Programming for Data Analytics

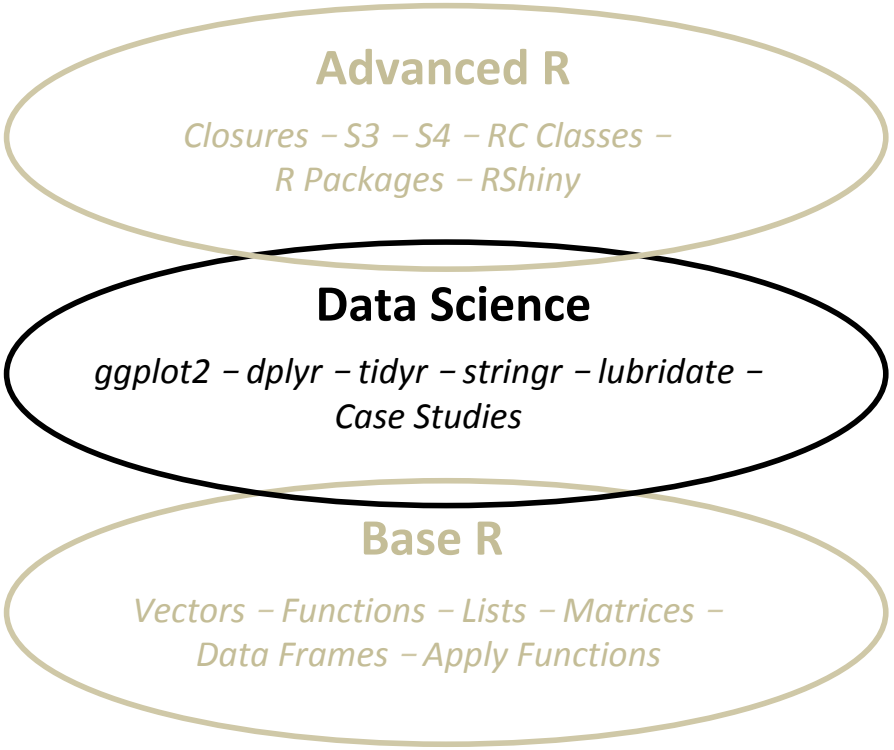
7. Relational data with dplyr and tidyr

Prof. Jim Duggan,
School of Computer Science
National University of Ireland Galway.

https://github.com/JimDuggan/CT5102

Lecture Overview

- Relational data in dplyr
- Mutating joins
- Filtering joins
- tidyr overview
  - pivot_longer()
  - pivot_wider()
  - separate()



(1) Relational Data with dplyr

- Typically, data analysis involves many tables of data that must be combined to answer questions
- Collectively, multiple tables of data are called *relational data*
- Relations are always defined between a pair of tables

key	val_x
1	x1
2	x2
3	x3

key	val_y
1	y1
2	y2
4	y3

Keys

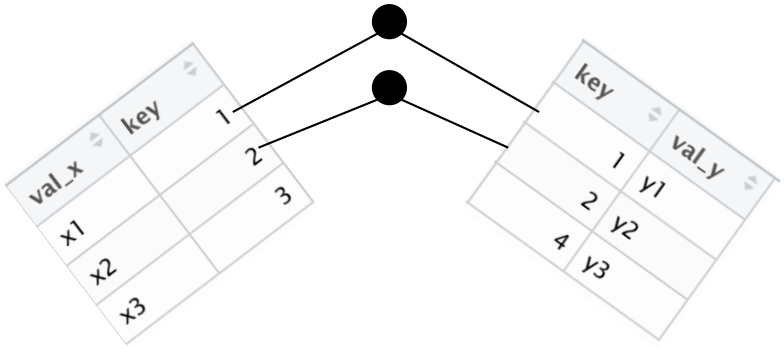
- The variables used to connect each pair of tables are called keys
- **A key is a variable (or set of variables) that uniquely identifies an observation**
- There are two types of keys:
  - A ***primary key*** uniquely identifies an observation in its own table
  - A ***foreign key*** uniquely identifies an observation in another table.

(2) Mutating Joins

- Allows you to combine variables from two tables
- First matches observations by their keys, and then copies across variables from one table to another
- Similar to mutate(), the join functions add variables to the right

Join Types

- Inner Join:
  - matches pairs of observations when their keys are equal
  - Unmatched rows are not included in the result



key	val_x	val_y
1	x1	y1
2	x2	y2

```
inner_join(x,y,by="key")
```

Outer Joins

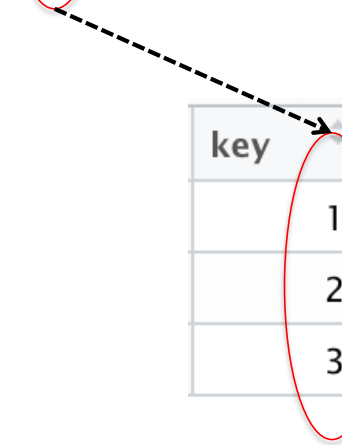
- An outer join keeps observations that appear in at least one of the tables. There are three types of outer joins (x,y)
  - A *left join* keeps all observations in x
  - A *right join* keeps all observations in y
  - A *full join* keeps all observations in x and y

Left Join

```
left_join(x,y,by="key")
```

key	val_x
1	x1
2	x2
3	x3

key	val_y
1	y1
2	y2
4	y3



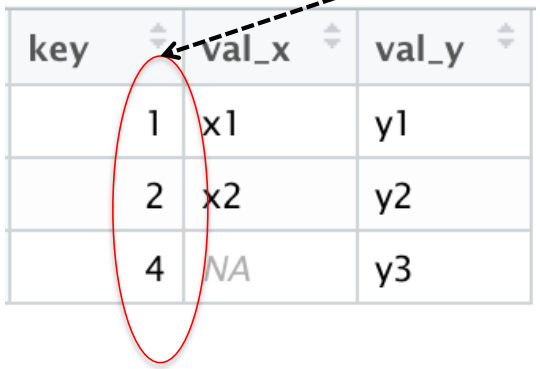
key	val_x	val_y
1	x1	y1
2	x2	y2
3	x3	*NA*

Right Join

```
right_join(x,y,by="key")
```

key	val_x
1	x1
2	x2
3	x3

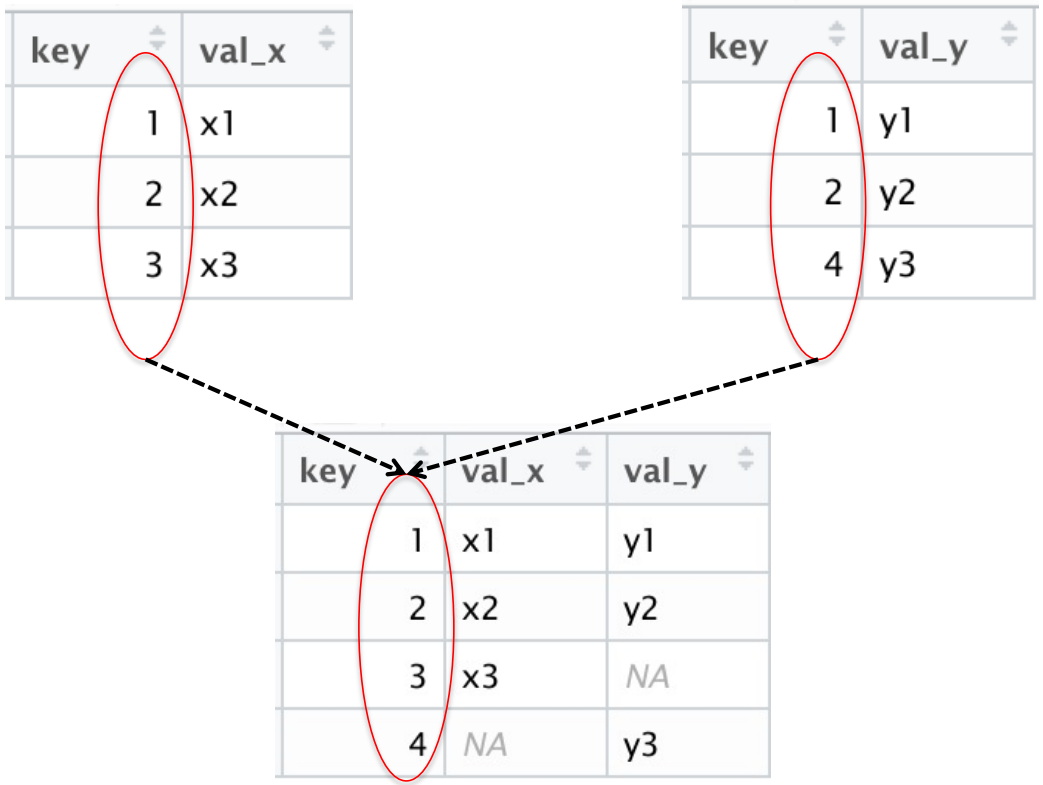
key	val_y
1	y1
2	y2
4	y3



key	val_x	val_y
1	x1	y1
2	x2	y2
4	*NA*	y3

Full Join

```
full_join(x,y,by="key")
```



key	val_x
1	x1
2	x2
3	x3

key	val_y
1	y1
2	y2
4	y3

key	val_x	val_y
1	x1	y1
2	x2	y2
3	x3	*NA*
4	*NA*	y3

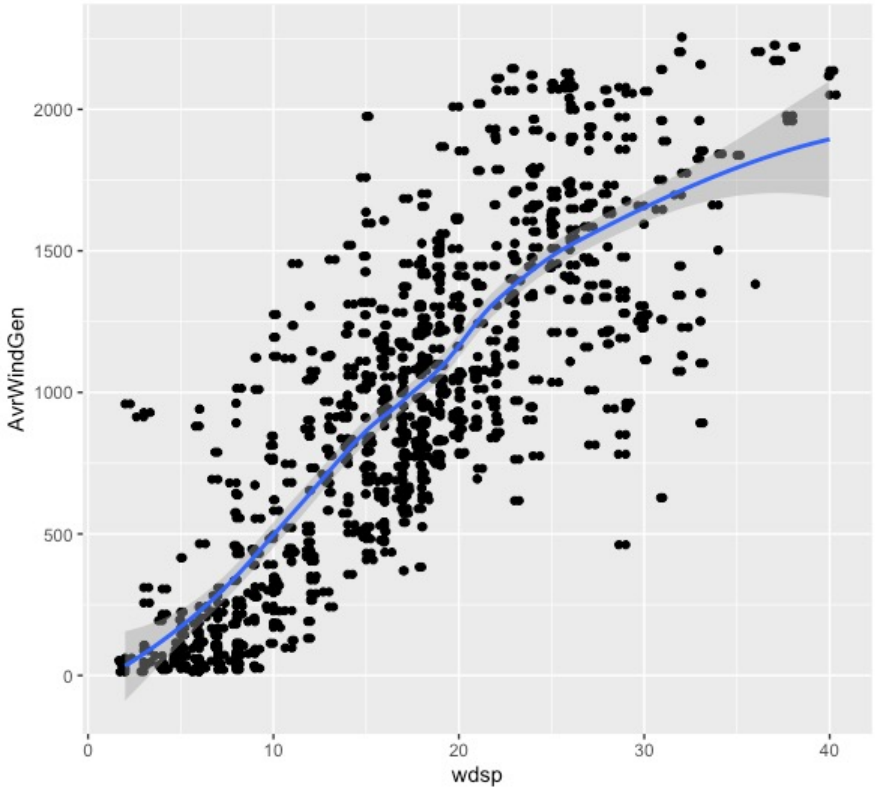
Aimsir17 – common keys

```
> glimpse(observations)
Rows: 219,000
Columns: 12
$ station <chr> "ATHENRY", "ATHENRY", "ATH…
$ year    <dbl> 2017, 2017, 2017, 2017, 20…
$ month   <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1,…
$ day     <int> 1, 1, 1, 1, 1, 1, 1, 1, 1,…
$ hour    <int> 0, 1, 2, 3, 4, 5, 6, 7, 8,…
$ date    <dttm> 2017-01-01 00:00:00, 2017…
$ rain    <dbl> 0.0, 0.0, 0.0, 0.1, 0.1, 0…
$ temp    <dbl> 5.2, 4.7, 4.2, 3.5, 3.2, 2…
$ rhum    <dbl> 89, 89, 90, 87, 89, 91, 89…
$ msl     <dbl> 1021.9, 1022.0, 1022.1, 10…
$ wdsp    <dbl> 8, 9, 8, 9, 8, 8, 7, 7, 7,…
$ wddir   <dbl> 320, 320, 320, 330, 330, 3…
```

```
> glimpse(eirgrid17)
Rows: 35,040
Columns: 15
$ year                <dbl> 2017, 2017, 201…
$ month               <dbl> 1, 1, 1, 1, 1, …
$ day                 <int> 1, 1, 1, 1, 1, …
$ hour                <int> 0, 0, 0, 0, 1, …
$ minute              <int> 0, 15, 30, 45, …
$ date                <dttm> 2017-01-01 00:…
$ NIGeneration        <dbl> 889.005, 922.23…
$ NIDemand            <dbl> 775.931, 770.23…
$ NIWindAvailability  <dbl> 175.065, 182.86…
$ NIWindGeneration    <dbl> 198.202, 207.76…
$ IEGeneration        <dbl> 3288.57, 3282.1…
$ IEDemand            <dbl> 2921.44, 2884.1…
$ IEWindAvailability  <dbl> 1064.79, 965.60…
$ IEWindGeneration    <dbl> 1044.72, 957.74…
$ SNSP                <chr> "28.4%", "26.4%…
```

Challenge 7.1 – left_join

- For October 2017, filter all observations for mace head
- Create hourly observations (mean) of wind energy generated
- Join the data sets
- Plot the wind speed v wind energy generated, and add a model (also use jitter to show more points)



(3) Filtering Joins

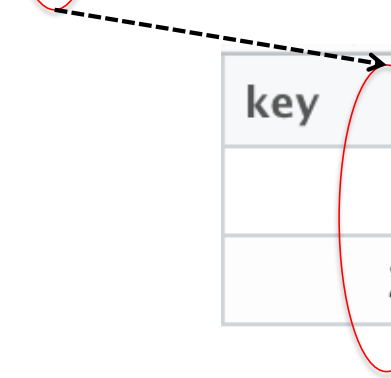
- Match observations in the same way as mutating joins, but affect the observations, not the variables
- Two types:
  - semi_join(x,y) keeps all observations in x that have a match in y
  - anti_join(x,y), drops all observations in x that have a match in y.

Semi Join

```
semi_join(x,y,by="key")
```

key	val_x
1	x1
2	x2
3	x3

key	val_y
1	y1
2	y2
4	y3



key	val_x
1	x1
2	x2

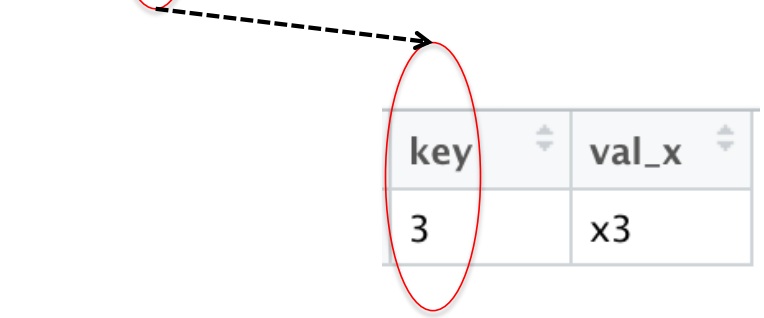
keeps all observations in x that have a match in y

Anti Join

```
anti_join(x,y,by="key")
```

key	val_x
1	x1
2	x2
3	x3

key	val_y
1	y1
2	y2
4	y3



key	val_x
3	x3

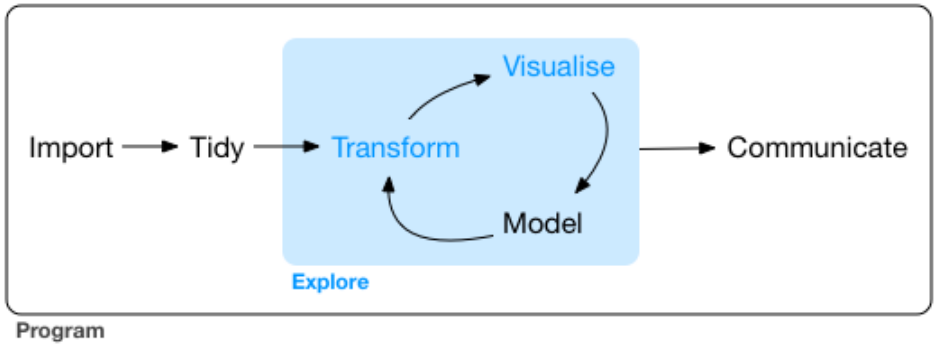
drops all observations in x that have a match in y

Challenge 7.2

- Explore the relationship between minimum daily temperature and maximum daily electricity demand
- Use three weather stations as examples: BELMULLET, DUBLIN AIPRORT and VALENTIA OBSERVATORY.

(4) Tidy Data - Overview

- What is data tidying?
  - Structuring datasets to facilitate analysis
- The tidy data standard is designed to:
  - Facilitate initial exploration and analysis of data
  - Simplify the development of data analysis tools that work well together
- Principles closely related to relational algebra (Codd 1990)
- Related packages: tidyr, ggplot2, dplyr



Why tidy data? (Wickham et al. p150)

- Advantage to picking one consistent way of storing data. Easier to learn tools that work with tidy data because they have a underlying uniformity
- Specific advantage to placing variables in columns because it allows R's vectorised functions to shine.
- dplyr, ggplot2 designed to work with tidy data

Rules for a Tidy Dataset

- Each variable must have its own column
- Each observation must have its own row
- Each value must have its own cell

- *Put every dataset in a tibble*
- *Put each variable in a column*

https://rpubs.com/bradleyboehmke/data_wrangling

Example in R – untidy data

```
> ex
# A tibble: 51 x 11
   StudentID CX1000 CX1001 CX1002 CX1003 CX1004 CX1005 CX1006 CX1007 CX1008 CX1009
       <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>
 1   1111111     56     51     78     85     63     45     55     59     52     76
 2   1111112     56     64     68     80     70     39     46     60     55     74
 3   1111113     52     61     63     81     71     49     54     61     54     76
 4   1111114     50     42     72     81     63     44     62     59     56     68
 5   1111115     67     53     77     84     65     52     63     62     52     71
 6   1111116     45     57     62     32     61     56     62     51     55     79
 7   1111117     67     58     54     77     75     44     58     62     57     77
 8   1111118     69     50     66     78     72     39     60     58     57     84
 9   1111119     70     56     62     80     71     52     60     63     54     70
10   1111120     51     52     46     82     74     42     66     63     55     73
# … with 41 more rows
```

Identify the variables

Variables

- Student
- Subject
- Result

```
> ex
# A tibble: 51 x 11
   StudentID CX1000 CX1001 CX1002 CX1003 CX1004 CX1005 CX1006 CX1007 CX1008 CX1009
       <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>
 1   1111111     56     51     78     85     63     45     55     59     52     76
 2   1111112     56     64     68     80     70     39     46     60     55     74
 3   1111113     52     61     63     81     71     49     54     61     54     76
 4   1111114     50     42     72     81     63     44     62     59     56     68
 5   1111115     67     53     77     84     65     52     63     62     52     71
 6   1111116     45     57     62     32     61     56     62     51     55     79
 7   1111117     67     58     54     77     75     44     58     62     57     77
 8   1111118     69     50     66     78     72     39     60     58     57     84
 9   1111119     70     56     62     80     71     52     60     63     54     70
10   1111120     51     52     46     82     74     42     66     63     55     73
# … with 41 more rows
```

https://rpubs.com/bradleyboehmke/data_wrangling

The goal…

```
> ex
# A tibble: 51 x 11
   StudentID CX1000 CX1001 CX1002 CX1003 CX1004 CX1005 CX1006 CX1007 CX1008 CX1009
       <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>
 1   1111111     56     51     78     85     63     45     55     59     52     76
 2   1111112     56     64     68     80     70     39     46     60     55     74
 3   1111113     52     61     63     81     71     49     54     61     54     76
 4   1111114     50     42     72     81     63     44     62     59     56     68
 5   1111115     67     53     77     84     65     52     63     62     52     71
 6   1111116     45     57     62     32     61     56     62     51     55     79
 7   1111117     67     58     54     77     75     44     58     62     57     77
 8   1111118     69     50     66     78     72     39     60     58     57     84
 9   1111119     70     56     62     80     71     52     60     63     54     70
10   1111120     51     52     46     82     74     42     66     63     55     73
# … with 41 more rows
```

```
# A tibble: 510 x 3
   StudentID Subject Grade
       <dbl> <chr>   <dbl>
 1   1111111 CX1000     56
 2   1111111 CX1001     51
 3   1111111 CX1002     78
 4   1111111 CX1003     85
 5   1111111 CX1004     63
 6   1111111 CX1005     45
 7   1111111 CX1006     55
 8   1111111 CX1007     59
 9   1111111 CX1008     52
10   1111111 CX1009     76
# … with 500 more rows
```

tidyr package – sample functions of data tidying

- **pivot_longer()** takes multiple columns, and gathers them into key-value pairs: it makes “wide” data longer
- **pivot_wider()** takes two columns (key and value) and spreads into multiple columns, it makes long data wider
- **separate()** splits a single column into multiple columns

pivot_longer()

`pivot_longer()` "lengthens" data, increasing the number of rows and decreasing the number of columns. The inverse transformation is `pivot_wider()`

Learn more in `vignette("pivot")`.

```
pivot_longer(
  data,
  cols,
  names_to = "name",
  names_prefix = NULL,
  names_sep = NULL,
  names_pattern = NULL,
  names_ptypes = list(),
  names_transform = list(),
  names_repair = "check_unique",
  values_to = "value",
  values_drop_na = FALSE,
  values_ptypes = list(),
  values_transform = list(),
  ...
)
```

Arguments

data A data frame to pivot.

cols <`tidy-select`> Columns to pivot into longer format.

names_to A string specifying the name of the column to create from the data stored in the column names of `data`.

Can be a character vector, creating multiple columns, if `names_sep` or `names_pattern` is provided. In this case, there are two special values you can take advantage of:

- `NA` will discard that component of the name.
- `.value` indicates that component of the name defines the name of the column containing the cell values, overriding `values_to`.

values_to A string specifying the name of the column to create from the data stored in cell values. If `names_to` is a character containing the special `.value` sentinel, this value will be ignored, and the name of the value column will be derived from part of the existing column names.

For example…

```
> ex
# A tibble: 51 x 11
   StudentID CX1000 CX1001 CX1002 CX1003 CX1004 CX1005 CX1006 CX1007 CX1008 CX1009
       <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>
 1   1111111     56     51     78     85     63     45     55     59     52     76
 2   1111112     56     64     68     80     70     39     46     60     55     74
 3   1111113     52     61     63     81     71     49     54     61     54     76
 4   1111114     50     42     72     81     63     44     62     59     56     68
 5   1111115     67     53     77     84     65     52     63     62     52     71
 6   1111116     45     57     62     32     61     56     62     51     55     79
 7   1111117     67     58     54     77     75     44     58     62     57     77
 8   1111118     69     50     66     78     72     39     60     58     57     84
 9   1111119     70     56     62     80     71     52     60     63     54     70
10   1111120     51     52     46     82     74     42     66     63     55     73
# … with 41 more rows
```

```
# A tibble: 510 x 3
   StudentID Subject Grade
       <dbl> <chr>   <dbl>
 1   1111111 CX1000     56
 2   1111111 CX1001     51
 3   1111111 CX1002     78
 4   1111111 CX1003     85
 5   1111111 CX1004     63
 6   1111111 CX1005     45
 7   1111111 CX1006     55
 8   1111111 CX1007     59
 9   1111111 CX1008     52
10   1111111 CX1009     76
# … with 500 more rows
```

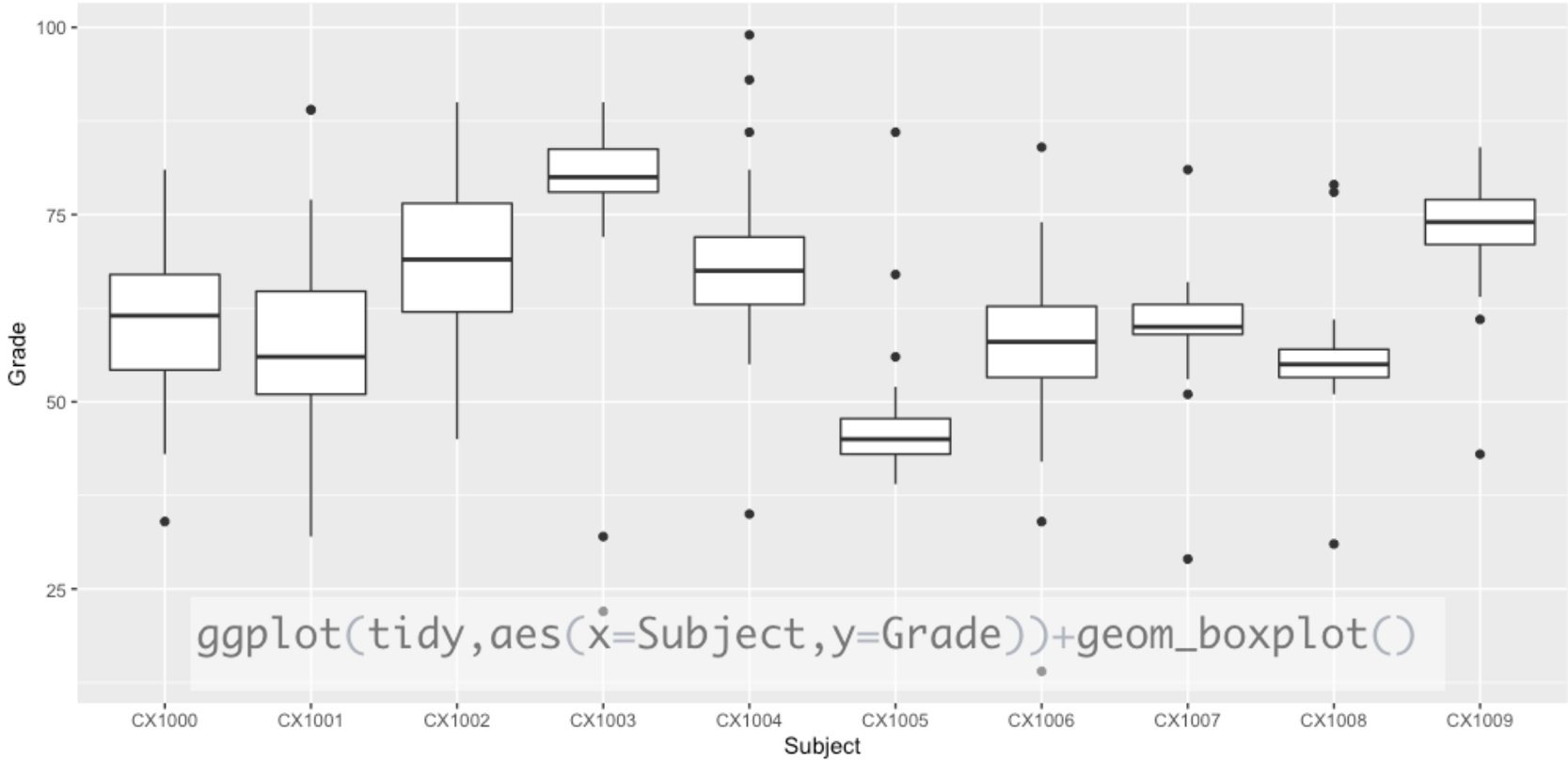
pivot_longer() Function call

```
> ex
# A tibble: 51 x 11
   StudentID CX1000 CX1001 CX1002 CX1003 CX1004 CX1005 CX1006 CX1007 CX1008 CX1009
       <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>
 1   1111111     56     51     78     85     63     45     55     59     52     76
 2   1111112     56     64     68     80     70     39     46     60     55     74
 3   1111113     52     61     63     81     71     49     54     61     54     76
 4   1111114     50     42     72     81     63     44     62     59     56     68
 5   1111115     67     53     77     84     65     52     63     62     52     71
 6   1111116     45     57     62     32     61     56     62     51     55     79
 7   1111117     67     58     54     77     75     44     58     62     57     77
 8   1111118     69     50     66     78     72     39     60     58     57     84
 9   1111119     70     56     62     80     71     52     60     63     54     70
10   1111120     51     52     46     82     74     42     66     63     55     73
# … with 41 more rows
```

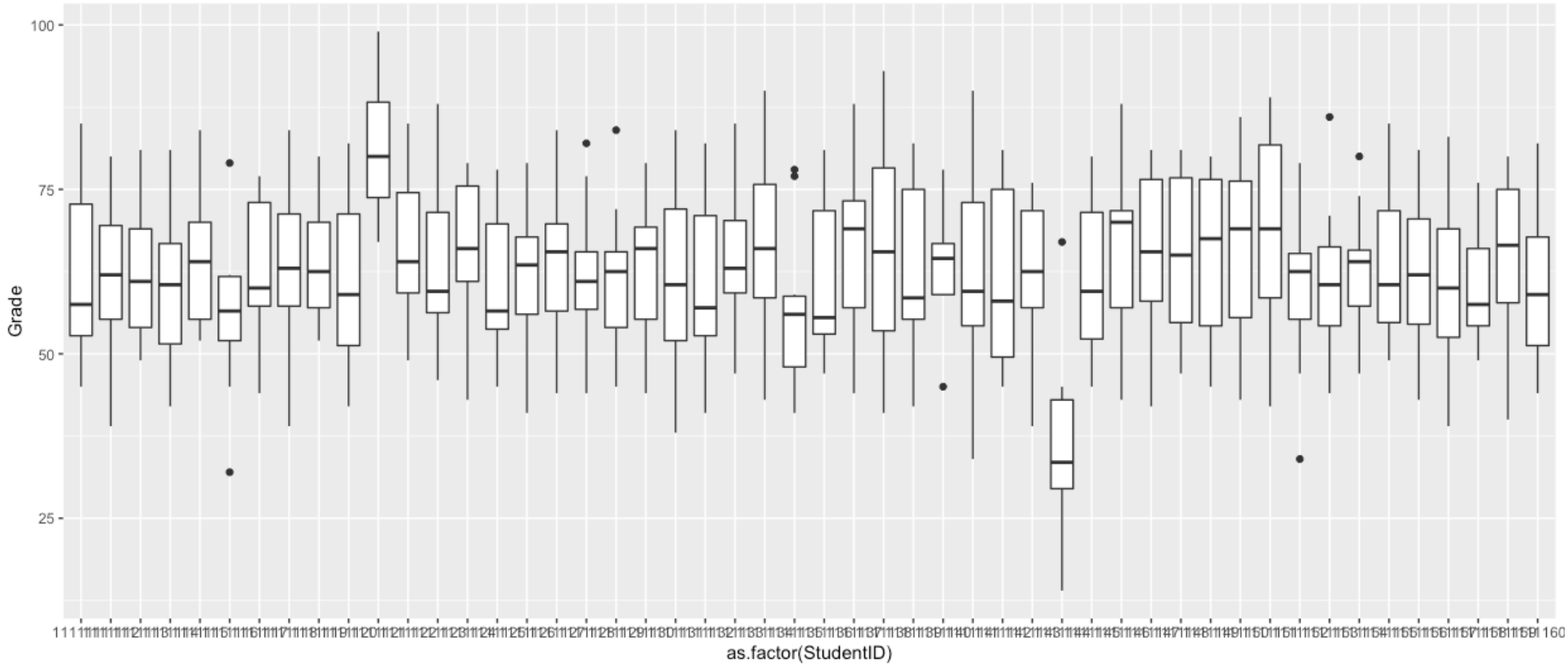
```
library(readr)

ex <- read_csv("code/AY 20_21/06 dplyr 01/ExamDataBroad.csv")

tidy <- pivot_longer(ex,
                     -StudentID,
                     names_to="Subject",
                     values_to="Grade")
```

```
# A tibble: 510 x 3
   StudentID Subject Grade
       <dbl> <chr>   <dbl>
 1   1111111 CX1000     56
 2   1111111 CX1001     51
 3   1111111 CX1002     78
 4   1111111 CX1003     85
 5   1111111 CX1004     63
 6   1111111 CX1005     45
 7   1111111 CX1006     55
 8   1111111 CX1007     59
 9   1111111 CX1008     52
10   1111111 CX1009     76
# … with 500 more rows
```

Tidy data Supports data exploration

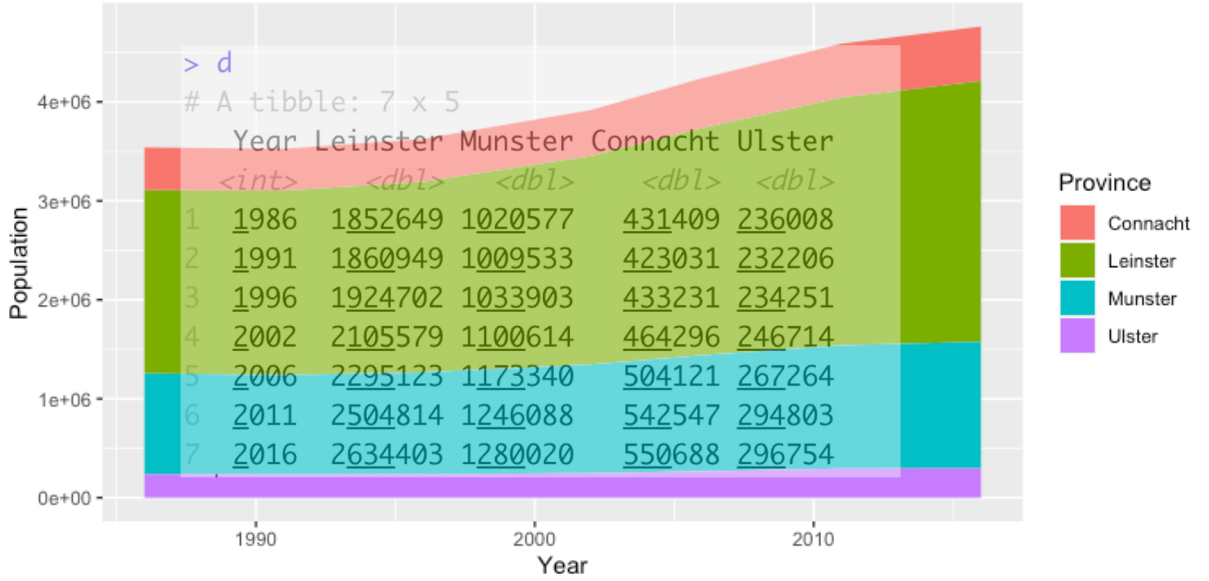


```
ggplot(tidy,aes(x=as.factor(StudentID),y=Grade))+geom_boxplot()
```



Challenge 7.3

- Transform the census data to tidy format and create the following plot



pivot_wider()

`pivot_wider()` "widens" data, increasing the number of columns and decreasing the number of rows. The inverse transformation is `pivot_longer()`.

Learn more in `vignette("pivot")`.

https://tidyr.tidyverse.org/reference/pivot_wider.html

```
pivot_wider(
  data,
  id_cols = NULL,
  names_from = name,
  names_prefix = "",
  names_sep = "_",
  names_glue = NULL,
  names_sort = FALSE,
  names_repair = "check_unique",
  values_from = value,
  values_fill = NULL,
  values_fn = NULL,
  ...
)
```

```
wide <- tidy %>%
        pivot_wider(names_from = Subject,
                    values_from = Grade)
```

```
# A tibble: 510 x 3
   StudentID Subject Grade
       <dbl> <chr>   <dbl>
 1   1111111 CX1000     56
 2   1111111 CX1001     51
 3   1111111 CX1002     78
 4   1111111 CX1003     85
 5   1111111 CX1004     63
 6   1111111 CX1005     45
 7   1111111 CX1006     55
 8   1111111 CX1007     59
 9   1111111 CX1008     52
10   1111111 CX1009     76
# … with 500 more rows
```

```
> wide
# A tibble: 50 x 11
   StudentID CX1000 CX1001 CX1002 CX1003 CX1004 CX1005 CX1006 CX1007 CX1008 CX1009
       <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>
 1   1111111     56     51     78     85     63     45     55     59     52     76
 2   1111112     56     64     68     80     70     39     46     60     55     74
 3   1111113     52     61     63     81     71     49     54     61     54     76
 4   1111114     50     42     72     81     63     44     62     59     56     68
 5   1111115     67     53     77     84     65     52     63     62     52     71
 6   1111116     45     57     62     32     61     56     62     51     55     79
 7   1111117     67     58     54     77     75     44     58     62     57     77
 8   1111118     69     50     66     78     72     39     60     58     57     84
 9   1111119     70     56     62     80     71     52     60     63     54     70
10   1111120     51     52     46     82     74     42     66     63     55     73
# … with 40 more rows
```

separate()

- Separate pulls apart one column into multiple columns
- It splits the information based on finding a non-alphanumeric character
- Separator can be defined (sep="/")
- A converter can find best type for the result, if needed.

```
> table3
# A tibble: 6 x 3
      country  year              rate
*       <chr> <int>             <chr>
1 Afghanistan  1999      745/19987071
2 Afghanistan  2000     2666/20595360
3      Brazil  1999   37737/172006362
4      Brazil  2000   80488/174504898
5       China  1999 212258/1272915272
6       China  2000 213766/1280428583
```

```
Function:       separate(data, col, into, sep = " ", remove = TRUE, convert = FALSE)
Same as:        data %>% separate(col, into, sep = " ", remove = TRUE, convert = FALSE)

Arguments:
        data:           data frame
        col:            column name representing current variable
        into:           names of variables representing new variables
        sep:            how to separate current variable (char, num, or symbol)
        remove:         if TRUE, remove input column from output data frame
        convert:        if TRUE will automatically convert values to logical, integer, numeric, complex or
                        factor as appropriate
```

```
> table3 %>%
+   separate(rate,into=c("cases","population"),
+            convert=TRUE)
# A tibble: 6 x 4
      country  year  cases population
*       <chr> <int>  <int>      <int>
1 Afghanistan  1999    745   19987071
2 Afghanistan  2000   2666   20595360
3      Brazil  1999  37737  172006362
4      Brazil  2000  80488  174504898
5       China  1999 212258 1272915272
6       China  2000 213766 1280428583
```

Lecture Summary

- Relational data in dplyr
- Mutating joins
- Filtering joins
- tidyr overview
  - pivot_longer()
  - pivot_wider()
  - separate()