R

André Silva de Queiroz

Sumário

Capítulo 1

Análise Descritiva

Toda análise estatística deve-se iniciar a partir da análise descritiva dos dados, também chamada de análise exploratória. Essa importante etapa é o primeiro contato que o pesquisador tem com a base de dados[1].

1.1 Leitura dos dados no R

O R é uma linguagem de programação multipropósito focada na análise de dados. Todavia, entre as suas muitas funções não está o gerenciamento de dados em si. Ou seja, o fluxo de trabalho com o R sempre começa com a leitura dos dados a partir de uma fonte externa. Por isso mesmo, o R é capaz de ler arquivos de dados em inúmeros formatos. Desde os mais comuns como .txt ou .xlsx até os mais específicos como, por exemplo, .sav e .dta (arquivos dos softwares estatísticos SPSS e Stata, respectivamente).

1.1.1 Dados em arquivos de texto

O arquivo mais comum para armazenar dados para trabalhar no R é o formato de texto (.txt). Esse tipo de arquivo pode ser lido no R através da função `read.table()`.

```
> dados <- read.table(file = "base.txt", header = TRUE)
```

A função `read.table()` possui muitas "opções", formalmente chamadas de argumentos. Na chamada anterior aparecem dois deles: `file`, o único argumento obrigatório da função, e `header`. Os principais argumentos estão descritos na Tabela 1.1.

O argumento `sep`, apesar de omitido na chamada anterior, é muito importante e é prudente que esteja sempre explícito ao ler uma base de dados. O seu valor padrão é "um espaço em branco". Ou seja, a importação dos dados no R levará em conta que as variáveis no arquivo de texto estarão separadas por um espaço em branco. Isso pode

[1] Nesse capítulo será utilizada uma base de dados de uma pesquisa de opinião fictícia.

Tabela 1.1: Principais argumentos da função `read.table()`.

Argumento	Descrição
`file`	Nome do arquivo que contém os dados.
`header`	Indica se a primeira linha contém os nomes das variáveis.
`sep`	Símbolo que separa as colunas em variáveis distintas.
`dec`	Caractere que separa a parte decimal de um número.

ser um problema quando uma variável categórica tiver um rótulo composto por mais de uma palavra. Por exemplo, no nome de municípios ou de estados. A opção padrão do argumento `sep` irá interpretar "Distrito Federal" como observações distintas, "Distrito" e "Federal". Para resolver esse problema, procure utilizar arquivos de texto separados por tabulações e importe esse dados no R utilizando o argumento `sep = "\t"`.

A função `read.table()` apenas lê o arquivo de texto que contém a base de dados e "imprime" o resultado na tela do computador. Para realmente alocar o objeto na mémoria para que o R possa trabalhar com a base carregada é preciso utilizar o operador <-. Esse operador, que lembra o formato de uma seta à esquerda, salva o resultado da execução de um comando no R num objeto que se tornará disponível para utilização futura. Para consultar os objetos salvos na memória utilize a função `objects()`.

```
> dados <- read.table(file = "base.txt", header = TRUE, sep = "\t")
```

Uma função bastante útil é a `file.choose()`. Ao adicionar essa função como a opção do argumento `file` na chamada anterior, uma janela é aberta que permite selecionar o arquivo de texto desejado navegando pelas pastas do computador.

Arquivos separados por vírgulas (.csv) podem ser lidos no R mudando apenas o valor de `sep`. Faça `sep = ","` ou `sep = ";"`, dependendo se o arquivo é separado por vírgulas ou pontos e vírgulas, respectivamente.

Para saber a lista completa dos argumentos possíveis e uma descrição detalhada sobre a função `read.table()`, digite no R o nome da função precedido por um ponto de interrogação.

```
> ?read.table
```

Esse último comando apresentado pode ser usado em qualquer função do R.

1.1.2 Dados em arquivos do Excel

O R não lê nativamente arquivos .xls e .xlsx, mas isso absolutamente não significa que o R não trabalhe com esses tipos de arquivos. Apesar dos arquivos do Excel serem uma forma muito comum de armazenar dados, eles fazem parte de um pacote maior de

softwares proprietários. Provavelmente por isso os desenvolvedores do R não dão suporte direto a esse formato.

Para ler arquivos .xls e .xlsx no R é necessário utilizar algum pacote[2] adicional que cumpra essa finalidade. Existem alguns pacotes que tornam o R capaz de importar arquivos do Excel. Nos exemplos que seguem, será utilizada a biblioteca `xlsx`.

Num computador com acesso à internet, para instalar um pacote adicional no R deve ser utilizada a função `install.packages()`.

```
> install.packages(pkgs = "xlsx")
```

Uma vez instalado, o pacote pode ser carregado no R através da função `library()`.

```
> library(package = xlsx)
```

Depois de carregada com sucesso, a biblioteca disponibiliza a função `read.xlsx()`. Essa função permite ler tanto os arquivos .xls quanto os arquivos .xlsx.

```
> dados <- read.xlsx(file = "base.xlsx", sheetIndex = 1)
```

A chamada anterior apresenta a função `read.xlsx()` com seus dois argumentos obrigatórios: `file` e `sheetIndex`. A Tabela 1.2 descreve esses e alguns outros argumentos que podem ser úteis dessa função.

Tabela 1.2: Principais argumentos da função `read.xlsx()`.

Argumento	Descrição
`file`	Nome do arquivo que contém os dados.
`sheetIndex`	Número da posição da planilha a ser lida.
`sheetName`	Nome da planilha a ser lida.
`header`	Indica se a primeira linha contém os nomes das variáveis.
`colIndex`	Número das colunas a serem lidas.
`rowIndex`	Número das linhas a serem lidas.

Ler arquivos do Excel no R pode ser um problema em algumas situações. Por exemplo, se o computador não tiver acesso a internet não será possível baixar essas bibliotecas adicionais. Em todo caso, uma base de dados num arquivo Excel pode sempre ser salva num arquivo de texto separado por tabulações e depois lida no R através da função `read.table()`, como mostrado na seção anterior.

[2] Um pacote, ou uma biblioteca de funções, são programas desenvolvidos em R e disponibilizados por seus autores. Em geral gratuitos, os pacotes podem ser baixados através de repositórios na internet. O mais famoso repositório de R pode ser acessado através do endereço https://cran.r-project.org/.

1.2 Descrição dos dados

Quando uma base de dados é lida no R, a primeira questão que surge é saber se o arquivo de dados foi carregado corretamente. Existem algumas funções que permitem avaliar isso. Inicialmente, a função `head()` mostra as primeiras seis linhas da base.

```
> head(x = dados)

  id      sexo idade    vinculo estacionamento instalacoes ambiente
1  1  Feminino    36   Servidor        Regular         Bom      Bom
2  2  Feminino    42   Servidor        Regular       Ótimo      Bom
3  3  Feminino    31   Servidor           Ruim       Ótimo      Bom
4  4 Masculino    22 Estagiário            Bom       Ótimo    Ótimo
5  5  Feminino    36   Servidor            Bom         Bom      Bom
6  6  Feminino    29   Servidor        Regular         Bom    Ótimo
```

De maneira análoga, a função `tail()` mostra as seis últimas linhas da base de dados.

Outras duas funções importantes nessa etapa de avaliar o sucesso no carregamento da base são: `summary()` e `str()`. A primeira mostra, de forma resumida, uma descrição das observações em cada variável. A segunda descreve as classes atribuídas automaticamente pelo R às variáveis da base.

```
> summary(object = dados)

       id                sexo         idade              vinculo
 Min.   :  1,00   Feminino :177   Min.   :18,00   Estagiário  : 48
 1st Qu.: 96,75   Masculino:207   1st Qu.:29,00   Servidor    :254
 Median :192,50                   Median :33,00   Terceirizado: 82
 Mean   :192,50                   Mean   :33,01
 3rd Qu.:288,25                   3rd Qu.:38,00
 Max.   :384,00                   Max.   :55,00
 estacionamento  instalacoes     ambiente
 Bom    :168    Bom    :142   Bom    :149
 Ótimo  : 20    Ótimo  :190   Ótimo  :175
 Regular:140    Regular: 39   Regular: 43
 Ruim   : 56    Ruim   : 13   Ruim   : 17
```

A função `View()` abre uma janela de visualização dos dados semelhante a uma planilha somente leitura. Para editar a base de dados a partir dessa planilha utilize a função `edit()` e lembre-se de salvar o resultado da edição num objeto no R através do operador `<-`.

```
> View(x = dados)
```

1.2.1 Acesso a variáveis da base

O objeto `dados` é uma representação da base lida a partir da fonte externa (arquivo de texto ou Excel) no R. Todos os objetos no R pertencem a alguma classe. No caso, o objeto `dados` é da classe `data.frame`.

```
> class(x = dados)

[1] "data.frame"
```

O `data.frame` é a estrutura básica de armazenamento de dados presente no R. Uma maneira de acessar as variáveis dentro do `data.frame` é por meio do operador `$`.

```
> dados$idade

  [1] 36 42 31 22 36 29 35 40 37 21 23 33 18 19 21 30 32 29 35 45 28 32 39
 [24] 38 30 27 21 37 43 18 33 38 40 37 22 32 35 21 20 40 31 36 34 36 34 37
 [47] 27 39 23 40 43 33 41 25 29 18 32 30 19 20 41 42 22 37 39 44 36 32 34
 [70] 39 30 32 33 30 50 33 28 45 36 32 23 39 44 32 31 32 38 39 19 34 28 29
 [93] 33 27 32 39 38 34 22 43 20 44 44 37 32 35 34 48 30 29 40 22 34 43 21
[116] 31 20 28 23 36 35 35 35 37 23 39 33 40 42 37 35 22 36 38 22 32 42 36
[139] 21 38 39 31 42 20 22 30 19 32 38 36 31 33 37 42 38 39 35 35 37 28 21
[162] 32 42 32 42 18 27 29 27 30 40 30 29 40 33 31 30 45 32 41 21 40 33 36
[185] 44 30 28 27 30 28 35 36 33 31 38 20 33 37 20 36 42 42 39 31 38 31 24
[208] 30 35 32 31 30 41 32 28 35 35 26 36 36 34 43 22 19 26 36 28 35 38 28
[231] 37 38 32 43 32 34 30 47 39 44 34 38 18 23 26 29 30 41 24 32 32 19 33
[254] 28 30 30 25 37 41 32 22 22 33 50 39 34 38 34 22 38 40 43 37 19 43 31
[277] 20 49 32 38 44 33 27 41 32 25 36 20 31 41 31 52 41 37 35 39 25 40 37
[300] 39 39 35 34 30 45 40 30 45 49 38 30 38 28 35 26 37 39 22 30 28 37 38
[323] 34 39 20 33 34 21 20 32 22 29 38 31 36 40 42 33 36 23 28 39 55 30 38
[346] 36 18 27 31 39 40 31 32 36 36 37 33 43 31 41 33 32 33 40 32 18 18 39
[369] 35 23 39 30 33 27 24 35 39 23 28 43 47 35 27 32
```

Outra forma de acessar os dados é através da notação matricial.

```
> dados[1, 4]

[1] Servidor
Levels: Estagiário Servidor Terceirizado
```

O primeiro valor se refere à posição na linha e o segundo na coluna. No exemplo, é acessado o valor na primeira linha da quarta coluna. Se um valor for omitido serão impressos na tela todos os dados da dimensão "em branco".

```
> dados[, "idade"]
```

No exemplo, a coluna desejada foi expressa pelo nome da variável[3]. A notação matricial é bastante útil em recursos intermediários e avançados do R. As funções `with()` e `attach()` também acessam[4] variáveis em um `data.frame`.

A base de dados possui algumas variáveis categóricas ordinais. Porém, quando a leitura dos dados é feita, o R ordena as categorias alfabeticamente. A função `factor()` corrige isso recodificando e sobrescrevendo a variável lida com a informação ordinal da categoria.

```
> dados$ambiente <- factor(x = dados$ambiente,
+                          levels = c("Ruim", "Regular", "Bom", "Ótimo"),
+                          ordered = TRUE)
```

1.2.2 Criação de variáveis

A função `transform()` permite que sejam criadas novas variáveis na base de dados carregada no R. Por exemplo, suponha que as variáveis categóricas `ambiente`, `estacionamento` e `instalacoes` sejam recodificadas numa escala numérica e uma nova variável, denominada índice de satisfação individual, seja a média aritmética dessas três variáveis.

```
> dados <- transform("_data" = dados,
+                    isi = (as.numeric(ambiente) + as.numeric(estacionamento) +
+                           as.numeric(instalacoes)) / 3)
```

A função `as.numeric()` recodifica uma variável categórica ordinal de acordo com a ordem dos seus fatores. No caso, "Ruim" passa a ser 1, "Regular" assume o valor 2, e "Bom" e "Ótimo" recebem os valores 3 e 4, respectivamente.

1.2.3 Medidas descritivas

O R possui diversas funções que calculam medidas descritivas como a média, mediana, variância, desvio padrão[5] e soma.

```
> mean(x = dados$idade)

[1] 33,01042

> median(x = dados$idade)
```

[3] Para ter acesso aos nomes das variáveis no `data.frame`, utilize a função `names()`.
[4] Para mais detalhes, acesse a documentação dessas funções através do operador `?`.
[5] O R calcula a variância e o desvio padrão amostrais.

```
[1] 33

> var(x = dados$idade)

[1] 52,61608

> sd(x = dados$idade)

[1] 7,253694

> sum(x = dados$idade)

[1] 12676

> length(x = dados$idade)

[1] 384
```

A função `fivenum()` retorna as cinco medidas resumo de Tukey. São elas: mínimo, corte de 25%, mediana, corte de 75% e máximo.

```
> fivenum(x = dados$idade)

[1] 18 29 33 38 55
```

A função `table()` mostra a contagem das observações da variável em forma de tabulação.

```
> table(vinculo = dados$vinculo)

vinculo
  Estagiário     Servidor Terceirizado
          48          254           82

> table(vinculo = dados$vinculo, ambiente = dados$ambiente)

              ambiente
vinculo        Ruim Regular Bom Ótimo
  Estagiário      2       2   4    40
  Servidor       14      40 137    63
  Terceirizado    1       1   8    72
```

É possível também calcular a tabulação das distribuições proporcionais das variáveis através da função `prop.table()`. Essa função possui o argumento opcional `margin` que permite calcular as distribuições proporcionais com relação às linhas ou às colunas.

```
> prop.table(table(vinculo = dados$vinculo, ambiente = dados$ambiente))

              ambiente
vinculo               Ruim     Regular         Bom       Ótimo
  Estagiário   0,005208333 0,005208333 0,010416667 0,104166667
  Servidor     0,036458333 0,104166667 0,356770833 0,164062500
  Terceirizado 0,002604167 0,002604167 0,020833333 0,187500000
```

A função `aggregate()` permite calcular medidas a partir de partições personalizadas da população. Por exemplo, através dessa função é possível calcular a média do recém criado índice de satisfação individual de acordo com o vínculo do respondente com a instituição.

```
> aggregate(x = list(isi = dados$isi),
+           by = list(vinculo = dados$vinculo), FUN = mean)

       vinculo      isi
1   Estagiário 3,111111
2     Servidor 2,925197
3 Terceirizado 3,130081
```

Ou ainda, numa segunda etapa é possível particionar o cálculo da média do índice através de mais de uma variável.

```
> aggregate(x = list(isi = dados$isi),
+           by = list(vinculo = dados$vinculo, sexo = dados$sexo),
+           FUN = mean)

       vinculo      sexo      isi
1   Estagiário  Feminino 2,968254
2     Servidor  Feminino 2,947222
3 Terceirizado  Feminino 3,129630
4   Estagiário Masculino 3,222222
5     Servidor Masculino 2,905473
6 Terceirizado Masculino 3,130435
```

1.2.4 Particionamento da base de dados

Em alguma análise, pode ser o interesse separar definitivamente a base de dados em partições específicas. Por exemplo, deseja-se tratar separadamente as pessoas que responderam que as condições de estacionamento na instituição são ruins. A função `subset()` é utilizada nessa situação.

```
> dados.est.ruim <- subset(x = dados, subset = estacionamento == "Ruim")
> summary(object = dados.est.ruim)

       id                sexo        idade               vinculo
 Min.   :  3,00   Feminino :33   Min.   :18,00   Estagiário  : 7
 1st Qu.: 79,25   Masculino:23   1st Qu.:28,75   Servidor    :36
 Median :177,00                  Median :33,00   Terceirizado:13
 Mean   :184,12                  Mean   :32,77
 3rd Qu.:281,75                  3rd Qu.:38,25
 Max.   :384,00                  Max.   :45,00
 estacionamento  instalacoes    ambiente       isi
 Ruim   :56     Ruim   : 1   Ruim   : 5   Min.   :1,667
 Regular: 0     Regular: 4   Regular: 5   1st Qu.:2,333
 Bom    : 0     Bom    :26   Bom    :19   Median :2,667
 Ótimo  : 0     Ótimo  :25   Ótimo  :27   Mean   :2,518
                                          3rd Qu.:2,667
                                          Max.   :3,000
```

O resultado da função `subset()` é um novo `data.frame` que pode ser utilizado exatamente como a base originalmente lida.

As proposições podem ser agrupadas através dos operadores lógicos "E" (&) e "OU" (|). O operador lógico de negação (!) também pode ser utilizado para particionar a base de dados em subgrupos mais específicos.

```
> subset(x = dados, subset = estacionamento != "Ótimo" & idade > 40)
```

Na chamada anterior, por exemplo, é extraído um particionamento da base com os indivíduos que têm a opinião sobre o estacionamento diferente de ótimo e com idade acima de 40 anos.

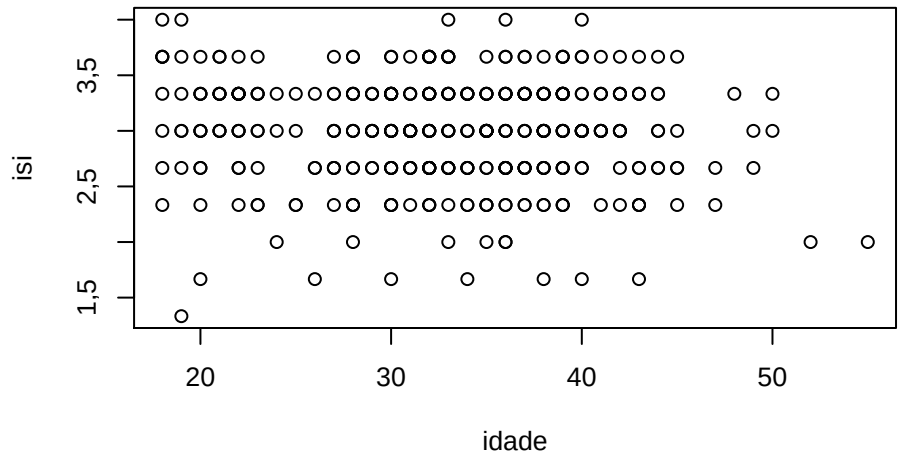
1.3 Gráficos

A visualização de dados através de gráficos é um dos pontos fortes do R. O programa possui pacotes que permitem criar gráficos altamente customizáveis e limitados apenas pela imaginação do estatístico. Nessa seção serão apresentados os principais gráficos que podem ser feitos através do pacote `graphics`, a biblioteca básica para criar gráficos no R.

1.3.1 Gráfico de dispersão

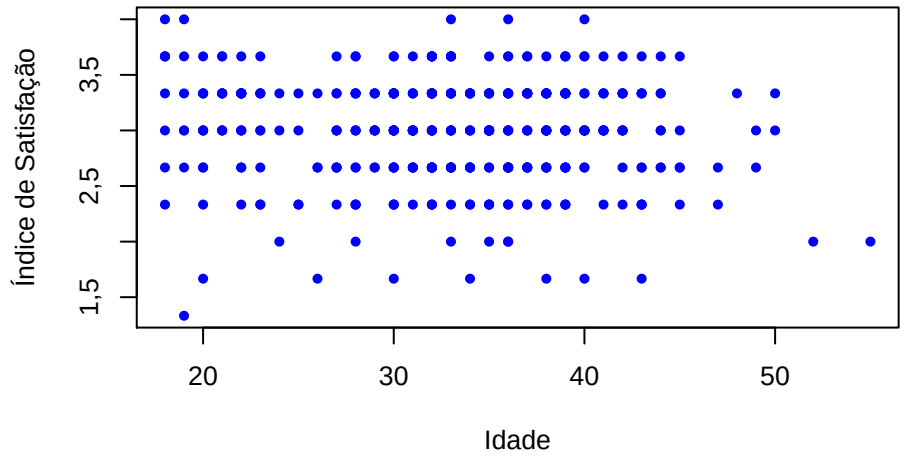
Os gráficos de dispersão são representações de variáveis num plano cartesiano. No R, um gráfico desse tipo é gerado com a função `plot()`.

```
> plot(formula = isi ~ idade, data = dados)
```



A função `plot()` possui um número enorme de argumentos que servem para personalizar o gráfico.

```
> plot(formula = isi ~ idade, data = dados, pch = 20, col = "blue",
+      xlab = "Idade", ylab = "Índice de Satisfação")
```



Os argumentos `pch` e `col` alteram a forma[6] e a cor[7] dos pontos. Os argumentos `xlab` e `ylab` renomeiam os eixos x e y, respectivamente. A Tabela 1.3 mostra alguns argumentos que customizam os gráficos do pacote `graphics`. Para a lista completa dos argumentos e a descrição de como cada um funciona, basta acessar a ajuda da função `par()`.

```
> ?par
```

1.3.2 Gráfico de barras

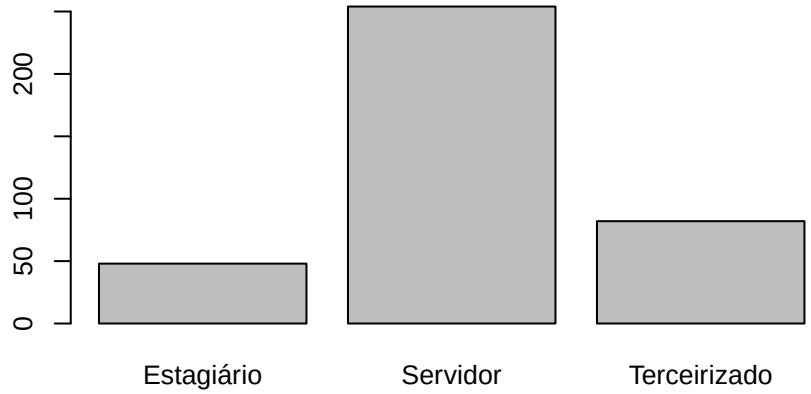
Os gráficos de barras (ou colunas) são muito comuns e utilizados para representar frequências ou estatísticas de variáveis categóricas. Eles podem ser produzidos no R a partir da função `barplot()`.

[6] As opções disponíveis podem ser acessadas na documentação da função `points()`.
[7] A lista das cores programadas em R pode ser acessada pela função `colors()`.

Tabela 1.3: Alguns argumentos de personalização do pacote `graphics`.

Argumento	Descrição
`bty`	Caractere que determina o tipo de caixa ao redor do gráfico.
`cex`	Valor que determina um aumento nos textos.
`col`	Cor dos pontos.
`lty`	Tipo da linha, quando houver.
`lwd`	Espessura da linha, quando houver.
`main`	Título do gráfico.
`mar`	Largura das margens.
`pch`	Formato dos pontos.
`xlab`	Rótulo do eixo x.
`ylab`	Rótulo do eixo y.

```
> barplot(height = table(dados$vinculo))
```

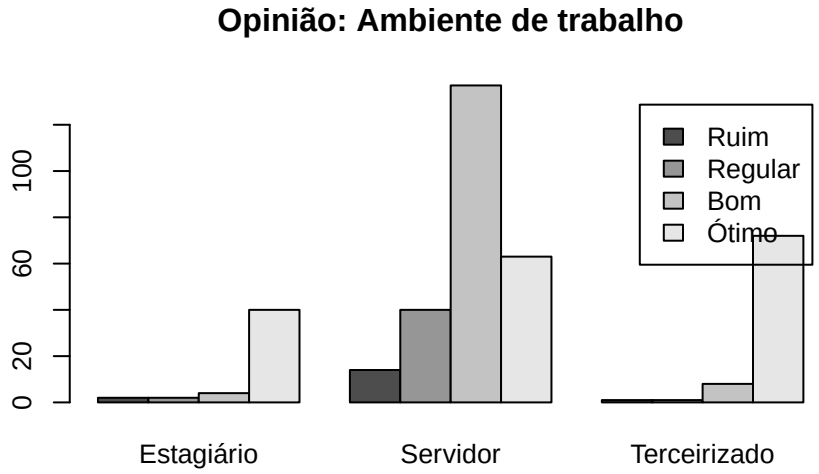


A função `barplot()` requer que os dados estejam tabulados. Portanto, a função `table()` pode ser usada diretamente no argumento `height` da função `barplot()` para tabular os dados e passar essa informação automaticamente no momento da criação do gráfico. Os gráficos de barra também podem ser personalizados. Alguns argumentos úteis da função `barplot()` estão na Tabela 1.4.

Tabela 1.4: Alguns argumentos da função `barplot()`.

Argumento	Descrição
`height`	Medidas das categorias.
`space`	Espaço ao lado esquerdo antes de cada barra.
`beside`	Indica se as barras devem ser lado a lado.
`horiz`	Indica se as barras devem estar na horizontal.
`legend`	Indica se a legenda deve ser incluída.
`col`	Cor de preenchimento.
`density`	Densidade de preenchimento.
`border`	Cor das bordas.

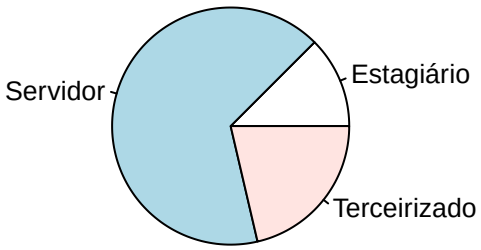
```
> barplot(height = table(dados$ambiente, dados$vinculo), legend = TRUE,
+         beside = TRUE, main = "Opinião: Ambiente de trabalho")
```



1.3.3 Gráfico de pizza

Os gráficos de pizza podem ser feitos no R através da função `pie()`.

```
> pie(x = table(dados$vinculo))
```

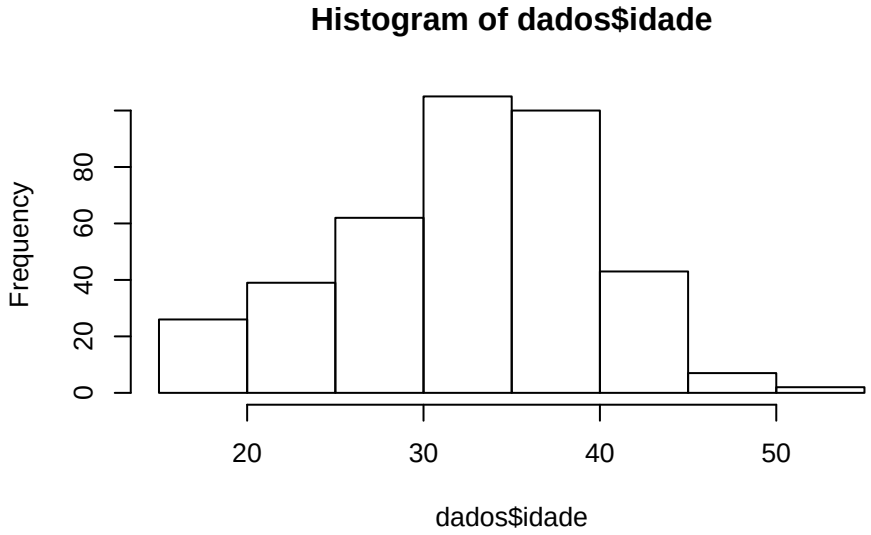


Não existem muitas opções para esse tipo de gráfico, além de mudança das cores e alterações nas especificações de rotação.

1.3.4 Histograma

Os histogramas mostram a densidade (ou distribuição de frequência) de uma variável númerica. Eles são criados no R pelo pacote `graphics` com a função `hist()`.

```
> hist(x = dados$idade)
```

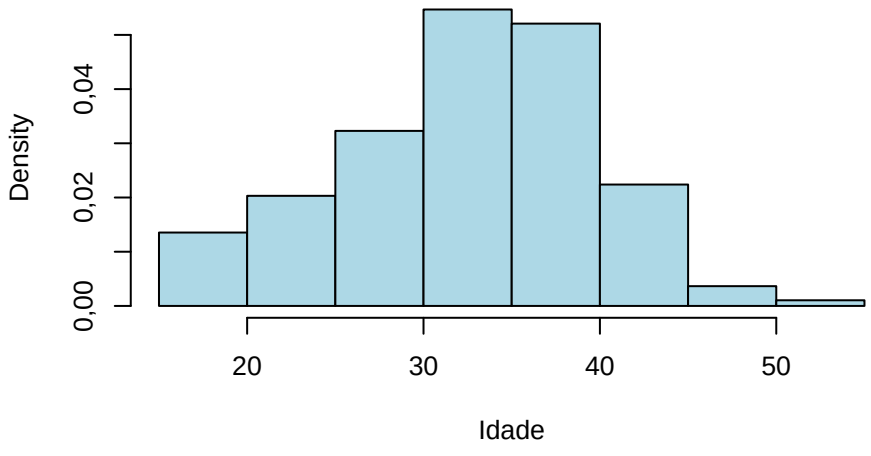


Por padrão, o histograma é construído a partir da distribuição de frequências da variável. A mudança dessa característica é feita pelo argumento `freq`. Esse e outros argumentos úteis podem ser vistos na Tabela 1.5.

Tabela 1.5: Alguns argumentos da função `hist()`.

Argumento	Descrição
`breaks`	Número de quebras.
`freq`	Indica se o histograma deve ser feito a partir da distribuição de frequências.
`col`	Cor de preenchimento.
`density`	Densidade de preenchimento.
`border`	Cor da borda.
`main`	Título do gráfico.
`xlab`	Rótulo do eixo x.
`ylab`	Rótulo do eixo y.

```
> hist(x = dados$idade, freq = FALSE, col = "lightblue",
+      main = "", xlab = "Idade")
```



1.4 Exercícios

1. Leia o arquivo `base-ex.txt` no R e salve num objeto chamado `dados.texto`.
2. Leia o arquivo `base-ex.csv` no R e salve num objeto chamado `dados.csv`. Dica: abra o arquivo antes no bloco de notas para saber qual o caractere de separação das variáveis.
3. Verifique se o número de observações nos dois objetos salvos nos exercícios anteriores é o mesmo. Dica: a função `nrow()` mostra a quantidade de linhas de um `data.frame`.
4. Gere o resumo de todas as variáveis do objeto `dados.texto`.
5. Calcule a média, mediana e desvio padrão do variável `idade` do objeto `dados.texto`.
6. Gere a tabulação conjunta (em valores proporcionais) das variáveis `vinculo` e `ambiente`.
7. Calcule (na escala de 0 a 1) o índice de satisfação individual, que é a média aritmética das variáveis `estacionamento`, `instalacoes` e `ambiente`.
8. Calcule a média do índice de satisfação segundo a variável `vinculo`.
9. Gere um gráfico de barras com a distribuição de frequências da variável `estacionamento`.
10. Gere um gráfico de barras com a distribuição de frequências da variável `estacionamento` segundo as categorias da variável `vinculo`.
11. Gere um histograma com a idade dos respondentes.
12. Gere um histograma com a idade dos estagiários.

Capítulo 2

Projeções

Um modelo estatístico muitas vezes é proposto para entender como um sistema se comporta, especialmente para determinar a relação entre as variáveis de interesse. Entretanto, a função mais importante num modelo de séries temporais tem que ser a sua capacidade de fazer predições com a menor incerteza possível.

Existem pelo menos três fatores que podem afetar a predição feita sobre um valor. O primeiro deles é o quanto se sabe sobre o sistema estudado. Evidentemente que quanto maior for o conhecimento sobre o objeto de estudo e quão eficiente for o estatístico na maneira de refletir esse conhecimento na proposição do modelo, mais precisas serão as previsões estimadas. Outro fator é a disponibilidade dos dados em si. E por fim, a própria estimativa pode afetar a realização dos dados através de uma política de prevenção ou superação de expectativas.

É possível fazer projeções sobre qualquer sistema, mesmo sem nenhum dado disponível. Todavia, serão abordados nesse texto métodos quantitativos de estimação, que partem do princípio que seja razoável supor que os padrões observados no passado, persistirão no futuro. Serão feitas projeções sobre a série de processos recebidos no Tribunal Superior do Trabalho. A biblioteca de funções `forecast` do R será utilizada para o cálculo das estimativas.

```
> library(forecast)
```

2.1 Metodologia

As primeiras etapas num estudo dessa natureza é definir bem o problema e coletar os dados disponíveis. No caso, serão feitas algumas projeções sobre a série de processos recebidos no TST de janeiro de 2004 a dezembro de 2015. Para efeitos de simplificação, serão desconsideradas quaisquer outras covariáveis que possam claramente afetar essa série de dados, como os processos recebidos e julgados em 1º e 2º graus. Também será

desconsiderada a conjuntura econômica do País no período. Os dados foram coletados a partir das séries disponibilizadas pela Seção de Acompanhamento Estatístico do TST.

Após essas etapas iniciais deve ser feita uma análise descritiva da série em busca de basicamente três características: tendência, sazonalidade e ciclos, para enfim seguir em frente na escolha e estimação do modelo. Um vez feito isso, o modelo deve ser utilizado para estimar as projeções desejadas. A última etapa é avaliar o modelo utilizado a partir da própria realização dos dados. Essa etapa será deixada de lado nesse contexto, porém trata-se de um passo muito importante que o estatístico deve ter sempre em mente.

2.2 Análise descritiva

Os dados podem ser lidos no R através da função `read.table()`.

```
> dados <- read.table(file = file.choose(), header = TRUE, sep = "\t")
```

Uma janela é aberta que deve ser utilizada para selecionar o arquivo `recebidos-tst.txt`. O arquivo é lido e os dados são salvos num objeto da classe `data.frame`.

```
> class(dados)

[1] "data.frame"

> head(dados)

   ano mes recebidos
1 2004   1      6232
2 2004   2     10681
3 2004   3     13495
4 2004   4     11085
5 2004   5     12991
6 2004   6     15099
```

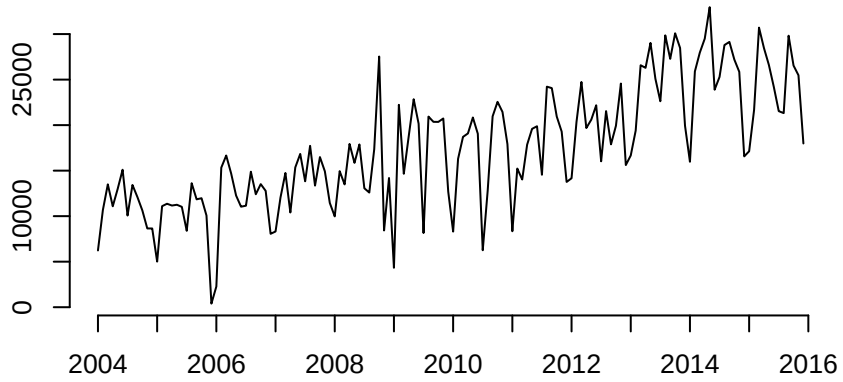
Existem na base três colunas: `ano`, `mes` e `recebidos`. Como os dados pertencem a uma série temporal, pode-se criar um objeto da classe `ts`[1] com a finalidade de facilitar o fluxo de trabalho.

```
> recebidos.mes <- ts(data = dados$recebidos, start = 2004, frequency = 12)
```

A função `ts()` transforma uma série de dados regulares numa série de dados temporais. O argumento `start` determina o ano de início e o argumento `frequency` define quantas observações existem por ano. O R agora entende que os dados pertencem a uma série de tempo.

[1] A classe `ts` (do inglês, time series) define objetos do tipo séries temporais.

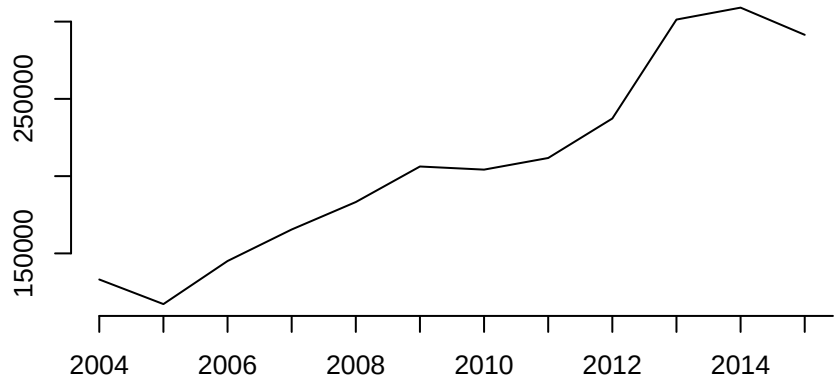
```
> plot(recebidos.mes, bty = "n", xaxp = c(2004, 2016, 12), xlab = "",
+      ylab = "", main = "Processos recebidos mensalmente no TST")
```



A série de processos recebidos por ano pode ser calculada através da aplicação da função `aggregate()` na série mensal. A função `ts()` transforma o resultado do passo anterior num objeto da classe de séries temporais.

```
> dados.ano <- aggregate(x = list(recebidos = dados$recebidos),
+                        by = list(ano = dados$ano), FUN = sum)
> recebidos.ano <- ts(dados.ano$recebidos, start = 2004, frequency = 1)
```

```
> plot(recebidos.ano, bty = "n", xaxp = c(2004, 2016, 12), xlab = "",
+      ylab = "", main = "Processos recebidos anualmente no TST")
```

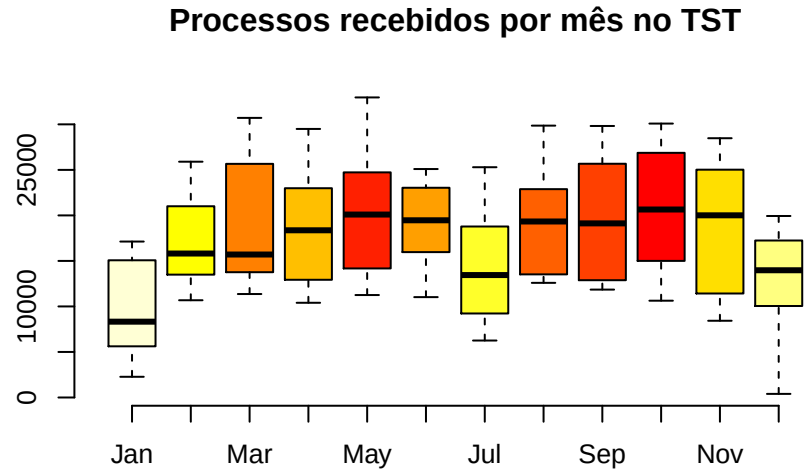


Os gráficos das duas séries permitem perceber uma tendência de crescimento aproximadamente linear na realização de dados. É fácil traçar mentalmente uma reta que "guia" a série de processos recebidos no TST.

Um padrão sazonal é outra característica que deve ser identificada, ou não, no conjunto de dado. Um comportamento sazonal é facilmente confundido com um ciclo. Enquanto o primeiro é explicitamente governado por um padrão temporal periódico, o segundo ocorre sem obedecer uma passagem de tempo bem definida.

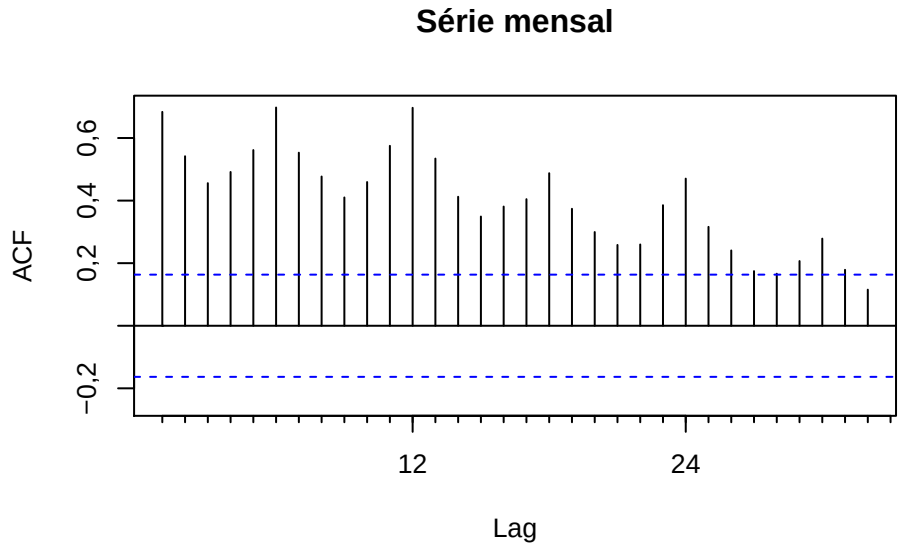
A série de dados anuais não aparenta ter um comportamento sazonal. Por sua vez, o gráfico da série mensal sugere que possa existir um padrão periódico bem definido. O diagrama de caixas (boxplot) mostra uma redução na quantidade de processos recebidos nos meses de janeiro, julho e dezembro, indicando assim uma possível sazonalidade.

```
> ordem <- order(order(aggregate(list(recebidos = dados$recebidos),
+             list(mes = dados$mes), mean)$recebidos, decreasing = TRUE))
> cores <- heat.colors(12)[ordem]
> boxplot(recebidos ~ mes, data = dados, axes = FALSE, col = cores,
+         main = "Processos recebidos por mês no TST")
> axis(1, at = 1:12, labels = month.abb)
> axis(2)
```



Uma série temporal é autocorrelacionada quando as observações apresentam uma correlação linear com valores separados por um número fixo de observações anteriores. A função `Acf()` calcula a função de autocorrelação e estima intervalos que determinam se o valor calculado é estatisticamente significativo ou não.

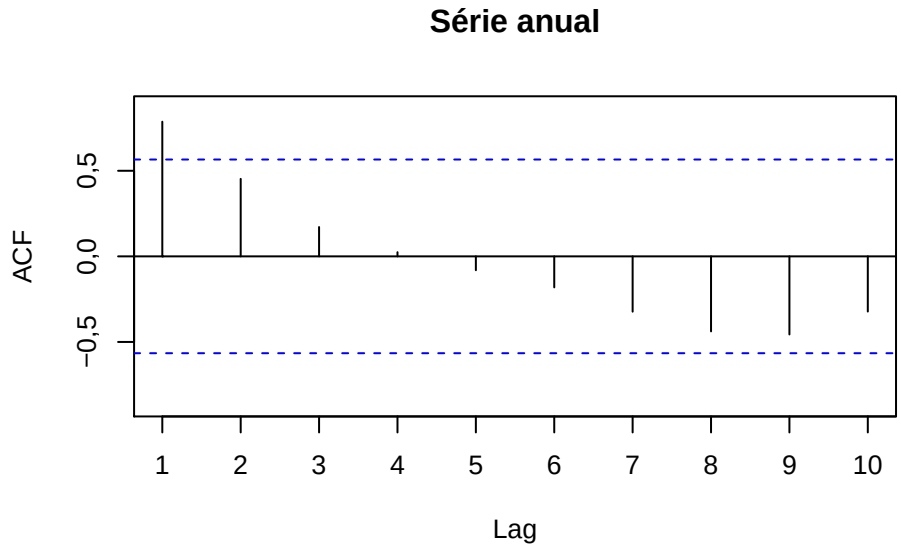
```
> Acf(recebidos.mes, lag.max = 32, main = "Série mensal")
```



Como pode ser observado, existe uma autocorrelação bastante forte na série mensal. O que era esperado, uma vez que a série aparenta possuir um efeito sazonal.

As observações da série anual, por sua vez, apresentam uma autocorrelação estatisticamente significativa apenas com a observação imediatamente anterior. Uma série temporal sem autocorrelação é chamada de ruído branco.

```
> Acf(recebidos.ano, main = "Série anual")
```



Com base nessas análises, alguns modelos de predição podem ser propostos.

2.3 Modelo linear

O modelo linear define o comportamento de uma variável resposta de acordo com uma combinação linear de variáveis explicativas.

2.3.1 Modelo linear simples

O modelo linear simples leva em consideração apenas uma variável explicativa. Desse modo, é possível aplica-lo apenas aos dados anuais (uma vez que os dados mensais apresentam uma possível sazonalidade relacionada aos meses). O modelo linear foi implementado na biblioteca `forecast` a partir da função `tslm()`.

```
> modelo1 <- tslm(recebidos.ano ~ trend)
```

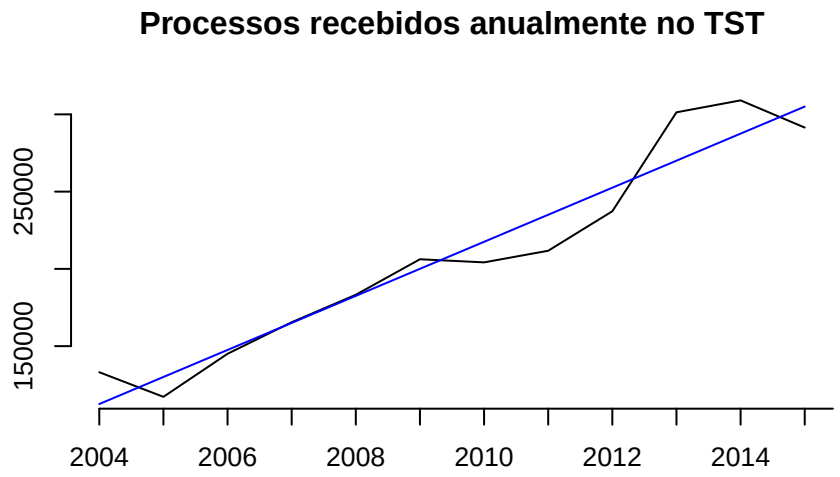
O argumento `trend` na chamada anterior indica que deve ser estimado um modelo linear a partir de uma série temporal que apenas apresenta uma tendência constante de crescimento (ou decréscimo).

As estimativas dos coeficientes foram estatisticamente significativas com 95% de confiança, como pode ser observado no resumo do `modelo1`.

```
> summary(modelo1)


Call:
tslm(formula = recebidos.ano ~ trend)

Residuals:
   Min     1Q Median     3Q    Max
-23292 -13379  -1021   9799  31297

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)    95034      11030   8,616 6,11e-06 ***
trend          17499       1499  11,676 3,78e-07 ***
---
Signif. codes:  0 '***' 0,001 '**' 0,01 '*' 0,05 '.' 0,1 ' ' 1

Residual standard error: 17920 on 10 degrees of freedom
Multiple R-squared:  0,9317,Adjusted R-squared:  0,9248
F-statistic: 136,3 on 1 and 10 DF,  p-value: 3,776e-07
```

Os coeficientes do modelo linear são de fácil interpretação. No caso, segundo o modelo, é esperado um aumento anual médio de 17.499 processos recebidos ano a ano. O intercepto indica a quantidade de processos recebidos no ano zero, o que nesse contexto não faz sentido. Portanto é apenas uma entidade matemática sem interpretação com sentido prático.

```
> plot(recebidos.ano, bty = "n", xaxp = c(2004, 2016, 12), xlab = "",
+      ylab = "", main = "Processos recebidos anualmente no TST")
> lines(fitted(modelo1), col = "blue")
```



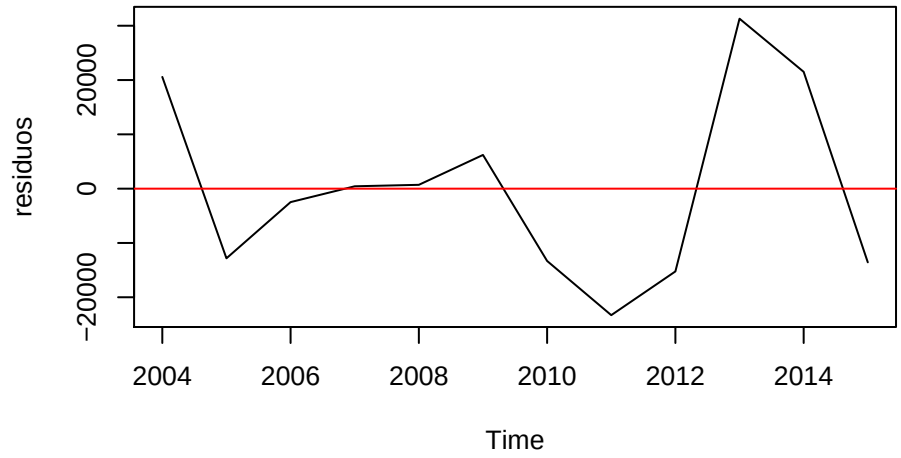
Para avaliar a adequação do modelo é necessário fazer uma análise dos resíduos[2]. No R os resíduos de um modelo podem ser calculados pela função `residuals()`.

[2]Resíduo é a diferença entre o valor observado e o valor ajustado pelo modelo.

```
> residuos <- residuals(modelo1)
```

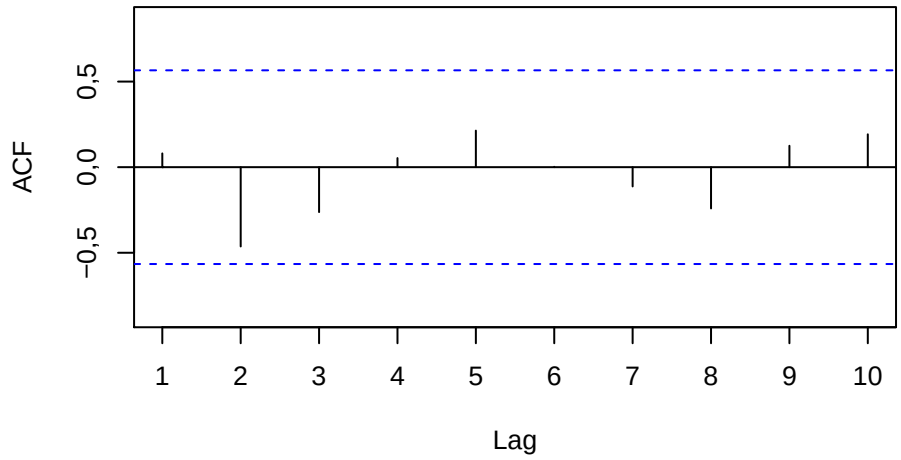
Bons ajustes irão gerar resíduos não correlacionados e não viesados. Para efeitos de predição, é desejável que os resíduos tenham distribuição normal.

```
> plot(residuos)
> abline(h = 0, col = "red")
```



A série de resíduos se comporta aparentemente sem viés, conforme o desejado. O gráfico da função de autocorrelação mostra também que os resíduos não possuem alguma autocorrelação estatisticamente significativa.

```
> Acf(residuos, main = "")
```

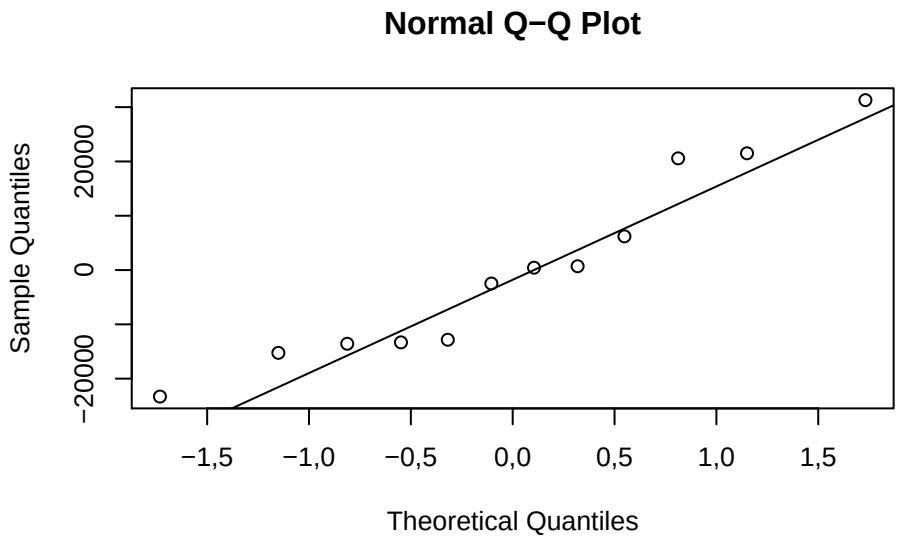


Se os resíduos forem autocorrelacionados, significa que existem ainda variáveis influentes que foram deixadas de fora na modelagem[3].

Por fim, quanto a normalidade, o gráfico de quantis da distribuição normal sugere que os resíduos podem ser normalmente distribuídos.

[3] Experimente modelar os dados mensais com um modelo linear apenas com tendência.

```
> qqnorm(residuos)
> qqline(residuos)
```



O teste de hipóteses de Shapiro-Wilk corrobora com a sugestão do gráfico de quantis. Esse teste parte da hipótese nula de normalidade e o seu resultado mostra que não há evidências estatísticas para se rejeitar essa hipótese sob o nível de 95% de confiança[4].

```
> shapiro.test(residuos)


Shapiro-Wilk normality test

data:  residuos
W = 0,92672, p-value = 0,3466
```

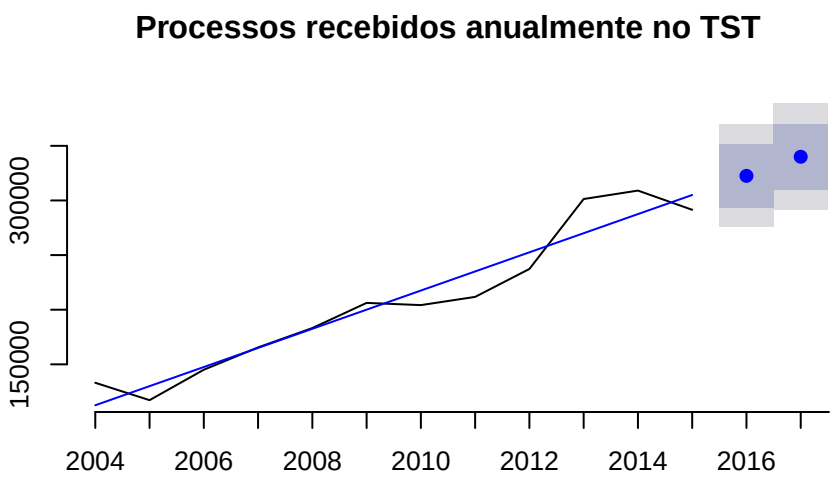
Se os resíduos forem independentes e normalmente distribuídos será possível construir intervalos de predições confiáveis. A função `forecast()` calcula as projeções desejadas. O argumento `h` indica quantas projeções devem ser estimadas.

```
> projecoes <- forecast(modelo1, h = 2)
> projecoes

     Point Forecast    Lo 80    Hi 80    Lo 95    Hi 95
2016       322521,2 293645,7 351396,8 275633,4 369409,1
2017       340020,2 310137,1 369903,4 291496,3 388544,2
```

Essas são as projeções para os dois próximos anos de processos recebidos mensalmente no TST, sob o modelo linear simples. São também apresentados os intervalos de 80% e 95% de predição.

[4]Há de se destacar que a massa de dados anuais talvez seja muito pequena para conduzir esse teste de hipóteses. No caso, deveria-se coletar os dados de processos recebidos em anos anteriores a 2004.

```
> plot(projecoes, bty = "n", xaxp = c(2004, 2018, 14), xlab = "",
+      ylab = "", main = "Processos recebidos anualmente no TST")
> lines(fitted(modelo1), col = "blue")
```



2.3.2 Modelo linear múltiplo

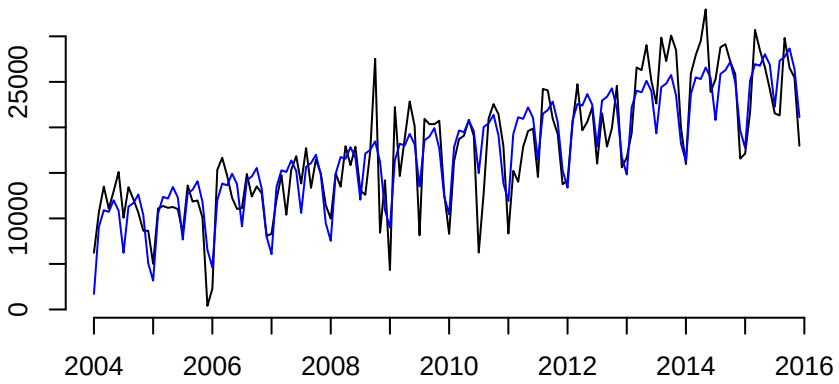
Para modelar os dados mensais é preciso adicionar o efeito sazonal no modelo.

```
> modelo2 <- tslm(recebidos.mes ~ trend + season)
```

O novo argumento `season` indica que a série apresenta sazonalidade, porém observe que nada foi informado sobre a frequência do efeito como mensal, trimestral ou semestral, por exemplo. O algoritmo de estimação do modelo implementado no R assume a frequência associada ao objeto criado pela função `ts()`.

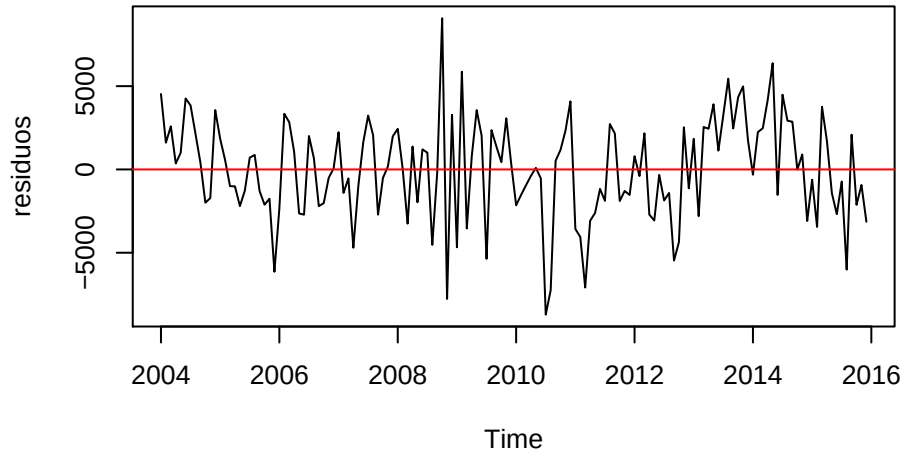
O modelo foi ajustado e os detalhes podem ser observados através da função `summary()`. O gráfico a seguir mostra o ajuste encontrado.

```
> plot(recebidos.mes, bty = "n", xaxp = c(2004, 2016, 12), xlab = "",
+      ylab = "", main = "Processos recebidos mensalmente no TST")
> lines(fitted(modelo2), col = "blue")
```



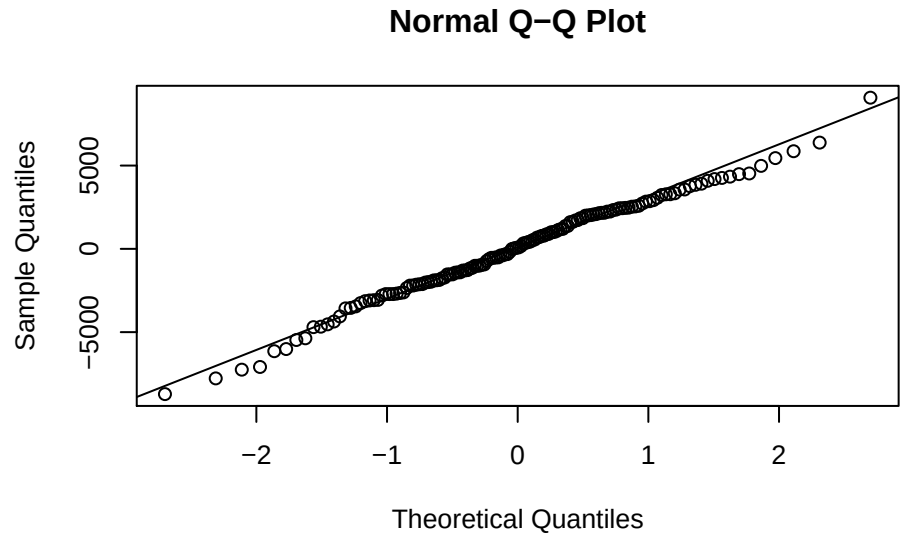
Como no caso anterior, a qualidade do ajuste deve ser avaliada através da análise dos resíduos.

```
> residuos <- residuals(modelo2)
> plot(residuos)
> abline(h = 0, col = "red")
```



No geral, os resíduos parecem distribuídos de forma aceitável ao redor da média. Exceto nos valores próximos de 2013 e 2014, quando parece existir uma elevação no nível da série de resíduos. O gráfico de quantis da distribuição normal sugere que a hipótese de normalidade não deva ser rejeita. Fato que é confirmado com o teste não paramétrico de Shapiro-Wilk.

```
> qqnorm(residuos)
> qqline(residuos)
```




```
> shapiro.test(residuos)


Shapiro-Wilk normality test
```

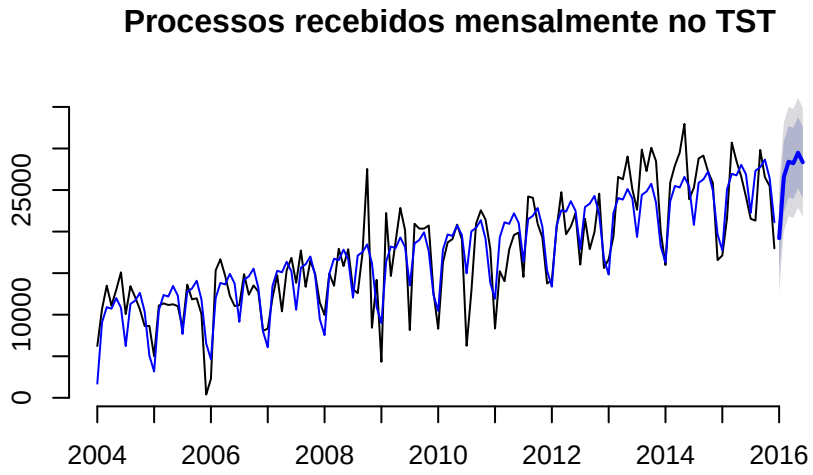
```
data:  residuos
W = 0,99093, p-value = 0,4825
```

Dessa maneira as projeções podem ser feitas com os intervalos de predição consistentes, apesar de pouco precisos.

```
> projecoes <- forecast(modelo2, h = 6)
> projecoes

         Point Forecast    Lo 80    Hi 80    Lo 95    Hi 95
Jan 2016       19204,45 14924,01 23484,89 12630,37 25778,54
Feb 2016       26574,37 22293,93 30854,81 20000,28 33148,45
Mar 2016       28406,78 24126,34 32687,23 21832,70 34980,87
Apr 2016       28224,20 23943,76 32504,64 21650,12 34798,29
May 2016       29494,28 25213,84 33774,73 22920,20 36068,37
Jun 2016       28338,78 24058,34 32619,23 21764,70 34912,87
```

Essas são as projeções de processos recebidos mensalmente para o próximo semestre no TST, sob o modelo linear múltiplo. O gráfico a seguir ilustra essas estimativas.

```
> plot(projecoes, bty = "n", xaxp = c(2004, 2016, 12), xlab = "",
+      ylab = "", main = "Processos recebidos mensalmente no TST")
> lines(fitted(modelo2), col = "blue")
```



2.4 Modelos de alisamento exponencial

Um modelo alternativo ao linear é o modelo de alisamento exponencial. Esse modelo foi proposto inicialmente em 1950. Ele produz projeções baseadas em médias ponderadas das observações passadas. Os pesos dessas observações vão ficando cada vez menores conforme os valores vão ficando mais no passado.

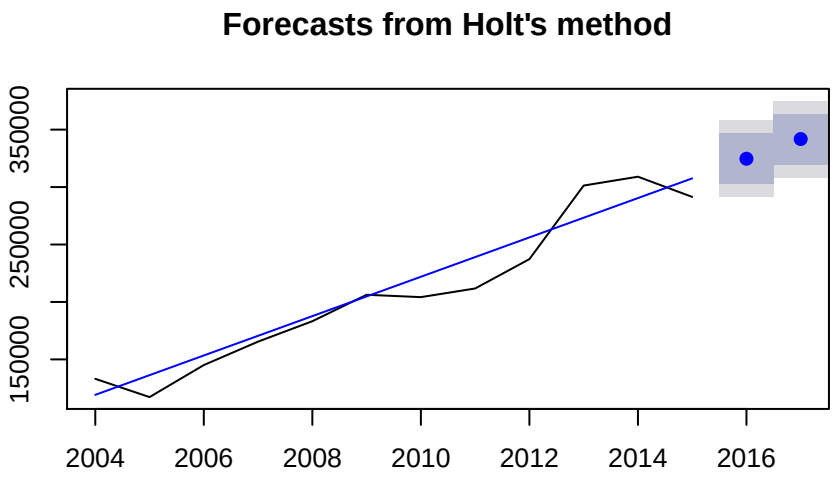
2.4.1 Modelo de Holt

O modelo de Holt leva em consideração uma série temporal apenas com tendência. No R ele é estimado pela função `holt()`.

```
> modelo3 <- holt(recebidos.ano, h = 2)
> modelo3

     Point Forecast    Lo 80    Hi 80    Lo 95    Hi 95
2016       324691,2 302853,9 346528,6 291293,9 358088,6
2017       341818,9 319981,6 363656,3 308421,6 375216,3
```

Foi utilizada a série de processos recebidos anualmente, pois, como visto anteriormente, a série mensal possui um efeito sazonal além da tendência. Observe que a função `holt()` estima o modelo e produz imediatamente as estimativas. O gráfico mostra a série de dados com o respectivo resultado produzido pelo modelo.

```
> plot(modelo3)
> lines(fitted(modelo3), col = "blue")
```



O modelo não requer pressuposto inicial e as estimativas pontuais serão boas sob qualquer distribuição de resíduos, porém a hipótese de normalidade dos resíduoes é necessária para o cálculo dos intervalos de predição. Todavia, isso é facilmente contornado no R utilizando os intervalos calculados via bootstrap. Para fazer isso, basta acrescentar os parâmetros `bootstrap` e `simulate` com o valor verdadeiro na hora de chamar a função.

```
> modelo4 <- holt(recebidos.ano, h = 2, bootstrap = TRUE, simulate = TRUE)
> modelo4

     Point Forecast    Lo 80    Hi 80    Lo 95    Hi 95
2016       324691,2 310198,9 347870,4 301948,2 357288,7
2017       341818,9 327323,7 365002,7 319073,3 374417,5
```

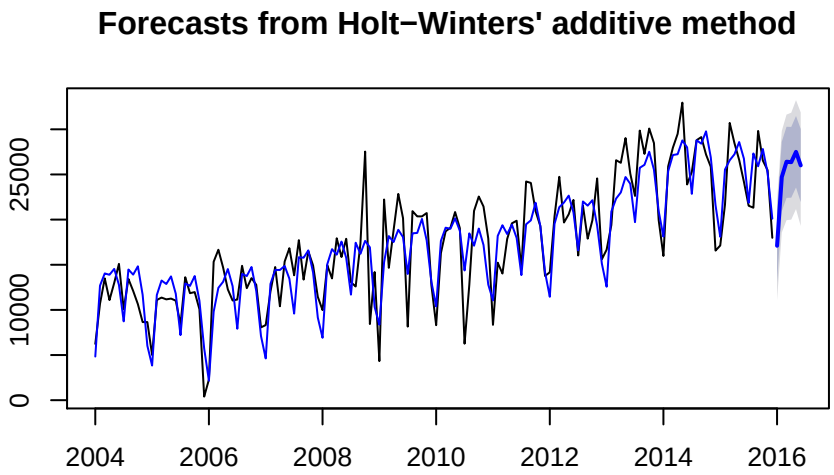
2.4.2 Modelo de Holt-Winters

O modelo de Holt-Winters adiciona a sazonalidade no cálculo das projeções. Ele é estimado no R através da função `hw()`.

```
> modelo5 <- hw(recebidos.mes, h = 6, bootstrap = TRUE, simulate = TRUE)
> modelo5

         Point Forecast    Lo 80    Hi 80    Lo 95    Hi 95
Jan 2016       17098,18 13023,25 21029,97 11074,42 22017,18
Feb 2016       24714,73 20920,16 28611,58 18561,23 29809,07
Mar 2016       26414,97 22402,49 30284,81 19939,24 31631,13
Apr 2016       26360,77 22429,53 30264,28 19997,14 31838,72
May 2016       27502,98 23558,86 31472,68 21167,27 33229,90
Jun 2016       26006,38 21936,70 30037,10 19298,57 31872,92
```

Por apresentar tendência e efeito sazonal como observado em análises anteriores, dessa vez, foi utilizada a série de processos recebidos por mês. Os intervalos de predição foram calculados por métodos computacionais e o gráfico a seguir ilustra os resultados.

```
> plot(modelo5)
> lines(fitted(modelo5), col = "blue")
```



O modelo de Holt-Winters possui uma variação em relação ao método de estimar as projeções. Por padrão, é feito um cálculo com efeitos sazonais aditivos. Esse método é preferível quando a variabilidade da série é constante no tempo. Se a variabilidade dos dados aumentar conforme o nível da série aumenta, deve-se utilizar um efeito multiplicativo no cálculo das estimativas. Isso é feito a partir do parâmetro `seasonal`.

Apesar de serem uma boa alternativa, sobretudo por não necessitarem de pressuposto inicial sobre a distribuição dos resíduos para o cálculo de intervalos de predição, os modelos de alisamento exponencial têm um problema com estimativas a longo prazo. Como o cálculo das projeções é baseado em médias ponderadas das observações anteriores, a longo prazo, as estimativas serão computadas sob projeções iniciais.

2.5 Modelos ARIMA

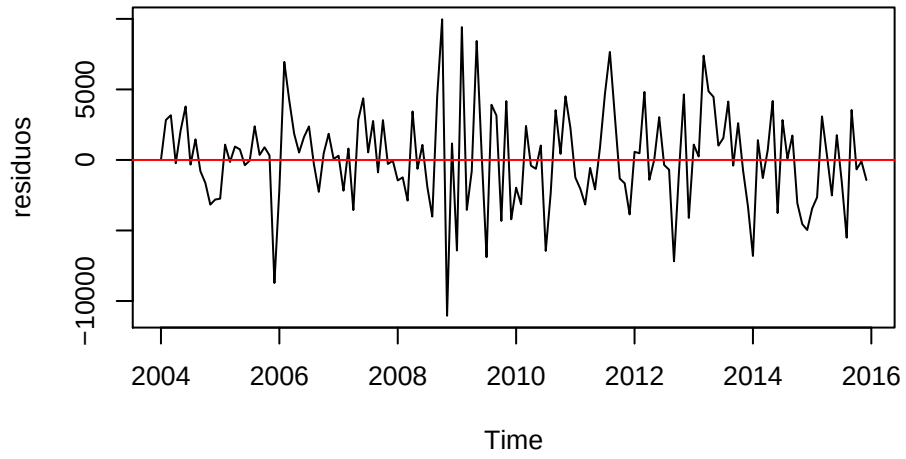
Existe uma classe de modelos de séries temporais baseada em relações de autocorrelações: os modelos autorregressivos, os modelos de médias moveis e a junção dos dois. Esse modelos funcionam apenas com séries estacionárias, isto é, sem tendência e sazonalidade. A fim de corrigir esse problema, surgem os modelos ARIMA. Eles, inicialmente, transformam a série de dados numa série estacionária e então aplicam as técnicas associadas a essas séries.

Os modelos ARIMA possuem um conjunto grande de parâmetros que devem ser determinados um a um através de métodos indiretos como gráficos de diagnósticos, um processo extremamente trabalhoso. A biblioteca de funções `forecast` ganhou notoriedade, pois o autor inovou com um algoritmo capaz de achar os parâmetros do modelo de forma bastante eficiente. A função `auto.arima()` faz todo o trabalho pesado.

```
> modelo6 <- auto.arima(recebidos.mes, seasonal = TRUE)
```

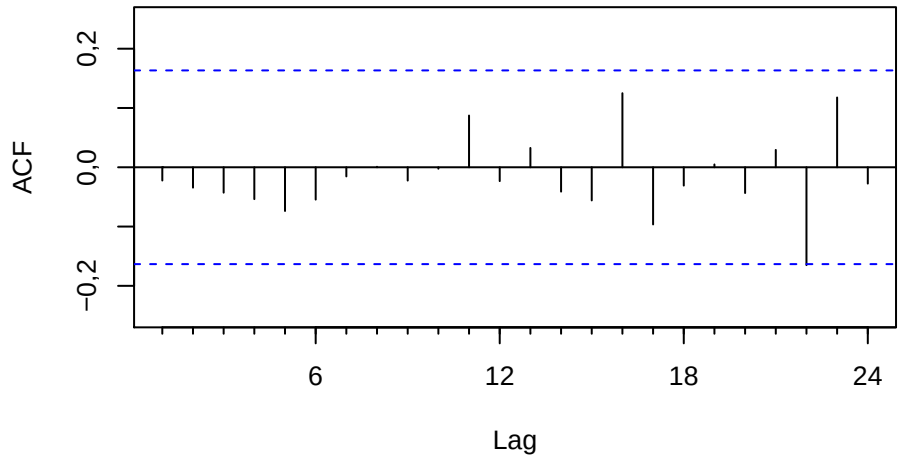
A função seleciona o melhor modelo, porém os resíduos devem ser avaliados para saber se a escolha foi conveniente.

```
> residuos <- residuals(modelo6)
> plot(residuos)
> abline(h = 0, col = "red")
```



Os resíduos devem ser uma série de ruído branco.

```
> Acf(residuos, main = "")
```



O gráfico anterior mostra que não há autocorrelação estatisticamente significativa nos resíduos. O teste de Ljung-Box testa a hipótese de ruído branco.

```
> Box.test(residuos, type = "Ljung-Box")


Box-Ljung test

data:  residuos
X-squared = 0,073655, df = 1, p-value = 0,7861
```

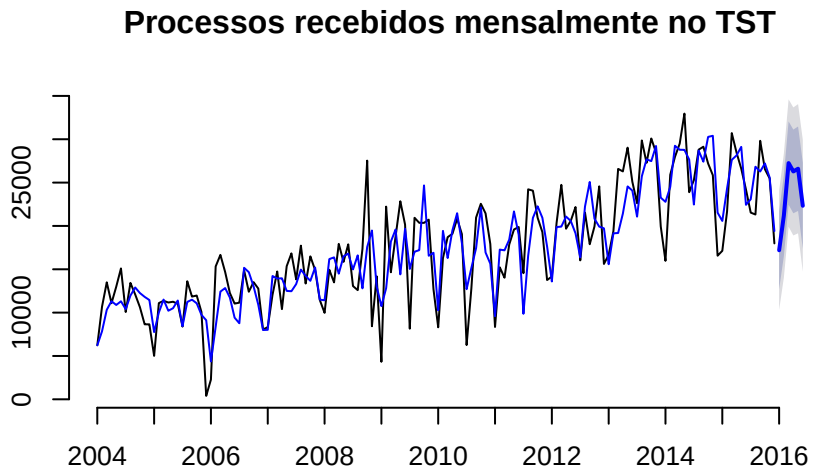
Como pode ser observado, a hipótese nula de ruído branco não pode ser rejeitada. Dessa maneira, é possível calcular as projeções com base nesse modelo[5].

```
> projecoes <- forecast(modelo6, h = 6)
> projecoes

         Point Forecast    Lo 80    Hi 80    Lo 95    Hi 95
Jan 2016       17192,35 12697,04 21687,66 10317,37 24067,33
Feb 2016       21184,61 16510,18 25859,03 14035,69 28333,52
Mar 2016       27251,97 22451,12 32052,82 19909,70 34594,23
Apr 2016       26286,10 21458,68 31113,52 18903,20 33669,01
May 2016       26599,84 21729,87 31469,80 19151,87 34047,80
Jun 2016       22322,73 17352,01 27293,46 14720,66 29924,80
```

Essas são as projeções de processos recebidos nos próximos seis meses no TST. O gráfico mostra as estimativas junto com a série de observações.

```
> plot(projecoes, bty = "n", xaxp = c(2004, 2016, 12), xlab = "",
+      ylab = "", main = "Processos recebidos mensalmente no TST")
> lines(fitted(modelo6), col = "blue")
```

[5] Se a hipótese de ruído branco fosse rejeitada, o modelo deveria ser descartado.



2.6 Exercícios

1. Leia o arquivo de dados `recebidos-trt.txt` no e R e salve no objeto `dados`.
2. Escolha um TRT do seu interesse e particione a base de dados num novo `data.frame` chamado `recebidos`. Dica: utilize a função `subset()` .
3. Utilize a função `ts()` para transformar a base `recebidos` num objeto da classe `ts` com o nome `recebidos.mes`.
4. Utilize a função `aggregate()` para gerar a base de processos recebidos por ano no TRT e salve num objeto da classe `ts` chamado de `recebidos.ano`.
5. Explore as bases de dados (`recebidos.ano` e `recebidos.mes`)e tente achar padrões como: tendência, sazonalidade e ciclos.
6. Utilize um modelo de regressão linear (se possível) para projetar os três próximos anos na série anual e os 6 próximos meses na série mensal. Faça uma análise da consistência dos intervalos de predição a partir dos resíduos.
7. Utilize um modelo de alisamento exponencial (se possível) para projetar os três próximos anos na série anual e os 6 próximos meses na série mensal. Faça uma análise da consistência dos intervalos de predição a partir dos resíduos.
8. Utilize um modelo ARIMA (se possível) para projetar os três próximos anos na série anual e os 6 próximos meses na série mensal. Faça uma análise da consistência das intervalos de predição a partir dos resíduos.
9. Compare as estimativas dos diferentes modelos e diga qual é o modelo mais adequado.

Respostas

Capítulo 1

1. ```
   dados.texto <- read.table(file = file.choose(), header = TRUE,
                             sep = "\t")
   ```

2. ```
   dados.csv <- read.table(file = file.choose(), header = FALSE,
                           sep = ";")
   ```

3. ```
   nrow(dados.texto)
   nrow(dados.csv)
   ```

4. ```
   summary(dados.texto)
   ```

5. ```
   mean(dados.texto$idade)
   median(dados.texto$idade)
   sd(dados.texto$idade)
   ```

6. ```
   prop.table(table(dados.texto$vinculo, dados.texto$ambiente))
   ```

7. ```
   # ordenação das variáveis categoricas
   dados.texto$estacionamento <- factor(dados.texto$estacionamento,
          levels = c("Ruim", "Regular", "Bom", "Ótimo"), ordered = TRUE)

   dados.texto$instalacoes <- factor(dados.texto$instalacoes,
          levels = c("Ruim", "Regular", "Bom", "Ótimo"), ordered = TRUE)

   dados.texto$ambiente <- factor(dados.texto$ambiente,
   ```
   ```

```
      levels = c("Ruim", "Regular", "Bom", "Ótimo"), ordered = TRUE)

# criação da variável "is" (escala 1 a 4)
dados.texto <- transform(dados.texto, is =
                           (as.numeric(dados.texto$estacionamento) +
                            as.numeric(dados.texto$instalacoes) +
                            as.numeric(dados.texto$ambiente)) / 3)

# transformação da escala de "is" para (0, 1)
dados.texto <- transform(dados.texto, is = (is - 1) / 3)
```

8. ```
   aggregate(x = list(is = dados.texto$is),
             by = list(vinculo = dados.texto$vinculo), FUN = mean)
   ```

9. ```
   barplot(table(dados.texto$estacionamento))
   ```

10. ```
    barplot(table(dados.texto$vinculo, dados.texto$estacionamento),
                  beside = TRUE, legend = TRUE)
    ```

11. ```
    hist(dados.texto$idade)
    ```

12. ```
    est <- subset(dados.texto, vinculo == "Estagiário")
    hist(est$idade, breaks = 4, col = "lightblue")
    ```
    ```

Capítulo 2

1. ```
   dados <- read.table(file = file.choose(), header = TRUE, sep = "\t")
   ```

2. ```
   recebidos <- subset(dados, trt == 10)
   ```

3. ```
   recebidos.mes <- ts(recebidos$recebidos, start = 2004, frequency = 12)
   ```

4. ```
   recebidos.ano <- aggregate(list(recebidos = recebidos$recebidos),
                              list(ano = recebidos$ano), sum)
   recebidos.ano <- ts(recebidos$recebidos, start = 2004, frequency = 1)
   ```

5. Tal qual texto.

6. Tal qual texto.

7. Tal qual texto.

8. Tal qual texto.

9. Tal qual texto.

Referências Bibliográficas

[1] Michael J. Crawley. The R Book. Wiley, Chichester, 2nd edition, 2012.

[2] Rob J. Hyndman and George Athanasopoulos. Forecasting: Principles and practice. Livro online disponível gratuitamente, 2016.

[3] Norman Matloff. The Art of R Programming. No Starch Press, San Francisco, 1st edition, 2011.