Data Exploration by Tidyverse

Saeed Omidi

4/11/2019

Introduction

Tidyverse is a collection of packages that are generally used in everyday data analysis. This includes, *dplyr* and *ggplot2*. The aim of this short tutorial is to help you getting familiarized with these tools.

Data description

We are going to work with the following data:

- `fvt`: variant table for the entire run, which consists of 13540 variants from 28
- `fft`: the fusion table for the same number of samples, this consist of 20

Let's look at the data:

```
class(fvt)
```

```
## [1] "data.frame"
```

```
fvt <- dplyr::as_tibble(fvt)
```

dplyr functions

There are many useful functions implemented within *dplyr* package. However, we will only cover a handful of functions which happened to be the mostly used ones.

```
df <- fvt %>%
  dplyr::mutate(
    unique_id = sprintf('%s_%s', gene, c.DNA),
    var_frac = as.numeric(stringr::str_replace(var_percent, '%$', ''))
    )  %>%
  dplyr::select(sample, gene, alt, ref, type, var_frac, unique_id, filter)
```

```
## Warning: package 'bindrcpp' was built under R version 3.4.4
```

```
df
```

```
## # A tibble: 13,540 x 8
##    sample     gene  alt   ref   type  var_frac unique_id  filter
##    <chr>      <chr> <chr> <chr> <chr>    <dbl> <chr>      <chr>
##  1 ADNSTANDA~ AKT1  T     C     SNP        0.8 AKT1_c.10~ low_variant_frac~
##  2 ADNSTANDA~ ALK   T     G     SNP       18.8 ALK_c.325~ low_alt_read_gro~
##  3 ADNSTANDA~ AXL   G     A     SNP        0.9 AXL_c.226~ low_variant_frac~
##  4 ADNSTANDA~ AXL   T     C     SNP        4.5 AXL_c.229~ low_variant_frac~
##  5 ADNSTANDA~ AXL   T     C     SNP        0.6 AXL_c.233~ low_variant_frac~
##  6 ADNSTANDA~ AXL   T     C     SNP        1.6 AXL_c.233~ low_variant_frac~
##  7 ADNSTANDA~ AXL   C     T     SNP        2.4 AXL_c.233~ low_variant_frac~
##  8 ADNSTANDA~ AXL   G     A     SNP        0.8 AXL_c.245~ low_variant_frac~
```

```
##  9 ADNSTANDA~ AXL   T     G     SNP        0.8 AXL_c.251~ low_variant_frac~
## 10 ADNSTANDA~ AXL   G     A     SNP        0.2 AXL_c.257~ low_variant_frac~
## # ... with 13,530 more rows
```

With function `mutate` we can calculate new columns and with `select` we can select or de-select some of the columns. For instance:

```
dplyr::select(df, -gene, -filter, sample_name = sample)
```

```
## # A tibble: 13,540 x 6
##    sample_name             alt   ref   type  var_frac unique_id
##    <chr>                   <chr> <chr> <chr>    <dbl> <chr>
##  1 ADNSTANDARD-CTL-SERIE26 T     C     SNP        0.8 AKT1_c.10G>A
##  2 ADNSTANDARD-CTL-SERIE26 T     G     SNP       18.8 ALK_c.3250C>A
##  3 ADNSTANDARD-CTL-SERIE26 G     A     SNP        0.9 AXL_c.2264A>G
##  4 ADNSTANDARD-CTL-SERIE26 T     C     SNP        4.5 AXL_c.2299C>T
##  5 ADNSTANDARD-CTL-SERIE26 T     C     SNP        0.6 AXL_c.2333+17C>T
##  6 ADNSTANDARD-CTL-SERIE26 T     C     SNP        1.6 AXL_c.2333+51C>T
##  7 ADNSTANDARD-CTL-SERIE26 C     T     SNP        2.4 AXL_c.2333+83T>C
##  8 ADNSTANDARD-CTL-SERIE26 G     A     SNP        0.8 AXL_c.2454A>G
##  9 ADNSTANDARD-CTL-SERIE26 T     G     SNP        0.8 AXL_c.2513G>T
## 10 ADNSTANDARD-CTL-SERIE26 G     A     SNP        0.2 AXL_c.2576A>G
## # ... with 13,530 more rows
```

Next, we are often interested to select some of the rows based on some conditions. This is done by `filter`.

```
df %>%
  dplyr::filter(type == 'SNP' & filter == '.')
```

```
## # A tibble: 161 x 8
##    sample            gene   alt   ref   type  var_frac unique_id     filter
##    <chr>             <chr>  <chr> <chr> <chr>    <dbl> <chr>         <chr>
##  1 ADNSTANDARD-CTL-~ MET    T     C     SNP       76.7 MET_c.3912C>T .
##  2 ADNSTANDARD-CTL-~ PIK3CA G     A     SNP       22.9 PIK3CA_c.314~ .
##  3 ADNSTANDARD-CTL-~ FGFR1  A     G     SNP       13.8 FGFR1_c.2400~ .
##  4 ADNSTANDARD-CTL-~ THADA  A     T     SNP       15.8 THADA_c.4153~ .
##  5 ADNSTANDARD-CTL-~ IDH2   A     G     SNP       11.9 IDH2_c.528C>T .
##  6 ADNSTANDARD-CTL-~ KRAS   T     C     SNP       12.7 KRAS_c.38G>A  .
##  7 ADNSTANDARD-CTL-~ AXL    A     T     SNP       20.3 AXL_c.1766T>A .
##  8 ADNSTANDARD-CTL-~ ALK    T     G     SNP       71.2 ALK_c.3375C>A .
##  9 BM180086          THADA  A     C     SNP       40.5 THADA_c.3744~ .
## 10 BM180086          MET    T     C     SNP       42.5 MET_c.3912C>T .
## # ... with 151 more rows
```

Above we saw the example of a condition with '"&'" operation. We can also use `|` and `!` for logical OR and NOT, respectively. Sometimes it's also useful to use `%in%` function for checking membership.

```
df %>%
  dplyr::filter(ref %in% c('C', 'T') & grepl('possible_deamination', filter))
```

```
## # A tibble: 2,612 x 8
##    sample     gene  alt   ref   type  var_frac unique_id  filter
##    <chr>      <chr> <chr> <chr> <chr>    <dbl> <chr>      <chr>
##  1 ADNSTANDA~ AKT1  T     C     SNP        0.8 AKT1_c.10~ low_variant_frac~
##  2 ADNSTANDA~ AXL   T     C     SNP        4.5 AXL_c.229~ low_variant_frac~
##  3 ADNSTANDA~ AXL   T     C     SNP        0.6 AXL_c.233~ low_variant_frac~
##  4 ADNSTANDA~ AXL   T     C     SNP        0.3 AXL_c.260~ low_variant_frac~
##  5 ADNSTANDA~ AXL   T     C     SNP        0.2 AXL_c.265~ low_variant_frac~
```

```
##  6 ADNSTANDA~ MET   T     C     SNP        0.2 MET_c.113~ low_variant_frac~
##  7 ADNSTANDA~ MET   T     C     SNP        0   MET_c.152~ low_variant_frac~
##  8 ADNSTANDA~ MET   T     C     SNP        0.7 MET_c.153~ low_variant_frac~
##  9 ADNSTANDA~ MET   T     C     SNP        0.3 MET_c.160~ low_variant_frac~
## 10 ADNSTANDA~ MET   T     C     SNP        0.4 MET_c.162~ low_variant_frac~
## # ... with 2,602 more rows
```

Sorting the data can be done by `arrange`. Here is how:

```
df %>%
  dplyr::filter(type == 'SNP' & filter == '.') %>%
  dplyr::select(-filter, -type) %>%
  dplyr::arrange(desc(var_frac))
```

```
## # A tibble: 161 x 6
##    sample   gene  alt   ref   var_frac unique_id
##    <chr>    <chr> <chr> <chr>    <dbl> <chr>
##  1 BM180086 HRAS  G     A          100 HRAS_c.81T>C
##  2 BM180191 RET   T     G          100 RET_c.2307G>T
##  3 BM180193 RET   T     G          100 RET_c.2307G>T
##  4 BM180193 HRAS  G     A          100 HRAS_c.81T>C
##  5 BM180197 RET   T     G          100 RET_c.2307G>T
##  6 BM180200 RET   T     G          100 RET_c.2307G>T
##  7 BM180200 THADA A     T          100 THADA_c.4153A>T
##  8 BM180218 MET   T     C          100 MET_c.3912C>T
##  9 BM180219 RET   T     G          100 RET_c.2307G>T
## 10 BM180226 MET   T     C          100 MET_c.3912C>T
## # ... with 151 more rows
```

In some applications, specially if we're dealing with huge number of rows, we would like to take a random subset of the data.

```
df %>%
  dplyr::filter(type == 'SNP' & filter == '.') %>%
  dplyr::sample_n(10)
```

```
## # A tibble: 10 x 8
##    sample             gene  alt   ref   type  var_frac unique_id     filter
##    <chr>              <chr> <chr> <chr> <chr>    <dbl> <chr>         <chr>
##  1 BM180225           EGFR  G     T     SNP       33.7 EGFR_c.2184+~ .
##  2 ADNSTANDARD-CTL-S~ KRAS  T     C     SNP       12.7 KRAS_c.38G>A  .
##  3 BM180253           CCND1 A     G     SNP       57.8 CCND1_c.723G~ .
##  4 BM180226           THADA A     T     SNP       72.7 THADA_c.4153~ .
##  5 BM180197           CCND1 A     G     SNP        9.2 CCND1_c.723G~ .
##  6 BM180173           NRAS  T     C     SNP        5   NRAS_c.109G>A .
##  7 BM180128           THADA A     T     SNP       99.8 THADA_c.4153~ .
##  8 BM180286           HRAS  G     A     SNP      100   HRAS_c.81T>C  .
##  9 BM180197           FGFR3 C     T     SNP       84   FGFR3_c.882T~ .
## 10 BM180191           ALK   A     G     SNP        5.8 ALK_c.3187C>T .
```

Similarly, we can use `sample_frac(f)` to take a fraction `f` of the rows, where `f` is a value between 0 and 1.

Grouping the data

Often we use grouping functionality to group the data and calculate statistics on the group.

```
df %>%
  dplyr::filter(filter == '.') %>%
  dplyr::group_by(sample) %>%
  dplyr::summarise(
    total = n(),
    mean_var_frac = round(mean(var_frac), 2))
```

```
## # A tibble: 27 x 3
##    sample                   total mean_var_frac
##    <chr>                    <int>         <dbl>
##  1 ADNSTANDARD-CTL-SERIE26      9          28.9
##  2 BM180086                     4          70.7
##  3 BM180128                     5          45.1
##  4 BM180173                    11          14.6
##  5 BM180174                     7          44.9
##  6 BM180189                     4          30.1
##  7 BM180191                     6          48.2
##  8 BM180193                     6          62.3
##  9 BM180197                    14          58.7
## 10 BM180200                     3          89.3
## # ... with 17 more rows
```

Grouping can be easily extended to more than one variable. For example, we want to group samples and variant types. Bellow, shows hows it's done:

```
df %>%
  dplyr::filter(filter == '.') %>%
  dplyr::group_by(sample, type) %>%
  dplyr::summarise(
    total = n(),
    mean_var_frac = round(mean(var_frac), 2)) %>%
  dplyr::arrange(desc(total))
```

```
## # A tibble: 33 x 4
## # Groups:   sample [27]
##    sample                   type  total mean_var_frac
##    <chr>                    <chr> <int>         <dbl>
##  1 BM180225                 SNP      18          46.7
##  2 BM180197                 SNP      14          58.7
##  3 BM180173                 SNP      11          14.6
##  4 BM180241                 SNP      10          24.8
##  5 ADNSTANDARD-CTL-SERIE26 SNP       8          30.7
##  6 BM180226                 SNP       7          37.2
##  7 BM180266                 SNP       7          63.4
##  8 BM180174                 SNP       6          49.1
##  9 BM180191                 SNP       6          48.2
## 10 BM180193                 SNP       6          62.3
## # ... with 23 more rows
```

Grouping can be used to identify samples with high amount of C>T:G>A substitution.

```
deamination.df <- df %>%
  dplyr::filter(type != 'INDEL') %>%
  dplyr::mutate(mutation = sprintf('%s>%s', ref, alt)) %>%
  dplyr::group_by(sample, mutation) %>%
  dplyr::summarise(
```

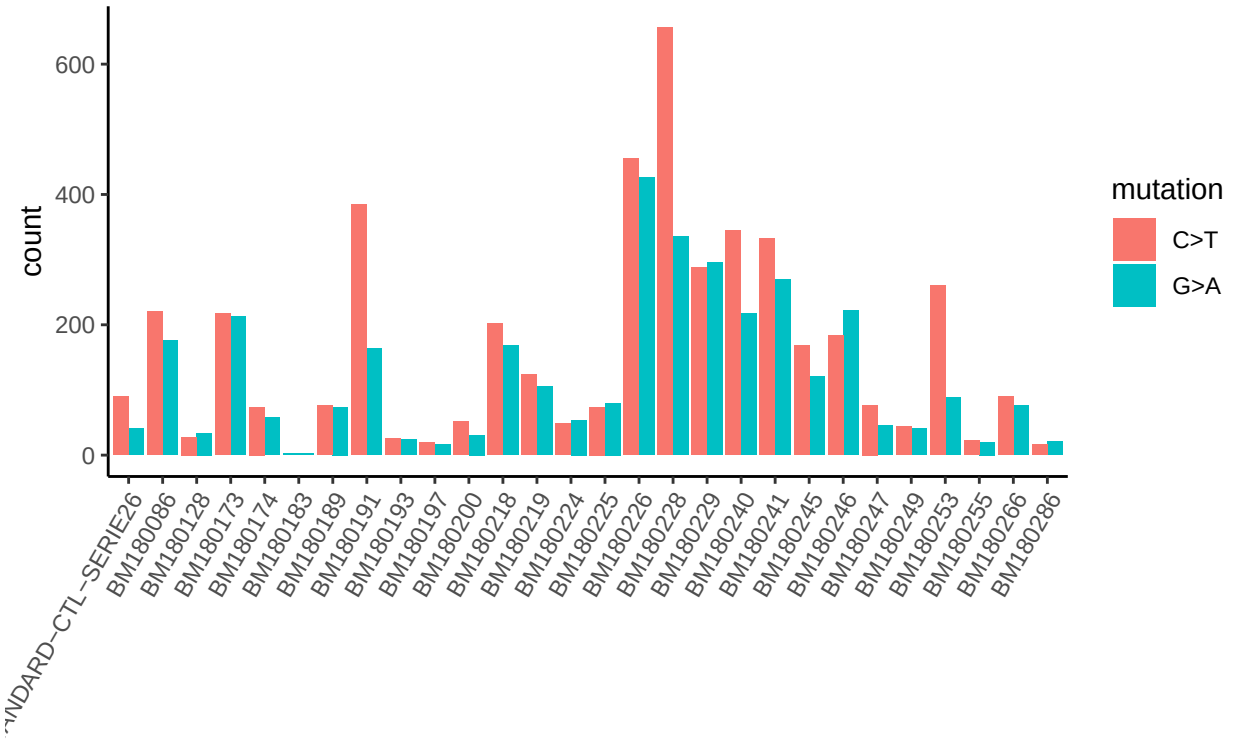
```
    count = n(),
    mean_var_frac = mean(var_frac, na.rm = T)
  ) %>%
  dplyr::filter(mutation %in% c('C>T', 'G>A'))

deamination.df
```

```
## # A tibble: 55 x 4
## # Groups:   sample [28]
##    sample                  mutation count mean_var_frac
##    <chr>                   <chr>    <int>         <dbl>
##  1 ADNSTANDARD-CTL-SERIE26 C>T         90          7.43
##  2 ADNSTANDARD-CTL-SERIE26 G>A         41         12.7
##  3 BM180086                C>T        220          2.55
##  4 BM180086                G>A        176          4.68
##  5 BM180128                C>T         28         50.1
##  6 BM180128                G>A         34         18.4
##  7 BM180173                C>T        217          9.47
##  8 BM180173                G>A        213          6.78
##  9 BM180174                C>T         74         16.7
## 10 BM180174                G>A         58         12.2
## # ... with 45 more rows
```

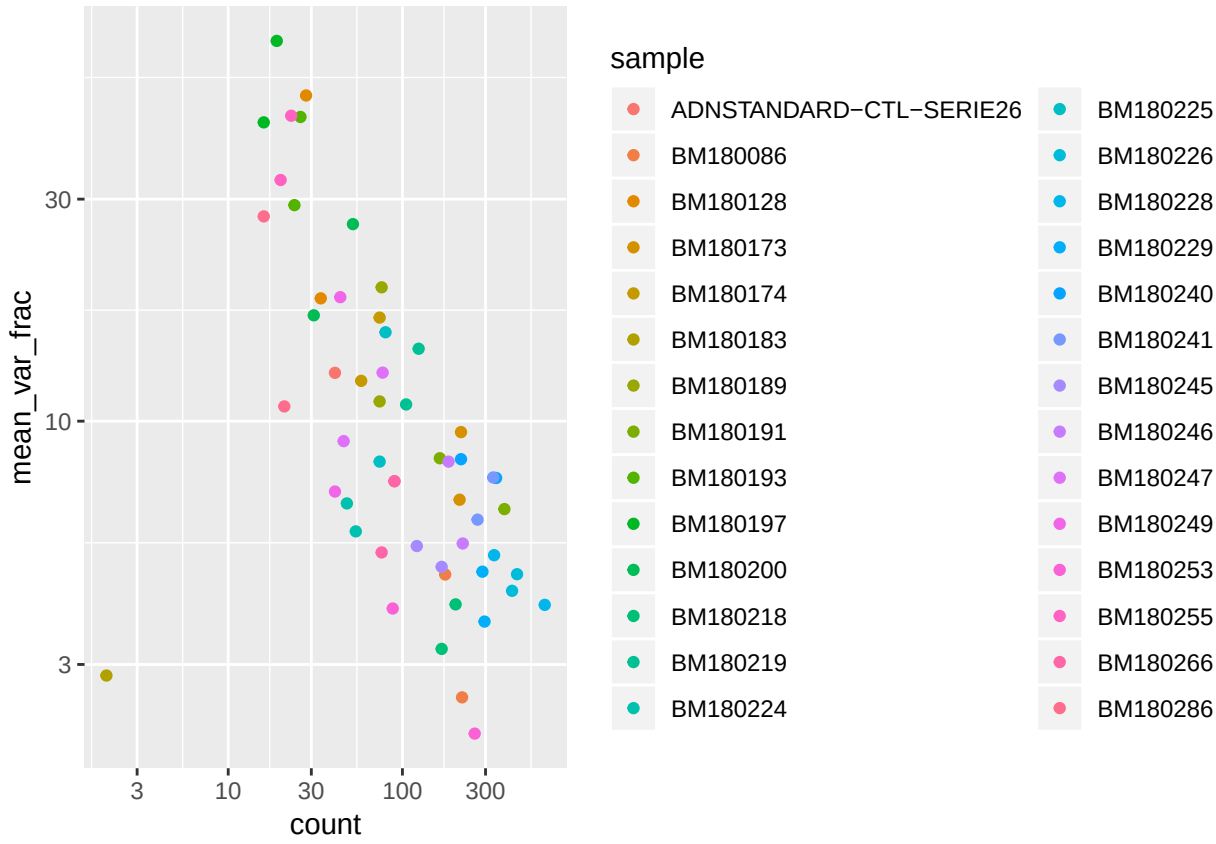
We can now use ggplot2 to easily visualize the data:

```
ggplot2::ggplot(deamination.df) +
  ggplot2::geom_bar(aes(x = sample, y = count, fill = mutation), stat = 'identity', position = 'dodge') +
  ggplot2::theme_classic() +
  ggplot2::theme(axis.text.x = element_text(angle = 60, hjust = 1))
```



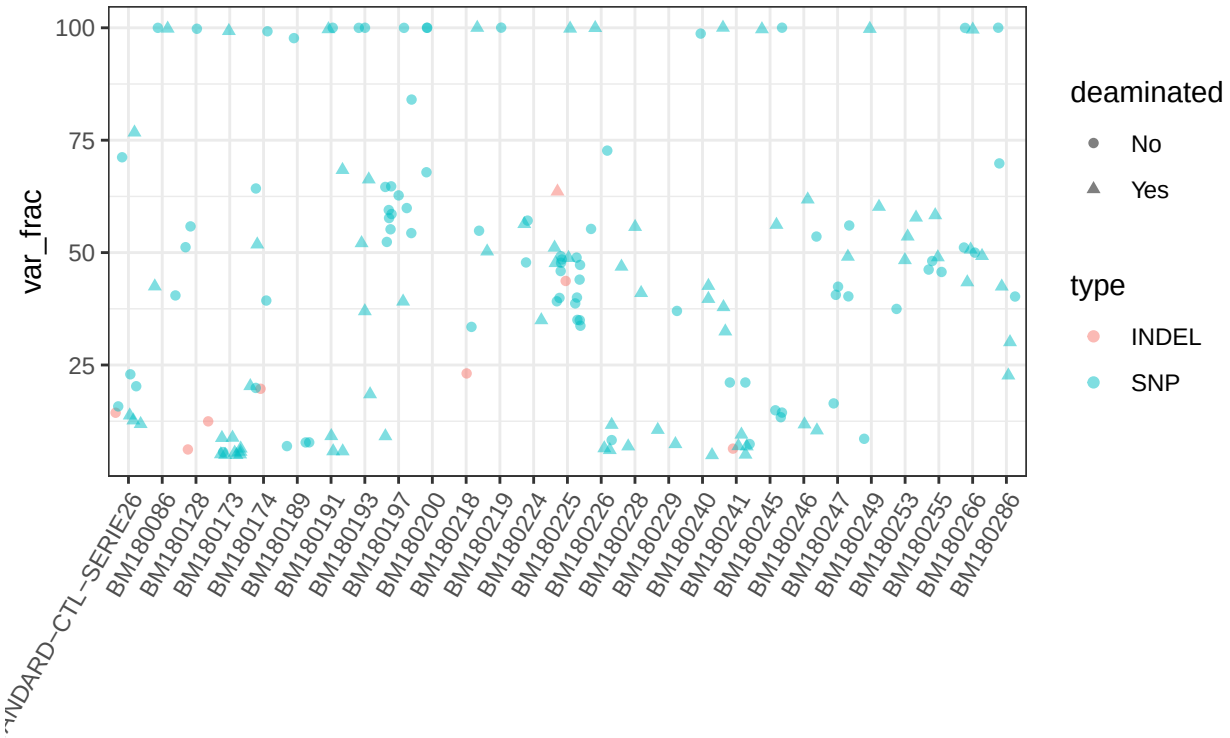
Another way of visualizing the data can be by looking at the relationship between average variant fraction and the total number of deaminated bases:

```
ggplot2::ggplot(deamination.df) +
  ggplot2::geom_point(aes(x = count, y = mean_var_frac, col = sample)) +
  scale_x_log10() + scale_y_log10()
```



Let's dig in more into `ggplot2`. In the following example we draw a plot of variant fraction for different samples.

```
df %>%
  dplyr::filter(filter == '.') %>%
  dplyr::mutate(deaminated = ifelse((ref == 'C' & alt == 'T') | (ref == 'G' & alt == 'A'), 'Yes', 'No'))
  ggplot2::ggplot() +
  geom_point(aes(x = sample, y = var_frac, colour = type, pch = deaminated), position = 'jitter', alpha
  ggplot2::theme_bw() +
  ggplot2::theme(axis.text.x = element_text(angle = 60, hjust = 1)) +
  xlab('')
```



In some tasks, we want to combine a number of columns together. For instance, what we already calculated for `unique_id`. This can be done also by some of the *tidyr* functionalities:

```
fvt %>%
  tidyr::unite(unique_id, c('gene', 'chromosome', 'genome_position', 'ref', 'alt'), sep = '-') %>%
  dplyr::select(sample, unique_id)
```

```
## # A tibble: 13,540 x 2
##    sample                  unique_id
##    <chr>                   <chr>
##  1 ADNSTANDARD-CTL-SERIE26 AKT1-14-105258971-C-T
##  2 ADNSTANDARD-CTL-SERIE26 ALK-2-29446317-G-T
##  3 ADNSTANDARD-CTL-SERIE26 AXL-19-41763465-A-G
##  4 ADNSTANDARD-CTL-SERIE26 AXL-19-41763500-C-T
##  5 ADNSTANDARD-CTL-SERIE26 AXL-19-41763551-C-T
##  6 ADNSTANDARD-CTL-SERIE26 AXL-19-41763585-C-T
##  7 ADNSTANDARD-CTL-SERIE26 AXL-19-41763617-T-C
##  8 ADNSTANDARD-CTL-SERIE26 AXL-19-41765578-A-G
##  9 ADNSTANDARD-CTL-SERIE26 AXL-19-41765637-G-T
## 10 ADNSTANDARD-CTL-SERIE26 AXL-19-41765700-A-G
## # ... with 13,530 more rows
```

Joining the tables

Pivoting operations