OUSHIRO, Livia. "TRATAMENTO DE DADOS COM O R PARA ANÁLISES SOCIOLINGUÍSTICAS", p.134-177. In Raquel Meister Ko. Freitag (Organizadora). **Metodologia de Coleta e Manipulação de Dados em Sociolinguística**, São Paulo: Editora Edgard Blücher, 2014. http://dx.doi.org/10.5151/BlucherOA-MCMDS-10cap

10 CAPÍTULO

TRATAMENTO DE DADOS COM O R PARA ANÁLISES SOCIOLINGUÍSTICAS

Livia Oushiro

INTRODUÇÃO

Ao se debruçar sobre a língua em uso, o linguista inevitavelmente se depara com a variação linguística. No português falado no Brasil, encontram-se, por exemplo, diversas realizações para o "r", tanto em posição de ataque quanto em coda silábica (como tepe, vibrante múltipla, retroflexo, fricativa velar, etc.); o emprego dos pronomes "tu", "você", "ocê", "cê", para se referir ao interlocutor; o uso de sintagmas verbais plurais ora com marcação de número apenas no sujeito (por exemplo, "eles foi"), ora com marcação redundante no sujeito e no verbo (por exemplo, "eles foram").

Um dos principais fundamentos dos estudos sociolinguísticos é a premissa de que a variação linguística – verificada em todas as línguas, em todas as comunidades e, em última instância, na fala de um mesmo indivíduo – faz parte do sistema linguístico e da competência comunicativa dos falantes. A variação linguística não só é inerente, como também é ordenada (WEINREICH; LABOV; HERZOG, 1968): as flutuações observadas formam padrões que podem ser descritos e analisados pelo estudioso da língua em uso.

A observação desses padrões, no entanto, requer a análise de uma grande quantidade de dados. A partir da observação de poucas ocorrências, de um ou poucos falantes, dificilmente se poderia chegar a conclusões confiáveis sobre quais falantes tendem a empregar uma ou outra forma, em quais contextos (linguísticos

ou sociais) elas tendem a ocorrer e por que a variação ocorre do modo como se observa. A sociolinguística variacionista se assenta sobre o Paradigma Quantitativo (BAYLEY, 2002; GUY, 1993), que busca modelar a competência comunicativa dos falantes através da análise de formas linguísticas variáveis em seus contextos de uso, a fim de derivar afirmações acerca da probabilidade de coocorrência de uma forma linguística variável e as características contextuais.

Desse modo, o sociolinguista variacionista lida com uma grande quantidade de dados. Entre as suas diversas tarefas, incluem-se: (i) a coleta de dados (em geral, na forma de gravações de entrevistas sociolinguísticas com falantes de uma comunidade); (ii) a transcrição dessas gravações; (iii) a definição de uma variável sociolinguística e de seus contextos linguísticos possíveis (o contexto variável); (iv) a identificação de ocorrências no *corpus* de entrevistas; (v) o levantamento de hipóteses sobre fatores, de natureza social e linguística, que estejam correlacionados ao uso da variável; (vi) a codificação das ocorrências de acordo com as hipóteses levantadas; (vii) a análise quantitativa dos dados no GoldVarb X ou RBrul e (viii) a interpretação de resultados obtidos.

Algumas dessas tarefas exigem conhecimento especializado e criatividade para ser bem executadas – por exemplo, a coleta de boas gravações, o levantamento de hipóteses, a interpretação de resultados. Algumas outras podem ser bastante repetitivas, mecânicas e previsíveis – como a identificação de ocorrências no *corpus* (quando já se definiu a variável e seu envelope de variação) e a sua extração para codificação. Para esse segundo conjunto de tarefas, em princípio, não é necessário um conhecimento especializado; por exemplo, não é necessário saber o que é um fonema para copiar e colar certos trechos de texto em uma planilha de codificação.

As tarefas repetitivas, mecânicas e previsíveis podem ser automatizadas através do uso do computador. Nesse sentido, o programa R (R CORE TEAM 2013) é de grande valia para a otimização do tempo empregado na execução dessas tarefas. O R é uma linguagem de programação voltada à análise de dados, que pode ser utilizada para realizar computações estatísticas e gráficas, compilar e anotar *corpora*, produzir listas de frequências, entre diversas outras tarefas. Uma de suas principais vantagens é o fato de ser gratuito e estar disponível para uma variedade de plataformas (UNIX, Windows e MacOS).

Sendo uma linguagem de programação, o R permite que o usuário customize uma série de tarefas que deseja executar e, consequentemente, tenha maior controle sobre os resultados obtidos. Isso significa, no entanto, que ao invés de clicar em botões com funções limitadas e pré-definidas, o usuário normalmente define as funções que deseja executar através de *linhas de comando*, que instruem o programa sobre o que fazer. Uma sequência de linhas de comando é chamada de *script* ou *código*. O exemplo (1) a seguir mostra um pequeno *script*, que instrui o

R a carregar um arquivo de transcrição na memória, apagar as marcas de parênteses e salvar o arquivo limpo.

(1)

```
> FabianaB<-scan(file=choose.files(),what="char",sep="\n")¶
> FabianaB.limpo<-gsub("\\(|\\)","",FabianaB)¶
> cat(FabianaB.limpo,file="FabianaB-limpo.txt",sep="\n")¶
```

Embora isso possa parecer complicado inicialmente, um pouco de prática levará o usuário a se familiarizar com o ambiente. Em geral, o esforço de criar um script só precisa ser feito uma vez, já que podemos salvar o código e reutilizá-lo quantas vezes forem necessárias, modificando apenas pequenas partes para readaptá-lo às novas demandas. Além disso, há uma série de scripts escritos previamente por outros usuários, na forma de *funções* e *pacotes*, que podemos baixar da internet e utilizar em nossas próprias tarefas. Tal é o caso das funções `identificacao()`, `extracao()` e `amostragem()`, do pacote `dmsocio`, que serão mais detalhadamente exploradas na seção 4 adiante. Antes de descrever a aplicação dessas funções, é necessário tratar da instalação do programa (seção 1), de conceitos básicos e algumas funções úteis para sua utilização (seção 2), e de arranjos prévios na organização de nosso *corpus* (seção 3). O artigo se encerra com uma visão perspectiva dessas funções no âmbito das análises sociolinguísticas e com a indicação de leituras adicionais para um maior aprofundamento das aplicações do R aos estudos linguísticos.

Não é demais salientar que, em se tratando de um tutorial prático, este texto foi pensado para ser lido com um computador à mão, de modo que o leitor possa reproduzir os exemplos durante a leitura. Não há como aprender a usar o R sem utilizá-lo. Portanto, mãos à massa!

1. INSTALAÇÃO DO PROGRAMA R

O primeiro passo para começar a utilizar o R é sua instalação. O programa pode ser baixado gratuitamente do site do Projeto R, no endereço <http://cran.r-project.org/>. Na seção *Download and Install R*, clique no link referente ao seu sistema operacional (Linux, MacOS ou Windows) e instale o R no diretório sugerido.

Inicie o programa. Você verá que o R possui uma interface bastante simples: um menu na parte superior com algumas opções usuais (Arquivo, Editar, etc.); alguns botões de comandos mais frequentes (Abrir script, Salvar, Imprimir, etc.); e uma janela do *Console* que informa a versão instalada e algumas breves notas

sobre o R. Na última linha, os sinais em (2) indicam que o R está pronto para receber comandos.

(2)

```
>|
```

Existe atualmente uma interface mais "amigável", chamada RStudio, que disponibiliza algumas ferramentas adicionais diretamente na interface gráfica, como a visualização dos *scripts* abertos recentemente, o histórico de linhas de comando executadas e a lista de pacotes instalados. O RStudio é, de fato, apenas uma interface gráfica e alternativa e sua utilização requer a instalação do programa R em seu computador (como fizemos na etapa acima). A instalação do RStudio é opcional, mas altamente recomendável. O programa pode ser baixado a partir do endereço <http://www.rstudio.com/ide/download/>. Ao escolher *Download RStudio Desktop*, o site identifica automaticamente o seu sistema operacional e mostra a versão adequada no item *Recommended for your system*. Baixe essa versão para o seu computador, instale e inicie o programa.

A interface do RStudio apresenta quatro janelas (Figura 1):

i. *Source*, para visualização e edição de *scripts*;
ii. *Environment* e *History*, com os objetos carregados na memória do R para a presente sessão e com o histórico de linhas de comando executadas;
iii. *Console*, no qual as funções e os *scripts* são executados;
iv. *Files*, *Plots*, *Packages*, *Help* e *Viewer*, respectivamente, para arquivos, gráficos, pacotes, ajuda e visualizador.

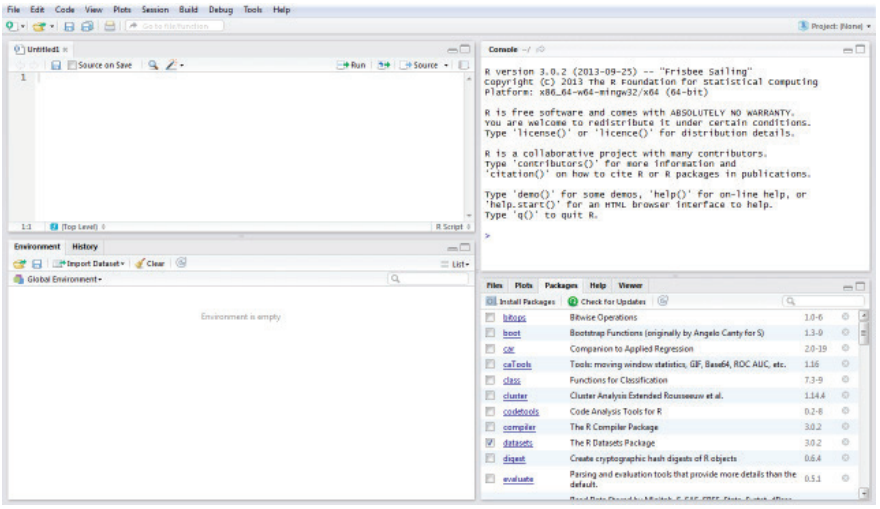


Figura 1 – Interface do RStudio.

Além de instalar o programa R e o RStudio, baixe os materiais de apoio a esse tutorial que estão disponíveis em <http://projetosp2010.fflch.usp.br/dmsocio>. Nessa página, clique sobre o *link Scripts+Exemplos+Minicorpora* e salve o arquivo em seu computador. Essa é uma pasta zipada que contém : (i) o arquivo "dmsocio.R", com o código para as funções `identificacao()`, `extracao()` e `amostragem()`; (ii) o arquivo "dmsocio-exemplos.R", com instruções simplificadas, modelos de como usar as funções do pacote dmsocio e os exemplos deste tutorial; (iii) os arquivos CodifR-12ent.txt e Ex26-MauricioB.txt; e (iv) dois *Minicorpora* (dmsocio-SP2010 e dmsocio-codif-R), que contêm cada qual a transcrição de 12 entrevistas sociolinguísticas do Projeto SP2010 (MENDES; OUSHIRO, 2013) e que serão utilizados na exemplificação da aplicação das funções. Há também uma pasta chamada "UTF-8", com os mesmos arquivos .txt em formato UTF-8 de codificação. Para abrir o arquivo zipado é necessário ter um descompactador, como os programas Winzip ou Zipeg. Depois de descompactá-lo, abra o arquivo "dmsocio-exemplos.R" no RStudio (*File* > *Open file*...), que usaremos na seção 4.

2. CONCEITOS BÁSICOS PARA USO DO PROGRAMA R

Como visto em (2) acima, os símbolos >| indicam que o programa R está pronto para receber comandos. Você pode, por exemplo, pedir que o R calcule o resultado de 2 + 3 como na linha de comando abaixo:[1]

(3)

```
> 2 + 3 ¶
```

Ao digitar 2 + 3 e pressionar ENTER, o R fornecerá a resposta da operação matemática dada em (3):

(4)

```
[1] 5
```

O número [1] simplesmente dá uma referência de quantos resultados existem para o cálculo requerido (para outras funções, o número de resultados pode estar na casa de centenas, milhares, milhões...). A resposta para a linha de comando em (3) é fornecida logo em seguida: 5. No entanto, deve estar claro que nosso verdadeiro interesse em utilizar o R não é como uma simples calculadora, mas que o programa realize operações muito mais complexas. Ainda em um exemplo simples, pode-se pedir que o R guarde o resultado dentro de um objeto nomeado pelo próprio usuário, para ser acionado posteriormente. Para isso, basta especificar o nome do objeto, seguido dos símbolos <-, que iconicamente indicam onde o R deve guardar tal informação. Desse modo, em vez do exemplo (3), podemos instruir que o R calcule o resultado de 2 + 3 e guarde-o em um objeto chamado **x** (ou qualquer outro nome que você queira dar[2]) do seguinte modo:

(5)

```
> x <– 2 + 3 ¶
```

1 Nos exemplos fornecidos daqui em diante, ">" indica que o R está disponível para novos comandos, e "¶" representa a tecla [ENTER], para que a linha de comando seja executada. Não se trata, portanto, de caracteres que devam ser digitados. Todos os exemplos deste artigo se encontram ao final do arquivo "dmsocio-exemplos.R". No RStudio, linhas de comando em um script aberto em *Source* são rodadas no console através de [CTRL] + [ENTER] (Windows) ou [Command] + [ENTER] (MacOS).

2 Em princípio, pode-se atribuir qualquer nome a um objeto. No entanto, a utilização de um mesmo nome para objetos diferentes fará com que o programa substitua o antigo.

Repare que, desta vez, o R não fornece o resultado do cálculo como fez em (4); ele pode ser acessado instruindo o R a mostrar qual é o conteúdo de **x** (ou outro nome que você tenha atribuído ao objeto), ao que o R responde 5:

(6)

```
> x ¶
[1] 5
```

Isso significa que **x** agora funciona como uma variável que equivale a 5 e pode ser utilizado em outras computações. Assim, se digitarmos a linha de comando **x+10**, o R retornará o resultado 15.

(7)

```
> x + 10 ¶
[1] 15
```

O objeto criado **x** deixará de ter o valor 5 até que a sessão corrente do R seja encerrada, até que o nome **x** seja atribuído a outro objeto (8), ou até que seu valor seja removido com a função `rm()` – *remover* (9):

(8)

```
x <- (16-4)*2 ¶
```

(9)

```
rm(x) ¶
```

Além de guardar valores numéricos, o R também pode guardar valores textuais, como caracteres, palavras, sentenças, transcrições de entrevistas sociolinguísticas e tabelas de dados, que concernem mais propriamente aos nossos interesses presentes. Tais informações podem ser armazenadas em diferentes tipos de objetos, chamados vetores e *dataframes*.

2.1. Vetores e dataframes

O tipo mais simples de uma estrutura de dados (e aquele que mais utilizaremos nos *scripts* para análises sociolinguísticas) é o *vetor*, uma sequência unidimensional de elementos como números ou caracteres (palavras, sentenças, textos). De fato, o objeto **x** criado acima é um vetor que contém apenas um elemento, 5.

Em lugar de valores numéricos, um vetor pode abarcar valores textuais, como na criação do vetor **y** abaixo:

(10)

```
> y <- "gato" ¶
```

Note que, para ser entendido como uma sequência de caracteres, o(s) valor(es) atribuído(s) ao vetor deve(m) vir entre aspas. Na próxima subseção, veremos como criar vetores com mais de um elemento.

O *dataframe* é uma estrutura de dados bidimensional, com linhas e colunas, equivalente a tabelas (como, por exemplo, uma planilha de dados do Excel). Colocado de outro modo, o *dataframe* é um conjunto de vetores de mesma extensão (mesmo número de linhas ou número de colunas). Para nossos propósitos, o *dataframe* será importante para a criação de planilhas de dados extraídos das transcrições de entrevistas sociolinguísticas, ou para que o R possa "ler" planilhas de dados previamente criadas.

2.2. Funções e argumentos

Como visto, o R funciona através de uma interface em que o usuário digita certas linhas de comandos a serem executados e o programa retorna os resultados no console ou os armazena em objetos. Na maior parte dos casos, as linhas de comando contêm *funções*, que instruem o programa sobre o que deve ser feito. Cada função contém um ou mais *argumentos*, que são variáveis pré-definidas e indicam, entre outras coisas, os dados com que se deseja trabalhar e como eles devem ser tratados.

O item (11) mostra a sintaxe básica de funções no R: ela consiste no nome da função e a especificação dos argumentos dentro de parênteses e separados por vírgulas.

(11)

```
funcao(arg. 1, arg. 2, arg. 3...)
```

Três funções serão empregadas na utilização dos *scripts* para análise de dados sociolinguísticos: `c()`, `getwd()/ setwd()`, e `read.table()`.

`c()`: Função genérica para combinar elementos dentro de um vetor. Nos exemplos (5) e (10), havíamos criado os vetores **x** e **y**, cada qual com apenas um elemento. Com a função `c()`, podemos criar um vetor com um número maior de elementos, por exemplo, os nomes das variáveis sociais de nosso *corpus*:

(12)

```
>variaveis.sociais<-c("sexo.genero","faixa.etaria","escolaridade")¶
>variaveis.sociais¶
[1] "sexo.genero" "faixa.etaria" "escolaridade"
```

Em (12), `variaveis.sociais` é o nome do vetor/objeto que contém os argumentos "sexo.genero", "faixa.etaria" e "escolaridade". Se pedirmos ao R que mostre o que é `variaveis.sociais` (linha 2 do exemplo), ele retornará os valores do vetor (linha 3 do exemplo). É importante notar que o nome do vetor poderia ser qualquer outro (por exemplo, "var.sociais", "aa", "x" etc.), e que o número de elementos a serem combinados (três, no exemplo acima) poderia ser dois, quatro, vinte, dez mil...

`getwd()/ setwd()`: `getwd()` é uma função que mostra qual é o diretório de trabalho (wd = *working directory*) atual do R. Quando se deseja buscar um arquivo ou salvar resultados dentro do computador, essa é a pasta que o R usará como referência. Para especificar um diretório de trabalho diferente, usa-se a função `setwd()`, cujo argumento é o caminho completo da pasta dentro do sistema, definido entre aspas. Por exemplo:
(13)

```
setwd("C:/Documentos/dmsocio/dmsocio-SP2010")¶
```

No RStudio, em vez de digitar o caminho completo, pode-se definir o diretório de trabalho na aba *Files*, em uma das janelas da interface. O programa indica o diretório de trabalho atual no topo da janela (Figura 2a). Uma nova pasta pode ser escolhida clicando sobre as reticências no canto direito; na janela que se abrir, indique a localização da nova pasta (Figura 2b). O caminho completo para a nova pasta aparece no topo da aba. Em seguida, clique em *More* e em *Set As Working Directory* (Figura 2c).

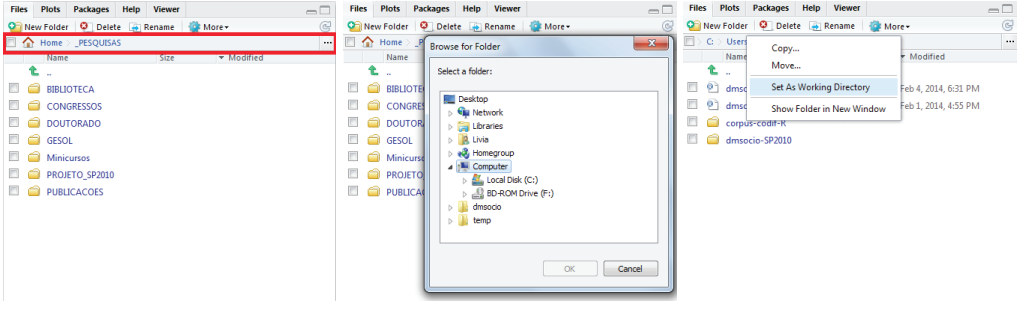


Figura 2 – Aba *Files* do RStudio: (a) diretório atual; (b) navegação para uma nova pasta; (c) definição de nova pasta de trabalho.

`read.table()`: Função para carregar um arquivo de dados no formato de tabela (por exemplo, uma planilha do Excel armazenada em formato .txt) em um vetor do R. Essa função possui diversos argumentos, alguns dos quais com um valor padrão (*default*). Em geral, a existência de um valor padrão ocorre quando há um número limitado de opções para o preenchimento de um argumento – por exemplo, **T** (verdadeiro, do inglês *true*) ou **F** (falso, do inglês *false*). Note que, no caso das funções `c()`e `getwd()/setwd()`, não há um valor padrão para os respectivos argumentos, dado que o número de possibilidades de preenchimento é infinito: para `c()`, pode-se definir quaisquer elementos a serem combinados e em qualquer número; para `getwd()/setwd()`, o caminho do diretório de trabalho depende das pastas existentes no sistema do usuário, que podem ter qualquer nome. Quando uma função possui valor padrão para certos argumentos, pode-se simplesmente não especificá-los, caso em que o R automaticamente assume que o valor do argumento é o padrão. Os argumentos de `read.table()` mais relevantes para nossos propósitos, com seus respectivos valores padrão, são os seguintes:

(14)
```
read.table(file=...,header=F,sep="",quote="",comment.char="#")
```

- O argumento *file*, obrigatório, define o arquivo a ser lido pelo R. Pode-se especificar o caminho completo até ele (por exemplo, C:/Documentos/dmsocio/CodifR-12ent.txt) ou, quando o arquivo está no diretório de trabalho definido com `setwd()`, basta especificar o nome do arquivo (por exemplo, CodifR-12ent.txt). Alternativamente, o argumento de *file* pode ser especificado como `choose.files()` (no Linux, como `file.choose()`); nesse caso, o R abre uma janela de navegação na qual o usuário pode localizar o arquivo desejado e clicar sobre ele;
- *header* especifica se a primeira linha da tabela de dados a ser lida pelo R contém o nome das colunas ou não. O valor padrão é **F** (= falso), mas se a tabela que você deseja carregar no R possui o nome das colunas, esse argumento deve ser definido como **T** (= verdadeiro);
- *sep* define o caractere que separa os campos da tabela (espaços, tabulações – tabs, vírgulas, ponto-e-vírgulas, etc.). Em muitos casos, utilizaremos o tab como separador de campos, representado no R como `"\t"`;
- *quote* define os caracteres usados para citação. Na maior parte dos casos, é melhor defini-lo como `quote=""` (ou seja, nenhum), para evitar problemas na leitura do arquivo pelo R;

- *comment.char* define o caractere usado para comentários. Em linguagens de programação, o caractere de comentário é usado para instruir o programa a ignorar tudo que vem depois dele, de modo que o usuário possa fazer anotações sobre suas linhas de comando. No R, o caractere de comentário padrão é # mas, para nossos propósitos (e a depender das normas de transcrição de seu *corpus*), provavelmente você preferirá defini-lo como `comment.char=""`.

Os materiais de treino para uso do pacote `dmsocio` incluem uma tabela de dados chamada "CodifR-12ent.txt". Ela pode ser carregada no R através da linha de comando exemplificada em (15), que instrui o programa: (i) a abrir a janela de navegação (`file=choose.files()`); (ii) que a primeira linha da tabela é o nome das colunas (`header=T`); (iii) que o caractere que separa os campos é o tab (`sep="\t"`); (iv) que não há caractere de citação nem de comentário (`quote=""; comment.char=""`); e (v) que tais dados devem ser armazenados no vetor chamado `dadosR` (`dadosR<-`).

(15)

```
dadosR<-read.table(file=choose.files(), header=T, sep="\t",
quote="", comment.char="")¶
```

Além dessas funções, é útil descrever outras cinco adicionais: `scan()`, `grep()`, `gsub()`, `cat()` e `write.table()`. Embora você não precise conhecê-las para utilizar o pacote `dmsocio`, trata-se de funções que são empregadas dentro das rotinas dos *scripts* que compõem o pacote e que podem ser úteis em suas próprias explorações de arquivos de dados.

`scan()`: Função para carregar dados de um arquivo de texto (por exemplo, um arquivo de transcrição) em um vetor. Assim como a função `read.table()`, `scan()` possui diversos argumentos. Os mais relevantes para nossos propósitos são discriminados em (16) e exemplificados em (17):

(16)

```
scan(file=..., what="double(0)", sep="", comment.char="")
```

(17)

```
> FabianaB<-scan(file=choose.files(),what="char",sep="\n")¶
> FabianaB ¶
```

- *file* é obrigatório e especifica o caminho do arquivo a ser carregado, de modo semelhante à função `read.table()`. A especificação `file=choose.files()` abre a janela de navegação para que o arquivo seja localizado manualmente;

- *what* especifica o tipo de dados que o arquivo contém: números (`what="double(0)"`) ou caracteres de texto (`what="char"`) – este último, mais frequentemente, será o nosso caso;
- *sep* define o caractere que separa os elementos do vetor. O *default* `sep=""` divide o texto em palavras e guarda cada uma como um elemento do vetor; `sep="\n"` divide o texto por parágrafos (cada parágrafo se torna um elemento do vetor);
- *comment.char* define o caractere usado para comentários, de modo semelhante à função `read.table()`.

No exemplo em (17), pedimos ao R que abra a janela de localização; podemos então clicar sobre o arquivo "SP2012-009-F26SEL-FabianaB.txt", na pasta "dmsocio-2010", para carregar essa transcrição no vetor `FabianaB`. Ao digitar o nome do objeto recém-criado no console, o R retorna seus elementos (a transcrição).[3]

É importante apontar que qualquer operação realizada com o vetor (por exemplo, as funções `grep()` e `gsub()` discutidas abaixo) **não** altera o arquivo original, uma vez que os dados estão armazenados na sessão corrente do R. Se desejamos salvar as alterações ou resultados, podemos usar as funções `cat()` ou `write.table()` (ver adiante).

`grep()`: Função que realiza a busca de padrões dentro de um vetor.

(18)

```
grep(pattern, x, ignore.case=F, value=F)
```

(19)

```
> chopes<-grep("chopes", FabianaB, ignore.case=T, value=F)¶
> chopes ¶
```

- *pattern* se refere ao padrão a ser buscado (a subseção 2.3 tratará disso mais detalhadamente);
- *x* define o vetor em que a busca deve ser realizada;

3 No exemplo 17, se os caracteres acentuados como "ã" e "ó" não aparecerem corretamente no console após digitar `"FabianaB"`, tente uma das soluções seguintes: (i) clique sobre *Tools* no menu superior e, em seguida, em *Global options*; na janela que aparecer, no menu à esquerda, clique em *General* e mude o *Default text encoding* para *Windows-1252*; ou (sobretudo para usuários de Linux) (ii) utilize os arquivos em formato .txt – UTF-8 da pasta "dmsocio", mantendo o *Default text encoding* como UTF-8.

- *ignore.case* define se a diferença entre letras maiúsculas e minúsculas (por exemplo, entre "SP" e "sp") deve ser ignorada ou não. O *default* é **F**, ou seja, o R não deve ignorar a diferença maiúsculas e minúsculas, mas o usuário pode definir o argumento como **T** caso a diferença seja irrelevante;
- *value* retorna a localização das ocorrências quando **F**, ou as próprias ocorrências quando **T**.

No exemplo em (19), a função `grep()` instrui o programa a buscar ocorrências do item lexical "chopes" no vetor `FabianaB`, e guardar os resultados no vetor `chopes`. Como `value=F`, o resultado é a sua localização dentro do vetor.

`gsub()`: Função que localiza padrões em um vetor e os substitui por outro valor. Os argumentos *pattern* e *x* são definidos de modo semelhante à função `grep()`. O argumento adicional *replacement* define o valor que deve substituir o padrão.

(20)

```
gsub(pattern, replacement, x, ignore.case=F)
```

(21)

```
> FabianaB.falantes<-gsub("D1","Doc",FabianaB)¶
> FabianaB.falantes<-gsub("S1","Inf",FabianaB.falantes)¶
> FabianaB.falantes¶
```

No exemplo (21), substituimos todas as ocorrências de "D1" por "Doc", e de "S1" por "Inf". O resultado pode ser visualizado com a chamada do vetor `FabianaB.falantes`.

`cat()/write.table()`: Funções que permitem salvar, respectivamente, vetores e *dataframes* em arquivos. Em ambas, *x* se refere ao objeto a ser salvo, *file* ao nome do arquivo (junto com sua extensão, por exemplo, .txt), e *sep* define o tipo de separador dos elementos (espaços `" "`, parágrafo `"\n"`, tab `"\t"`, etc.). `write.table()` possui o argumento adicional *append*, que define se novos valores devem ser anexados ao fim da tabela (**T**) ou não (**F**).

(22)

```
cat(x, file="",sep="")
```

(23)

```
> cat(FabianaB.falantes, file = "FabianaB-part.txt", sep="\n")¶
```

(24)
```
write.table(x, file="", append=F, sep="")
```

No exemplo em (23), a função `cat()` salva o vetor recém-criado `FabianaB.falantes` no arquivo "FabianaB-part.txt" no atual diretório padrão.

2.3. Expressões regulares

Expressões regulares são sequências de caracteres que especificam padrões de busca. Através do uso de caracteres especiais, pode-se definir um padrão que contém não apenas grafemas específicos como "a", "bol" ou "chopes", mas também a localização desses caracteres dentro de uma palavra ou expressão, caracteres opcionais ou conjuntos de caracteres que podem ocupar uma determinada posição. O Quadro 1 apresenta os caracteres especiais que serão mais utilizados em nossas definições de padrões a serem buscados nas entrevistas sociolinguísticas.

[]	Trata o que está dentro como uma classe de caracteres
|	"Ou"
.	"Qualquer caractere"
*	"Zero ou mais ocorrências da expressão regular precedente"
+	"Uma ou mais ocorrências da expressão regular precedente"
?	"Zero ou uma ocorrência da expressão regular precedente"
\\b	Fronteira de palavra
\\d	Qualquer dígito
\\s	Qualquer espaço
\\	Caracteres de escape

Quadro 1 – Caracteres especiais para definição de expressões regulares.

Tomemos como exemplo dois pequenos trechos extraídos da entrevista com o informante MauricioB, parte do *corpus* do Projeto SP2010 e de nossa amostra para treino do uso das funções do pacote dmsocio, em que o informante fala sobre os problemas de São Paulo. Em (25), "D1" representa a fala do documentador e "S1" a fala da informante.

(25)

```
 1  S1 já já ouvi/ já vi várias pesquisas tal porque o que que
 2     está apavorando hoje o morador de São Paulo...
 3     é a maldita violência né?
 4  D1 uhum
 5  S1 é a maldita violência então isso a pessoa... pensa
 6     duas vezes três vezes por mais que apaixonada seja
 7  S1 ela vai falar não eu quero um lugar pra... pra ter
 8     mais tranquilidade
 9  S1 mas em contrapartida você também lê toda hora
10     que... os lugares mais calmos do interior já estão
11     sendo
12  S1 vítimas aí de arrastão e de assalto de tudo quanto
13     que é tipo
14  D1 aham
15  S1 então não sei até que... até que ponto valeria a pena
```

[...]

```
16  S1 poluição está
17  S1 São Paulo hoje em dia... fica três dia sem chover
18     você começa a lacrimejar já começa
19  S1 a travar tua garganta
20  S1 (quer dizer)
21  D1 você sofre com isso assim com essas coisas?
22  S1 ah eu não sentia até tempo atrás hoje em dia eu sinto
```

O mesmo trecho pode ser carregado em um vetor no R com a linha de comando em (26). Ao abrir a janela de navegação, escolha o arquivo "Ex26-MauricioB.txt".

(26)

```
> MauricioB<-scan(file=choose.files(),what="char",sep="")¶
> MauricioB¶
```

Imagine que um pesquisador esteja interessado em buscar todas as ocorrências da realização de /e/ nasal, em posição pré-consonantal, que não ocorram no início, nem no final de palavra. Se o pesquisador tiver acesso a uma transcrição fonética do *corpus*, sua tarefa será bem mais simples. No entanto, na maior parte dos casos, o sociolinguista lida com transcrições ortográficas das gravações. A

tarefa aqui é definir possibilidades ortográficas (do mundo da escrita) que representam um segmento fônico (do mundo da fala).

No trecho acima, as palavras-alvo são as ocorrências da palavra "violência" (linhas 3 e 5), "pensa" (linha 5), "sendo" (linha 11), "sentia" e "tempo" (linha 22). Uma busca apenas pela sequência "en" encontraria as palavras "pensa", "sendo" e "sentia", mas deixaria de encontrar as ocorrências da palavra "violência", que contém o acento circunflexo, e "tempo", em que /e/ nasal é representado graficamente por "em". Ao mesmo tempo, a busca por "en" também encontraria as palavras "então" e "pena", que não fazem parte do conjunto: a primeira pelo fato de /e/ nasal estar em posição inicial da palavra e a segunda por ser seguido de vogal. Por outro lado, a busca pela sequência "em" teria problemas semelhantes: apesar de encontrar, corretamente, a ocorrência da palavra "tempo", a busca também retornaria com as palavras "em" e "sem", em que /e/ nasal se encontra no início e no final da palavra, respectivamente.

Desse modo, queremos que a busca inclua: (i) não apenas o grafema "e", mas também suas versões acentuadas ("ê" e "é"); (ii) não apenas os grafemas "e", "ê" e "é" seguidos de "n", mas também aqueles seguidos de "m"; (iii) que as sequências dos grafemas "en", "ên", "én", "em", "êm", "ém" sejam seguidas de consoantes ("b", "c", "ç", "d" etc.) e (iv) que tais sequências não ocorram nem no início nem no final da palavra.

Caso resolvêssemos fazer uma tal busca em um editor de texto (como o Word ou Bloco de Notas), seriam necessárias múltiplas definições da sequência de caracteres desejada – contando as seis combinações de "e"/"ê"/"é" com "n"/"m" acima, com todas as possíveis combinações de consoantes seguintes, o número total de definições de sequências certamente estaria na casa de centenas. Com o emprego de expressões regulares e dos caracteres especiais do Quadro 1, é possível definir o padrão desejado de modo bastante econômico. Vejamos:

Os caracteres especiais `[]` tratam aquilo que está dentro como uma classe de caracteres. Para definir que desejamos ocorrências de "e", "ê" ou "é", podemos criar uma classe de caracteres `[eêé]`[4]. De modo semelhante, podemos definir a presença de "n" ou "m" como `[nm]` e a presença de uma consoante como `[bcç-dfgjklpqrstvxz]`. Veja que essa última classe de caracteres não inclui "h" (pois faria com que palavras como "nenhum" entrassem no conjunto de palavras-alvo), tampouco "n" ou "m" (pois buscaria sequências de "nn" e "mm", inexistentes no português). Até o momento temos, portanto, a seguinte sequência de caracteres:

(27)

```
[eêé][nm][bcçdfgjklpqrstvxz]
```

4 A mesma sequência de caracteres poderia ser notada como [elêlé], utilizando o caractere especial | (ou). No entanto, como [] cria uma classe de caracteres, o uso de |, nesse caso, seria redundante.

Note que a definição de que /e/ nasal não pode ocorrer em final de palavra já está incluída na sequência em (27), uma vez que a presença de uma consoante após [nm] garante que o segmento não estará no final na palavra.

Agora precisamos definir que a sequência em (27) não deve ocorrer em início de vocábulo. Para tanto, podemos usar os caracteres especiais `\\b`, que delimitam fronteiras de palavra. Se a sequência não pode ocorrer no início, isso significa que há algum caractere – qualquer que seja – após o seu início, o que é representado pelo ponto final (.) (qualquer caracter); além disso, pode haver apenas um caractere antes da sequência (como na palavra "t-emp-o"), ou pode haver mais (como na palavra "viol-ênc-ia"), o que se representa pelo caractere especial + (uma ou mais ocorrências da expressão regular precedente, nesse caso, o ponto final). Podemos, portanto, atualizar nossa definição como em (28), e inseri-la na função `grep()` para localizar tais ocorrências no vetor `MauricioB` (29):

(28)

```
\\b.+[eêé][nm][bcçdfgjklpqrstvxz]
```

(29)

```
> grep("\\b.+[eêé][nm][bcçdfgjklpqrstvxz]", MauricioB, value=T)¶
```

A expressão regular em (28-29) identifica corretamente as ocorrências de /e/ nasal "violência", "pensa", "sendo", "sentia" e "tempo", ao mesmo tempo em que não identifica as palavras "então", "pena", "também", "em" e "sem" como parte do conjunto de palavras-alvo. Ela cumpre os dois critérios que devem ser atendidos quando se define uma expressão regular: (i) que abarque **todos** os casos do padrão buscado; e (ii) que abarque **somente** os casos do padrão buscado.

Alguns exemplos adicionais (incluindo a utilização de caracteres especiais não discutidos nessa subseção) serão apresentados adiante.

3. PREPARAÇÃO

3.1. Regra #1: Conheça sua variável!

O exemplo fornecido na subseção anterior pode parecer idiossincrático: por que buscar "ocorrências da realização de /e/ nasal, em posição pré-consonantal, que não ocorram no início e nem no final de palavra?" Tal definição faz parte do contexto variável para o estudo da realização de /e/ nasal no português paulistano como um monotongo (por exemplo, [fa.ˈzẽ.da]) ou ditongo (por exemplo, [fa.ˈzẽ̃j.da]) (OUSHIRO, 2013). Na análise qualitativa do *corpus*, notou-se que

as ocorrências de /e/ nasal em sílabas átonas e em posição inicial (por exemplo, "então", "em" e "engravidar") ou final (por exemplo, "fazem" e "vagem") eram quase que invariavelmente realizadas como [ĩ] – e, no caso das pós-tônicas finais, por vezes desnasalizadas [i]. Essas ocorrências podem ser entendidas como manifestação de outras variáveis, a saber, o alçamento de vogais pré e pós- tônicas (/e/ → [i]) (Cf. por exemplo, VIEGAS, 1987; BATTISTI, 1993; CELIA, 2004; TENANI; SILVEIRA, 2008) e a desnasalização (/ẽ/ → [i]) (Cf. por exemplo, VOTRE, 1978; GUY, 1981; BATTISTI, 2002; SCHWINDT; SILVA, 2009). Em monossílabos tônicos (por exemplo, "tem" e "vem") ou oxítonas (por exemplo, "também" e "armazém"), o segmento é invariavelmente realizado como ditongo [ẽj̃]. Em posição pré-vocálica (por exemplo, "pena") ou quando há assimilação de "nd" (como em "fazendo"), o segmento é invariamente realizado como monotongo.

Todas essas observações advêm da leitura da bibliografia relevante e de uma análise qualitativa cuidadosa do *corpus*, com vistas à definição do *contexto variável* (LABOV, 1969): qual é o conjunto das variantes e em quais contextos elas podem ocorrer (mesmo que não tenham ocorrido)?

Nesse ponto, vale destacar o argumento lançado inicialmente: para tarefas repetitivas, mecânicas e previsíveis, o R é uma ferramenta poderosa na redução do tempo empregado para o seu cumprimento, mas o programa por si só não pode fornecer respostas a perguntas que cabem ao linguista e que dependem de seu conhecimento especializado. Em outras palavras: o programa não pensa por você. Ele pode realizar tão somente as tarefas que você definir.

Na aplicação das funções `identificacao()` e `extracao()`adiante, discutiremos exemplos com outras três variáveis do português: (i) o emprego de diminutivos; (ii) a pronúncia de (-r) em coda silábica e (iii) o uso dos pronomes de primeira pessoa do plural "nós" e "a gente". Por ora, deixemos de lado a definição das expressões regulares correspondentes; antes disso, é necessário refletir sobre as questões que justificam o interesse por esses fenômenos linguísticos e tomar decisões quanto ao contexto variável.

Em um levantamento preliminar sobre a fala *gay* no português (MENDES, 2011), alguns dos informantes disseram associar o uso de diminutivos (por exemplo, "casinha" e "amiguinho") com a fala das mulheres e que, quando empregado de modo "exagerado", um falante do sexo masculino poderia ser percebido como *gay*. Desse modo, Mendes (2012) investigou se de fato há diferenças no emprego de diminutivos de acordo com o gênero dos falantes – homens e mulheres, *gays* e heterossexuais. Tal empreitada, no entanto, conduziu a um desafio metodológico: como definir o contexto variável para o uso de diminutivos? Em português, os diminutivos podem ocorrer em qualquer substantivo ("casa", "casinha"), adjetivo ("perto", "pertinho"), advérbio ("falar rápido", "falar rapidinho") e gerúndio ("tá chovendo", "tá chovendinho"). Extrair *todas*

as ocorrências dessas classes de palavras e codificá-las de acordo com a presença ou não de sufixo de diminutivo parecia ineficaz. Mendes então decidiu coletar apenas as ocorrências de diminutivos no *corpus* e, sabendo o número total de palavras por entrevista sociolinguística, calculou o número de ocorrências por mil palavras. Esse índice foi usado na comparação da frequência de diminutivos na fala de homens e mulheres, *gays* e heterossexuais.

A pronúncia de /r/ em português, tanto em posição de ataque quanto em posição de coda silábica, possui múltiplas variantes: vibrante múltipla, tepe, aproximante retroflexa, fricativa velar e glotal (surda e sonora), apagamento. As diferentes realizações também costumam ser associadas às diferentes variedades regionais e sociais. Em posição de coda silábica, as variantes mais comumente associadas ao português paulistano são o tepe e o retroflexo. Na capital paulista, o retroflexo tradicionalmente se associa aos falantes "caipiras", mas as suas indexicalidades parecem estar se expandindo para abarcar uma identidade de morador de periferia urbana. Em seu estudo sobre a realização de /-r/ em coda, Oushiro e Mendes (2013) incluíram as ocorrências tanto em contexto medial (por exemplo, "porta") quanto final (por exemplo, "mulher") e excluíram ocorrências de (-r) em fronteira de palavra seguido por vogais (por exemplo, "por isso"), uma vez que em tais ocorrências o segmento /r/ passa a ocupar a posição de ataque silábico ([po.ˈri.sʊ]).

Por fim, um dos principais interesses em analisar o emprego variável do pronome de primeira pessoa do plural é o processo de mudança linguística com o surgimento de um novo pronome, "a gente", que se encontra em alternância com o pronome mais antigo "nós" (ZILLES, 2005). Para esse caso, cabe ao pesquisador decidir se analisará somente os casos nominativos, na posição de sujeito ("nós fomos/foi", "a gente fomos/foi"), ou se também analisará os pronomes em outras posições sintáticas, no caso acusativo ("ele nos viu", "ele viu a gente"), como possessivo ("o nosso bairro", "o bairro da gente"), etc. Tais decisões devem ser tomadas em consonância com as questões que norteiam a pesquisa, em diálogo com a literatura relevante e de acordo com os dados disponíveis ao pesquisador.

3.2 Regra #2: Conheça seu corpus

A definição de expressões regulares para busca de ocorrências no *corpus* depende fundamentalmente das normas de transcrição empregadas, que geralmente diferem entre grupos de estudos e pesquisadores (cf. CASTILHO; PRETTI, 1986 para o Projeto NURC, TENANI; GONÇALVES, s/d para o Projeto ALIP, MELLO; RASO, 2009 para o C-ORAL-BRASIL, MENDES; OUSHIRO, 2013 para o Projeto SP2010).

Um princípio geral que norteia a estipulação de normas de transcrição é a representação da fala por meio escrito, de forma fiel à língua oral, mas inteligível, de modo que o material coletado possa ser mais facilmente analisado. Entretanto, as normas de transcrição também levam em conta os interesses específicos dos pesquisadores e seus respectivos grupos de pesquisa, tendo em vista as análises que serão desenvolvidas a partir do material coletado (análises fonéticas, morfológicas, sintáticas, textuais, etc.)

Diferentes normas de marcação de pausas, alongamento de vogais, apagamento e assimilação de segmentos, hesitações, turnos dos falantes, dados contextuais, presença de cabeçalho, etc. fazem com que a busca de uma mesma expressão regular retorne resultados diferentes, a depender das convenções adotadas. Tomemos as sentenças em (30), todas inventadas, para examinar essa questão.

(30)
a. na 3ª, 4ª série, eu era muito bagunceira
b. daí ele gritou assim... "abre a PO::::rta!"
c. eles vão e [barulho de porta batendo] fazem
d. eu num vou falá(r) p(r)a ele que ele (es)tá errado né?

Um pesquisador interessado em analisar a pronúncia de (-r) em coda silábica deve coletar, desses pequenos exemplos, os itens "3ª" e "4ª" em (30a), "PO::::rta" em (30b), e "falá(r)" em (30d), e ignorar a ocorrência de "porta" em (30c), que se refere à anotação de um dado contextual e, portanto, não se trata de um dado linguístico. Uma expressão regular que define a variável (-r) em coda como [aáâeéêiíoóôuú]r[bcçdfgjklmnpqstvxwz] – ou seja, a sequência de uma vogal, o grafema "r" e uma consoante – não seria capaz de identificar as ocorrências corretamente – de fato, ela só identificaria a ocorrência em (30c), que é justamente aquela que não interessa na análise.

A depender das normas de transcrição do *corpus* com que você está trabalhando, pode ser mais interessante fazer certos ajustes *antes* de realizar a busca automática por ocorrências, ao invés de tentar dar conta de todas as possibilidades de transcrição ortográfica. Para fazer isso, vamos primeiro combinar os dados de (30) em um vetor chamado `exemplo.R`:

(31)

```
 > exemplo.R<-c("na 3ª, 4ª série, eu era muito bagunceira",
"daí ele gritou assim... "abre a PO::::rta!"",
"eles vão e [barulho de porta batendo] fazem",
"eu num vou falá(r) p(r)a ele que ele (es)tá errado né?")¶
```

Nesse exemplo, sabemos de antemão que as palavras "3ª" e "4ª" contêm ocorrências de (-r) em coda silábica; no entanto, em um *corpus* maior, o pesquisador pode primeiro querer inspecionar quais e quantas ocorrências contêm dígitos (`\\d`) não transcritos por extenso. Isso pode ser feito através do uso de uma expressão regular com o `grep()`:

(32)

```
> grep("\\d", exemplo.R, value=T) ¶
[1] "na 3ª, 4ª série, eu era muito bagunceira"
```

O pesquisador agora sabe que existem as palavras "3ª" e "4ª" que devem ser substituídas por sua transcrição por extenso. Depois disso, podemos instruir o R a substituir os sinais `:()[]` – e qualquer conteúdo dentro de `[]` – por `""` (ou seja, nada, o que equivale a apagá-los). Isso pode ser feito em uma única linha de comando, separando cada valor a ser apagado com | (ou). Aqui, no entanto, deve-se tomar uma medida adicional com os sinais de parênteses e colchetes, que fazem parte de um conjunto de caracteres com significado especial no R. Esses incluem: ^ $ * . + ? ! { } – alguns descritos na subseção 2.3 (ver também Gries, 2009, p. 81). Para que sejam entendidos literalmente, e não com o seu significado especial, devemos notá-los junto ao símbolo de escape `\\` – por exemplo, `\\(`. A mesma função `gsub()` pode ser utilizada para substituir "3ª" por "terceira" e "4ª" por "quarta". Note que em (33) abaixo, cada alteração é armazenada em um novo vetor (`exemplo.R1`, `exemplo.R2`, `exemplo.R3`) e que cada nova alteração é realizada no vetor mais atualizado.

(33)

```
> exemploR1<-gsub(":|\\(|\\)|\\[.+\\]", "", exemplo.R)¶
> exemplo.R2<-gsub("3ª", "terceira", exemplo.R1)¶
> exemplo.R3<-gsub("4ª", "quarta", exemplo.R2)¶
> exemplo.R3¶
```

Todas essas considerações também podem ser levadas em conta caso você esteja na fase de transcrição do *corpus*: quais normas facilitarão, posteriormente, o tratamento de dados[5]?

Muitos projetos de estudos sociolinguísticos costumam disponibilizar certas informações da ficha social do informante (sexo, idade, escolaridade, etc.) na forma de cabeçalho. A Figura 3 é um exemplo retirado da Amostra Censo/1980

5 As normas de transcrição do Projeto SP2010 foram definidas com essa preocupação em mente: a facilidade de tratamento de dados com o R. O manual de transcrições desse projeto (MENDES; OUSHIRO, 2013) pode ser acessado no endereço <http://projetosp2010.fflch.usp.br/producao-bibliografica>, no link *Manual de Transcrições*.

do Projeto PEUL[6] (PAIVA; SCHERRE, 1999). Além de dados institucionais, o cabeçalho identifica o informante, sua idade, escolaridade, bairro de residência e profissão. Esses dados (ou parte deles) poderão ser extraídos automaticamente com o R, para codificação das variáveis sociais de interesse na análise.

Universidade Federal do Rio de Janeiro - UFRJ
Centro de Letras e Artes – CLA
Faculdade de Letras
Departamento de Lingüística e Filologia
Programa de Estudos sobre o Uso da Língua - PEUL
Banco de Dados do PEUL/UFRJ
AMOSTRA CENSO/1980

Falante: 01 Sam
Idade: 18 anos
Escolaridade: Fundamental 1
Bairro: Santa Cruz
Profissão: ajudante de pedreiro

(gravação feita em ambiente com eco)
E- (fala cortada) (ruídos) (inint.) ("com sua mãe"). Com quem que você se dá melhor, dos seus irmãos?
F- Meus irmão, eu me dou melhor [com]- com o meu irmão abaixo de mim. (est)
E- Ele é muito mais novo?
F- Não, ele tem (falam) uns quinze anos. (est)

Figura 3 – Exemplo de transcrição do PEUL, com cabeçalho.

3.3. Arquivos

Antes de discutir a aplicação das funções do pacote `dmsocio`, alguns cuidados adicionais são necessários na preparação do *corpus*. Ainda que o programa R seja capaz de "ler" diversos tipos de arquivo, as funções que serão discutidas na próxima seção requerem que os arquivos de transcrição e o de dados estejam salvos no formato .txt (*Texto sem formatação* no Word e *Texto delimitado por tabulador* no Excel). Esse é, de fato, o formato mais flexível para o tratamento de dados textuais.

Em muitos casos, o sociolinguista dispõe de transcrições em formato .doc, .docx, .odt ou .pdf. No caso dos primeiros, basta abrir os arquivos em um editor de texto como o Word ou o Writer e salvá-los no formato .txt. Para arquivos em .pdf, há uma série de programas gratuitos na Internet que realizam a conversão automática de arquivos .pdf para .txt.

6 Disponível em <http://www.letras.ufrj.br/peul/cen80texto.html>. Acesso em: 30 jan. 2014.

Como se mencionou na subseção anterior, pode ser útil incluir um cabeçalho com informações sociais de cada informante, caso o seu *corpus* não tenha um. Os tipos de informação e a ordem em que aparecem podem variar, mas é importante manter consistência entre os arquivos: se a idade do informante aparece na quinta linha, ela deve aparecer na quinta linha para todos os informantes e transcrições. O mesmo vale para outras informações: se a fala do documentador é marcada por "E- " ("e" maiúsculo, traço, espaço – ver Figura 3), isso deve ocorrer ao longo de todas as transcrições. A regra mais básica aqui é: não importa o que se faça, faça consistentemente.

Para os arquivos de transcrição (mas não para os arquivos de dados), recomenda-se apagar as marcas de tabulação, se houver. No R, isso pode ser feito com a função `gsub()` (ver exemplo 33 acima), utilizando "\t" para representar os tabs. Isso é necessário pois a função `extracao()` gera uma tabela de dados separados por tabs; caso o arquivo original já contenha essas marcas, o de dados final pode acabar contendo algumas colunas adicionais indesejadas.

4. FUNÇÕES `IDENTIFICACAO()`, `EXTRACAO()`, `AMOSTRAGEM()`

Esta seção discute a aplicação de três funções do pacote `dmsocio`, desenvolvidas especificamente para o tratamento de dados sociolinguísticos. Os exemplos discutidos utilizam as transcrições do *minicorpus* "dmsocio-SP2010" (12 entrevistas sociolinguísticas do Projeto SP2010) e os arquivos codificados na pasta "dmsocio-codif-R". Para que as funções estejam disponíveis em cada sessão do R, é necessário rodar uma linha de comando que as carrega na memória:

(34)

```
> source("~/dmsocio.R")¶
```

em que "~" corresponde ao caminho completo da pasta em que se encontra o arquivo "`dmsocio.R`" no seu computador. Alternativamente, se você estiver conectado à internet pode rodar a seguinte linha de comando:

(35)

```
>source("http://tinyurl.com/dmsocio")¶
```

Essa segunda opção lê o script dmsocio.R diretamente do portal do Projeto SP2010. A vantagem dessa segunda opção é ter acesso a versões mais recentes, com todas as atualizações.

4.1. Função `identificacao()`

A função `identificacao()` busca as ocorrências de um determinado padrão em um *corpus* e gera novos arquivos de transcrição com marcações de ocorrências e novo nome, na mesma pasta em que estão os arquivos originais. Ela possui diversos argumentos, que são descritos a seguir:

`padrao`	Obrigatório. Definição da sequência de caracteres (expressão regular) que identifica as variantes da variável. Deve ser especificado entre aspas;
`simbolo. marcacao`	Opcional. Definição dos símbolos que serão usados para identificar as ocorrências da variável no *corpus*. *Default* = "<>";
`posicao.marcacao`	Obrigatório. Lógico (T ou F). Se T, a marcação é colocada após o padrão. Se F, a marcação é colocada antes do padrão. Default=T. Nota: Para posicao.marcacao=F, a definição da variável deve começar com \\b (levando em conta o início da palavra);
`ignorar.linhas`	Opcional. Vetor com identificação dos participantes, cujas falas devem ser ignoradas na busca do padrão. *Default*=NULL.
`stoplist`	Opcional. Vetor com palavras que devem ser ignoradas na busca do padrão. *Default*=NULL.
`novos.arquivos`	Opcional. Sequência de caracteres a ser adicionada ao nome dos arquivos originais, para diferenciar os arquivos com marcações de ocorrência na mesma pasta. Deve ser especificado entre aspas. *Default*="marcacoes".

Note, primeiramente, que quatro dos argumentos da função `identificacao()` são opcionais: `simbolo.marcacao`, `ignorar.linhas`, `stoplist` e `novos.arquivos`. Se tais argumentos não são especificados, a função utiliza cada valor *default* respectivo.

Essa função requer que se especifique a pasta em que se encontram as transcrições originais, nas quais o padrão será buscado. Como visto em 2.2, isso pode ser feito com a função `setwd()` ou através da aba *Files* do RStudio.

A seguir, discutimos exemplos da aplicação da função na localização de três variáveis: diminutivos, (-r) em coda silábica e pronome de primeira pessoa do plural.

Exemplo 1: diminutivos

O caso mais simples de uso da função `identificacao()` é aquele em que se especifica somente o padrão a ser buscado. Suponha que queremos investigar o uso dos diminutivos no português, em seu caso mais comum: o diminutivo sintético com sufixo "-inh-"[7]. Precisamos, primeiro, definir para o R o que é um diminutivo através de uma expressão regular. Antes de continuar a leitura, imagine como você definiria tal expressão, usando os caracteres especiais vistos em 2.3.

A busca deve incluir palavras terminadas em "-inho", "-inha", "-inhos" ou "-inhas". Como "o" e "a" se alternam, podemos especificar `inh[oa]`; como o morfema –s de plural é opcional, podemos usar o caractere especial ? (zero ou uma ocorrência da expressão regular precedente): `inh[oa]s?`; por fim, queremos que essa sequência esteja no final da palavra: `inh[oa]s?\\b` é o padrão a ser buscado. Inserindo-o na função `identificacao()` (apenas com os argumentos obrigatórios), temos:

(36)

```
>identificacao(padrao="inh[oa]s?\\b", posicao.marcacao=T)¶
```

O tempo para rodar cada função pode variar consideravelmente (de poucos segundos a alguns minutos), a depender principalmente do tamanho do *corpus* e do número de ocorrências do padrão buscado. Enquanto a função está sendo rodada, aparece um símbolo vermelho (STOP) no canto superior direito do console no RStudio. A função terá terminado quando esse símbolo desaparecer e aparecerem os sinais >| no console.

Os novos arquivos de transcrição, com a adição de "marcacoes-" ao nome original, podem ser visualizados na mesma pasta em que estavam os arquivos originais ou na aba *Files* do RStudio. Examinemos os resultados no arquivo marcacoes-SP2012-009-F26SEL-FabianaB.txt no próprio RStudio: clique sobre o nome do arquivo, que abrirá na janela *Source* (dos *scripts*). Em seguida, digite [CTRL]/[Command]+F para abrir a janela de *Localizar/Substituir* (respectivamente, *Find* – o campo com a lupa – e *Replace*), e digite <> no campo *Localizar* (Figura 4). Clique sobre *Next* para visualizar as próximas marcações.

7 Deve-se lembrar que diminutivos também têm a forma analítica (por exemplo, homem pequeno). Para os sufixos sintéticos, Cunha & Cintra (2007) listam 22 sufixos: -inho(a); -zinho(a); -ino(a); -im; -elho(a); -ejo; -ilho(a); -acho(a); -icho(a); -ucho(a); -ebre; -eco(a); -ico(a); -ela; -ete; -eto(a); -ito(a); -zito(a); -ote(a); -isco(a); -usco(a); e -ola. De todos eles, no entanto, o sufixo *–inh-* é o mais produtivo.

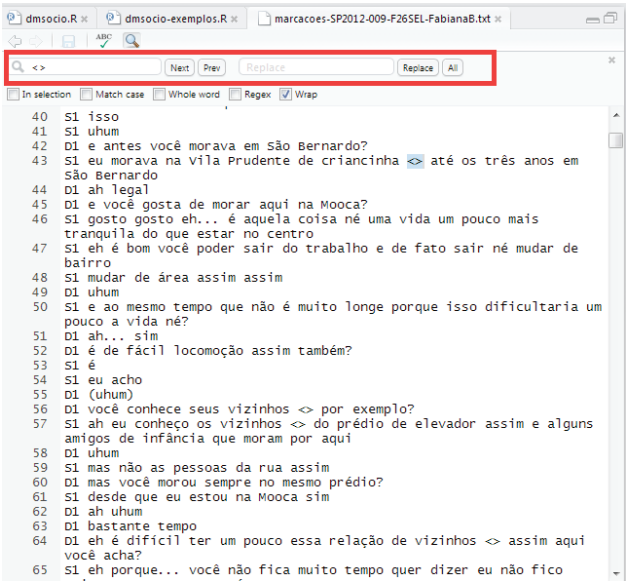


Figura 4 – *Localizar/Substituir* no RStudio

Nesse arquivo, encontramos a marcação de 75 ocorrências: "criancinha" na linha 43, "vizinhos" nas linhas 56, 57, 64, "minha" e "tinha" na linha 76, etc. O R localizou, corretamente, todas as ocorrências de palavras que terminam com "-inho, -inha, -inhos, -inhas". No entanto, deparamo-nos com dois percalços: o primeiro é que o R localizou ocorrências na fala de D1, que é o documentador (por exemplo, nas linhas 56 e 64). Em geral, em estudos sociolinguísticos, estamos interessados em analisar a falar do informante e não nossas próprias falas. Segundo, o R localizou ocorrências de palavras terminadas em -inh- que não são diminutivos, como "vizinhos", "minha" e "tinha".

Vamos lidar com o primeiro deles: o argumento `ignorar.linhas` da função `identificacao()` serve justamente para especificar ao R quais linhas da transcrição devem ser ignoradas na identificação das ocorrências. No *corpus* do Projeto SP2010, o documentador é sempre identificado por "D1", e possíveis outros falantes que participam da gravação são identificados por "S2", "S3" etc. (lembre-se da regra #2: conheça seu *corpus*!). Nas transcrições do *minicorpus*, há entrevistas com até cinco falantes (portanto, até "S5"), além de D1 e S1. Além disso, há linhas separadas para Dados Contextuais e tópicos do Roteiro, que tampouco contêm falas do informante, e as linhas do cabeçalho. O argumento `ignorar.linhas` requer um vetor que especifica quais linhas devem ser ignoradas, o que pode ser realizado com a função `c()`. Em outros *corpora*, é claro, esse vetor seria definido de modo diferente.

(37)

```
>ignorar<-c("D1", "S2", "S3", "S4", "S5", "Dados Contextuais",
"Roteiro","Universidade de", "Faculdade de", "Departamento
 de", "Grupo de", "Projeto SP2010", "(FAPESP)", "Documentador:",
"Informante:", "Arquivos:", "Perfil:", "Sexo/Gênero:", "Idade:",
"Escolaridade:", "Região:", "Zona:", "Geração:", "Renda
 individual:", "Renda familiar:")¶
```

Nosso segundo percalço foi o R localizar ocorrências que não são diminutivos. Trata-se de uma decisão com base em um critério semântico (portanto, não repetitiva, mecânica ou previsível) que o programa, de fato, não poderia fazer por si só, apenas com a definição da expressão regular. Veja também que não é o caso de criar um novo padrão de busca; se instruíssemos o R, por exemplo, a **não** buscar ocorrências que terminem em "minha" ou "tinha", ele deixaria de encontrar possíveis dados como "caminha" (=cama pequena) ou "cartinha", que são diminutivos. Na verdade, o padrão de busca está correto; queremos apenas que o R ignore a ocorrência de certos itens lexicais específicos. Isso pode ser feito através do argumento `stoplist`, um vetor que contém as palavras a serem ignoradas na busca.

Adiante, com a função `extracao()` (seção 4.2), veremos uma maneira mais simples de verificar quais palavras devem entrar na `stoplist` de um modo que não requer a revisão de todas as transcrições. Por ora, fiquemos com as ocorrências apenas da entrevista com FabianaB, para fins de exemplificar a aplicação do argumento `stoplist`. Ao examinar as marcações nessa transcrição, encontramos as seguintes ocorrências de palavras com -inh- que não são diminutivos: "vizinhos" (em que já podemos imaginar que pode haver ocorrências de vizinho, vizinha, vizinhas); "minha", "tinha" e "carinho". Assim, criamos o vetor `pal.stoplist`:

(38)

```
> pal.stoplist<-c("vizinh[oa]s?", "minha", "tinha", "carinho")¶
```

Podemos agora atualizar a função `identificacao()` com os argumentos `ignorar.linhas` e `stoplist`. Antes disso, no entanto, devemos apagar os arquivos "marcacoes-", que já não nos servem mais.

(39)

```
>identificacao(padrao="inh[oa]s?\\b",
        posicao.marcacao=T,
        ignorar.linhas=ignorar,
        stoplist=pal.stoplist)¶
```

Em (39), adicionamos os argumentos `ignorar.linhas` com o vetor `ignorar` e o argumento `stoplist` com o vetor `pal.stoplist` (os argumentos aparecem em diferentes linhas apenas para facilidade de visualização). Ao examinar o arquivo "marcacoes-SP2012-009-F26SEL-FabianaB.txt", vemos que agora as marcações estão corretas: "criancinha" na linha 43, "barzinhos" na linha 78, "senhorzinhos" na linha 133, etc. Os arquivos estão prontos para a codificação da variável dependente, cujas variantes podem ser indicadas dentro dos símbolos <>, de acordo com os códigos estipulados pelo pesquisador[8].

Cabe ainda comentar os demais argumentos da função, que até agora deixamos com a opção *default* ao não especificá-los. O argumento `novos.arquivos` (*default* `"marcacoes"`) pode ser modificado por qualquer outro nome definido pelo usuário, assim como para `simbolo.marcacao` (*default* `"<>"`); para esse último, no entanto, o ideal é que o símbolo escolhido não seja empregado nas normas de transcrição, para que identifique, de modo inequívoco, a ocorrência da variável. Deve-se lembrar, também, que certos caracteres têm significado especial no R (como parênteses, colchetes, chaves, pontuações, etc.); se deseja utilizá-los como símbolo de marcação da variável, devem-se usar os caracteres de escape – por exemplo, para parênteses: `"\\(\\)"`.

O argumento `posicao.marcacao` pode ser definido como antes (=F) ou depois (=T) do padrão buscado. A sua definição como **F** é preferível em padrões que envolvem o início de palavra (diferentemente do caso do diminutivo).

Exemplo 2: (-r) em coda silábica

A variável (-r) em coda silábica exemplifica a aplicação da função `identificacao()` consecutivas vezes, a fim de marcar todas as ocorrências do *corpus*. Trata-se de uma solução possível quando a definição de uma única expressão regular que abarque todas as palavras-alvo se torna muito complicada.

Nesse caso, como na identificação de diminutivos, adotaremos `simbolo.marcacao="<>"`, `posicao.marcacao=T`, `ignorar.linhas=ignorar`, e `novos.arquivos="marcacoes"`; `stoplist`, por sua vez, será mantida como =NULL.

Vejamos inicialmente, como a variável (-r) em coda silábica pode ser definida através de uma expressão regular. Foneticamente, /r/ em coda ocorre após vogais orais, e antes de consoantes (por exemplo, "porta") ou em final de palavra

8 Mendes (2012) codificou as ocorrências de diminutivos do seguinte modo: (i) usos lexicalizados, como *sozinho*, e *barzinho* (que é um tipo específico de bar); (ii) usos metafóricos, como *fumar um cigarrinho* (em que *cigarrinho* não significa um *cigarro pequeno*); e (iii) usos literais, como *a gente vive ali... numa casa bem pequenininha*.

(por exemplo, "mulher"). Ortograficamente, as vogais orais podem ser definidas como uma classe de caracteres [aáâeéêiíoóôuú], e as consoantes como [bcçdfgjklmnpqstvwxz]. Note que na classe das consoantes não se incluem "h" nem "r". Podemos, portanto, definir ocorrências de /r/ em coda seguidas de consoante como:

(40)

```
[aáâeéêiíoóôuú]r[bcçdfgjklmnpqstvwxz]
```

Se desejamos que a marcação da ocorrência apareça após a palavra (por exemplo, "porta <>"), e não após a sequência (por exemplo, "port <>a"), é necessário especificar o fim da palavra \\b como parte da expressão regular, que pode ocorrer zero, um ou mais caracteres ".*?" após a sequência VrC:

(41)

```
[aáâeéêiíoóôuú]r[bcçdfgjklmnpqstvwxz].*?\\b
```

```
>identificacao(padrao="[aáâeéêiíoóôuú]r[bcçdfgjklmnpqstvwxz].*?\\b",
     simbolo.marcacao="<>",
     posicao.marcacao=T,
     ignorar.linhas=ignorar,
     stoplist=NULL,
     novos.arquivos="marcacoes")¶
```

A definição acima, no entanto, tem duas limitações: (i) para palavras com mais de uma ocorrência de /r/ em coda (por exemplo, "supermercado"), ela marca apenas uma ocorrência da variável; e (ii) não localiza ocorrências em final de palavra (como "mulher"), que não têm uma consoante após /r/.

A primeira decorre do modo como o R processa os objetos: ao encontrar um item lexical como "supermercado", o R encontra a primeira ocorrência de /r/ em coda na sequência "erm" (uma vogal, o grafema "r" e uma consoante), e a segunda ocorrência "erc" se encaixa na definição de "zero, um ou mais caracteres seguido(s) da fronteira de palavra". Tais caracteres são exauridos na leitura da sequência, o R insere a marcação <> e segue rastreando as demais ocorrências no restante do *corpus*.

Ao invés de tentar formular uma expressão regular mais complexa, que tenha que encontrar casos com uma ou mais ocorrências, pode-se simplesmente criar uma nova que localizará apenas as palavras com duas ocorrências de /r/ em coda, e rodar a função novamente sobre os *arquivos já marcados* com uma ocorrência por palavra. A ocorrência de dois segmentos de /r/ em coda pode ser definida como:

(42)

```
[aáâeéêiíoóôuú]r[bcçdfgjklmnpqstvwxz].*?[aáâeéêiíoóôuú]r
[bcçdfgjklmnpqstvwxz].*?\\b
```

ou seja, a ocorrência de uma vogal, grafema "r" e uma consoante; zero, um ou mais caracteres; outra vogal, outro grafema "r" e outra consoante; zero, um ou mais caracteres; e a fronteira de palavra. Desse modo, podemos apagar os arquivos originais (tendo feito, claro, uma cópia desses em outra pasta segura) e rodar a função abaixo apenas sobre os arquivos "marcacoes".

(43)

```
>identificacao(padrao="[aáâeéêiíoóôuú]r[bcçdfgjklmnpqstvwxz].*?
      [aáâeéêiíoóôuú]r[bcçdfgjklmnpqstvwxz].*?\\b",
      simbolo.marcacao="<>",
      posicao.marcacao=T,
      ignorar.linhas=ignorar,
      stoplist=NULL,
      novos.arquivos="marcacoes")¶
```

Por fim, solução semelhante pode ser aplicada aos casos de /r/ em final de palavra[9]. O padrão pode ser atualizado como em (44) abaixo – uma sequência de vogal, o grafema *r* e fronteira de palavra – e a função pode ser rodada sobre os arquivos agora chamados "marcacoes-marcacoes-...":

(44)

```
>identificacao(padrao="[aáâeéêiíoóôuú]r\\b",
      simbolo.marcacao="<>",
      posicao.marcacao=T,
      ignorar.linhas=ignorar,
      stoplist=NULL,
      novos.arquivos="marcacoes")¶
```

9 Você pode pensar: mas por que não incluir \\b dentro da classe de consoantes, de modo que *final de palavra* seja uma opção à presença de outros grafemas consonantais como *b*, *c*, *ç*, *etc.*? Essa solução não seria possível pelo fato de \\b não ser um caractere e, portanto, não poder integrar uma classe de caracteres.

Os arquivos finais "marcacoes-marcacoes-marcacoes-...", sobre os quais a função foi rodada três vezes, são aqueles que contêm, corretamente, a marcação de todas as ocorrências de (-r) em coda silábica[10].

Exemplo 3: "nós" vs. "a gente"

O emprego da variável do pronome de primeira pessoa do plural exemplifica o caso de variáveis que permitem a codificação automática a partir da função `identificacao()`. No caso dos diminutivos, a codificação de acordo com critérios estabelecidos pelo pesquisador, como Mendes (2012), depende de uma análise contextual mais detalhada, caso a caso (ver nota 8). Para as ocorrências de -r em coda silábica, é necessário ouvir os arquivos de áudio para determinar se o informante realizou o segmento como tepe, retroflexo, apagamento, etc. (como será o caso para variáveis fonéticas).

No caso da variável "nós" *vs.* "a gente", podemos elaborar duas expressões regulares, uma para cada variante, e rodar a função duas vezes, de modo semelhante ao que foi feito com /r/ em coda. O argumento `simbolo.marcacao` pode então ser definido como `"<N>"` para os casos de "nós" e `"<G>"` para os casos de "a gente".

Imagine que o pesquisador decidiu analisar todos os casos de "nós", não apenas no caso nominativo, mas também em outras funções sintáticas. A expressão regular deve então abarcar as ocorrências de "nós", "nos", "conosco", "nosso", "nossa", "nossos" e "nossas".

(45)

```
nós\\b|\\bnos\\b|conosco|noss[oa]s?\\b
```

```
>identificacao(padrao="nós\\b|\\bnos\\b|conosco|noss[oa]s?\\b",
     simbolo.marcacao="<N>",
     posicao.marcacao=T,
     ignorar.linhas=ignorar,
     stoplist=NULL,
     novos.arquivos="marcacoes")¶
```

Em (45), as ocorrências de "nos" são delimitadas tanto no início quanto no final da palavra por `\\b`, para evitar que a função localize ocorrências como "a<u>nos</u>", "me<u>nos</u>", "<u>nos</u>tálgico", etc. Ao examinar as marcações nos arquivos de

10 Casos de /r/ em final de palavra seguidos de vogal, como "po<u>r i</u>sso" e "conta<u>r u</u>ma estória", possivelmente devem ser descartados. No entanto, é preferível ouvir tais ocorrências antes de descartá-las, para se certificar que não há uma pausa entre a palavra com /r/ e a vogal.

transcrição, no entanto, percebemos que várias ocorrências marcadas não se referem ao pronome de primeira pessoa do plural. São casos de "nos" preposição ("em" + "os", por exemplo, linha 72 de FabianaB: "... nos finais de semana à noite") e de "nossa" como interjeição (por exemplo, linha 359 de FabianaB: "nossa está molhado aqui").

Cabe aqui pensar se é o caso de rever nossa definição da expressão regular, ou de criar uma `stoplist`. Contudo, diferentemente dos casos discutidos para /e/ nasal ou para o diminutivo, aqui se trata de coincidências lexicais: se excluirmos as ocorrências de "nossa" e "nos", excluiremos também os **verdadeiros** casos de primeira pessoa do plural. Nessa situação, é necessário remover as ocorrências "a mais" uma a uma, analisando-as em seus contextos. Isso pode ser feito diretamente na interface do RStudio, na aba *Files*, através da janela *Localizar/Substituir* ([CTRL]/[Command]+F). Em *Localizar*, digite "`<N>` " (com um espaço após >) e não digite nada em substituir. Nos casos de marcação indevida, clique em *Replace* para apagar a marcação.

Vejamos agora os casos de "a gente". Aqui, a definição da expressão regular é mais simples, já que o pronome "a gente" não apresenta múltiplas formas como "nós". A expressão `a\\sgente` (sequência de "a", espaço, "gente") encontrará tanto as ocorrências do pronome isolado quanto aquelas em que se amalgama as preposições (por exemplo, "na gente", "pra gente"), contanto que não se especifique uma fronteira de palavra no início. No entanto, essa definição também localizará ocorrências como "muita gente", "pouca gente", "tanta gente" – é o caso de definir uma `stoplist`. Portanto:

(46)

```
>stoplist.G<-c("muita\\sgente","tanta\\sgente","pouca\\sgente")¶
>identificacao(padrao="a\\sgente",
       simbolo.marcacao="<G>",
       posicao.marcacao=T,
       ignorar.linhas=ignorar,
       stoplist=stoplist.G,
       novos.arquivos="marcacoes")¶
```

Ao rodar o código acima sobre os arquivos já marcados com <N>, os arquivos finais "marcacoes-marcacoes-..." conterão todas as ocorrências de "nós" e "a gente", devidamente codificadas para a variável dependente.

4.2 Função extracao()

A função `extracao()` busca um padrão em um conjunto de transcrições e retorna os resultados na forma de uma tabela que contém a ocorrência, o contexto precedente e o seguinte, e sua localização na transcrição. Opcionalmente, a função também retorna a codificação da variável dependente e de variáveis sociais. Assim como na função `identificacao()`, é necessário primeiro especificar como diretório de trabalho a pasta em que se encontram as transcrições, através da função `setwd()` ou através da aba *Files* (seção 2.2).

A descrição de cada argumento se encontra a seguir.

`padrao` — Obrigatório. Definição da sequência de caracteres a ser buscada. Deve ser especificado entre aspas.

`palavras.cont.precedente` — Opcional. Numérico. Número de palavras do contexto precedente a serem extraídas. *Default*=5.

`palavras.ocorrencia` — Opcional. Numérico. Número de palavras a serem extraídas para a coluna de ocorrência. *Default*=1.

`palavras.cont.seguinte` — Opcional. Numérico. Número de palavras do contexto seguinte a serem extraídas. *Default*=5.

`stoplist` — Opcional. Vetor com palavras que devem ignoradas na busca do padrão. *Default*=NULL.

`ignorar.linhas` — Opcional. Vetor com identificação dos participantes, cujas falas devem ser ignoradas na busca do padrão. *Default*=NULL.

`var.dependente` — Opcional. Numérico. Localização do caractere de codificação dentro do termo que contém o padrão buscado. *Default*=NULL.

`loc.variaveis.sociais` — Opcional. Vetor com o número das linhas em que se encontram as informações das variáveis sociais a serem extraídas. *Default*=NULL.

`nomes.colunas.variaveis` — Opcional. Vetor com o nome das variáveis sociais a serem extraídas, na mesma sequência do vetor com sua localização. *Default*=NULL.

`file` — Opcional. Nome do arquivo com dados extraídos. *Default*="DadosExtraidos.txt". O nome deve conter a extensao .txt e ser especificado entre aspas.

O único argumento obrigatório nesta função é a especificação do padrão a ser buscado, na forma de uma expressão regular. A especificação desse argumento pode ser feita de modo idêntico ao da função `identificacao()`. Tomemos como exemplo a expressão regular para os diminutivos, como definida em (36) acima (deixando todos os demais argumentos como *default*).

(47)

```
> extracao("inh[oa]s?\\b")
```

Os resultados se encontram em um arquivo denominado "DadosExtraidos.txt", que se localiza no atual diretório de trabalho. Como se trata de uma tabela, é preferível examinar esses dados em uma planilha, no programa Excel ou Calc. Abra o arquivo em um desses programas[11]. A Figura 5 abaixo mostra as primeiras linhas de resultados:

	A	B	C	D	E	F
1	Contexto.Precedente	Ocorrencia	Contexto.Seguinte	GFs sociais...	Localizacao	
2	eu morava na Vila Prudente de	criancinha	até os três anos em		Paragrafo: 43	
3	(uhum) D1 você conhece seus	vizinhos	por exemplo? S1 ah		Paragrafo: 56	
4	S1 ah eu conheço os	vizinhos	do prédio de elevador assim		Paragrafo: 57	
5	ter um pouco essa relação de	vizinhos	assim aqui você acha?		Paragrafo: 64	
6	mudado assim do que na na	minha	época da adolescência que tinha		Paragrafo: 76	
7	na minha época da adolescência que	tinha	pouca coisa D1 uhum		Paragrafo: 76	
8	D1 uhum S1 tem muitos	barzinhos	aqui né tem ali uma		Paragrafo: 78	
9	tem ali uma rua próxima à	minha	casa S1 onde... montaram		Paragrafo: 78	
10	não faz muito tempo né porque	tinha	claro sempre S1 ali		Paragrafo: 87	
11	você morava na Vila Prudente é	pertinho	também né? S1 é		Paragrafo: 100	
12	ah isso tem com os mais	senhorzinhos	assim né S1 porque...		Paragrafo: 133	
13	assim né S1 porque... os	senhorzinhos	da Mooca são aqueles italianos		Paragrafo: 134	
14	Mooca que é um grupo de	velhinhos	que vai... senhorzinhos Dados		Paragrafo: 136	
15	um grupo de velhinhos que vai...	senhorzinhos	Dados Contextuais [risos-S1]		Paragrafo: 136	
16	Mooca não S1 o que	tinha	anteriormente eram as festas do		Paragrafo: 143	
17	isso que tem/ são mais... os	velhinhos	então será que se reúnem		Paragrafo: 151	
18	D1 uhum S1 perto da	minha	casa tem até um bar		Paragrafo: 157	
19	de rock S1 onde uns	senhorzinhos	têm um grupo de seresta		Paragrafo: 158	
20	estar aqui S1 que eu	tinha	que acordar cedo pra pegar		Paragrafo: 183	
21	metrô Dados Contextuais [barulho de	latinha	se abrindo] D1 uhn		Paragrafo: 184	
22	com quem? S1 com a	minha	mãe e com a minha		Paragrafo: 224	
23	a minha mãe e com a	minha	irmã D1 ah		Paragrafo: 224	
24	vocês brin/ você brincava na rua	tinha	essa tradição de brincar na		Paragrafo: 229	
25	era uma criança muito ativa eu	tinha	diversas atividades no dia... então		Paragrafo: 233	

DadosExtraidos

Figura 5 – Arquivo "DadosExtraidos.txt" para busca do padrão `inh[oa]s?\\b`.

11 Pode-se clicar com o botão direito sobre o arquivo, escolher *Abrir com...* e selecionar um programa de planilhas. Alternativamente, no Excel ou no Calc, clique em *Arquivo* > *Abrir* e selecione o arquivo "DadosExtraidos.txt" (no Excel, é necessário especificar a busca com a opção *Todos os arquivos* para que a janela exiba o arquivo criado, que está em formato .txt). Uma janela se abrirá, pedindo que o usuário especifique o tipo de campo (Delimitado), a linha pela qual se deve iniciar a importação (1) e a origem do arquivo. Verifique se os caracteres aparecem corretamente na *Visualização* da parte inferior e, se necessário, mude a opção em *Origem de arquivo*. Clique em *Avançar*. Na segunda etapa, selecione *Tabulação* como delimitador e clique novamente em *Avançar*. Na terceira etapa, selecione *Geral* para o formato dos dados e clique em *Concluir*.

A tabela contém cinco colunas: *Contexto.Precedente*, *Ocorrencia*, *Contexto.Seguinte*, *GFs sociais*... (que está vazia por não havermos especificado esse argumento na função), e *Localização* (com indicação do parágrafo em que se localiza a ocorrência em sua respectiva transcrição). As palavras-alvo estão separadas na coluna B: veja que, nessa extração, foram localizadas as palavras "vizinhos", "minha", "tinha" etc. – que não são diminutivos – já que não especificamos a `stoplist` em (47) acima. No total, houve 1.706 ocorrências de palavras terminadas em -inh- no *minicorpus* "dmsocio-SP2010".

Como as ocorrências estão separadas, essa coluna pode nos auxiliar a identificar *todas* as ocorrências de palavras terminadas em -inh- que não interessam em nossa análise (lembre-se que, em (38), havíamos especificado as palavras indesejadas que encontramos apenas na entrevista com FabianaB). Tanto o Excel quanto o Calc possuem ferramentas para remover valores duplicados. No Excel, copie a coluna B para uma coluna vazia (preferencialmente, em uma outra planilha); com os dados selecionados, clique em *Dados* > *Remover Duplicatas*. No Calc, o processo envolve alguns passos a mais: copie a coluna B para uma coluna vazia, e com os dados selecionados clique em *Dados* > *Filtros* > *Filtro padrão*. Na janela que se abrir, na primeira linha, escolha *Não vazio* no campo *Valor*. Clique em *Mais opções*, selecione *Sem duplicação* e *Copiar resultados para* e, em seguida, clique em uma célula vazia da tabela. Clique em *OK*.

Das 1.706 ocorrências, há 261 valores únicos. A partir dessa lista – mais prática do que examinar a lista de 1.706 ocorrências, ou mesmo reler todas as transcrições – o pesquisador pode com facilidade criar a lista completa de palavras que devem constar no `stoplist`. Esse caso exemplifica como a função `extracao()` pode ser empregada na *exploração* de dados do *corpus*, sem o objetivo específico de criar uma planilha de codificação.

O pesquisador também pode empregá-la, por exemplo, para testar uma expressão regular, caso não tenha certeza de que a definição formulada localiza corretamente todas e somente as ocorrências referentes à sua variável. Desse modo, ela pode ser utilizada *antes* de aplicar a função `identificacao()`, para verificar rapidamente que tipos de dados se encaixam na expressão regular.

A função `extracao()` também pode ser usada para extrair dados já identificados da variável sociolinguística – finalidade para a qual foi criada de fato – e evitar o trabalho braçal de copiar e colar centenas ou milhares de ocorrências para uma planilha de codificação. Caso o pesquisador já tenha identificado as ocorrências (seja com a função `identificacao()`, seja manualmente) e as codificado no arquivo de transcrições, o padrão a ser buscado pode ser especificado simplesmente com os símbolos usados para marcação de ocorrências (por exemplo, "<N>|<G>").

Para exemplificar esse caso, vamos utilizar os dados do *minicorpus* "dmsocio-codif-R". Trata-se de outro conjunto de 12 entrevistas sociolinguísticas, que

fazem parte do *corpus* de 118 gravações analisadas por Oushiro (2011), já ouvidas e codificadas quanto à pronúncia de (-r) em coda silábica. Os seguintes códigos foram utilizados:

T	Tepe
R	Aproximante retroflexa
A	Apagamento
H	Aspirada (fricativa velar ou glota, surda ou sonora)
V	Realização intermediária ou duvidosa (inclui trechos com muito ruído de fundo)
M	Dados metalinguísticos (por exemplo, "cariocas falam po[x]ta")
E	Palavras estrangeiras (por exemplo, "Big Brother", "Museu d'Orsay")

Há múltiplas possibilidades de análise: podem-se extrair todos os dados, a fim de verificar a sua distribuição e frequências no *corpus*; podem-se também desconsiderar as ocorrências de dados metalinguísticos (que não se referem a realizações "espontâneas" do falante), de palavras estrangeiras (cuja pronúncia pode ser aportuguesada ou não) e de realizações intermediárias (nas quais o pesquisador ficou em dúvida quanto à codificação); pode-se analisar o apagamento de /r/, em contraposição às suas variantes não-apagadas tepe, retroflexo e aspirada; pode-se analisar apenas a pronúncia de tepe *vs*. retroflexo, que são as variantes mais prototípicas da variedade paulistana e se associam mais fortemente às identidades locais.

Essas decisões, como já mencionado, dependem das questões que norteiam a pesquisa. Se nos decidirmos pelo último caso (tepe *vs*. retroflexo), podemos definir o padrão a ser extraído simplesmente como "<T>|<R>" (ou "<[TR]>"). Examinemos, também, os demais argumentos da função `extracao()`.

Palavras.cont.precedente e *palavras.cont.seguinte* são argumentos numéricos que especificam quantas palavras devem ser extraídas antes e depois da ocorrência. Em geral, para variáveis fonéticas, poucas palavras (digamos, 3 ou 4) bastam para estabelecer um contexto que permita a codificação de outras variáveis linguísticas, como contexto fônico precedente, contexto fônico seguinte, classe morfológica, etc. Para variáveis morfológicas, sintáticas e discursivas, normalmente é preferível a extração de um contexto textual maior (digamos, 15 ou 20 palavras). Para -r em coda, vamos estabelecer 4 palavras.

O número de palavras da coluna *Ocorrencia* pode conter apenas o padrão buscado (neste caso, "<T>" ou "<R>"*)*, ou incluir um número maior de palavras

antecedentes. No presente exemplo, é interessante extrair a codificação da ocorrência junto com o dado em si (a palavra com /r/ em coda), de modo que vamos estabelecer `palavras.ocorrencia=2`.

Os argumentos *stoplist* e *ignorar.linhas* funcionam de forma idêntica à sua aplicação na função `identificacao()`. Neste caso, ambos podem ser =*NULL*: não há palavras que devem ser evitadas e as marcações de ocorrência foram feitas apenas na fala do informante.

O argumento *var.dependente* indica, numericamente, a localização do caractere de codificação da variável dependente dentro do *padrão* estabelecido. Neste exemplo, o código da variante empregada pelo falante em cada ocorrência se encontra na segunda posição de `<T>` e `<R>`; estabelecemos, portanto, `var.dependente=2`.

O argumento *loc.variaveis.sociais* pressupõe a existência de um cabeçalho em cada transcrição. Ele é preenchido com um vetor que indica a localização de informações da ficha social do informante (por exemplo, sexo/gênero, faixa etária, etc.) de acordo com a linha em que aparecem. O cabeçalho das transcrições do *minicorpus* "dmsocio-codif-R" indica as seguintes informações:

`#cab`	
`2009`	Ano de gravação
`F`	Sexo: F – feminino; M - masculino
`27`	Idade em anos
`1`	Faixa etária: 1 – 20 a 34; 2 – 35 a 59; 3 – 60 ou mais
`C`	Escolaridade: C – até Ens. Médio; S – Ens. Superior
`P`	Região de residência: C – bairro central; P – bairro periférico
`L`	Zona de residência: N – norte; S – sul; L – leste; O – oeste; C – centro
`0`	Geração na cidade: 0 – pais não paulistanos; 1 – mãe ou pai paulistano; 2 – mãe e pai paulistano; 3 – um(a) avô/avó paulistano; 4 – dois ou mais avós paulistanos
`I`	Origem dos pais: P – São Paulo-capital; I – interior SP/MG; N – região N/NE; E – estrangeira; X – mista
`Z`	Mobilidade: B – sempre morou no mesmo bairro; Z – sempre morou na mesma zona; M – já morou em diferentes zonas ou outra cidade
`PamelaR`	Pseudônimo do informante
`Artur Alvim`	Bairro de residência
`D1: Livia Oushiro`	Documentador
`Duração total: 01h10min32seg`	
`Início: 00h00min00seg`	
`Fim: 01h00min00seg`	
`Comentários:`	

O cabeçalho, é claro, pode diferir de *corpus* para *corpus*. Se foi digitado de modo consistente em todas as transcrições, é possível indicar em um vetor a localização das variáveis sociais relevantes que o pesquisador deseja incluir em sua análise. O vetor `var.sociais` abaixo indica a localização das variáveis sexo/gênero, faixa etária, escolaridade, região de residência e pseudônimo do informante que se encontram, respectivamente, nas linhas 3, 5, 6, 7 e 12 dos cabeçalhos.

(48)

```
> var.sociais<-c(3, 5, 6, 7, 12) ¶
```

O argumento *nomes.colunas.variaveis*, por sua vez, especifica o nome que deve ser atribuído às variáveis sociais, na mesma ordem em que foram especificadas para *var.sociais*.[12] Neste caso:

(49)

```
> nomes.var.sociais<-c("sexo.genero", "faixa.etaria",
          "escolaridade", "regiao.residencia",
          "informante")¶
```

Por fim, o argumento *file* especifica o nome do arquivo com a tabela de dados extraídos. O *default* "DadosExtraidos.txt" pode ser modificado pelo usuário, atentando-se ao requisito de que o nome deve vir entre aspas e com a extensão .txt. Aqui, usaremos o nome "DadosExtraidos-RT.txt".

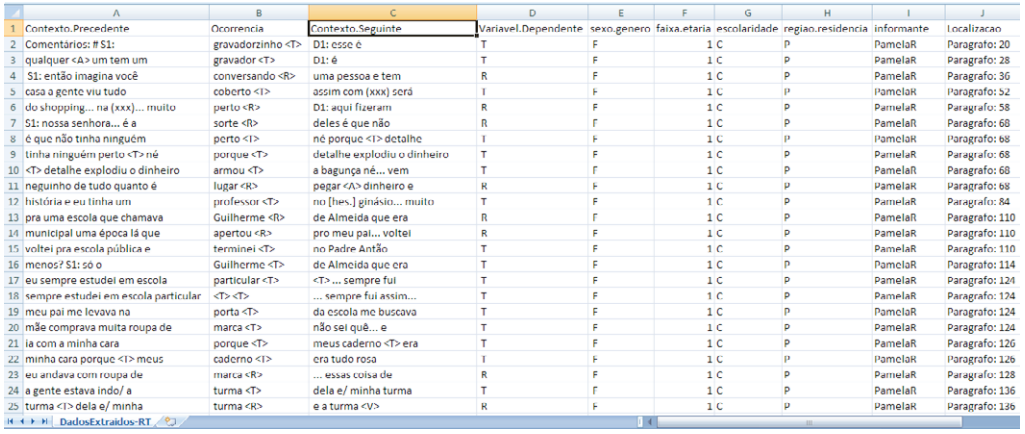
Desse modo, temos então:

(50)

```
> extracao(padrao="<T>|<R>",
     palavras.cont.precedente=4,
     palavras.ocorrencia=2,
     palavras.cont.seguinte=4,
     stoplist=NULL,
     ignorar.linhas=NULL,
     var.dependente=2,
     loc.variaveis.sociais=var.sociais,
     nomes.colunas.variaveis=nomes.var.sociais,
     file="DadosExtraidos-RT.txt")
```

12 É possível especificar somente um vetor para `loc.variaveis.sociais` e deixar `nomes.colunas.variaveis` como =NULL, caso em que as variáveis serão nomeadas "Var.1", "Var.2", etc. No entanto, se os nomes das variáveis forem especificados, não é possível deixar o argumento `loc.variaveis.sociais` como =NULL, já que o script demandará a localização das variáveis.

O resultado é uma tabela, cujas primeiras linhas podem ser visualizadas na Figura 6. Em alguns minutos, o pesquisador tem em mão uma tabela de dados semipronta para ser analisada em programas como GoldVarb X ou RBrul, algo que poderia levar horas ou dias se fosse criada manualmente.



	A	B	C	D	E	F	G	H	I	J
1	Contexto.Precedente	Ocorrencia	Contexto.Seguinte	Variavel.Dependente	sexo.genero	faixa.etaria	escolaridade	regiao.residencia	informante	Localizacao
2	Comentários: # S1:	gravadorzinho <T>	D1: esse é	T	F	1	C	P	PamelaR	Paragrafo: 20
3	qualquer <A> um tem um	gravador <T>	D1: é	T	F	1	C	P	PamelaR	Paragrafo: 28
4	S1: então imagina você	conversando <R>	uma pessoa e tem	R	F	1	C	P	PamelaR	Paragrafo: 36
5	casa a gente viu tudo	coberto <I>	assim com (xxx) será	I	F	1	C	P	PamelaR	Paragrafo: 52
6	do shopping... na (xxx)... muito	perto <R>	D1: aqui fizeram	R	F	1	C	P	PamelaR	Paragrafo: 58
7	S1: nossa senhora... é a	sorte <R>	deles é que não	R	F	1	C	P	PamelaR	Paragrafo: 68
8	é que não tinha ninguém	perto <I>	né porque <I> detalhe	I	F	1	C	P	PamelaR	Paragrafo: 68
9	tinha ninguém perto <T> né	porque <T>	detalhe explodiu o dinheiro	T	F	1	C	P	PamelaR	Paragrafo: 68
10	<T> detalhe explodiu o dinheiro	armou <T>	a bagunça né... vem	T	F	1	C	P	PamelaR	Paragrafo: 68
11	neguinho de tudo quanto é	lugar <R>	pegar <A> dinheiro e	R	F	1	C	P	PamelaR	Paragrafo: 68
12	história e eu tinha um	professor <T>	no [hes.] ginásio... muito	T	F	1	C	P	PamelaR	Paragrafo: 84
13	pra uma escola que chamava	Guilherme <R>	de Almeida que era	R	F	1	C	P	PamelaR	Paragrafo: 110
14	municipal uma época lá que	apertou <R>	pro meu pai... voltei	R	F	1	C	P	PamelaR	Paragrafo: 110
15	voltei pra escola pública e	terminei <T>	no Padre Antão	T	F	1	C	P	PamelaR	Paragrafo: 110
16	menos? S1: só o	Guilherme <T>	de Almeida que era	T	F	1	C	P	PamelaR	Paragrafo: 114
17	eu sempre estudei em escola	particular <T>	<T> ... sempre fui	T	F	1	C	P	PamelaR	Paragrafo: 124
18	sempre estudei em escola particular	<T> <T>	... sempre fui assim...	T	F	1	C	P	PamelaR	Paragrafo: 124
19	meu pai me levava na	porta <T>	da escola me buscava	T	F	1	C	P	PamelaR	Paragrafo: 124
20	mãe comprava muita roupa de	marca <T>	não sei quê... e	T	F	1	C	P	PamelaR	Paragrafo: 124
21	ia com a minha cara	porque <T>	meus caderno <T> era	T	F	1	C	P	PamelaR	Paragrafo: 126
22	minha cara porque <I> meus	caderno <I>	era tudo rosa	I	F	1	C	P	PamelaR	Paragrafo: 126
23	eu andava com roupa de	marca <R>	... essas coisa de	R	F	1	C	P	PamelaR	Paragrafo: 128
24	a gente estava indo/ a	turma <T>	dela e/ minha turma	T	F	1	C	P	PamelaR	Paragrafo: 136
25	turma <I> dela e/ minha	turma <R>	e a turma <V>	R	F	1	C	P	PamelaR	Paragrafo: 136

DadosExtraidos-RT

Figura 6 – Dados de tepe e retroflexo extraídos da amostra "dmsocio-codif-R".

4.3 Função `amostragem()`

A função `amostragem()` seleciona aleatoriamente um determinado número de dados, com base em uma coluna de referência – por exemplo, 50 dados por informante. A tabela "DadosExtraidos-RT.txt" revela que houve um total de 3.282 dados de /r/ tepe ou retroflexo na pequena amostra de 12 entrevistas sociolinguísticas – uma média de 274 dados por falante. Tal costuma ser o caso para variáveis fonéticas, em que não raro se obtêm milhares ou dezenas de milhares de ocorrências da variável sociolinguística. De fato, em sua amostra de 118 informantes paulistanos, Oushiro (2012) obteve mais de 70 mil dados de /r/ em coda silábica e cerca de 35 mil para as variantes tepe e retroflexa.

Em estudos sociolinguísticos de variáveis fonéticas, nem sempre é necessário trabalhar com todos os dados (WOLFRAM, 1993). De um lado prático, uma subamostra torna a tarefa de codificar as variáveis independentes (em nosso caso, apenas as linguísticas, já que as variáveis sociais já estão codificadas) mais manejável. Mais importante, do lado teórico-metodológico, uma subamostra – contanto que aleatória – costuma revelar os mesmos padrões de variação e permite controlar o peso que cada informante tem nos resultados finais, se cada um

pelo mesmo número de dados. Ademais, é preferível analisar um menor número de dados de uma maior quantidade de informantes, do que analisar todos de poucos informantes – a amostra com maior quantidade de informantes tenderá a revelar padrões que mais se aproximam daqueles empregados na comunidade.

É importante frisar, no entanto, que a aplicação da função `amostragem()` é recomendada apenas em casos de variáveis sociolinguísticas muito frequentes, cuja distribuição é razoavelmente balanceada na amostra. Em outros casos, é preferível analisar o conjunto completo de dados.

Essa função requer a instalação de um pacote adicional, chamado NCStats (OGLE, s/d), que não consta na instalação base do R. A própria função `amostragem()` verifica se o pacote já está instalado no computador e, em caso negativo – como provavelmente será na primeira vez que a função for empregada –, busca-o na Internet e faz sua instalação. Se o computador não estiver conectado à internet, aparecerá uma mensagem de erro avisando que não existe o pacote NCStats. A conexão à internet, portanto, é requerida, mas apenas na primeira utilização da função. Nas próximas vezes, o programa já estará instalado.

A função contém quatro argumentos, descritos e exemplificados a seguir:

`numero.dados` Opcional. Numérico. Número de dados a serem selecionados aleatoriamente da coluna de referência. *Default* = 50.

`coluna.referencia` Obrigatório. Numérico. Coluna que deve ser usada como referência para amostragem. A = 1, B = 2, C = 3, etc.

`data` Obrigatório. *Dataframe* com planilha de dados a serem amostrados.

`novo.arquivo` Opcional. Nome do novo arquivo com os dados amostrados. *Default*="DadosAmostrados.txt"

Suponha que, do conjunto total de 3.282 dados de tepes e retroflexos, da amostra "dmsocio-codif-R", o pesquisador decida extrair 100 dados aleatórios por falante, identificados na coluna I (=coluna 9 da esquerda para a direita) da tabela "DadosExtraidos-RT". Primeiro, devem-se carregar os dados em um *dataframe*, através da função `read.table()`. Se se trata do mesmo arquivo gerado com a função `extracao()`, seus argumentos serão:

(51)

```
>dadosR<-read.table(file=choose.files(), header=T, sep="\t",
quote="", comment.char="")¶
```

A função `amostragem()` pode então ser definida como

(52)

```
> amostragem(numero.dados=100,
      coluna.referencia=9,
      data=dadosR,
      novo.arquivo="DadosAmostrados-RT.txt")¶
```

O arquivo "DadosAmostrados-RT.txt", com 1.200 dados (100 dados x 12 falantes) é criado no atual diretório de trabalho.

CONSIDERAÇÕES FINAIS

Este texto é uma introdução prática ao uso do programa R como ferramenta para análises sociolinguísticas. Além de apresentar alguns conceitos e funções básicas, demonstrou-se a aplicação de três funções desenvolvidas especificamente para certas tarefas de análise sociolinguística, que compreendem a preparação do arquivo de dados: `identificacao()`, `extracao()` e `amostragem()`. Os dois primeiros, em especial, também podem ser empregados como métodos de exploração de dados do *corpus*, na fase de análise qualitativa.

O principal objetivo dessas funções é automatizar a realização de tarefas mecânicas e repetitivas, que não constituem a verdadeira tarefa do sociolinguista; ao reduzir o tempo empregado nessas tarefas, o pesquisador pode se dedicar àquilo que de fato constitui o seu papel: à descrição e à análise da heterogeneidade ordenada. O pesquisador terá mais tempo para realizar análises estatísticas, interpretar os resultados e refinar suas análises.

O programa R também pode ser utilizado na realização de diferentes tipos de análises estatísticas, tópico que está além do escopo deste artigo. No entanto, espera-se que, ao se familiarizar com o programa e com as possibilidades de tratamento de dados que ele oferece, o leitor se encoraje a aprofundar seus conhecimentos sobre o R. Ao final do artigo, encontra-se uma pequena lista de referências selecionadas para esse fim.

Boas análises!

LEITURAS RECOMENDADAS

- Para saber mais sobre cada função, digite "?nomedafunção" no console do R. Por exemplo, ?scan. A descrição da função, de seus argumentos e de seus usos aparece na aba *Help* do RStudio;

- GRIES, S. Th. *Quantitative Corpus Linguistics with R*. A practical introduction. New York/London: Routledge, 2009a.

  Trata-se de um livro prático e didático sobre o uso do R para processamento de dados textuais. A obra se volta, principalmente, para a Linguística de *Corpus*, mas exemplifica muitos usos que são de grande interesse para a Sociolinguística. O pacote dmsocio foi desenvolvido com base em seu conteúdo.
- BAAYEN, R. H. *Analyzing Linguistic Data*: a practical introduction to statistics using R. Cambridge: Cambridge University Press, 2008;
- DALGAARD, P. *Introductory statistics with R*. New York: Springer, 2008;
- GRIES, S. Th. *Statistics for Linguistics with R*. Berlin/New York: Mouton de Gruyter, 2009b.

  Após a preparação do arquivo de dados, o próximo passo – e mais importante – é a análise propriamente dita. Essas obras introduzem o leitor a diversos tipos de análises estatísticas, para além dos programas GoldVarb X e RBrul. Dalgaard (2008) é para um público geral e explica diversos conceitos básicos de estatística. Baayen (2008) e Gries (2009b) se voltam especificamente ao público de linguistas.

AGRADECIMENTO

Agradeço ao Grupo de Estudos do R da USP, sobretudo a Fernanda Canever, pelos constantes diálogos em nossa aprendizagem sobre o programa e por testar as funções em outro sistema operacional.

REFERÊNCIAS

BAAYEN, R. H. *Analyzing linguistic data*: a practical introduction to statistics using R. Cambridge: Cambridge University Press, 2008.

BATTISTI, E. *Elevação das vogais médias pretônicas em sílaba inicial de vocábulo na fala gaúcha*. Porto Alegre, 1993. 125f. Dissertação (Mestrado). Universidade Federal do Rio Grande do Sul.

_____. A redução dos ditongos nasais átonos. In: BISOL, L.; BRESCANCINI, C. (eds.), *Fonologia e variação: recortes do português brasileiro*. Porto Alegre: EdiPUCRS, 2002.

BAYLEY, R. The quantitative paradigm. In: CHAMBERS, J.K.; TRUDGILL, P.; SCHILLING-ESTES, N. (eds.), *The Handbook of Language Variation and Change*, p. 117-141. Malden, MA: Blackwell, 2002.

CASTILHO, A.; PRETI, D. (eds.) *A linguagem falada culta na cidade de São Paulo*: materiais para seu estudo, vol. I – Elocuções Formais. São Paulo: T.A. Queiroz, 1986.

CELIA, G. F. *As vogais médias pretônicas na fala culta de Nova Venécia*. Campinas, 2004. 114f. Dissertação (Mestrado). IEL/Unicamp.

CUNHA, C.; CINTRA, L. *A Nova gramática do português contemporâneo*. 3ª edição revista. Rio de Janeiro: Lexikon Informática, 2007.

DALGAARD, P. *Introductory statistics with R*. New York: Springer, 2008.

GRIES, S. Th. *Quantitative corpus linguistics with R*: a practical introduction. New York/ London: Routledge, 2009a.

_____. *Statistics for linguistics with R*. Berlin/New York: Mouton de Gruyter, 2009b.

GUY, G. R. The quantitative analysis of linguistic variation. In: PRESTON, D. (ed.), *American Dialect Research*, p. 223-249. Amsterdam: Benjamins, 1993.

_____. Linguistic variation in brazilian portuguese: aspects of the phonology, syntax and language history.Pennsylvania, 1981. 406f. Tese (Doutorado). University of Pennsylvania.

LABOV, W. (1969). Contraction, deletion, and inherent variability of the English copula. *Language*, vol. 45(4): 715-762, 1969.

MELLO, H.; RASO, T. Para a transcrição da fala espontânea: o caso do C-ORAL-BRASIL. *Revista Portuguesa de Humanidades*, Portugal, v.13, n. 1, p. 301-325, 2009.

MENDES, R. B. Gênero/sexo, variação linguística e intolerância. In: BARROS, D. L. P. (ed.): *Preconceito e intolerância:* Reflexões linguístico-discursivas. São Paulo: Editora do Mackenzie, p. 171-192, 2011.

_____. Diminutivos como marcadores de sexo/gênero. *Revista Linguística*, *v*.8, n.1, p.113-124, 2012.

MENDES, R. B.; OUSHIRO, L. Documentação do Projeto SP2010 – Construção de uma amostra da fala paulistana, 2013. Disponível em <http://projetosp2010.fflch.usp.br/producao-bibliografica>. Acessado em: 01 maio 2014.

OGLE, D. H. NCStats package, v. 0.4.0. Disponível em <https://rforge.net/NCStats/>. acessado em: 01 maio 2014.

OUSHIRO, L. Relatório científico parcial apresentado à FAPESP. (Projeto: Identidade na pluralidade: produção e percepção linguística na cidade de São Paulo, Processo no. 2011/09122-6), 2012.

_____. Ditongação do /e/ nasal no português paulistano. In: SEMINÁRIO DO GEL, 61., 2013, São Paulo. Programação – 61º Seminário do GEL; 2013. v. 1.

OUSHIRO, L.; MENDES, R. B. A pronúncia de /r/ em coda silábica no português paulistano. *Revista do* GEL, São Paulo, v. 8, n. 2, p. 66-95, 2013.

PAIVA, M. C.; SCHERRE, M. M. P. Retrospectiva sociolinguística: contribuições do PEUL. DELTA [*online*], vol.15, n.spe, p. 201-232, 1999.

R CORE TEAM (2013). *R: A language and environment for statistical computing*. R Foundation for Statistical Computing, Vienna, Austria. Disponível em: <http://www.R-project.org/>. Acessado em: 01 maio 2014.

SCHWINDT, L. C.; SILVA, T. B. da. Panorama da redução da nasalidade em ditongos átonos finais do português do sul do Brasil. In: BISOL, L.; COLLISCHONN, G. (eds.). *Português do Sul do Brasil*: variação fonológica, Porto Alegre: EdiPUCRS, p. 13-33, 2009.

TENANI, L.; GONÇALVES, S. C. L. *Manual do sistema de transcrição de dados* (v.5) – Projeto ALIP (Amostra Linguística do Interior Paulista). Ms, s/d.

TENANI, L.; SILVEIRA, A. A. M. O alçamento das vogais médias na variedade culta do noroeste paulista. *Alfa: Revista de Linguística*, São Paulo: v. 52, n. 2, p.447-464, 2008.

VIEGAS, M. C. (1987). Alçamento das vogais médias pretônicas: uma abordagem sociolinguística. Minas Gerais, 1987. Dissertação (Mestrado). Universidade Federal de Minas Gerais.

VOTRE, S. J. *Aspectos da variação fonológica na fala do Rio de Janeiro*. Rio de Janeiro: Pontifícia Universidade Católica do Rio de Janeiro, 1978.

WEINREICH, U.; LABOV, W.; HERZOG, M. I. Empirical foundations for a theory of language change. In: LEHMANN, W.P.; MALKIEL, Y. (eds.). *Directions for Historical Linguistics*: A Symposium. Austin: University of Texas Press, 1968.

WOLFRAM, W. Identifying and interpreting variables. In: PRESTON, D. R. (ed.), *American Dialect Research*, Amsterdam/Philadelphia: John Benjamins, p. 193-221, 1993.

ZILLES, A. The development of a new pronoun: The linguistic and social embedding of a gente in Brazilian Portuguese. *Language Variation and Change*, vol. 17, p. 19-53, 2005.