学習者コーパス論で R

（2019 年度版 sugiura@nagoya-u. jp）

■2019-11-01

1）学習者コーパス NICER の概要： ICLE との比較

2）NICER1.1 のダウンロード

http://sgr.gsid.nagoya-u.ac.jp/wordpress/?page_id=1138

より NICER1_1.zip をダウンロードしてください。

マウス右ボタンクリックで、「すべて展開」

以下のファイルとフォルダーが入っている。

サブコーパス名	（説明）
NICER_NNS	（学習者コーパス。349 ファイル）
NICER_NS	（母語話者コーパス。71 ファイル）

関連ファイル	（説明）
NICER1_0readme_2019-04-04.txt	（概要の説明）
Learner_Profile_List.xls	（学習者情報一覧）
Learner_Instructions.pdf	（学習者用指示文）
Learner_Questionnaier.pdf	（学習者用質問事項）
Native_Profile_List.xls	（母語話者情報一覧）
Native_Instructions.pdf	（母語話者用指示文）
Native_Questionnaire.pdf	（母語話者向け質問事項）

R のダウンロード

https://cran.r-project.org/

Download R for Windows

R の起動と終了

起動：「スタート」から選ぶ

終了：「メニュー」の「ファイル」から「終了」を選ぶか、コマンド　q()
　　（作業スペースの保存　＝＞　パソコン室での作業の場合、自分の USB メモリーに）

R による簡単な計算

コンソールでコマンド
　「R Console」という窓
　プロンプトの右にコマンド（命令）を書く

（注）#はコメント

```
+       # 足す
-       # 引く
*       # かける
/       # 割る
^       # べき乗
sqrt(x)       #ルート（1/2 乗）
log(x)        #x の自然対数を取る。底は e
log2(x)       #x の対数を取る。底は 2
```

●計算例

```
> 1 + 2
[1] 3
> 4 -3
[1] 1
> 5 * 6
[1] 30
> 8 / 2
[1] 4
> 2^3
[1] 8
> sqrt(9)
[1] 3
> log(2)
[1] 0.6931472
> log2(8)
[1] 3
```

変数の作成と値の代入

変数という入れ物に値を代入する。
#変数名は、英数字と_. が使える。R のコマンドと重複しないように。

変数

数字と文字
文字はダブルクオートに入れる

　　<- で入れる

```
> abc <- 3          # abc という変数を作り、3 を代入。
```

```
> abc                 # abc の中身を表示させる。
[1] 3                 # 3 が入っている。

> abc <- 300          # abc の値に 300 が上書きされる。
> abc                 # abc の中身を表示。
[1] 300               #中身が 300 に変わった。

> efj <- 6

> jke <- 2987

> ls()                # これまでに作った変数一覧を表示。
[1] "abc" "efj" "jke"

> rm(jke)             #変数 jke を削除。

> ls() [1] "abc" "efj"
```

文字と数字を区別する

```
> namae <- "sugiura"    # namae には sugiura という文字列が入る。

> class(abc)            # class(変数名)で、変数に入っているものが数字なら numeric を、
                        # 文字なら character を返す。
[1] "numeric"

> class(namae)
[1] "character"
```

配列(R では「ベクトル」と呼ぶ)

配列＝入れ物が複数ならんでいる変数
　　<- c(, , ,) で入れる

```
> kazu <- c(2,4,6,8)
                        # kazu という配列を作り、1 つ目の要素に 2、2 つ目の要素に 4
                        # 3 つ目に 6, 4 つ目に 8 を代入。
> kazu                  # kazu の中身を表示
[1] 2 4 6 8

> names <- c("I","you","he") # 配列の中身は文字列でもいい。
> names[1] "I" "you" "he"

> length(names)         #length(配列名)で、該当配列の要素数を返す。
[1] 3                   # 配列 names の要素は、 "I" "you" "he"の 3 つ。
```

変数や配列に何があるか調べる
```
> ls()
```

要素がいくつ入っているか調べる
　　length(配列名)

```
> length(names)
```

作業スペース・履歴の保存と読み込み

作業スペースをUSBメモリーに保存しておくと、作業内容をとっておける。

#　作業を中断する場合、作業スペースと履歴を保存しておくと、作成した変数や使用したコマンドが保存されるので便利。

作業スペースの保存（「ワークスペース」と呼ばれることもある）

1. Rのメニューバーより、「ファイル」＞「作業スペースの保存」を選択
2. 保存したい場所を指定し、ファイル名を付ける。この時、拡張子を.RData

履歴の保存

1. Rのメニューバーより、「ファイル」＞「履歴の保存」を選択
2. 保存したい場所を指定し、ファイル名を付ける。拡張子を.Rhistory

作業スペースの読み込み

1. Rのメニューバーより、「ファイル」＞「作業スペースの読み込み」を選択
2. 読み込みたいファイル名を選択し、「開く」をクリック
3. もしくは、読み込みたいファイルを保存したフォルダを開き、該当ファイルをダブルクリック。

履歴の読み込み

1. Rのメニューバーより、「ファイル」＞「履歴の読み込み」を選択
2. 読み込みたいファイル名を選択し、「開く」をクリック

作業したコンソール画面上の記録の保存

1. Rのメニューバーより、「ファイル」＞「ファイルを保存」を選択
2. 保存したい場所を指定し、ファイルを保存する。
   デフォルトでは、lastsave.txt というファイル名。日付をつけて保存すると便利。

ファイルとして保存されているデータの読み込み

```
scan(file="ファイルの場所と名前", what="char")
      # どこのフォルダーの何という名前のファイルか （後で作業ディレクトリの話）
      # 文字データなので、what="char"を指定
```

・ウインドを開いて、GUI でファイルを選ぶには、（MacOS は、file.choose() ）

```
> scan(choose.files(), what="char")
```

・例：母語話者ファイル（NS501.txt）を読み込んでみる。

```
(出力例)
Read 853 items
  [1] "@Begin"                    "@Participants:"
  [3] "NS501"                     "@PID:"
  [5] "PIDNS501"                  "@Age:"
  [7] "27"                        "@Sex:"
  [9] "M"                         "@L1:"
 [11] "AmE"                       "@FatherL1:"
 [13] "none"                      "@MotherL1:"
 (以下省略)
```

セパレーターの話

データの単位を何で区切るか
デフォルトは　スペース

行単位で読み込むには、セパレーターを指定する

```
      sep="¥n"
```

```
> scan(choose.files(), what="char", sep="¥n")    #セパレータを改行マーク(¥n)に変更。
                                                 #デフォルトではスペース。
(出力例)
Read 104 items
  [1] "@Begin"
  [2] "@Participants:¥tNS501"
  [3] "@PID:¥tPIDNS501"
  [4] "@Age:¥t27"
  [5] "@Sex:¥tM"
  [6] "@L1:¥tAmE"
  [7] "@FatherL1:¥tnone"
  [8] "@MotherL1:¥tnone"
  [9] "@AcademicBackground:¥tM1"
 (以下省略)
```

読み込んだファイルを R の中にデータとして保存する

母語話者データのファイル　NS501.txt を読み込んで R の中に変数（配列）として保存。

配列名「ns501」という名前にすることにする

```
> ns501 <- scan(choose.files(), what="char", sep ="¥n")
```

「作業ディレクトリ」とファイルの一覧

「作業ディレクトリ」の移動

「ファイル」＞「ディレクトリの変更」
（「File」　＞「Change dir」）

PC のファイルシステム内の「どこにいるか」（どこのフォルダー（ディレクトリ）内か）を確認

```
> getwd()
[1] "D:/.../NICER1_1_1/NICER_NNS"
>
```

ファイル・フォルダの一覧表示

```
> list.files()
  [1] "JPN501.txt" "JPN502.txt" "JPN503.txt" "JPN504.txt"
  [5] "JPN505.txt" "JPN506.txt" "JPN507.txt" "JPN508.txt"
  [9] "JPN509.txt" "JPN510.txt" "JPN511.txt" "JPN512.txt"
 [13] "JPN513.txt" "JPN514.txt" "JPN515.txt" "JPN516.txt"
```

フォルダの中への移動（＝作業ディレクトリーの移動）

```
> setwd()

> setwd("NICER_NNS")        ＃引用符に入れる点に注意

> setwd("..")               ＃一つ上へ移動

> setwd("../NICER_NS")      ＃一つ上へ行ってその下にある NICE-NS へ移動
```

ファイルの読み込み

★「choose.files()」の代わりに、ファイル名を直接明記する

```
> scan("JPN501.txt", what="char")

> scan("JPN501.txt", what="char", sep="¥n")
```

R の中にデータとして読み込む

```
> jpn501 <- scan("JPN501.txt", what="char", sep="¥n")
```

R エディタ

「ファイル」＞「新しいスクリプト」
　　で、「R エディタ」を開いて、そこに命令をコピペ&編集

■作業■

命令をコピペして、ファイル名を修正し、母語話者データ、学習者データ、それぞれ 10 ファイルずつ、読み込んで R の中に配列として保存してください。

NICER データのフォーマット

現在の NICER のデータは、CHAT フォーマットになっている。

　　http://childes.psy.cmu.edu/

　　「CHAT Transcription Manual」
　　『今日から使える発話データベース CHILDES 入門』
　　宮田 Susanne 編　Brian MacWhinney 監修　　ひつじ書房　2004 年 11 月

・CHAT フォーマットでは、一つのファイルに、「ヘッダー情報」と「本文情報」が入っている。
　ヘッダー部分は、行頭が @ で始まる。
　本文部分は、　　行頭が * で始まる。

```
@Begin
@Languages: en
@Participants: CHI Ross Child, FAT Brian Father
@ID: en|macwhinney|CHI|2;10.10|||Target_Child||
@ID: en|macwhinney|FAT|35;2.|||Target_Child||
*ROS: why isn't Mommy coming?
%com: Mother usually picks Ross up around 4 PM.
*FAT: don't worry.
*FAT: she'll be here soon.
*CHI: good.
@End
```

MacWhinney, B. (2000). The CHILDES Project: Tools for Analyzing Talk. 3rd Edition. Mahwah, NJ: Lawrence Erlbaum Associates

---

文字列の検索

　　grep("検索文字列", 変数名)

```
> grep("However", ns501)
[1] 44 53 83
```

要素番号（＝行番号）が表示される。

●grep で、要素番号でなく、中身そのものを出力するオプション

　　value=T

```
> grep("However", ns501, value=T)
[1] "*NS501:¥tHowever in the French educational system instead of a head or a body
[2] "*NS501:¥tHowever what the French lose in logical flow they gain in critical
[3] "*NS501:¥tHowever, sadly with the continuous failings of the American educational
```

```
> grep("however", ns501, value=T)
[1] "*NS501:¥tThis makes the facts easy to access, however, it does not force the writer
```

★別のやり方
●該当する行のデータを取り出す
ns501[grep("However", ns501)]

★grep で正規表現を使う
(R では正規表現「*」の「エスケープ」に「¥」を二重に使う点に注意)

http://stat.biopapyrus.net/r/regex.html

```
> grep("[hH]owever", ns501, value=T)
[1] "*NS501:¥tHowever in the French educational system instead of a head or a body there
[2] "*NS501:¥tThis makes the facts easy to access, however, it does not force the writer
[3] "*NS501:¥tHowever what the French lose in logical flow they gain in critical thinking."
[4] "*NS501:¥tHowever, sadly with the continuous failings of the American
```

もしくは、grep の大文字小文字区別しないオプションを付ける

```
> grep("however", ns501, value=T, ignore.case=T)
```

■課題■

学習者と母語話者が使っている表現を検索して比べてみよう。

■2019-11-08

作業を始める前に、、、
1）ディレクトリの変更
2）作業スペースの読み込み
3）履歴の読み込み

4）今どこにいるか　getwd()
5）そこにどんなファイルがあるか　list.files()
6）ディレクトリの移動　setwd("NICER_NS")

R で正規表現

https://stats.biopapyrus.jp/r/devel/regex.html

他のプログラミング言語と違う注意点

エスケープは２回 ¥¥

文字クラスの指定も２回 [[　]]

メタ文字

. ピリオド一つは何でも一文字

* アスタリスクは直前の文字の０回以上の繰り返し

*? とすると「最短一致」

+ プラス記号は直前の文字の１回以上の繰り返し

+? とすると「最短一致」

? 疑問符は直前の文字（項目）の０回もしくは１回の繰り返し。つまり、あってもなくてもよい。

(　) 丸かっこで囲むとその中の文字列が連続したカタマリとして扱われる。

文字クラスを表す [　] との違いに注意。

| は、その前後の文字のどちらか。(　|　) で囲むと、その前後の文字列のどちらか。

例：　(am|is|are)

^ は、行頭

$ は、行末

¥w で半角英数とアンダースコア

¥W ¥w 以外のもの

¥b 「単語」の境界

例：　¥bthe¥b

文字クラス

[ ] で囲まれた「クラス」はその中のどれか一文字

例：[Hh]owever

[-] で、－の前後の連続する文字列の範囲　例：[a-z] は小文字ａからｚまで

例：[a-zA-Z]

例：[0-9]

例：[a-zA-Z0-9]

[^] と最初に ^ を置くと、そこに並ぶいずれの文字でもないもの、という意味（つまり否定）

[:space:] で半角スペース・タブ記号・改行記号

[:punct:] で句読点類

[:digit:] で数字

[:lower:] 小文字

[:upper:] 大文字

[:alpha:] 大文字・小文字

[:alnum:] で半角英数

(準備)先週の復習:テキストファイルからデータを読み込む

```
jpn501 <- scan("JPN501.txt", what="char", sep="\n")
jpn502 <- scan("JPN502.txt", what="char", sep="\n")
jpn503 <- scan("JPN503.txt", what="char", sep="\n")
jpn504 <- scan("JPN504.txt", what="char", sep="\n")
jpn505 <- scan("JPN505.txt", what="char", sep="\n")
jpn506 <- scan("JPN506.txt", what="char", sep="\n")
jpn507 <- scan("JPN507.txt", what="char", sep="\n")
jpn508 <- scan("JPN508.txt", what="char", sep="\n")
jpn509 <- scan("JPN509.txt", what="char", sep="\n")
jpn510 <- scan("JPN510.txt", what="char", sep="\n")
ns501 <- scan("NS501.txt", what="char", sep="\n")
ns502 <- scan("NS502.txt", what="char", sep="\n")
ns503 <- scan("NS503.txt", what="char", sep="\n")
ns504 <- scan("NS504.txt", what="char", sep="\n")
ns505 <- scan("NS505.txt", what="char", sep="\n")
ns506 <- scan("NS506.txt", what="char", sep="\n")
ns507 <- scan("NS507.txt", what="char", sep="\n")
ns508 <- scan("NS508.txt", what="char", sep="\n")
ns509 <- scan("NS509.txt", what="char", sep="\n")
ns510 <- scan("NS510.txt", what="char", sep="\n")
```

データをまとめる

```
> ns10 <-c(ns501, ns502, ns503, ns504, ns505, ns506, ns507, ns508,
ns509, ns510)


> jpn10 <-c(jpn501, jpn502, jpn503, jpn504, jpn505, jpn506, jpn507,
jpn508, jpn509, jpn510)
```

まとめたデータで検索

```
grep("[hH]owever", ns10, value=T)
 [1] "*NS501:\tHowever in the French educational system instead of a head or a body
 [2] "*NS501:\tThis makes the facts easy to access, however, it does not force the
 [3] "*NS501:\tHowever what the French lose in logical flow they gain in critical
 [4] "*NS501:\tHowever, sadly with the continuous failings of the American
 [5] "*NS502:\tHowever, with growing competition in workplaces and with newer jobs
 [6] "*NS503:\tHowever, I worry that in today's increasingly global society, in which
 [7] "*NS503:\tHumanities-based education and analysis, however, has the potential
 [8] "*NS504:\tHowever, both systems are not completely different, as they both take
 [9] "*NS505:\tHowever Australians have sought to distinguish themselves from the
[10] "*NS505:\tHowever, there are also some negative aspects to Australia's sporting
[11] "*NS505:\tAustralia presents itself to the world as a sporting nation, however I
[12] "*NS507:\tHowever, the situation in the United States is much different with
[13] "*NS507:\tHowever, a similar attitude is displayed when an American finds
[14] "*NS508:\tHowever, as English began to increase in popularity worldwide its
[15] "*NS508:\tHowever, even thought the education method is quite successful the
[16] "*NS508:\tHowever, that is most certainly easier said than done."
[17] "*NS509:\tHowever, today paper money is made of the same materials and the
[18] "*NS509:\tHowever, I think that people need to reassess what is important to
[19] "*NS509:\tHowever, when it comes down to it, it is all just paper."
[20] "*NS509:\tHowever, the value of material objects is completely up to us as
[21] "*NS510:\tHowever, there are those that claim that any opposition towards these
```

データの中身を「見てみる」（一部を取り出す）

R に読み込んだ母語話者データ　ns501　を例に。

行単位で読み込んであるものとする。（★行単位＝配列の「要素」が「行」）

先頭部分を見る

head(変数名)

head(変数名, 要素数)

> head(ns501, 25)

末尾部分を見る

tail(変数名, 要素数)

> tail(ns501, 25)

★下からデータを取る＝上のデータを削除する
マイナスの値を指定することで、上からその数だけ削除したものが出力される

tail (変数名, -1)

つまりこれは、結果的に、最初の一行の削除と同じこと

●練習

先頭１４行を見る

末尾６７行を見る

●２５行目から２７行目だけを見るには？

上から２７行目出力した後で、下から３行とれば、結果的に２５、２６、２７行目が出る。

head(ns501, 27) した後で、tail(, 3) をする。

★処理を連続して行う

結果を保存するには　「変数 <- 命令」なので、結果を一時保存して、

```
> ns501.27 <- head(ns501, 27)
> tail(ns501.27, 3)
```

これを途中で保存せずに、まとめてするには、括弧の中に入れてしまう。「入れ子」で連続処理

```
> tail(head(ns501, 27), 3)
```

●特定の行だけ取り出すには？　（行：配列の要素）＝＞　要素番号を指定（要素番号は１から）

```
> ns501[76]
> ns501[76:78]              #76 行目から 78 行目まで（コロンで範囲の指定）
> ns501[c(76, 77, 78)]
> ns501[c(76, 78)]          #  76 と 78 だけ
```

単語リストの作成

1)まず、単語リストを作るために本文データのみを抜き出す

・単語リストを作るには、本文情報の部分のみを使う。
・データのフォーマットがどうなっているかを踏まえて、「処理」を行う。
・各データは、一文一行で、各行頭に「発話者コード」が入っている。
　学習者データは、例えば、JPN501:　となっている。
　語簿話者データは例えば、NS501:　となっている。
・必要な行だけ取り出すにはどうしたらよいか。

＝＞母語話者データなら、NS501 という文字列が入っている行を取り出せばよいはず。

```
> grep("NS501", ns501, value=T)
```

●結果を見てみると一つ目と二つ目の要素は該当しない（ヘッダーの説明）ものなので不要

```
 [1] "@Participants:¥tNS501"
 [2] "@PID:¥tPIDNS501"
```

＝＞　上の二行を削除
　　tail の応用で削除

●母語話者データのデータ部分のみ抽出（抽出した結果を新しい変数 ns501.data に入れる）

```
> ns501.data <- tail(grep("NS501", ns501, value=T), -2)
> ns501.data
```

2)各行単位で入っているデータを、単語単位にバラバラにする

●文字列をバラバラにする命令

strsplit(変数名, ″セパレータ″)

```
> strsplit(ns501.data, ″ ″)
```

ns501.data をスペースで切って行単位の中で、単語ごとにバラバラにする

バラバラにした結果は「リスト」に入る（２次元データ）（各行の中で単語単位になっている）

```
[[1]]
[1] [2] ...
[[2]]
[1] [2] ...
[[3]]
[1] [2] ...
```

```
> ns501.data.list <- strsplit(ns501.data, ″ ″)
```

一旦「ns501.data.list」に入れる

●リストの要素をバラバラにする（リスト形式をやめる＝行単位をやめる）

unlist(変数名)

```
> unlist(ns501.data.list)
```

unlist(strsplit(ns501.data, ″ ″)) でも同じ

```
> tmp <- unlist(ns501.data.list)
```

3)バラバラになった単語を並べ替える（token 述べ語）

sort(変数名)

```
> sort(tmp)
```

もしくは

```
> sort(unlist(ns501.data.list))
```

★これで、単語の一覧ができた（述べ語）

4)並べ替えた単語の一覧の重複をまとめる(type 異なり語)

(★UNIX のコマンド uniq とつづりが違うので注意)

unique(変数名)

> unique(sort(unlist(ns501.data.list)))

これで一応「単語リスト」はできるが、できたものをよく見てみよう。
これでよいかな？

■単語ではないもの（話者記号）が入っている。　★これは入っていてほしくない。
●話者記号「*NS501:¥t」が入っている　＝＞　削除したい

5)不必要な「文字」を削除するには

●削除するとは、、、何もなしで置き換える

●置換コマンド

gsub(元の文字列, 新しい文字列, 対象データ)

不要なのは「*NS501:¥t」

6) じゃ、最初から、やり直してみよう。(最初に話者記号を取り除いておこう):

単語リストの作成のまとめ

> gsub("*NS501:¥t", "", ns501.data)

うまくいかない、、、行頭の * が残ってしまう、、、なんでかな？

特殊文字「*」は、「エスケープ」する必要がある
　　＝＞　¥¥　を前につけてエスケープ

> gsub("¥¥*NS501:¥t", "", ns501.data)

この結果を、ns501.data2 という変数で保存

> ns501.data2 <- gsub("¥¥*NS501:¥t", "", ns501.data)

★単語リストを作るには、あと、「句読点の問題」と「大文字・小文字」の問題がある。

7)演習

句読点と大文字・小文字はひとまず、そのままにして、単語リスト（述べ語・異なり語）を作成してみよう。

ns501.token <- ここにコマンドを書く
ns501.type <- ここにコマンドを書く

述べ語数はいくつかな？
異なり語数はいくつかな？

★宿題

１）復習として、学習者と母語話者の１番から３番のデータについて、延べ語数（総語数）と異なり語数を出してみよう。できれば１０個ずつ全部やってみよう。

２）句読点の問題を解決してみよう。（オプション）
　　ヒント：gsub を使う。句読点は[[:punct:]]

３）大文字と小文字の問題を解決してみよう。（オプション）
　　ヒント：小文字にする命令　tolower(変数)

■2019-11-15

やってきたことの確認

０）データをＲに読み込む（行単位）
```
> ns501 <- scan("NS501.txt", what="char", sep="\n")
```

１）本文部分のみ取り出す
```
> ns501.body <- tail(grep("NS501", ns501, value=T), -2)
```

２）発話記号を削除
```
> ns501.data <- gsub("\\*NS501:\t", "", ns501.body)
```

３）単語リストを作るために、単語ごとにばらばらにする
（行という単位の中で、さらに、「単語」という単位になる（リスト形式））
```
> ns501.data.list <- strsplit(ns501.data, " ")
```

４）二次元になっている「リスト」形式データを
（行という単位をなしにして、「単語」だけを単位とするには、リスト形式をやめる。
```
> ns501.data.unlist <- unlist(ns501.data.list)
```

５）アルファベット順に並べ替える。
（すべての「単語」が、一行一単語に）（行数＝総語数、延べ語数、token）
```
> ns501.token <- sort(ns501.data.unlist)
```

６）重複しているものを１つにする。
（異なっている「単語」ごとに、一行一単語（行数＝異なり語数、type）
```
> ns501.type <- unique(ns501.token)
```

７）type と toke の数を数える
```
> length(ns501.type)
```

```
> length(ns501.token)
```

★残された問題

・大文字と小文字

・英数字以外の記号類

■残された問題の解決

●大文字はすべて小文字に統一

tolower(変数)

```
> ns501.data2 <- tolower(ns501.data)
```

●英数字以外の記号の削除

記号を何もないもので置き換える

どんな「記号」があるか、、、、記号を全部書いていたのではとても大変、、、

逆に、必要なのは、アルファベットか数字だけ。じゃ、それら以外を削除する。

＝英数字以外を削除　＝＞　正規表現

●英数字を正規表現でどう表記するか？

```
a-z
A-Z
0-9

[a-zA-Z0-9]
```

それ「以外」（それではないもの）をどう表記するか。

```
[^a-zA-Z0-9]
```

ということは、英数字以外を削除というのは、、、

```
gsub("[^a-zA-Z0-9]", "", 変数)
> gsub("[^a-zA-Z0-9]", "", ns501.data2)
```

＝＞おっと、これでは、文字が全部つながってしまう。（スペースも削除されてしまった）

```
> gsub("[^a-zA-Z0-9]", " ", ns501.data2)
```

＝＞スペースがそもそもあったところに記号があると、スペース二つになってしまう。

＝＞スペース二つをスペース一つに置き換えてみよう。

```
> ns501.data3 <- gsub("[^a-zA-Z0-9]", " ", ns501.data2)
> gsub("  ", " ", ns501.data3)
```

★二つ以上スペースがあってもとにかく一つにするには、正規表現 + を使って

```
gsub(" +", " ", 変数)
> gsub(" +", " ", ns501.data3)
```

●もうひとつの表記法（メタ文字）

英数字　　\w

英数字以外　\W　　（大文字）

なので、

gsub("\\W", " ", 変数)

●句読点を削除する別の方法

[:punct:]

gsub("[[:punct:]]", " ", 変数)

★これで、「残された問題」は解決するはず。

♪しかし、、、本質的な問題は、「単語」とは何か、という問題。

＝＞　どう定義するか？　自分で決めて、その定義を記述しておく。

例１）「don't」は「一単語」と扱う。
　　＝＞　事前に「don't」を「donAPSt」に置き換えておく。

　　gsub("don't", "donAPSt", 変数)

　　＝＞　縮約形の置き換えリストを用意しておく。

　　＝＞　置き換えリストを使って「前処理」と「後処理」をする。

例２）「don't」は、「do」と「n't」に分ける。
　　＝＞　事前に「don't」を「do」と「nAPSt」に置き換えておく。（間にスペース）

　　gsub("don't", "do nAPSt", 変数)

★いずれにせよ、最終的なリストを完成するときに、元に戻しておく。

★さて、では、もう一度、総語数と異なり語数を出してみよう。

★10種類の処理（復習）

ns501.data（行単位でデータ部分のみが入っている）

１）話者記号を取る
```
gsub("\\*NS501:\t", " ", 変数)
```

２）「スペース」でばらばら要素にする（行単位のまま、単語がばらばらに）
```
strsplit(変数, " ")
```

３）「リスト」形式をやめて要素の一覧に（行単位をなくして単語がばらばらに）
```
unlist(変数)
```

４）記号類を削除
```
gsub("\\W", " ", 変数)
```

５）大文字を小文字に
```
tolower(変数)
```

６）並び替え（アルファベット順の総語リスト）
```
sort(変数)
```

７）重複をなくす（異なり語リスト）
```
unique(変数)
```

８）二つ以上のスペースを一つにするのも忘れないでね。
```
gsub(" +", " ", 変数)
```

９）行末のスペースの削除　　★行末を表す正規表現　$
```
gsub(" $", "", 変数)
```

10）行頭のスペースの削除　　★行頭を表す正規表現　^
```
gsub("^ ", "", 変数)
```

■作業の仕方のコツ

１）やり方１
　順に一時的な変数に入れていく

２）やり方２
　どんどん入れ子にしていく

０）一方で作業をRエディターでメモしておく
　コマンドを、そこからコピーして実行
　コマンドをマウスで選択しておいて、「選択中のコードを実行」

★現実的には、まず、細かい処理を先に済ませておく。（前処理）

★準備として、

1）話者記号を取る	gsub("*NS501:\t", " ", 変数)
2）大文字小文字問題と	tolower(変数)
3）句読点問題と	gsub("\\W", " ", 変数)
5）二つのスペースの問題と	gsub(" +", " ", 変数)
4）行末スペース問題と	gsub(" $", "", 変数)
5）行頭スペース問題を	gsub("^ ", "", 変数)

処理しておく。

★入れ子式に命令を連続処理

```
gsub("^ ", "", gsub(" $", "", gsub(" +", " ", gsub("\\W", " ",
tolower(gsub("\\*NS501:\t", " ", 変数))))))
```

ns501.body（行単位で本文部分のみが入っている）を対象に処理をしておく。

```
> gsub("^ ", "", gsub(" $", "", gsub(" +", " ", gsub("\\W", " ",
tolower(gsub("\\*NS501:\t", " ", ns501.body))))))
```

前処理をした結果を変数 ns501.words に保存

```
> ns501.words <- gsub("^ ", "", gsub(" $", "", gsub(" +", " ",
gsub("\\W", " ", tolower(gsub("\\*NS501:\t", " ", ns501.body))))))
```

★その後で、「本処理」

▲アルファベット順の総語（token）リスト

```
> ns501.token2 <- sort(unlist(strsplit(ns501.words, " ")))
```

▲アルファベット順の異なり語（type）リスト

```
> ns501.type2 <- unique(ns501.token2)
```

●要素数を数える

```
length()
```

●出来上がったリストの出力（テキストファイルとして保存）

write(変数名, "ファイル名")

```
> write(ns501.token2, "ns501-token.txt")
> write(ns501.type2, "ns501-type.txt")
```

★どこに保存されたか？　今いるディレクトリー（フォルダー）

それはどこ？

```
> getwd()
```

■オマケ

●行数を数える

```
nrow()
```

●ちなみに列数を数える

```
ncol()
```

●変数の名前を変える

新しい名前 <- 古い名前

```
rm(古い名前)
```

単語の頻度一覧表の作成

```
table(述べ語数の変数)
```

```
> table(ns501.token2)
```

これだけ！

```
> ns501.table <- table(ns501.token2)
```

●テーブルをファイルに保存しましょう。

注意点

1）どこに保存されるか確認。カレントディレクトリーは： `getwd()`
上へ移動は： `setwd("..")`

2）テーブルの保存は命令が違う！

```
write.table(ns501.table, "保存ファイル名")
```

```
> write.table(ns501.table, "ns501-table.txt")
```

これで、表（テーブル）として保存される。

●エクセルで開いてみよう。

区切りは、デフォルトは　スペース

区切りをタブに変えるには、オプションを付けて保存。
（エクセルで読み込むときに「区切り文字」をスペースにするのも手）

```
write.table(ns501.table, "保存ファイル名", sep="\t")
```

★同じような処理を何回もする＝＞　プログラミングしてひとつの「命令」としてまとめる

function() で独自の命令（関数）を作成

■同じような処理をまとめて一つの命令にする
　自分が作る新しい命令の名前は勝手に付ける。習慣として 「my 何とか」とする。

```
============================================================
自分の新しい命令 <- function(){
        ここに実行したい命令を順に書いていく
        ここに実行したい命令を順に書いていく
        ここに実行したい命令を順に書いていく
        ここに実行したい命令を順に書いていく
        }
============================================================
```

■myData()

データを読み込み、本文のテキスト部分だけを変数に入れる

```
-----------------------------------------
myData <- function(){
      lines.tmp <- scan(choose.files(), what="char", sep="\n")
            #ファイルを選択。
      data.tmp <- grep("\\*(JPN|NS)...:\t", lines.tmp, value=T)
            #*で始まり
            # JPN か NS があって、
            # その後ろに、3 文字あって、
            # その後ろに、コロンの記号とタブ記号がある行のみ。
      body.tmp <- gsub("\\*(JPN|NS)...:\t", "", data.tmp)
            #行頭の記号とタブ記号を削除。
      body.tmp <- body.tmp[body.tmp != ""]
            # 空の要素を削除（空でない要素のみを残す）する「イディオム」
}
-----------------------------------------
```

★メニューの「ファイル」から「新しいスクリプト」を選んで、「R エディタ」に書きこむ。
★「R エディタ」の中で、該当部分をマウスで選んでおいて、メニューの「編集」から「カーソル行または選択中の R コードを実行」すれば、「myData」というコマンドが出来上がる。
　ls() で確認。（myData が表示される）

●コマンドの実行（この例だと、単に読み込むだけなので結果は残らない）

```
> myData()
```

●コマンドの実行結果の保存（読み込んだものを変数に保存する）

```
変数名 <- myData()
```

を実行。　開くファイルを聞かれる（命令の中の choose.files が実行され）ので、指定する。

たとえば、

```
> ns502.body <- myData()
```

今日の課題:

１）myData()に加筆修正して、コーパスデータをウィンドウで選んだら、単語の頻度一覧表がでる関数を作成してみよう。

２）学習者・母語話者それぞれ 10 ファイルの単語頻度一覧表を作成しよう。

■11 月 22 日： R でプログラミング

■異なり語数と、述べ語数がわかれば、いろいろわかる。

1）テキスト（英文エッセイ）の長さ（含まれる単語数＝述べ語数）

2）TTR

Type/Token

3）Guiraud index

Type/sqrt(Token)

★それぞれのファイルの Type と Toke と TTR と Guiraud Index をいちいち命令を入れていくというのは、何度も同じことを繰り返すことになって、面倒だなぁ、、、、

■myTTR()

TTR をひとまとめのコマンドに

読み込んであるデータ（例えば ns502.body）を対象に、そのデータの変数名を指定すれば、言語指標を出してくれるような命令を考える。

★独自命令を作る関数 function() でカッコの中に変数を入れる点がポイント。

命令実行の際に、カッコ内に変数を指定すると、その変数を対象に、それ以降の命令が実行されるようになる。

```
myTTR <- function(a){                                  # a の位置に変数が来る
     body.lower <- tolower(a)                          # その変数を対象に処理する
                                                       # 小文字にして
     body.nopunc <- gsub("\\W", " ", body.lower)
                                                       # 記号をスペースに
     body.single <- gsub(" +", " ", body.nopunc)
                                                       # 重複スペースを一つに
     body.clear <- gsub(" $", "", body.single)
                                                       # 文末スペースの削除
     body.token <- unlist(strsplit(body.clear, " "))
                                                       #
     body.token <- body.token[body.token != ""]
                                              # 空の要素を削除する「イディオム」
     length(unique(body.token))/length(body.token)
}
```

これでコマンドとして実行

* スクリプト中の「a」の部分に、「ns502.body」と記述すると、これを対象に、処理が行われる。

```
> myTTR(ns502.body)
```

●もう一つ別の方法

```
myTTRb <- function(a){
     body.lower <- tolower(a)
     body.token <- unlist(strsplit(body.lower, "\\W"))
     body.token <- body.token[body.token != ""]
     length(unique(body.token))/length(body.token)
}
```

■myGI()

Guiraid index（token をルートに入れるだけ）

```
myGI <- function(a){
     body.lower <- tolower(a)
     body.token <- unlist(strsplit(body.lower, "\\W"))
     body.token <- body.token[body.token != ""]
     length(unique(body.token))/sqrt(length(body.token))
}

> myGI(ns502.body)
```

■myAWL()

平均単語長（Average Word Length）

平均単語長とは、単語を構成する文字数の平均（Average Word Length）

総文字数　÷　総単語数　ででるはず。

総単語数は、述べ語数なので、わかってる。

じゃ、総文字数はどう出すか？

★文字数を数えるコマンド（関数）：　nchar()

これだと、カッコ内の変数に含まれる総文字数（スペースも含む）がでる。

じゃ、単語を全部スペースなしでつなげたもの（長い文字列）が入った変数をつくればよい。

★要素をつなげるコマンド：　paste(変数, collapse="")
こうしておいて、総文字数を調べればよい。

```
myAWL <- function(a){
      body.lower <- tolower(a)
      body.token <- unlist(strsplit(body.lower, "\\W"))
      body.token <- body.token[body.token != ""]
      nchar(paste(body.token, collapse=""))/length(body.token)
}

> myAWL(ns502.body)
```

◆途中で中身を出力させる小技

単語の一覧も出力するには、print()　命令を入れておく

```
myAWLp <- function(a){
      body.lower <- tolower(a)
      body.token <- unlist(strsplit(body.lower, "\\W"))
      body.token <- body.token[body.token != ""]
      print(body.token)
            #このタイミングでプリント
      nchar(paste(body.token, collapse=""))/length(body.token)
}
```

★発展課題：出力される単語を単語長の長いもの順にならべて出力できるといいな。

■myASL()

平均文長 (Average Sentence Length)

一文に含まれる単語数の平均

総単語数　÷　総文数

（1）ファイル全体で文がいくつあるか調べる。
（2）ファイル全体で総単語数を調べる。
（3）単語数を文数で割ると平均が出る。

```
myASL <- function(a){
      body.lower <- tolower(a)
      body.token <- unlist(strsplit(body.lower, "\\W"))
      body.token <- body.token[body.token != ""]
      length(body.token)/length(a)
}

> myASL(ns502.body)
```

★文の一覧も出力するには、 print() 命令を入れておく

```
myASLp <- function(a){
      print(a)
            #このタイミングでプリント
      body.lower <- tolower(a)
      body.token <- unlist(strsplit(body.lower, "\\W"))
      body.token <- body.token[body.token != ""]
      length(body.token)/length(a)
}
```

◆　命令を「入れ子」にする

ここまでは、一旦本文データを変数に読み込んで（ns502.body を作って）、それを対象に処理してきたが、いちいち変数に保存しなくても、命令を「入れ子」にする手もある。

```
> myASL(myData())
```

これで、GUI で開いたウィンドウでファイルを選べば、平均文長の結果だけが出力される。

よく使うのであれば、一つの命令にまとめてしまったほうが便利。

■myTT()

ファイルを指定すると、Type と Token の両方を一度に出力する関数

★併記して出力 cat(　　,　　, "\n")
　最後に、「改行」を示す「"\n"」をつける。

```
myTT <- function(){
      lines.tmp <- scan(choose.files(), what="char", sep="\n")
            #ファイルを選択。
      data.tmp <- grep("\\*(JPN|NS)...:\t", lines.tmp, value=T)
            #*で始まる行のみ。
      body.tmp <- gsub("\\*(JPN|NS)...:\t", "", data.tmp)
            #行頭の記号とタグを削除。
      body.tmp <- body.tmp[body.tmp != ""]
            # 空の要素を削除する「イディオム」

      body.lower <- tolower(body.tmp)
      body.token <- unlist(strsplit(body.lower, "\\W"))
      body.token <- body.token[body.token != ""]
            # 空の要素を削除する「イディオム」

      cat(length(unique(body.token)), length(body.token), "\n")
            # 二つの要素を配列に
}

> myTT()
```

★一つ一つのファイルを開いて選ぶのは、ファイルがたくさんあると面倒!

■複数のファイルに対して繰り返し自動的に実行する

1）特定のファイルを処理するプログラムを作る

2）ファイル一覧のすべてのファイルに対して同じ処理を繰り返すようにする

●準備：ファイル名を指定して実行するには、

scan()命令で、choose.files()をせずに、ファイル名を指定する。

```
scan(choose.files(), what="char", sep="\n")

scan("JPN501.txt" , what="char", sep="\n")     # JPN501.txt を例に
```

★話を分かりやすくするために、

★作業ディレクトリーを、データのあるディレクトリーに変更　★重要★

＝＞メニューの「ファイル」から、「ディレクトリの変更」

getwd() で確認。

そこにあるファイル一覧を表示して確認

```
> list.files()
```

dir() でも OK

●1）JPN501.txt 専用のプログラムを作ってみる。

```
# 特定のファイルだけを処理するスクリプト
# ------------------------------
myTTJPN501 <- function(){
      lines.tmp <- scan("JPN501.txt", what="char", sep="\n")
            #ファイルを選択。
      data.tmp <- grep("\\*(JPN|NS)...:\t", lines.tmp, value=T)
            #*で始まる行のみ。
      body.tmp <- gsub("\\*(JPN|NS)...:\t", "", data.tmp)
            #行頭の記号とタグを削除。
      body.tmp <- body.tmp[body.tmp != ""]
            # 空の要素を削除する「イディオム」

      body.lower <- tolower(body.tmp)
      body.token <- unlist(strsplit(body.lower, "\\W"))
      body.token <- body.token[body.token != ""]
            # 空の要素を削除する「イディオム」

      cat(length(unique(body.token)), length(body.token), "\n")
}
# ------------------------------
```

後は実行：

```
> myTTJPN501()
```

●２）フォルダー（ディレクトリー）内にあるファイルすべてについて実行する。

★何が必要か？
＝＞　ファイルの一覧

★それをどうするとよいか？
＝＞　順番に繰り返してファイルを指定する

★ファイル名の一覧は

```
list.files()
```

ファイルの一覧を変数に入れる

```
files <- list.files()
```

★繰り返して実行（制御：プログラムの流れをコントロールする）

```
for (条件) {
            すること
            すること
}
```

★「条件」部分の書き方

　変数 in その範囲

例： i in 1:19

　１から１９まで、順に i に入れる

例： i in files

　filesの中の個々のファイルを、順に i に入れる

```
# ディレクトリー内のすべてのファイルを処理するスクリプト
# ------------------------------------------
myTTfiles <- function() {
      files <- list.files()
      for (i in files) {

            type <- 0
            token <- 0
                  # typeとtokenの変数の初期化

            lines.tmp <- scan(i, what="char", sep="\n")
                  #ファイルを選択。
            data.tmp <- grep("\\*(JPN|NS)...:\t", lines.tmp, value=T)
                  #*で始まる行のみ。
            body.tmp <- gsub("\\*(JPN|NS)...:\t", "", data.tmp)
                  #行頭の記号とタグを削除。
            body.tmp <- body.tmp[body.tmp != ""]
                  # 空の要素を削除する「イディオム」

            body.lower <- tolower(body.tmp)
            body.token <- unlist(strsplit(body.lower, "\\W"))
            body.token <- body.token[body.token != ""]
                  # 空の要素を削除する「イディオム」

            type <- length(unique(body.token))
            token <- length(body.token)

            cat(type, token, "\n")
                  #cat は結果を出力する命令（改行が必要なら "\n"も出力する）

      }
}
# ------------------------------------------
```

後は実行：

```
> myTTfiles()
```

■宿題

フォルダー内のすべてのファイルの token・type・TTR・GI・AWL・ASL　を一度に出力するプログラムを作ってみよう。

（画面に結果が出力されるところまででよいです。）

（四捨五入のコマンドは　round()　小数点以下３桁目を四捨五入して２ケタまでだすには、
round(変数, digits = 2)

■11 月 29 日

■結果をファイルに保存する。

▲ファイル一覧のファイルを順に処理し、結果を「指定したファイル名」で保存する。

★結果を保存するファイルの保存場所に注意、一つ上のフォルダーに入れること。

```
# 作業ディレクトリー内のすべてのファイルを処理し、結果をファイルに保存するスクリプト
        # 使用上の注意          1）対象は作業ディレクトリー内のファイルすべて
        #                       2）結果を保存するファイルはフォルダーの外（一つ上のフォルダ）
```

★結果の保存は cat() にオプションを指定する。

```
cat(type, token, "\n", file=output.file, append=T)
```

```
# ------------------------------------------------------------------
myBasicInfo <- function() {

output.file = choose.files()
      #結果を保存するファイル名を指定する。保存場所に注意
      # （ファイル名を聞かれたら適当に名前を付ける。例えば jpnTT.txt）

files <- list.files()

for (i in files) {

type <- 0
token <- 0
      # typeとtokenの変数の初期化

      lines.tmp <- scan(i, what="char", sep="\n")
            #ファイルを選択。
      data.tmp <- grep("\\*(JPN|NS).:\t", lines.tmp, value=T)
            #*で始まる行のみ。
      body.tmp <- gsub("\\*(JPN|NS).:\t", "", data.tmp)
            #行頭の記号とタグを削除。
      body.tmp <- body.tmp[body.tmp != ""]
            # 空の要素を削除する「イディオム」

      body.lower <- tolower(body.tmp)
      body.token <- unlist(strsplit(body.lower, "\\W"))
      body.token <- body.token[body.token != ""]
            # 空の要素を削除する「イディオム」

      type <- length(unique(body.token))
      token <- length(body.token)
      ttr <- type/token
      GI <- type/sqrt(token)
      AWL <- nchar(paste(body.token, collapse = ""))/token
      ASL <- length(body.token)/length(body.tmp)

      cat(type, token, ttr, GI, AWL, ASL, "\n", file=output.file,
append=T)
            #cat は出力をファイルに保存もできる
            #ファイル名は、最初に output.file で決めたファイル名
      }
}
# ------------------------------------------------------------------
```

ファイルに保存された。（エクセルで開いてみてみよう。）

■処理結果のファイルを読み込む（例： basicInfo に保存する）

```
> basicInfo <- read.table(choose.files())
```

```
> head(basicInfo)
  V1  V2        V3       V4       V5       V6
1 135 319 0.4231975 7.558549 4.304075 10.63333
2 161 356 0.4522472 8.532983 4.233146 12.27586
3 121 201 0.6019900 8.534682 4.746269 15.46154
4 140 260 0.5384615 8.682431 4.761538  9.62963
5 175 420 0.4166667 8.539126 3.995238 16.80000
6 124 261 0.4750958 7.675407 4.072797 13.05000
```

見出しがない。

見出しを付けるには、names(変数) <- c("見出し名", "見出し名")

```
> names(basicInfo) <- c("Type", "Token", "TTR", "GI", "AWL", "ASL")
> head(basicInfo)
  Type Token       TTR       GI      AWL      ASL
1  135   319 0.4231975 7.558549 4.304075 10.63333
2  161   356 0.4522472 8.532983 4.233146 12.27586
3  121   201 0.6019900 8.534682 4.746269 15.46154
4  140   260 0.5384615 8.682431 4.761538  9.62963
5  175   420 0.4166667 8.539126 3.995238 16.80000
6  124   261 0.4750958 7.675407 4.072797 13.05000
```

■基本統計量を見る

```
> summary(basicInfo)
      Type           Token            TTR
 Min.   : 50.0   Min.   : 90.0   Min.   :0.2519   (最小値)
 1st Qu.:103.0   1st Qu.:216.0   1st Qu.:0.4181   (四分位点の1/4値)
 Median :124.0   Median :269.0   Median :0.4629   (中央値)
 Mean   :128.2   Mean   :284.6   Mean   :0.4642   (平均)
 3rd Qu.:148.0   3rd Qu.:329.0   3rd Qu.:0.5106   (四分位点の3/4値)
 Max.   :252.0   Max.   :736.0   Max.   :0.6581   (最大値)

       GI              AWL             ASL
 Min.   : 4.499   Min.   :3.368   Min.   : 7.04
 1st Qu.: 7.008   1st Qu.:4.116   1st Qu.:11.04
 Median : 7.559   Median :4.366   Median :12.43
 Mean   : 7.617   Mean   :4.363   Mean   :12.89
 3rd Qu.: 8.359   3rd Qu.:4.608   3rd Qu.:14.44
 Max.   :10.453   Max.   :5.388   Max.   :24.26
```

★特定のカラムだけ見るには　$　を使う

```
> basicInfo$Type
```

```
> summary(basicInfo$Type)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   50.0   103.0   124.0   128.2   148.0   252.0
```

■グラフで見てみる

■箱ひげ図：　boxplot()

http://sugiura-ken.org/wiki/wiki.cgi/exp?page=boxplot

データの分布を見る（比較する）

- 真ん中の太い線：中央値 Median（50%）、平均ではないので注意
- 箱の範囲：25%から 75%の範囲（これを四分範囲と呼ぶ）
  （つまり、半分のデータはこの範囲に入る）
- ひげの範囲：箱の端から箱の長さの 1.5 倍以内にある「実際の」数値の最大のもの最小のもの
- 外れている○印：外れ値（ひげの範囲外のもの）

```
> boxplot(basicInfo)
```

```
> boxplot(basicInfo$Type)
```

```
> boxplot(basicInfo$Type, basicInfo$Token)
```

■散布図 plot()

```
plot(basicInfo)              # 散布図を書く。二次元

plot(basicInfo$Token)        # Token だけ。ファイル番号順に。
plot(sort(basicInfo$Token))      # Token で並び変えて、散布図に。

plot(basicInfo$Type)             # 異なり語
plot(sort(basicInfo$Type)) # 異なり語　並び変えて
```

sort（並び替え）のオプション：降順　decreasing = T

■棒グラフ barplot()

```
barplot(basicInfo$Type)
barplot(sort(basicInfo$Type))    # 並び変えて

barplot(basicInfo$Token)
barplot(sort(basicInfo$Token))   # 並び変えて
```

■ヒストグラム：　hist()

```
> hist(basicInfo$Token)
```

★グラフに色を付けるオプション　col="色"

★「TTR はテキストの長さに影響を受ける」といわれている。確かめてみよう。

「テキストの長さ」とは、Token の数のこと。

Token の数と TTR の関係の散布図を描いてみる。（相関が見える）

```
plot(basicInfo$Token, basicInfo$TTR)
grid()        # 格子を入れてみる
```

■　散布図の「平滑化」（スムーズにつながるように線を引く）

```
lines(lowess(basicInfo$TTR~basicInfo$Token))
```

lowess()で平滑化をして（y 軸～x 軸）
lines() で線を引く

★ギローインデックスはどうだろうか？

色を変えて、一枚に両方プロットして比べてみよう。

グラフを重ねる命令　par(new=T)

```
plot(basicInfo$Token, basicInfo$TTR)
lines(lowess(basicInfo$TTR~basicInfo$Token))
par(new=T)
plot(basicInfo$Token, basicInfo$GI, col ="red")
lines(lowess(basicInfo$GI~basicInfo$Token), col = "red")
```

（y 軸の尺度が違う点に注意。一枚に入るように自動的に調整されている）

★「TTR はテキストの長さに影響を受ける 」ということは、

テキストの長さ(総語数)と TTR の間に相関があるということ

（１） ピアソン(pearson)の相関係数（パラメトリック）
（２） スピアマン(spearman)の相関係数（ノンパラメトリック）
（３） ケンドール(kendall)の相関係数（ノンパラメトリック）

★パラ？
http://aoki2.si.gunma-u.ac.jp/lecture/Kentei/nonpara.html

＝＞　正規分布をしているかを検定してみればよい。

★正規分布の検定
shapiro.test() で Shapiro-Wilk 検定
(p 値が .05 より大きいと、正規分布しているとみなせる)

```
> shapiro.test(basicInfo$Token)

        Shapiro-Wilk normality test

data:  basicInfo$Token
W = 0.93409, p-value = 2.606e-11
```

■相関係数を出して、かつ有意かどうかも確かめる
http://oku.edu.mie-u.ac.jp/~okumura/stat/correlation.php

cor.test(x, y, method="spearman")
(p 値が .05 より小さいと、相関があるとみなせる)

```
> cor.test(basicInfo$Token, basicInfo$TTR, method="spearman")

        Spearman's rank correlation rho

data:  basicInfo$Token and basicInfo$TTR
S = 11380428, p-value < 2.2e-16
alternative hypothesis: true rho is not equal to 0
sample estimates:
      rho
-0.6063387
```

★　宿題

１）母語話者についても各種指標を出して、学習者と比べてみる。

２）グラフも描いて比べてみる。

■12 月 06 日

★では、テキストの長さの影響を受けないようにするにはどうするか。

１）テキストの長さを、一定にしてそろえる。何語にそろえるか？

２）テキストの長さに関係なく、一定の指標が出るようにする。

＝＞ MATTR

「The Moving-Average Type-Token Ratio.pdf」

Covington and McFall (2008) （リソースに PDF）

```
# 阿部大輔 2014
mattr = function () {
      wttr = 0
      ttrsum = 0

      x = scan(choose.files(), what="char", sep="¥n")
      y = tolower(unlist(strsplit(x, "¥¥W+")))

      numwords = length(y) #元々の語数を記録しておく

      y = c(y,y) #y を一周するとまた先頭の語が出てくるように y と y をつなげる

      for(i in 1:numwords){ #↑で記録した総語数の回数ループする
            mado = y[i:(99+i)]      # 100 語の幅で順に右にずらしていく
            wttr = length(unique(sort(mado)))/100 #100 語ずつ＝token が 100
            ttrsum = ttrsum + wttr  # 出てきた TTR を足していく
      }
      ttrsum/numwords    # 総 TTR 数を総語数の回数で割って平均を出す
}
```

★１００語の幅（窓）になっている。
★何語の幅が適切か？

★于君の実装 2015
https://github.com/rongmu/mattr/blob/master/mattr.R

＝＝＝＝＝＝＝＝＝＝＝＝■語彙頻度順上位語リスト＝＝＝＝＝＝＝＝＝＝＝＝

（myTT の応用）

■上位１０位まで

```
-------------------
myTop10 <- function(){
	lines.tmp <- scan(choose.files(), what="char", sep="¥n")
	data.tmp <- grep("¥¥*(JPN|NS)...:¥t", lines.tmp, value=T)
	body.tmp <- gsub("¥¥*(JPN|NS)...:¥t", "", data.tmp)
	body.tmp <- body.tmp[body.tmp != ""] #空行を削除。
	body.tmp.lower <- tolower(body.tmp) # 小文字に
	token.tmp <- unlist(strsplit(body.tmp.lower, "¥¥W+")) # 単語に
	token.tmp = token.tmp[token.tmp !=""] # あるのだけに

	#c(length(unique(token.tmp)), length(token.tmp)) # 二つの要素を配列に

	head(sort(table(token.tmp), decreasing=T), 10)
	# ★語彙リスト　降順並べ替え　上位１０
}
-------------------
```

■汎用性を持たせる：上位何位までも決められるように（引数で指定）

```
-------------------
myTop <- function(a){
	lines.tmp <- scan(choose.files(), what="char", sep="¥n")
	data.tmp <- grep("¥¥*(JPN|NS)...:¥t", lines.tmp, value=T)
	body.tmp <- gsub("¥¥*(JPN|NS)...:¥t", "", data.tmp)
	body.tmp <- body.tmp[body.tmp != ""] #空行を削除。
	body.tmp.lower <- tolower(body.tmp) # 小文字に
	token.tmp <- unlist(strsplit(body.tmp.lower, "¥¥W+")) # 単語に
	token.tmp = token.tmp[token.tmp !=""] # あるのだけに

	head(sort(table(token.tmp), decreasing=T), a) # ★順位を a で指定

}
-------------------

> myTop(20)
```

■連語表現の抽出

（概念をホワイトボードで説明）

```
--------------------
my2gram <- function(){
	lines.tmp <- scan(choose.files(), what="char", sep="¥n")
	data.tmp <- grep("¥¥*(JPN|NS)...:¥t", lines.tmp, value=T)
	body.tmp <- gsub("¥¥*(JPN|NS)...:¥t", "", data.tmp)
	body.tmp <- body.tmp[body.tmp != ""] #空行を削除。
	body.tmp.lower <- tolower(body.tmp) # 小文字に
	token.tmp <- unlist(strsplit(body.tmp.lower, "¥¥W+")) # 単語に
	token.tmp = token.tmp[token.tmp !=""] # あるのだけに

	for (i in 1:(length(token.tmp)-1)) {
	cat(token.tmp[i], token.tmp[i+1], "¥n")
	}
}
--------------------


> my2gram()
```

★ cat にオプションを付けて、ファイルに保存。
	file="ファイル名"

上書きではなく、追記するオプションも付ける。
	append=T

★ 保存するファイルを固定ではなく、最初に指定できるようにする。

```
output.file <- choose.files()

--------------------
my2gramSave <- function(){
	output.file <- choose.files()
	lines.tmp <- scan(choose.files(), what="char", sep="¥n")
	data.tmp <- grep("¥¥*(JPN|NS)...:¥t", lines.tmp, value=T)
	body.tmp <- gsub("¥¥*(JPN|NS)...:¥t", "", data.tmp)
	body.tmp <- body.tmp[body.tmp != ""] #空行を削除。
	body.tmp.lower <- tolower(body.tmp) # 小文字に
	token.tmp <- unlist(strsplit(body.tmp.lower, "¥¥W+")) # 単語に
	token.tmp = token.tmp[token.tmp !=""] # あるのだけに

	for (i in 1:(length(token.tmp)-1)) {
	cat(token.tmp[i], token.tmp[i+1], "¥n", file=output.file, append=T)
	}
}
--------------------


> my2gramp()
```

最初に保存ファイル名を入力（例　2gram.txt）
次に、分析対象ファイルを選択
結果はファイルとして保存される

●保存されたファイルを読み込んでRの中に入れる

```
> jpn2gram <- scan(choose.files(), what="char", sep="\n")
```

●頻度一覧表にする

```
> table(jpn2gram)
```

■上位10位の2グラム表現

```
> head(sort(table(jpn2gram), decreasing = T), 10)
```

■上位何位までか引数で指定できるように

```
--------------------
myTopBigram <- function(a) {
      tmp <- table(scan(choose.files(), what="char", sep="¥n"))
head(sort(tmp, decreasing=T), a)
}
--------------------
```

```
> myTopBigram(20)
```

★さ、これで、ファイルと順位を指定したら、その順位までのバイグラム表現を一気に表示してくれるプログラムを書きたくなったでしょ。

★で、次に、同様に3グラムで順位を表示してくれるものを書きたくなったでしょ。

■（復習）これまでの言語指標＋α をまとめて出力

```
-----------------------------------------------------------
myIndex <- function(){
      output.file = choose.files()
      files <- list.files()

      for (i in files) {
      lines.tmp <- scan(i, what="char", sep="¥n")
      data.tmp <- grep("¥¥*(JPN|NS)...:¥t", lines.tmp, value=T)
      body.tmp <- gsub("¥¥*(JPN|NS)...:¥t", "", data.tmp)
      body.tmp <- body.tmp[body.tmp != ""]
      body.lower <- tolower(body.tmp)

      body.token <- unlist(strsplit(body.lower, "¥¥W"))
      body.token <- body.token[body.token != ""]
      body.type <- unique(body.token)

      Token <- length(body.token)
      Type <- length(body.type)
      NoS <- length(body.tmp)
      TTR <- Type/Token
      GI <- Type/sqrt(Token)
      AWL <- nchar(paste(body.token, collapse=""))/Token
      ASL <- Token/NoS

      cat(Token, Type, NoS, TTR, GI, AWL, ASL, "¥n", file=output.file, append=T)
      }
}
-----------------------------------------------------
> jpn.indices <- read.table(choose.files())
> names(jpn.indices) <- c("Token","Type","NoS","TTR","GI","AWL","ASL")
> head(jpn.indices)
  Token Type NoS       TTR       GI      AWL      ASL
1   319  135  30 0.4231975 7.558549 4.304075 10.63333
2   356  161  29 0.4522472 8.532983 4.233146 12.27586
3   201  121  13 0.6019900 8.534682 4.746269 15.46154
4   260  140  27 0.5384615 8.682431 4.761538  9.62963
5   420  175  25 0.4166667 8.539126 3.995238 16.80000
```

■Criterion のスコアをヘッダー情報から取り入れる

★ヘッダー情報

```
-------------------------------------------------
@Begin
@Participants:    JPN501
@PID: PIDJP501
@Age: 21
 (中略)
@TopicEase: 4
@Topic:     sports
@Criterion: 4
-------------------------------------------------
```

１）属性情報のなかで Criterion のところから値だけ取り出す。

２）@Criterion の行に着目　＝　その行を取り出して調べる

「@Criterion」という文字列を検索　＝＞　grep

(lines.tmp に一文一行でデータが入っているとして)

```
criterion.tmp <- grep("@Criterion", lines.tmp, value = T)
```

３）その行の値のところだけ残して前を消す

```
score <- gsub("@Criterion:¥t", "", criterion.tmp)
```

■今日の課題

１）これまでの言語指標に加え MATTR も出力できるようにプログラムを修正してみよう。

１）言語指標に加えて Criterion のスコアも併せて出力するプログラムを作ってみよう。

２）エッセイのスコアと、各種言語指標とはどんな関係になっていると考えられるだろうか。

■2019-12-13

■myTextIndex2.R をダウンロードして保存する。

source(choose.files()) で、保存したファイルを読み込む。

■もしくは、インターネット上のソースファイルを呼び込む。 sorce("アドレス")

source("http://sugiura-ken.org/wiki/wiki.cgi/exp?page=myTextIndex%2ER&file=myTextIndex2%2ER&action=ATTACH")

myTextIndex2()

```
> JPNindex <- read.table(choose.files())
> head(JPNindex)
         V1 V2   V3  V4  V5        V6       V7        V8       V9       V10
1 JPN501.txt  4  994 260 123 0.2615694 8.246699 0.4919115 3.888330 8.081301
2 JPN502.txt  4  997 283 120 0.2838516 8.962700 0.5160281 3.925777 8.308333
3 JPN503.txt  3  561 207  70 0.3689840 8.739547 0.5566488 4.247772 8.014286
4 JPN504.txt  4  817 245 114 0.2998776 8.571465 0.4998409 4.057528 7.166667
5 JPN505.txt  4 1024 274 106 0.2675781 8.562500 0.5182812 3.676758 9.660377
6 JPN506.txt  3  638 197  93 0.3087774 7.799305 0.5082132 3.578370 6.860215
```

```
> names(JPNindex) <- c("file", "Score", "Token","Type","NoS","TTR","GI", "MATTR", "AWL","ASL")
```

```
> head(JPNindex)
       file Score Token Type NoS       TTR       GI     MATTR      AWL      ASL
1 JPN501.txt     4   994  260 123 0.2615694 8.246699 0.4919115 3.888330 8.081301
2 JPN502.txt     4   997  283 120 0.2838516 8.962700 0.5160281 3.925777 8.308333
3 JPN503.txt     3   561  207  70 0.3689840 8.739547 0.5566488 4.247772 8.014286
4 JPN504.txt     4   817  245 114 0.2998776 8.571465 0.4998409 4.057528 7.166667
5 JPN505.txt     4  1024  274 106 0.2675781 8.562500 0.5182812 3.676758 9.660377
6 JPN506.txt     3   638  197  93 0.3087774 7.799305 0.5082132 3.578370 6.860215
```

■Topic も情報にあった方がよい

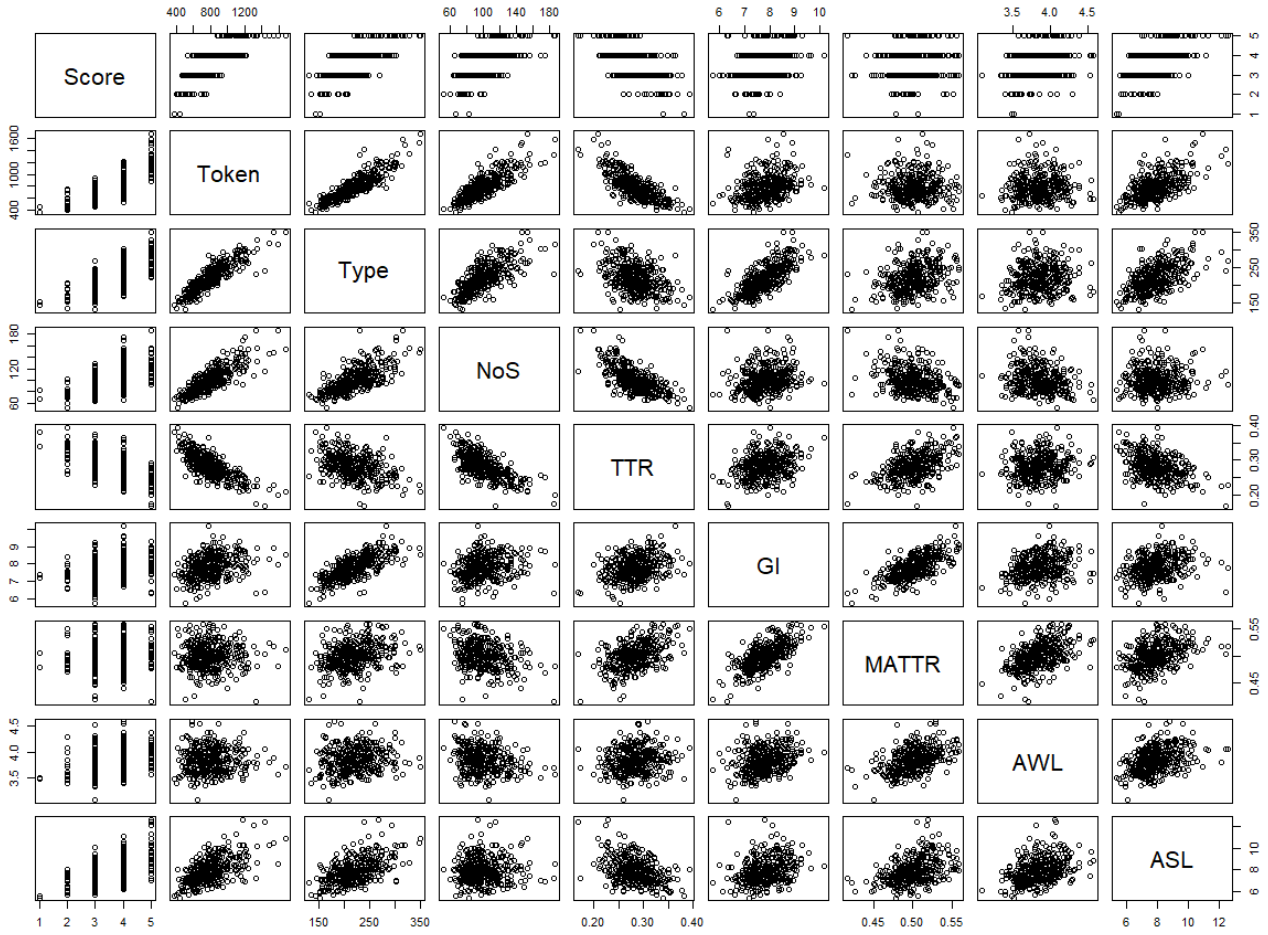
```
topic.tmp <- grep("@Topic:¥t", lines.tmp, value = T)
topic <- gsub("@Topic:¥t", "", topic.tmp)
```

```
> source(choose.files())

> myTextIndexTopic()

> JPNindexTopic <- read.table(choose.files())

> names(JPNindexTopic) <- c("file", "Topic", "Score", "Token", "Type", "NoS", "TTR",
"GI", "MATTR", "AWL", "ASL")

> head(JPNindexTopic)
       file     Topic Score Token Type NoS       TTR       GI     MATTR      AWL      ASL
1 JPN501.txt    sports     4   994  260 123 0.2615694 8.246699 0.4919115 3.888330 8.081301
2 JPN502.txt education     4   997  283 120 0.2838516 8.962700 0.5160281 3.925777 8.308333
3 JPN503.txt education     3   561  207  70 0.3689840 8.739547 0.5566488 4.247772 8.014286
4 JPN504.txt    sports     4   817  245 114 0.2998776 8.571465 0.4998409 4.057528 7.166667
5 JPN505.txt    sports     4  1024  274 106 0.2675781 8.562500 0.5182812 3.676758 9.660377
6 JPN506.txt     money     3   638  197  93 0.3087774 7.799305 0.5082132 3.578370 6.860215
```

■指標間の相関関係を見る散布図

```
> pairs(JPNindexTopic[ , 3:11])
```

★要素番号の指定の仕方に注意：　[行，列] で指定。
　１）行を空欄にする＝行については指定しない＝すべての行
　２）列について、３：１１　と指定＝３列目から１１列目まで



■相関係数

```
> cor(JPNindexTopic[ , 3:11])
      Score       Token       Type         NoS         TTR         GI       MATTR         AWL         ASL
Score     1          NA         NA          NA          NA         NA          NA          NA          NA
Token    NA 1.000000000  0.8834177  0.82325546 -0.7598818  0.3337103 -0.009647251  0.02643039  0.59849658
Type     NA 0.883417718  1.0000000  0.70261532 -0.4092411  0.7310491  0.317602053  0.14620073  0.57191896
NoS      NA 0.823255458  0.7026153  1.00000000 -0.6714658  0.2213156 -0.309020421 -0.22461770  0.05241909
TTR      NA -0.759881781 -0.4092411 -0.67146576  1.0000000  0.3106524  0.477444028  0.13280915 -0.41738091
GI       NA 0.333710295  0.7310491  0.22131559  0.3106524  1.0000000  0.682911462  0.25963190  0.28253707
MATTR    NA -0.009647251  0.3176021 -0.30902042  0.4774440  0.6829115  1.000000000  0.48604502  0.41597725
AWL      NA 0.026430388  0.1462007 -0.22461770  0.1328091  0.2596319  0.486045024  1.00000000  0.38267891
ASL      NA 0.598496582  0.5719190  0.05241909 -0.4173809  0.2825371  0.415977247  0.38267891  1.00000000
```

★あれ、なんか変だ。計算されていない＝「欠損値」があるかも

```
> anyNA(JPNindexTopic)
[1] TRUE
```

★cor()のオプションで、欠損値を除いて計算するものがある。

```
> cor(JPNindexTopic[ ,3:11], use="complete.obs")
          Score       Token       Type         NoS         TTR         GI       MATTR         AWL         ASL
Score 1.0000000  0.78221275  0.7317634  0.58329739 -0.5867421  0.3285678  0.12324002  0.28396471  0.58019072
Token 0.7822127  1.00000000  0.8828587  0.82230906 -0.7586738  0.3327915 -0.01079751  0.02490729  0.59834700
Type  0.7317634  0.88285867  1.0000000  0.70121175 -0.4064136  0.7311909  0.31730729  0.14480423  0.57148779
NoS   0.5832974  0.82230906  0.7012118  1.00000000 -0.6695073  0.2202444 -0.31114026 -0.22676094  0.05058879
TTR  -0.5867421 -0.75867376 -0.4064136 -0.66950729  1.0000000  0.3134123  0.47987940  0.13424715 -0.41699144
GI    0.3285678  0.33279146  0.7311909  0.22024436  0.3134123  1.0000000  0.68283595  0.25863007  0.28171006
MATTR 0.1232400 -0.01079751  0.3173073 -0.31114026  0.4798794  0.6828359  1.00000000  0.48608059  0.41570957
AWL   0.2839647  0.02490729  0.1448042 -0.22676094  0.1342472  0.2586301  0.48608059  1.00000000  0.38189501
ASL   0.5801907  0.59834700  0.5714878  0.05058879 -0.4169914  0.2817101  0.41570957  0.38189501  1.00000000
```

★小数点以下が長すぎるとわかりにくい。

```
> round(cor(JPNindexTopic[ , 3:11], use="complete.obs"), digits=2)
      Score Token  Type   NoS   TTR   GI MATTR   AWL   ASL
Score  1.00  0.78  0.73  0.58 -0.59 0.33  0.12  0.28  0.58
Token  0.78  1.00  0.88  0.82 -0.76 0.33 -0.01  0.02  0.60
Type   0.73  0.88  1.00  0.70 -0.41 0.73  0.32  0.14  0.57
NoS    0.58  0.82  0.70  1.00 -0.67 0.22 -0.31 -0.23  0.05
TTR   -0.59 -0.76 -0.41 -0.67  1.00 0.31  0.48  0.13 -0.42
GI     0.33  0.33  0.73  0.22  0.31 1.00  0.68  0.26  0.28
MATTR  0.12 -0.01  0.32 -0.31  0.48 0.68  1.00  0.49  0.42
AWL    0.28  0.02  0.14 -0.23  0.13 0.26  0.49  1.00  0.38
ASL    0.58  0.60  0.57  0.05 -0.42 0.28  0.42  0.38  1.00
```

■単回帰直線　Linear Model

y = ax + b

因果関係が想定されている。x が増えると、その結果 y が増える。

y が増える要因（原因）は一つ x だけ。（単回帰分析）

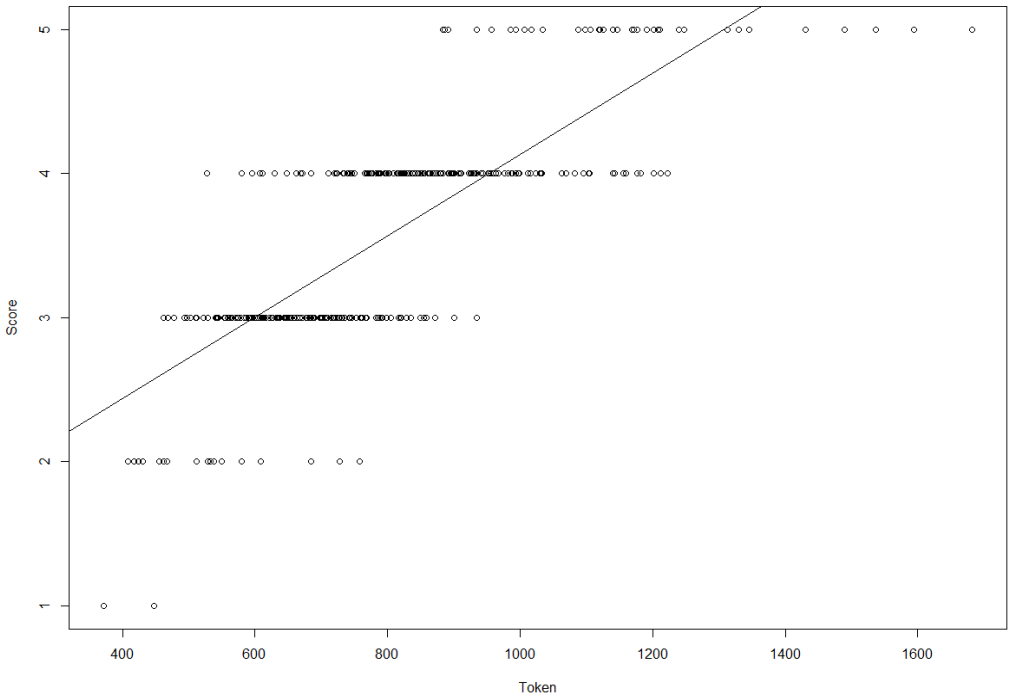
１）二種類のデータをプロットする　（例えば、Score と Token）

２）単回帰直線のモデルを作る

３）モデルをプロットする

４）概要を見る

```
> plot(Score ~ Token, data=JPNindexTopic)
> lm.result <- lm(Score ~ Token, data=JPNindexTopic)
> abline(lm.result)
> summary(lm.result)

Call:
lm(formula = Score ~ Token, data = JPNindexTopic)

Residuals:
     Min       1Q   Median       3Q      Max
-1.56905 -0.32869  0.01202  0.32700  1.20450

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 1.3037488  0.1002986   13.00   <2e-16 ***
Token       0.0028306  0.0001214   23.32   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.4782 on 345 degrees of freedom
  (2 observations deleted due to missingness)
Multiple R-squared:  0.6119,    Adjusted R-squared:  0.6107
F-statistic: 543.8 on 1 and 345 DF,  p-value: < 2.2e-16
```



■重回帰分析

http://sugiura-ken.org/wiki/wiki.cgi/exp?page=Multiple+Regression+Analysis

$y = ax_1 + bx_2 + cx_3 + \ldots + e$

y が増える原因は複数ある。複数の要因が影響した結果が y となる。

関数は lm()

lm(従属変数 ~ 独立変数１ + 独立変数２ + 独立変数３ + ...)

lm(Score ~ Token + Type + NoS + TTR + GI + MATTR + AWL + ASL)

結果を詳しく見るには、一旦保存して、その summary()を見る。

jukaiki <- lm(Score ~ Token + Type + NoS + TTR + GI + MATTR + AWL + ASL)

★変数の規模・単位が違うので、影響を等しく比べられるように「標準化」する。
関数は scale()

★ところが、独立変数間でお互いに相関が高いものがある。
　複数の相関が高いものを重ねて入れてしまうと、その影響が重複して評価されてしまう。

VIF(Variance Inflation factor)をチェック (10 以上はダメ、2 以下がよい)
組み合わせによるので、試行錯誤が必要。
car パッケージをインストールし、library(car)をする。
関数は vif()

★変数選択は、step()関数を使う

■頻度の検定　カイ自乗検定

例：　副詞の生起位置

a. However, such men don't make good husbands.
b. Such men, however, don't make good husbands.
c. Such men don't, however, make good husbands.
d. Such men don't make good husbands, however.
(Halliday, 1985: 82)

文頭・文中・文末のどこにでも置くことができる。

実際は、どうなのか、コーパスで観察する。

#LOB コーパス中の however の生起位置を調べたところ、次のような結果になった。

文頭 109
文中 347
文末 8

はて、本当にどこに置いてもよいのか、それとも、「偏り」があるのか。

頻度の差が、偶然による誤差のうちなのか、誤差とは言えないのか、カイ自乗検定で調べる。

```
--------------------
> however.data <- c(109, 347, 8)

> chisq.test(however.data)

--------------------
Chi-squared test for given probabilities
data: however.data
X-squared = 391.7371, df = 2, p-value < 2.2e-16
--------------------
```

#2.2e-16 意味は、2.2×10^{-16}　（つまりほとんどゼロ）

このバラつきは、偶然の誤差とは言えない。

★文末は、少ないのでほとんど使わないとして、文頭と文中の使い分けに何か原則があるのだろうか？

Sugiura, M. (1991：平成 3年 3月 1日). The Distribution Environment of the Connective ″However″ and the Principle of Its Position ---Based on the LOB Corpus---. 『中部大学女子短期大学紀要・言語文化研究』, 2, 47-63.

https://ci.nii.ac.jp/lognavi?name=nels&lang=ja&type=pdf&id=ART0000872455&naid=110000483331

■二次元の頻度差を考える。

イギリス英語とアメリカ英語で、 therefore の生起位置に違いがあるか。

位置　英　米
文頭　15　38
文中　96　53

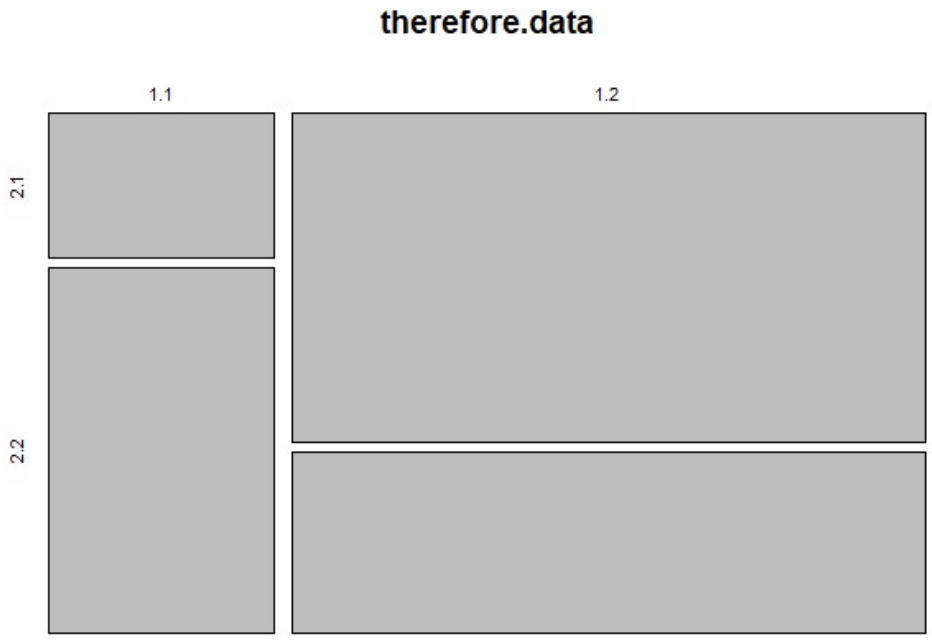
これもカイ自乗検定。

```
--------------------
> therefore.data <- matrix(c(15,96,38,53), nrow=2, ncol=2)

#matrix で行例作成。nrow は行数、ncol は列数。この場合は2×2
# c(  ,  ,  ,  )の中のデータは、まず列(row)順に並べていく

--------------------

     [,1]  [,2]
[1,]   15   38
[2,]   96   53

--------------------

> chisq.test(therefore.data)

-------------------------結果の出力-----------------------
Pearson's Chi-squared test with Yates' continuity correction
data:  therefore.data
X-squared = 19.1788, df = 1, p-value = 1.190e-05
------------------------------------------------------------------
```

（結果の解釈例）
イギリス英語とアメリカ英語で、therefore の生起位置に有意な偏りがある。

■可視化する mosaicplot()

```
> mosaicplot(therefore.data)
```



■どこに差があるかは「残差分析」をする。

★ところが、標準のパッケージには残差分析の関数は含まれていない。

他の人が作ったものを使わせてもらう。

１）ファイルをダウンロードして、手元に保存して、それを利用。
関数をテキストファイルで保存して、拡張子を「.R」にする。

source(choose.files()) で、ファイルを読み込む。

２）オンラインのファイルをそのまま URL で指定して読み込む。

群馬大学の青木先生のサイト
http://aoki2.si.gunma-u.ac.jp/R/

度数に関する検定
カイ二乗分布を使用する独立性の検定と残差分析

```
> source("http://aoki2.si.gunma-u.ac.jp/R/src/my-chisq-test.R", encoding="euc-jp")
```

★my-chisq-test.R というファイル名だが、関数名は my.chisq.test()

分析したい数値を関数に送り、結果を変数 a に代入。

```
> a <- my.chisq.test(therefore.data)
```

```
----------------------結果の出力----------------------
カイ二乗分布を用いる独立性の検定（残差分析）
data: therefore.data
X-squared = 20.6124, df = 1, p-value = 5.623e-06
------------------------------------------------------
```

```
# 自動的には残差分析まで出力してくれない。以下を実行。
> summary(a)

----------------------結果の出力----------------------
調整された残差
     [,1]   [,2]
[1,] -4.5401  4.5401
[2,]  4.5401 -4.5401

p 値
     [,1]       [,2]
[1,] 5.6231e-06 5.6231e-06
[2,] 5.6231e-06 5.6231e-06
------------------------------------------------------
```

（結果の解釈例）
カイ二乗検定の結果より、イギリス英語とアメリカ英語で、therefore の生起位置に有意な偏りがある。残差分析の結果より、イギリス英語では、アメリカ英語に比べて、therefore を文中で用いる頻度が有意に高く、文頭で用いる頻度が有意に低いことが分かる。

■対数尤度比（G スコア）
#カイ二乗検定よりも統計理論的には、対数尤度比検定を利用する方が望ましい。
（カイ二乗検定は対数尤度の近似、対数尤度比検定は対数尤度を直接扱う）

```
> source("http://aoki2.si.gunma-u.ac.jp/R/src/G2.R", encoding="euc-jp")
```

４．分析したい数値を関数に送り、結果を変数 a に代入

```
> G2(therefore.data)
```

```
-------------------------結果の出力----------------------
対数尤度比に基づく独立性の検定 (G-squared test)
data:  therefore.data
G-squared = 20.9249, df = 1, p-value = 4.776e-06
---------------------------------------------------------------
```

（結果の解釈例）
p< .001 より、イギリス英語とアメリカ英語で、therefore の生起位置に有意な偏りがある。

■オッズ比（Odds ratio）：確率の比率　２×２
★サンプルサイズに関係ない「効果量」としての指標

Fisher の直接確率検定（正確確率検定）で計算。

解釈の仕方：「１」が基準＝確率に差はない。１より大きければ、分子の方の確率が高い。

```
> fisher.test(therefore.data)

        Fisher's Exact Test for Count Data

data:  therefore.data
p-value = 9.958e-06
alternative hypothesis: true odds ratio is not equal to 1
95 percent confidence interval:
 0.1021974 0.4526589
sample estimates:
odds ratio
 0.2196899
```

■２×２以上の場合：クラメールの V（Cramer's V）
★サンプルサイズに関係ない「効果量」としての指標

解釈の仕方：０から１の間。

```
> install.packages("lsr", dependencies = T)

> library(lsr)
> cramersV(therefore.data)
[1] 0.3081308
> tmp.data <- c(38, 15, 53, 96)
> cramersV(tmp.data)
[1] 0.3378448
> tmp.data2 <- c(38, 15, 53, 956)
> cramersV(tmp.data2)
[1] 0.8674172
```

■コーパス分析パッケージを使う

例1）corpus

１）スクリプト単体のファイル　.R

２）一連のスクリプトをまとめたもの（パッケージ）

例：　corpus: Text Corpus Analysis

https://cran.r-project.org/web/packages/corpus/index.html

「パッケージのインストール」で、ほしいものを選択

library()　コマンドで自分の「ライブラリー」に登録して使えるようにする。

> library(corpus)

http://corpustext.com/articles/corpus.html

★具体的には、以下を参照：

http://sugiura-ken.org/wiki/wiki.cgi/exp?page=corpus

例2）quanteda: Quantitative Analysis of Textual Data

★具体的には、以下を参照:

http://sugiura-ken.org/wiki/wiki.cgi/exp?page=quanteda

以上