…The dataset contains six columns:

- **USUBJID**: Subject unique identifier
- **TRT01PN**: Treatment group (numeric)
- **TRT01P**: Treatment group
- **AVISITN**: Visit identifier (numeric)
- **AVISIT**: Visit identifier
- **AVAL**: haemoglobin concentration (g/dL)

```
library(dplyr)
data_source <- "https://raw.githubusercontent.com/VIS-SIG/Wonderful-Wednesdays/master/data/2020/2020-06-10/hgb_data.csv"
trial_data <- read.csv(data_source, stringsAsFactors = FALSE)
glimpse(trial_data)
```

```
Rows: 2,100
Columns: 6
$ USUBJID  "ABC123456.000001", "ABC123456.000001", "ABC123456.000001",...
$ TRT01PN  1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,...
$ TRT01P   "Treatment E", "Treatment E", "Treatment E", "Treatment E",...
$ AVISITN  10, 20, 30, 40, 50, 60, 70, 10, 20, 30, 40, 50, 60, 70, 10,...
$ AVISIT   "Baseline", "Week 4", "Week 8", "Week 12", "Week 16", "Week...
$ AVAL     10.197338, 11.272717, 12.288091, 13.508947, 12.195863, 10.1...
```

From the quick `glimpse()` you can see that each subject has multiple visits: Baseline, Week 4, Week 8 and so forth. To calculate change from baseline I have to subtract the baseline value from the value at each visit. Here's how.

```
trial_data_chg <- trial_data %>%
  arrange(USUBJID, AVISITN) %>%
  group_by(USUBJID) %>%
  mutate(CHG = AVAL - AVAL[1L]) %>%
  ungroup()
```

First, it's important to arrange the data of each subject in increasing order of `AVISITN`. Next, I grouped the data by `USUBJID` such that `AVAL[1L]` refers to the baseline value of each subject. Finally, I subtracted the baseline value from the value measured at each visit.

Let's have a look at the result to see whether this actually did what I said it would do.

```
trial_data_chg %>%
  select(USUBJID, AVISIT, AVAL, CHG) %>%
  print(n = 14)
```

```
# A tibble: 2,100 x 4
   USUBJID          AVISIT    AVAL     CHG

 1 ABC123456.000001 Baseline 10.2   0
 2 ABC123456.000001 Week 4   11.3   1.08
 3 ABC123456.000001 Week 8   12.3   2.09
 4 ABC123456.000001 Week 12  13.5   3.31
 5 ABC123456.000001 Week 16  12.2   2.00
 6 ABC123456.000001 Week 20  10.2  -0.0213
 7 ABC123456.000001 Week 24  12.3   2.09
 8 ABC123456.000002 Baseline  9.40  0
 9 ABC123456.000002 Week 4    9.14 -0.259
10 ABC123456.000002 Week 8    8.56 -0.838
11 ABC123456.000002 Week 12   8.59 -0.809
```

```
12 ABC123456.000002 Week 16   8.46 -0.942
13 ABC123456.000002 Week 20   8.78 -0.624
14 ABC123456.000002 Week 24   6.97 -2.43
# ... with 2,086 more rows
```

Looks good!

Alternatively, you could do this which circumvents the need to use `arrange()`.

```
trial_data_chg2 <- trial_data %>%
  group_by(USUBJID) %>%
  mutate(CHG = AVAL - AVAL[AVISIT == "Baseline"]) %>%
  ungroup()
```

Let's check that this in fact produced the same result.

```
diffdf::diffdf(trial_data_chg, trial_data_chg2)
```

```
No issues were found!
```

Awesome!

Next, I will show you how you can achieve the same result using only base `R`. I will use the good old split-apply-combine strategy.

```
by_subject <- split(trial_data, trial_data$USUBJID)
by_subject_chg <- lapply(by_subject, function(data) {
  data$CHG <- data$AVAL - data$AVAL[data$AVISIT == "Baseline"]
  data
})
trial_data_chg3 <- do.call(rbind, by_subject_chg)
diffdf::diffdf(trial_data_chg, trial_data_chg3)
```

```
No issues were found!
```

Compared to the `{dplyr}` approach this is a bit clumsy but it certainly does the job and you don't need any add-on packages. `split()`—as the name suggests—splits its input `data.frame` into a `list` of `data.frame`s, one for each level of the second argument. `lapply()` applies a function to every element of a `list`. `do.call(rbind, )` combines the datasets in the `list` back to a single `data.frame`.

Actually you can combine `split()` and `lapply()` into a single step using `by()` which makes it more concise.

```
by_subject_chg <- by(
  data = trial_data,
  INDICES = trial_data$USUBJID,
  FUN = function(data) {
    data$CHG <- data$AVAL - data$AVAL[data$AVISIT == "Baseline"]
    data
  }
)
trial_data_chg4 <- do.call(rbind, by_subject_chg)
diffdf::diffdf(trial_data_chg, trial_data_chg4)
```

```
No issues were found!
```

The example dataset contains only a single laboratory measure but in practice that's never the case. Let's have a look at how to calculate change from baseline when having multiple lab parameters in the dataset. To do so I will duplicate the dataset and add a variable called `PARAM` to identify different lab measures.

```
trial_data$PARAM <- "Hemoglobin"
trial_data2 <- trial_data
```

```
trial_data2$PARAM <- "WBC" # White Blood Count
trial_data_mult <- rbind(trial_data, trial_data2)
```

First, the {dplyr} version.

```
trial_data_mult_chg <- trial_data_mult %>%
  group_by(USUBJID, PARAM) %>%
  mutate(CHG = AVAL - AVAL[AVISIT == "Baseline"]) %>%
  ungroup()

trial_data_mult_chg %>%
  select(USUBJID, AVISIT, AVAL, CHG) %>%
  head(7)
```

```
# A tibble: 7 x 4
  USUBJID          AVISIT    AVAL     CHG

1 ABC123456.000001 Baseline  10.2  0
2 ABC123456.000001 Week 4    11.3  1.08
3 ABC123456.000001 Week 8    12.3  2.09
4 ABC123456.000001 Week 12   13.5  3.31
5 ABC123456.000001 Week 16   12.2  2.00
6 ABC123456.000001 Week 20   10.2 -0.0213
7 ABC123456.000001 Week 24   12.3  2.09
```

```
trial_data_mult_chg %>%
  select(USUBJID, AVISIT, AVAL, CHG) %>%
  tail(7)
```

```
# A tibble: 7 x 4
  USUBJID          AVISIT    AVAL    CHG

1 ABC123456.000300 Baseline  9.75  0
2 ABC123456.000300 Week 4    8.88 -0.871
3 ABC123456.000300 Week 8    8.57 -1.18
4 ABC123456.000300 Week 12   8.13 -1.62
5 ABC123456.000300 Week 16   8.87 -0.880
6 ABC123456.000300 Week 20  12.7   2.95
7 ABC123456.000300 Week 24  12.0   2.20
```

That was easy. All I had to do was to add a second grouping variable, i.e. PARAM.

Next, the base R version.

```
by_subject_chg <- by(
  data = trial_data_mult,
  INDICES = list(trial_data_mult$USUBJID, trial_data_mult$PARAM),
  FUN = function(data) {
    data$CHG <- data$AVAL - data$AVAL[data$AVISIT == "Baseline"]
    data
  }
)
trial_data_mult_chg2 <- do.call(rbind, by_subject_chg)
diffdf::diffdf(trial_data_mult_chg, trial_data_mult_chg2)
```

```
No issues were found!
```

Again a minimal change. Just pass a list of variables to split by to the INDICES argument of by().

There you have it, that's how you calculate change from baseline in R using {dplyr} and good old {base}.

If you enjoyed reading this post please share it with your friends and colleagues. Thank you!