2021夏学期ゼミ#8 2021/5/7

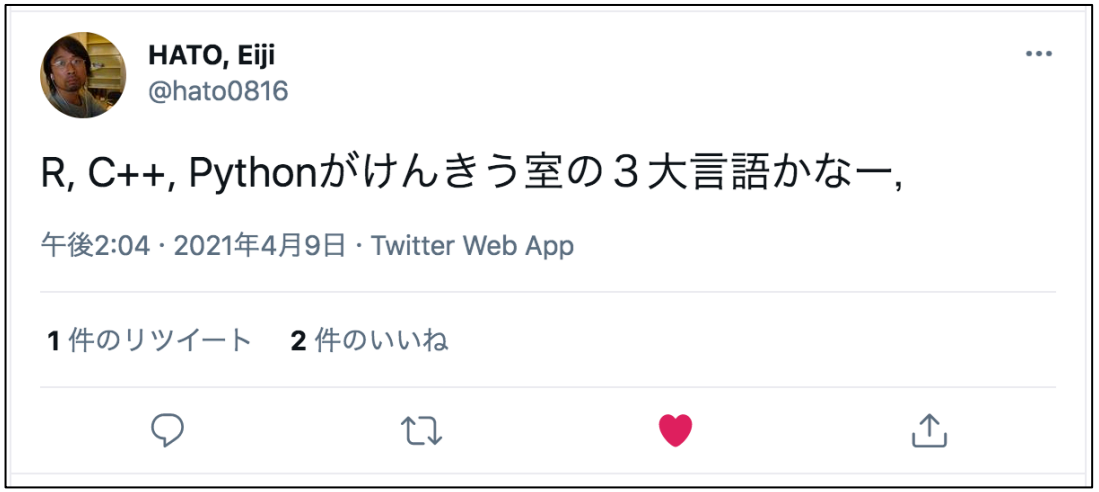
Rの使い方

交通・都市・国土学研究室

月田 光

0. 目次

1. R言語とは



20代使用言語別年収ランキング
(TECH Street調べ)

	言語	平均年収
1位	R	476万円
2位	Scala	440万円
3位	Objective-C	407万円
4位	COBOL	406万円
5位	Perl	405万円
6位	VC・VC++	404万円
7位	Python	403万円
8位	Swift	398万円
9位	Ruby	397万円
9位	Go	397万円
11位	SQL	392万円
12位	PL/SQL	391万円
13位	C++	390万円
14位	C	388万円
15位	VB	382万円
16位	C# .NET	381万円
17位	Java	380万円
18位	JavaScript	378万円
19位	VB.NET	373万円
20位	PHP	361万円
21位	F#	312万円

- 統計処理に特化したプログラミング言語と環境
- オープンソース・フリーソフトウェア
- 豊富な「パッケージ」を利用して、容易に機能拡張が可能
- Tipsがネットに豊富
  - http://cse.naro.affrc.go.jp/takezawa/r-tips/r.html
  - http://www.okadajp.org/RWiki/

- 特徴
  - データの分類・集計・整理が得意
  - ベクトル、行列演算が簡単
  - 初学者に優しい
  - **グラフ等を使った可視化ができる**

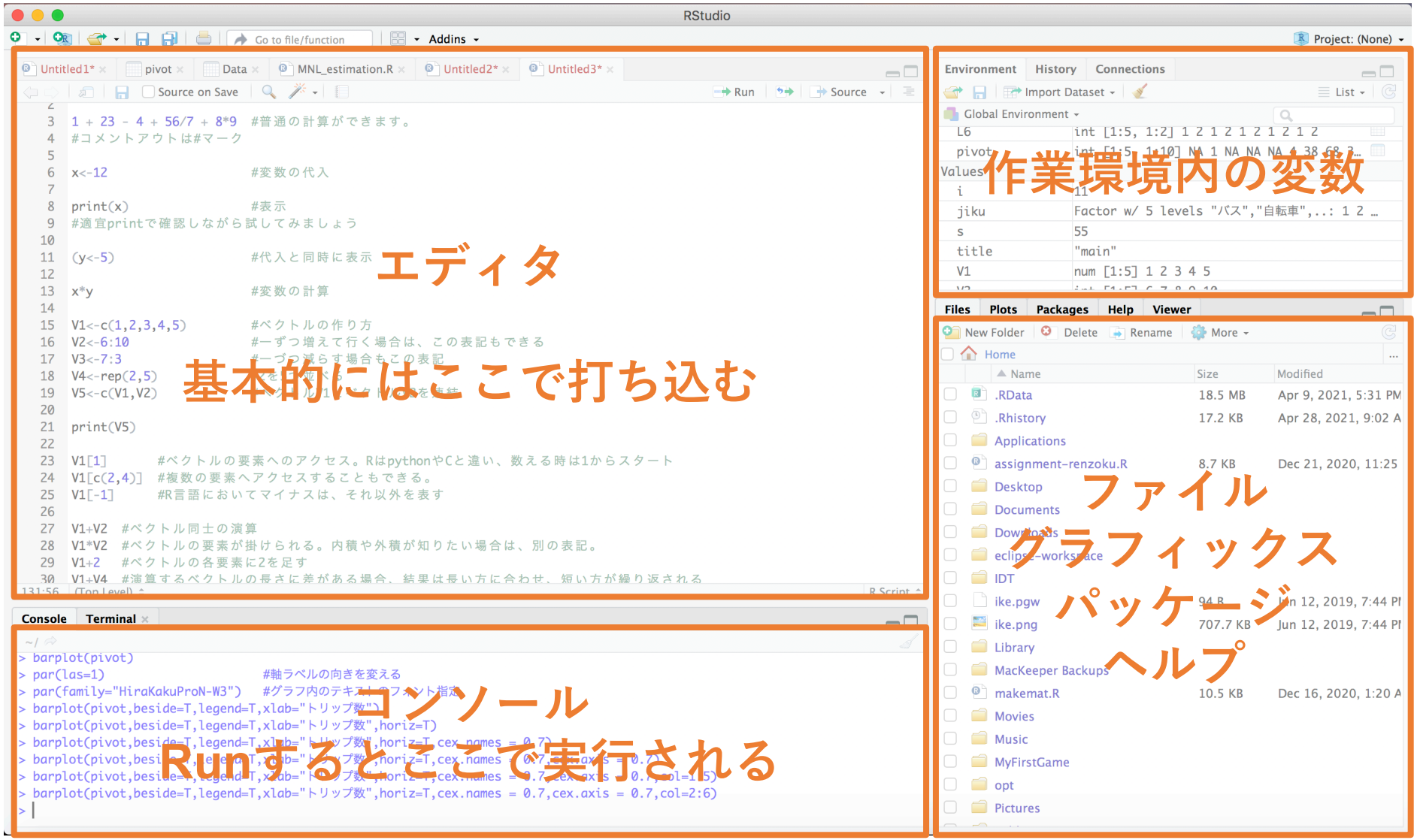
- RとRStudioをインストールしてください

- R本体
  - https://cran.r-project.org/bin/macosx/

- RStudio
  - https://www.rstudio.com/products/rstudio/download/
  - RStudio社が開発したRの綜合開発環境(IDE)
  - FreeのRStudio DesktopでOK

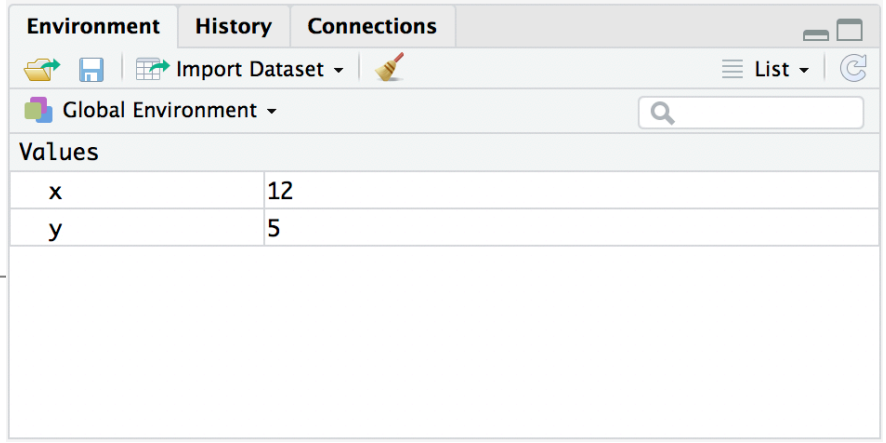
2. RStudioの構成

4つの枠(ペイン;pane)で構成。枠の配置や数はView>Panesから自由に設定可。



3. RStudioの使い方：基本的な計算、変数の代入


```
> 1 + 23 - 4 + 56/7 + 8*9  #普通の計算ができます。
[1] 100
> #コメントアウトは#マーク
>
> x<-12                    #変数の代入は<-
>
> print(x)                 #表示
[1] 12
> #適宜printで確認しながら試してみましょう
>
> (y<-5)                   #代入と同時に表示
[1] 5
>
> x*y                      #変数の計算
[1] 60
```



右上のペインに値が入っているのを確認しましょう

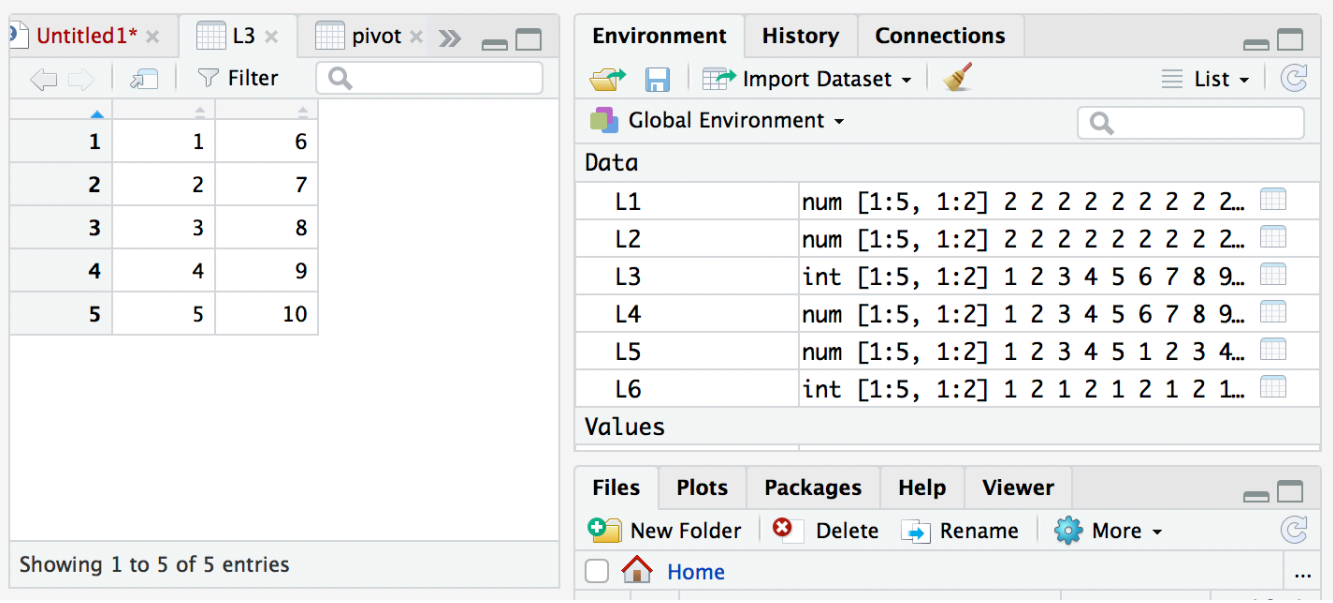
```
> rm(list=ls())   #変数が溜まってきたら削除
```

3. RStudioの使い方：ベクトル


```
> V1<-c(1,2,3,4,5)        #ベクトルの作り方
> V2<-6:10                #一ずつ増えて行く場合は、この表記もできる
> V3<-7:3                 #一づつ減らす場合もこの表記
> V4<-rep(2,5)            #2を5つ並べる
> V5<-c(V1,V2)            #ベクトルV1とベクトルV2を連結
>
> print(V5)
 [1]  1  2  3  4  5  6  7  8  9 10
>
> V1[1]        #ベクトルの要素へのアクセス。RはpythonやCと違い、数える時は1からスタート
[1] 1
> V1[c(2,4)]  #複数の要素へアクセスすることもできる。
[1] 2 4
> V1[-1]       #R言語においてマイナスは、それ以外を表す
[1] 2 3 4 5
>
> V1+V2                #ベクトル同士の演算
[1]  7  9 11 13 15
> V1*V2                #ベクトルの要素が掛けられる。内積や外積が知りたい場合は、別の表記。
[1]  6 14 24 36 50
> V1+2                 #ベクトルの各要素に2を足す
[1] 3 4 5 6 7
> V1+V4        #演算するベクトルの長さに差がある場合、結果は長い方に合わせ、短い方が繰り返される
[1] 3 4 5 6 7
```

3. RStudioの使い方：行列


```
> L1<-matrix(data=2,nrow=5,ncol=2)  #行列の作り方
> L2<-matrix(2,5,2)                 #引数名は省略しても良い
> L3<-matrix(1:10,5,2)               #行列の要素はベクトルで入れる
> L4<-matrix(V5,5,2)                 #行列の要素はベクトルで入れる
> L5<-matrix(V1,5,2)                 #行列の要素が足りない場合は、繰り返される
> L6<-matrix(1:2,5,2)                #行列の要素が足りない場合は、繰り返される
>
> print(L5)
     [,1] [,2]
[1,]    1    1
[2,]    2    2
[3,]    3    3
[4,]    4    4
[5,]    5    5
```



作った行列が右上のペインのData欄にある。
これを選択すると左上のペインの現れる。

3. RStudioの使い方：関数


```
> Square<-function(x){            #関数の作り方
+   return(x*x)                   #returnで返す値
+ }
>
> Square2<-function(x,y){       #2変数でも良い
+   return(x*x+y)
+ }
>
> z<-Square(5)                  #こんな感じで使う
> print(z)
[1] 25
> print(Square(5))              #直接printしても良い
[1] 25
> print(Square(V1))             #ベクトルを引数に入れてもいい
[1]  1  4  9 16 25
> print(Square(L3))             #行列を入れてもちゃんと返してくれる
     [,1] [,2]
[1,]    1   36
[2,]    4   49
[3,]    9   64
[4,]   16   81
[5,]   25  100
>
```

作った関数も右上のペインのData欄に現れる。

3. RStudioの使い方：if文


```
> IsOdd<-function(x){
+   if(x%%2==0){          #if文、%%は余りを求める演算
+     return(TRUE)
+   }else{
+     return(F)           #T/Fの一文字でも良い
+   }
+ }
>
> IsOdd(6)
[1] TRUE
>
> IsOdd2<-function(x){
+   return(x%%2==0)       #実はTRUE/FALSEだけならこの一文で出すことができる
+ }
>
> IsOdd3<-function(x){
+   if(IsOdd(x)){         #関数の中に関数を組み込んでも良い
+     return(x/2)
+   }else if(x>=0){       #三つ以上の条件分岐は、else ifを使う(elifは使えない)
+     return(x-1)
+   }else{
+     return(x+1)
+   }
+ }
>
```

3. RStudioの使い方：比較演算子、論理演算


```
> y>=0          #TRUE/FALSEで教えてくれる
[1] TRUE
> y<0           #==,<=,>=,<,>,!=といった比較演算子が使える
[1] FALSE
> y!=0          #!=はnot equal
[1] TRUE
>
> y>=0&&x>=0     #複数条件かつ&&
[1] TRUE
> y>=10||x>=10   #複数条件または||
[1] TRUE
> !y>=10         #否定!
[1] TRUE
>
> V2>=V3         #要素ごとに比較
[1] FALSE  TRUE  TRUE  TRUE  TRUE
> (V2>=7)&(V3>=6) #ベクトルの場合、&、|を使う
[1] FALSE  TRUE FALSE FALSE FALSE
>
```

比較演算子	
==	等しい
!=	等しくない
>=,<=	≥, ≤
>,<	>,<

論理演算(値同士)	
!	NOT(でない)
&&	AND(かつ)
||	OR(または)

論理演算(ベクトル)	
&	AND(かつ)
|	OR(または)

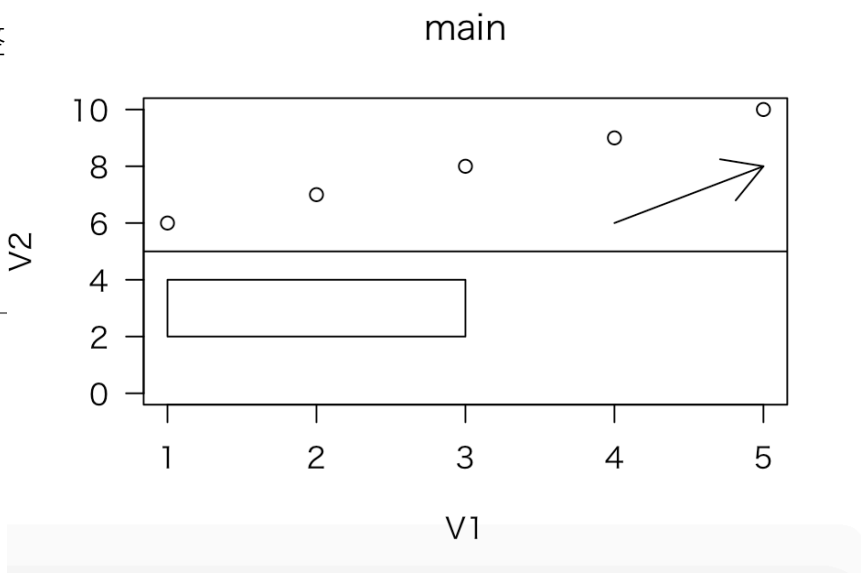
3. RStudioの使い方：for文、while文


```
> s<-0
> for(i in 1:10){     #繰り返し文for(ループ  in  リスト)   リストはベクトルの形で渡す
+   s<-s+i            #繰り返す式を{}の中に書く。ループ変数がリストの範囲内である限り式が繰り返される
+ }
> print(s)
[1] 55
>
> s<-0
> i<-0
> while(i<=10){   #while文
+   s<-s+i
+   i<-i+1
+ }
>
> s<-0
> i<-0
> while(TRUE){    #while文、TRUEにして回し、中身でbreakする方法もある。
+   s<-s+i
+   i<-i+1
+   if(i>10){break}
+ }
>
```

3. RStudioの使い方：グラフ作成

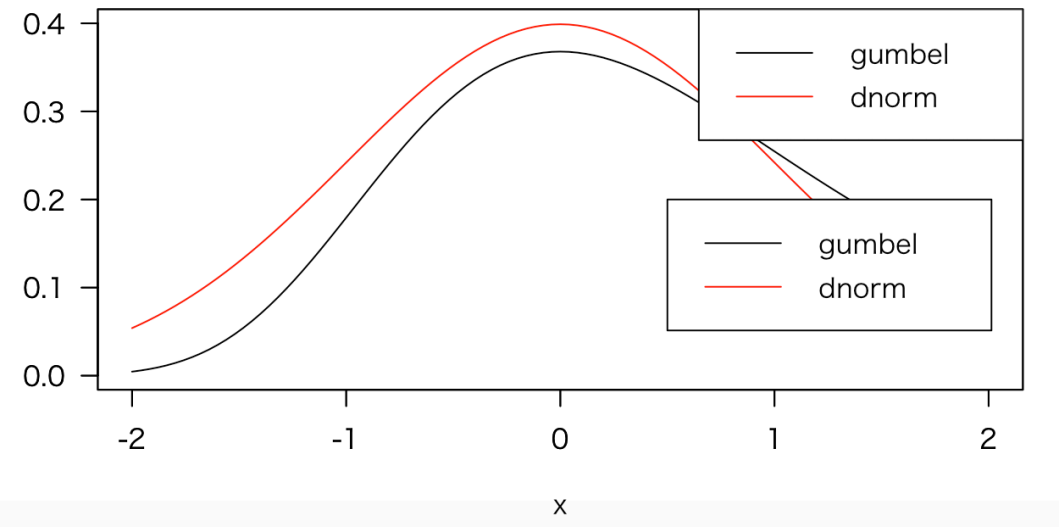
- 高水準作図関数:1枚の完成されたグラフを描く
  - 散布図plot()、ヒストグラムhist()、棒グラフbarplot()、円グラフpie()
  - 一次元の関数グラフcurve(関数,左端,右端)、二次元関数のグラフpersp(x軸,y軸,z軸)など
  - 他にも良さげなグラフがたくさん
    - http://cse.naro.affrc.go.jp/takezawa/r-tips/r/50.html
    - https://www.r-graph-gallery.com/all-graphs
- 低水準作図関数:完成されたグラフに図形や文字を追記

```
> #まず散布図を作る_とりあえずplot
> plot(1:10)
> plot(V1,V2)                  #x軸にV1、y軸にV2
> plot(V1,V2,ylim=c(0,10))     #y軸が気持ち悪かったので調整
> #グラフを装飾
> abline(h=5)                  #直線を追記
> rect(1,2,3,4)                #(1,2)から(3,4)へ矩形を書く
> arrows(4,6,5,8)              #(4,6)から(5,8)へ矢印を書く
> title("main")                #タイトルを追記
```



3. RStudioの使い方：グラフ作成2


```
> #1次元グラフ
> gumbel<-function(x){                    #まず適当に関数を作って
+   return(exp(-x)*exp(-(exp(-x))))
+ }
> plot(gumbel,-2,2)                       #(関数、下限、上限)でplot
> par(new=T)                              #上書き指定する
> plot(dnorm,-2,2, col=2)                 #標準正規分布を重ねた。これだけではy軸がバラバラ
>
> plot(gumbel,-2,2,ylim=c(0,0.4), ylab="", ann=F)       #ylimを設定、ann(軸とラベルの出力)をFに
> par(new=T)                                            #ylabを非表示
> plot(dnorm,-2,2, col="Red",ylim=c(0,0.4), ylab="")    #colは指定されたものなら色名でも可
> legend("topright",legend=c("gumbel","dnorm"),col=1:2,lty=1)   #凡例を表示
> legend(0.5,0.2,legend=c("gumbel","dnorm"),col=1:2,lty=1)      #位置で指定もできる
>
```



4. データ処理をしてみよう：下準備、読み込み

data.frame
- 数値ベクトル、文字ベクトル、因子ベクトルなど異なる型のデータをまとめて1つの変数として持つdata.frameという型が統計処理に便利。
- 各行・列にラベルをつけることができ、ラベルによる操作が可能。

作業ディレクトリ
- ファイルからデータやプログラムを読み込んだり、ファイルに書き込んだりする場所。

```
> getwd()                          #作業ディレクトリの確認
[1] "/Users/study"
> setwd("/Users/study/Downloads")   #作業ディレクトリの変更
> ### データファイルの読み込み
> Data <- read.csv("/Users/study/Downloads/YokohamaData.csv",header=T,fileEncoding = "cp932")
> #データの情報
> nrow(Data)   #行数
[1] 1522
> ncol(Data)   #列数
[1] 62
> colnames(Data)  #列名
```

4. データ処理をしてみよう：データへのアクセス


```
#データへのアクセス
Data[4]                                          #4列目を表示
Data$PurposeJP                                   #列名でも良い。$マークをつける
Data[c(3:5,31)]                                  #複数列を選んでも良い。
Data[1,4]                                        #要素にアクセスする時は行、列の順番で
Data[1,]                                         #カンマの後を省略すると全ての列に
Data[c(1:5,10,12),c(4,5)]                        #複数の要素にアクセス
Kitaku<-Data$PurposeJP=="帰宅"                   #T/Fのベクトルになる
print(Kitaku)
sum(Kitaku)                                      #帰宅の数(TRUEの数)を数えることができる
Data[Kitaku,]                                    #TRUEの行だけ表示
Data[Data$PurposeJP=="帰宅",]                #いきなりこう書いても同じ。つまり、目的が帰宅の行だけ表示
Data[Data$PurposeJP=="帰宅"&Data$TripDurationSec>=5000,]       #複数条件
Data[Data$PurposeJP==“帰宅”&Data$TripDurationSec>=5000,MainModeJP,drop=F]
                                                 #drop=Fを置いておくと一列でもベクトルに変化しない
subset(Data,PurposeJP=="買い物",c(PurposeJP,MainModeJP))       #subsetを使うこともできる

#基本操作
娯楽<-ifelse((Data$PurposeJP==“買い物”|Data$PurposeJP==“娯楽"),1,0)          #ベクトルを作成
Data<-cbind(Data,娯楽)                                                        #横に連結
Data$DepartureTime<-as.POSIXct('1899-12-30') + as.difftime(Data$DepartureTime, units = 'days')
                                                                              #要素の更新
Data[,7]<-as.POSIXct('1899-12-30') + as.difftime(Data[,7], units = 'days')    #時刻の型を直した
```

4. データ処理をしてみよう：ピボットグラフ


```
#ピボットグラフを作る
#install.packages("dplyr")          #初回のみ必要
#install.packages("tidyr")          #同上
library("dplyr")                    #基本操作を含むデータ処理が簡単にできる
library("tidyr")
#まず個数を集計
pivot<-Data%>%                      #%>%はdplyrで複数の処理を連結する演算子。左の値が右の値の第一引数。
  dplyr::group_by(PurposeJP,MainModeJP)%>%          #軸にしたい二つの変数でgroup_by
  dplyr::summarize(データ数=n())%>%  #dplyr中にあるn()という関数でデータ数をカウント
  tidyr::spread(PurposeJP,データ数)    #列にしたい軸変数名と、summarizeの中で設けた集計変数名を指定


#作った表(pivot)を整形
jiku<-pivot$MainModeJP              #1列目が行名なので、記録しておいて削除
pivot<-pivot[,-1]
pivot<-as.matrix(pivot)             #barplotはmatrix型じゃないといけないので
rownames(pivot)<-jiku
```

	ｻｲｸﾘﾝｸﾞ	その他	帰社	帰宅	業務	娯楽	散歩・回遊	出勤	食事	買い物
バス	NA	4	1	12	3	1	2	14	NA	4
自転車	1	38	9	70	13	6	4	39	8	23
自動車	NA	68	3	204	6	14	7	121	28	61
鉄道	NA	33	10	175	64	21	4	129	24	68
徒歩	NA	16	9	68	17	9	9	11	44	47

4. データ処理をしてみよう：ピボットグラフ


```
#いよいよ描写
> par(las=1)                                    #軸ラベルの向きを変える
> par(family="HiraKakuProN-W3")                 #グラフ内のテキストのフォント指定
> barplot(pivot,beside=T,legend=T,xlab="トリップ数",horiz=T,cex.names = 0.7,cex.axis = 0.7,col=2:6)
```