Introduction to R

Alexander McLain

Contents

1 Loading data in

This example will use the lead data discussed in the notes. First, we'll read the work on reading the data into R. I have the data in two formats as a .csv file and as a .xlsx file. The .csv extention is the most natural way to load data into R using the **read.csv** function. The main components of this function are

- **Header**: when importing a file into R, users need to tell the software whether a header is present so that it knows whether to treat the 1st line as variable names or observed data values.
- **Missing value**: this is used to denote a missing value. When reading the file, R will turn these entries into the form it recognizes: NA.

For example, we can load the data in as follows:

```
# First I set the working directory so that I don't have to include
# the full file path every time I want to load in some data.  The
# working directory is the filepath of where all of your data are.
# Note that it used / not \.
setwd("~/OneDrive - University of South Carolina/Teaching/755_Spring_2021/Examples R")
wide_lead <- read.csv("wide_lead.csv",header = TRUE, na.strings = "",
                      stringsAsFactors = FALSE)
is.data.frame(wide_lead)
```

```
## [1] TRUE
```

```
str(wide_lead)
```

```
## 'data.frame':    100 obs. of  6 variables:
##  $ ID : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ TRT: chr  "P" "A" "A" "P" ...
##  $ PB1: num  30.8 26.5 25.8 24.7 20.4 20.4 28.6 33.7 19.7 31.1 ...
##  $ PB2: num  26.9 14.8 23 24.5 2.8 5.4 20.8 31.6 14.9 31.2 ...
##  $ PB3: num  25.8 19.5 19.1 22 3.2 4.5 19.2 28.5 15.3 29.2 ...
##  $ PB4: num  23.8 21 23.2 22.5 9.4 11.9 18.4 25.1 14.7 30.1 ...
```

I always use **stringAsFactors = FALSE** and I suggest you do the same. Factors are important things that we'll use later on (they are similar to the class statement in SAS).

Now let's look at the first few rows of data using the head function.

```
head(wide_lead)
```

ID	TRT	PB1	PB2	PB3	PB4
1	P	30.8	26.9	25.8	23.8
2	A	26.5	14.8	19.5	21.0
3	A	25.8	23.0	19.1	23.2
4	P	24.7	24.5	22.0	22.5
5	A	20.4	2.8	3.2	9.4
6	A	20.4	5.4	4.5	11.9

Now, we'll load in the .xlsx version of this data set. To do this we need to load in a package.

For this portion we'll be using the **openxlsx** package, which we'll first need to install and load into our workspace. **You only need to install a package once, but you need to lead it (using library) every time.**

```
install.packages("openxlsx")
```

```
library(openxlsx)
wide_lead <- read.xlsx("wide_lead.xlsx", colNames = TRUE,na.strings = "")
is.data.frame(wide_lead)
```

```
## [1] TRUE
```

```
str(wide_lead)
```

```
## 'data.frame':    100 obs. of  6 variables:
##  $ ID : num  1 2 3 4 5 6 7 8 9 10 ...
##  $ TRT: chr  "P" "A" "A" "P" ...
##  $ PB1: num  30.8 26.5 25.8 24.7 20.4 20.4 28.6 33.7 19.7 31.1 ...
##  $ PB2: num  26.9 14.8 23 24.5 2.8 5.4 20.8 31.6 14.9 31.2 ...
##  $ PB3: num  25.8 19.5 19.1 22 3.2 4.5 19.2 28.5 15.3 29.2 ...
##  $ PB4: num  23.8 21 23.2 22.5 9.4 11.9 18.4 25.1 14.7 30.1 ...
```

```
head(wide_lead)
```

ID	TRT	PB1	PB2	PB3	PB4
1	P	30.8	26.9	25.8	23.8
2	A	26.5	14.8	19.5	21.0
3	A	25.8	23.0	19.1	23.2
4	P	24.7	24.5	22.0	22.5
5	A	20.4	2.8	3.2	9.4
6	A	20.4	5.4	4.5	11.9

Here, **colNames = TRUE** is analogous to **header = TRUE** is the **read.csv** function.

2 Transforming data from wide to long (and vice versa)

Now we have our data in, but the format is in the **wide** version and not the **long** version. Sometimes we'll want to use the wide version, but mostly we'll want to use the long version of the dataset.

To transform the data from wide to long we'll use the **pivot_longer** function in the **tidyr** package (to complete this step you'll need to install **tidyverse**).

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.0 --

## v ggplot2 3.3.3     v purrr   0.3.4
## v tibble  3.0.4     v dplyr   1.0.2
## v tidyr   1.1.2     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.0

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

The **pivot_longer** function has the following arguements:

```
## Pivot data from wide to long
##
## 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'.
##
## names_prefix: A regular expression used to remove matching text from
##           the start of each variable name.
##
## names_sep, names_pattern: If 'names_to' contains multiple values, these
##           arguments control how the column name is broken up.
##
##           'names_sep' takes the same specification as 'separate()', and
##           can either be a numeric vector (specifying positions to break
##           on), or a single string (specifying a regular expression to
##           split on).
##
##           'names_pattern' takes the same specification as 'extract()',
##           a regular expression containing matching groups (()).
```

```
##
##           If these arguments do not give you enough control, use
##           'pivot_longer_spec()' to create a spec object and process
##           manually as needed.
##
## names_ptypes, values_ptypes: A list of column name-prototype pairs. A
##           prototype (or ptype for short) is a zero-length vector (like
##           'integer()' or 'numeric()') that defines the type, class, and
##           attributes of a vector. Use these arguments to confirm that
##           the created columns are the types that you expect.
##
##           If not specified, the type of the columns generated from
##           'names_to' will be character, and the type of the variables
##           generated from 'values_to' will be the common type of the
##           input columns used to generate them.
##
## names_transform, values_transform: A list of column name-function
##           pairs. Use these arguments if you need to change the type of
##           specific columns. For example, 'names_transform = list(week =
##           as.integer)' would convert a character week variable to an
##           integer.
##
## names_repair: What happens if the output has invalid column names? The
##           default, '"check_unique"' is to error if the columns are
##           duplicated. Use '"minimal"' to allow duplicates in the
##           output, or '"unique"' to de-duplicated by adding numeric
##           suffixes. See 'vctrs::vec_as_names()' for more options.
##
## 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.
##
## values_drop_na: If 'TRUE', will drop rows that contain only 'NA's in
##           the 'value_to' column. This effectively converts explicit
##           missing values to implicit missing values, and should
##           generally be used only when missing values in 'data' were
##           created by its structure.
##
##      ...: Additional arguments passed on to methods.
```

2.1 Example of going from wide to long

```
long_lead <- pivot_longer(wide_lead, cols = starts_with("PB"), names_to = "time",
                          names_prefix = "PB", values_to = "PB",
                          values_drop_na = TRUE)
head(long_lead,20)
```

ID	TRT	time	PB
1	P	1	30.8
1	P	2	26.9
1	P	3	25.8

ID	TRT	time	PB
1	P	4	23.8
2	A	1	26.5
2	A	2	14.8
2	A	3	19.5
2	A	4	21.0
3	A	1	25.8
3	A	2	23.0
3	A	3	19.1
3	A	4	23.2
4	P	1	24.7
4	P	2	24.5
4	P	3	22.0
4	P	4	22.5
5	A	1	20.4
5	A	2	2.8
5	A	3	3.2
5	A	4	9.4

Now, let's go from long to wide. Here, we're going to use the **pivot__wider** function which has arguements.

```
## Pivot data from long to wide
##
## Arguments:
##
##     data: A data frame to pivot.
##
##  id_cols: <'tidy-select'> A set of columns that uniquely identifies
##           each observation. Defaults to all columns in 'data' except
##           for the columns specified in 'names_from' and 'values_from'.
##           Typically used when you have redundant variables, i.e.
##           variables whose values are perfectly correlated with existing
##           variables.
##
## names_from, values_from: <'tidy-select'> A pair of arguments describing
##           which column (or columns) to get the name of the output
##           column ('names_from'), and which column (or columns) to get
##           the cell values from ('values_from').
##
##           If 'values_from' contains multiple values, the value will be
##           added to the front of the output column.
##
## names_prefix: String added to the start of every variable name. This is
##           particularly useful if 'names_from' is a numeric vector and
##           you want to create syntactic variable names.
##
## names_sep: If 'names_from' or 'values_from' contains multiple
##           variables, this will be used to join their values together
##           into a single string to use as a column name.
##
## names_glue: Instead of 'names_sep' and 'names_prefix', you can supply a
##           glue specification that uses the 'names_from' columns (and
##           special '.value') to create custom column names.
```

```
##
## names_sort: Should the column names be sorted? If 'FALSE', the default,
##           column names are ordered by first appearance.
##
## names_repair: What happens if the output has invalid column names? The
##           default, '"check_unique"' is to error if the columns are
##           duplicated. Use '"minimal"' to allow duplicates in the
##           output, or '"unique"' to de-duplicated by adding numeric
##           suffixes. See 'vctrs::vec_as_names()' for more options.
##
## values_fill: Optionally, a (scalar) value that specifies what each
##           'value' should be filled in with when missing.
##
##           This can be a named list if you want to apply different
##           aggregations to different value columns.
##
## values_fn: Optionally, a function applied to the 'value' in each cell
##           in the output. You will typically use this when the
##           combination of 'id_cols' and 'value' column does not uniquely
##           identify an observation.
##
##           This can be a named list if you want to apply different
##           aggregations to different value columns.
##
##      ...: Additional arguments passed on to methods.
```

2.2 Example of going from long to wide

```
wide_lead2 <- pivot_wider(long_lead, id_cols = c(ID,TRT), names_from = time,
                          values_from = PB)
head(wide_lead2)
```

ID	TRT	1	2	3	4
1	P	30.8	26.9	25.8	23.8
2	A	26.5	14.8	19.5	21.0
3	A	25.8	23.0	19.1	23.2
4	P	24.7	24.5	22.0	22.5
5	A	20.4	2.8	3.2	9.4
6	A	20.4	5.4	4.5	11.9

```
wide_lead2 <- pivot_wider(long_lead, id_cols = c(ID,TRT), names_from = time,
                          values_from = PB, names_prefix = "PB")
head(wide_lead2)
```

ID	TRT	PB1	PB2	PB3	PB4
1	P	30.8	26.9	25.8	23.8
2	A	26.5	14.8	19.5	21.0
3	A	25.8	23.0	19.1	23.2
4	P	24.7	24.5	22.0	22.5
5	A	20.4	2.8	3.2	9.4
6	A	20.4	5.4	4.5	11.9

```r
wide_lead2 <- long_lead %>% pivot_wider(id_cols = c(ID,TRT), names_from = time,
                                        values_from = PB, names_prefix = "PB")
head(wide_lead2)
```

ID	TRT	PB1	PB2	PB3	PB4
1	P	30.8	26.9	25.8	23.8
2	A	26.5	14.8	19.5	21.0
3	A	25.8	23.0	19.1	23.2
4	P	24.7	24.5	22.0	22.5
5	A	20.4	2.8	3.2	9.4
6	A	20.4	5.4	4.5	11.9

3 Plotting longitudinal data in R

To do our plotting we'll use the **ggplot2** package. If you installed and loaded the *tidyverse* package above you've already installed and loaded **ggplot2**. To get a full introduction into plotting longitudinal data click here, which contains much more explanation than this document.

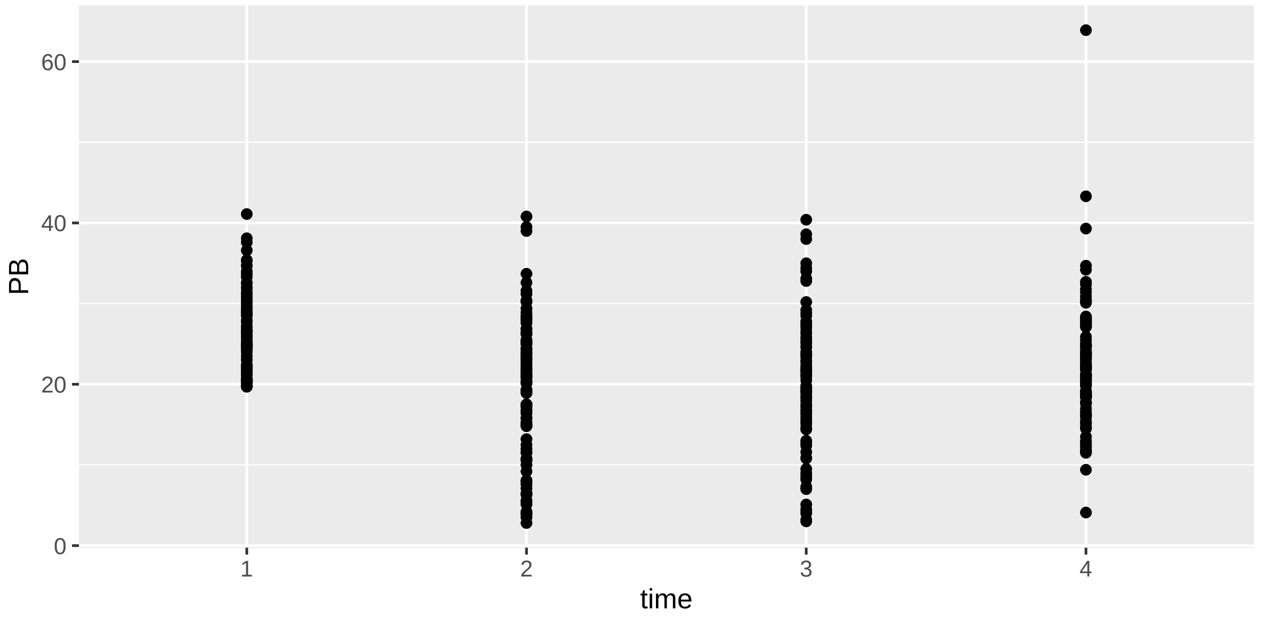
To start were going to "set the table" by creating a plot object. This object will not actually create a plot, it just sets the table for what were going to do later.

```r
p <- ggplot(data = long_lead, aes(x = time, y = PB, group = ID))
```

3.1 Plotting all data

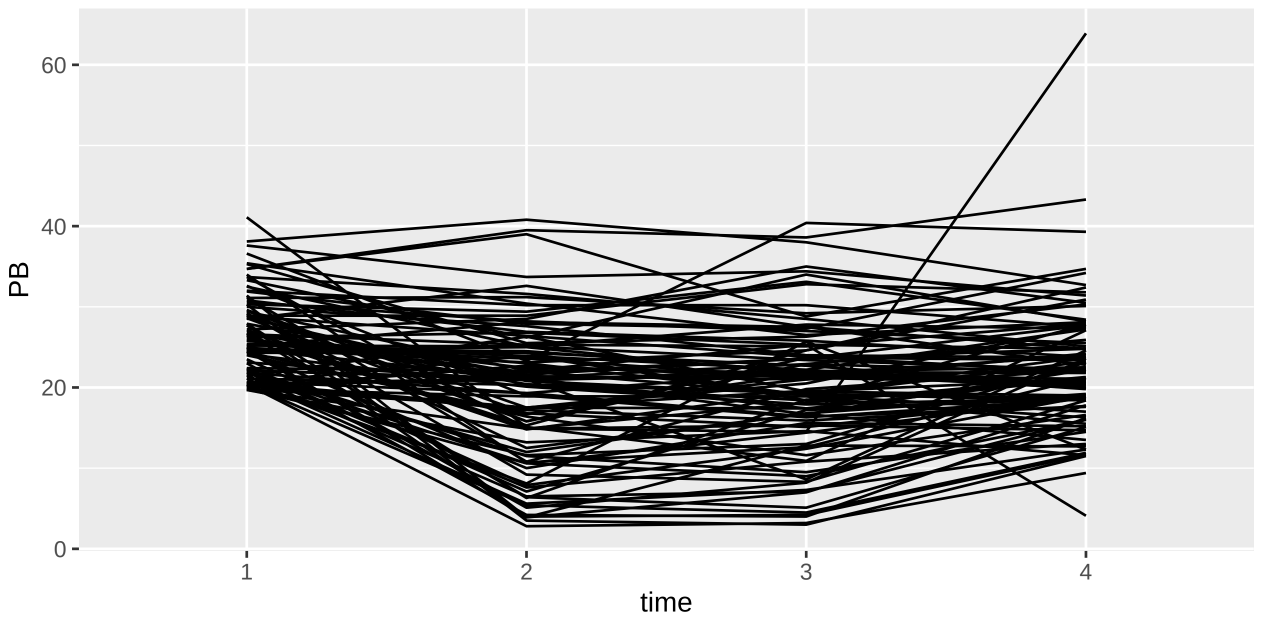
3.1.1 Just plotting points (a.k.a., scatterplot)

```r
p + geom_point()
```



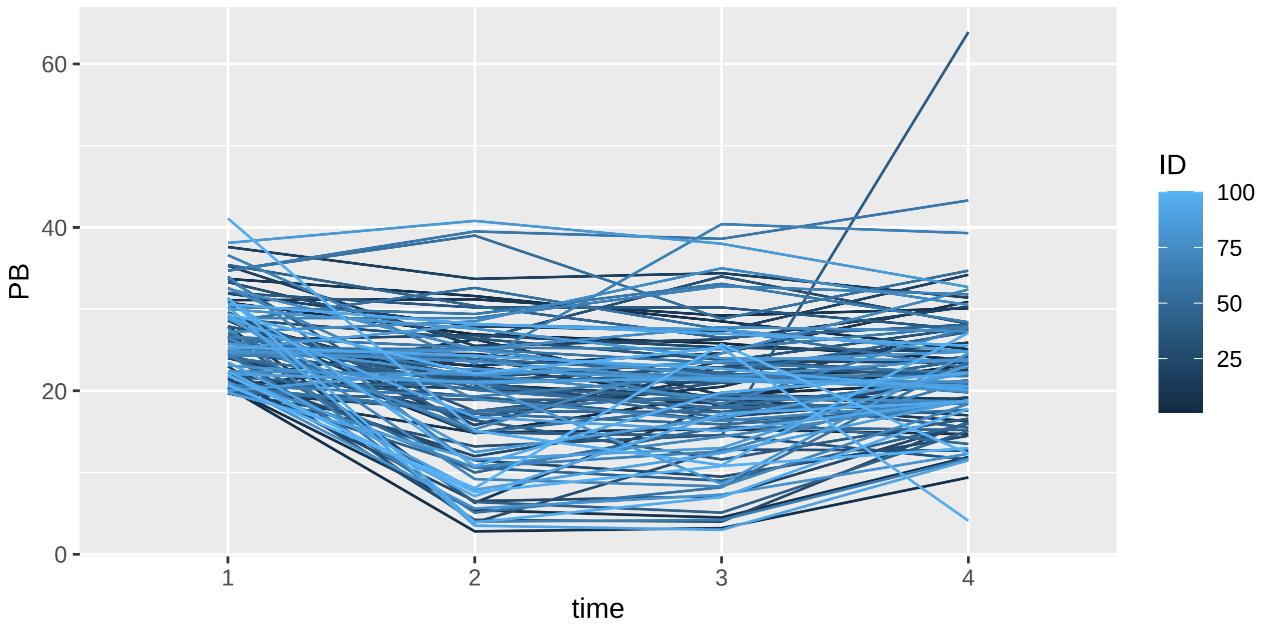
3.1.2 Simple spaghetti plot

```r
p + geom_line()
```

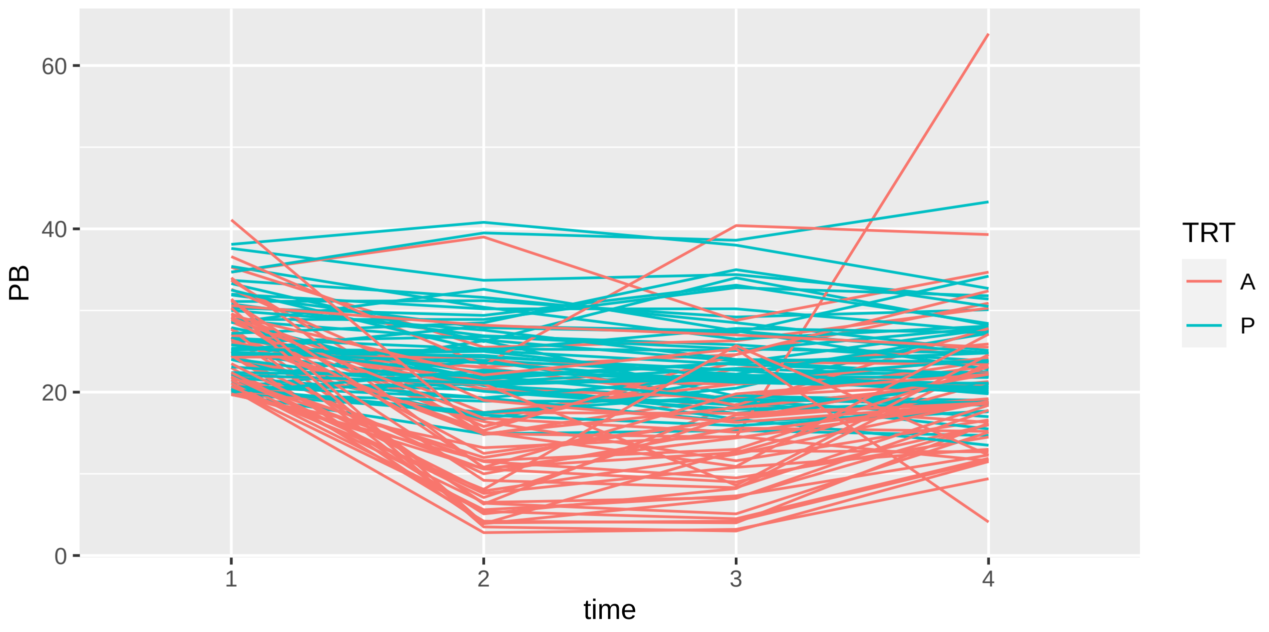


3.1.3 A colorful spaghetti plot

```r
p + geom_line(aes(colour = ID))
```

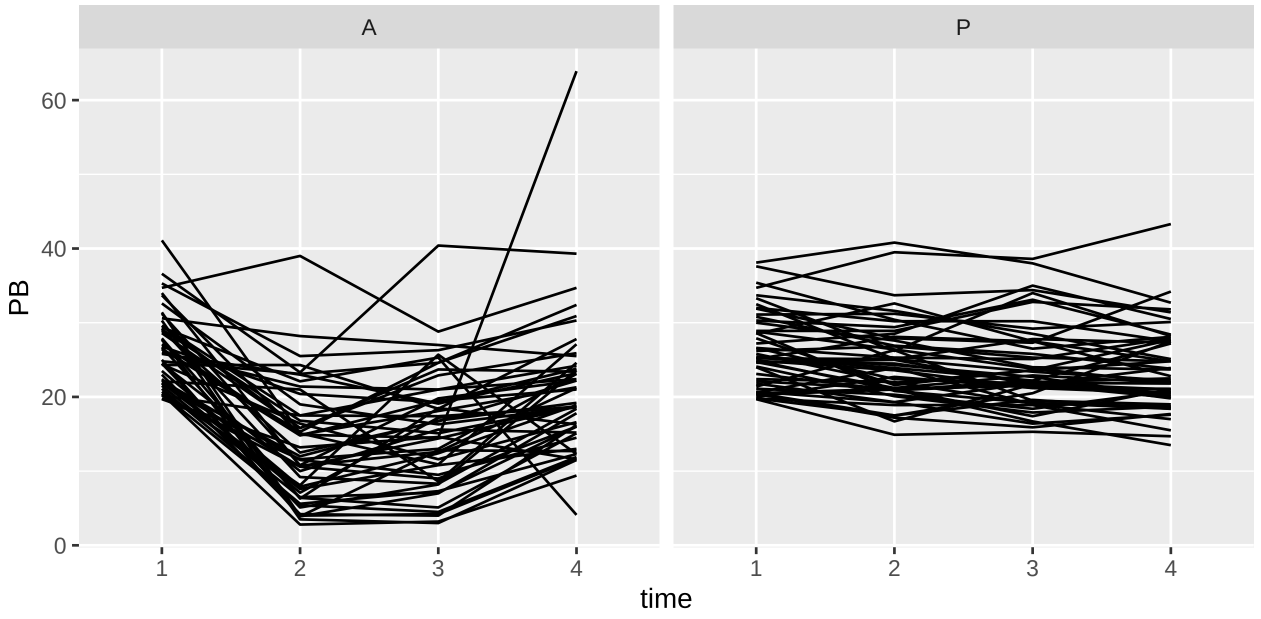


```r
p + geom_line(aes(colour = TRT))
```



3.1.4 Facet (condition) the graph base on the treatment (TRT) variable

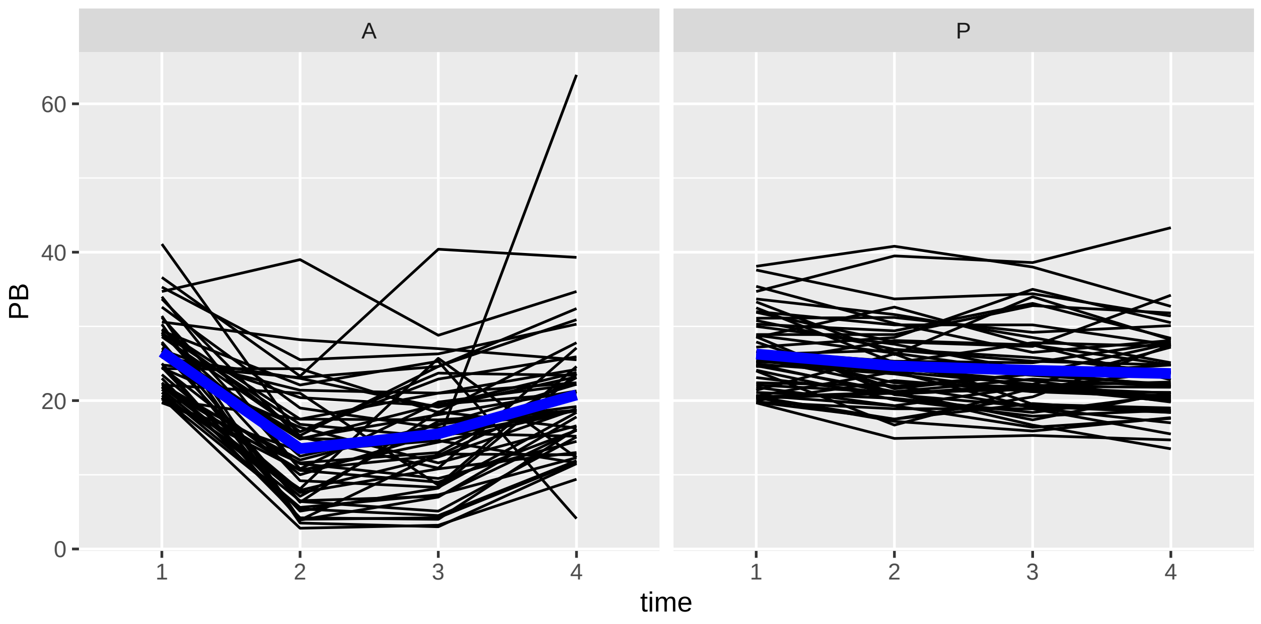
```
p + geom_line() + facet_grid(. ~ TRT)
```



3.2 Adding summary statistics to plots

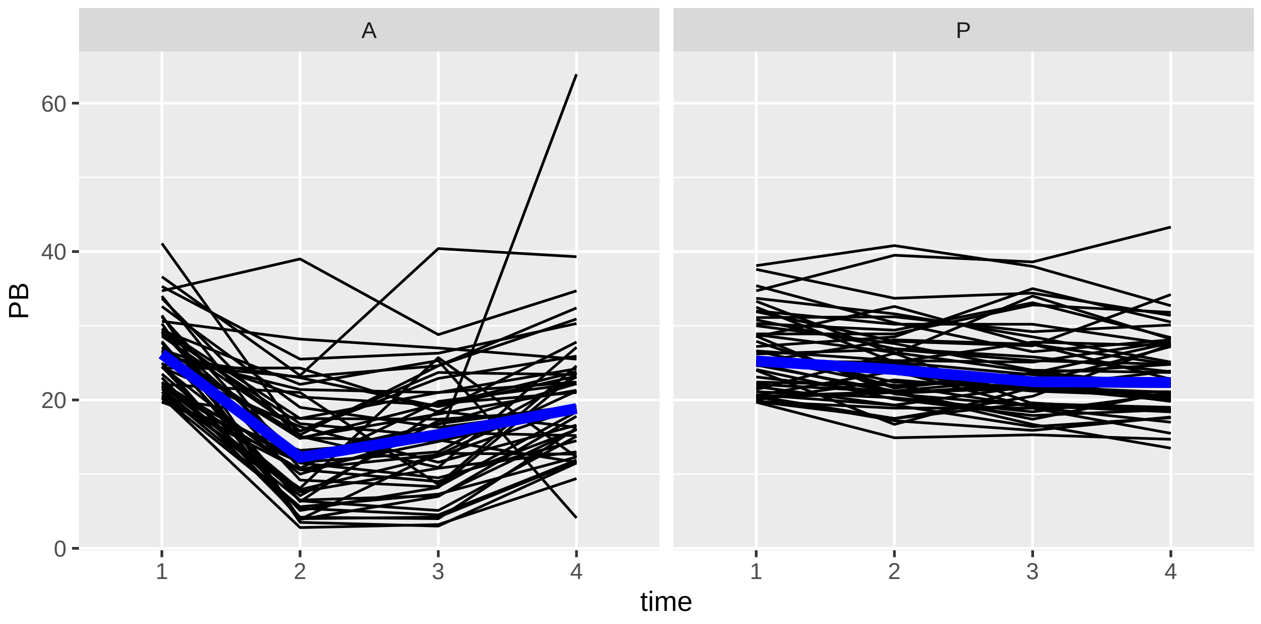
3.2.1 Adding a mean line to each plot

```
p + geom_line() +
  stat_summary(aes(group = 1), geom = "line", fun = mean,
               color = "blue", size = 2) +
  facet_grid(. ~ TRT)
```



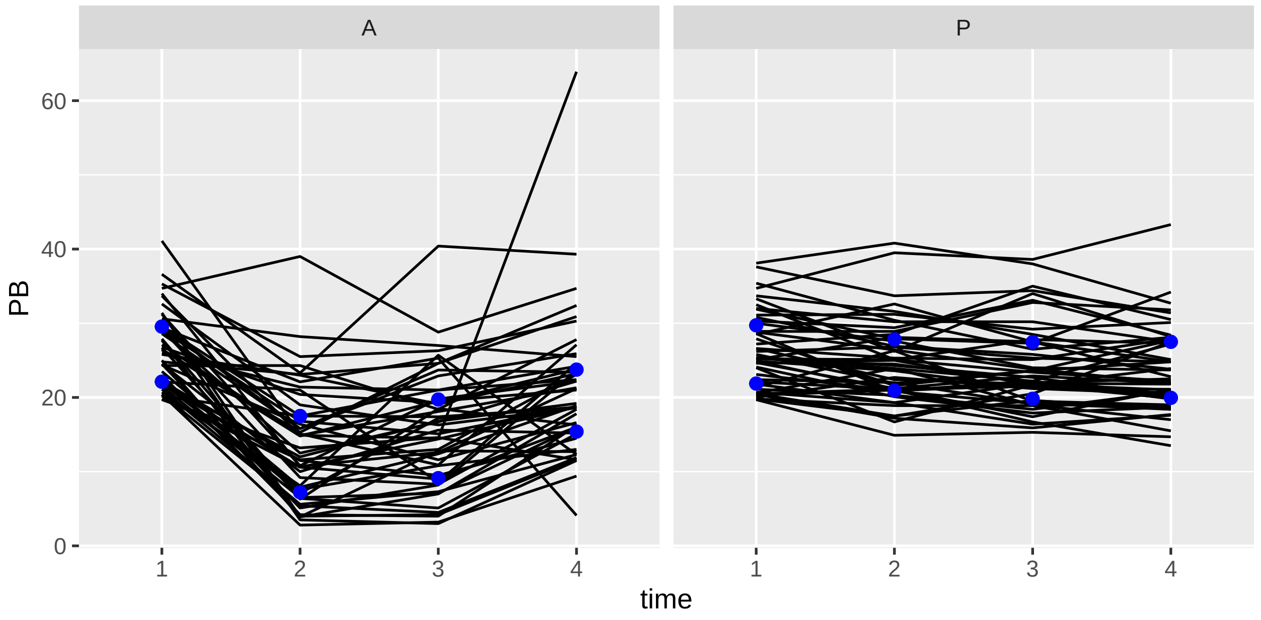
3.2.2 Adding a median line to each plot

```
p + geom_line()  +
  stat_summary(aes(group = 1), geom = "line", fun = median,
               color = "blue", size = 2) +
  facet_grid(. ~ TRT)
```



3.2.3 Adding the first and third quartiles to each plot (note: line doesn't work well here)

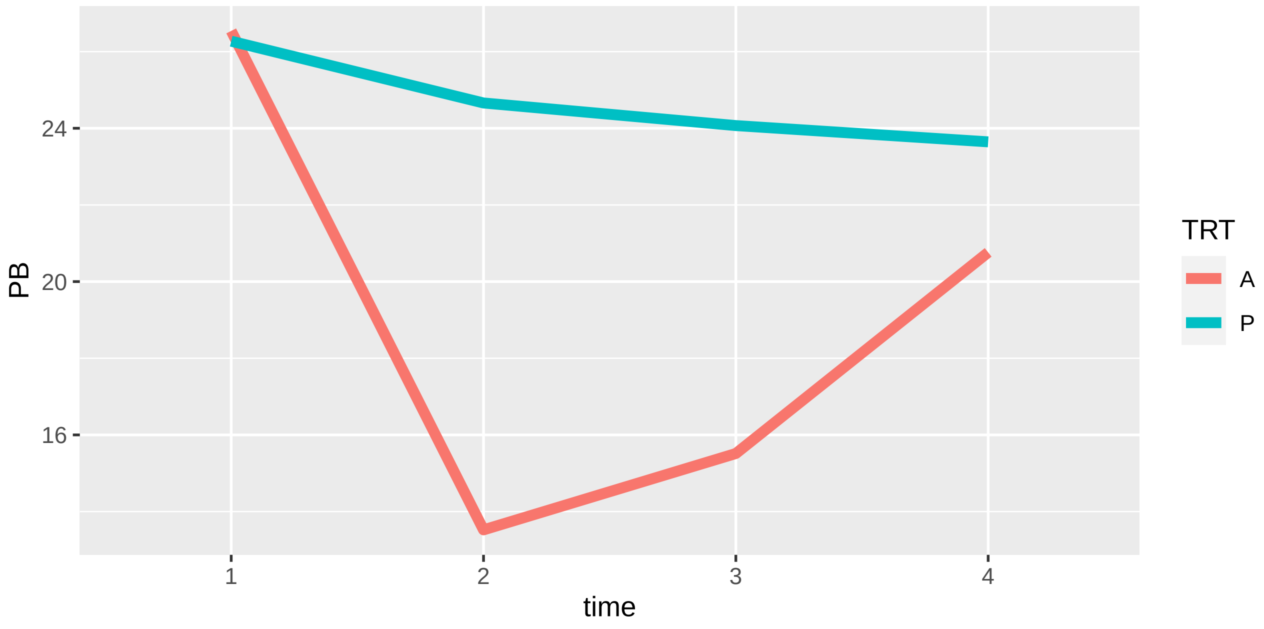
```
p + geom_line()  +
  stat_summary(aes(group = 1), geom = "point", fun = quantile,
               fun.args=(list(probs = c(0.25, 0.75))),
               color = "blue", size = 2) +
  facet_grid(. ~ TRT)
```



3.3 Plotting summary statistics by group

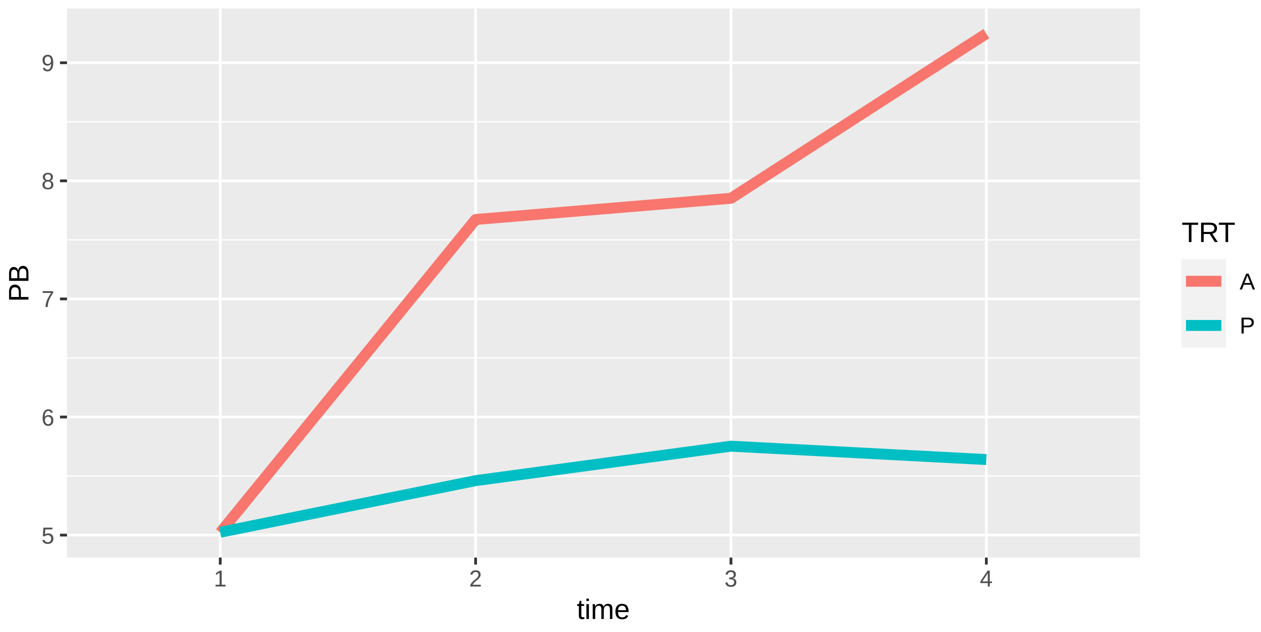
3.3.1 Plotting the means for each group

```
p + stat_summary(aes(group = TRT, color = TRT), geom = "line",
                 fun = mean, size = 2)
```



3.3.2 Plotting the standard deviation for each group

```
p + stat_summary(aes(group = TRT, color = TRT), geom = "line",
                 fun = sd, size = 2)
```



4 Example Two

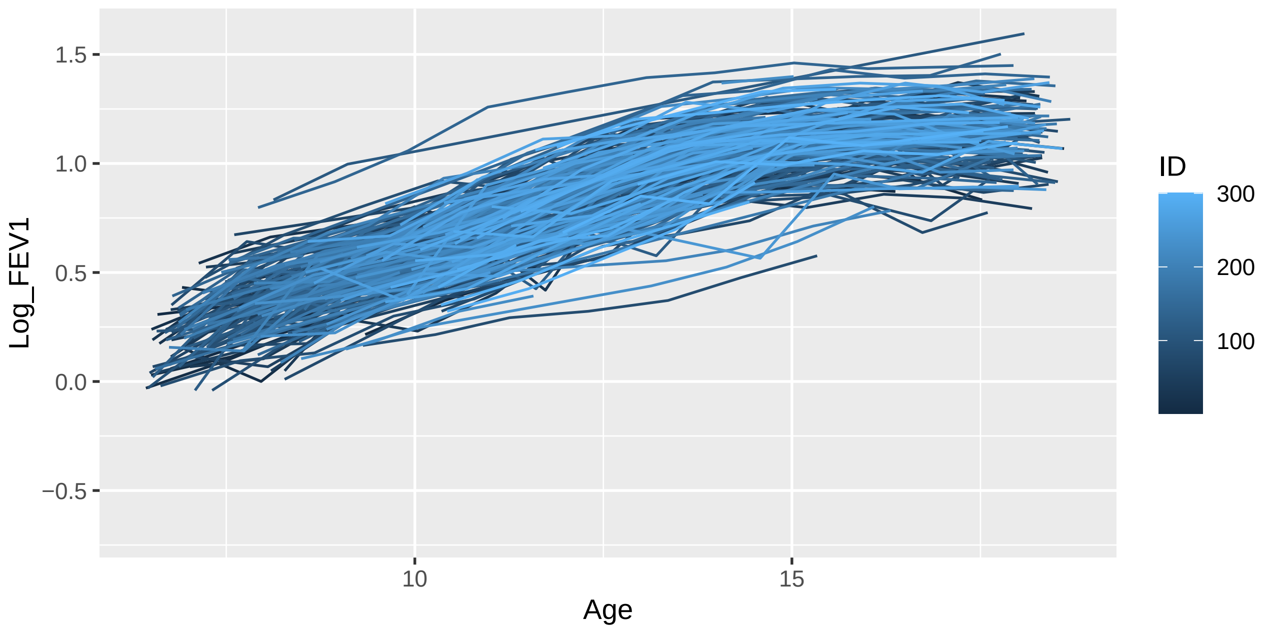
The Six Cities Study of Air Pollution and Health was a longitudinal study designed to characterize lung growth as measured by changes in pulmonary function in children and adolescents, and the factors that influence lung function growth. A cohort of 13,379 children born on or after 1967 was enrolled in six communities across the U.S.: Watertown (Massachusetts), Kingston and Harriman (Tennessee), a section of St. Louis (Missouri), Steubenville (Ohio), Portage (Wisconsin), and Topeka (Kansas). Most children were enrolled in the first or second grade (between the ages of six and seven) and measurements of study participants were obtained annually until graduation from high school or loss to follow-up. At each annual examination, spirometry, the measurement of pulmonary function, was performed and a respiratory health questionnaire was completed by a parent or guardian.

```r
Six_cities <- read.csv("Six_cities.csv", header = TRUE)
head(Six_cities,8)
```

ID	Height	Age	INI_Height	INI_Age	Log_FEV1
1	1.20	9.3415	1.20	9.3415	0.21511
1	1.28	10.3929	1.20	9.3415	0.37156
1	1.33	11.4524	1.20	9.3415	0.48858
1	1.42	12.4600	1.20	9.3415	0.75142
1	1.48	13.4182	1.20	9.3415	0.83291
1	1.50	15.4743	1.20	9.3415	0.89200
1	1.52	16.3723	1.20	9.3415	0.87129
2	1.13	6.5873	1.13	6.5873	0.30748

First, we'll "set the the table" and do a colorful spaghetti plot:

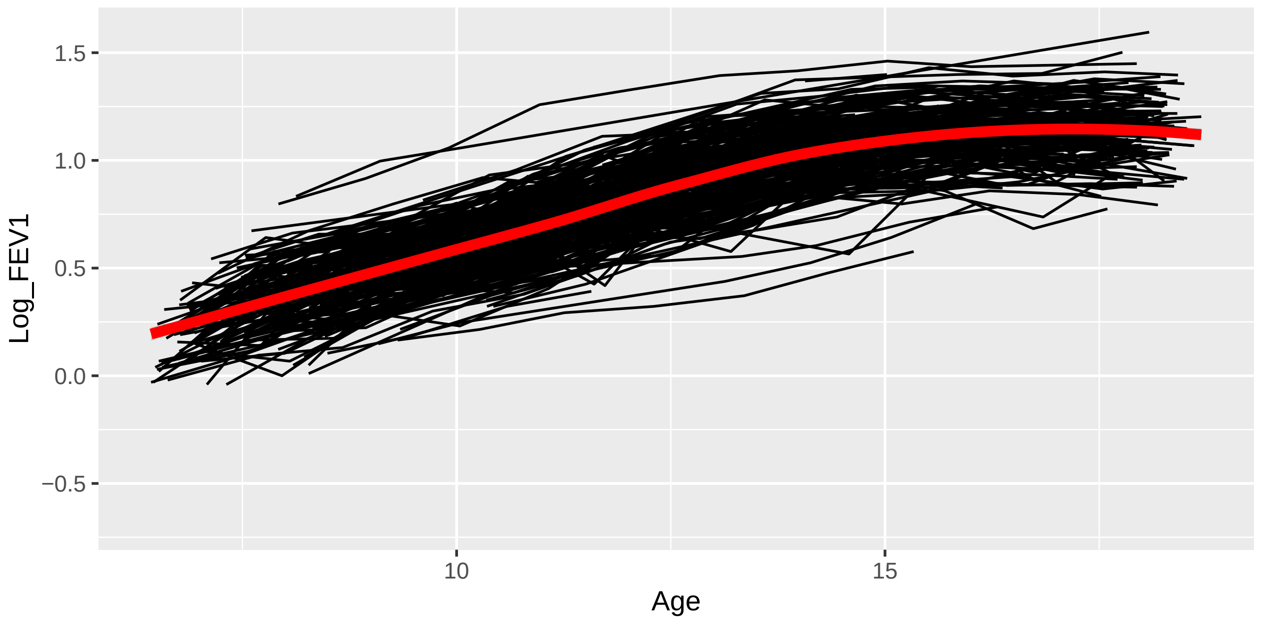
```r
p <- ggplot(Six_cities, aes(x = Age, y = Log_FEV1, group = ID))
p + geom_line(aes(color = ID))
```



Now we're going to add a smooth line and remove the color = ID portion.

```r
# Here, we'll use a "loess" smoother.  There are many other options
p + geom_line() + geom_smooth(aes(group = 1), method = "loess",
                              color = "red", size = 2)
```

```
## `geom_smooth()` using formula 'y ~ x'
```

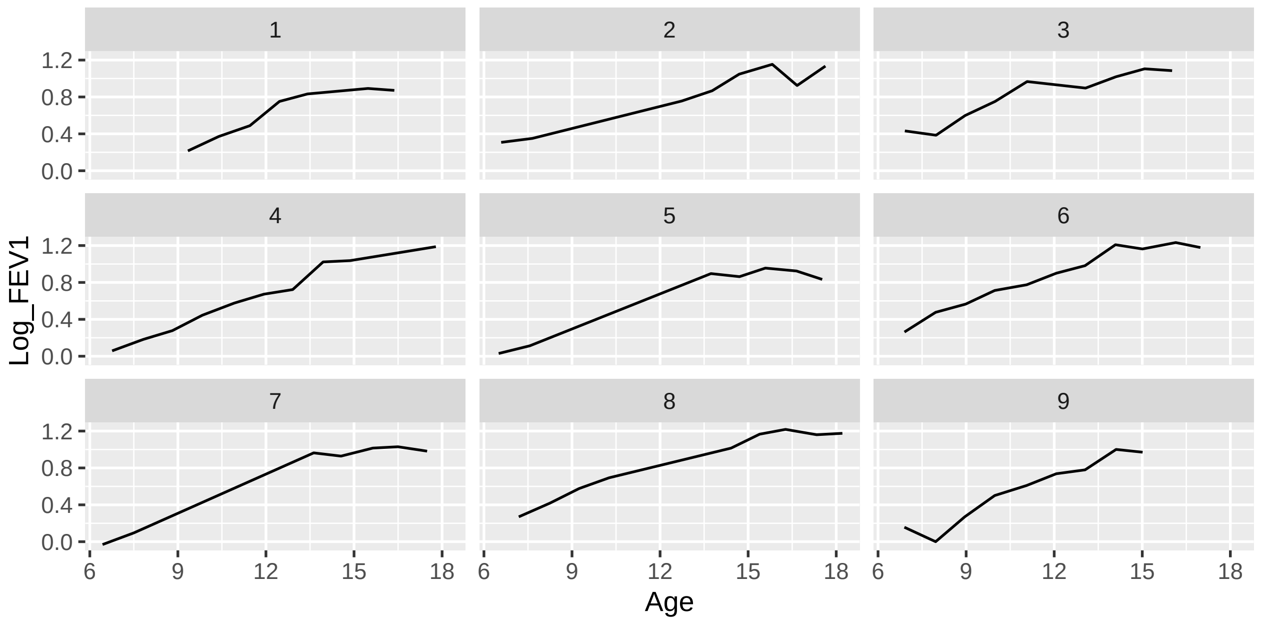


4.1 Plotting individual level data

Note that if we don't use the **aes(group = 1)** portion it will give a smoothed line for every subject. Let's try this out for the first 9 subjects.

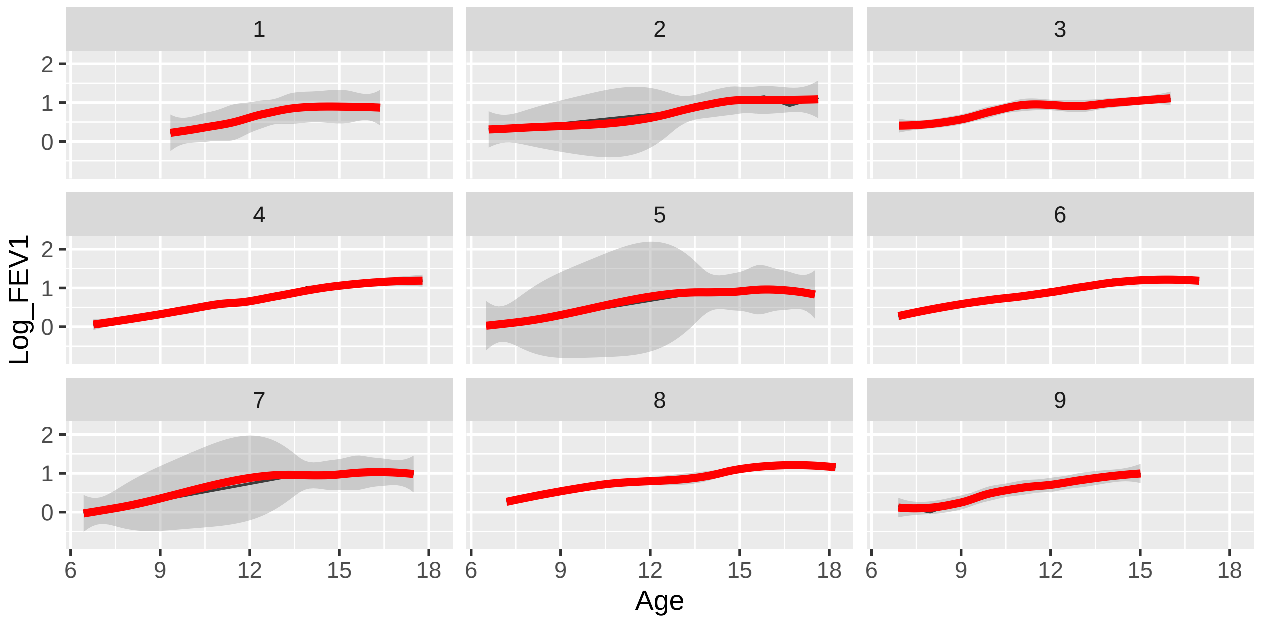
```r
Six_cities_sm <- Six_cities %>% filter(ID < 10)
p <- ggplot(Six_cities_sm, aes(x = Age, y = Log_FEV1, group = ID))
```

```
p + geom_line() + facet_wrap(. ~ ID)
```



```
p + geom_line() +
  geom_smooth(aes(group = 1), method = "loess", color = "red", size = 1.5) +
  facet_wrap(. ~ ID)
```

```
## `geom_smooth()` using formula 'y ~ x'
```


Log_FEV1
Age