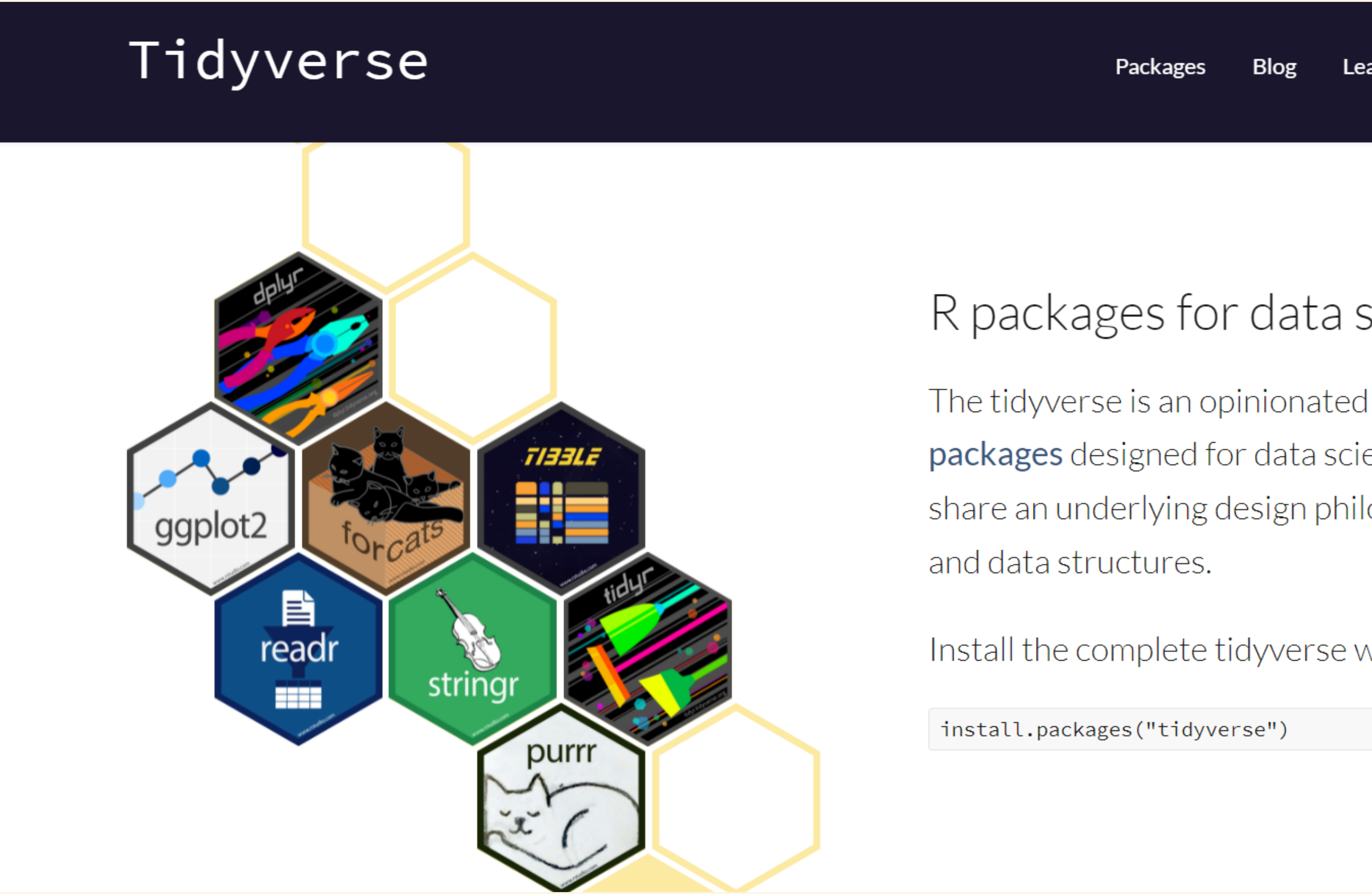
Data Processing in the Tidyverse

Using Administrative Data for
Clinical and Health Services Research

Overview

- Tidyverse plug
- Selecting columns with `select`
- Subsetting rows with `filter`
- Summarizing by group with `group_by` / `summarise`

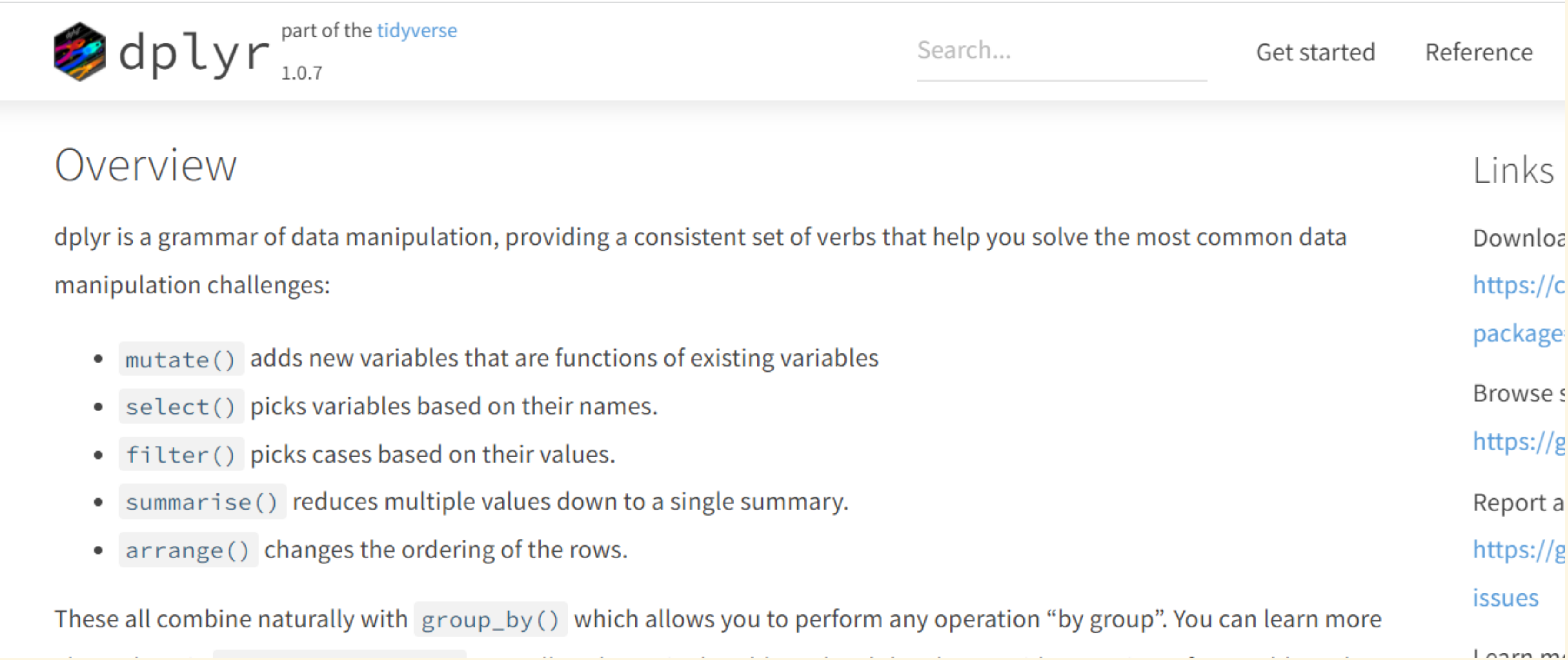


R packages for data s

The tidyverse is an opinionated
packages designed for data scie
share an underlying design philo
and data structures.

Install the complete tidyverse w

```
install.packages("tidyverse")
```



dplyr part of the tidyverse 1.0.7

Search...

Get started Reference

Overview

dplyr is a grammar of data manipulation, providing a consistent set of verbs that help you solve the most common data manipulation challenges:

- `mutate()` adds new variables that are functions of existing variables
- `select()` picks variables based on their names.
- `filter()` picks cases based on their values.
- `summarise()` reduces multiple values down to a single summary.
- `arrange()` changes the ordering of the rows.

These all combine naturally with `group_by()` which allows you to perform any operation “by group”. You can learn more

Links

Downloa

https://c

package

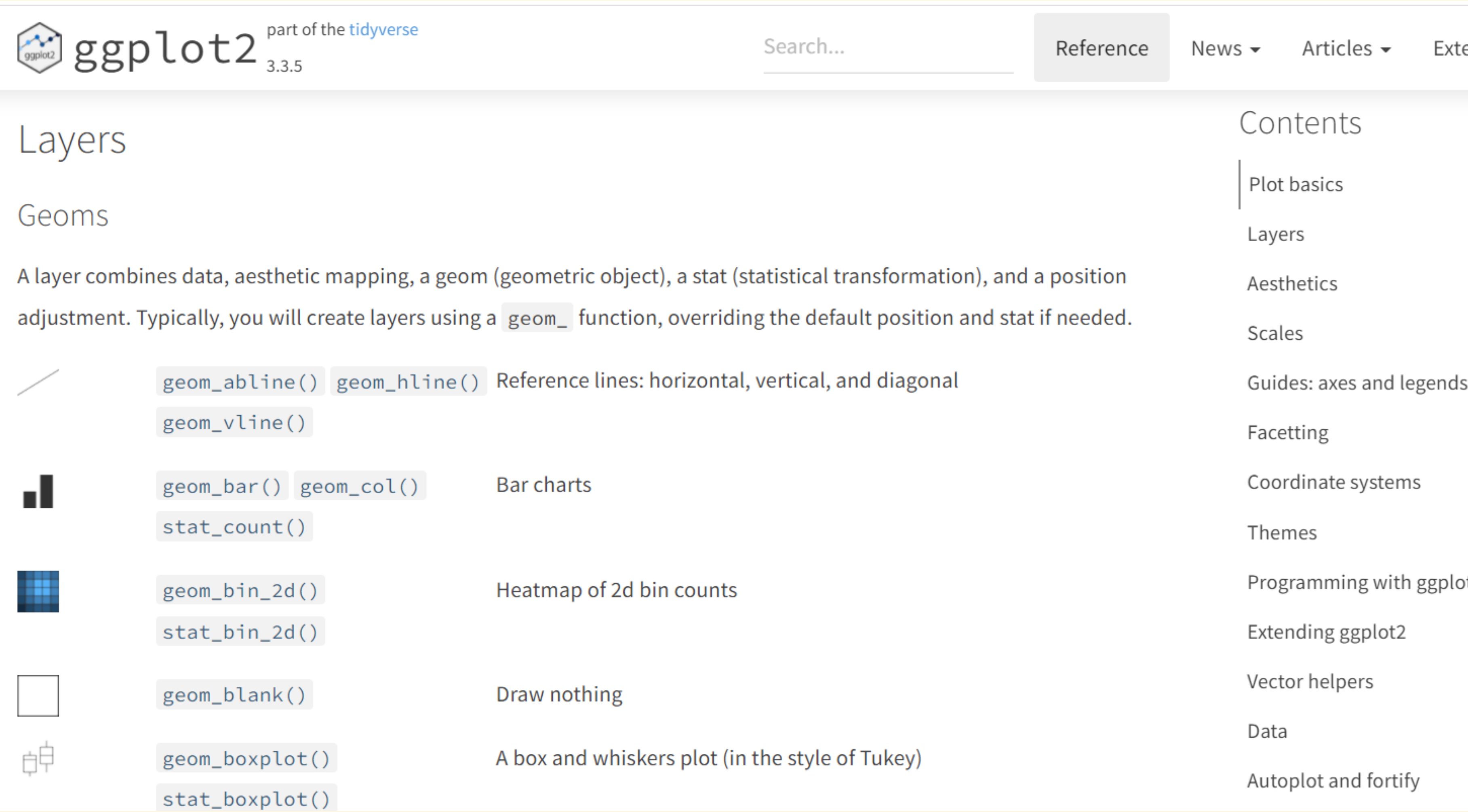
Browse s

https://g

Report a

https://g

issues



Layers

Geoms

A layer combines data, aesthetic mapping, a geom (geometric object), a stat (statistical transformation), and a position adjustment. Typically, you will create layers using a `geom_` function, overriding the default position and stat if needed.

	`geom_abline()` `geom_hline()` `geom_vline()`	Reference lines: horizontal, vertical, and diagonal
	`geom_bar()` `geom_col()` `stat_count()`	Bar charts
	`geom_bin_2d()` `stat_bin_2d()`	Heatmap of 2d bin counts
□	`geom_blank()`	Draw nothing
	`geom_boxplot()` `stat_boxplot()`	A box and whiskers plot (in the style of Tukey)

Contents

Plot basics
Layers
Aesthetics
Scales
Guides: axes and legends
Facetting
Coordinate systems
Themes
Programming with ggplo
Extending ggplot2
Vector helpers
Data
Autoplot and fortify

Selecting columns

- Specific columns in a data set can be chosen using `dplyr::select`
  - As with `mutate`, first argument is a data set, subsequent arguments are expressions, and the result is a new data set
- Offers a variety of ways to choose multiple columns at once, making it a significant improvement on approaches available in base R.

Selecting columns

By name

```
# Create a data set of discharge- and patient-specific identifiers.
disch_identifiers <- select(core1p, KEY, VisitLink, DaysToEvent, LOS)
colnames(disch_identifiers)
```

```
## [1] "KEY"         "VisitLink"   "DaysToEvent" "LOS"
```

Selecting columns

By a character vector

```
# Create a data set containing the variables related to patient
# residence.
vars_patient_residence <- c(
  "PSTATE", "PSTCO", "PSTCO2", "PSTCO_GEO", "ZIP")
disch_patient_residence <- select(
  core1p, all_of(vars_patient_residence))
colnames(disch_patient_residence)
```

```
## [1] "PSTATE"    "PSTCO"     "PSTCO2"    "PSTCO_GEO" "ZIP"
```

Selecting columns

By a common prefix

```r
# Create a data set consisting of the procedure code columns.
disch_prcodes <- select(core1p, starts_with("I10_PR"))
colnames(disch_prcodes)
```

```
##  [1] "I10_PR1"  "I10_PR2"  "I10_PR3"  "I10_PR4"  "I10_PR5"  "I10_PR6"
##  [7] "I10_PR7"  "I10_PR8"  "I10_PR9"  "I10_PR10" "I10_PR11" "I10_PR12"
## [13] "I10_PR13" "I10_PR14" "I10_PR15" "I10_PR16" "I10_PR17" "I10_PR18"
## [19] "I10_PR19" "I10_PR20" "I10_PR21" "I10_PR22" "I10_PR23" "I10_PR24"
## [25] "I10_PR25" "I10_PR26" "I10_PR27" "I10_PR28" "I10_PR29" "I10_PR30"
## [31] "I10_PR31"
```

Selecting columns

By a common suffix

```r
# Select all of the "from source" columns (i.e., columns whose values
# were not edited by HCUP).
disch_from_source <- select(core1p, ends_with("_X"))
colnames(disch_from_source)
```

```
## [1] "DISP_X"   "LOS_X"    "PAY1_X"   "RACE_X"   "TOTCHG_X"
```

Selecting columns

By a common sequence of characters somewhere in the names

```
# Select all columns with income-related information.
disch_income <- select(core1p, contains("INC"))
colnames(disch_income)
```

```
## [1] "MEDINCSTQ"   "ZIPINC_QRTL"
```

Selecting columns

By sequence based on column order

- Inspired by shorthand for sequence of integers: `1:3` ≡ `c(1,2,3)`

```r
# Select `VisitLink` and `DSHOSPID` and everything in between.
disch_VisitLink_to_DSHOSPID <- select(core1p, VisitLink:DSHOSPID)
colnames(disch_VisitLink_to_DSHOSPID)
```

```
##  [1] "VisitLink"   "AGE"         "AHOUR"       "ATYPE"      
##  [5] "AWEEKEND"    "DaysToEvent" "DIED"        "DISP_X"     
##  [9] "DISPUB04"    "DISPUNIFORM" "DQTR"        "DSHOSPID"   
```

- Using column positions can be risky
- Suppose we want to select all diagnosis code columns

```
disch_all_dx_codes_wrong <- select(
  core1p, I10_DX_Admitting:I10_DX34, I10_ECAUSE1:I10_ECAUSE6)
colnames(disch_all_dx_codes_wrong)
```

```
##   [1] "I10_DX_Admitting"     "I10_DX1"               "I10_DX2"
##   [6] "I10_DX5"               "I10_DX6"               "I10_DX7"
##  [11] "I10_DX10"              "I10_DX11"              "I10_DX12"
##  [16] "I10_DX15"              "I10_DX16"              "I10_DX17"
##  [21] "I10_DX20"              "I10_DX21"              "I10_DX22"
##  [26] "I10_DX25"              "I10_DX26"              "I10_DX27"
##  [31] "I10_DX30"              "I10_DX31"              "I10_ECAUSE1"
##  [36] "I10_ECAUSE4"           "I10_ECAUSE5"           "I10_ECAUSE6"
##  [41] "I10_NPR"               "I10_PR1"               "I10_PR2"
##  [46] "I10_PR5"               "I10_PR6"               "I10_PR7"
##  [51] "I10_PR10"              "I10_PR11"              "I10_PR12"
##  [56] "I10_PR15"              "I10_PR16"              "I10_PR17"
##  [61] "I10_PR20"              "I10_PR21"              "I10_PR22"
##  [66] "I10_PR25"              "I10_PR26"              "I10_PR27"
##  [71] "I10_PR30"              "I10_PR31"              "KEY"
##  [76] "MEDINCSTQ"             "PAY1"                  "PAY1_X"
##  [81] "PL_RUCC"               "PL_UIC"                "PL_UR_CAT4"
##  [86] "POA_Hosp_Edit1"        "POA_Hosp_Edit3_Value" "PointOfOriginUB04"
##  [91] "PRDAY3"                "PRDAY4"                "PRDAY5"
##  [96] "PRDAY8"                "PRDAY9"                "PRDAY10"
```

Selecting columns

By sequence based on column order

- Different years of data have different numbers of diagnosis codes
- 1% sample data set was formed by stacking the annual CORE files
    - `I10_DX_Admitting`-`I10_DX31` contiguous but not with `I10_DX32`-`I10_DX34`
- In general, selection based on column names is more reliable
- Regardless, always check your selection with `str`, `colnames`, or by printing or `View`ing the resulting data set

Selecting columns

By sequence based on column order

```
disch_all_dx_codes_right <- select(
  core1p, starts_with("I10_DX"), starts_with("I10_ECAUSE"))
colnames(disch_all_dx_codes_right)
```

```
##  [1] "I10_DX_Admitting" "I10_DX1"          "I10_DX2"
##  [4] "I10_DX3"          "I10_DX4"          "I10_DX5"
##  [7] "I10_DX6"          "I10_DX7"          "I10_DX8"
## [10] "I10_DX9"          "I10_DX10"         "I10_DX11"
## [13] "I10_DX12"         "I10_DX13"         "I10_DX14"
## [16] "I10_DX15"         "I10_DX16"         "I10_DX17"
## [19] "I10_DX18"         "I10_DX19"         "I10_DX20"
## [22] "I10_DX21"         "I10_DX22"         "I10_DX23"
## [25] "I10_DX24"         "I10_DX25"         "I10_DX26"
## [28] "I10_DX27"         "I10_DX28"         "I10_DX29"
## [31] "I10_DX30"         "I10_DX31"         "I10_DX32"
## [34] "I10_DX33"         "I10_DX34"         "I10_ECAUSE1"
## [37] "I10_ECAUSE2"      "I10_ECAUSE3"      "I10_ECAUSE4"
## [40] "I10_ECAUSE5"      "I10_ECAUSE6"
```

Selecting columns

By negation

- i.e., "select everything except..."

```
# Create a data set excluding all columns related to diagnosis and
# procedures codes, including POA codes and the `PRDAY`n fields.
disch_no_dx_or_pr_codes <- select(
  core1p,
  !c(
    starts_with("I10_"), starts_with("DXPOA"), starts_with("E_POA"),
    starts_with("PRDAY"))
)
colnames(disch_no_dx_or_pr_codes)
```

```
##  [1] "VisitLink"            "AGE"
##  [3] "AHOUR"                "ATYPE"
##  [5] "AWEEKEND"             "DaysToEvent"
##  [7] "DIED"                 "DISP_X"
##  [9] "DISPUB04"             "DISPUNIFORM"
## [11] "DQTR"                 "DSHOSPID"
```

Subsetting rows

- Aside from the column exclusion, `!` is often used in three cases:

1. Select nonmissing values

    - `!is.na(x)`

2. Checking for non-equality

    - `x != 0`

3. Select values *not in* a specific set of values

    - `!(x %in% y)`
    - Can be difficult to read if part of complex expression, so may want to use special 'not in' operator `x %nin% y`; this requires loading Hmisc with `library(Hmisc)`

Subsetting rows

- Relies on expressions that return `TRUE` or `FALSE`
- Used several of these types of expressions when defining variables in previous lecture:
    - `is.na(LOS)`
    - `AGE >= 18`
    - `PAY1 == 1`
    - `TRAN_IN %in% c(1, 2)`

Subsetting rows

- Tidyverse subsetting uses `filter`
- Can combine multiple expressions or write them as separate arguments
  - The arguments are implicitly combined using `&`

```
disch_adults1 <- filter(core1p, !is.na(AGE) & AGE >= 18)
disch_adults2 <- filter(core1p, !is.na(AGE), AGE >= 18)
```

- Expressions connected by 'or' have to be combined

```
disch_transfer_in_or_out <- filter(
  core1p, TRAN_IN %in% c(1, 2) | TRAN_OUT %in% c(1, 2))
```

Grouping data

- We can tell dplyr we want to process data by a grouping factor using `group_by`
  - Doesn't directly change the data set, just how other functions work

Summarizing by group

- `group_by` is most often used with `summarise` to compute summary statistics within each group

```
# Create a data set containing the number of discharges for each
# patient.
disch_grouped_by_patient <- group_by(core1p, VisitLink)
p_dischcharge_counts <- summarise(
  disch_grouped_by_patient, dischcharge_count = n())
```

- `group_by` paired with `summarise` changes the level of organization of the data—here from discharge level to patient level

Summarizing by group

```
disch_grouped_by_patient
```

```
## # A tibble: 102,733 × 194
## # Groups:   VisitLink [48,085]
##    VisitLink   AGE AHOUR ATYPE AWEEKEND DaysToEvent  DIED DISP_X
##        <dbl> <dbl> <dbl> <dbl>    <dbl>       <dbl> <dbl> <chr>
##  1  15965139    74  1400     1        0       18739     0 01
##  2  16050018    77  1300     1        0       18655     0 03
##  3  16050018    77  2300     1        1       18623     0 03
##  4   2092107    32  2300     1        0       17883     0 01
##  5   8726364    66  1600     1        0       17252     0 06
##  6    390742    61  2300     1        0       18869     0 06
##  7    390742    61     0     1        1       18844     0 01
##  8  24417038    62  2200     1        1       19810     0 01
##  9  24344014    67   600     1        0       20164     0 01
## 10  22892274     9   900     3        0       16852     0 01
## # … with 102,723 more rows, and 186 more variables: DISPUB04 <dbl>,
## #   DISPUNIFORM <dbl>, DQTR <dbl>, DSHOSPID <chr>, DXPOA1 <chr>,
## #   DXPOA2 <chr>, DXPOA3 <chr>, DXPOA4 <chr>, DXPOA5 <chr>,
## #   DXPOA6 <chr>, DXPOA7 <chr>, DXPOA8 <chr>, DXPOA9 <chr>,
## #   DXPOA10 <chr>, DXPOA11 <chr>, DXPOA12 <chr>, DXPOA13 <chr>,
## #   DXPOA14 <chr>, DXPOA15 <chr>, DXPOA16 <chr>, DXPOA17 <chr>,
```

Summarizing by group

```
p_dischcharge_counts
```

```
## # A tibble: 48,085 × 2
##    VisitLink dischcharge_count
##        <dbl>             <int>
##  1       365                 3
##  2       626                 3
##  3       641                 4
##  4       769                 1
##  5       846                 8
##  6       951                 9
##  7      1304                 1
##  8      1411                 2
##  9      1463                 3
## 10      2556                 1
## # … with 48,075 more rows
```

The pipe `%>%` operator

- Many packages in the Tidyverse are designed to work with the pipe operator, `%>%`
- Takes the result of the expression on the 'left-hand side' and passes it as the first argument to the expression on the 'right-hand side'.
- Useful because each of the tidyverse functions we've looked at so far— `mutate`, `select`, `filter`, `group_by`, and `summarise`— all take a data set as their first argument and produce a new data set as output

The pipe %>% operator

So we can write

```
disch_identifiers <- select(core1p, KEY, VisitLink, DaysToEvent, LOS)
```

as

```
disch_identifiers <- core1p %>%
  select(KEY, VisitLink, DaysToEvent, LOS)
```

- Most useful when needing to 'chain' several commands together but don't want to save intermediate results or write a nested set of function calls

Suppose we want the procedure code and procedure day columns for all adults starting in 2016. We could write

```
disch_adults <- filter(core1p, !is.na(AGE), AGE >= 18, YEAR >= 2016)
disch_adults_pra <- select(
  disch_adults,
  KEY, VisitLink, DaysToEvent, LOS, starts_with("I0_PR"),
  starts_with("PRDAY"))
```

or

```
disch_adults_prb <-
  select(
    filter(
      core1p,
      !is.na(AGE), AGE >= 18, YEAR >= 2016
    ),
    KEY, VisitLink, DaysToEvent, LOS, starts_with("I0_PR"),
    starts_with("PRDAY")
  )
```

The pipe %>% operator

But a bit clearer would be

```
disch_adults_prc <- core1p %>%
  filter(!is.na(AGE), AGE >= 18, YEAR >= 2016) %>%
  select(
    KEY, VisitLink, DaysToEvent, LOS, starts_with("I0_PR"),
    starts_with("PRDAY")
  )
```