第8回 大規模データを用いたデータフレーム操作実習(2)

2023年2月27日

目 次

1 ここで学ぶ事 **1**

2 データの準備 **1**

2.1 練習問題 2

2.2 真偽値への変更 3

3 因子型（Facotr）とは **4**

3.1 整数値符号から Factor の生成方法 5

3.2 Factor の比較 6

3.3 順序を持つ Factor の生成方法 7

3.4 長さが異なるベクトルの比較について 8

3.5 練習問題 9

4 データの条件抽出 **9**

4.1 練習問題 10

4.2 `subset()` 関数を用いたデータ抽出 10

4.3 練習問題 13

5 リスト型 **13**

5.1 データフレームとリスト 16

5.2 練習問題 17

5.3 リストの連結 17

6 データの保存 **23**

1 ここで学ぶ事

- 新たに Factor と List の 2 つのデータ型を学ぶ.
- 大規模擬似ミクロデータを用いてデータクリーニング（整形）の練習を行う.
- `subset()` 関数を使った条件抽出方法を学ぶ.

2 データの準備

データは第 5 回で使用した独立行政法人統計センターの大規模疑似ミクロデータを使用する．

```
> load("Microdata.RData")  # E ドライブないなら"E:/Microdata.RData"
> ls()                     # microdata オブジェクトが読み込まれた
[1] "microdata"
> dim(microdata)
[1] 45811    20
```

データ数は 4 万 5 千件を超える大規模データであることが分かる．次にデータの構造を見てみる．

```
> str(microdata)
'data.frame':        45811 obs. of  20 variables:
 $ X3City        : int  1 1 1 1 1 1 1 1 1 1 ...
 $ T_SeJinin     : int  2 2 2 2 2 2 2 2 2 2 ...
 $ T_SyuJinin    : int  1 1 1 1 1 1 1 1 1 1 ...
 $ T_JuSyoyu     : int  1 1 1 1 1 1 1 1 1 1 ...
 $ T_Syuhi       : int  1 1 1 1 1 1 1 1 1 1 ...
 $ T_Age_5s      : int  1 1 1 1 1 1 1 1 1 1 ...
 $ T_Age_65      : int  1 1 1 1 1 1 1 1 1 1 ...
 $ Weight        : num  895 895 895 895 895 ...
 $ Y_Income      : num  3917 6675 2790 3452 9252 ...
 $ L_Expenditure : num  201649 166381 114511 152109 192439 ...
 $ Food          : num  47756 34054 41664 34924 68882 ...
 $ Housing       : num  16028 7416 730 3418 2832 ...
 $ LFW           : num  9652 26313 22358 8131 23042 ...
 $ Furniture     : num  6702 17062 5413 4164 2598 ...
 $ Clothes       : num  8088 6989 1205 6970 5714 ...
 $ Health        : num  726 7637 5049 4247 2052 ...
 $ Transport     : num  21546 20773 17411 47698 37006 ...
 $ Education     : num  0 0 0 0 0 0 0 0 0 0 ...
 $ Recreation    : num  14433 19048 8605 36419 38588 ...
 $ OL_Expenditure: num  76719 27089 12077 6137 11725 ...
```

列名を見ても何のデータを表しているのかよく分からないので，以下の練習問題で適切な列名を設定しよう．

2.1 練習問題

microdata の列名には，符号表の Excel ファイル ippan_2009zensho_z.xls 内の「変数名」が使われている．以下の手順でデータフレームの列名を設定せよ．

(1) 符号表のエクセルファイルを R から読み込み，変数名と項目名をそれぞれ別々のベクトルに抽出せよ．ただし，同じインデックスの要素には対応する変数名と項目名を格納し，NA を取り除くこと．

(2) 符号表の変数名に対応する項目名を，microdata の列名に設定するために，符号表の変数名の順番と microdata の列名の順番が同じかどうか確認せよ．

(3) 上で順番が同じことが確認できたら，それを利用して「項目名」を microdata の列名に設定せよ．

以上の練習問題が無事完了していれば，以下のデータ構造になっているはずである．

```
> str(microdata)
'data.frame':        45811 obs. of  20 variables:
 $ ３大都市圏か否か: int  1 1 1 1 1 1 1 1 1 1 ...
 $ 世帯人員        : int  2 2 2 2 2 2 2 2 2 2 ...
 $ 就業人員        : int  1 1 1 1 1 1 1 1 1 1 ...
 $ 住宅の所有関係  : int  1 1 1 1 1 1 1 1 1 1 ...
 $ 就業・非就業の別: int  1 1 1 1 1 1 1 1 1 1 ...
 $ 年齢階級２      : int  1 1 1 1 1 1 1 1 1 1 ...
 $ 年齢階級１      : int  1 1 1 1 1 1 1 1 1 1 ...
 $ 集計用乗率      : num  895 895 895 895 895 ...
 $ 年間収入        : num  3917 6675 2790 3452 9252 ...
 $ 消費支出        : num  201649 166381 114511 152109 192439 ...
 $ 食料            : num  47756 34054 41664 34924 68882 ...
 $ 住居            : num  16028 7416 730 3418 2832 ...
 $ 光熱・水道      : num  9652 26313 22358 8131 23042 ...
 $ 家具・家事用品  : num  6702 17062 5413 4164 2598 ...
 $ 被服及び履物    : num  8088 6989 1205 6970 5714 ...
 $ 保健医療        : num  726 7637 5049 4247 2052 ...
 $ 交通・通信      : num  21546 20773 17411 47698 37006 ...
 $ 教育            : num  0 0 0 0 0 0 0 0 0 0 ...
 $ 教養娯楽        : num  14433 19048 8605 36419 38588 ...
 $ その他の消費支出: num  76719 27089 12077 6137 11725 ...
```

2.2 真偽値への変更

「３大都市圏か否か」,「住宅の所有関係」,「就業・非就業の別」の値は 1 と 2 で符号化されているが，TRUE，FALSE で表した方が自然である．列名もそれに応じて,「３大都市圏」,「持ち家所有」,「就業中」に変更するのが良いだろう．

変更する列名は 3 つなので手動で変更しても良いが，沢山あるケースはプログラムから自動で処理したい．これまで学んだ方法を使って自動で変更する例を以下に示す．まず，変更する列名の新旧対応関係のベクトルを用意し match() と replace() を用いて一気に変更する．

```
> old <- c("３大都市圏か否か","住宅の所有関係","就業・非就業の別") # 旧列名
> new <- c("３大都市圏","持ち家所有","就業中") # 新列名
> cnames <- colnames(microdata)
> index <- match(old, cnames) # old の各要素と一致する cnames 要素の index を取得
> cnames <- replace(cnames, index, new) # cnames 内の index 番目の要素を new と置き換える
> colnames(microdata) <- cnames # 列名を更新
```

次に，データを logical 型に変換する．多くのプログラミング言語同様，R でも 0 は FALSE に，それ以外の値は TRUE に評価されることを利用する．例えば以下のように 0 から 3 までの整数を as.logical() で真偽値に変換すると，0 以外は TRUE に変換されているのが確認できる．

```
> as.logical(c(0,1,2,3))
[1] FALSE  TRUE  TRUE  TRUE
```

３大都市圏の値は 1 と 0 なので，これをそのまま as.logical() で変更できる．住宅の所有関係と就業・非就業の別は 1 と 2 で符号化されているので 1 と等しいかで変換する．

```
> microdata$３大都市圏 <- as.logical(microdata$３大都市圏)
> microdata$持ち家所有 <- microdata$持ち家所有 == 1
> microdata$就業中 <- microdata$就業中 == 1
> str(microdata)
'data.frame':        45811 obs. of  20 variables:
 $ ３大都市圏      : logi  TRUE TRUE TRUE TRUE TRUE TRUE ...
 $ 世帯人員        : int  2 2 2 2 2 2 2 2 2 2 ...
 $ 就業人員        : int  1 1 1 1 1 1 1 1 1 1 ...
 $ 持ち家所有      : logi  TRUE TRUE TRUE TRUE TRUE TRUE ...
 $ 就業中          : logi  TRUE TRUE TRUE TRUE TRUE TRUE ...
 $ 年齢階級２      : int  1 1 1 1 1 1 1 1 1 1 ...
 $ 年齢階級１      : int  1 1 1 1 1 1 1 1 1 1 ...
 $ 集計用乗率      : num  895 895 895 895 895 ...
 $ 年間収入        : num  3917 6675 2790 3452 9252 ...
 $ 消費支出        : num  201649 166381 114511 152109 192439 ...
 $ 食料            : num  47756 34054 41664 34924 68882 ...
 $ 住居            : num  16028 7416 730 3418 2832 ...
 $ 光熱・水道      : num  9652 26313 22358 8131 23042 ...
 $ 家具・家事用品  : num  6702 17062 5413 4164 2598 ...
 $ 被服及び履物    : num  8088 6989 1205 6970 5714 ...
 $ 保健医療        : num  726 7637 5049 4247 2052 ...
 $ 交通・通信      : num  21546 20773 17411 47698 37006 ...
 $ 教育            : num  0 0 0 0 0 0 0 0 0 0 ...
 $ 教養娯楽        : num  14433 19048 8605 36419 38588 ...
 $ その他の消費支出: num  76719 27089 12077 6137 11725 ...
```

3　因子型（Facotr）とは

Excel や CSV ファイルからデータを読み込む際，文字列データは自動的に因子型（Factor）に変換される．Factor とは離散値を取るデータを表現するのに使われるデータ型である．

数値データは連続値を取るが，文字列データは何種類かに値を分類できる．こうした離散データ（カテゴリカル・データとも言う）は，文字列そのものを記録しておくより，各カテゴリに対応させた整数値を記録しておく方が，記憶容量の節約になる．また，データの値を比較する際は，整数値同士の比較の方が，文字列を 1 文字 1 文字比較するより処理が速い．

そこで R では，文字列データをデータフレームに読み込む際，自動的に Factor に変換して記録する．Factor は文字列と整数値との対応関係を属性として保持し，文字列の代わりに整数値をデータとして使用する．

例として九州に住む人々のデータを考えてみよう．九州在住者全員の居住県を文字列でデータフレームに記録するより，1 から 7 の整数値で表現する方が遥かに容量の節約になる．

ベクトルデータから手動で Factor を作成するには factor() 関数を使う．

```
> kyushu.str <- c("福岡", "佐賀", "熊本", "長崎", "宮崎", "鹿児島", "沖縄")
> kyushu <- factor(kyushu.str)
> kyushu.str  # 文字列のベクトル
[1] "福岡"   "佐賀"   "熊本"   "長崎"   "宮崎"   "鹿児島" "沖縄"
> kyushu      # factor のベクトル
[1] 福岡   佐賀   熊本   長崎   宮崎   鹿児島 沖縄
Levels: 沖縄 宮崎 熊本 佐賀 鹿児島 長崎 福岡
```

文字列ベクトル kyushu.str ではデータがダブルクォーテーションで囲まれているが，Factor のベクトル kyushu では囲まれていない．つまり，Factor の各データは文字列ではないことを示している．kyushu では Levels 表記が下に追加されている．これは Factor で管理している全カテゴリを示している．Factor の levels は日本語では「水準」と呼ばれる．

少数の名簿データの場合，幾つかの県の出身者は含まれていないかもしれない．例えば，クラスメート 4 名の出身地が以下の students.pref で与えられているときは以下のように levels 引数に水準集合を指定して Factor を作成することで，データに含まれていない水準を含めてデータと整数値の対応関係が構築される．

```
> students.pref <- c("福岡", "鹿児島", "福岡", "長崎")
> students <- factor(students.pref , levels=kyushu.str)
> students
[1] 福岡   鹿児島 福岡   長崎
Levels: 福岡 佐賀 熊本 長崎 宮崎 鹿児島 沖縄
```

1 人目と 3 人目が福岡県出身なので，students には「福岡」が 2 回現れている．これらのデータの値は実際には整数値だが，見やすいように Levels: に列挙されているラベル名を用いて表示されている．

上で，もし levels 引数を指定しなければ，与えられたデータのみが全ての水準として Levels が設定される．

```
> students2 <- factor(students.pref)
> students2
[1] 福岡   鹿児島 福岡   長崎
Levels: 鹿児島 長崎 福岡
```

Factor の水準数を確認するには nlevels() 関数を用いる．

```
> nlevels(students)
[1] 7
```

```
> nlevels(students2)
[1] 3
```

データの水準集合を得たいときは `levels()` 関数を使う．

```
> levels(students)
[1] "福岡"   "佐賀"   "熊本"   "長崎"   "宮崎"   "鹿児島" "沖縄"
> levels(students2)
[1] "鹿児島" "長崎"   "福岡"
```

与えられた水準集合に含まれないデータが含まれている場合は `NA` となる．以下の例では東京が `kyushu.str` で与えられる水準集合に含まれていないので `NA` となる．

```
> factor(c("福岡", "佐賀", "東京", "長崎"), levels=kyushu.str)
[1] 福岡 佐賀 <NA> 長崎
Levels: 福岡 佐賀 熊本 長崎 宮崎 鹿児島 沖縄
```

3.1　整数値符号から Factor の生成方法

前節では文字列から Factor を生成する方法を示した．ここでは，既にデータが整数値に符号化されている場合の Factor 生成方法を示す．

整数値をカテゴリに分類するためには，符号化された値を `levels` に指定し，そのラベル名を `labels` 引数に与える．

```
> data <- c(3, 1, 8, 5)
> nums <- factor(data, levels=1:5, labels=c("One", "Two", "Three", "Four", "Five"))
> nums
[1] Three One   <NA>  Five
Levels: One Two Three Four Five
```

上の例では `levels` に 1 から 5 までの整数値ベクトルを与えているので，水準集合に含まれない 8 は欠損値扱いで `NA` となる．

Factor の水準名は `levels()` 関数を使って変更することができる．

```
> levels(nums)[4] <- "四"
> nums
[1] Three One   <NA>  Five
Levels: One Two Three 四 Five
> levels(nums)[5] <- "五"
> nums
[1] Three One   <NA>  五
Levels: One Two Three 四 五
```

3.2 Factor の比較

Factor は内部で整数値のベクトルとして管理されているが，その値を比較する際には `levels` の値が用いられる．例えば，先に作成した `students` と `students2` では水準集合が異なるため，内部的には各データに割り当てられた整数値は異なる．

```
> as.numeric(students)
[1] 1 6 1 4
> as.numeric(students2)
[1] 3 1 3 2
```

`students2` はそもそもカテゴリ（水準）が全部で 3 つしかない Factor として作成されたので割り当てられている値も `students` と異なる．しかし，各 Factor のラベル名は等しいので，等号テストでは等しいと評価される．

```
> students[1] == students2[1]
[1] TRUE
> students[2] == students2[2]
[1] TRUE
> students[3] == students2[3]
[1] TRUE
> students[4] == students2[4]
[1] TRUE
```

以上から分かることは，Factor は各データに割り振られた整数値同士を直接比較するのではなく，整数値に対応する水準名を参照してから比較するということである．したがって，整数値同士の比較よりは処理時間がかかる．冒頭では Factor を使う利点として，容量の節約と文字列を 1 文字ごとに比較しなくて良いという 2 点を挙げたが，後者の利点いついては議論の余地があるかもしれない[1]．恐らく，文字列よりは Factor の方が高速に比較できると思われるが，今回使用する擬似ミクロデータのように既に整数値で符号化されている場合には，比較に間接参照を伴う Factor を用いるより，整数値を用いた方が処理速度は必ず早くなる．処理速度が要求されるシビアなプロジェクトでは，Factor ではなく整数値による符号を使おう．

さて，話しを値の比較に戻そう．一般に，ベクトルデータ同士の演算は要素ごとに適用される．等号テストも以下のようにベクトル要素に対して行われる．

```
> c(1,2,3,4) == c(2,2,4,4)
[1] FALSE  TRUE FALSE  TRUE
```

しかし，Factor の場合は，水準集合が同じでなければそもそも値の比較は行われないので注意しよう．

[1] R 内部の実装までは確かめていないが，一般に水準名に Lisp などで使われる「シンボル」と同じ方法を用いることで，文字列の比較を回避することは可能なので，文字列でデータを保持するよりは Factor の方が処理速度に対しても効率的であると考えられる．

```
> students == students2

 Ops.factor(students, students2) でエラー:
   因子の水準セットが異なっています
```

3.3 順序を持つ Factor の生成方法

県名を Factor として保持する `students` と `students2` は，実際には整数値をデータとして保持しているのだが，その値の大小関係は未定義である．例えば，第 1 要素の値を `as.integer()` で表示させせると，`students2[1]` の方が `students[1]` より大きいが Factor による比較はできない．

```
> as.integer(students[1])
[1] 1
> as.integer(students2[1])
[1] 3
> students[1] < students2[1]
[1] NA
```

しかし，大小関係に意味があるカテゴリカル・データも存在する．例えば成績データは良い例である．今，A，B，C，D，F の 5 段階で評価される数学の成績データが以下で与えられているとしよう．

```
> math.data <- c("C", "B", "C", "A", "D", "F", "A")
```

このような離散データで大小関係が維持できたら，C 以上の成績を合格者として抽出する際に便利である．こういう場合には，順序付きで Factor を生成する．順序付きで Factor を生成するには `ordered` 引数を `TRUE` を設定する．

```
> math <- factor(math.data, levels=c("A", "B", "C", "D", "F"), ordered=TRUE)
> math
[1] C B C A D F A
Levels: A < B < C < D < F
```

上の実行結果を見ると `Levels` 表記に不等号が追加されているのが分かる．`levels` 引数に与えたベクトルの順序を昇順（小さい順）とみなして大小関係が設定される．この場合，A が一番小さい順番となるため，C 以上を合格とする時に A は不合格となってしまう．

```
> math
[1] C B C A D F A
Levels: A < B < C < D < F
> math >= "C"
[1]  TRUE FALSE  TRUE FALSE  TRUE  TRUE FALSE
```

上の例では C，D，F が TRUE になっている．順番を逆にしたい場合は，levels に与えるベクトルを rev() 関数で反転（reverse）させれば良い．

```
> math <- factor(math.data, levels=rev(c("A", "B", "C", "D", "F")), ordered=TRUE)
> math
[1] C B C A D F A
Levels: F < D < C < B < A
> math >= "C"
[1]  TRUE  TRUE  TRUE  TRUE FALSE FALSE  TRUE
```

これで正しく C 以上を合格とすることができた．

3.4 長さが異なるベクトルの比較について

第 3.2 節でベクトル同士の比較は，その要素同士の比較になると述べた．しかし，上の C 以上を合格とする math >= "C"の式の左辺は要素数が 7 のベクトルであるのに対し，右辺は要素が 1 のベクトルである[2]．どうして長さが異なるベクトルの要素同士が比較できるのだろうか．

R では 2 項演算子に与えられた各項の長さが異なる場合，短い方のデータが繰り返し生成されて長い方に合わせてから演算が実行される．もし，多い方の要素数が少ない方の要素数の整数倍ではない場合，繰り返した時に短い方のデータが中途半端になるため，下のように警告メッセージがでる．

```
> math >= c("C", "C")
[1]  TRUE  TRUE  TRUE  TRUE FALSE FALSE  TRUE
 警告メッセージ:
1:  is.na(e1) | is.na(e2) で:
   長いオブジェクトの長さが短いオブジェクトの長さの倍数になっていません
2:  get(.Generic, mode = "function")(e1, e2) で:
   長いオブジェクトの長さが短いオブジェクトの長さの倍数になっていません
```

math >= "C"の右辺の要素数は 1 なので，整数倍して必ず左辺の要素数と一致させることができる．それで問題なく比較できたのである．

3.5 練習問題

擬似ミクロデータのカテゴリ値は文字列ではなく，効率の良い整数値に符号化されているが，ここではラベル名での扱いやすさを優先させ，Factor 型に変換する．世帯人員，就業人員，年齢階級２，年齢階級１の列を大小関係を維持し Factor に変換せよ．ただし水準に与えるラベル名は「符号内容」に記載のものを使用し，年齢階級２の「就業者以外」は水準に含めず,「就業者以外」に該当するデータは NA に設定すること．また，それぞれの水準の順序関係を維持すること．

[2]R の基本データ型はベクトルなので，1 つの値は要素が 1 つのベクトルと同じ．つまり，math >= c("C") と同じ．

4 データの条件抽出

ここでは，大規模データから条件に合致したデータを抽出・集計する方法を学ぶ．

これまで，添字指定箇所に TRUE，FALSE を返す条件を与えて，結果を抽出してきた．例えば，世帯人員が 3 人以上で，就業人員が 2 人以上，45 歳以上の就業者で，且つ 3 大都市圏の持ち家で暮らしている家計の年間収入を抽出したいとしよう．これをこれまで学んだ方法で抽出し，その結果を summary() 関数に渡してデータの分布を見たいすると以下のようになる．

```
> index = microdata$３大都市圏 &
+     microdata$世帯人員 == "３人以上" &
+     microdata$就業人員 == "２人以上" &
+     microdata$持ち家所有 &
+     microdata$年齢階級２ >= "（就業者）45～49 歳"
> summary(microdata[index, "年間収入"])
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
  964.4  6378.6  8717.6  9649.1 11765.2 42786.0     278
```

条件に合致する真偽値インデックスを生成するために，各列のアクセスに毎回 microdata$を記述するのは面倒である上，条件の内容も見辛い．attach() を使うと各列への簡便なアクセスが実現できる．attach() にデータフレームを渡すと，データフレームの列名が R の変数と同じ「環境」に登録される．まだ「環境」が何かについては学んでいないが，今は作成した関数や変数の名前を管理している場所だと理解しておけば十分である．

```
> attach(microdata) # これ以降 microdata の列名を普通の変数のようにアクセスできる．
> index = ３大都市圏 &
+     世帯人員 == "３人以上" &
+     就業人員 == "２人以上" &
+     持ち家所有 &
+     年齢階級２ >= "（就業者）45～49 歳"
> summary(microdata[index, "年間収入"])
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
  964.4  6378.6  8717.6  9649.1 11765.2 42786.0     278
```

先ほどと同じ結果が得られた．また，条件もすっきりしてとても見やすい．しかし，この状態だと，本来 microdata データフレーム内だけで使用していた列名が，変数と同じレベルの環境に存在するため，データフレームの外で同じ名前の変数を使うことができない．

例えば，あるデータフレームの列名に x という非常に汎用的な名前が付けられている場合，このデータフレームを attach() すると，変数 x を使えない．正確にいうと使えるが，新たに変数を作成すると今度はデータフレームの x 列に直接列名でアクセスできなくなる．したがって，attach() は一時的に使用すべきで使用後は detach() を実行し，登録された列名を環境から取り除いた方がよい．

```
> detach(microdata) # 環境から microdata の列名を取り除く
```

多くの列が含まれる大規模データから，複雑な条件でデータを抽出するには，もっと便利な subset() 関数を用いるのが一般的である．次節では subset() 関数によるスマートなデータ抽出法を学ぶ.

4.1 練習問題

皆さんが大学を卒業したばかりの状況が含まれるグループを attach() を用いて抽出し，その年間収入の統計サマリを算出せよ．ただし，ここで使用しているデータには単身者世帯が含まれていないので最も近い状況で構わない.

4.2 subset() 関数を用いたデータ抽出

「高度な統計分析をするわけではないので R なんか必要ない」「Excel が使えれば十分だ」などといった言葉を文系卒の社会人から時々耳にする．たとえ高度な統計分析を行わなくても，ここまで学んできたデータの整理・整形テクニックは，Excel だけによる作業よりも遥かに効率的で役に立つ．これから学ぶ subset() によるデータの条件抽出は，Excel だけで実現するには極めて難しく[3]，今後，統計分析業務に従事しなくても日々の業務で必ず役に立つので，しっかりマスターしよう.

subset() 関数は第 1 引数にデータフレームを取り，第 2 引数以降に真偽値を返す条件を記述する．subset() 関数では条件を記述する際，attach() した時のように列名を直接記述することができる．前節の最後と同じ条件による抽出を subset() 関数を用いると以下のように書ける.

```
> summary(subset(microdata, ３大都市圏 &
+                            世帯人員 == "３人以上" &
+                            就業人員 == "２人以上" &
+                            持ち家所有 &
+                            年齢階級２ >= "（就業者）45〜49 歳"
+                )$年間収入)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  964.4  6378.6  8717.6  9649.1 11765.2 42786.0
```

上の例では，条件に合致するデータ（行）を抽出したのち，使用するのは年間収入の列だけである．もし，条件に合致する特定の列だけを抽出したい場合は select 引数が使える.

```
> summary(subset(microdata, ３大都市圏 &
+                            世帯人員 == "３人以上" &
+                            就業人員 == "２人以上" &
+                            持ち家所有 &
+                            年齢階級２ >= "（就業者）45〜49 歳",
+                select="年間収入"))
    年間収入
 Min.   :  964.4
```

[3]もちろん VisualBasic やマクロを使えば Excel でも高度な条件抽出とデータ加工を行えるが，その場合，VisualBasic によるプログラミングを学ばなければならないので R を学ぶのと変わらない．実際には VisualBasic の方が汎用言語なので R を学ぶよりも根気がいるだろう.

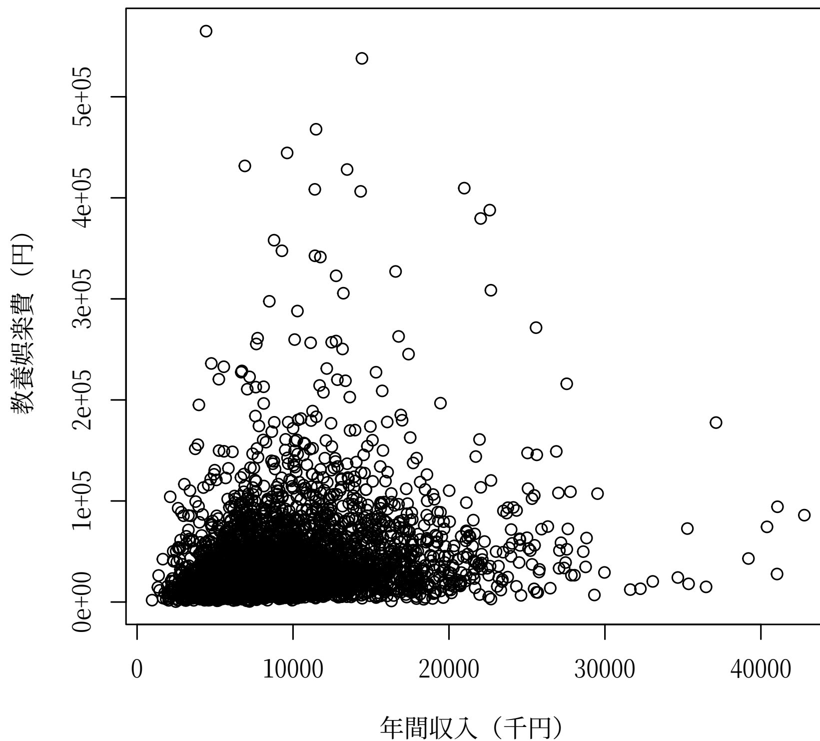
```
 1st Qu.: 6378.6
 Median : 8717.6
 Mean   : 9649.1
 3rd Qu.:11765.2
 Max.   :42786.0
```

上の結果は１つ前の結果と見た目が少し異なる．これは，subset() は条件に合致したデータフレームを返すためだ．summary() はデータフレームが与えられると列毎に統計サマリを算出する．1 つ前の方法では subset() が返したデータフーレムから$年間収入で年間収入列だけ抜き出したため，列がベクトルとして返される．ベクトル単体の統計サマリなので単純な表示となっていた．subset() で複数列を抽出するとその違いが顕著になる．

```
> zz <- subset(microdata, ３大都市圏 &
+                         世帯人員 == "３人以上" &
+                         就業人員 == "２人以上" &
+                         持ち家所有 &
+                         年齢階級２ >= "（就業者）45～49 歳",
+              select=c("年間収入", "教養娯楽"))  # 2 つの列を抽出する
> str(zz)     # データの構造を見てみるとデータフレームであることがわかる
'data.frame':        3931 obs. of  2 variables:
 $ 年間収入: num  14725 12422 5737 14203 8231 ...
 $ 教養娯楽: num  91093 24742 7386 73840 58477 ...
> summary(zz) # 列毎にサマリを計算する
    年間収入            教養娯楽
 Min.   :  964.4   Min.   :   414
 1st Qu.: 6378.6   1st Qu.: 14328
 Median : 8717.6   Median : 26271
 Mean   : 9649.1   Mean   : 38859
 3rd Qu.:11765.2   3rd Qu.: 48160
 Max.   :42786.0   Max.   :564954
```

45 歳以上の勤労世帯の年間収入と教養娯楽費を抽出したので，その散布図（scatter plot）をプロットして両者の関係を見てみよう．散布図は plot() 関数を使う．

```
> par(family="Japan1Ryumin") # これは Mac で日本語が文字化けした場合のみ必要
> plot(zz$年間収入, zz$教養娯楽,
+      xlab="年間収入（千円）", ylab="教養娯楽費（円）",
+      main="年間収入と教養娯楽費の関係")
```



年収 1 千万円手前から 2 千万円手前あたりの所得階層の勤労世帯が，教養娯楽費に多くの金額を投入しているのが分かる．縦軸の目盛りが `2e+05` となっているのは，`2*10^5 = 200,000` を表す．したがって，教養娯楽費が年間 10 万円を超えると，その他大勢よりも多目に教養娯楽費に支出している人であることが視覚的に分かる．サマリの結果と照らし合わせてより深く分布を吟味すると良いだろう．

4.3 練習問題

3 大都市圏以外の持ち家に住む世帯で，45 歳以上の就業者と非就業者を合わせたグループについて以下の問いに答えよ．

(1) 年間収入と教育費を抽出し統計サマリを求めよ．

(2) 年間収入の分布を見るためのヒストグラムを描け．

(3) 年間収入と教育費の散布図を描き結果を吟味せよ．

5 リスト型

第 3 節で新たに Factor 型を学んだ．本節ではもう 1 つ重要で頻繁に使うデータ型 List を学習する．List 型はベクトルのように要素が順番に並ぶシーケンシャルなデータ構造だが，ベクトルと違い要素にはどのようなデータ型でも格納できる．各要素が同じデータ構造である必要もない．

List を生成するには list() 関数を使う．

```
> lst <- list(a=c(1,2,3,4), b="データ", c=microdata[1:3, 1:2])
> lst
$a
[1] 1 2 3 4

$b
[1] "データ"

$c
  3 大都市圏 世帯人員
1       TRUE     2 人
2       TRUE     2 人
3       TRUE     2 人
```

上の例では 1 番目の要素に数字のベクトルデータを，2 番目の要素に文字列 1 個のベクトルデータを，3 番目に microdata から 1–3 行目，1–2 列目を抽出したデータフレームを格納したリストを作成している．リスト生成時に，名前付き引数を使って要素を指定すると，引数名が要素の名前に設定される．$演算子で要素に名前を使ってアクセスできる．

```
> lst$a
[1] 1 2 3 4
> lst$c
  3 大都市圏 世帯人員
1       TRUE     2 人
2       TRUE     2 人
3       TRUE     2 人
```

list() の要素は引数に渡された順番で格納されているので，インデックス番号でもアクセスすることができる．インデックス付けされた要素に格納されている「中身」を取り出すには 2 重括弧の [[]] 演算子を用いる．

```
> lst[[1]]
[1] 1 2 3 4
> lst[[3]]
  3 大都市圏 世帯人員
1       TRUE     2 人
2       TRUE     2 人
3       TRUE     2 人
```

ベクトルの要素にアクセスするのに使用した [] 演算子も使えるが，[] 演算子は構造保存式のデータ抽出演算子で，元のデータ構造を維持したまま要素を返す．つまり，ここの場合，抽出した要素は再びリストに格納されて返る．

```
> lst[1]  # 1 番目の要素を抽出．結果は再びリスト
$a
```

```
[1] 1 2 3 4
> lst[3]  # 3 番目の要素を抽出．結果は抽出した要素から成るリスト
$c
  3 大都市圏 世帯人員
1       TRUE      2 人
2       TRUE      2 人
3       TRUE      2 人
```

それぞれのデータ型（mode）を見てみればその違いがわかる．

```
> mode(lst[[1]])  # 数値ベクトル型
[1] "numeric"
> mode(lst[1])    # リスト型
[1] "list"
```

これまでベクトル型のデータ抽出には [] 演算子しか使ってこなかったが，[[]] も使える．前者はベクトルのデータ構造を維持したまま要素を抽出するのに対し，後者は値のみを抽出する．ベクトルに名前属性を指定してみると両者の違いがわかる．

```
> v <- c(1, 2, 3, 4, 5, 6)
> names(v) <- c("one", "two", "three", "four", "five", "six")
> v
  one   two three  four  five   six
    1     2     3     4     5     6
> v[[3]] # v から 3 番目の要素の「値」だけを抽出する
[1] 3
> v[3]   # v の構造を保存したままなので名前属性とともに返る
three
    3
```

両者の大きな違いは，データ構造を保存しない [[]] は元の構造を維持できないので，複数要素を抽出するような範囲指定ができない点である．

```
> v[1:3]   # 範囲の指定 OK
  one   two three
    1     2     3
> v[[1:3]] # 範囲の指定はできない
 v[[1:3]] でエラー:
  attempt to select more than one element in vectorIndex
```

[[]] 演算子も [] 演算子と同様，名前属性を使ったデータ抽出をサポートしている．

```
> v["five"]
five
   5
```

```
> v[["five"]]
[1] 5
```

再びリストに話しを戻そう．リストの要素名は名前属性（names）で変更できる．以下の例では 3 番目の要素名を「大規模ミクロデータ」に変更している．

```
> names(lst)
[1] "a" "b" "c"
> names(lst)[3] <- "大規模ミクロデータ"
> lst
$a
[1] 1 2 3 4

$b
[1] "データ"

$大規模ミクロデータ
  ３大都市圏 世帯人員
1       TRUE     ２人
2       TRUE     ２人
3       TRUE     ２人
```

リストは要素には何でも格納できるため，リストをリストの要素に保存することもできる．こうした特徴から，R の様々な関数は計算結果や分析結果をリストに保存して返すことが多い．分析結果を適切に取り出せるように，リストからの要素抽出方法はよく理解しておこう．

5.1 データフレームとリスト

第 3 回のプリントで，データフレームの実体は，長さが同じベクトルを要素にもつリストであると学んだ．リストとデータフレームの構造を調べると，どちらも各要素（データフレームの場合は各列）の名前の前に$演算子が付いているのが確認できる．

```
> ## リストの場合
> str(lst)
List of 3
 $ a                 : num [1:4] 1 2 3 4
 $ b                 : chr "データ"
 $ 大規模ミクロデータ:'data.frame':        3 obs. of  2 variables:
  ..$ ３大都市圏: logi [1:3] TRUE TRUE TRUE
  ..$ 世帯人員  : Ord.factor w/ 2 levels "２人"<"３人以上": 1 1 1
> ## データフレームの場合
> str(microdata)
'data.frame':        45811 obs. of  20 variables:
 $ ３大都市圏      : logi  TRUE TRUE TRUE TRUE TRUE TRUE ...
 $ 世帯人員        : Ord.factor w/ 2 levels "２人"<"３人以上": 1 1 1 1 1 1 1 1 1 1 ...
 $ 就業人員        : Ord.factor w/ 2 levels "１人"<"２人以上": 1 1 1 1 1 1 1 1 1 1 ...
 $ 持ち家所有      : logi  TRUE TRUE TRUE TRUE TRUE TRUE ...
```

```
 $ 就業中          : logi  TRUE TRUE TRUE TRUE TRUE TRUE ...
 $ 年齢階級２      : Ord.factor w/ 9 levels "（就業者）30 歳未満"<..: 1 1 1 1 1 1 1 1 1 1 ...
 $ 年齢階級１      : Ord.factor w/ 2 levels "65 歳未満"<"65 歳以上": 1 1 1 1 1 1 1 1 1 1 ...
 $ 集計用乗率      : num  895 895 895 895 895 ...
 $ 年間収入        : num  3917 6675 2790 3452 9252 ...
 $ 消費支出        : num  201649 166381 114511 152109 192439 ...
 $ 食料            : num  47756 34054 41664 34924 68882 ...
 $ 住居            : num  16028 7416 730 3418 2832 ...
 $ 光熱・水道      : num  9652 26313 22358 8131 23042 ...
 $ 家具・家事用品  : num  6702 17062 5413 4164 2598 ...
 $ 被服及び履物    : num  8088 6989 1205 6970 5714 ...
 $ 保健医療        : num  726 7637 5049 4247 2052 ...
 $ 交通・通信      : num  21546 20773 17411 47698 37006 ...
 $ 教育            : num  0 0 0 0 0 0 0 0 0 0 ...
 $ 教養娯楽        : num  14433 19048 8605 36419 38588 ...
 $ その他の消費支出: num  76719 27089 12077 6137 11725 ...
```

したがって，データフレームの列を抽出するのに，2 重括弧の [[]] を使うこともできる．ただし，先ほど学んだように範囲を指定して列を抽出することはできない.

```
> m4 <- microdata[[4]] # 4 列目「持ち家所有」の中身（ベクトル）を抽出
> m4[1:10]             # ベクトルデータの最初の 10 要素を表示
 [1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
```

[] 演算子で 4 列目を抽出すると構造が保存され,「持ち家所有」列だけからなるデータフレームが返る.

```
> d4 <- microdata[4]
> str(d4)
'data.frame':        45811 obs. of  1 variable:
 $ 持ち家所有: logi  TRUE TRUE TRUE TRUE TRUE TRUE ...
```

上のデータフレームはリストでもあるので，microdata[4] でリストの 4 つ目の要素，すなわち 4 列目を抽出していることと同じである．つまり，データフレームで 4 列目を抽出する microdata[, 4] と同じ結果になる．microdata[4] の方が表記は簡潔だが，後日プログラムを見直したときに microdata がベクトルなのかデータフレームなのか区別しがたい．データフレームから列を抽出するときは，プログラムの読み手に分かり易いよう microdata[, 4] を使う方がよいだろう.

ただし，リストから複数要素を取り出したいときには [] 演算子で範囲指定するしかないので，[[]] や$による要素アクセスだけでなく [] 演算子も使えるようにしておこう.

5.2 練習問題

microdata から「消費支出」列を抽出する方法を 5 通り示し，それぞれ抽出した結果から列の第 1 要素を取り出せ.

5.3　リストの連結

ここでは既に作成済みのリストに，新たに要素を追加する方法を幾つか紹介する．今，以下で作成済みのリストに新たに要素を追加したいとしよう．

```
> lst <- list(a=c(1,2,3,4), b="データ", c=microdata[1:3, 1:2])
```

このリストに list() 関数を使ってベクトルを要素 d として追加してみる．

```
> lst2 <- list(lst, d=c("a", "b", "c"))
> lst2
[[1]]
[[1]]$a
[1] 1 2 3 4

[[1]]$b
[1] "データ"

[[1]]$c
  ３大都市圏 世帯人員
1       TRUE     ２人
2       TRUE     ２人
3       TRUE     ２人



$d
[1] "a" "b" "c"
```

上の結果は少々見辛いが，まず 1 番目の要素 [[1]] に a，b，c の要素を持つリストが格納されている．実際，lst2[[1]] を評価してみれば分かるが，1 番目の要素には list() の第 1 要素に指定した lst がそのままリストとして格納されている．追加した要素 d は 2 番目の要素なので，lst2[[2]] でも lst2$d でもアクセスできる．

リストは何でも格納できるため，既存のリストに新たに要素を追加する場合には，構造に気をつけなければならない．list() 関数を使ってリストを追加すると，リストが入れ子になってしまうからだ．

既存のリストと，新たに追加する要素を同じレベルの要素としてフラットに繋げるには，ベクトル型の要素の接続に使った c() 関数を使う．しかし，以下の例では，新たに追加したいベクトルがそれぞれ 3 つの要素に分解されてしまう．

```
> c(lst, d=c("a", "b", "c"))
$a
[1] 1 2 3 4

$b
[1] "データ"

$c
```

```
  3大都市圏 世帯人員
1       TRUE     2人
2       TRUE     2人
3       TRUE     2人

$d1
[1] "a"

$d2
[1] "b"

$d3
[1] "c"
```

以下のように名前付き引数を指定しなくてもベクトルが同じように 3 つの要素に分解される．

```
> c(lst, c("a", "b", "c"))
$a
[1] 1 2 3 4

$b
[1] "データ"

$c
  3大都市圏 世帯人員
1       TRUE     2人
2       TRUE     2人
3       TRUE     2人

[[4]]
[1] "a"

[[5]]
[1] "b"

[[6]]
[1] "c"
```

リストの結合に c() 関数を使用すると，ベクトルはまずリストに変換してから他のリストと結合される．3 つの要素から成るベクトルは 3 つの要素から成るリストに変換され，それから連結する 2 つのリストの要素がフラットに接続されるのである．

したがって，ベクトル c("a", "b", "c") をリストの 1 要素として既存リストに追加するには，以下のように一旦，ベクトルを要素とするリストを作成してから c() 関数に渡せばよい．

```
> c(lst, list(d=c("a", "b", "c")))
$a
[1] 1 2 3 4
```

```
$b
[1] "データ"

$c
  3大都市圏 世帯人員
1       TRUE    2人
2       TRUE    2人
3       TRUE    2人

$d
[1] "a" "b" "c"
```

たいていのプログラミング言語では，変数に値を代入する前に，変数を事前に準備する必要がある（いわゆる変数宣言）．R ではその必要なく，存在しない変数に値を代入すると自動的に変数が作成される．これはベクトルやリストの「要素」に関しても同様で，以下のようにリストのインデックスを使ったり名前を使って新たに要素を追加することもできる．いま lst には 3 つの要素 a，b，c が入っている．

```
> lst
$a
[1] 1 2 3 4

$b
[1] "データ"

$c
  3大都市圏 世帯人員
1       TRUE    2人
2       TRUE    2人
3       TRUE    2人
```

ここに新たに存在しない要素名 d でアクセスすることで新たな要素が作成される．

```
> lst$d <- c("a", "b", "c")
> lst
$a
[1] 1 2 3 4

$b
[1] "データ"

$c
  3大都市圏 世帯人員
1       TRUE    2人
2       TRUE    2人
3       TRUE    2人

$d
```

```
[1] "a" "b" "c"
```

文字列で要素にアクセスして追加することもできる．

```
> lst[["e"]] <- 100:105
> lst
$a
[1] 1 2 3 4

$b
[1] "データ"

$c
  3大都市圏 世帯人員
1       TRUE     2人
2       TRUE     2人
3       TRUE     2人

$d
[1] "a" "b" "c"

$e
[1] 100 101 102 103 104 105
```

現在 lst には 5 つの要素が含まれているので，インデックス番号を使うことで最後尾に要素を追加してみる．

```
> lst[[length(lst)+1]] <- c("ここが", "最後尾", "です")
> lst
$a
[1] 1 2 3 4

$b
[1] "データ"

$c
  3大都市圏 世帯人員
1       TRUE     2人
2       TRUE     2人
3       TRUE     2人

$d
[1] "a" "b" "c"

$e
[1] 100 101 102 103 104 105

[[6]]
```

```
[1] "ここが" "最後尾" "です"
```

番号でアクセスして追加したので，最後の要素には名前が付いていない．

前節でデータフレームはリストの一種であると紹介した．したがって，データフレームをリストに追加するときには注意が必要だ．データフレームをリストに追加しようとして c() 関数を使うと，リストの要素である「列データ」が他のリスト要素とフラットに結合されてしまう．

例えば，2 つのデータフレームから成るリスト dlst に 3 つ目のデータフレームを新たに追加したいとしよう．

```
> dlst <- list(a=data.frame(ID=c("a","b","c"), name=c("太郎", "花子", "一郎")),
+              b=data.frame(date=c(1,2), weather=c("晴","曇")))
> dlst
$a
  ID name
1  a 太郎
2  b 花子
3  c 一郎

$b
  date weather
1    1      晴
2    2      曇
```

追加したいデータフレームを変数 d に格納して c() で dlst に追加してみる．

```
> d <- data.frame(国名=c("日本","カナダ"), 略名=c("JP","CA"))
> c(dlst, d)
$a
  ID name
1  a 太郎
2  b 花子
3  c 一郎

$b
  date weather
1    1      晴
2    2      曇

$国名
[1] "日本"   "カナダ"

$略名
[1] "JP" "CA"
```

データフレーム d はリストでもあるので，データフレームの列がリストの要素としてそれぞれトップレベルに追加されているのが分かるだろう．さらに文字列ベクトルは自動的に factor に変換されていることも確認できる．

2 つのデータフレームを保持する既存リスト dlst に 3 つ目のデータフレームを追加するには，存在しないインデックス番号か名前でアクセスすることで追加しなければならない．

```
> dlst[["d"]] <- d
> dlst
$a
  ID name
1  a 太郎
2  b 花子
3  c 一郎

$b
  date weather
1    1      晴
2    2      曇

$d
    国名 略名
1   日本   JP
2 カナダ   CA
```

リストへの要素の追加は，ループ処理中にリストを逐次構築していく際に必要になる．第 9 回のプリントではループを使わずに apply 系関数を使ってリストを逐次構築していく方法を示すが，ここで紹介した方法がどうしても必要なケースに直面することもあるだろう．リストは何でも保持できる汎用性の高いデータ構造なので，その操作方法をしっかり身につけておこう．

6 データの保存

今回行った大規模ミクロデータの整形処理結果を次回以降も使えるように保存しておこう．第 5 回では Microdata.RData に保存した．同じファイルに複数のオブジェクト（作成した変数等）を保存することも可能だが，今回は新たに別ファイル Microdata2.RData に保存しておく．パスは各自の環境に合わせて変更すること．

```
> save(microdata, file="Microdata2.RData")
```

もし仮に，今回行ったデータ整形処理を全て別の変数 microdata2 に対して行っている場合には，オリジナルの microdata と修正済みの microdata2 を同じファイル（Microdata.RData）に保存したいであろう．その時は以下のように 2 つのオブジェクトを並べて記述することで 1 つのファイルに保存される．

```
> save(microdata, microdata2, file="Microdata.RData")
```

このファイルをロードすれば 2 つのオブジェクトが読み込まれる．