algumas dessas funções merecem uma melhor explicação, para esclarecer alguns detalhes de sua utilização.

- A função **help()** é de grande utilidade nas situações em que se tem dúvida no código de algum comando. Ela nos descreve a função sobre a qual se tem dúvida, apresenta os parâmetros que a função espera receber e mostra alguns exemplos de aplicação.
- A função **setwd()** serve para definir o diretório, de modo que se deve colocar entre as aspas o caminho do seu computador até o diretório. Para isso clique no diretório com o botão direito e selecione a opção **propriedades** - aqui é possível visualizar o endereço do diretório. Entretanto, para que a função funcione no R, é necessário acrescentar mais uma barra ou trocar todas as barras por contra-barras. O endereço no comando `setwd` deve estar na forma: `C://Users//Diretorio`.
- A função **install.packages()** é de grande importância pelo fato de existir um enorme universo de funções que já foram previamente programadas no R, disponíveis em alguns pacotes especiais. Ao instalar o R, apenas as configurações mínimas para seu funcionamento básico são instaladas - os pacotes que vem na instalação base - Para realizar tarefas mais complicadas pode ser necessário instalar pacotes adicionais (packages), sendo necessário o uso da função `install.packages`. Por exemplo, para usar as funções de cálculo da assimetria e curtose é necessário que se instale e carregue o pacote "moments". Assim, seria usado o comando `install.packages ("moments")`.

Funções Matemáticas do R

Apesar de ser uma forma bem restrita de utilização do R, ele pode servir de calculadora para contas mais simples. Quando se trata de grandes bases de dados, torna-se inviável

realizar os cálculos digitando manualmente cada um dos elementos da base. A tabela a seguir apresenta as principais funções matemáticas realizada para todos os objetos no conjunto de dados especificados.

Função	Descrição
sum('objeto') ou +	Soma de todos os objetos
prod('objeto') ou *	Produto de todos elementos do objeto
sqrt('objeto')	Raiz quadrada dos objetos
abs('objeto')	Valor absoluto dos objetos
exp('objeto')	Exponencial dos objetos
log('objeto')	Logaritmo na base neperiana dos objetos

Objetos do R

Os objetos de manipulação do R são: vetores, matrizes, array, data frames, listas e valores faltantes. A utilização deles se diferencia de acordo com as proporções e o tipo das entradas que serão armazenadas no objeto. Segue, a seguir, uma tabela explicando cada um dos objetos. Vale a pena destacar que os data frames são essenciais em trabalhos mais elaborados, pelo fato desses lidarem com grandes bases de dados, com elementos de diferentes naturezas. Já os valores faltantes são necessários em situações em que temos casas não preenchidas. Por exemplo, se tivéssemos uma base de dados que apresentasse uma variável com informações anuais a partir do ano 2000 e outra variável com informações a partir de 2010. Para trabalharmos com as duas variáveis ao mesmo tempo, seria necessário completar as lacunas da segunda variável dos anos 2000 a 2009 com NA ("Not Available").

Função	Descrição	Comando
Vetores	Conjunto de elementos de uma mesma natureza	c('dados')
Matrizes	Conjunto de elementos de uma mesma natureza organizado em linhas e colunas	matrix('dados', n° linhas, n° colunas)
Array	Conjunto de elementos de um número arbitrário de dimensões	array('dados',dimensão)
Data frames	Matrizes que admitem elementos de naturezas diferentes por coluna	data.frame('dados')
Listas	Generalizações de vetores, representa uma coleção de objetos	list('dados')
Valores faltantes e especiais	Caracteres especiais para definir dados faltantes e outros	NA(Not Available), Nan(Not a Number), -inf e inf (menos e mais infinito)

Manipulando os objetos do R

Nessa seção será ensinado, através de exemplos práticos, como criar e manipular os diferentes objetos do R. Para criação dos mesmos, as funções a seguir podem servir de ferramentas. Para a manipulação, temos uma gama extensa de funções específicas de cada objeto, destacadas a seguir.

1. **c('numero1','número2')** - concatena o primeiro elemento no segundo.

2. **seq('numero inicial','número final','saltos')** - cria sequências de acordo com o padrão definido.
3. **rep('objeto a ser replicado','número de réplicas')** - replicação do objeto definido n vezes.
4. **gl(i,r)** - cria um vetor representado pela sequência de 1 até i, com r repetições em cada nível.
5. **rbind('objeto a ser concatenado','objeto principal'),cbind('objeto a ser concatenado','objeto principal')** - concatena linhas/colunas à matrizes já existentes.

Vetores

- Criando vetores:

Vetor é uma estrutura unidimensional, em que os valores são armazenados na memória em sequência, um após o outro, que permite acessar livremente qualquer valor do conjunto.

Vamos usar o comando **c()** para criar um vetor com os números 1, 2 e 10.

```
c(1, 2, 10)
## [1] 1 2 10
```

Ainda com o comando **c()**, agora será criado um vetor com os números de 1 a 10.

```
c(1:10)
## [1] 1 2 3 4 5 6 7 8 9 10
```

O comando **seq()** será usado para criar um vetor com a sequência de números de 0 a 10, 2 a 2.

```
seq(0, 10, 2)
## [1] 0 2 4 6 8 10
```

Agora o mesmo comando é usado para criar um vetor com a sequência de números de 0 a 10, 1 a 1.

```
seq(0, 10, 1)
## [1] 0 1 2 3 4 5 6 7 8 9 10
```

Ainda com o comando **seq()** só que sem especificar o início e o espaçamento é criado um vetor com a sequência 1 a 1 de números de 1 até 10 nesse caso.

```
seq(10)
## [1] 1 2 3 4 5 6 7 8 9 10
```

O comando **rep()** está sendo usado para criar um vetor com a replicação do número 1, 10 vezes.

```
rep(1, 10)
## [1] 1 1 1 1 1 1 1 1 1 1
```

Ainda com o **rep()** está sendo criado um vetor com a replicação da sequência de 1 a 10, 2 vezes.

```
rep(1:10, 2)
## [1] 1 2 3 4 5 6 7 8 9 10 1 2 3 4 5 6 7 8 9 10
```

Ainda é possível criar um vetor com a replicação do caractere NA, nesse caso 10 vezes.

```
rep(NA, 10)
## [1] NA NA NA NA NA NA NA NA NA NA
```

- Operacionalizando vetores:

Para demonstrar as operações e manipulações possíveis com vetores está sendo criado um vetor com os número 1, 5 e os inteiros de 7 a 15. A esse vetor foi dado o nome de **vetor1**.

```
vetor1 = c(1, 5, 7:15)
vetor1
## [1] 1 5 7 8 9 10 11 12 13 14 15
```

Para obter o comprimento do **vetor1**(número de elementos), pode-se usar o comando **length()**.

```
length(vetor1)
## [1] 11
```

Para orientar o vetor anteriormente criado pode-se usar o comando **sort()**.

```
sort(vetor1)
## [1] 1 5 7 8 9 10 11 12 13 14 15
```

Para retornar as posições dos elementos do **vetor1** pode-se usar o comando **rank()**.

```
rank(vetor1)
## [1] 1 2 3 4 5 6 7 8 9 10 11
```

O comando **max()** é responsável por retornar o valor máximo do **vetor1**.

```
max(vetor1)
## [1] 15
```

Enquanto que o comando **min()** retorna o valor mínimo do **vetor1**.


```
min(vetor1)
## [1] 1
```

O comando **which.max()** é responsável por retornar a posição do valor máximo do vetor1.

```
which.max(vetor1)
## [1] 11
```

O comando **which.min()** é responsável por retornar a posição do valor mínimo do vetor1.

```
which.min(vetor1)
## [1] 1
```

Utilizando colchetes, pode-se obter uma parte de interesse do vetor. Nesse caso é retornado o elemento da posição 2 do vetor1

```
vetor1[2]
## [1] 5
```

Utilizando o sinal de menos dentro do colchete, “[−2]” no caso, é retornado o vetor1 sem o elemento da posição 2

```
vetor1[-2]
## [1] 1 7 8 9 10 11 12 13 14 15
```

Matrizes

- Criando matrizes:

Matriz é uma estrutura bidimensional, em que os valores são armazenados na memória do em linhas e colunas, que permite acessar livremente qualquer valor do conjunto.

O comando usado para criação de matriz é o **matrix()**. Para exemplificar sua aplicação, será criado uma matriz com a sequência dos números de 1 a 10, com 2 linhas e 5 colunas.

```
matrix(data = 1:10, nrow = 2, ncol = 5)
##      [,1] [,2] [,3] [,4] [,5]
## [1,]  1   3   5   7   9
## [2,]  2   4   6   8  10
```

Ainda usando o comando **matrix()**, será criada uma matriz com a sequência dos números de 1 a 10, com 2 linhas e 5 colunas nomeando as linhas e as colunas.

```
matrix(data = 1:10, nrow = 2, ncol = 5, dimnames = list(c("L1", "L2"), c("C1",  
  "C2", "C3", "C4", "C5")))  
##      C1 C2 C3 C4 C5  
## L1   1  3  5  7  9  
## L2   2  4  6  8 10
```

É possível criar a mesma matriz do exemplo anterior, sem definir na linha de comando os dados e o número de colunas.

```
matrix(data = 1:10, 2, 5, dimnames = list(c("L1", "L2"), c("C1", "C2", "C3",  
  "C4", "C5")))  
##      C1 C2 C3 C4 C5  
## L1   1  3  5  7  9  
## L2   2  4  6  8 10
```

Denominando a matriz anterior de Matriz1 e a chamando para que seja visualizada.

```
Matriz1 = matrix(data = 1:10, 2, 5, dimnames = list(c("L1", "L2"), c("C1", "C2",  
  "C3", "C4", "C5")))  
Matriz1  
##      C1 C2 C3 C4 C5  
## L1   1  3  5  7  9  
## L2   2  4  6  8 10
```

Usando o comando **cbind()** pode-se adicionar uma nova coluna de elementos à Matriz1, no caso, um coluna de NA.

```
cbind(Matriz1, c(NA, NA))  
##      C1 C2 C3 C4 C5  
## L1   1  3  5  7  9 NA  
## L2   2  4  6  8 10 NA
```

Usando o comando **rbind()** pode-se adicionar uma nova linha de elementos NA à Matriz1.

```
rbind(Matriz1, rep(NA))  
##      C1 C2 C3 C4 C5  
## L1   1  3  5  7  9  
## L2   2  4  6  8 10  
##      NA NA NA NA NA
```

- Operacionalizando Matrizes:

Para demonstrar as operações e manipulações possíveis com matrizes, será criado a matriz1, composta por 6 números.

```
matriz1 = matrix(c(10, 5, 25, 60, 30, 10, 8, 9, 21), nrow = 3)
matriz1
##      [,1] [,2] [,3]
## [1,]  10  60   8
## [2,]   5  30   9
## [3,]  25  10  21
```

Para obter as dimensões da matriz1 (3 linhas e 3 colunas), pode-se usar o comando **dim()** .

```
dim(matriz1)
## [1] 3 3
```

Utilizando colchetes é possível selecionar o elemento de interesse da matriz. Nesse caso está sendo selecionado o elemento da linha 2 e coluna 3 da matriz1.

```
matriz1[2, 3]
## [1] 9
```

Ainda utilizando colchetes quando se deixa uma das especificações em branco (da linha ou coluna), está sendo selecionado todos os elementos da coluna/linha especificada. No exemplo estão sendo selecionados os elementos da coluna 1 da matriz1.

```
matriz1[, 1]
## [1] 10  5 25
```

Agora com colchetes, estão sendo selecionados os elementos da linha 2 da matriz1.

```
matriz1[2, ]
## [1]  5 30  9
```

Pode-se usar o comando **det()** para calcular o determinante da matriz1.

```
det(matriz1)
## [1] 7000
```

Já o comando **solve()** é usado para calcular a inversa da matriz1.

```
solve(matriz1)
##      [,1]      [,2]      [,3]
## [1,] 0.07714286 -0.168571429  0.042857143
## [2,] 0.01714286  0.001428571 -0.007142857
## [3,] -0.10000000  0.200000000  0.000000000
```

Com colchete pode-se redefinir os elementos da matriz. Nesse caso está sendo alterado o elemento da 2 linha e 3 coluna da matriz1 para NA.

```
matriz1[2, 3] = NA
matriz1
##      [,1] [,2] [,3]
## [1,]   10   60    8
## [2,]    5   30   NA
## [3,]   25   10   21
```

Agora redefinindo todos elementos da 2 coluna da matriz1 para NA.

```
matriz1[, 2] = NA
matriz1
##      [,1] [,2] [,3]
## [1,]   10   NA    8
## [2,]    5   NA   NA
## [3,]   25   NA   21
```

Array

- Criando Array:

Array é uma estrutura com um conjunto de elementos de um número arbitrário de dimensões , em que os valores são armazenados na memória em sequência, um após o outro, e pode-se livremente acessar qualquer valor do conjunto.

O comando utilizado na criação de um array é o **array()**. No exemplo está sendo criado um array dos números de 1 a 10, com 3 linhas, 4 colunas e duas matrizes

```
array(1:10, c(3, 4, 2))
## , , 1
##
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8    1
## [3,]    3    6    9    2
##
## , , 2
##
##      [,1] [,2] [,3] [,4]
## [1,]    3    6    9    2
## [2,]    4    7   10    3
## [3,]    5    8    1    4
```

A operacionalização de Array é igual a de matrizes.

Data Frame

- Criando Data frame:

O data frame é similar a uma matriz, mas tem a vantagem de admitir elementos de diferentes naturezas.

O comando utilizado para criar um data frame é **data.frame()**. A seguir um exemplo da aplicação desse comando para criar um data frame com o vetor com os números 20, 23 e 30, vetor de letras A, B, C e a repetição do 1 3 vezes.

```
data.frame(c(20, 23, 30), c("A", "B", "C"), rep(1, 3))
##      c.20..23..30. c..A....B....C.. rep.1..3.
## 1             20             A             1
## 2             23             B             1
## 3             30             C             1
```

Nomeando o data frame anterior de Data_frame1

```
Data_frame1 = data.frame(c(20, 23, 30), c("A", "B", "C"), rep(1, 3))
```

Usando o comando **names()** para dar nome as coluna do Data_frame1 de D1, D2, D3 e o chamando na linha de comando.

```
names(Data_frame1) = c("D1", "D2", "D3")
Data_frame1
##   D1 D2 D3
## 1 20  A  1
## 2 23  B  1
## 3 30  C  1
```

Novamente designando o nome das colunas, só que dessa vez usando apenas o comando **data.frame()**.

```
data.frame(D1 = c(20, 23, 30), D2 = c("A", "B", "C"), D3 = rep(1, 3))
##   D1 D2 D3
## 1 20  A  1
## 2 23  B  1
## 3 30  C  1
```

- Operacionalizando Data Frames:

Para demonstrar as operações e manipulações possíveis com data frames, será criado o Data1, composta pelos vetores id, idade e gênero.

```
Data1 = data.frame(id = c(1:10), idade = c(16, 18, 19, 20, 22, 21, 23, 16, 17, 18), genero = c("M", "F", "M", "F", "M", "F", "M", "F", "M", "F"))
```

Abaixo visualiza-se a base Data1

```
Data1
##      id idade genero
## 1     1    16      M
## 2     2    18      F
## 3     3    19      M
## 4     4    20      F
## 5     5    22      M
## 6     6    21      F
## 7     7    23      M
## 8     8    16      F
## 9     9    17      M
## 10    10    18      F
```

Também está sendo criado outro data frame com os elementos id e altura e sendo nomeado de Data2.

```
Data2 = data.frame(id = c(1:10), altura = c(1.6, 1.82, 1.9, 1.52, 2, 1.76, 1.63, 1.6, 1.7, 1.8))
```

```
Data2
##      id altura
## 1     1   1.60
## 2     2   1.82
## 3     3   1.90
## 4     4   1.52
## 5     5   2.00
## 6     6   1.76
## 7     7   1.63
## 8     8   1.60
## 9     9   1.70
## 10    10   1.80
```

Usando \$ pode-se especificar o conjunto de elementos desejado. Por exemplo pode-se chamar os elementos da coluna “idade” de Data1.

```
Data1$idade
## [1] 16 18 19 20 22 21 23 16 17 18
```

Com o mesmo comando está sendo chamado os elementos da coluna “altura” da Data2.

```
Data2$altura
## [1] 1.60 1.82 1.90 1.52 2.00 1.76 1.63 1.60 1.70 1.80
```

Utilizando o comando **attach()** pode-se fixar o data frame Data1 na workspace.

```
attach(Data1)
```

Ao chamar o elemento idade, será apresentado os valores da coluna idade do Data1 já que ela está fixada. Se tentássemos o mesmo com gênero -coluna do Data2- daria erro pelo fato do Data2 não estar fixado.

```
idade
## [1] 16 18 19 20 22 21 23 16 17 18
```

Utilizando o comando **detach()** pode-se desanexar o data frame Data 1 da workspace.

```
detach(Data1)
```

Com o comando **merge()** pode-se unir o Data1 e Data2 a partir da coluna de interesse, por exemplo a coluna “id”. Denominou-se o novo data frame de Data3.

```
Data3 = merge(Data1, Data2, by = "id")
Data3
##      id idade genero altura
## 1     1    16      M    1.60
## 2     2    18      F    1.82
## 3     3    19      M    1.90
## 4     4    20      F    1.52
## 5     5    22      M    2.00
## 6     6    21      F    1.76
## 7     7    23      M    1.63
## 8     8    16      F    1.60
## 9     9    17      M    1.70
## 10    10    18      F    1.80
```

Uma função muito útil para data frames é a **summary()**, dado que ela produz diversas estatísticas sobre a base de dados, como pode ser visto no exemplo a seguir:

```
summary(Data3)
##           id           idade      genero      altura
## Min.      : 1.00   Min.    :16.00   F:5     Min.    :1.520
## 1st Qu.: 3.25   1st Qu.:17.25   M:5     1st Qu.:1.607
## Median : 5.50   Median :18.50                Median :1.730
## Mean    : 5.50   Mean    :19.00                Mean    :1.733
## 3rd Qu.: 7.75   3rd Qu.:20.75                3rd Qu.:1.815
## Max.    :10.00   Max.    :23.00                Max.    :2.000
```

Lista

- Criando Lista:

Lista é uma estrutura que representa uma coleção de objetos. Pode englobar vetores, matrizes arrays e data frames. O comando usado para criar uma lista é **list()**.

Será usado o comando **list()** para criar uma lista com 4 componentes, sendo a primeira dos elementos de 101 a 99, a segunda a repetição do elemento NA 10 vezes, a terceira as letras de “a” a “e” e a quarta a sequência de 1 a 10, de 3 em 3.

```
lista1 = list(c(101:99), rep(NA, 10), letters[1:5], seq(1, 10, 3))
lista1
## [[1]]
## [1] 101 100 99
##
## [[2]]
## [1] NA NA NA NA NA NA NA NA NA NA
##
## [[3]]
## [1] "a" "b" "c" "d" "e"
##
## [[4]]
## [1] 1 4 7 10
```

Usando duplo colchetes para chamar o componente 3 da lista

```
lista1[[3]]
## [1] "a" "b" "c" "d" "e"
```

Usando o comando **names()** para nomear as componentes da lista

```
names(lista1) = c("L1", "L2", "L3", "L4")
lista1
## $L1
## [1] 101 100 99
##
## $L2
## [1] NA NA NA NA NA NA NA NA NA NA
##
## $L3
## [1] "a" "b" "c" "d" "e"
##
## $L4
## [1] 1 4 7 10
```

- Operacionalizando Listas:

Para demonstrar as operações e manipulações possíveis com listas será utilizada a lista1, criada anteriormente.

Vamos usar o comando **length()** para obter a quantidade de componentes na lista1.

```
length(lista1)
## [1] 4
```

Usando o comando **list()** será criada uma nova lista

```
lista2 = list("Carlos", "Joao", "Antonio")
lista2
## [[1]]
## [1] "Carlos"
##
## [[2]]
## [1] "Joao"
##
## [[3]]
## [1] "Antonio"
```

Através do comando **c()**, pode-se concatenar a lista1 e a lista2

```
c(lista1, lista2)
## $L1
## [1] 101 100 99
##
## $L2
## [1] NA NA NA NA NA NA NA NA NA
##
## $L3
## [1] "a" "b" "c" "d" "e"
##
## $L4
## [1] 1 4 7 10
##
## [[5]]
## [1] "Carlos"
##
## [[6]]
## [1] "Joao"
##
## [[7]]
## [1] "Antonio"
```

Com o comando **append()** pode-se especificar o local que a lista2 será incluída. Nesse caso está se acrescentando o conteúdo da lista2 após o segundo componente da lista1.

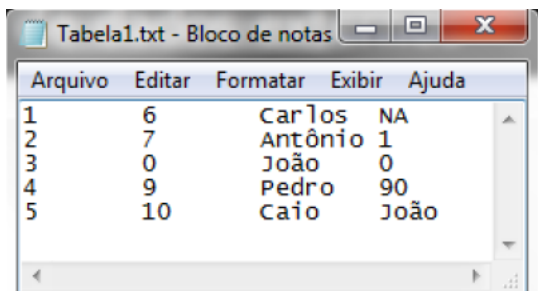
```
append(lista1, lista2, after = 2)
## $L1
## [1] 101 100 99
##
## $L2
## [1] NA NA NA NA NA NA NA NA NA
##
## [[3]]
## [1] "Carlos"
##
## [[4]]
## [1] "Joao"
##
## [[5]]
## [1] "Antonio"
##
## $L3
## [1] "a" "b" "c" "d" "e"
##
## $L4
## [1] 1 4 7 10
```

Importar e exportar dados do R

Importação

Para importar dados externos no R, temos diversas maneiras, dependendo do formato da base de dados a ser lida. Desse modo, antes de tudo, devemos verificar o conteúdo do arquivo a ser lido e o modo como os seus elementos estão separados. Dentre os formatos, os mais utilizadas são o txt e csv, explicados a seguir.

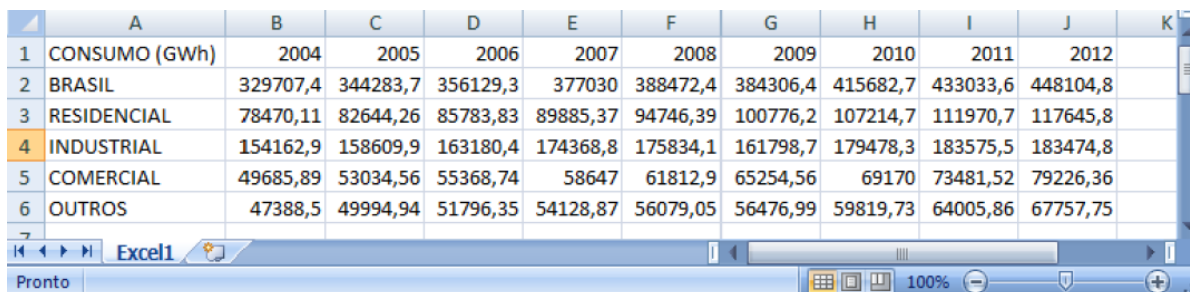
A função **read.table** é usada para ler os dados de um arquivo de texto, armazenando-o no formato de data frame. Nele é possível especificar um argumento de cabeçalho, separadores e forma de lidar com valores em falta e não preenchidos ou linhas em branco. Assim, a função é utilizada deste modo: **read.table("NomeDoArquivo.txt")**



Arquivo	Editar	Formatar	Exibir	Ajuda
1	6	Carlos	NA	
2	7	Antônio	1	
3	0	João	0	
4	9	Pedro	90	
5	10	Caio	João	

```
read.table(Tabela1.txt)
```

A função **read.csv** é usada para ler dados em arquivos com variáveis separadas por vírgula (CSV), armazenando-os como data frame. Sua grande utilidade se dá pelo fato dela conseguir ler dados salvos pelo Excel, um dos softwares mais utilizados para armazenamento de grandes bases de dados. Um comando similar, também usado, é o **read.csv2**, que serve para ler tabelas já formatadas sem cabeçalho, com separador e com casas decimais separadas por pontos. A função é utilizada deste modo: **read.csv("NomeDoArquivo.csv")**



	A	B	C	D	E	F	G	H	I	J	K
1	CONSUMO (GWh)	2004	2005	2006	2007	2008	2009	2010	2011	2012	
2	BRASIL	329707,4	344283,7	356129,3	377030	388472,4	384306,4	415682,7	433033,6	448104,8	
3	RESIDENCIAL	78470,11	82644,26	85783,83	89885,37	94746,39	100776,2	107214,7	111970,7	117645,8	
4	INDUSTRIAL	154162,9	158609,9	163180,4	174368,8	175834,1	161798,7	179478,3	183575,5	183474,8	
5	COMERCIAL	49685,89	53034,56	55368,74	58647	61812,9	65254,56	69170	73481,52	79226,36	
6	OUTROS	47388,5	49994,94	51796,35	54128,87	56079,05	56476,99	59819,73	64005,86	67757,75	

```
read.csv2(Excel1.csv)
```

Exportação

Após a criação e manipulação dos objetos no R, muitas vezes é de interesse exportá-los para fora do R. Para isso, existem funções que devem ser escolhidas de acordo com o formato da base de dados que se pretende gerar. Dentre os formatos, os mais utilizadas são o txt e csv, explicados a seguir.

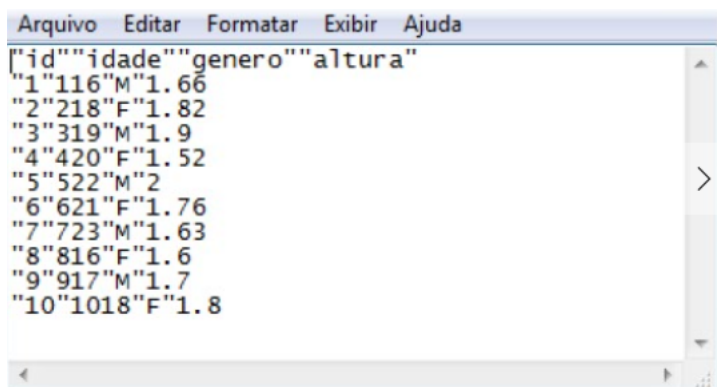
Temos a função **write.table()**, que exporta os objetos trabalhados no R no formato de texto (.txt), podendo ser visualizado no Bloco de Notas.

```
write.table(NomeDoObjeto,NomeDoArquivoGerado.txt)
```

Data3

```
##      id idade genero altura
## 1     1    16      M    1.60
## 2     2    18      F    1.82
## 3     3    19      M    1.90
## 4     4    20      F    1.52
## 5     5    22      M    2.00
## 6     6    21      F    1.76
## 7     7    23      M    1.63
## 8     8    16      F    1.60
## 9     9    17      M    1.70
## 10    10    18      F    1.80
```

```
write.table(Data3, "Data3.txt")
```




```
"id","idade","genero","altura"
"1","16","M","1.66"
"2","18","F","1.82"
"3","19","M","1.9"
"4","20","F","1.52"
"5","22","M","2"
"6","21","F","1.76"
"7","23","M","1.63"
"8","16","F","1.6"
"9","17","M","1.7"
"10","18","F","1.8"
```

Já a função **write.csv()**, serve para exportar os objetos e salvá-los na forma de planilha (.csv) no excell. Um comando similar também usado é o **write.csv2()**, utilizado para escrever arquivos formatados com separadores e casas decimais separadas por ponto. **write.csv(NomeDoObjeto,NomeDoArquivoGerado.txt)**

Data3

```
##      id idade genero altura
## 1     1    16      M    1.60
## 2     2    18      F    1.82
## 3     3    19      M    1.90
## 4     4    20      F    1.52
## 5     5    22      M    2.00
## 6     6    21      F    1.76
## 7     7    23      M    1.63
## 8     8    16      F    1.60
## 9     9    17      M    1.70
## 10    10    18      F    1.80
```

```
write.csv(Data3, "Data3.csv")
```



	A	B	C	D	E	F	G	H
1		id	idade	genero	altura			
2		1	16	M	1,66			
3		2	18	F	1,82			
4		3	19	M	1,9			
5		4	20	F	1,52			
6		5	22	M	2			
7		6	21	F	1,76			
8		7	23	M	1,63			
9		8	16	F	1,6			
10		9	17	M	1,7			
11		10	18	F	1,8			
12								

Gráficos

O R apresenta uma gama extensa de funções gráficas, sendo assim uma poderosa ferramenta de trabalho. Isso pelo fato dos gráficos permitirem uma rápida visualização da base de dados, indicando padrões, acontecimentos específicos e singularidades, permitindo, assim, uma análise muito mais completa. Cada um dos gráficos apresenta uma aplicabilidade diferente, de modo que conhecê-los é fundamental para uma correta utilização e leitura. Nessa seção serão apresentados diversos tipos de gráficos, a customização dos mesmos e como salvá-los em seu computador, após concluídos.

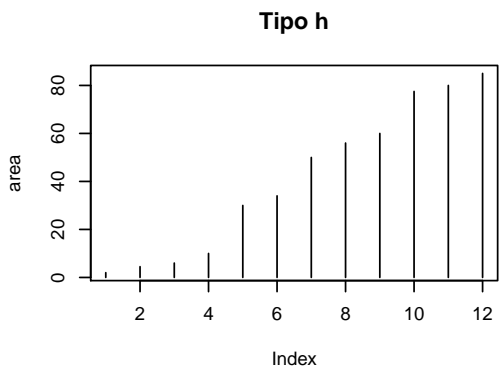
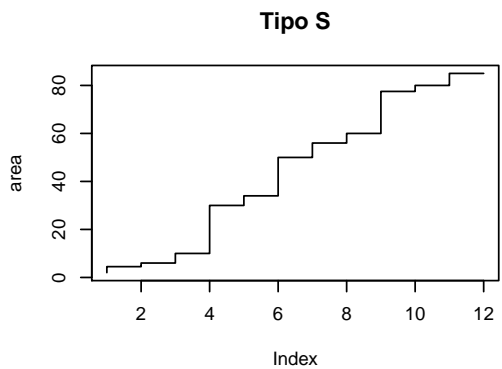
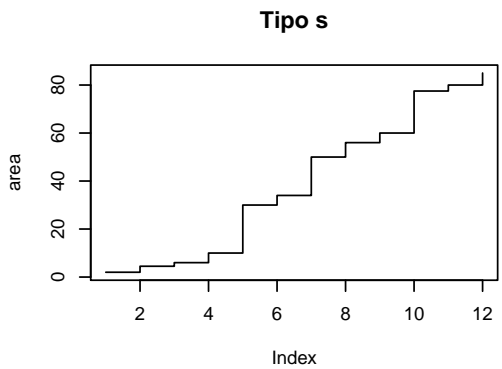
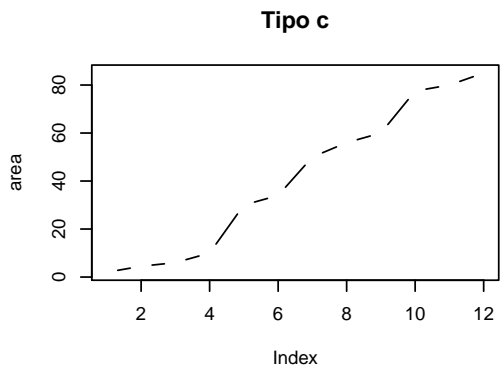
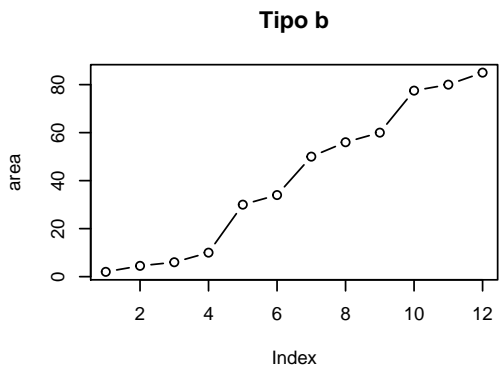
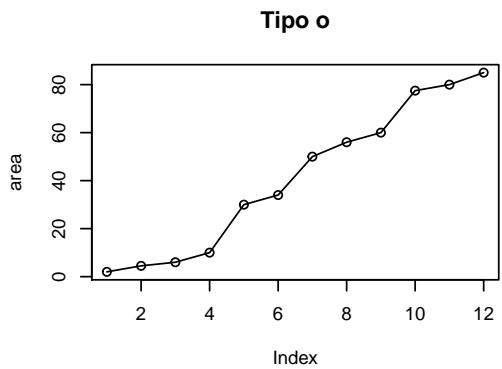
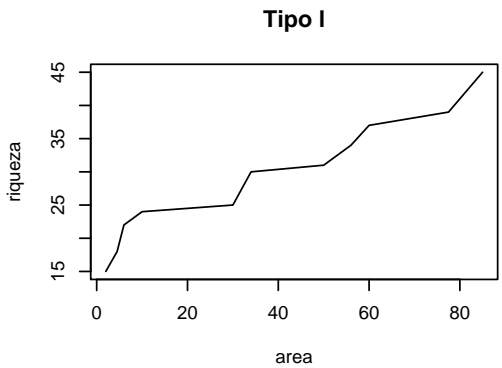
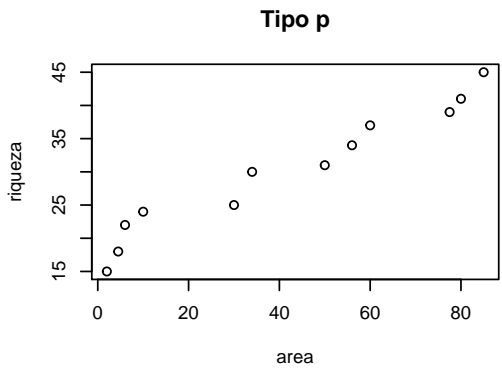
Gráfico de linhas/pontos

O gráfico de linha é um tipo de gráfico que exibe informações (somente variáveis numéricas) com uma série de pontos de dados chamados de marcadores. Estes marcadores podem aparecer ligados ou não por um linha. Esse tipo de gráfico é muitas vezes usado para visualizar uma tendência nos dados em intervalos de tempo. Sua função no R é: **plot(x,y)** ou **plot(x ~ y)**. A variável x será a correspondente ao eixo horizontal e a y ao eixo vertical. Temos diversos tipos desse gráfico, seja pela ligação ou não dos marcadores, ou pelo tipo de ligação.

Para exemplificar alguns tipos de plot estão sendo criados os vetores riqueza e area, compostos somente por números.

```
riqueza = c(15, 18, 22, 24, 25, 30, 31, 34, 37, 39, 41, 45)
area = c(2, 4.5, 6, 10, 30, 34, 50, 56, 60, 77.5, 80, 85)
```

```
par(mfrow = c(4, 2))
plot(area, riqueza, type = "p", main = "Tipo p")
plot(riqueza ~ area, type = "l", main = "Tipo l")
plot(area, type = "o", main = "Tipo o")
plot(area, type = "b", main = "Tipo b")
plot(area, type = "c", main = "Tipo c")
plot(area, type = "s", main = "Tipo s")
plot(area, type = "S", main = "Tipo S")
plot(area, type = "h", main = "Tipo h")
```



```
par(mfrow = c(1, 1))
```

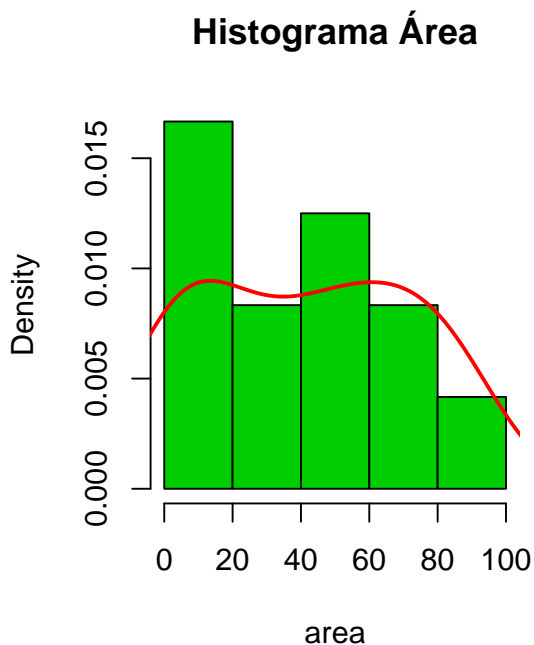
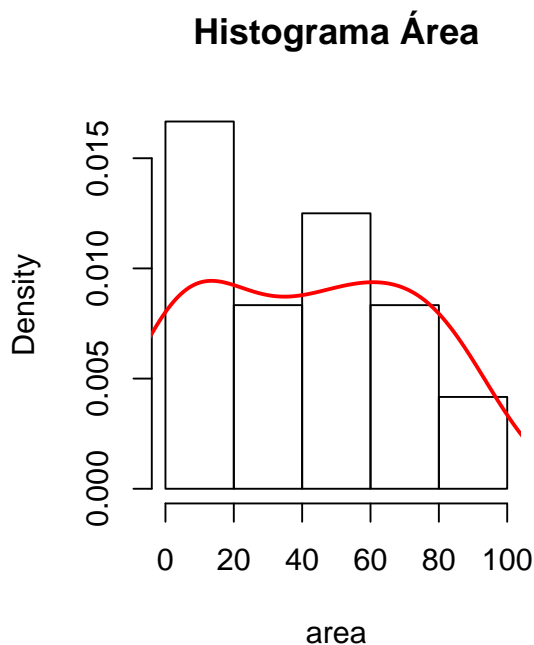
Histograma

O histograma, também conhecido como distribuição de frequências ou diagrama das frequências, é a representação gráfica, em colunas (retângulos), de um conjunto de dados previamente tabulado e dividido em classes igualmente espaçadas. A base de cada retângulo representa uma classe e a altura de cada retângulo representa a quantidade ou frequência com que o valor dessa classe ocorreu no conjunto de dados. A função no R para produzir um histograma é **hist(BaseDeDados)**.

Uma função que pode ser interessante de ser utilizada junto do histograma é a de linhas de densidade. Com ela podemos observar a distribuição de probabilidade do histograma no formato de curva. Seu código é: **lines(density(BaseDeDados))**.

A seguir, exemplo de aplicação das duas funções juntas, na base de dados já utilizada anteriormente:

```
par(mfrow = c(1, 2))  
hist(area, main = "Histograma Área", probability = T)  
lines(density(area), col = 2, lwd = 2)  
hist(area, col = 3, main = "Histograma Área", probability = T)  
lines(density(area), col = 2, lwd = 2)
```



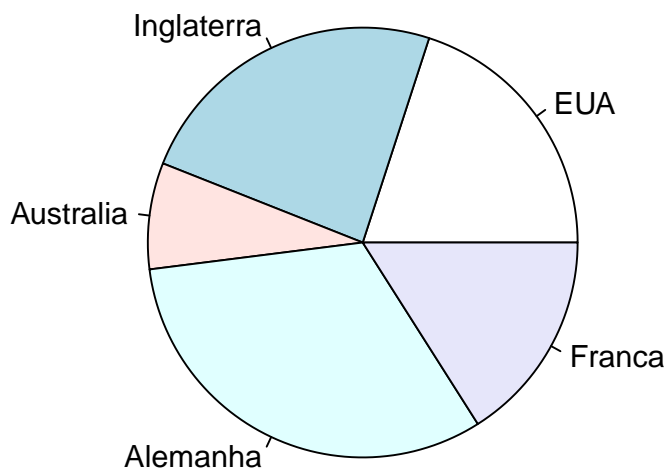
```
par(mfrow = c(1, 1))
```

Gráfico de Pizza

O gráfico de pizza, também conhecido como gráfico de setores ou gráfico circular, é um diagrama circular onde os valores de cada categoria estatística representada são proporcionais às respectivas frequências. Assim, ele é uma ferramenta eficaz para representar a fração de um todo que cada variável representa. Sua formula no R é **pie(x; labels =)**. Para utilizá-lo, deve-se ter como entrada da função dois vetores - um com os valores de cada uma das variáveis e outro com os nomes das variáveis -

A seguir, um exemplo em que *pedacos* é um vetor com as proporções de cada país e *nomes* é um vetor com os nomes dos países:

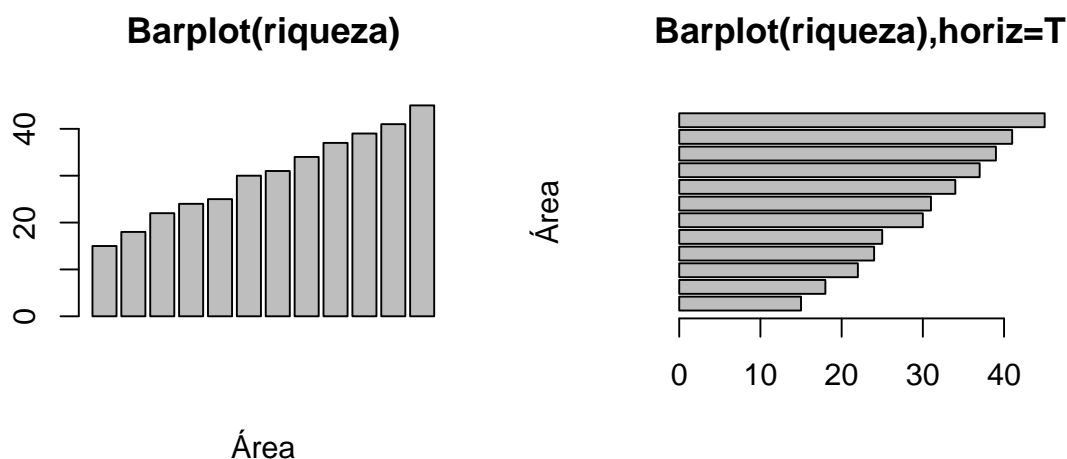
```
pedacos = c(10, 12, 4, 16, 8)
nomes = c("EUA", "Inglaterra", "Australia", "Alemanha", "Franca")
pie(pedacos, labels = nomes)
```



Barplot ou Gráfico de Barras

O barplot é um gráfico utilizado para medir a dimensão de cada elemento, seguindo sua ordem de ocorrência. No R, a função que produz gráfico de barras é **barplot()**, onde cada barra representa a medida de cada elemento de um vetor. Para trocar as barras verticais por horizontais, basta colocar **horiz = T**. A sintaxe geral da função está abaixo:

```
par(mfrow = c(1, 2))
barplot(riqueza, main = "Barplot(riqueza)", xlab = "Área")
barplot(riqueza, horiz = T, main = "Barplot(riqueza),horiz=T", ylab = "Área")
```

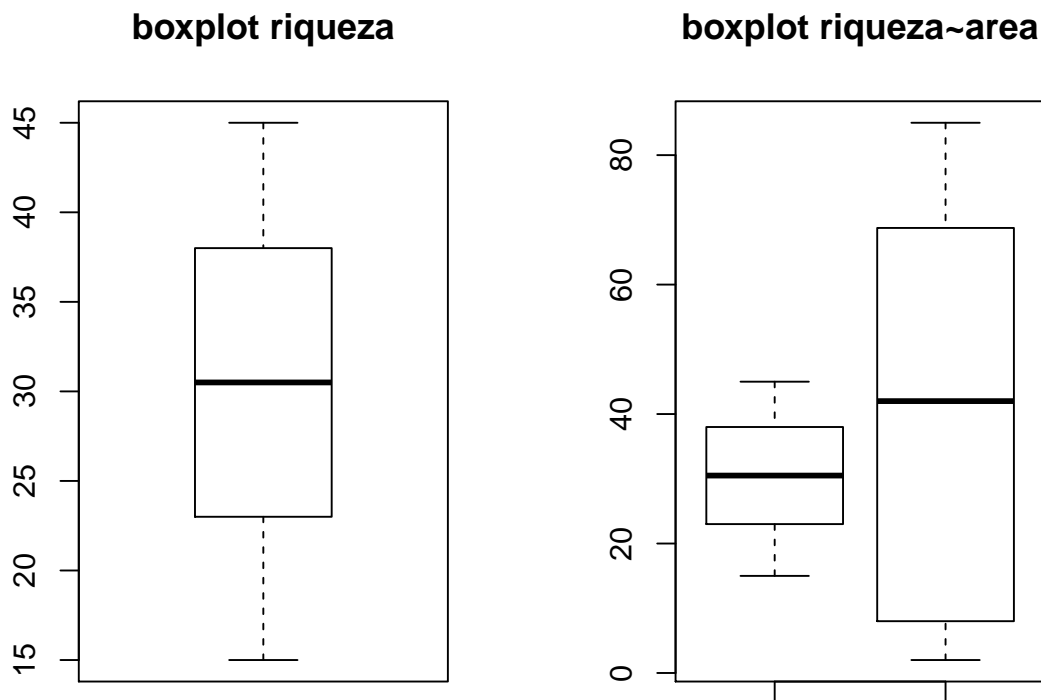


```
par(mfrow = c(1, 1))
```

Boxplot ou gráfico de caixa

O boxplot, ou diagrama de caixa, é uma ferramenta para localizar e analisar a variação de uma variável ou de um grupo dentre diferentes grupos de dados. É apresentado no eixo vertical a variável a ser analisada e no eixo horizontal um fator de interesse. O limite inferior da caixa representa o quartil inferior da variável que contém 25% (1/4) das menores medidas. Já o limite superior da caixa representa o quartil superior que contém 75% (3/4) dos valores observados. No meio da caixa temos uma linha que marca a mediana e ainda temos um segmento de reta vertical conectando o topo da caixa ao maior valor observado e outro segmento conectando a base da caixa ao menor valor observado. A sintaxe desse gráfico é **boxplot()**. Temos abaixo um exemplo de aplicação do boxplot com as bases riqueza e area, criadas anteriormente:

```
par(mfrow = c(1, 2))
boxplot(riqueza, main = "boxplot riqueza")
boxplot(riqueza, area, main = "boxplot riqueza-area")
```



```
par(mfrow = c(1, 1))
```

Apresentação dos gráficos

A customização dos gráficos é uma etapa muito importante para facilitar sua compreensão para qualquer pessoa. É possível adicionar legenda ao gráfico, colocar nomes nos eixos, mudar os tamanho, cores e muitas outras características da apresentação do gráfico.

Uma função usada para a customização é a **par()**. Após definidas as alterações com essa função, estas terão efeito em todos os gráficos posteriormente plotados, até que sejam alteradas novamente. Um exemplo da função **par()** que serve para mudar o número de gráficos plotados na mesma janela é **par(mfrow=c(NumeroDeLinhas, NumeroDeColunas))**.

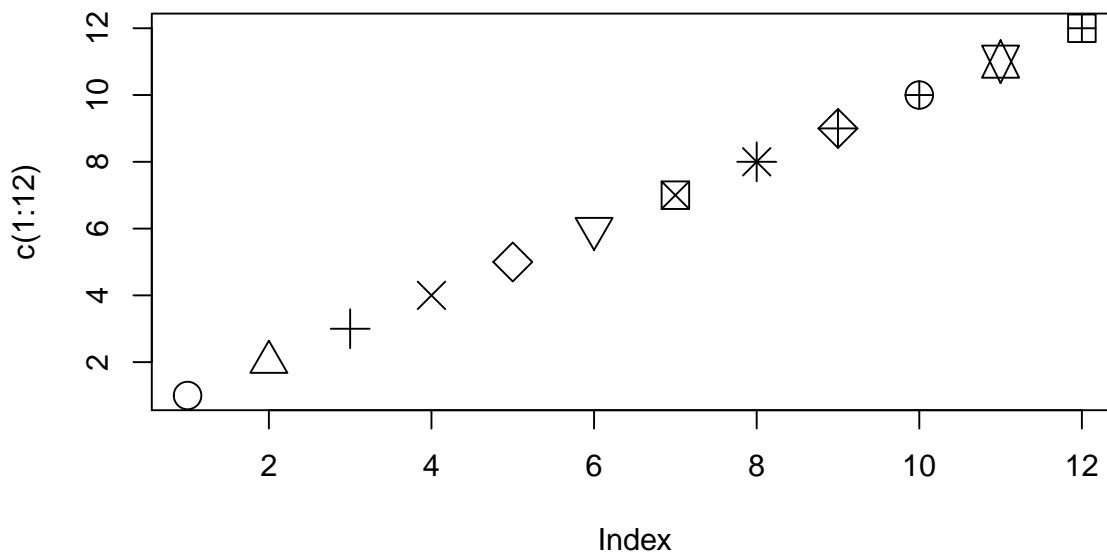
A outra maneira de especificar os parâmetro é utilizar as opções de alteração dentro dos próprios comandos gráficos. Dentre as possíveis mudanças na apresentação dos gráficos temos:

- A função **cex**, responsável por definir o tamanho do textos gráficos

Opção	Descrição
cex	tamanho do texto em relação ao padrão ¹
cex.axis	ampliação das medidas dos eixos em relação ao cex
cex.lab	ampliação do texto dos eixos em relação ao cex
cex.main	ampliação do texto do título em relação ao cex
cex.sub	ampliação do texto do subtítulo em relação ao cex

- A função pch, responsável por definir os símbolos utilizados para marcar os pontos do gráfico

```
plot(c(1:12), pch = c(1:12), cex = 2)
```



- A função lty e largura da linha

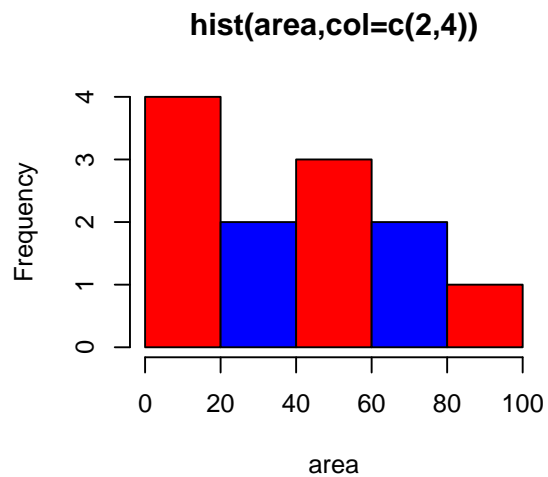
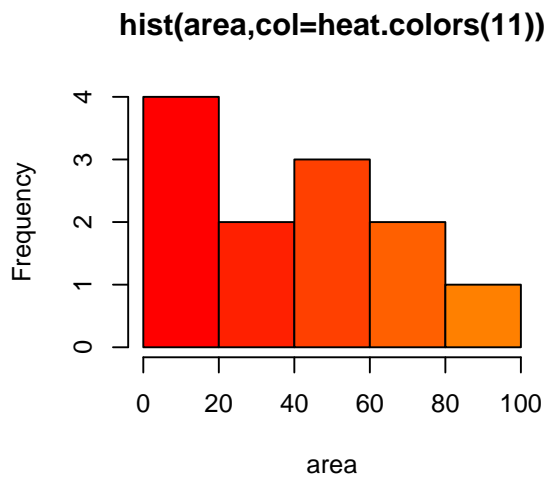
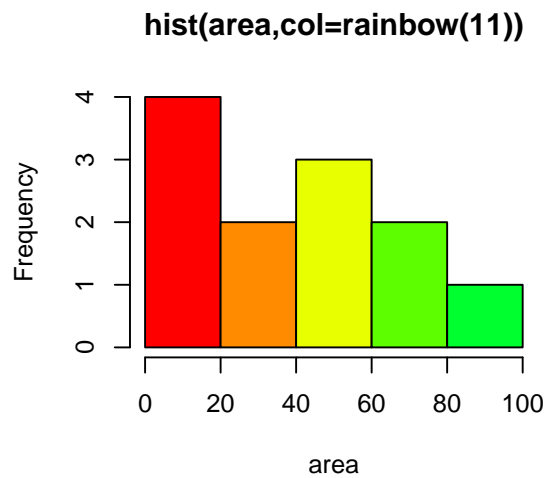
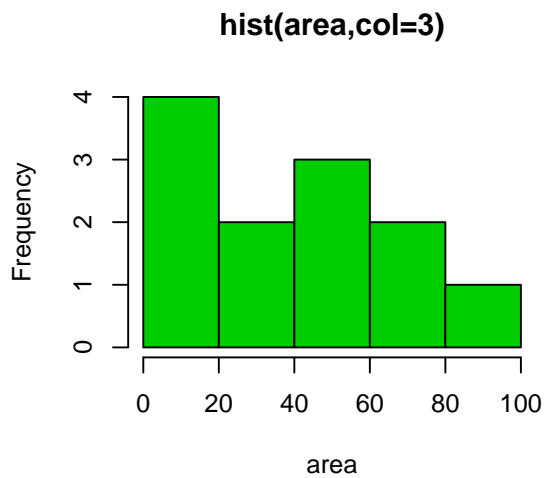
Opção	Descrição
lty	tipo de linha
lwd	largura da linha em relação ao padrão ²

¹1 é o padrão, 1.5 é 50% maior, 0.5 é 50% menor

²lwd = 2 é duas vezes mais largo que o padrão 1

- A função para definir as cores

```
par(mfrow = c(2, 2))
hist(area, main = "hist(area,col=3)", col = 3)
hist(area, main = "hist(area,col=rainbow(11))", col = rainbow(11))
hist(area, main = "hist(area,col=heat.colors(11))", col = heat.colors(11))
hist(area, main = "hist(area,col=c(2,4))", col = c(2, 4))
```



```
par(mfrow = c(1, 1))
```

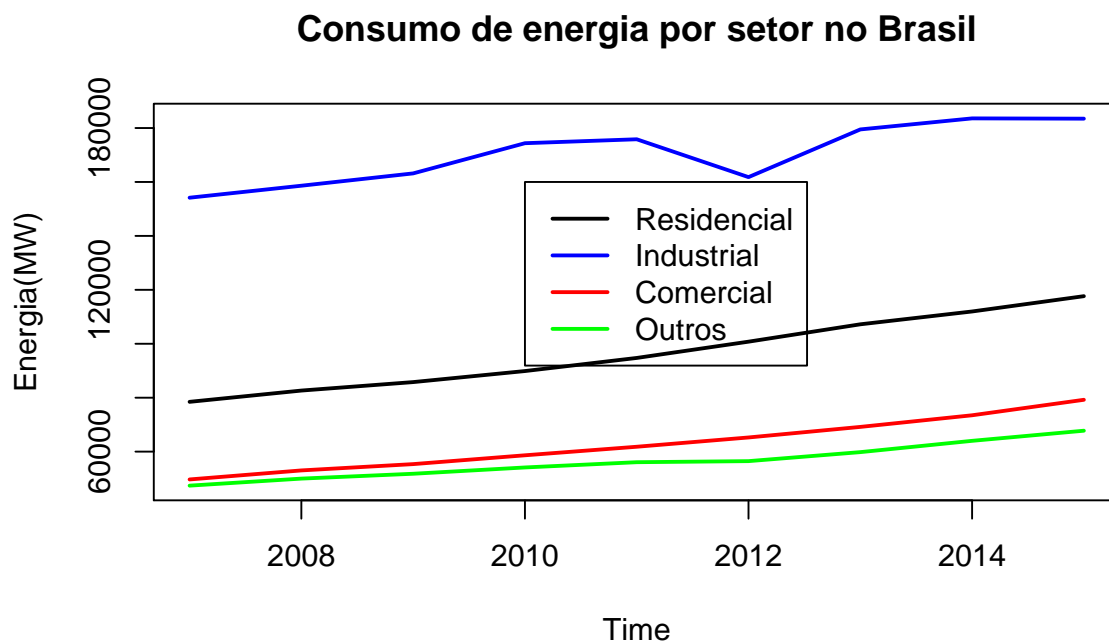
- A função para definir as fontes

É possível facilmente ajustar o tamanho das fontes e estilo, porém a família da fonte é um pouco mais complicado.

Opção	Descrição
font	Inteiro especificando o estilo a ser usado no texto ³
font.axis	fonte das medidas dos eixos
font.lab	fonte do texto dos eixos
font.main	fonte do texto do título
font.sub	fonte do texto do subtítulo
family	família de fonte para textos desenhados ⁴

- A função para adicionar legenda é legend()

```
Consumo = read.csv2("Excel1.csv")
Consumo = ts(Consumo, start = c(2007), freq = 1)
plot(Consumo[, 2], type = "l", main = "Consumo de energia por setor no Brasil",
     ylab = "Energia(MW)", lwd = 2, ylim = c(min(Consumo[, -1]), max(Consumo[,
     -1])))
lines(Consumo[, 3], col = "blue", lwd = 2)
lines(Consumo[, 4], col = "red", lwd = 2)
lines(Consumo[, 5], col = "green", lwd = 2)
legend(x = 2010, y = 160000, legend = c("Residencial", "Industrial", "Comercial",
    "Outros"), lwd = 2, col = c("black", "blue", "red", "green"))
```



Nela se pode especificar o local que será apresentado com as variáveis x e y e depois a própria leg-

³1=normal, 2=negrito, 3=itálico, 4= negrito e itálico,5=símbolo

⁴Valores padrão são "serif", "sans", "mono" e "symbol"

enda. Ex: `legend(x=200,y=120000,legend=c("Sudeste","Sul","Nordeste","Norte"))`. Outra opção para definir o local da legenda é colocar `x= "topleft"` ou outro local no mesmo modelo, buscando a melhor apresentação possível.

Salvando gráficos

Após a realização dos gráficos, muitas vezes é interessante salvá-los para serem utilizados em outros locais. Na plataforma RStudio é muito simples salvar os gráficos, basta utilizar o comando Export e selecionar Save as image. Já no R temos diversas funções para salvar os gráficos, variando o formato em que a imagem é salva. Elas estão apresentadas na tabela a seguir:

Função	Saída
<code>pdf("mygraph.pdf")</code>	arquivo pdf
<code>win.metafile("mygraph.wmf")</code>	windows metafile
<code>png("mygraph.png")</code>	arquivo png
<code>jpeg("mygraph.jpg")</code>	arquivo jpeg
<code>bmp("mygraph.bmp")</code>	arquivo bmp
<code>postscript("mygraph.ps")</code>	arquivo postscript

Funções

Nessa seção serão apresentados os comando de lógica e alguns comandos que auxiliam a criação de novas funções. O software R, além de ser um programa para análises estatísticas, é acima de tudo uma linguagem de programação, com a qual podemos programar nossas próprias funções. Assim, existe uma gama extensa de recursos que os usuários podem utilizar na análise de dados, auxiliando na criação de novas funções para suprir as necessidades particulares de cada trabalho.

Comandos de Lógica

Os comandos de lógica são operadores que servem para comparar objetos, retornando **TRUE**, caso a comparação testada seja verdadeira e **FALSE**, caso a comparação testada seja falsa. Os comandos lógicos são:

Significado	Símbolo	Exemplo
Maior que	<code>></code>	<code>10 > 5</code> (verdade)
Menor que	<code><</code>	<code>10 < 5</code> (falso)
Igualdade	<code>==</code>	<code>10 == 5</code> (falso)
Diferença	<code>!=</code>	<code>10 != 5</code> (verdade)
Maior ou igual que	<code>>=</code>	<code>10 >= 5</code> (verdade)
Menor ou igual que	<code><=</code>	<code>10 <= 5</code> (falso)

Está sendo criado o vetor x composto por 5 elementos.

```
x = c(1, 2, 9, 4, 5)
```

Agora, está sendo criado um outro vetor, chamado de y, que também composto por 5 elementos.

```
y = c(1, 2, 6, 7, 8)
```

É possível comparar os elementos do vetor x com o do vetor y, 1 a 1. Nessa comparação são comparados os elementos da mesma posição de x e y. No exemplo, usando o símbolo de maior que (>), é retornado TRUE quando os de x são maiores e FALSE quando são menores.

```
x > y
## [1] FALSE FALSE  TRUE FALSE FALSE
```

Trocando o símbolo anterior pelo de menor que (<), é retornado TRUE quando os elementos de x são menores e FALSE quando são maiores.

```
x < y
## [1] FALSE FALSE FALSE  TRUE  TRUE
```

Agora está sendo comparado se os elementos de x são iguais aos de y (Lembre que a igualdade no R é representada por dois iguais ==). É retornado TRUE quando os elementos são iguais e FALSE caso contrário

```
x == y
## [1]  TRUE  TRUE FALSE FALSE FALSE
```

Está sendo comparado se os elementos de x são diferentes aos de y. Desse modo, é retornado TRUE quando os elementos de x são diferentes aos de y e FALSE caso contrário.

```
x != y
## [1] FALSE FALSE  TRUE  TRUE  TRUE
```

Está sendo comparado se os elementos de x são maiores ou iguais aos de y. Se forem é retornado TRUE, caso contrário, é retornado FALSE.

```
x >= y
## [1]  TRUE  TRUE  TRUE FALSE FALSE
```

Está sendo comparado se os elementos de x são menores ou iguais aos de y. Se forem é retornado TRUE, caso contrário, é retornado FALSE.

```
x <= y
## [1]  TRUE  TRUE FALSE  TRUE  TRUE
```

Função which

A função **which()** funciona como se fosse uma pergunta: Quais? Assim, a função retorna a posição dos elementos que se enquadram na resposta da pergunta. Caso se queira os valores dos elementos, é necessário colocar a função **which()** entre colchetes [**which()**], ao lado do objeto em pesquisa.

Está sendo criado um vetor com 10 números para se testar essa função

```
a = c(2, 4, 6, 8, 10, 12, 14, 16, 18, 20)
```

É retornado um vetor contendo TRUE quando o elemento de a for maior que 10 e FALSE se for menor ou igual a 10

```
a > 10
## [1] FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
```

Usando o comando **which()** é retornado a posição no vetor dos valores de a que são maiores que 10. Como se perguntássemos, onde estão os valores maiores que 10 em a?

```
which(a > 10)
## [1]  6  7  8  9 10
```

Usando a função **which()** entre colchetes é retornado os valores de a que são maiores que 10 (Se difere do exemplo anterior porque é retornado o valor e não a posição dos elementos maiores que 10)

```
a[which(a > 10)]
## [1] 12 14 16 18 20
```

Também pode-se usar a função **which()** para selecionar segmentos de uma tabela de dados. Lembre-se que, anteriormente, já foi demonstrado a seleção de partes de uma tabela usando colchetes, a partir da especificação das linhas/colunas que se pretende usar. Já com o comando **which()** pode-se selecionar as partes desejadas de acordo com o critério de decisão escolhido. A vantagem desta função é que enquanto com colchetes se seleciona os elementos de interesse apenas pela posição, como a função pode-se usar qualquer critério de decisão.

Nesse caso estão sendo selecionados os terremotos que mataram mais de 670 pessoas:

```
quakes[which(quakes$depth > 670), ]
##      lat  long depth mag stations
## 256 -20.32 180.88   680 4.2         22
## 287 -19.30 180.60   671 4.2         16
```

Usando o comando nrow de terremotos com mais de 670 morte dividido pelo total de terremotos, vezes 100 é calculado o percentual de terremotos que mataram mais de 670 pessoas.


```
nrow(quakes[which(quakes$depth > 670), ])/nrow(quakes) * 100
## [1] 0.2
```

Agora é retornado apenas os terremotos que cumpram os dois requisitos - tendo matado mais de 650 pessoas e com uma duração menor ou igual a 180

```
quakes[which(quakes$depth > 650 & quakes$long <= 180), ]
##      lat   long depth mag stations
## 141 -12.66 169.46  658 4.6       43
## 301 -13.09 169.28  654 4.4       22
## 804 -12.93 169.52  663 4.4       30
## 857 -14.82 171.17  658 4.7       49
```

Funções aggregate e by

As funções **aggregate** e **by** podem ser utilizadas para aplicar uma função em todas as colunas de uma tabela, enquanto a função **tapply** faz isso apenas uma coluna por vez. As duas funções retornam os mesmos resultados, mas de formas diferentes. Enquanto a função **aggregate** retorna os resultados sem nenhuma separação, a função **by** apresenta uma linha pontilhada separando cada elemento do objeto. A função que será aplicada pelas colunas da tabela aparece como argumento **FUN= função**. Nos exemplos abaixo foram usadas a função média e summary na base de dados sobre flores (já existente no R)

Utilizando o comando **aggregate()** será calculado a média de comprimento por cada espécie de flor.

```
aggregate(iris$Sepal.Length ~ iris$Species, FUN = mean)
##   iris$Species iris$Sepal.Length
## 1      setosa           5.006
## 2 versicolor           5.936
## 3  virginica           6.588
```

Está sendo utilizado o comando **aggregate()** para calcular as medidas estatísticas de comprimento por cada espécie de flor.

```
aggregate(iris$Sepal.Length ~ iris$Species, FUN = summary)
##   iris$Species iris$Sepal.Length.Min. iris$Sepal.Length.1st Qu.
## 1      setosa           4.300           4.800
## 2 versicolor           4.900           5.600
## 3  virginica           4.900           6.225
##   iris$Sepal.Length.Median iris$Sepal.Length.Mean
## 1           5.000           5.006
## 2           5.900           5.936
## 3           6.500           6.588
##   iris$Sepal.Length.3rd Qu. iris$Sepal.Length.Max.
```

```
## 1          5.200          5.800
## 2          6.300          7.000
## 3          6.900          7.900
```

Ainda usando o comando **aggregate()**, pode-se calcular as médias de todas as características por cada espécie de flor.

```
aggregate(. ~ Species, data = iris, mean)
##      Species Sepal.Length Sepal.Width Petal.Length Petal.Width
## 1    setosa      5.006      3.428      1.462      0.246
## 2 versicolor      5.936      2.770      4.260      1.326
## 3  virginica      6.588      2.974      5.552      2.026
```

Agora utilizando o comando **by()** está sendo calculado a média de comprimento por cada espécie de flor, separando-as com uma linha pontilhada.

```
by(iris$Sepal.Length, iris$Species, FUN = mean)
## iris$Species: setosa
## [1] 5.006
## -----
## iris$Species: versicolor
## [1] 5.936
## -----
## iris$Species: virginica
## [1] 6.588
```

Está sendo utilizado o comando **by()** para calcular as medidas estatísticas de comprimento por cada espécie de flor, separadas por uma linha pontilhada

```
by(iris$Sepal.Length, iris$Species, FUN = summary)
## iris$Species: setosa
##      Min. 1st Qu.  Median      Mean 3rd Qu.      Max.
##      4.300   4.800   5.000   5.006   5.200   5.800
## -----
## iris$Species: versicolor
##      Min. 1st Qu.  Median      Mean 3rd Qu.      Max.
##      4.900   5.600   5.900   5.936   6.300   7.000
## -----
## iris$Species: virginica
##      Min. 1st Qu.  Median      Mean 3rd Qu.      Max.
##      4.900   6.225   6.500   6.588   6.900   7.900
```

Ainda com o comando **by()** as medidas estatísticas pode-se calcular todas as características por cada espécie de flor, separadas por uma linha pontilhada

```
by(iris, iris$Species, summary)
## iris$Species: setosa
##   Sepal.Length   Sepal.Width   Petal.Length   Petal.Width
##   Min.    :4.300   Min.    :2.300   Min.    :1.000   Min.    :0.100
##   1st Qu.:4.800   1st Qu.:3.200   1st Qu.:1.400   1st Qu.:0.200
##   Median :5.000   Median :3.400   Median :1.500   Median :0.200
##   Mean    :5.006   Mean    :3.428   Mean    :1.462   Mean    :0.246
##   3rd Qu.:5.200   3rd Qu.:3.675   3rd Qu.:1.575   3rd Qu.:0.300
##   Max.    :5.800   Max.    :4.400   Max.    :1.900   Max.    :0.600
##           Species
##   setosa      :50
##   versicolor: 0
##   virginica   : 0
##
##
## -----
## iris$Species: versicolor
##   Sepal.Length   Sepal.Width   Petal.Length   Petal.Width
##   Min.    :4.900   Min.    :2.000   Min.    :3.00   Min.    :1.000
##   1st Qu.:5.600   1st Qu.:2.525   1st Qu.:4.00   1st Qu.:1.200
##   Median :5.900   Median :2.800   Median :4.35   Median :1.300
##   Mean    :5.936   Mean    :2.770   Mean    :4.26   Mean    :1.326
##   3rd Qu.:6.300   3rd Qu.:3.000   3rd Qu.:4.60   3rd Qu.:1.500
##   Max.    :7.000   Max.    :3.400   Max.    :5.10   Max.    :1.800
##           Species
##   setosa      : 0
##   versicolor:50
##   virginica   : 0
##
##
## -----
## iris$Species: virginica
##   Sepal.Length   Sepal.Width   Petal.Length   Petal.Width
##   Min.    :4.900   Min.    :2.200   Min.    :4.500   Min.    :1.400
##   1st Qu.:6.225   1st Qu.:2.800   1st Qu.:5.100   1st Qu.:1.800
##   Median :6.500   Median :3.000   Median :5.550   Median :2.000
##   Mean    :6.588   Mean    :2.974   Mean    :5.552   Mean    :2.026
##   3rd Qu.:6.900   3rd Qu.:3.175   3rd Qu.:5.875   3rd Qu.:2.300
##   Max.    :7.900   Max.    :3.800   Max.    :6.900   Max.    :2.500
##           Species
##   setosa      : 0
##   versicolor: 0
##   virginica   :50
```

```
##  
##  
##
```

Condição if e else

Uma declaração **if else** em programação é uma instrução condicional que executa um conjunto diferente de declarações, dependendo se uma expressão é verdadeira ou falsa. Simplificando, a função **if else** significa se isso for verdadeiro, então seja aquilo, senão, seja aquilo outro.

Uma declaração típica **if else** é semelhante a apresentada abaixo:

É criado a variável x e atribuído o valor 1 a ela. Se x for igual a 1 (para dizer igual à usamos dois sinais de igual) então é retornado VERDADE, senão é retornado MENTIRA. Nesse exemplo, como x é igual a 1, então o R retornou VERDADE.

```
x = 1  
if (x == 1) {  
  "VERDADE"  
} else {  
  "MENTIRA"  
}  
## [1] "VERDADE"
```

Caso tivesse sido atribuído a x o valor 3, seria retornado MENTIRA, como se observa a seguir:

```
x = 3  
if (x == 1) {  
  "VERDADE"  
} else {  
  "MENTIRA"  
}  
## [1] "MENTIRA"
```

É importante ressaltar que entre as chaves após o **if** e o **else**, podemos ter tarefas mais complexas e que ainda é possível encadearmos condições if/else - de modo que dentro de uma condição if/else tenha-se outra condição if/else. As duas formas de escrita da função são:

- **if(condição){tarefa1}else{tarefa2}**
- **ifelse(condição,tarefa1,tarefa2)**

A diferença entre esses comandos está que o primeiro admite um conjunto de condições e tarefas, enquanto o segundo só aceita uma condição.

Exemplificando as duas formas de escrita para uma função que retorna 0 caso o valor seja maior que 50 ou 1 caso contrário. Para isso, se criou a variável x com valor 100. Como é maior que 50, é retornado 0.

```
x = 100
if (x > 50) {
  0
} else {
  1
}
## [1] 0
```

```
ifelse(x > 50, 0, 1)
## [1] 0
```

Comando for e while

Em programação, um loop é uma sequência de instruções que são continuamente repetidas até que uma determinada condição seja alcançada. No R, dois comandos são utilizados para tal: **for** e **while**. Esses comandos servem para definir quando se deve parar dentro de um looping.

O comando **for** segue o seguinte formato: *for(i in 1:N){conjunto de tarefas}*. Isso quer dizer que: para cada valor i o R vai calcular os comandos que estão entre as chaves {conjunto de tarefas}. O “i in 1:N” indica que os valores de i serão i = 1, depois i=2, até i = N. Para salvar os resultados que serão calculados no **for** precisa-se criar um objeto vazio que receberá os valores calculados.

No exemplo a seguir foi criado uma variável vazia (igual a NULL) chamada de aux para armazenar os valores dentro do for. A variável i serve para percorrer o vetor, de modo que a cada rodada i é acrescentado uma unidade, assim será 1, depois 2,... até 5 e assim será retornado a raiz quadrada de cada um desses valores de i.

```
aux = NULL
for (i in 1:5) {
  aux[i] = i + sqrt(i)
}
aux
## [1] 2.000000 3.414214 4.732051 6.000000 7.236068
```

Já o comando **while** segue o seguinte formato: **while(condição satisfeita){conjunto de tarefas}**. No caso dessa função, é necessário dentro das chaves criar uma variável que servirá para definir quando se deve parar dentro do looping. Assim, a condição satisfeita será os valores que essa variável assume para parar de realizar os comando entre chaves. No exemplo a seguir, o “a = a + 1” indica que o valor de a será o antecessor acrescentando 1 unidade a cada vez que corre o looping. Assim, no começo a=0, depois a=1, até a = condição. Igual ao **for**, para salvar os resultados que serão calculados no **while()** precisa-se criar um objeto vazio que receberá os valores calculados.

No exemplo a seguir foi criado a variável a para percorrer o while e a variável d para armazenar/apresentar os resultados, ambas iniciando com o valor zero. Dentro do **while()** é indicado quando irá acabar o looping, nesse caso, quando a assumir um valor maior ou igual a

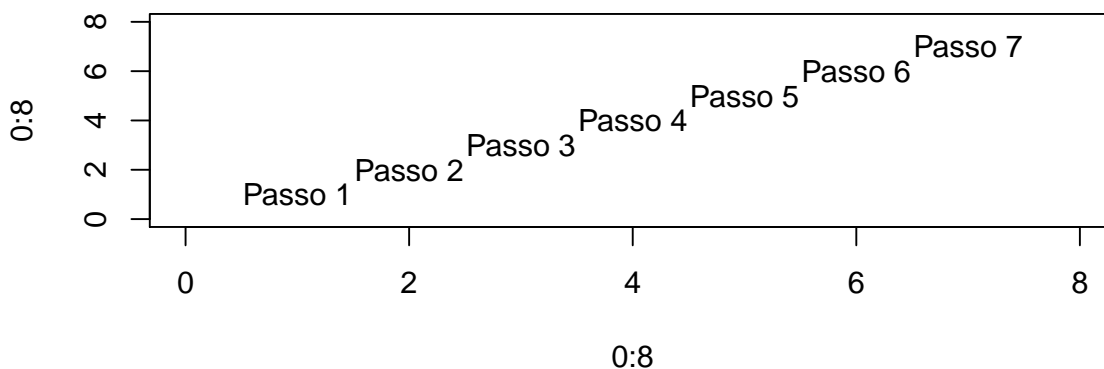
3. Dentro das chaves a variável `d` segue a regra de interesse, aumentando de 20 em 20 e sendo apresentado cada resultado, enquanto que a variável `a` é aumentada em 1 unidade a cada vez que entra no `while`, servindo para percorrer o looping.

```
a = 0
d = 0

while (a < 3) {
  a = a + 1
  d = d + 20
  print(d)
}
## [1] 20
## [1] 40
## [1] 60
```

Tanto com o `for()` quanto com o `while()`, o R é capaz de percorrer loopings enormes em muito pouco tempo. Desse modo, essas ferramentas são muito eficazes na manipulação de grandes bases de dados. Uma função interessante que permite uma maior compreensão do funcionamento de ambos os comandos é a `Sys.sleep(1)`. Ela permite visualizar esses comandos sendo executados de forma mais lenta, permitindo que se acompanhe o looping passo a passo. A seguir, para exemplificar a utilização dessa função, será plotado no gráfico no local *ixi* o texto Passo *i*. Quando colocado no R com a função `Sys.sleep()`, cada execução do `for` será feita com o espaçamento de 1 segundo, permitindo uma melhor visualização do comando `for()`.

```
plot(0:8, 0:8, type = "n")
for (i in 1:7) {
  text(i, i, paste("Passo", i))
  Sys.sleep(1)
}
```



Comando function

Todas as funções que foram citadas nessa seção podem ser utilizadas sozinhas, mas também auxiliam na criação de novas funções. Aprender a usar o R e não aprender a fazer programas básicos em R, ainda assim será vantajoso. Porém, aprender a programar será extremamente mais vantajoso. O comando básico para programar novas funções é o **function**. Uma função é um comando que a partir dos dados de entrada (argumentos) te retorna uma saída. No software R, uma função é criada seguindo o seguinte modelo: *nome.da.funcao=function(argumento1,. . . ,argumentoN){tarefa1;. . . ;tarefaN}*

A primeira parte da sintaxe é o nome da função, que é necessário para salvar a função que será criada. A segunda parte (function) é a hora em que é indicado para o R que uma função será criada. A lista de argumentos é uma lista, separada por vírgulas, que apresenta os argumentos que serão avaliados pela sua função. O corpo da função é a parte em que é escrito o “algoritmo” que será utilizado para fazer as tarefas/cálculos que deseja. Esta parte vem entre chaves. Para chamar a função basta digitar o nome da função seguida de seus argumentos entre parênteses: *nome.da.funcao(argumento1,. . . ,argumentoN)*.

A seguir será exemplificado como se cria uma nova função no R. Será criada uma função chamada de **funcao.teste**, que recebe 3 parâmetros e retorna a soma do dois primeiros parâmetros com o quadrado do terceiro. Os argumentos que a função recebe são a, b, c. O algoritmo da função é a soma de a com b, com c ao quadrado e é retornado esse resultado. Após criada a função, foi testado seu funcionamento enviando os elementos 2, 2 e 3.

```
funcao.teste = function(a, b, c) {  
  resultado = a + b + c^2  
  return(resultado)  
}  
funcao.teste(2, 2, 3)  
## [1] 13
```

Agora está sendo criada uma função que recebe um objeto como parâmetro. Ela testa se o objeto é um vetor. Caso seja, ela calcula a soma e o produtório dos elementos do mesmo. Caso contrário, ela retorna uma frase explicando que só funciona com vetores.

Para testar a função criada, foi enviado a matriz aux. Como aux não é um vetor, a função retorna erro. Quando testada a funcao.teste2 com o vetor criado aux2, é retornado a soma e o produtório dos elementos de vetor aux2

```
funcao.teste2 = function(x) {  
  if (is.vector(x)) {  
    val1 = sum(x)  
    val2 = prod(x)  
    cat("soma =", val1, "\n", "produtório =", val2, "\n")  
  } else {  
    print("A função só pode ser aplicada em um vetor")  
  }  
}
```

```
aux = matrix(seq(1:10), ncol = 5)
funcao.teste2(aux)
## [1] "A função só pode ser aplicada em um vetor"
```

```
aux2 = c(1:10)
funcao.teste2(aux2)
## soma = 55
## produtório = 3628800
```

Esta nova função realiza uma amostra aleatória do vetor `x`, `n` vezes, com reposição. É usada a função **sample** para retornar um conjunto aleatório de resultados dos lançamentos de uma moeda. Para testar a função criada, foi criado um vetor com os possíveis resultados de um lançamento (cara ou coroa) e depois enviado para a função esse vetor e o número de lançamentos desejados, nos exemplos, 2 e 10. Um comando interessante para resumir os resultados obtidos em casos com muitas repetições é o **table**. Ele contabiliza todos os resultados em um vetor ou data.frame, permitindo saber a quantidade de vezes que foi obtido cada resultado.

```
jogar.moeda = function(x, n) {
  sample(x, n, replace = T)
}
moeda = c("cara", "coroa")
jogar.moeda(moeda, 2)
## [1] "coroa" "cara"
```

```
jogar.moeda(moeda, 10)
## [1] "coroa" "coroa" "cara" "cara" "coroa" "cara" "coroa" "cara"
## [9] "coroa" "cara"
```

```
table(jogar.moeda(moeda, 1000))
##
## cara coroa
## 496 504
```

Esta função criada para jogar moedas é muito simples e nem é necessária, pois utilizando diretamente a função **sample** pode-se jogar moedas do mesmo modo, veja:

```
sample(c("cara", "coroa"), 10, replace = T)
## [1] "coroa" "coroa" "cara" "cara" "cara" "coroa" "cara" "coroa"
## [9] "cara" "coroa"
```

Porém, podem existir funções que sejam necessárias e que não existam no R e, neste caso, há a possibilidade de criar sua própria função. Por exemplo, suponha que não exista uma função do R para calcular o mínimo, ainda assim seria possível criar sua própria função para calculá-lo.

Essa função recebe um parâmetro, tendo a condição desse parâmetro recebido ser um vetor. Percorre o vetor comparando os elementos 1 a 1, deixando na variável criada min o menor elemento de cada comparação. Ao rodar todo vetor, tem-se a certeza que o valor mínimo está na variável min. É importante programar um aviso para caso a função não funcione, normalmente na condição **else**.

```
calculo.min = function(x) {  
  if (is.vector(x)) {  
    min = x[1]  
    for (i in 2:length(x)) {  
      if (x[i] < min) {  
        min = x[i]  
      } else {  
        min = min  
      }  
    }  
    cat("Valor mínimo igual a", min, "\n")  
  } else {  
    print("A função só pode ser aplicada em um vetor")  
  }  
}
```

Após criada a função, ela foi testada com um vetor que possui os números inteiros de -1000 a 150, e como esperado, a saída da função **calculo.min** foi o -1000.

```
valores = c(-1000:150)  
calculo.min(valores)  
## Valor mínimo igual a -1000
```

É importante lembrar de sempre fazer a condição de ter acontecido algum erro, por exemplo o parâmetro recebido pela função não é o esperado. Assim, uma função útil é a **stop**. Ela segue o seguinte modelo *stop*("Erro: Os parâmetros recebidos não estão de acordo com a utilidade da função")

Estatísticas Descritivas

As estatísticas descritivas são divididas em dois tipos de medidas: as medidas de dispersão e as medidas de posição.

- Medidas de posição: visam representar a posição onde a maioria dos dados se encontra (exemplos: média, moda e mediana).
- Medidas de dispersão: representam a variabilidade nos dados (exemplos: variância, desvio padrão e erro padrão).

Comandos no R

Medida	Fórmula matemática	Comando no R
Média	$\hat{Y} = \frac{1}{n} \sum_{i=1}^n Y_i$	mean()
Média das linhas	-	rowMeans()
Média das colunas	-	colMeans()
Mínimo	-	min()
Máximo	-	max()
Mediana	-	median
Variância	$s^2 = \frac{1}{n-1} \sum_{i=1}^n (Y_i - \hat{Y})^2$	var()
Desvio Padrão	$s = \sqrt{\frac{1}{n-1} \sum_{i=1}^n (Y_i - \hat{Y})^2}$	sd()
Quartis	-	quantile()
Curtose	-	kurtosis()
Assimetria	-	skewness()

A seguir está exemplificado o cálculo dessas medidas na base de dados CO2 (base já incluída no R). Para visualizar como é essa base deve-se usar o comando **data()**. Também é interessante determinar antes dos cálculos o número de casas decimais que se deseja. Para isso se usa o comando **options(digits=x)**. Ainda com o comando **options()** pode-se desabilitar a notação científica. O comando usado é **options(scipen=999)**

```
options(digits = 8)
data(CO2)
```

Utilizando o comando **mean()** é possível calcular a média da variável conc da base CO2

```
mean(CO2$conc)
## [1] 435
```

Utilizando o comando **rowMeans()** é possível calcular a média por linha das variáveis conc e uptake da base CO2

```
rowMeans(CO2[, 4:5])
##      1      2      3      4      5      6      7      8      9     10
## 55.50 102.70 142.40 193.60 267.65 357.10 519.85  54.30 101.15 143.55
##    11     12     13     14     15     16     17     18     19     20
## 195.90 270.30 358.20 522.15  55.60 103.70 145.15 196.05 271.45 359.45
##    21     22     23     24     25     26     27     28     29     30
## 522.75  54.60  99.55 140.15 192.30 266.25 355.20 519.35  52.15 101.15
##    31     32     33     34     35     36     37     38     39     40
## 142.50 194.40 269.30 356.25 521.20  55.05  98.00 144.05 192.00 269.45
##    41     42     43     44     45     46     47     48     49     50
## 357.30 520.70  52.80  97.10 138.10 190.00 265.45 353.70 517.75  53.50
```

```
##      51      52      53      54      55      56      57      58      59      60
##  98.50 140.30 190.90 266.20 353.05 515.75 53.15 97.20 137.90 188.95
##      61      62      63      64      65      66      67      68      69      70
## 264.25 351.55 513.90 52.75 94.95 134.05 184.45 259.75 348.60 510.95
##      71      72      73      74      75      76      77      78      79      80
## 51.35 93.20 131.15 181.50 256.25 344.35 507.20 52.80 96.50 133.95
##      81      82      83      84
## 183.95 258.95 346.95 509.95
```

Já para calcular as médias das colunas é utilizado o comando **colMeans()**. No exemplo a seguir é calculado a média por coluna das variáveis conc e uptake da mesma base.

```
colMeans(CO2[, 4:5])
##      conc      uptake
## 435.000000 27.213095
```

Utilizando o comando min é calculado o valor mínimo da variável conc da base CO2

```
min(CO2$conc)
## [1] 95
```

Utilizando o comando max é calculado o valor máximo da variável conc da base CO2

```
max(CO2$conc)
## [1] 1000
```

Utilizando o comando median é calculado o valor mediano da variável conc da base CO2

```
median(CO2$conc)
## [1] 350
```

Utilizando o comando var é calculado o variância da variável conc da base CO2.

```
var(CO2$conc)
## [1] 87571.084
```

Utilizando o comando sd é calculado o desvio padrão da variável conc da base CO2.

```
sd(CO2$conc)
## [1] 295.92412
```

Outra medida interessante de ser calculada é o coeficiente de variação (CV). Ele é empregado para estimar a precisão de experimentos. Para calcular o coeficiente basta fazer o desvio padrão sobre a média. A seguir será exemplificado esse cálculo com a coluna conc da base CO2.

```
Cv = sd(CO2$conc)/mean(CO2$conc)
Cv
## [1] 0.68028533
```

Utilizando o comando quantile é calculado o 1º quartil da variável conc da base CO2.

```
quantile(CO2$conc, 0.25)
## 25%
## 175
```

Utilizando o comando quantile é calculado o percentil 85% da variável conc da base CO2

```
quantile(CO2$conc, 0.85)
## 85%
## 675
```

A curtose é uma medida de dispersão que caracteriza o pico ou “achatamento” da curva da função de distribuição de probabilidade e a assimetria é a falta correspondência dos dois lados de uma distribuição. Para calcula-las no R é necessário ter o pacote moments e então usar os comando **kurtosis()** e **skewness()**. A seguir serão exemplificados esses cálculos com a coluna conc da base CO2.

```
library(moments)
kurtosis(CO2$conc)
## [1] 2.3735172
```

```
skewness(CO2$conc)
## [1] 0.73319958
```

Como já foi mencionado anteriormente, o R possui uma função que faz um resumo dos dados, apresentando seis medidas de posição que descrevem os dados (mínimo, máximo, média, mediana, primeiro quartil e terceiro quartil)

- A função é chamada **summary()**

Através da função summary será calculado as 6 medidas da base CO2

```
summary(CO2)
```

	Plant	Type	Treatment	conc
## Qn1	: 7	Quebec :42	nonchilled:42	Min. : 95
## Qn2	: 7	Mississippi:42	chilled :42	1st Qu.: 175
## Qn3	: 7			Median : 350
## Qc1	: 7			Mean : 435
## Qc3	: 7			3rd Qu.: 675

```
## Qc2      : 7                               Max.    :1000
## (Other):42
##      uptake
## Min.     : 7.700
## 1st Qu.:17.900
## Median  :28.300
## Mean    :27.213
## 3rd Qu.:37.125
## Max.    :45.500
##
```

Agora está sendo repetido o procedimento anterior só que para a coluna conc.

```
summary(CO2$conc)
##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##       95     175     350     435     675     1000
```

Distribuições de Probabilidade

O que são

Em estatística, uma distribuição de probabilidade descreve a chance que uma variável pode assumir ao longo de um espaço de valores. Ela é uma função cujo domínio são os valores da variável e cuja imagem são as probabilidades de a variável assumir cada valor do domínio. Essa distribuição de probabilidade pode ser discreta ou contínua.

- Discreta: São distribuições de variáveis discretas, que descrevem características mensuráveis que podem assumir apenas um número finito ou infinito contável de valores. Geralmente são o resultado de contagens.
- Contínua: São distribuições de variáveis contínuas, que descrevem características mensuráveis que assumem valores em uma escala contínua (na reta real), para as quais valores fracionais fazem sentido. Geralmente são medidas obtidas através de algum instrumento.

Funcionalidades no R

- Para cada distribuição que o R lida existe um nome raiz, por exemplo, o nome raiz da função de distribuição Normal é *norm*.
- Esse nome raiz tem como prefixo uma das letras abaixo:
 - **p** - calcula a função de probabilidade acumulada $F(x)$ no ponto
 - **q** - calcula o quantil correspondente a uma dada probabilidade
 - **d** - calcula a densidade de probabilidade de $f(x)$ no ponto
 - **r** - retira uma amostra aleatória da distribuição especificada

-Assim, para a função Normal, as funções já existendo são **pnorm()**, **qnorm()**, **dnorm()** e **rnorm()**.

Comandos no R

Distribuição	Comandos no R				
Beta	pbeta	qbeta	dbeta	rbeta	
Binomial	pbinom	qbinom	dbinom	rbinom	
Cauchy	pcauchy	qcauchy	dcauchy	rcauchy	
Qui-quadrado	pchisq	qchisq	dchisq	rchisq	
Exponencial	pexp	qexp	dexp	rexp	
Gamma	pgamma	qgamma	dgamma	rgamma	
Geométrica	pgeom	qgeom	dgeom	rgeom	
Hipergeométrica	phyper	qhyper	dhyper	rhyper	
Logística	plogis	qlogis	dlogis	rlogis	
Log-Normal	plnorm	qlnorm	dlnorm	rlnorm	
Binomial Negativa	pnbinom	qnbinom	dnbinom	rnbinom	
Normal	pnorm	qnorm	dnorm	rnorm	
Poisson	ppois	qpois	dpois	rpois	
t-Student	pt	qt	dt	rt	
Uniforme	punif	qunif	dunif	runif	
Weibull	pweibull	qweibull	dweibull	rweibull	

- Alguns exemplos no R?

Com o comando **pnorm()** está sendo calculado a área abaixo da curva da Normal padrão a esquerda de 0.

```
pnorm(0)
## [1] 0.5
```

Com o comando **qnorm()** está sendo calculado o percentil 90% da distribuição Normal padrão.

```
qnorm(0.9)
## [1] 1.2815516
```

Com o comando **rnorm()** estão sendo gerados 100 valores aleatórios da distribuição Normal padrão.

```
rnorm(50)
## [1] -0.011736044 -0.818567521 -0.175441498 -0.992284306 0.069140320
## [6] -0.291421053 -1.429642816 -0.029233845 0.318924892 -0.813950118
## [11] 0.854932375 -1.212715145 1.652368013 0.354144324 0.893159866
## [16] -1.121954551 0.040354118 -0.363030781 0.257688882 -0.984874370
## [21] 1.211443534 -2.345005536 -0.314275770 -0.151394798 -0.699558014
## [26] -1.045803549 0.741547486 0.319559510 -0.565133132 -1.659296828
## [31] 0.169850659 2.337182887 0.397172482 0.203664606 -1.486223751
```

```
## [36]  0.714393895  1.028065673  0.121731326  0.442048451 -0.026066822
## [41] -0.172371299  1.614811690 -0.359611302  0.159086541 -0.124349018
## [46]  0.545400995  0.533359446  0.620879915  1.689687500 -0.275926013
```

Ainda com o comando **rnorm()** pode-se especificar algumas medidas dos valores aleatórios. No exemplo a seguir são calculados 100 valores aleatórios de uma distribuição Normal com média 50 e desvio padrão 10.

```
rnorm(50, mean = 50, sd = 10)
## [1] 70.970946 34.057424 54.247588 34.559981 52.877853 64.306992 52.713549
## [8] 58.786622 53.571173 49.945618 34.726226 43.831943 48.790313 39.809744
## [15] 56.838843 62.298239 23.897001 45.906576 58.903609 34.895893 61.952046
## [22] 54.336386 68.186295 63.772580 47.231897 29.878128 43.695904 49.311297
## [29] 55.440378 54.944573 48.980708 55.634397 42.466732 59.395795 55.487677
## [36] 42.832956 38.338109 51.201776 60.452669 42.733798 53.724855 55.191655
## [43] 50.203572 37.439867 62.656736 60.058500 41.608687 57.580650 47.626117
## [50] 56.762978
```

A seguir, um exemplo mais realista da aplicação das distribuições.

- Os scores do QI de crianças são normalmente distribuídos com média 100 e desvio padrão 15. Qual a proporção esperada de crianças com QI entre 80 e 120? Dica: $P(80 < X < 120) = P(X < 120) - P(X < 80)$

Para resolver o exercício calcula-se com a função **pnorm()** a proporção até 120 da distribuição especificada e subtrai a proporção até 80, chegando na faixa desejada.

```
pnorm(120, mean = 100, sd = 15) - pnorm(80, mean = 100, sd = 15)
## [1] 0.81757756
```

- Agora digamos que o QI seja distribuído como uma Binomial de tamanho 100 e probabilidade 0.85. Como se calcula a proporção esperada de crianças com QI entre 80 e 120?

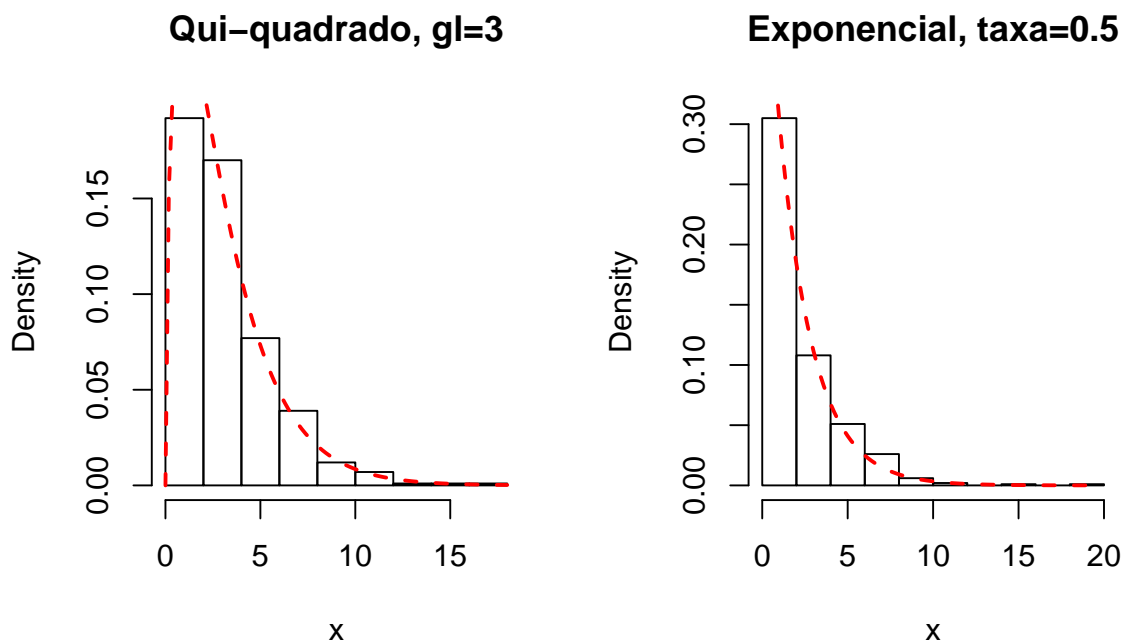
Como agora a distribuição é binomial, realiza-se a proporção de cada um dos valores inteiros entre 80 e 120 e depois se soma todos.

```
sum(dbinom(81:119, 100, 0.85))
## [1] 0.89345574
```

Plotando histograma e distribuição de probabilidade no R

Como já foi mostrado no momento da explicação da utilização do histograma, a função de distribuição de probabilidade é um interessante recurso visual nesse tipo de gráfico. A seguir tem-se mais um exemplo da aplicação dos dois recursos juntos:

```
par(mfrow = c(1, 2))
x = rchisq(500, df = 3)
hist(x, freq = F, main = "Qui-quadrado, gl=3")
curve(dchisq(x, df = 3), col = 2, lty = 2, lwd = 2, add = TRUE)
x = rexp(500, rate = 0.5)
hist(x, freq = F, main = "Exponencial, taxa=0.5")
curve(dexp(x, rate = 0.5), col = 2, lty = 2, lwd = 2, add = TRUE)
```



```
par(mfrow = c(1, 1))
```

Testes de Hipóteses

Ao ser feita determinada afirmação sobre uma população, mais especificamente sobre um parâmetro dessa população, é natural desejar saber se os resultados experimentais provenientes de uma amostra contrariam, ou não, tal afirmação. Para isso, fazemos um teste de hipóteses.

Em todos os testes, a seguir, será considerado o mesmo experimento: + um experimento onde foram realizadas N medidas de uma variável aleatória X e em cada medida, a variável X assume os valores x_1, x_2, \dots, x_N , sendo μ a média e σ^2 a variância.

Caso não seja especificado o nível de significância da distribuição, é usado 5%, que é o nível padrão.

Teste para média

O teste para média consiste em verificar, através de uma amostra, se a média da população atende o caso em teste, para um certo nível de significância desejado.

Fixando $H_0 : \mu = \mu_0$, a hipótese nula pode ter uma das três formas abaixo:

$$\begin{cases} H_1 : \mu \neq \mu_0 \\ H_1 : \mu > \mu_0 \\ H_1 : \mu < \mu_0 \end{cases}$$

Para esse cálculo no R, se usa o comando `t.test()`

Caso o teste não seja bilateral, é necessário especificar dentro dos parenteses se ele é unilaral para a esquerda ou para direita. Para isso se usa `alternative="less"` - Quando só interessar a calda esquerda da distribuição - ou `alternative="greater"` - Quando só interessar a calda direita da distribuição.

- Um engenheiro de produção quer testar, com base nos dados da tabela a seguir e para um nível de significância $\alpha = 0.05$, se a altura média de uma haste está próxima do valor nominal de 1055 mm. Uma amostra de 20 hastes foi analisada e as medidas obtidas são dadas a seguir:

Está sendo criado o vetor dados com as medidas das 20 hastes.

```
dados = c(903.88, 1036.92, 1098.04, 1011.26, 1020.7, 915.38, 1014.53, 1097.79,
          934.52, 1214.08, 993.45, 1120.19, 860.41, 1039.19, 950.38, 941.83, 936.78,
          1086.98, 1144.94, 1066.12)
```

Usando o comando `t.test()` pode-se testar $H_0 : \mu = 1055$, isto é, se a média observada é estatisticamente igual a 1055. No comando, como não foi especificado a alternativa, é realizado um teste bilateral.

```
t.test(dados, mu = 1055)
##
##  One Sample t-test
##
## data:  dados
## t = -1.74402, df = 19, p-value = 0.097313
## alternative hypothesis: true mean is not equal to 1055
## 95 percent confidence interval:
##   976.60666 1062.13034
## sample estimates:
## mean of x
## 1019.3685
```

Note que, com nível de significância de 5%, não rejeitamos a hipótese nula de que a média da base de dados é estatisticamente igual a 1055.

Para esse exercício deve ser considerada a hipótese nula bilateral, porém, é possível testar as demais formas, como segue:

Mesmo teste só que agora com a hipótese alternativa = menor (calda esquerda).

```
t.test(dados, mu = 1055, alternative = "less")
##
## One Sample t-test
##
## data: dados
## t = -1.74402, df = 19, p-value = 0.048656
## alternative hypothesis: true mean is less than 1055
## 95 percent confidence interval:
##      -Inf 1054.6958
## sample estimates:
## mean of x
## 1019.3685
```

Mesmo teste só que agora com a hipótese alternativa = maior (calda direita).

```
t.test(dados, mu = 1055, alternative = "greater")
##
## One Sample t-test
##
## data: dados
## t = -1.74402, df = 19, p-value = 0.95134
## alternative hypothesis: true mean is greater than 1055
## 95 percent confidence interval:
##  984.0412      Inf
## sample estimates:
## mean of x
## 1019.3685
```

Teste para proporção (Teste Qui-Quadrado de Pearson)

O teste para proporção é usado para verificar se a frequência com que um determinado acontecimento observado em uma amostra se desvia significativamente ou não da frequência com que ele é esperado.

Também é usado para comparar a distribuição de diversos acontecimentos em diferentes amostras, a fim de avaliar se as proporções observadas destes eventos mostram ou não diferenças significativas ou se as amostras diferem significativamente quanto às proporções desses acontecimentos.

Os eventos tem que ser independentes

Tal teste usa como referência a distribuição Qui-Quadrado

Considerando uma população em que a proporção de indivíduos portadores de certa característica é p .

Definimos a variável aleatória X como sendo 1 (hum) se o individuo tiver a característica e 0 (zero) caso contrário. Assim, temos que $X \sim \text{Bernoulli}(p)$

Se quer testar $H_0 : p = p_0$, onde H_1 pode assumir as seguintes formas:

$$\begin{cases} H_1 : p \neq p_0 \\ H_1 : p < p_0 \\ H_1 : p > p_0 \end{cases}$$

Para esse cálculo no R, se usa o comando **prop.test()**

A seguir um exemplo da utilização dessa função:

- De uma amostra de tamanho 200, houveram 175 registros de pessoas doentes. Gostariásse de testar se essa proporção é estatisticamente igual a 0.9.

Para isso é criando um vetor com 175 repetições de 1 e 25 repetições de 0. Depois é usado a função **table()** para apresentar o vetor resultante na forma de tabela e se chama de sucessos a coluna da quantidade de vezes que se obteve o valor 1. Agora já é possível realizar o teste para proporção para a variável sucessos, enviando a variável, o número total e a proporção que se quer testar para a função **prop.test()**.

```
dados = c(rep(1, 175), rep(0, 25))
table(dados)
## dados
##    0    1
##   25 175
```

```
sucessos = table(dados)[[2]]
sucessos
## [1] 175
```

```
prop.test(sucessos, n = 200, p = 0.9)
##
## 1-sample proportions test with continuity correction
##
## data:  sucessos out of 200, null probability 0.9
## X-squared = 1.125, df = 1, p-value = 0.28884
## alternative hypothesis: true p is not equal to 0.9
## 95 percent confidence interval:
##  0.81915838 0.91595304
## sample estimates:
##      p
## 0.875
```

Observa-se que, com nível de 5% de significância, não há evidências para rejeitar H_0 . Assim, podemos acreditar que a proporção é estatisticamente igual a 0.9.

Teste para taxa

Este teste é usado para determinar se a taxa de ocorrência para um grupo é diferente de um valor especificado, para um certo nível de significância desejado. Ele ainda permite calcular um intervalo de valores que provavelmente inclui a taxa de ocorrência da população.

Considere uma população e X uma variável aleatória que representa determinada característica desta população, com distribuição de Poisson com parâmetro λ .

Querendo testar $H_0 : \lambda = \lambda_0$, onde H_1 pode assumir as seguintes formas:

$$\begin{cases} H_1 : \lambda \neq \lambda_0 \\ H_1 : \lambda < \lambda_0 \\ H_1 : \lambda > \lambda_0 \end{cases}$$

Para esse cálculo no R, se usa o comando **poisson.test()**.

A seguir um exemplo da utilização dessa função:

- Em uma amostra de 25 operários foi aplicada uma nova metodologia e observou-se que a média foi de 5 erros por semana, enquanto com a metodologia antiga tinha-se em média 4 erros po semana. Com essas informações, podemos falar que há diferença significativa entre a antiga e a nova metodologia?

Primeiramente deve-se notar que nesse caso $H_1 : \lambda < 4$, pois esta sendo testado se a nova metodologia é melhor que a antiga.

Para obter a resposta será utilizado o comando **poisson.test()** para testar se o resultado de 5 erros é significamente diferente do anterior. Assim, os parâmetros recebidos pela função são: O número total, o número de erros, a parcela anterior de erros. O teste realizado é bilateral.

```
poisson.test(25, 5, r = 4/25)
##
## Exact Poisson test
##
## data: 25 time base: 5
## number of events = 25, time base = 5, p-value < 2.22e-16
## alternative hypothesis: true event rate is not equal to 0.16
## 95 percent confidence interval:
## 3.2357364 7.3809863
## sample estimates:
## event rate
## 5
```

Observa-se que não há evidências para rejeitar H_0 . Assim, pode-se acreditar que a nova metodologia é tão eficiente quanto a metodologia anterior.

Teste para variância

O teste para variância é usado para determinar se a variância de uma população é igual ou não a um determinado valor

Seja X_1, X_2, \dots, X_n uma amostra aleatória de tamanho n retirada de uma população normal $N(\mu, \sigma^2)$.

Suponha que se deseja testar uma hipótese sobre a variância σ^2 desta população. Para isso, pode-se usar a distribuição Qui-quadrado.

As hipóteses são $H_0 : \sigma^2 = \sigma_0^2$, onde H_1 pode assumir as seguintes formas:

$$\left\{ \begin{array}{l} H_1 : \sigma^2 \neq \sigma_0^2 \\ H_1 : \sigma^2 < \sigma_0^2 \\ H_1 : \sigma^2 > \sigma_0^2 \end{array} \right.$$

Para esse cálculo no R, se usa o comando **varTest()** e para usar este comando é necessário carregar o pacote **EnvStats**.

A seguir um exemplo da utilização dessa função:

- Uma amostra aleatória de 20 garrafas resulta em uma variância da amostra do volume de enchimento de $s^2 = 0.0153$. Se a variância do volume de enchimento exceder 0.01, existirá uma proporção inaceitável de garrafas cujo enchimento não foi completo ou foi em demasia.

Há evidência nos dados da amostra sugerindo que o fabricante tenha um problema com garrafas com falta ou excesso de detergente?

Primeiramente deve-se notar que nesse caso $H_1 : \sigma^2 > 0.01$, pois está sendo testado se a variância do volume é menor que 0.01. Para resolver esse exercício foi carregado o pacote EnvStats, se criou um vetor com diversos valores que foi enviado para a função **varTest()**, junto com o sigma com valor de 0,01 e a alternativa de ser maior que sigma.

```
library(EnvStats)

dados = c(2.15921, 1.882974, 1.837874, 2.02035, 1.830111, 1.833311, 2.023907,
          2.062808, 2.120751, 2.038978, 2.057847, 2.224181, 2.040718, 2.087178, 1.924255,
          2.125798, 2.040933, 1.911035, 2.012782, 1.918585)
varTest(dados, sigma.square = 0.01, alternative = "greater")
##
## Results of Hypothesis Test
## -----
##
## Null Hypothesis:                variance = 0.01
##
## Alternative Hypothesis:         True variance is greater than 0.01
##
```

```
## Test Name: Chi-Squared Test on Variance
##
## Estimated Parameter(s): variance = 0.01277012
##
## Data: dados
##
## Test Statistic: Chi-Squared = 24.263227
##
## Test Statistic Parameter: df = 19
##
## P-value: 0.18624831
##
## 95% Confidence Interval: LCL = 0.0080492329
## UCL = Inf
```

Observa-se que não há evidências de que a variância do volume de enchimento exceda 0.01

Teste para comparação de duas variâncias (Teste F)

O teste de comparação de duas variâncias é usado para testar hipóteses que concluem se a variância de duas populações é igual ou diferente, para um certo nível de significância desejado.

Suponha que se quer comparar as variâncias σ_1^2 e σ_2^2 de duas populações Normais independentes.

As hipóteses possíveis são: $H_0 : \sigma_1^2 = \sigma_2^2$, onde H_1 pode assumir as seguintes formas:

$$\begin{cases} H_1 : \sigma_1^2 \neq \sigma_2^2 \\ H_1 : \sigma_1^2 < \sigma_2^2 \\ H_1 : \sigma_1^2 > \sigma_2^2 \end{cases}$$

Para esse cálculo no R, se usa o comando **var.test()**.

A seguir um exemplo da utilização dessa função:

- Um analista da qualidade quer avaliar se existe diferença entre as variabilidades na produção de eixo desenvolvido por dois sistemas de usinagem. A medições de duas populações independentes com distribuição Normal encontram-se a seguir (vetores usina1 e usina2)

```
usina1 = c(18.7997, 20.5035, 18.6214, 19.9192, 21.117, 20.8353, 17.527, 17.078,
17.6197, 21.4255, 18.7545, 19.2026, 18.4187, 20.7641, 21.0553, 17.5905,
18.7561, 18.9772, 20.3084, 18.8988, 19.1688, 19.2898, 22.059, 18.5854, 17.8896)

usina2 = c(21.1609, 26.1371, 21.4737, 30.9934, 22.8421, 24.4133, 20.4137, 25.5475,
21.8791, 22.6706, 24.7531, 25.7219, 22.6389, 26.2308, 26.7998, 28.4708,
26.9941, 25.1489, 24.6179, 27.0194, 25.0589, 22.1119, 20.3069, 23.6758,
27.1201, 29.6136, 25.9948, 18.223, 23.7336, 22.4208)
```

Pode-se dizer que as variâncias de ambas são iguais?

Para se responder a pergunta, deve-se fazer o teste de duas variâncias entre as duas usinas. Para isso usa-se o comando `var.test()`. Como o nível de significância não foi especificado é usado 5%

```
var.test(usina1, usina2)
##
## Results of Hypothesis Test
## -----
##
## Null Hypothesis:                ratio of variances = 1
##
## Alternative Hypothesis:         True ratio of variances is not equal to 1
##
## Test Name:                      F test to compare two variances
##
## Estimated Parameter(s):         ratio of variances = 0.22258598
##
## Data:                          usina1 and usina2
##
## Test Statistic:                 F = 0.22258598
##
## Test Statistic Parameters:      num df   = 24
##                                denom df = 29
##
## P-value:                       0.0003578537
##
## 95% Confidence Interval:        LCL = 0.10333582
##                                UCL = 0.49357167
```

Observa-se que com 5% de nível de significância rejeitamos H_0 , isto é, há evidências estatísticas para acreditar que as variâncias são diferentes

Teste para comparação de duas médias

Este teste é usado para comparar se a média de duas populações são iguais ou não, para um certo nível de significância desejado.

Suponha que se necessite comparar duas médias de duas populações independentes e ambas com distribuição Normal.

As hipóteses possíveis são $H_0 : \mu_1 = \mu_2$, onde H_1 pode assumir as seguintes formas:

$$\begin{cases} H_1 : \mu_1 \neq \mu_2 \\ H_1 : \mu_1 < \mu_2 \\ H_1 : \mu_1 > \mu_2 \end{cases}$$

Neste teste, dois casos podem ser tratados + 1º Caso: Variâncias iguais, porém desconhecidas
+ 2º Caso: Variâncias desconhecidas e diferentes

Em ambos os casos, será usado um teste t, em que a diferença no R será indicada na declaração dos argumentos

Para esse cálculo no R, se usa o comando **t.test()**.

A seguir um exemplo da utilização de cada caso dessa função:

- Decida se existe diferença significativa entre as médias populacionais dado que as variâncias são iguais (1º caso)

Para responder a pergunta foram criados dois vetores (amostra1 e amostra2), e com o comando **t.test()** foi realizado o teste de comparação da média das duas amostras, com variâncias iguais.

```
amostra1 = c(18.8, 17.591, 20.835, 19.169, 18.755, 20.504, 18.756, 17.527, 19.29,
             19.203, 18.621, 18.977, 17.078, 22.059, 18.419, 19.919, 20.308, 17.62, 18.585,
             20.764, 21.117, 18.899, 21.426, 17.89, 21.055)
amostra2 = c(22.284, 22.057, 22.629, 24.62, 21.491, 21.198, 21.901, 22.881,
             22.86, 22.058, 22.699, 22.909, 25.302, 17.968, 24.515, 23.15, 24.662, 23.327,
             22.447, 23.382, 22.426, 22.787, 21.983, 24.534, 22.771, 21.043, 21.203,
             24.009, 21.917, 21.152)
t.test(amostra1, amostra2)
##
## Results of Hypothesis Test
## -----
##
## Null Hypothesis:                difference in means = 0
##
## Alternative Hypothesis:         True difference in means is not equal to 0
##
## Test Name:                     Welch Two Sample t-test
##
## Estimated Parameter(s):        mean of x = 19.32668
##                                mean of y = 22.60550
##
## Data:                          amostra1 and amostra2
##
## Test Statistic:                t = -8.6651172
##
## Test Statistic Parameter:      df = 52.094731
##
## P-value:                      1.1323049e-11
##
## 95% Confidence Interval:       LCL = -4.0380883
##                                UCL = -2.5195517
```


Observa-se que com 5% de nível de significância rejeitamos H_0 , isto é, há evidências estatísticas para acreditar que as médias são diferentes

Agora, usando o exemplo do Teste para comparação de duas variâncias para realizar o teste para comparação das médias das usinas 1 e 2 sendo as variâncias diferentes e desconhecidas (2º caso). Para isso, dentro da função `t.test()` se coloca a opção: `var.equal=F`

```
t.test(usina1, usina2, var.equal = F)
##
## Results of Hypothesis Test
## -----
##
## Null Hypothesis:                difference in means = 0
##
## Alternative Hypothesis:         True difference in means is not equal to 0
##
## Test Name:                     Welch Two Sample t-test
##
## Estimated Parameter(s):        mean of x = 19.326604
##                                mean of y = 24.472880
##
## Data:                          usina1 and usina2
##
## Test Statistic:                t = -8.6718188
##
## Test Statistic Parameter:      df = 42.865626
##
## P-value:                      5.5611654e-11
##
## 95% Confidence Interval:       LCL = -6.343187
##                                UCL = -3.949365
```

Observa-se que com 5% de nível de significância rejeitamos H_0 , isto é, há evidências estatísticas para acreditar que as médias são diferentes

Teste t pareado

Usa-se este teste para determinar se a média das diferenças entre duas amostras pareadas é diferente de 0 (ou um valor alvo) e para calcular um intervalo de valores que provavelmente inclui a média das diferenças de uma população.

Este teste é similar aos testes de igualdade de variâncias e aos testes de médias. A diferença é que enquanto que para os testes de igualdade de variâncias é necessário que as duas populações sejam independentes, para o teste t pareado as duas populações podem ser dependentes.

A seguir um exemplo da utilização dessa função:

- Considerando duas amostras dependentes X_1, \dots, X_n e Y_1, \dots, Y_n . Neste caso considera-se

observações pareadas, isto é, pode-se considerar que temos na realidade uma amostra de pares $(X_1, Y_1), \dots, (X_n, Y_n)$

Deve-se definir $D_i = X_i - Y_i$, para $i = 1, 2, \dots, n$. Assim se obtem a amostra D_1, \dots, D_n e vamos considerar que $D_i \sim N(\mu_D, \sigma_D^2)$.

- As hipóteses possíveis são $H_0 : \mu_D = 0$, onde H_1 pode assumir as seguintes formas:

$$\begin{cases} H_1 : \mu_D \neq 0 \\ H_1 : \mu_D < 0 \\ H_1 : \mu_D > 0 \end{cases}$$

Para esse cálculo no R, também se usa o comando `t.test()`, porém adiciona-se o argumento `paired=TRUE`.

A seguir um exemplo da utilização dessa função:

- Consideremos 20 amostras de dois laboratórios A e B, onde os testes são realizados da mesma forma (há correlação), verifique se as médias são iguais

Primeiramente foram criados os vetores labA e labB, e depois com o comando `t.test()` se testou se os dois labs são independentes.

```
labA = c(1.00552, -1.49928, 0.21367, 0.44658, 0.62766, 0.31091, -0.83878, -0.29054,
        -0.08487, -1.26465, -0.06353, -1.07632, -1.34134, -0.55062, 1.61848, 0.50997,
        0.76027, 0.68061, -1.91464, -0.20072)
labB = c(0.01942, -0.46512, 0.53218, -0.14844, -0.60021, 0.06495, 0.33013, 0.12116,
        0.74269, -1.64232, 0.05497, 0.76342, 1.74131, -0.06392, -1.88146, -0.76135,
        -0.23009, -1.168, 0.88392, 0.96512)
t.test(labA, labB, paired = T)
##
## Results of Hypothesis Test
## -----
##
## Null Hypothesis:                difference in means = 0
##
## Alternative Hypothesis:         True difference in means is not equal to 0
##
## Test Name:                     Paired t-test
##
## Estimated Parameter(s):        mean of the differences = -0.110499
##
## Data:                          labA and labB
##
```

```
## Test Statistic:          t = -0.31494076
##
## Test Statistic Parameter: df = 19
##
## P-value:                 0.75623921
##
## 95% Confidence Interval:  LCL = -0.84485001
##                          UCL =  0.62385201
```

Observa-se que com 5% de nível de significância não rejeitamos H_0 , isto é, não há evidências estatísticas para acreditar que as médias são diferentes

Teste para comparação de duas proporções

Usa-se o teste para comparação de duas proporções para descobrir se existe ou não diferença entre as proporções de duas populações.

Considerando X e Y variáveis aleatórias que representam determinada característica de duas populações com distribuição de Bernoulli com parâmetros p_1 e p_2 , respectivamente.

As hipóteses possíveis são $H_0 : p_1 = p_2$, onde H_1 pode assumir as seguintes formas:

$$\begin{cases} H_1 : p_1 \neq p_2 \\ H_1 : p_1 < p_2 \\ H_1 : p_1 > p_2 \end{cases}$$

Para esse cálculo no R, se usa o comando **prop.test()**.

A seguir um exemplo da utilização dessa função:

- Teste se a taxa de reclamação de duas empresas é a mesma, onde de uma amostra de 100 serviços realizados a empresa A obteve 12 reclamações e a empresa B obteve 18 em 120 serviços.

Para isso deve-se usar o **prop.test()**, com um vetor com o número de reclamações de cada empresa e outro vetor com o número total de serviços de cada empresa.

```
prop.test(x = c(12, 18), n = c(100, 120))
##
## Results of Hypothesis Test
## -----
##
## Alternative Hypothesis:          two.sided
##
## Test Name:                      2-sample test for equality of proportions with continuity
##
## Estimated Parameter(s):         prop 1 = 0.12
```

```
##                                prop 2 = 0.15
##
## Data:                          c(12, 18) out of c(100, 120)
##
## Test Statistic:                X-squared = 0.20102339
##
## Test Statistic Parameter:      df = 1
##
## P-value:                       0.65389606
##
## 95% Confidence Interval:        LCL = -0.129378254
##                                UCL = 0.069378254
```

Observa-se que com 5% de nível de significância não rejeitamos H_0 , isto é, não há evidências estatísticas para acreditar que as proporções são diferentes

Teste de Kolmogorov-Smirnov (Teste de Normalidade)

-Grande parte dos problemas que encontramos em estatística são tratados com a hipótese que os dados são retirados de uma população com uma distribuição de probabilidade específica.

O teste de Kolmogorov - Smirnov é usado para determinar se duas distribuições de probabilidade subjacentes diferem uma da outra ou se uma das distribuições de probabilidade subjacentes difere da distribuição em hipótese. Deve-se lembrar que esse teste tem base em amostras finitas.

Um exemplo da aplicação do teste de Kolmogorov - Smirnov seria avaliar a hipótese H_0 : *Os dados não seguem uma distribuição normal*

Para esse cálculo no R, se usa o comando utilizado é o **ks.test()**.

A seguir um exemplo da utilização dessa função:

- Gere duas amostras aleatórias com distribuição Normal padrão e Qui-quadrado e teste via o teste de Kolmogorov-Smirnov se elas vem da mesma distribuição.

Assim, deve-se criar as duas amostras (amostra1 e amostra2) com **rnorm()** e **rchisq()** e realizar o teste Kolmogorov-Smirnov para as duas amostras

```
amostra1 = rnorm(1000)
amostra2 = rchisq(1000, 2)
ks.test(amostra1, amostra2)
##
## Results of Hypothesis Test
## -----
##
## Alternative Hypothesis:          two-sided
##
```

```
## Test Name:                Two-sample Kolmogorov-Smirnov test
##
## Data:                    amostra1 and amostra2
##
## Test Statistic:          D = 0.525
##
## P-value:                 0
```

Observa-se que com 5% de nível de significância rejeitamos H_0 , isto é, há evidências estatísticas para acreditar que as amostras foram geradas de distribuições diferentes

Análise de Séries Temporais

Uma série temporal é uma sequência de valores que uma variável assume em intervalos regulares durante um período de tempo. Mais especificamente, é uma sequência de pontos (dados numéricos) em ordem sucessiva, geralmente ocorrendo em intervalos uniformes.

Em modelos de regressão linear com dados cross-section a ordem das observações é irrelevante para a análise. Já em séries temporais a ordem dos dados é fundamental. Assim, uma característica muito importante deste tipo de base de dados é que as observações vizinhas são dependentes e o interesse é analisar e modelar esta dependência.

Essas séries são de enorme importância pois descrevem inúmeros acontecimentos do nosso dia a dia, sendo um dos objetos de estudo de diversas áreas. A seguir alguns exemplos de aplicações dessas séries:

- Economia: preços diários de ações; taxa de desemprego.
- Medicina: níveis de eletrocardiograma ou eletroencefalograma.
- Epidemiologia: casos semanais de sarampo; casos mensais de AIDS.
- Meteorologia: temperatura diária; registro de marés.

Comando Time Series

O Software R já possui um função especial para criar objetos de séries temporais, o comando **ts()**. Nele deve-se delimitar a base de dados (que deve ser uma série temporal), a frequência dessa base - por exemplo, se for um dado diário: a frequência será 7, se for mensal a frequência será 12 - e indicar o início que é a data que essa série começa. Seu enunciado é *ts(Base de dados lida, start=c(Unidade de comeco, Segmento dessa unidade que se começa), freq = x)*.

A seguir um exemplo da utilização da função **ts()** sobre a base de dados da energia natural afluyente do Brasil (que é um série temporal). Como essa base de dado é bem grande, um função interessante para garantir que ela foi lida corretamente é a **head()**. Ela mostra apenas as primeiras observações da base lida. A seguir cada subsistema está sendo armazenado em um objeto diferente, começando no ano 2000, no primeiro mês e com frequência mensal.

```
base = read.csv2("ENA_mensal_Mwmed_MLT_1931_20152.csv")
head(base)
##      Mwmed ENA_SE_Mwmedio ENA_S_Mwmedio ENA_NE_Mwmedio ENA_N_Mwmedio
## 1 jan/31      56898      7413      14124      10739
## 2 fev/31      86501      3312      13168      13585
## 3 mar/31      88663      3535      18892      22060
## 4 abr/31      64588      2356      20906      21727
## 5 mai/31      43086      17566     14301      9751
## 6 jun/31      32155      16508      7186      5648
```

```
sudeste = ts(base[, 2], start = c(1931, 1), freq = 12)
sul = ts(base[, 3], start = c(1931, 1), freq = 12)
nordeste = ts(base[, 4], start = c(1931, 1), freq = 12)
norte = ts(base[, 5], start = c(1931, 1), freq = 12)
```

Continuando a análise dessa base, pode-se calcular as medidas estatística já mencionadas anteriormente. Dentre elas, será destacado a seguir o cálculo dos percentis de cada subsistema. Com o comando **quantile()** está sendo obtido, para cada subsistema, o valor que marca a parcela dos 5% dos menores valores, depois 10% e assim por diante.

```
percentil_SE = quantile(sudeste, probs = c(0.05, 0.1, 0.25, 0.5, 0.75, 0.9,
0.95))
percentil_SE
##      5%      10%      25%      50%      75%      90%      95%
## 14097.45 15847.50 20755.00 28911.00 45493.75 61061.10 70252.30
```

```
percentil_S = quantile(sul, probs = c(0.05, 0.1, 0.25, 0.5, 0.75, 0.9, 0.95))
percentil_S
##      5%      10%      25%      50%      75%      90%      95%
## 2565.85 3231.60 4735.50 7658.00 11825.00 17980.80 21813.90
```

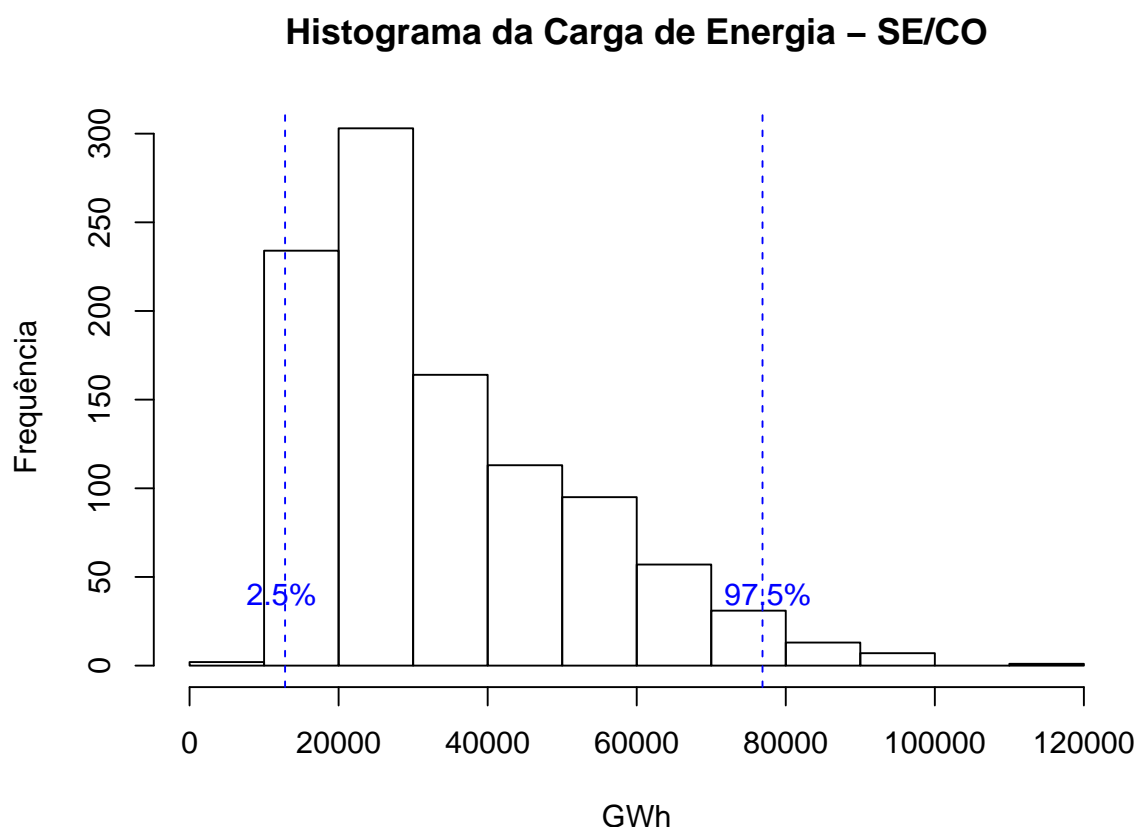
```
percentil_NE = quantile(nordeste, probs = c(0.05, 0.1, 0.25, 0.5, 0.75, 0.9,
0.95))
percentil_NE
##      5%      10%      25%      50%      75%      90%      95%
## 2334.15 2728.20 3622.50 5704.00 11390.75 16162.50 19213.65
```

```
percentil_N = quantile(norte, probs = c(0.05, 0.1, 0.25, 0.5, 0.75, 0.9, 0.95))
percentil_N
##      5%      10%      25%      50%      75%      90%      95%
## 1371.85 1559.70 2276.00 4827.00 10831.50 15383.50 18093.20
```

```
percentis = cbind(percentil_SE, percentil_S, percentil_NE, percentil_N)
```

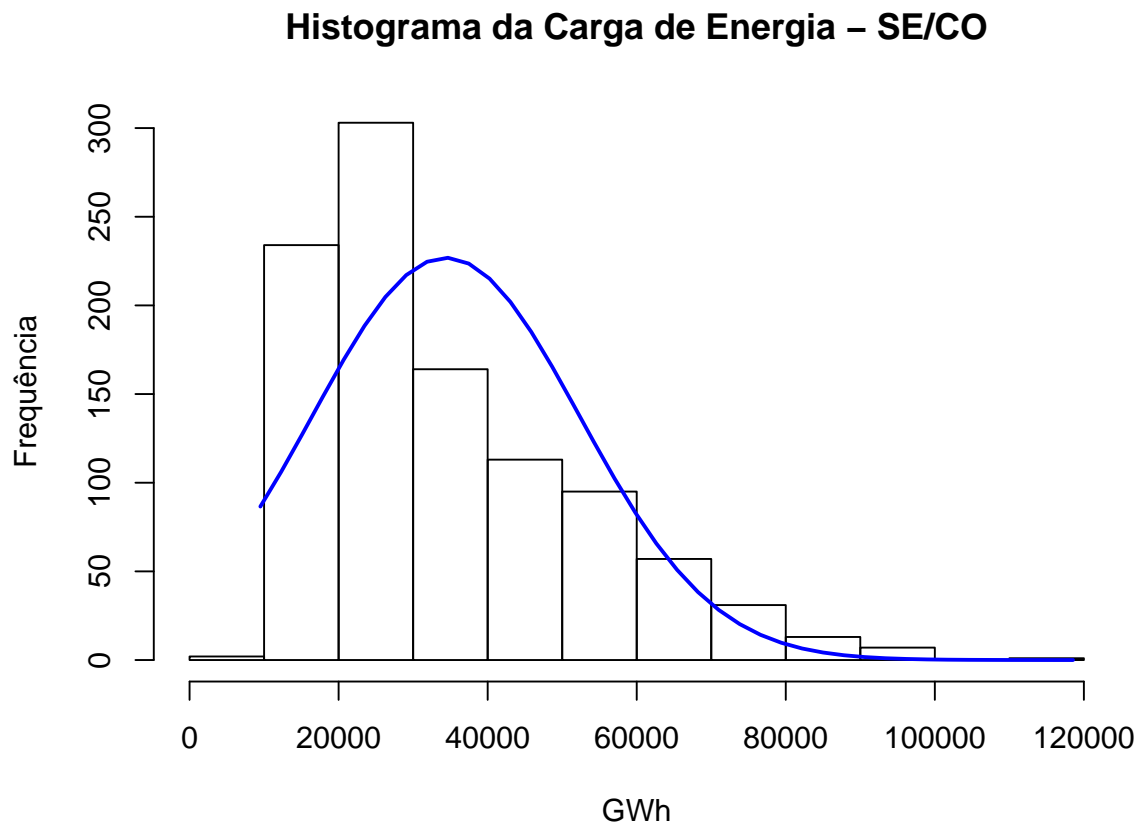
Pode ser interessante usar esses percentis nos histogramas, pois com ele é possível visualizar a linha que marca o nível de significância da base. Para isso será usado o comando **abline()** com as probabilidades de interesse.

```
hist(sudeste, main = "Histograma da Carga de Energia - SE/CO", xlab = "GWh",  
     ylab = "Frequência")  
abline(v = quantile(sudeste, probs = 0.025), lty = 2, col = 4)  
abline(v = quantile(sudeste, probs = 0.975), lty = 2, col = 4)  
text(x = c(quantile(sudeste, probs = 0.025) - 500, quantile(sudeste, probs = 0.975) +  
          700), y = 40, labels = c("2.5%", "97.5%"), col = 4)
```



Também é possível adicionar a densidade de uma distribuição conhecida, para verificar se a base se aproxima visualmente. Para isso, será criada uma distribuição conhecida com a mesma média da base. A seguir segue um exemplo com a distribuição normal na base da ENA.

```
h = hist(sudeste, main = "Histograma da Carga de Energia - SE/CO", xlab = "GWh",  
        ylab = "Frequência")  
xfit = seq(min(sudeste), max(sudeste), length = 40)  
yfit = dnorm(xfit, mean = mean(sudeste), sd = sd(sudeste))  
yfit = yfit * diff(h$mids[1:2]) * length(sudeste)  
lines(xfit, yfit, col = "blue", lwd = 2)
```



Através do resultado, pode-se visualizar que a a distribuição do sudeste se assemelha com a normal sem a calda da esquerda.

Teste de Jarque-Bera

Existe um teste especial para verificar se a base segue a distribuição normal, que é o teste Jarque Bera. Ele que só é valido para grandes amostras. Este teste é baseado na assimetria e curtose de uma variável aleatória. Simplificando, esse teste verifica se a variável tem distribuição normal ou não. Como a distribuição normal tem assimetria igual a 0 e curtose igual a 3, o teste dará positivo quando a variável tiver valores similares. Para realizar esse teste no R deve-se instalar o pacote tseries. O comando para o teste é **jarque.bera.test()**.

```
library(tseries)
```

```
jarque.bera.test(sudeste) #H0=normal
##
## Results of Hypothesis Test
## -----
##
## Alternative Hypothesis:
```



```
##
## Test Name:                Jarque Bera Test
##
## Data:                     sudeste
##
## Test Statistic:           X-squared = 217.95139
##
## Test Statistic Parameter: df = 2
##
## P-value:                  0
```

Correlação

Como existem mais de uma série nessa base (uma de cada subsistema), pode-se querer saber a correlação entre elas. Para isso, serão usados três métodos diferentes: Pearson, Spearman e Kendall. O comando usado no R é **corr(objeto, method="nome")**.

- Pearson: O coeficiente de correlação de Pearson (r) ou coeficiente de correlação produto-momento ou o r de Pearson mede o grau da correlação linear entre duas variáveis quantitativas. É um índice adimensional com valores situados entre -1,0 e 1.0 inclusive, que reflete a intensidade de uma relação linear entre dois conjuntos de dados. Quando $r=-1$ (correlação negativa perfeita), $r=0$ (sem correlação) e $r=1$ (correlação positiva perfeita).
- Spearman: O coeficiente de correlação de postos de Spearman, denominado pela letra grega rho, é uma medida de correlação não-paramétrica. Ao contrário do coeficiente de correlação de Pearson não requer a suposição que a relação entre as variáveis é linear, nem requer que as variáveis sejam quantitativas. Pode ser usado para as variáveis medidas no nível ordinal.
- Kendall: O coeficiente de correlação por postos de Kendall, denominado pela letra grega tau, é uma medida de associação para variáveis ordinais. Uma vantagem de tau sobre o coeficiente de Spearman é que pode ser generalizado para um coeficiente de correlação parcial.

A seguir serão realizados os três métodos para a base da energia natural afluyente (ENA), já usada anteriormente. Para não ocupar tanto espaço, serão calculadas as correlações apenas dos primeiros dados, dado que a base é muito grande.

```
iniciobase = base[1:5, ]
cor(iniciobase[, -1], method = "pearson")
##           ENA_SE_Mwmedio ENA_S_Mwmedio ENA_NE_Mwmedio ENA_N_Mwmedio
## ENA_SE_Mwmedio      1.00000000 -0.79618938  0.20230365  0.59268023
## ENA_S_Mwmedio      -0.79618938  1.00000000 -0.47554808 -0.71611187
## ENA_NE_Mwmedio      0.20230365 -0.47554808  1.00000000  0.90798370
## ENA_N_Mwmedio      0.59268023 -0.71611187  0.90798370  1.00000000
```

```
cor(iniciobase[, -1], method = "spearman")
##           ENA_SE_Mwmedio ENA_S_Mwmedio ENA_NE_Mwmedio ENA_N_Mwmedio
## ENA_SE_Mwmedio           1.0          -0.6           0.1           0.9
## ENA_S_Mwmedio           -0.6           1.0          -0.3          -0.7
## ENA_NE_Mwmedio           0.1          -0.3           1.0           0.5
## ENA_N_Mwmedio           0.9          -0.7           0.5           1.0
```

```
cor(iniciobase[, -1], method = "kendall")
##           ENA_SE_Mwmedio ENA_S_Mwmedio ENA_NE_Mwmedio ENA_N_Mwmedio
## ENA_SE_Mwmedio           1.0          -0.4           0.0           0.8
## ENA_S_Mwmedio           -0.4           1.0          -0.2          -0.6
## ENA_NE_Mwmedio           0.0          -0.2           1.0           0.2
## ENA_N_Mwmedio           0.8          -0.6           0.2           1.0
```

Outra opção é usar o comando “rcorr” do pacote “Hmisc”, que também gera como resultado os níveis de significância das correlações:

Então primeiramente carregamos o pacote que já foi instalado previamente com a função `install.packages('Hmisc')`.

```
library(Hmisc)
```

E agora pode-se calcular as correlações das séries da base de dado com função `rcorr()`.

```
rcorr(as.matrix(iniciobase[, -1]), type = "pearson")
##           ENA_SE_Mwmedio ENA_S_Mwmedio ENA_NE_Mwmedio ENA_N_Mwmedio
## ENA_SE_Mwmedio           1.00          -0.80           0.20           0.59
## ENA_S_Mwmedio           -0.80           1.00          -0.48          -0.72
## ENA_NE_Mwmedio           0.20          -0.48           1.00           0.91
## ENA_N_Mwmedio           0.59          -0.72           0.91           1.00
##
## n= 5
##
##
## P
##           ENA_SE_Mwmedio ENA_S_Mwmedio ENA_NE_Mwmedio ENA_N_Mwmedio
## ENA_SE_Mwmedio           0.1070           0.7442           0.2922
## ENA_S_Mwmedio 0.1070           0.4182           0.1736
## ENA_NE_Mwmedio 0.7442           0.4182           0.0330
## ENA_N_Mwmedio 0.2922           0.1736           0.0330
```

```
rcorr(as.matrix(iniciobase[, -1]), type = "spearman")
##           ENA_SE_Mwmedio ENA_S_Mwmedio ENA_NE_Mwmedio ENA_N_Mwmedio
## ENA_SE_Mwmedio           1.0          -0.6           0.1           0.9
## ENA_S_Mwmedio           -0.6           1.0          -0.3          -0.7
```

```
## ENA_NE_Mwmedio      0.1      -0.3      1.0      0.5
## ENA_N_Mwmedio       0.9      -0.7      0.5      1.0
##
## n= 5
##
##
## P
##          ENA_SE_Mwmedio ENA_S_Mwmedio ENA_NE_Mwmedio ENA_N_Mwmedio
## ENA_SE_Mwmedio          0.2848      0.8729      0.0374
## ENA_S_Mwmedio  0.2848          0.6238      0.1881
## ENA_NE_Mwmedio 0.8729      0.6238          0.3910
## ENA_N_Mwmedio  0.0374      0.1881      0.3910
```

Análise por mês

Para se analisar uma série por mês, primeiramente tem que se criar uma base que guarde esses valores já separadamente. Ao olhar o arquivo se parece muito com o de séries temporais já gerados, porém nesse caso será um data.frame/matriz. Será exemplificada essa análise com a coluna sudeste na base de dados da ENA. Para armazenar os dados separadamente por cada mês está sendo usado a função **which()** junto com a função **cycle()**. Assim foi possível separar todos os meses janeiro de todos os anos, fevereiro de todos os anos e assim em diante.

```
head(sudeste)
## [1] 56898 86501 88663 64588 43086 32155
```

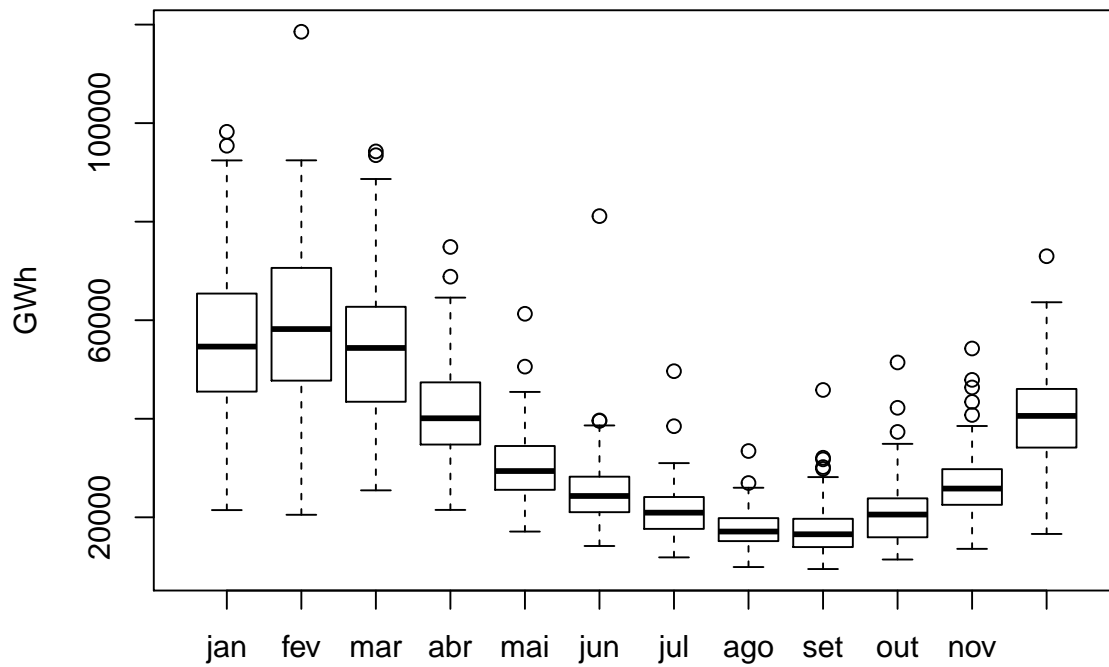
```
mes_SE = cbind(sudeste[which(cycle(sudeste) == 1)], sudeste[which(cycle(sudeste) ==
2)], sudeste[which(cycle(sudeste) == 3)], sudeste[which(cycle(sudeste) ==
4)], sudeste[which(cycle(sudeste) == 5)], sudeste[which(cycle(sudeste) ==
6)], sudeste[which(cycle(sudeste) == 7)], sudeste[which(cycle(sudeste) ==
8)], sudeste[which(cycle(sudeste) == 9)], sudeste[which(cycle(sudeste) ==
10)], sudeste[which(cycle(sudeste) == 11)], sudeste[which(cycle(sudeste) ==
12)])

colnames(mes_SE) = c("jan", "fev", "mar", "abr", "mai", "jun", "jul", "ago",
"set", "out", "nov", "dec")
```

Agora é possível plotar os boxplots por mês, permitindo a visualização da tendência de cada mês do ano. Assim, é possível identificar o caráter sazonal das séries da energia natural afluyente de cada subsistema e observar os períodos de seca e de chuvas.

```
boxplot(mes_SE, main = "Boxplot da Carga de Energia - SE/CO", ylab = "GWh")
```

Boxplot da Carga de Energia – SE/CO

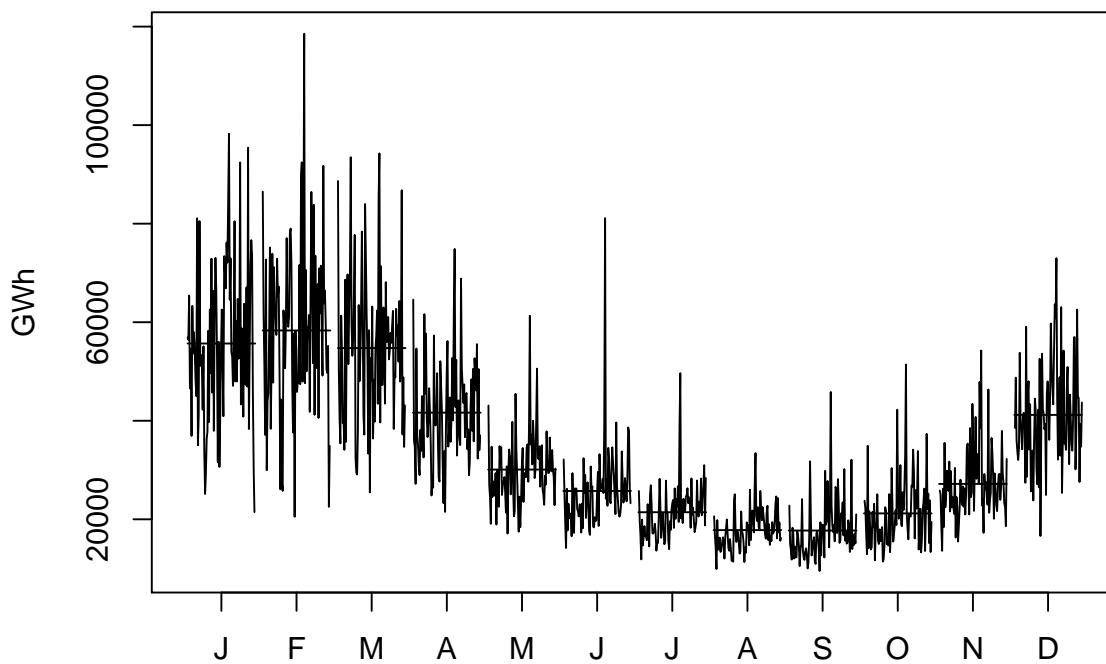


Monthplots

Por mês, também é possível usar o comando **Monthplot()**, que apresenta a média dos meses com uma linha horizontal e todos os valores dos meses durante o histórico numa linha vertical. Esse gráfico é interessante também para verificar o desvio dos valores dos meses em relação a média.

```
monthplot(sudeste, main = "Monthplot da Carga de Energia - SE/CO", ylab = "GWh")
```

Monthplot da Carga de Energia – SE/CO



FAC e FACP

A função de autocorrelação mede o grau de correlação de uma variável, em um dado instante, consigo mesma, em um instante de tempo posterior. Ela permite que se analise o grau de irregularidade de um sinal. Pode ser definida como a razão entre a autocovariância e a variância para um conjunto de dados.

Uma das utilidades da função de autocorrelação é na determinação do passo da reconstrução para se obter a não conformidade presente no processo, caso haja alguma.

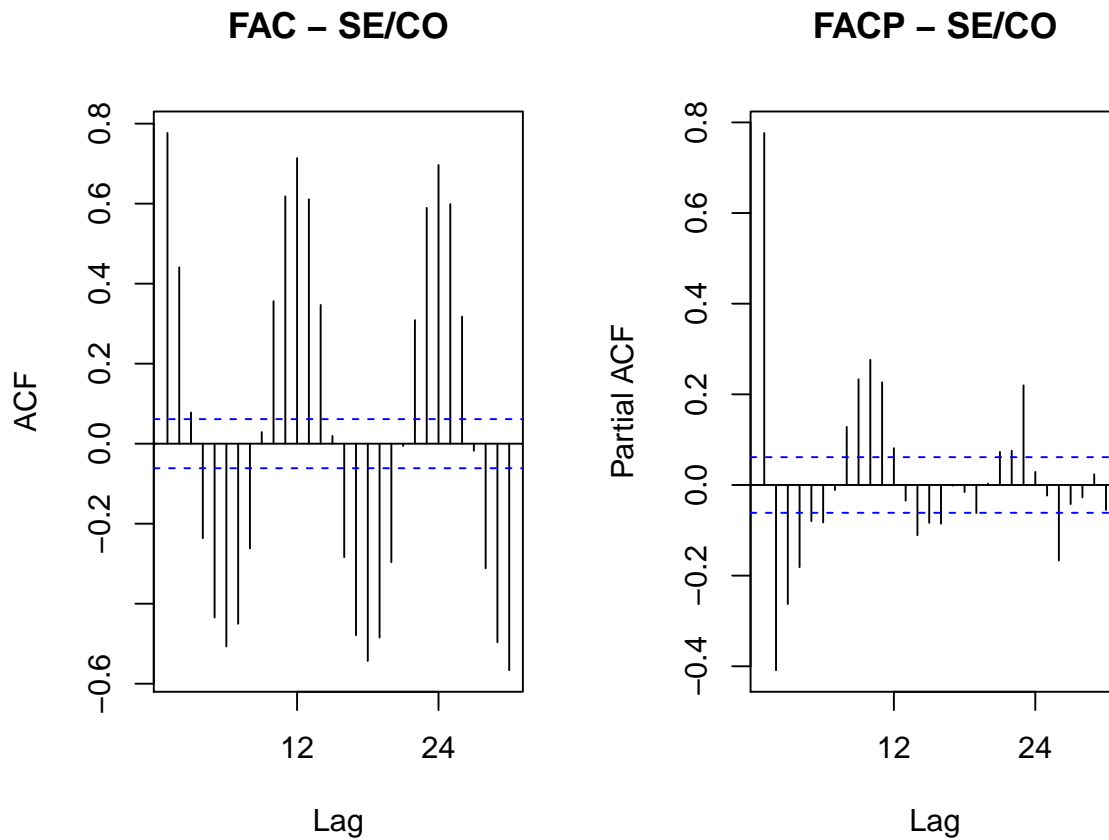
No R, é possível verificar as Funções de Autocorrelação (FAC) e Funções de Autocorrelação parcial (FACP). Note que os comandos usados são **Acf()** e **Pac()** ao invés de **acf()** e **pacf()**, pois os primeiros geram eixos “estranhos” enquanto os demais corrigem esse defeito. Para usar tais comandos, o pacote ‘forecast’ tem que estar carregado.

No exemplo a seguir, primeiro se carregou o pacote ‘forecast’, que já estava instalado.

```
library(forecast)
```

Para depois calcular as Funções de Autocorrelação e as Funções de Autocorrelação parcial do subsistema sudeste - mesmo que foi usado nas análise anteriores.

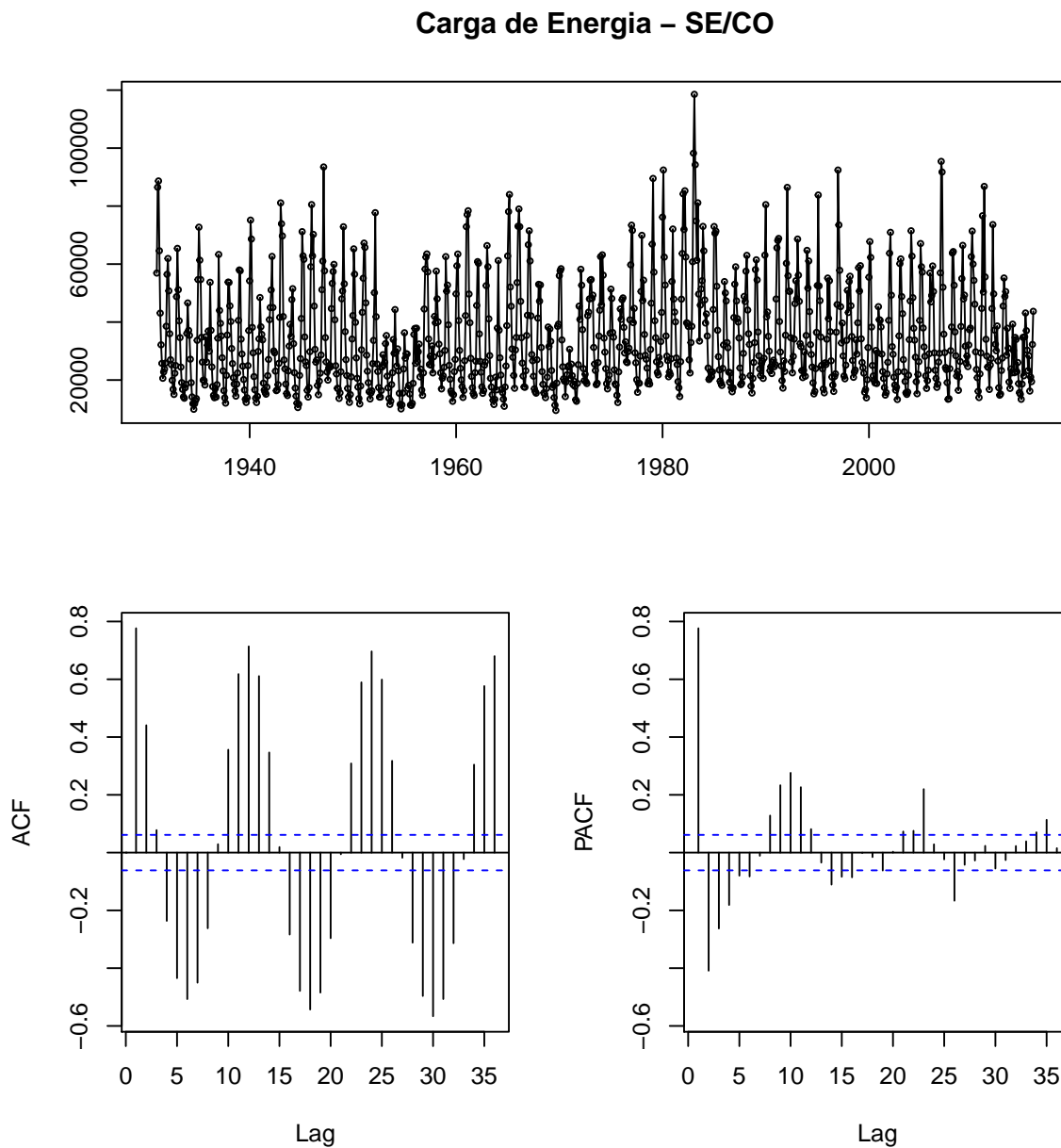
```
par(mfrow = c(1, 2))
Acf(sudeste, main = "FAC - SE/CO")
Pacf(sudeste, main = "FACP - SE/CO")
```



```
par(mfrow = c(1, 1))
```

Outra opção é usar o comando possível para o mesmo cálculo é o **tsdisplay()**, que produz os mesmos gráficos das funções **Acf()** e **Pacf()** e ainda uma previsão da continuação da série no futuro, facilitando a identificação da tendência da série.

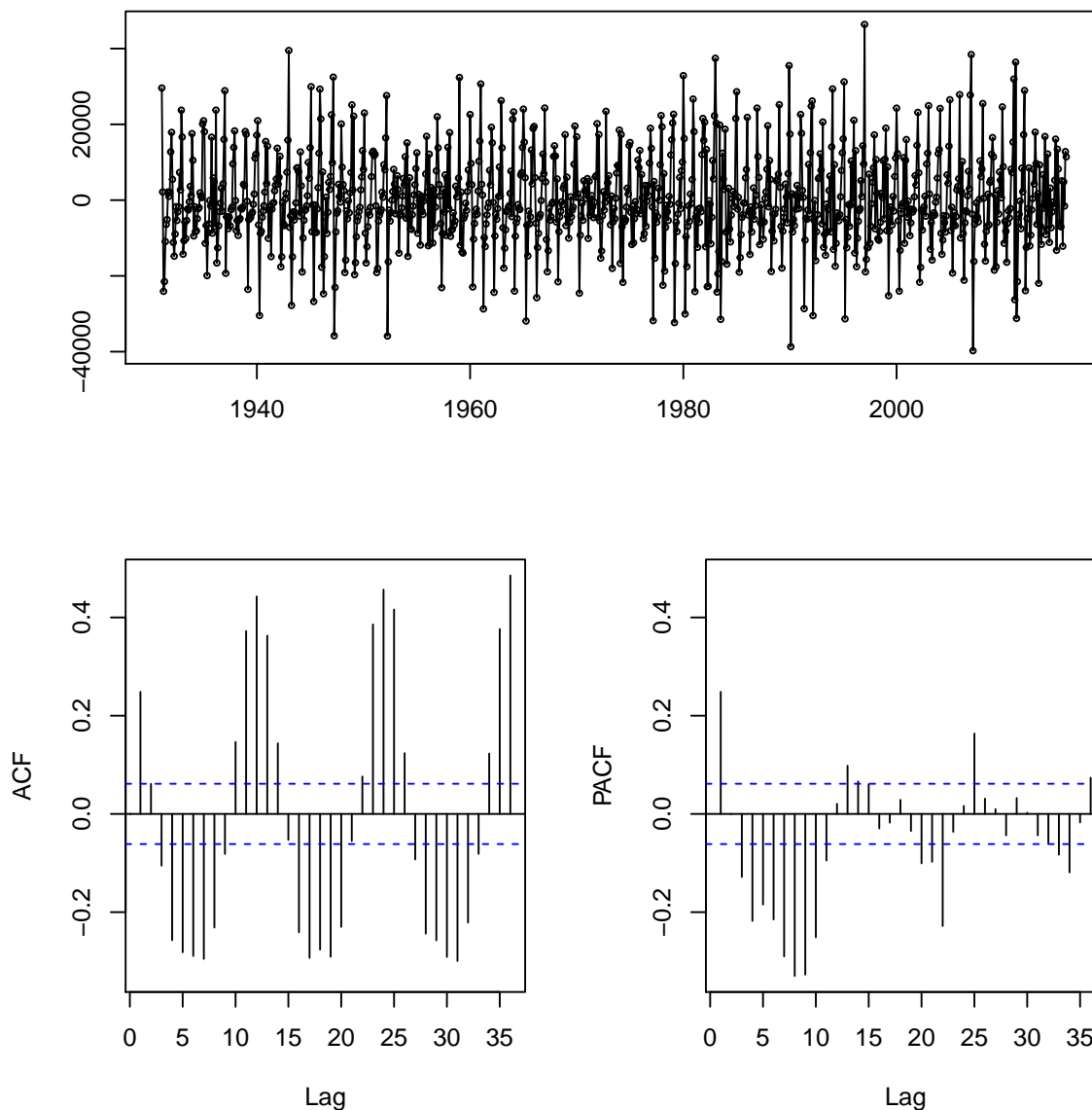
```
tsdisplay(sudeste, main = "Carga de Energia - SE/CO")
```



Repare que a FAC demonstrou um decaimento lento, por conta da não estacionariedade das séries (tendência crescente), para “consertar” isso uma solução é diferenciar 1 vez a série. Para realizar essa diferenciação, se usa o comando `se_diff()`

```
se_diff = diff(sudeste)
tsdisplay(se_diff, main = "1ª diferença da Carga de Energia – SE/CO")
```

1ª diferença da Carga de Energia – SE/CO



Na nova série, pode-se observar que a tendência de decaimento foi anulada.

Conclusão

Na presente apostila é possível encontrar a apresentação de conceitos básicos do software R assim como a aplicação por meio de exemplos. O conhecimento apresentado é de suma importância na formação do engenheiro de produção por permitir a utilização de um software livre de fácil aplicação mesmo em problemas de difícil solução.

Além das seções aqui apresentadas é possível expandir o projeto para apresentar ao leitor soluções de problemas específicos, como por exemplo a criação de uma tábua de mortalidade ou

geração de simulações do andamento de uma fila de banco.

Referências

Adrian Trapletti and Kurt Hornik (2015). **tseries: Time Series Analysis and Computational Finance**. R package version 0.10-34.

Braun, W.J. & Murdoch, D.J. (2007) **A first course in statistical programming with R**. Cambridge University Press, Cambridge.

Chambers, J.M. (2008) **Software for data analysis: Programming with R**.

Frank E Harrell Jr, with contributions from Charles Dupont and many others. (2016). **Hmisc: Harrell Miscellaneous**. R package version 3.17-4. <https://CRAN.R-project.org/package=Hmisc>

Hyndman RJ (2016). **forecast: Forecasting functions for time series and linear models**. R package version 7.1, .

Hyndman RJ and Khandakar Y (2008). “Automatic time series forecasting: the forecast package for R.” **Journal of Statistical Software**, 26(3), pp. 1-22. .

Lukasz Komsta and Frederick Novomestky (2015). **moments: Moments, cumulants, skewness, kurtosis and related tests**. R package version 0.14. <https://CRAN.R-project.org/package=moments>

Millard SP (2013). **EnvStats: An R Package for Environmental Statistics**. Springer, New York. ISBN 978-1-4614-8455-4, .

R Core Team (2016). **R: A language and environment for statistical computing**. R Foundation for Statistical Computing, Vienna, Austria. URL <https://www.R-project.org/>.

RStudio Team (2015). **RStudio: Integrated Development for R**. RStudio, Inc., Boston, MA URL <http://www.rstudio.com/>.

Yihui Xie (2014) **knitr: A Comprehensive Tool for Reproducible Research in R**. In Victoria Stodden, Friedrich Leisch and Roger D. Peng, editors, *Implementing Reproducible Computational Research*. Chapman and Hall/CRC. ISBN 978-1466561595

Yihui Xie (2015) **Dynamic Documents with R and knitr**. 2nd edition. Chapman and Hall/CRC. ISBN 978-1498716963

Yihui Xie (2016). **knitr: A General-Purpose Package for Dynamic Report Generation in R**. R package version 1.12.3.

Anexo

Energia Natural Afluente por subsistema do Brasil: uma análise de representatividade da Média de Longo Termo (MLT)

RESUMO

A Média de Longo Termo é a média das vazões naturais médias, correspondentes a um mesmo período, verificadas durante a série histórica de observações. Essa variável é de suma importância para o sistema energético Brasileiro, pois é utilizada nos modelos de otimização do despacho hidrotérmico. Este trabalho tem como objetivo a verificação da aderência dos valores mensais da Média de Longo Termo vigente com os valores observados em 2015 da série de Energia Natural Afluente. Para cumprir esse objetivo foram utilizados gráficos de séries temporais, histogramas e boxplots para verificação visual do comportamento da série além de inferência estatística com testes de hipótese. Foi possível então concluir que para todos os meses dos subsistemas Sul e Nordeste e alguns meses do subsistema Norte os valores considerados de MLT atualmente são satisfatórios, enquanto que para o subsistema Sudeste/Centro-Oeste nenhum mês apresentou aderência ao histórico de dados.

PALAVRAS CHAVE: Média de Longo Termo, Sistema Energético Brasileiro, Aderência.

Tópicos: EST, EN.

ABSTRACT

The Long Term Mean is the average of the natural inflow rate corresponding to the same period, recorded over the time series of observations. This variable is an important paramount to the Brazilian energy system, since it is used in the hydrothermal dispatch optimization models. This study aims to verify the compliance of the monthly values of the Long Term Mean currently used with the values observed in 2015 for the Natural Inflow Energy. To reach this goal it was used time series graphs, histograms and box plots to visually check the behavior of the series and statistical inference hypothesis test. It was concluded that for every month of the South and Northeast subsystems and some months of North subsystem the LTM values considered nowadays are satisfactory, while for the Southeast/Miwest subsystem no month showed adherence to the historical data.

KEYWORDS: Long Term Mean, Brazilian Energy System, Adherence.

Paper Topics: EST, EN.

Introdução

O Sistema Interligado Nacional (SIN), operado pelo Operador Nacional do Sistema (ONS), é o sistema de produção e transmissão de energia elétrica do Brasil, considerado um sistema hidrotérmico de grande porte, com forte predominância de usinas hidrelétricas (72,6% da geração total de energia em 2014) [ONS 2014]. Considerado único em âmbito global por conta de seu tamanho, o SIN possui múltiplos proprietários formados pelas empresas das regiões Sul, Sudeste, Centro-Oeste, Nordeste e parte da região Norte. Apenas 1,7% da energia produzida no país encontra-se fora do SIN, em pequenos sistemas isolados localizados principalmente na região amazônica [ONS 2014].

Planejar o setor elétrico brasileiro significa, basicamente, tomar decisões a cerca do despacho das hidroelétricas e termoeletricas, com o risco de perdas financeiras ou racionamento no caso de decisões equivocadas, como aconteceu em 2001, quando três dos quatros subsistemas foram afetados. Assim, o planejamento da operação energética envolve determinar metas para a geração das usinas hidrelétricas e termelétricas para cada etapa ao longo do horizonte de estudo, dada a demanda de energia, as restrições de operação das plantas e as restrições do sistema elétrico [Pereira 1989]. Como pode ser visto é bastante claro que a operação de um sistema elétrico como o brasileiro é uma tarefa bastante complexa dada a sua extensão e singularidade no mundo.

Uma das características mais importantes de um sistema de geração com predominância hidroelétrica é a forte dependência dos regimes de afluência. Como tal, a incerteza associada ao planejamento energético requer a modelagem estocástica das séries hidrológicas de forma adequada e coerente. Para o Brasil, os modelos que geram cenários hídricos, a fim de otimizar o desempenho da operação do sistema tem como variável estocástica a Energia Natural Afluente ou ENA [Oliveira, Souza e Marcato 2015]. A ENA consiste no potencial de energia que pode ser produzida a partir das vazões naturais afluentes aos reservatórios, seus valores são expressos em MW médios ou em percentual da média histórica de longo termo, MLT. Por sua vez, a partir do histórico de vazões, o ONS gera uma média de ENA para cada mês, sendo esse valor o representativo da média de longo prazo para o ano vigente.

A partir dos valores estimados de MLT são realizados os planejamentos de operação e geração de energia. Dada essa importância, o objetivo deste trabalho é o de verificar, a partir de testes estatísticos e visualização gráfica, se a MLT considerada está sendo representativa para cada subsistema ao comparar as vazões observadas em 2015 com os valores de MLT vigentes.

Material e Métodos

Análise Descritiva das Séries

O sistema energético brasileiro é dividido em quatro grandes subsistemas: Sudeste/Centro-Oeste (SE/CO), Sul (S), Nordeste (NE) e Norte (N). O subsistema SE/CO abrange as regiões Sudeste e Centro-Oeste do país, além dos estados de Rondônia e Acre. Já o subsistema Sul abrange a região Sul do país, enquanto que o subsistema Norte é responsável pela região Norte com exceção do estado do Maranhão. Por fim, o subsistema Norte cobre parte dos estados do Amapá, Pará, Tocantins, Maranhão e Amazonas. As séries de Energia Natural Afluente serão tratadas de forma a representar os grandes subsistemas, sendo assim, as análises serão feitas em relação a quatro diferentes séries temporais.

Tanto as séries de Energia Natural Afluente quanto os valores da Média de Longo Termo atualmente utilizadas podem ser obtidas através do site do Operador Nacional do Sistema Elétrico [ONS 2014]. Todas as análises aqui realizadas foram feitas a partir do software R [R Core Team 2015] e dos pacotes knitr [Xie 2016] [Xie 2015] [Xie 2014], kfigr [Koohafkan 2015], TSA [Chan & Ripley 2012] e tseries [Trapletti & Hornik 2015].

No presente capítulo será realizada a análise comportamental das quatro séries de ENA onde serão apresentados gráficos para facilitar a visualização do leitor, medidas estatísticas e também testes de hipóteses. Na Figura 1 é possível observar a evolução temporal da ENA de cada subsistema no período de janeiro de 1931 a dezembro de 2015. Identifica-se primeiramente um padrão semelhante entre os subsistemas SE/CO, NE e N, em que se evidencia o caráter sazonal dessas séries. Já no subsistema Sul, esse padrão sazonal não se mostra de forma clara, onde observa-se valores extremos (outliers) em diversos períodos. Apesar das diferenças, uma constante em todos os subsistemas é o caráter não tendencioso dessas séries, de modo que no longo-prazo, não existe uma expectativa de aumento, nem de redução do nível da série.

Histórico Energia Natural Afluente (MW médio)

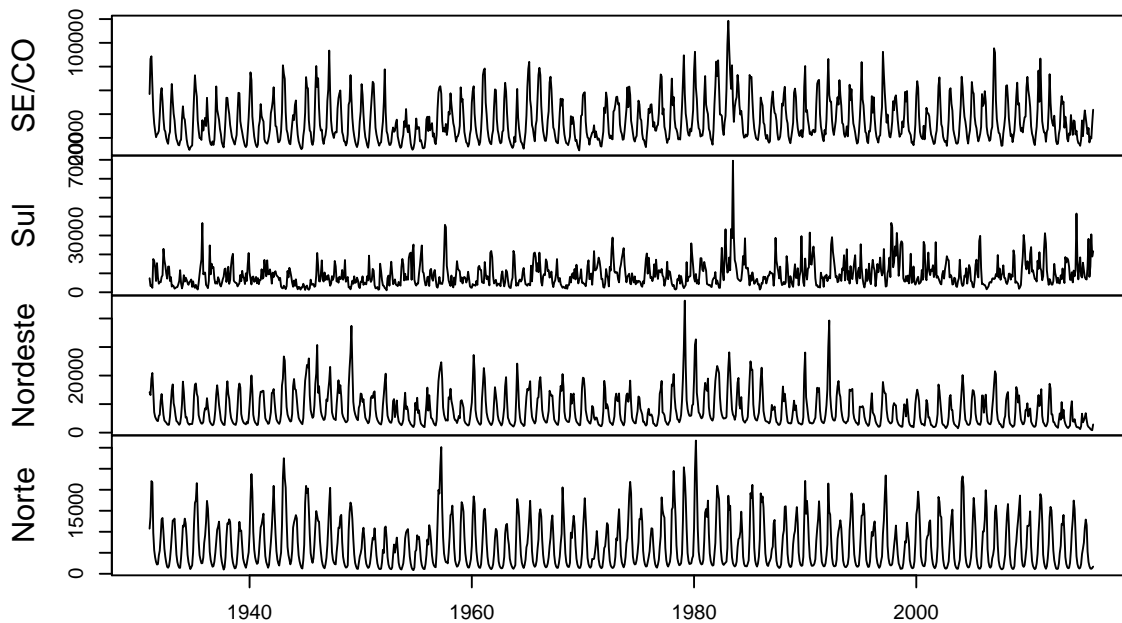


Figure 1: Gráficos das séries de Energia Natural Afluente

Na Tabela 1 são apresentadas algumas das mais importantes medidas estatísticas para cada subsistema como média, mediana, desvio padrão, coeficiente de variação, assimetria, curtose, mínimo, máximo e quartis. Para maiores detalhes ver [Morettin e Bussab 2004]. Constata-se que o subsistema SE/CO possui as maiores medidas centrais (média e mediana), o que é consistente dado que é o subsistema que apresenta os maiores valores de ENA para todos os períodos.

Nota-se também que apesar do subsistema SE/CO ter o maior valor de desvio padrão, o subsistema Norte é o que apresenta a maior variância relativa, com um coeficiente de variação de 0.82, superando os demais subsistemas.

Table 11: Análise descritiva das séries de Energia Natural Afluente

	SE/CO	S	NE	N
Média	34398.92	9371.5	8032.95	6988.22
Mediana	28911.00	7658.0	5704.00	4827.00
Desv. Pad.	17936.25	6550.1	5965.31	5697.59
Coef. Var.	0.52	0.7	0.74	0.82
Assimetria	2.10	2.1	2.10	2.10
Curtose	4.99	5.0	4.99	4.99
Min.	9501.00	1043.0	846.00	868.00
Max.	118545.00	69521.0	46244.00	31706.00
Q1	20755.00	4735.5	3622.50	2276.00
Q3	45493.75	11825.0	11390.75	10831.50

Os valores de assimetria quantificam quão simétrica é a distribuição dos dados, enquanto a medida de curtose quantifica se o formato da distribuição dos dados coincide com a distribuição Gaussiana. Se a assimetria for igual a 0, então os dados são perfeitamente simétricos, porém é aceitável que valores entre -1 e 1 também indiquem simetria dos dados. É esperado que uma distribuição Gaussiana possua valor de curtose igual a 3 (mesocúrtica), sendo assim, valores abaixo de 3 indicam uma distribuição mais “achatada” (platicúrtica) enquanto curtose acima de 3 representa distribuições mais “altas” (afuniladas) e são chamadas de leptocúrtica. Sendo assim, como verifica-se que todas as séries possuem valores semelhantes de assimetria (2.1) e curtose (4.99) espera-se que a distribuição dos dados não seja simétrica e possua um formato mais “afunilado”.

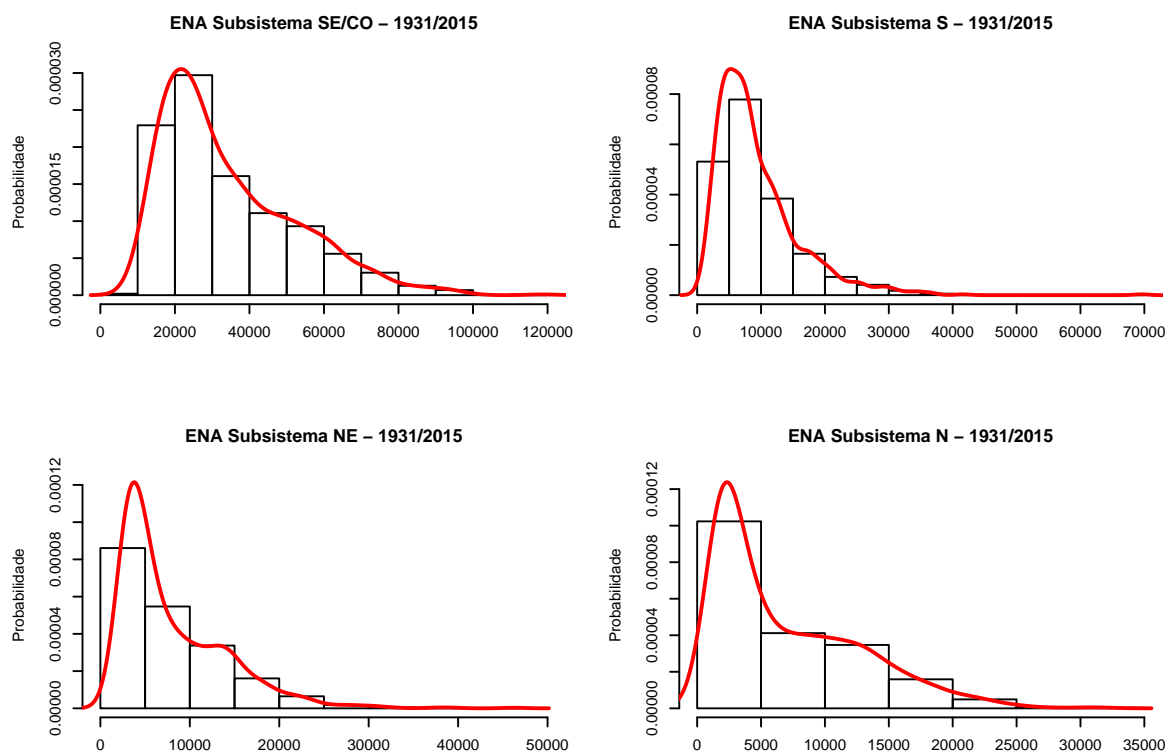


Figure 2: Histograma das séries de Energia Natural Afluente

O comportamento descrito pode ser confirmado pela Figura 2 que apresenta o histograma das ENA's conjuntamente com as funções de probabilidade de densidade adequadas. Concluindo assim que as séries dos subsistemas são assimétricas positivas e leptocúrticas, caracterizadas por distribuições de caudas pesadas, que apresentam uma distribuição afunilada e mais concentrada que a distribuição Normal (Gaussiana).

Para se analisar a distribuição das ENA observadas, foi realizado o teste de Wilcoxon-Mann-Whitney [Corder e Foreman 2014], cujo propósito consiste em determinar se duas distribuições de probabilidade subjacentes diferem uma da outra ou se assemelham. Isto é, trata-se do seguinte teste de hipótese: H_0 série de dados segue distribuição X versus H_1 série de dados não segue distribuição X .

Sendo assim, adotando nível de significância de 5%, se o pvalor encontrado for menor que 0.05 rejeita-se a hipótese dos dados possuírem distribuição X , enquanto que pvalor maior que 5% não rejeita-se a hipótese nula dos dados seguirem a distribuição X .

Table 12: Teste de aderência Wilcoxon-Mann-Whitney

	SE/CO	Sul	Nordeste	Norte
Normal	0.0046	0.00	0.023	0.025
Qui-Quadrado	0.0000	0.00	0.000	0.000
Gama	0.3264	0.76	0.248	0.173

	SE/CO	Sul	Nordeste	Norte
Weibull	0.0000	0.00	0.000	0.000
LogNormal	0.0000	0.00	0.000	0.000

De acordo com a Tabela 2 e o teste de hipóteses descrito acima, ao nível de 5% de significância todas as séries de ENA obtiveram valores significantes para aderência a distribuição Gama, isto é, não rejeita-se que os dados em estudo possuam uma distribuição Gama.

Outra informação que pode ser extraída da Tabela 1 é referente ao comportamento dos quartis das séries. Um quartil é qualquer um dos três valores que divide o conjunto ordenado de dados em quatro partes iguais, e assim cada parte representa 1/4 da amostra ou população, assim quartil 25% = Q1, quartil 50% = Q2 = Mediana e quartil 75% = Q3. Por exemplo, para o caso do subsistema SE/CO, 25% dos dados são menores que 20.755 MW médio. Esse tipo de estatística auxilia na análise de assimetria dos dados e pode ser representada visualmente pelo gráfico de Boxplot, como na Figura 3. Note que para todos os subsistemas a média está mais próxima do quartil 25% do que do quartil 75%, identificando uma assimetria já confirmada. Destaca-se também a dissimilaridade em valores absolutos dos subsistemas Sudeste/Centro-Oeste para os demais.

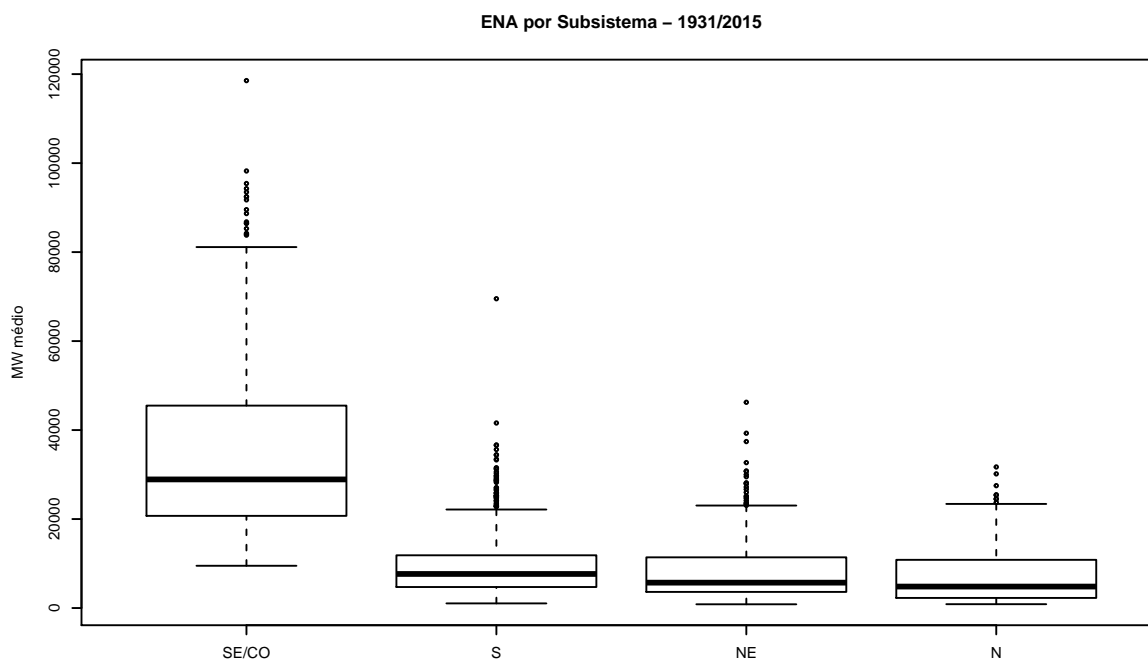


Figure 3: Boxplot das séries de Energia Natural Afluente

A série de Energia Natural Afluente, conforme visto anteriormente, é uma variável hidrológica que sofre total influência dos regimes de chuva, com isso é possível observar períodos ditos secos e úmidos. Sendo assim, é interessante analisar os valores históricos para cada mês.

Na Figura 4 estão representadas mensalmente os valores de ENA para cada um dos subsistemas em formato de boxplot. Pode-se observar que em três subsistemas, SE/CO, Nordeste e Norte, o caráter sazonal da ENA se faz presente onde nos meses entre Maio e Novembro há um período de seca e de Dezembro a Abril há um período mais úmido.

Uma grande singularidade no Brasil é a grande diferença dos regimes de chuva entre os subsistemas. Note que o subsistema Sul se difere amplamente dos demais, não contendo nenhum mês tipicamente seco ou úmido, fato evidenciado na grande semelhança dos boxplot de todos os meses. Esse comportamento, é um indicativo da falta de sazonalidade desse subsistema, se contrapondo ao restante do país.

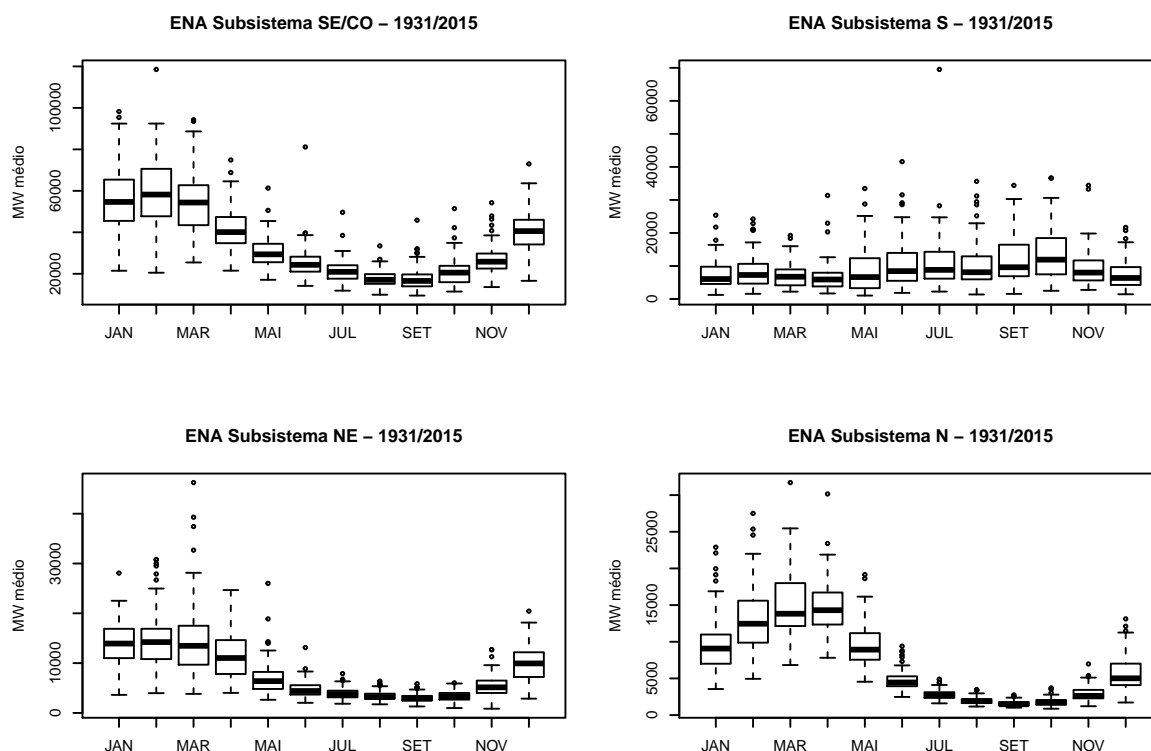


Figure 4: Boxplot das séries de Energia Natural Afluente

Aplicação e Resultados

Como dito na Introdução, o principal objetivo deste trabalho é a verificação da aderência da Média de Longo Termo vigente com os dados observados de Energia Natural Afluente para o ano de 2015. Sendo assim o primeiro passo será a comparação entre a ENA observada em 2015 e a MLT vigente utilizando o gráfico de barras conforme pode ser visto na Figura 5. Note que, com exceção do subsistema Sul, a MLT superestima todos os meses quando comparada com a ENA observada, chegando a uma diferença de ser realizada somente 15% da Energia Natural Afluente esperada pela MLT do mês de novembro no subsistema Nordeste como pode ser visto na Tabela 3.

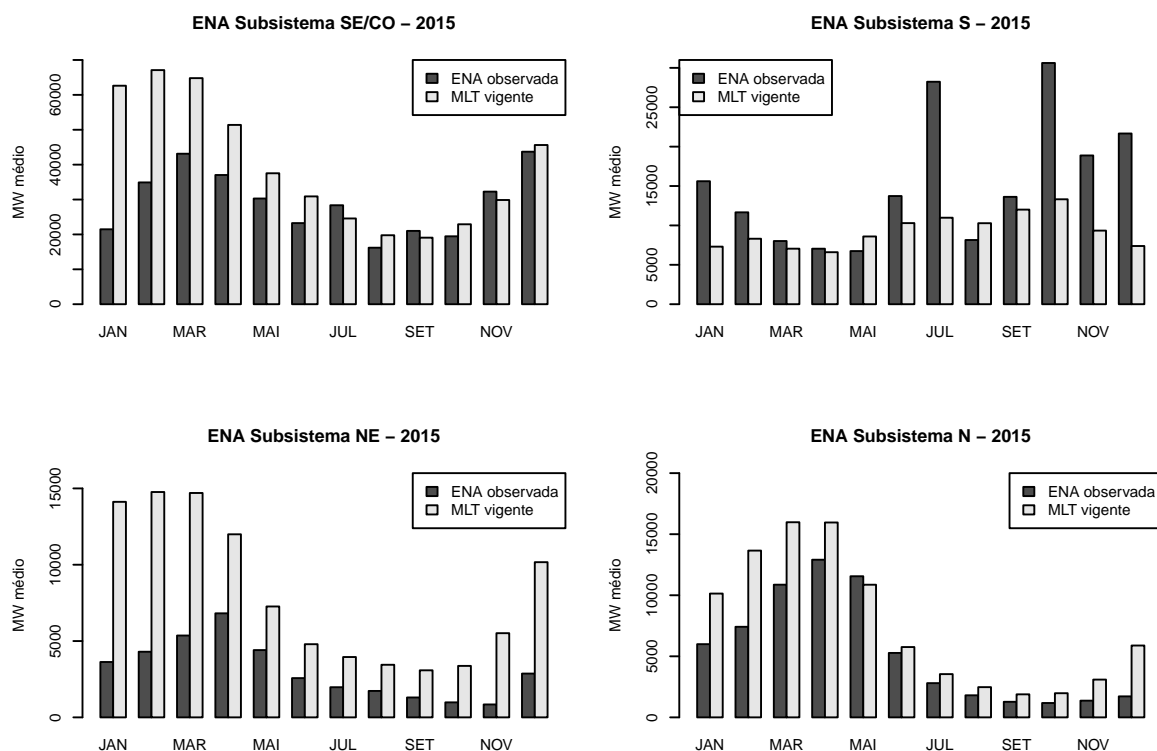


Figure 5: Comparação entre: ENA observada e MLT vigente

Table 13: % da ENA observada em relação a MLT vigente

	JAN	FEV	MAR	ABR	MAI	JUN	JUL	AGO	SET	OUT	NOV	DEZ
SE/CO	34	52	67	72	81	75	115	82	110	85	108	96
Sul	214	140	114	107	78	133	257	79	113	230	202	293
Nordeste	26	29	36	57	61	54	50	50	42	29	15	28
Norte	59	54	68	81	106	92	79	73	67	59	44	29

Visualmente concluiu-se que a média de longo termo não é representativa da Energia Natural Afluente, pois superestima a ENA esperada para três dos quatro subsistemas. Para uma primeira confirmação da hipótese de sobrestimação será construído um intervalo interquartilico (entre os quartis 1 e 3) para cada mês de cada subsistema utilizando o histórico da série de 1931 até 2014. Caso o valor da MLT vigente daquele mês se encontre dentro do intervalo interquartilico então pode-se dizer que os valores de MLT são válidos, caso contrário, os valores de MLT não são adequados.

Veja na Figura 6 que para os subsistemas Sul e Nordeste nenhum valor da MLT violou o intervalo interquartilico, enquanto que para os subsistemas Sudeste/Centro-Oeste e Norte uma sequência de meses da MLT ultrapassou o limite superior, confirmando uma superestimação dos valores esperados de Energia Natural Afluente.


$$H_0: \text{m\'edia ENA } m\hat{e}s_i = MLT_i$$

$$H_1: \text{m\'edia ENA } m\hat{e}s_i \neq MLT_i$$

Considerando, mais uma vez, um nível de significância de 5% observa-se na Tabela 4 que a Média de Longo Termo vigente para todos os meses dos subsistemas Sul e Nordeste é aderente a média da série histórica de ENA, para o subsistema Nordeste 7 de 12 meses (58.3%) da MLT não foram significantemente iguais a média histórica da ENA. Já para o subsistema Sudeste/Centro-Oeste, segundo o teste de hipóteses, nenhum mês da MLT foi representativo em relação à série histórica.

[illegible]

Conclusão

Dada a importância da variável Média de Longo Termo, este trabalho teve como objetivo a aplicação de testes estatísticos para averiguar a aderência de tal variável a série histórica de Energia Natural Afluente. Para atingir tal objetivo foram empregadas técnicas estatísticas de visualização via gráficos de séries temporais, histogramas e boxplots além de testes de hipóteses.

Em um primeiro momento foi estudado o comportamento das quatro séries de ENA, uma para cada subsistema. Conclui-se que, com exceção do subsistema Sul, as séries temporais possuem comportamento sazonal, identificando os períodos ditos secos (menos chuva) e úmidos (mais chuva). Também foi constatado que a distribuição de probabilidade dos quatro subsistemas era semelhante e confirmado via teste de hipóteses que tais distribuições poderiam ser consideradas advindas da distribuição Gama.

Para início da comparação entre os valores atualmente utilizados de Média de Longo Termo e a Energia Natural Afluente observada em 2015 foi utilizado um gráfico de barras, mostrando que para três dos quatro subsistemas o valor da MLT sobrestimou o realizado. Em seguida foi construído um intervalo de confiança a partir dos intervalos interquartílicos mensais das séries históricas e comparado com o valor vigente mensal da MLT. Tal exercício mostrou mais uma vez que a variável MLT para os subsistemas Sul e Nordeste é completamente satisfatória enquanto que para os subsistemas SE/CO e Norte vários meses não foram contemplados com a MLT. A última confirmação da aderência foi feita a partir do teste de comparação de média (t-student) onde a hipótese nula é de igualdade entre a média histórica da ENA e o valor vigente da MLT daquele mês. Mais uma vez o subsistema Sul e Nordeste obteve valores significantes para igualdade entre as médias, alguns meses subsistema Norte apresentaram o mesmo resultado, porém o subsistema SE/CO rejeitou a variável MLT para todos os meses.

Conclui-se então que a partir das análises e testes empregados a variável Média de Longo Termo é aderente às séries históricas dos subsistemas Sul e Nordeste, porém tal variável deve ser revista para os demais subsistemas, dado que os testes apontaram a não aderência da MLT para esses casos.

Referências

- Trapletti, A. and Hornik, K. 2015. tseries: Time Series Analysis and Computational Finance. R package version 0.10-34.
- Corder, G. W. and Foreman, D. I. 2014. “Nonparametric Statistics: A Step-by-Step Approach”. Wiley. ISBN 978-1118840313.
- Chan, K. and Ripley, B. 2012. TSA: Time Series Analysis. R package version 1.01. <https://CRAN.R-project.org/package=TSA>
- McDonald, J. H. 2008. Handbook of Biological Statistics Sparky House Publishing. Baltimore.
- Koohafkan, M. C. 2015. kfigr: Integrated Code Chunk Anchoring and Referencing for R Markdown Documents. R package version 1.2. <https://CRAN.R-project.org/package=kfigr>
- Morettin, P. A. and Bussab, W. de O. 2004. Estatística Básica. 5ª edição, Editora Saraiva.
- Oliveira, F. L. C., Souza, R. C. and Marcato, A. L. M. 2015. A Time Series Model for Building Scenarios Trees Applied to Stochastic Optimisation. International Journal of Electrical Power

& Energy Systems 67:16-38.

ONS. 2014. Operador Nacional Do Sistema Elétrico. www.ons.com.br.

ONS. 2015. Avaliação de Curto Prazo da Operação, Energias Naturais Afluentes dos Subsistemas. Operador Nacional do Sistema Elétrico. www.ons.org.br/operacao/enas_subsistemas.aspx

Pereira, M.V.F. 1989. Optimal Stochastic Operations Scheduling of Large Hydroelectric Systems. International Journal of Electric Power and Energy Systems 11: 161-69.

R Core Team. 2015. R: A Language and Environment for Statistical Computing. Vienna, Austria: R Foundation for Statistical Computing. www.R-project.org/.

Xie, Y. 2016. knitr: A General-Purpose Package for Dynamic Report Generation in R. R package version 1.12.3.

Xie, Y. 2015. Dynamic Documents with R and knitr. 2nd edition. Chapman and Hall/CRC. ISBN 978-1498716963

Xie, Y. 2014. knitr: A Comprehensive Tool for Reproducible Research in R. In Victoria Stodden, Friedrich Leisch and Roger D. Peng, editors, Implementing Reproducible Computational Research. Chapman and Hall/CRC. ISBN 978-1466561595