Chapter 1

Functions in tidyverse

1.1 The packages dplyr and tidyr - the core of tidyverse

These two packages are an important component of the tidyverse. The package `dplyr` was written by Hadley Wickham to try to simplify, speed-up and regularize commands and their arguments for manipulating data frames. The functions in **dplyr** operate on a data frame (their first argument) and the output is a data frame. HW refers to these elementary functions as 'Single table verbs'. This allows us to 'pipe' a continuous sequence of commands with each one taking a data frame as its first argument and passing its output (also a dataframe) to the next command in the pipe. Functions in `dplyr` are designed to form simple manipulations and in order to do more complicated tasks, the user constructs a sequence of function calls. This is like a 'grammar' of manipulation of data objects. The package is quite new (2014) and still under development but has been welcomed by the R community as a potentially unifying approach. There are many different ways of manipulating data frames in R, with inconsistent organisation of arguments and outputs.

The package `tidyr` complements `dplyr` with functions that, in the words of Hadley Wickham, 'provides a bunch of tools to help tidy up your messy datasets'.

Note:

Many of these functions have corresponding functions in base R

1.1.1 Data set

We will use the data set BackPain which is a quite large data set both concerning the number of observations and 24 variables which means that it is a little difficult to get an overview. In the examples we will use a random sample (more about the **sample** function in the simulation section) of the full data set. The imported data set is a tibble so all variables are not printed. However this is enabled with the function **glimpse**.

```
BackPain<-read_csv("../data/BackPain.csv") # The resulting object is a tibble
BackPain

# A tibble: 34,122 x 25
      id residence sex      age wealthQ  physical  country backPain30 agegr maritalS eduS
   <dbl> <chr>     <chr>  <dbl> <chr>    <chr>     <chr>   <chr>      <chr> <chr>    <chr>
 1     1 Urban     Female    66 Q3       high phy~ China   No         60-69 Div/Wid~ Comp~
 2     2 Urban     Female    63 Q4       mod phys~ China   No         60-69 Div/Wid~ Comp~
```

```
 3     3 Urban     Female    76 Q5 rich~ high phy~ China   No         70-79 Div/Wid~ No p~
 4     4 Rural     Male      64 Q4       high phy~ Mexico  No         60-69 Married~ Comp~
 5     5 Rural     Female    67 Q1 poor~ low phys~ China   No         60-69 Married~ No p~
 6     6 Urban     Female    74 Q4       mod phys~ China   No         70-79 Div/Wid~ Comp~
 7     7 Rural     Female    51 Q3       low phys~ South ~ No         50-59 Married~ No p~
 8     8 Urban     Female    80 Q5 rich~ low phys~ South ~ No         80+   Div/Wid~ Comp~
 9     9 Urban     Male      67 Q4       high phy~ China   No         60-69 Married~ No p~
10    10 Urban     Male      75 Q5 rich~ mod phys~ China   No         70-79 Married~ Comp~
# ... with 34,112 more rows, and 14 more variables: workS <chr>, bmi <dbl>, bmi4 <chr>,
#   waistc <dbl>, smoke <chr>, alcohol <chr>, arthritis <chr>, angina <chr>,
#   depression <chr>, asthma <chr>, diabetes <chr>, comorb <dbl>, disability <dbl>,
#   height <dbl>

set.seed(1001)
bP<-BackPain[sample(nrow(BackPain),10000),] # let us take a subsample
tibble::glimpse(bP) # this is a way of getting an overview of all variables

Rows: 10,000
Columns: 25
$ id         <dbl> 27043, 27471, 5323, 9068, 28828, 23775, 24138, 17074, 18398, 25789, 8~
$ residence  <chr> "Urban", "Urban", "Rural", "Rural", "Urban", "Urban", "Rural", "Urban~
$ sex        <chr> "Female", "Female", "Female", "Female", "Female", "Female", "Female",~
$ age        <dbl> 57, 64, 60, 70, 58, 60, 67, 63, 60, 76, 71, 65, 65, 51, 52, 53, 62, 7~
$ wealthQ    <chr> "Q3", "Q3", "Q2", "Q1 poorest", "Q5 richest", "Q1 poorest", "Q2", "Q5~
$ physical   <chr> "mod phys act", "mod phys act", "high phys act", "low phys act", "mod~
$ country    <chr> "South Africa", "China", "China", "Ghana", "Mexico", "Mexico", "China~
$ backPain30 <chr> "Yes", "Yes", "No", "No", "Yes", "Yes", "Yes", "No", "No", "Yes", "No~
$ agegr      <chr> "50-59", "60-69", "60-69", "70-79", "50-59", "60-69", "60-69", "60-69~
$ maritalS   <chr> "Married/Cohab", "Married/Cohab", "Married/Cohab", "Div/Wid/Sep", "Ma~
$ eduS       <chr> "No primary", "No primary", "No primary", "No primary", "Compl Uni/Co~
$ workS      <chr> "currently not working", "never worked", "currently working", "curren~
$ bmi        <dbl> 30.46875, 24.94150, 28.21869, 16.15668, 32.83567, 31.14878, 26.15515,~
$ bmi4       <chr> "Obese", "Normal", "Pre-Obese", "Underweight", "Obese", "Obese", "Pre~
$ waistc     <dbl> 108.0, 82.0, 80.0, 86.0, 123.0, 101.5, 89.1, 68.0, 72.0, 87.4, 94.1, ~
$ smoke      <chr> "Never/Not Daily/Current", "Never/Not Daily/Current", "Never/Not Dail~
$ alcohol    <chr> "Abstainers", "Abstainers", "Abstainers", NA, "Abstainers", "Abstaine~
$ arthritis  <chr> "yes", "yes", "no", "no", "no", "yes", "yes", "no", "no", "no", "yes"~
$ angina     <chr> "yes", "no", "no", NA, "no", "no", "no", "no", "no", "no", "yes", "ye~
$ depression <chr> "no", "no", "no", "yes", "no", "yes", "no", "no", "no", "no", "no", "~
$ asthma     <chr> "no", "no", "no", "no", "no", "no", "no", "no", "no", "no", "no", "ye~
$ diabetes   <chr> "yes", "no", "no", "no", "yes", "no", "no", "no", "no", "no", "no", "~
$ comorb     <dbl> 2, 1, 0, 1, 1, 2, 1, 0, 0, 0, 2, 2, 0, 0, 0, 0, 2, 1, 0, 0, 1, 0, 2, ~
$ disability <dbl> 0.000000, 13.888890, 8.333333, 66.666660, 27.777780, 11.111110, 13.88~
$ height     <dbl> NA, 156, 155, NA, 162, NA, 155, 166, 172, 159, 155, 164, 170, NA, 165~
```

There is no text variables converted to factors in the imported data set. However, it is more convenient to use factors so we change all character vectors to factors by using the verb function **mutate_if**.

```
bP <- mutate_if(bP,is.character, as.factor)
head(bP)

# A tibble: 6 x 25
     id residence sex      age wealthQ  physical  country backPain30 agegr maritalS eduS
  <dbl> <fct>     <fct>  <dbl> <fct>    <fct>     <fct>   <fct>      <fct> <fct>    <fct>
1 27043 Urban     Female    57 Q3       mod phys~ South ~ Yes        50-59 Married~ No pr~
2 27471 Urban     Female    64 Q3       mod phys~ China   Yes        60-69 Married~ No pr~
3  5323 Rural     Female    60 Q2       high phy~ China   No         60-69 Married~ No pr~
```

```
4  9068 Rural     Female    70 Q1 poor~ low phys~ Ghana   No         70-79 Div/Wid~ No pr~
5 28828 Urban     Female    58 Q5 rich~ mod phys~ Mexico  Yes        50-59 Married~ Compl~
6 23775 Urban     Female    60 Q1 poor~ high phy~ Mexico  Yes        60-69 Div/Wid~ No pr~
# ... with 14 more variables: workS <fct>, bmi <dbl>, bmi4 <fct>, waistc <dbl>,
#   smoke <fct>, alcohol <fct>, arthritis <fct>, angina <fct>, depression <fct>,
#   asthma <fct>, diabetes <fct>, comorb <dbl>, disability <dbl>, height <dbl>
```

1.2 Single table verbs

We will now look at the following single table verbs:

In tidyverse	**Examples of corresponding functions in basic R**
select()	matrix form df[,column selection]
filter()	subset() or in matrix form df[row selection,]
slice()	matrix form df[selection by row numbers,]
arrange()	order()
rename()	names(df)[column number]<-"newvar"
mutate()	df$newvar<-expression
count()	table() or xtabs()
group_by()	?
summarise()	?

Some of these also have so called scoped variants of these, e.g. mutate_if. Se help on scoped.

1.2.1 `select()`

Column selection can be done using column names (always preferred) or their numerical position.

```
bPs <- select(bP,country, residence, sex, height, disability, diabetes)

Error in select(bP, country, residence, sex, height, disability, diabetes):  unused arguments
(country, residence, sex, height, disability, diabetes)

head(bPs)

# A tibble: 6 x 6
# Groups:   country, sex, residence [1]
  country sex    residence wealthQ    count mean.disability
  <fct>   <fct>  <fct>     <fct>      <int>           <dbl>
1 China   Female Rural     Q1 poorest   277           15.7
2 China   Female Rural     Q2           260           12.4
3 China   Female Rural     Q3           172           11.8
4 China   Female Rural     Q4           156            9.99
5 China   Female Rural     Q5 richest    99            7.10
6 China   Female Rural     <NA>           6           24.5
```

In next example it is using a sequence of column numbers. Observe the alternative printing by using `kable`.

```
bPs <- select(bP, 1:4)

Error in select(bP, 1:4):  unused argument (1:4)

kable(head(bPs))
```

country	sex	residence	wealthQ	count	mean.disability
China	Female	Rural	Q1 poorest	277	15.683915
China	Female	Rural	Q2	260	12.403846
China	Female	Rural	Q3	172	11.757106
China	Female	Rural	Q4	156	9.989316
China	Female	Rural	Q5 richest	99	7.098765
China	Female	Rural	NA	6	24.537038

You can use the ":" between named columns, for an interval. You can delete a column by selecting it with a minus sign in front of name or number:

```
bPs <- select(bP, -c(sex,age))

Error in select(bP, -c(sex, age)):  unused argument (-c(sex, age))

head(bPs)

# A tibble: 6 x 6
# Groups:   country, sex, residence [1]
  country sex    residence wealthQ    count mean.disability
  <fct>   <fct>  <fct>     <fct>      <int>           <dbl>
1 China   Female Rural     Q1 poorest   277           15.7
2 China   Female Rural     Q2           260           12.4
3 China   Female Rural     Q3           172           11.8
4 China   Female Rural     Q4           156            9.99
5 China   Female Rural     Q5 richest    99            7.10
6 China   Female Rural     <NA>           6           24.5
```

or:

```
bPs <- select(bP, -(3:4))

Error in select(bP, -(3:4)):  unused argument (-(3:4))

head(bPs)

# A tibble: 6 x 6
# Groups:   country, sex, residence [1]
  country sex    residence wealthQ    count mean.disability
  <fct>   <fct>  <fct>     <fct>      <int>           <dbl>
1 China   Female Rural     Q1 poorest   277           15.7
2 China   Female Rural     Q2           260           12.4
3 China   Female Rural     Q3           172           11.8
4 China   Female Rural     Q4           156            9.99
5 China   Female Rural     Q5 richest    99            7.10
6 China   Female Rural     <NA>           6           24.5
```

1.2.2 `filter()`

With `filter()` you select **rows** of a data frame. Note the double '=' signs as usual. If you use more than one selection condition there is "and" between. NA in a filter variable is dropped.

```
bPf <- filter(bP, country == 'China' ,
               residence == 'Rural', sex=='Female')
bPf

# A tibble: 970 x 25
      id residence sex      age wealthQ  physical country backPain30 agegr maritalS eduS
   <dbl> <fct>     <fct>  <dbl> <fct>    <fct>    <fct>   <fct>      <fct> <fct>    <fct>
 1  5323 Rural     Female    60 Q2       high ph~ China   No         60-69 Married~ No pr~
 2 24138 Rural     Female    67 Q2       mod phy~ China   Yes        60-69 Married~ No pr~
 3  4626 Rural     Female    55 Q5 rich~ high ph~ China   No         50-59 Married~ No pr~
 4 15300 Rural     Female    53 Q5 rich~ high ph~ China   No         50-59 Married~ Compl~
 5  9083 Rural     Female    61 Q4       low phy~ China   No         60-69 Married~ Compl~
 6  9827 Rural     Female    58 Q5 rich~ high ph~ China   No         50-59 Married~ No pr~
 7   605 Rural     Female    50 Q2       low phy~ China   No         50-59 Married~ Compl~
 8  5993 Rural     Female    52 Q2       high ph~ China   No         50-59 Married~ No pr~
 9 28901 Rural     Female    68 Q1 poor~ high ph~ China   Yes        60-69 Married~ No pr~
10 18931 Rural     Female    64 Q1 poor~ high ph~ China   No         60-69 Married~ Compl~
# ... with 960 more rows, and 14 more variables: workS <fct>, bmi <dbl>, bmi4 <fct>,
#   waistc <dbl>, smoke <fct>, alcohol <fct>, arthritis <fct>, angina <fct>,
#   depression <fct>, asthma <fct>, diabetes <fct>, comorb <dbl>, disability <dbl>,
#   height <dbl>
```

We can get rid of those pesky NA's using the `complete.cases` function.

```
bPnf <-  select(bP,bmi,waistc,age, height)

Error in select(bP, bmi, waistc, age, height):  unused arguments (bmi, waistc, age, height)

summary(bPnf)

Error in summary(bPnf):  object 'bPnf' not found

bPf <-  filter(bPnf,complete.cases(bmi,waistc,age)) # data are now also complete in bmi and waistc

Error in filter(bPnf, complete.cases(bmi, waistc, age)):  object 'bPnf' not found

# complete.cases operate on variables while omit.na operates on a data frame
summary(bPf)

       id          residence        sex             age                wealthQ
 Min.   :   44   Rural:970    Female:970   Min.   :50.00   Q1 poorest:277
 1st Qu.: 8163   Urban:  0    Male  :  0   1st Qu.:55.00   Q2        :260
 Median :16685                             Median :60.00   Q3        :172
 Mean   :16617                             Mean   :62.16   Q4        :156
 3rd Qu.:24634                             3rd Qu.:68.00   Q5 richest: 99
 Max.   :34115                             Max.   :90.00   NA's      :  6

          physical                   country    backPain30   agegr              maritalS
 high phys act:404   China             :970   No  :648    50-59:457   Div/Wid/Sep   :197
 low phys act :347   Ghana             :  0   Yes :285    60-69:305   Married/Cohab:772
 mod phys act :202   India             :  0   NA's: 37    70-79:164   Never Married:   1
 NA's         : 17   Mexico            :  0               80+  : 44
                     Russian Federation:  0
                     South Africa      :  0

             eduS                        workS             bmi                  bmi4
 Compl Primary :161   currently not working:243   Min.   :13.30   Normal     :542
 Compl Sec/HS  : 77   currently working    :502   1st Qu.:21.06   Obese      : 50
 Compl Uni/Coll:  0   never worked         :208   Median :23.62   Pre-Obese  :278
 No primary    :732   NA's                 : 17   Mean   :23.76   Underweight: 57
```

```
                                                               3rd Qu.:26.10   NA's       : 43
                                                               Max.   :42.56
                                                               NA's   :43
    waistc                               smoke
 Min.   : 58.20   Current Daily           : 35
 1st Qu.: 76.35   Never/Not Daily/Current:918
 Median : 83.40   NA's                    : 17
 Mean   : 83.52
 3rd Qu.: 90.10
 Max.   :115.40
 NA's   :42
                                     alcohol    arthritis   angina    depression
 Abstainers                              :840   no  :730   no  :647   no  :920
 Non-heavy/Infreq heavy/Freq heavy drinkers: 66   yes :214   yes : 90   yes : 24
 NA's                                    : 64   NA's: 26   NA's:233   NA's: 26




  asthma     diabetes       comorb          disability        height
 no  :917   no  :919   Min.   :0.0000   Min.   : 0.000   Min.   : 80.0
 yes : 27   yes : 32   1st Qu.:0.0000   1st Qu.: 2.778   1st Qu.:150.0
 NA's: 26   NA's: 19   Median :0.0000   Median : 8.333   Median :153.0
                       Mean   :0.3924   Mean   :12.371   Mean   :152.7
                       3rd Qu.:1.0000   3rd Qu.:16.667   3rd Qu.:158.0
                       Max.   :2.0000   Max.   :86.111   Max.   :172.0
                       NA's   :17                        NA's   :49
```

1.2.3 slice()

Rows can also be selected by position using `slice()`.

```
bPs5 <- slice(bP, 1:5)
bPs5

# A tibble: 5 x 25
     id residence sex      age wealthQ  physical   country backPain30 agegr maritalS eduS
  <dbl> <fct>     <fct>  <dbl> <fct>    <fct>      <fct>   <fct>      <fct> <fct>    <fct>
1 27043 Urban     Female    57 Q3       mod phys~  South ~ Yes        50-59 Married~ No pr~
2 27471 Urban     Female    64 Q3       mod phys~  China   Yes        60-69 Married~ No pr~
3  5323 Rural     Female    60 Q2       high phy~  China   No         60-69 Married~ No pr~
4  9068 Rural     Female    70 Q1 poor~ low phys~  Ghana   No         70-79 Div/Wid~ No pr~
5 28828 Urban     Female    58 Q5 rich~ mod phys~  Mexico  Yes        50-59 Married~ Compl~
# ... with 14 more variables: workS <fct>, bmi <dbl>, bmi4 <fct>, waistc <dbl>,
#   smoke <fct>, alcohol <fct>, arthritis <fct>, angina <fct>, depression <fct>,
#   asthma <fct>, diabetes <fct>, comorb <dbl>, disability <dbl>, height <dbl>

bPs10 <- slice(bP,seq(1,nrow(bP), by =10))  # Select every 10th observation
head(bPs10)

# A tibble: 6 x 25
     id residence sex      age wealthQ physical  country   backPain30 agegr maritalS eduS
  <dbl> <fct>     <fct>  <dbl> <fct>   <fct>     <fct>     <fct>      <fct> <fct>    <fct>
1 27043 Urban     Female    57 Q3      mod phys~ South Af~ Yes        50-59 Married~ No p~
2  8120 Urban     Female    71 Q2      mod phys~ China     No         70-79 Married~ Comp~
3 26658 Rural     Female    63 Q4      high phy~ Russian ~ Yes        60-69 Married~ Comp~
```

```
4 21774 Rural     Male      58 Q4      high phy~ Ghana     Yes        50-59 Married~ No p~
5 17090 Urban     Male      54 Q3      high phy~ China     No         50-59 Married~ Comp~
6 30346 Urban     Female    65 Q4      low phys~ China     Yes        60-69 Married~ No p~
# ... with 14 more variables: workS <fct>, bmi <dbl>, bmi4 <fct>, waistc <dbl>,
#   smoke <fct>, alcohol <fct>, arthritis <fct>, angina <fct>, depression <fct>,
#   asthma <fct>, diabetes <fct>, comorb <dbl>, disability <dbl>, height <dbl>
```

1.2.4 `arrange()`

The function `arrange()` is used to reorder rows.You provide a column name to control the ordering; if you want to resolve ties, add more column names. Default is ascending order. Because there are many variables so we limit the example to a few of them.

```
bPa <- arrange(bP,waistc)
head(select(bPa,residence,sex,waistc,age,wealthQ),10)    # default is ascending order

Error in select(bPa, residence, sex, waistc, age, wealthQ): unused arguments (residence,
sex, waistc, age, wealthQ)
```

We can sort by more variables and choose descending order is done as follows:

```
bPa <- arrange(bP,desc(waistc),age)
head(select(bPa,residence,sex,waistc,age,wealthQ),10)

Error in select(bPa, residence, sex, waistc, age, wealthQ): unused arguments (residence,
sex, waistc, age, wealthQ)
```

1.2.5 `rename()`

Renaming columns is straightforward:

```
bPr <- rename(bP, wealthQuantile = wealthQ)  # New name = old name
head(bPr)

# A tibble: 6 x 25
     id residence sex      age wealthQuantile physical  country  backPain30 agegr maritalS
  <dbl> <fct>     <fct>  <dbl> <fct>          <fct>     <fct>    <fct>      <fct> <fct>
1 27043 Urban     Female    57 Q3             mod phys~ South A~ Yes        50-59 Married~
2 27471 Urban     Female    64 Q3             mod phys~ China    Yes        60-69 Married~
3  5323 Rural     Female    60 Q2             high phy~ China    No         60-69 Married~
4  9068 Rural     Female    70 Q1 poorest     low phys~ Ghana    No         70-79 Div/Wid~
5 28828 Urban     Female    58 Q5 richest     mod phys~ Mexico   Yes        50-59 Married~
6 23775 Urban     Female    60 Q1 poorest     high phy~ Mexico   Yes        60-69 Div/Wid~
# ... with 15 more variables: eduS <fct>, workS <fct>, bmi <dbl>, bmi4 <fct>,
#   waistc <dbl>, smoke <fct>, alcohol <fct>, arthritis <fct>, angina <fct>,
#   depression <fct>, asthma <fct>, diabetes <fct>, comorb <dbl>, disability <dbl>,
#   height <dbl>
```

1.2.6 `mutate()`

The function `mutate()` adds new columns which are calculated from old columns:

```
bPm <- mutate(bP, heightInches = height/2.54)
head(select(bPm,residence,sex,age,waistc,height,heightInches))

Error in select(bPm, residence, sex, age, waistc, height, heightInches):  unused arguments
(residence, sex, age, waistc, height, heightInches)
```

It can also be used to change/add multiple columns:

```
bPm <- mutate(bP, heightM = height/100,
              wHr = waistc/height,
              sAge = age-50)
select(bPm, residence,sex,age,height,waistc, heightM, wHr, sAge)

## Error in select(bPm, residence, sex, age, height, waistc, heightM, wHr, :  unused arguments
(residence, sex, age, height, waistc, heightM, wHr, sAge)
```

1.2.7 count()

We can summarise individual factors with counts of their levels using the function `count()`. It reminds of ftable but the result is a tibble/data frame.

```
count(bP,wealthQ)

# A tibble: 6 x 2
  wealthQ        n
  <fct>      <int>
1 Q1 poorest  1889
2 Q2          1983
3 Q3          1888
4 Q4          2054
5 Q5 richest  2159
6 <NA>          27

bPc <- count(bP,country, residence, wealthQ)
bPc

# A tibble: 66 x 4
   country residence wealthQ        n
   <fct>   <fct>     <fct>      <int>
 1 China   Rural     Q1 poorest   514
 2 China   Rural     Q2           513
 3 China   Rural     Q3           355
 4 China   Rural     Q4           335
 5 China   Rural     Q5 richest   194
 6 China   Rural     <NA>           7
 7 China   Urban     Q1 poorest   254
 8 China   Urban     Q2           249
 9 China   Urban     Q3           384
10 China   Urban     Q4           434
# ... with 56 more rows
```

The result **bPc** is a multi-way table in long format and the resulting object is a tibble. Thus all of it will not be printed out. If you want to see all of it try **as.data.frame(bPc)**

Notice the warnings are there because there are a significant number of NA in the data.

1.2.8 base::summary()

It can somtimes be useful to use the `base::summary()` function after selecting only those columns which are of interest. For factors you get one-way frequency tables.

```
summary(select(bP, residence, country, agegr,age))

Error in select(bP, residence, country, agegr, age): unused arguments (residence, country,
agegr, age)
```

1.3 Pipes

One characteristic with tidyverse is that pipes "%>%" are allowed. In this section we shall look at some examples how they are used. A pipe is a link leading from one tibble (or data frame) to another via a verb (function). A quick command for "%>%" is "ctrl shift m".

First we want to select some variables and print the first 4 rows. Compare with the earlier examples when bP was an argument.

```
bP %>% select(age,sex,wealthQ,physical,bmi) %>% head(4)

Error in select(., age, sex, wealthQ, physical, bmi): unused arguments (age, sex, wealthQ,
physical, bmi)
```

We can also in the same command set the result to a data frame (or tibble) using "<-" or "=" on the left side as usual.

```
bPp<-bP %>% select(age,sex,wealthQ,physical,bmi)  %>%
  filter(sex=="Female",age<60) %>%   arrange(bmi)

Error in select(., age, sex, wealthQ, physical, bmi): unused arguments (age, sex, wealthQ,
physical, bmi)

bPp

# A tibble: 4 x 2
  bmi2          n
  <fct>     <int>
1 (0,20]      344
2 (20,30]    1338
3 (30,Inf]    421
4 <NA>         94

bPp<-bP  %>% filter(sex=="Female",age<60) %>%
  mutate(bmi2=cut(bmi,c(0,20,30,Inf))) %>% count(bmi2)
bPp

# A tibble: 4 x 2
  bmi2          n
  <fct>     <int>
1 (0,20]      344
2 (20,30]    1338
3 (30,Inf]    421
4 <NA>         94
```

```
bPp<-bP %>%  filter(sex=="Female",age<60) %>%
  mutate(bmi2=cut(bmi,c(0,20,30,Inf)),agesq=age^2) %>% slice(1:5) %>%
  select(-c(5,7:11,13:24))

Error in select(., -c(5, 7:11, 13:24)):  unused argument (-c(5, 7:11, 13:24))

bPp

# A tibble: 4 x 2
  bmi2          n
  <fct>     <int>
1 (0,20]      344
2 (20,30]    1338
3 (30,Inf]    421
4 <NA>         94
```

1.4 Manipulating factors with `tidyverse tools`

Factors (basically categorical variables) are somewhat complicated objects but they can be very useful if we can learn how handle them.

We have already seen how to do the conversion of all character variables in a dataframe to factors:

```
bP <- mutate_if(bP, is.character, as.factor)
```

We note that this should be done with care, because there will certainly be times when converion of character variables to factors is inappropriate - for which reason, the tidyverse dataframe functions do not automatically convert character variables to factors, which is the default in older R functions.

Here we will take a closer look at how to manipulate factors. We'll begin here by selecting a factor variable from the backPain data frame - country, in which the order of the 'levels' (categories) of the factor is not significant (nominal variable). Notice that R saves storage of factors by saving them as numerics and relating the numerics to the levels.

```
levels(bP$country)

[1] "China"              "Ghana"              "India"              "Mexico"
[5] "Russian Federation" "South Africa"

str(select(bP, country) )

Error in select(bP, country):  unused argument (country)

bP %>% select(country) %>% mutate(numeric.country=as.numeric(country)) %>% slice(1:5)

Error in select(., country):  unused argument (country)

  # print numeric values of first 5 elements
```

Often coding for levels (input codes) is abbreviated (or sometimes more lengthy than we might like for display). By default, when plotting the level names are used on the plot. If the names are not suitable, we can then define a more suitable name for the level (we 're-code' it) and then the new code name will be used in the plots and at the same time R will re-name the level in the dataframe.

There is considerable confusion about the use of 'labels' with levels and `tidyverse` offers tidier (and clearer) solutions!

Basically we want to be able to do four things:

1. Coerce numeric and/or character variables to factors where it is appropriate,
2. Re-order to something more sensible. The default alphabetical order of the levels is often unsatisfactory for presentation purposes, either in a table or a plot. Use `fct_relevel`.
3. Re-name unnecessarily terse, long or meaningless names. Use `fct_recode`
4. Cut a numeric variable into named groups to create a new factor variable. Use one of:
   - `cut_interval` which makes n groups with equal range,
   - `cut_number` which makes n groups each with approximately equal numbers of observations or
   - `cut_width`, which makes groups with a specified 'bin width'

You can see examples of the cut functions here.

1.4.1 Coercing numeric and character variables to a factor

Here is how to coerce a simple numeric variable. (In this case `comorb` is coded as the number of comorbidities, 0, 1, 2 - where 2 may be 2 or more.)

```
bP %>% select(comorb) %>% str()

Error in select(., comorb):  unused argument (comorb)

bP %>% select(comorb) %>% mutate(comorb=as.factor(comorb)) %>% str()

Error in select(., comorb):  unused argument (comorb)
```

1.4.2 Changing the codes of a factor

We sometimes need to change the codes to give us meaningful labels for presentation, we use **fct_recode**. The recoding definition in the `fct_recode` function is "new name" = "old name". We start with converting comorb to a factor

```
bPm<-bP %>% mutate(comorb2=as.factor(comorb))%>%
  mutate(comorb3 = fct_recode(comorb2,"None" = "0","One" = "1","Two or more"= "2"))
bPm %>% select(comorb,comorb2,comorb3) %>% slice(1:5)

Error in select(., comorb, comorb2, comorb3):  unused arguments (comorb, comorb2, comorb3)
```

1.4.3 Re-coding (renaming) categories (levels) in factors

If you simply want to change the coding of some of the levels, this is also easily done with **fct_recode**. Here is the call to reduce the length of the Russian Federation code and South Africa.

```
bP %>% count(country)
```

```
# A tibble: 6 x 2
  country                n
  <fct>              <int>
1 China               3810
2 Ghana               1300
3 India               1945
4 Mexico               661
5 Russian Federation  1169
6 South Africa        1115
```

```
bPr <- bP %>%
  mutate(country = fct_recode(country,
                              "Russian Fed" = "Russian Federation",
                              "Sth Africa" = "South Africa") )
bPr %>% count(country)
```

```
# A tibble: 6 x 2
  country         n
  <fct>       <int>
1 China        3810
2 Ghana        1300
3 India        1945
4 Mexico        661
5 Russian Fed  1169
6 Sth Africa   1115
```

If you look back at the preceding script you will see that there we had the level name "Russian Federation". We've changed the associated level name to "Russian Fed" and a similar change was made to "South Africa"

1.4.4 Re-ordering factor levels

To re-order, you must specify the levels in your desired order as the arguments of **fct_relevel** following the variable name. The spelling must be exactly as in the original!

```
levels(bP$bmi4)
```

```
[1] "Normal"      "Obese"       "Pre-Obese"   "Underweight"
```

```
 bPm <- bP %>%  mutate(bmi4 = fct_relevel(bmi4, "Underweight","Normal", "Pre-Obese", "Obese"))
 levels(bPm$bmi4)
```

```
[1] "Underweight" "Normal"      "Pre-Obese"   "Obese"
```

1.4.5 Converting numeric to grouped factors using the `cut_` functions

The function `cut()` provides a quick way of converting numeric data to grouped factors. The `forcats` package (included in `tidyverse`) simplifies and extends these calls a little.

Here's how to create 6 levels with approximately the same number of observations in each group:

```
bP %>% mutate(height6 =  cut_number(height, n = 6)) %>% count(height6)
```

```
# A tibble: 7 x 2
```

```
  height6       n
  <fct>     <int>
1 [65,150]   1174
2 (150,156]   967
3 (156,160]  1124
4 (160,165]  1040
5 (165,170]   917
6 (170,210]   662
7 <NA>       4116
```

We use `cut_interval` to cut with approximately equal ranges:

```
bPc4<-bP %>% mutate(height4 = cut_interval(height, n=4))
bPc4 %>% count(height4)

# A tibble: 5 x 2
  height4       n
  <fct>     <int>
1 [65,101]     65
2 (101,138]   111
3 (138,174]  5264
4 (174,210]   444
5 <NA>       4116
```

.... and `cut_width` let's you choose the range.

```
bPc <- bP %>% mutate(height4 = cut_width(height, width = 20))
summary(bPc$height4)

  [50,70]    (70,90]   (90,110]  (110,130]  (130,150]  (150,170]  (170,190]  (190,210]       NA's
        1         43         28         64       1038       4048        635         27       4116
```

If you want to choose your own breaks, you must revert to the base R function `cut`. We can also add labels to all the **cut_** functions above.

```
bPc <- bP %>% mutate(height4 = cut(height, breaks = c(0,120,150,170,Inf)))
summary(bPc$height4)

  (0,120] (120,150] (150,170] (170,Inf]      NA's
      110      1064      4048       662      4116

bPc <- bP %>% mutate(height4 = cut(height, breaks = c(0,120,150,170,Inf),
                  labels = c("Very Short", "Short", "Average", "Tall" )))
bPc %>% count(height4)

# A tibble: 5 x 2
  height4        n
  <fct>      <int>
1 Very Short   110
2 Short       1064
3 Average     4048
4 Tall         662
5 <NA>        4116

bPc %>% mutate(height4=fct_explicit_na(height4, na_level = "(Missing)"))  %>% count(height4)

# A tibble: 5 x 2
```

```
  height4        n
  <fct>      <int>
1 Very Short   110
2 Short       1064
3 Average     4048
4 Tall         662
5 (Missing)   4116
```

..and finally, here's how to combine levels. Simply recode grouped levels to the same name:

```
bPc6<-bP %>% mutate(height6 =  cut_number(height, n = 6,
labels = c("Very Short", "Short", "Average", "Tall", "Very Tall", "Extremely Tall")))

summary(bPc6$height6)

##     Very Short          Short        Average           Tall      Very Tall Extremely Tall
##           1174            967           1124           1040            917            662
##           NA's
##           4116

bP64 <- bPc6 %>%  mutate(h6_to_4 = fct_recode(height6,
                                          "short" = "Very Short",
                                          "short" = "Short",
                                          "very tall" = "Very Tall",
                                          "very tall" = "Extremely Tall"))
xtabs(~h6_to_4+height6,data=bP64)

##            height6
## h6_to_4     Very Short Short Average Tall Very Tall Extremely Tall
##   short           1174   967       0    0         0              0
##   Average            0     0    1124    0         0              0
##   Tall               0     0       0 1040         0              0
##   very tall          0     0       0    0       917            662
```

The package `forcats` is also very useful for dealing wth factors. See Chapter 15 of 'R for data science' [**Wickham2017**] for detailed examples and extensions on the use of the functions we have seen here.

1.5 `group_by()`

The `summarise()` function and those we have discussed above, become much more powerful when we use grouping operations with the verb/function `group_by()`.

The other verbs are affected by grouping as follows:

- Grouped `select()` is the same as ungrouped select(), excepted that grouping variables are always retained.
- Grouped `arrange()` orders first by grouping variables
- The `slice()` function extracts rows within each group.
- The `count()` function counts the number of rows with each unique value of variable, so it is particularly useful for counting the frequency of levels in factors.
- The `summarise()` function is particularly useful when applied to grouped variables, and is explained in detail below.

1.5.1 Summarising groups

In summarising groups we can add columns containing the statistics (mean, sd, max, IQR etc) for every group combination of the set specified. In our bP data, we group using factor variables.

```
bPs <- bP %>%
  group_by(residence) %>%
  summarise(mean.disability = mean(disability))
bPs

# A tibble: 2 x 2
  residence mean.disability
  <fct>               <dbl>
1 Rural                19.4
2 Urban                16.0
```

Here's a further example in which we add more groups and statistics for more variables.

```
bPs <- bP %>% group_by(country,residence, sex) %>%
  summarise(mean.disability = mean(disability), disIQR = IQR(disability),
  Bmi = mean(bmi,na.rm=T))

`summarise()` has grouped output by 'country', 'residence'.  You can override using the
`.groups` argument.

bPs

# A tibble: 24 x 6
# Groups:   country, residence [12]
   country residence sex    mean.disability disIQR   Bmi
   <fct>   <fct>     <fct>            <dbl>  <dbl> <dbl>
 1 China   Rural     Female           12.4   13.9   23.8
 2 China   Rural     Male              8.57  11.1   23.1
 3 China   Urban     Female            8.16  11.1   24.8
 4 China   Urban     Male              6.62   8.33  24.6
 5 Ghana   Rural     Female           27.4   25     22.2
 6 Ghana   Rural     Male             19.2   25.0   21.3
 7 Ghana   Urban     Female           25.9   30.6   25.3
 8 Ghana   Urban     Male             18.9   30.6   24.4
 9 India   Rural     Female           33.6   25     20.7
10 India   Rural     Male             24.9   25     19.8
# ... with 14 more rows
```

Observe the na.rm=T which was necessary because of missing values.

We may want to change the order in the table.

```
bPs <- bP %>% group_by(sex,country,residence) %>%
  summarise(mean.disability = mean(disability), disIQR = IQR(disability), Bmi = mean(bmi,na.rm=T))

`summarise()` has grouped output by 'sex', 'country'.  You can override using the `.groups`
argument.

bPs

# A tibble: 24 x 6
# Groups:   sex, country [12]
   sex    country             residence mean.disability disIQR   Bmi
```

```
   <fct>  <fct>              <fct>              <dbl>  <dbl> <dbl>
 1 Female China              Rural              12.4    13.9  23.8
 2 Female China              Urban               8.16   11.1  24.8
 3 Female Ghana              Rural              27.4    25    22.2
 4 Female Ghana              Urban              25.9    30.6  25.3
 5 Female India              Rural              33.6    25    20.7
 6 Female India              Urban              27.8    27.8  22.4
 7 Female Mexico             Rural              22.2    27.8  27.6
 8 Female Mexico             Urban              19.7    25.0  29.1
 9 Female Russian Federation Rural              22.1    25.0  30.9
10 Female Russian Federation Urban              23.8    27.8  29.3
# ... with 14 more rows
```

..or filter to look at only one country.

```
bPs <- bP %>%
  filter(country== "China")%>%
  group_by(sex,residence) %>%
  summarise(mean.disability = mean(disability), disIQR = IQR(disability),
  Bmi = mean (bmi,na.rm=T))

`summarise()` has grouped output by 'sex'.  You can override using the `.groups` argument.

bPs

# A tibble: 4 x 5
# Groups:   sex [2]
  sex    residence mean.disability disIQR   Bmi
  <fct>  <fct>               <dbl>  <dbl> <dbl>
1 Female Rural               12.4    13.9   23.8
2 Female Urban                8.16   11.1   24.8
3 Male   Rural                8.57   11.1   23.1
4 Male   Urban                6.62    8.33  24.6
```

When groups vary significantly in size it is prudent to include counts of observations.

```
bPs <- group_by(bP, country, sex, residence, wealthQ) %>%
  summarise( count=n(), mean.disability=mean(disability))

`summarise()` has grouped output by 'country', 'sex', 'residence'.  You can override using
the `.groups` argument.

bPs

# A tibble: 130 x 6
# Groups:   country, sex, residence [24]
   country sex    residence wealthQ    count mean.disability
   <fct>   <fct>  <fct>     <fct>      <int>           <dbl>
 1 China   Female Rural     Q1 poorest   277           15.7
 2 China   Female Rural     Q2           260           12.4
 3 China   Female Rural     Q3           172           11.8
 4 China   Female Rural     Q4           156            9.99
 5 China   Female Rural     Q5 richest    99            7.10
 6 China   Female Rural     <NA>           6           24.5
 7 China   Female Urban     Q1 poorest   137           16.7
 8 China   Female Urban     Q2           146           10.8
 9 China   Female Urban     Q3           210            8.90
10 China   Female Urban     Q4           237            5.92
# ... with 120 more rows
```

1.6 `left_join()`: Merging two data frames

There is a set of functions in `dplyr` for merging data frames. Here we'll just demonstrate the `left_join()` function.

We'll first create a data set iDf with individual ID (id) and a household ID (hhID). Then we create a second data set on households (hhDf) which will relate to the the first data set through the household ID (hhID) variable.

```
ID <- 1:15
hhID <- c(1,1,1,1,1,2,2,3,3,3,4,4,4,4,4)
iData1 <- LETTERS[1:15]
iData2 <- letters[12:26]

iDf <- data.frame(id = as.factor(ID),
                  hhID = as.factor(hhID),
                  iD1 = iData1,
                  iD2 = iData2)
iDf
```

```
##    id hhID iD1 iD2
## 1   1    1   A   l
## 2   2    1   B   m
## 3   3    1   C   n
## 4   4    1   D   o
## 5   5    1   E   p
## 6   6    2   F   q
## 7   7    2   G   r
## 8   8    3   H   s
## 9   9    3   I   t
## 10 10    3   J   u
## 11 11    4   K   v
## 12 12    4   L   w
## 13 13    4   M   x
## 14 14    4   N   y
## 15 15    4   O   z
```

```
hhID <- 1:4
hData1 <- c("X1", "X2", "X3", "X4")
hData2 <- letters[5:8]
hhDf <- data.frame(hhID = as.factor(hhID),
                   hD1 = hData1,
                   hD2 = hData2)
hhDf
```

```
##   hhID hD1 hD2
## 1    1  X1   e
## 2    2  X2   f
## 3    3  X3   g
## 4    4  X4   h
```

Now let's see if we can create a combined data frame in which the individual data frame rows are maintained, but have added to them the variables from the household data frame with values of those variables corresponding to the household listed in the individual's data frame.

We'll firstly try a left_join using the common household ID (hhID) as the 'key'. Notice how data in each household are repeted for individuals in the same household.

```
merged <- left_join(iDf, hhDf, by="hhID")
merged

##    id hhID iD1 iD2 hD1 hD2
## 1   1    1   A   l  X1   e
## 2   2    1   B   m  X1   e
## 3   3    1   C   n  X1   e
## 4   4    1   D   o  X1   e
## 5   5    1   E   p  X1   e
## 6   6    2   F   q  X2   f
## 7   7    2   G   r  X2   f
## 8   8    3   H   s  X3   g
## 9   9    3   I   t  X3   g
## 10 10    3   J   u  X3   g
## 11 11    4   K   v  X4   h
## 12 12    4   L   w  X4   h
## 13 13    4   M   x  X4   h
## 14 14    4   N   y  X4   h
## 15 15    4   O   z  X4   h
```

There is also a base R function **base::merge** that can be used to merge (link) data in a similar way.

1.7 Tidyr functions

1.7.1 Functions for converting between long and wide format

Since data can be stored in different ways there is sometimes a need to convert the data to the desired form. Two types of storage is wide and long format. Let us think of an example where timber volume (cubic meters) is measured whith three different methods in three areas. The data can then be stored as wide format where the volume for each method is represented as separate variables (vectors). The functions we will use are from the tidyr package.

```
[1] "Wide format"
  area method1 method2 method3
1    1     210     242     207
2    2     135     135     111
3    3     187     201     214
```

In long format the vectors for each of the three methods are stacked. This means that there will be three times as many rows in the new data frame. We now only need one column for the volume values but we also need a column (here called method) with information on the type of method.

```
[1] "Long format"
  area method volume
1    1      1    210
2    2      1    135
3    3      1    187
4    1      2    242
5    2      2    135
6    3      2    201
7    1      3    207
8    2      3    111
9    3      3    214
```

The functions used are **gather** and **spread** which can be thought of as inverse functions of each other. Observe that we need to give two new variable names: **key** which holds the information of the kind of data (methodx) here chosen as `method` and **value** (volume of methodx) here chosen as `volume` while `area` is kept.

```
xwide<-data.frame(area=c(1:3),method1=c(210,135,187),method2=c(242,135,201),
                  method3=c(207,111,214))
xwide

  area method1 method2 method3
1    1     210     242     207
2    2     135     135     111
3    3     187     201     214

xlong<-xwide %>% gather(key=method,value=volume,method1,method2,method3)
xlong

  area  method volume
1    1 method1    210
2    2 method1    135
3    3 method1    187
4    1 method2    242
5    2 method2    135
6    3 method2    201
7    1 method3    207
8    2 method3    111
9    3 method3    214

xlong2<-xwide %>% gather(key=method,value=volume,-area)
    # Alternative for the same result, all variables stacked except area

xlong %>% spread(key=method,value=volume)

  area method1 method2 method3
1    1     210     242     207
2    2     135     135     111
3    3     187     201     214
```

...and so we are back at the data frame we started with.

1.7.2 Example - the dataset Subliminal

Let us first import the dataset. Data in the dataset Subliminal are from an intervention study where 18 students were randomized to receive either of two messages with the intention to see if this would affect their performance on the mathematics exam. The control group received neutral messages whereas the intervention group received messages confirming their learning process. All students participated in a summer school in mathematics and were tested at the begining and end of the intervention. The dataset contains the following variables:

Message:	If the student received neutral or confirmatory messages
Before:	Test result at the beginning of the study
After:	Test result at the end of the study
Improvement:	Improvement of their results (After-Before)

Observe that we need to give two new variable names **key** which holds the information of the kind of data (variable) here chosen as `Time` and **value** here chosen as `Result` while `Message`, `Improvement` and `ind.nr` is kept.

```
library(haven)
Subliminal <- read_sav("../data/Subliminal.sav")

sub<-Subliminal %>% cbind(ind.nr=1:nrow(Subliminal)) # add individual number
sub %>% slice(1:5)

   Message Before After Improvement ind.nr
1 positive     18    24           6      1
2 positive     18    25           7      2
3 positive     21    33          12      3
4 positive     18    29          11      4
5 positive     18    33          15      5

nrow(sub)

[1] 18

sub_lf<-sub %>% gather(key=Time,value=Result,Before,After)
sub_lf %>% arrange(ind.nr) %>% filter(ind.nr<=5)

    Message Improvement ind.nr   Time Result
1  positive           6      1 Before     18
2  positive           6      1  After     24
3  positive           7      2 Before     18
4  positive           7      2  After     25
5  positive          12      3 Before     21
6  positive          12      3  After     33
7  positive          11      4 Before     18
8  positive          11      4  After     29
9  positive          15      5 Before     18
10 positive          15      5  After     33

nrow(sub_lf)

[1] 36

sub_wf<-sub_lf %>% spread(key=Time,value=Result) %>% arrange(ind.nr)

head(sub_wf)

   Message Improvement ind.nr After Before
1 positive           6      1    24     18
2 positive           7      2    25     18
3 positive          12      3    33     21
4 positive          11      4    29     18
5 positive          15      5    33     18
6 positive          16      6    36     20

nrow(sub_wf)

[1] 18
```

In fact the different variables that is gathered do not have to be repeated measurements of the same kind. It can be completely different measures. Let us see an example using the sample of the BackPain data.

```
bPg<- bP %>% gather(key=var,value=value,disability,bmi,age)
bPg %>% count(var)

# A tibble: 3 x 2
  var            n
  <chr>      <int>
1 age        10000
2 bmi        10000
3 disability 10000

# the first three rows in each group
bPg %>% group_by(var) %>% select(var,value,residence,sex,wealthQ,physical,country) %>% slice(1:3)

Error in select(., var, value, residence, sex, wealthQ, physical, country):  unused arguments
(var, value, residence, sex, wealthQ, physical, country)
```

Own experimentation

The last tibble bP is cut to show only the first three rows per group. Check if it looks ok also further down. Can you restore it to the original by use of spread? Try out other combinations of variables when using gather.