

# Basic Tabular Data Manipulation

Alejandro Schuler, adapted from Steve Bagley and based on R for Data Science by Hadley Wickham, updated to include GTEx sample data by Nicole Ferraro  
2019, updated July 2021

- filter rows of a dataset based on conditions
- arrange rows of a dataset based on one or more columns
- select columns of a dataset
- mutate existing columns to create new columns
- group and summarize data by one or more columns
- use the pipe to combine multiple operations

# dplyr

This section shows the basic data frame functions (“verbs”) in the `dplyr` package (part of `tidyverse`).



# dplyr verbs

Each operation takes a data frame and produces a new data frame.

- `filter()` picks out rows according to specified conditions
- `select()` picks out columns according to their names
- `arrange()` sorts the row by values in some column(s)
- `mutate()` creates new columns, often based on operations on other columns
- `summarize()` collapses many values in one or more columns down to one value per column

These can all be used in conjunction with `group_by()` which changes the scope of each function from operating on the entire dataset to operating on it group-by-group. These six functions provide the “verbs” for a language of data manipulation.

All work similarly:

1. The first argument is a data frame.
2. The subsequent arguments describe what to do with the data frame, using the variable names (without quotes).
3. The result is a new data frame.

Together these properties make it easy to chain together multiple simple steps to achieve a complex result.

# GTE<sub>x</sub> data

This is a subset of the Genotype Tissue Expression (GTE<sub>x</sub>) dataset

- **The full dataset.** Includes gene expression data, measured via RNA-sequencing, from 54 post-mortem tissues in ~800 individuals. Whole genome sequencing is also available for these individuals as part of the GTE<sub>x</sub> v8 release, available through dbGaP.
- **The sub<sub>set</sub>ted dataset.** We are looking at expression data for just 78 individuals here, in four tissues including blood, heart, lung and liver.
- **Data processing** The expression values have been normalized and corrected for technical covariates and are now in the form of Z-scores, which indicate the distance of a given expression value from the mean across all measurements of that gene in that tissue.
- **Goal.** We will use the data here to illustrate different functions for data transformation, often focused on extracting individuals with extremely high or low expression values for a given gene as compared to the distribution across all samples.

**NOTE:** If copying the code, make sure there are no spaces in the download link (where it wraps to a new line).

```
# Read subsetted data from online file - make sure there are no spaces
gtex_data = read_tsv('https://raw.githubusercontent.com/alejandroschuler/r4ds-
courses/advance-2020/data/gtex.tissue.zscores.advance2020.txt')

# Check number of rows
nrow(gtex_data)
[1] 389922
```



# Filter rows with filter()

- `filter()` lets you filter out rows of a dataset that meet a certain condition
- It takes two arguments: the dataset and the condition

```
filter(gtex_data, Blood >= 12)
# A tibble: 12 x 7
```

	Gene <chr>	Ind <chr>	Blood <dbl>	Heart <dbl>	Lung <dbl>	Liver <dbl>	NTissues <dbl>
1	AC012358.7	GTEX-VUSG	13.6	-1.43	1.22	-0.39	4
2	DCSTAMP	GTEX-12696	13.6	NA	-0.57	-0.91	3
3	DIAPH2-AS1	GTEX-VUSG	12.2	-0.33	1.18	0.67	4
4	DNASE2B	GTEX-12696	14.4	-0.82	-0.92	0.35	4
5	FFAR4	GTEX-12696	12.9	-0.96	-0.67	0.18	4
6	GAPDHP33	GTEX-UPK5	13.8	1.52	-1.48	-1.84	4
7	GTF2A1L	GTEX-VUSG	12.2	1.67	0.78	0.09	4
8	GTF2IP14	GTEX-11NV4	12.2	7.26	5.79	7.06	4
9	KCNT1	GTEX-1KANB	13.5	3.14	0.62	-0.37	4
10	KLK3	GTEX-147F4	15.7	-0.74	-0.44	-0.02	4
11	NAPSA	GTEX-1CB4J	12.3	-0.29	-0.44	-0.14	4
12	REN	GTEX-U8XE	18.9	-0.57	NA	0.09	3

# Exercise

- What is the result of running this code?

```
nrow(gtex_data)
[1] 389922
```

```
filter(gtex_data, NTissues <= 2)
filter(gtex_data, Heart <= -5)
nrow(gtex_data)
```

- Remember, functions usually do not change their arguments!

```
low_expression_blood = filter(gtex_data, Blood <= -5)
low_expression_blood_heart = filter(low_expression_blood, Heart <= -5)
nrow(low_expression_blood_heart)
[1] 3
```

# Combining constraints in filter

```
filter(gt看_data, Blood <= -5, Heart <= -5)
# A tibble: 3 x 7
  Gene      Ind      Blood  Heart  Lung Liver NTissues
  <chr>    <chr>    <dbl>  <dbl>  <dbl> <dbl>    <dbl>
1 ATP5A1  GTEX-YFC4  -5.35  -6.05  -7.96 -4.4      4
2 GHITM   GTEX-WK11  -5.7   -7.24  -7.37 -4.06     4
3 MTATP6P1 GTEX-1KD5A -9.18 -10.1  -10.3  -9.52     4
```

- This filters by the **conjunction** of the two constraints—both must be satisfied.
- Constraints appear as second (and third...) arguments, separated by commas.



# Filtering out all rows

```
filter(gtex_data, NTissues > 5)
# A tibble: 0 x 7
# ... with 7 variables: Gene <chr>, Ind <chr>, Blood <dbl>, Heart <dbl>,
#   Lung <dbl>, Liver <dbl>, NTissues <dbl>
```

- If the constraint is too severe, then you will select **no** rows, and produce a zero row sized tibble.

# Comparison operators

- `==` and `!=` test for equality and inequality (do not use `=` for equality)
- `>` and `<` test for greater-than and less-than
- `>=` and `<=` are greater-than-or-equal and less-than-or-equal
- these can also be used directly on vectors outside of data frames

```
c(1, 5, -22, 4) > 0  
[1]  TRUE  TRUE FALSE  TRUE
```

Aside: computers are not perfect, so be careful with checking equality

```
sqrt(2) ^ 2 == 2  
[1] FALSE  
1 / 49 * 49 == 1  
[1] FALSE
```

You can use `near()` to check that two numbers are the same (up to “machine precision”)

```
near(sqrt(2) ^ 2, 2)  
[1] TRUE  
near(1 / 49 * 49, 1)  
[1] TRUE
```

# Comparing to NA

- The other “gotcha” is that `==` cannot be used to compare to `NA`:

```
x = NA
x == NA
[1] NA
```

- The result actually makes sense though, because I'm asking if “I don't know” is the same as “I don't know”. Since either side could be any value, the right answer is “I don't know”.
- To check if something is `NA`, use `is.na()`

```
x = NA
is.na(x)
[1] TRUE
```

# Logical conjunctions

Logic in R utilizes the following vocabulary: and (&), or (|), in (%in%), not (!), and equals (==)

- When **OR** (pipe symbol: |) is used to join two conditions, *at least one condition* must be true for the statement to be true. Otherwise it's false.

```
filter(gtex_data, Lung > 6 | Liver < -6)
# A tibble: 73 x 7
  Gene      Ind      Blood Heart  Lung Liver NTissues
  <chr>    <chr>    <dbl> <dbl> <dbl> <dbl>   <dbl>
1 ACOT12   GTEX-12WSD  5.43  0.53  8.2   0.71    4
2 ACSL6    GTEX-X261   2.45  1.04  7.03  2.4     4
3 ADAL     GTEX-1EWIQ  0.69 -0.15  6.28 -0.52    4
4 AGAP2    GTEX-1GN73  2.32  1.46  6.1   0.89    4
5 ALDOB    GTEX-12WSD  0.93 -0.42  6.06 -0.08    4
...
```

- When **AND** (ampersand symbol: &) is used to join two conditions, *both conditions* must be true for the statement to be true. Otherwise it's false. Separating conditions by a comma is the same as using & inside filter()

```
filter(gtex_data, Lung > 6 & Liver < -6)
# A tibble: 0 x 7
# ... with 7 variables: Gene <chr>, Ind <chr>, Blood <dbl>, Heart <dbl>,
#   Lung <dbl>, Liver <dbl>, NTissues <dbl>
```

- Multiple conjunctions can describe complex logical conditions

# Logical conjunctions

```
filter(gt看_data, !(Blood < 6 | Lung < 6))  
# A tibble: 5 x 7  
  Gene          Ind      Blood Heart  Lung Liver NTissues  
  <chr>        <chr>    <dbl> <dbl> <dbl> <dbl>    <dbl>  
1 CTAG2       GTEX-17HGU  6.61  0.65  7.4   2.85     4  
2 GTF2IP14    GTEX-X3Y1  10.3   7.46  8.12  3.67     4  
3 KLK3        GTEX-X261  11.1   0.02  8.39  5.02     4  
4 RP11-1228E12.1 GTEX-1KANB  6.18  4.08  9.69  6.63     4  
5 TDRD1       GTEX-ZEX8  10.3   3.47  6.19  0.3      4
```

- The exclamation point ! means “NOT”, which negates the logical condition

# Logical conjunctions

```
filter(gtex_data, NTissues %in% c(1,2)) # equivalent to filter(gtex_data,
NTissues==1 | NTissues==2)
# A tibble: 132 x 7
  Gene      Ind      Blood Heart  Lung Liver NTissues
  <chr>    <chr>    <dbl> <dbl> <dbl> <dbl>   <dbl>
1 AC016757.3 GTEX-131YS  1.43  NA    NA    -0.07     2
2 ACTG1P1    GTEX-15RJE  NA    NA    -0.41 -0.1      2
3 ACVR2B-AS1 GTEX-1GN73 -0.86  NA    NA    0.46      2
4 ADAMTSL1   GTEX-1LGRB -0.48  NA    NA    -0.76      2
5 ADGRF5     GTEX-ZPU1  -0.13  NA    NA    -0.68      2
6 AOAHA     GTEX-11GSP -1.27  NA    NA    0.41      2
7 ARV1       GTEX-12WSD -1.07  NA    NA    1.27      2
8 BORCS7     GTEX-X261  NA     0.93  NA    0.11      2
9 C16orf46   GTEX-ZEX8  -0.95  NA    NA    0.21      2
10 C4orf19   GTEX-ZVT3  -0.88  NA    NA    0.82      2
# ... with 122 more rows
```

- `%in%` returns true for all elements of the thing on the left that are also elements of the thing on the right. This is actually shorthand for a match function (use `help('%in%')` to learn more)

**Caution!** `in` (without the flanking percent signs) has a different meaning - we'll get to this later in the course.

## Exercise: High expression

- How many individuals have high expression ( $Z > 3$ ) for CTAG2 in both blood and lung?



## Exercise: High expression

- How many individuals have high expression ( $Z > 3$ ) for CTAG2 in both blood and lung?

```
nrow(filter(gtex_data, Gene=="CTAG2", Blood > 3, Lung > 3))  
[1] 3
```

## Exercise: Low expression

- Create a dataset of all individual-gene pairs with low expression ( $Z < -3$ ) in blood and heart tissues

## Exercise: Low expression

- Create a dataset of all individual-gene pairs with low expression ( $Z < -3$ ) in blood and heart tissues

```
filter(gtex_data, Blood < -3, Heart < -3)
# A tibble: 26 x 7
```

	Gene	Ind	Blood	Heart	Lung	Liver	NTissues
	<chr>	<chr>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
1	ABC7-42389800N19.1	GTEX-14E7W	-4.24	-3.29	-3.55	-3.2	4
2	ALDH9A1	GTEX-11NUK	-3.45	-4.78	-4.15	-3.8	4
3	ARPC3	GTEX-1MJIX	-4.21	-3.76	-4.56	2.42	4
4	ATL3	GTEX-ZF29	-5.07	-4.22	-2.77	-2.82	4
5	ATP5A1	GTEX-YFC4	-5.35	-6.05	-7.96	-4.4	4
6	CCDC163	GTEX-1KANB	-3.06	-3.05	-0.94	0.48	4
7	COQ6	GTEX-ZEX8	-4.49	-3.11	-0.19	-0.9	4
8	CRYL1	GTEX-WFON	-3.77	-6.42	-6.14	-4.11	4
9	CTPS2	GTEX-1E2YA	-3.92	-6.71	-4.06	-1.98	4
10	ENGASE	GTEX-147JS	-3.96	-3.17	-3.62	-3.22	4

```
# ... with 16 more rows
```

## Exercise: High expression in all tissues

- Create a dataset of all individual-gene pairs with high expression ( $Z > 3$ ) in all four tissues

## Exercise: High expression in all tissues

- Create a dataset of all individual-gene pairs with high expression ( $Z > 3$ ) in all four tissues

```
filter(gtex_data, Blood > 3, Heart > 3, Lung > 3, Liver > 3)
# A tibble: 16 x 7
```

	Gene	Ind	Blood	Heart	Lung	Liver	NTissues
	<chr>	<chr>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
1	AC004453.8	GTEX-12WSI	6.2	4.31	4.57	3.77	4
2	AC004453.8	GTEX-14DAQ	7.76	6.98	5.64	4.81	4
3	CTAG2	GTEX-YECK	4.73	8.37	6.11	3.11	4
4	CTD-2525I3.5	GTEX-14E1K	3.27	8.78	5.9	4.76	4
5	EMC3-AS1	GTEX-1KANB	5.33	4	4.21	4.87	4
6	GTF2IP14	GTEX-11NV4	12.2	7.26	5.79	7.06	4
7	GTF2IP14	GTEX-X3Y1	10.3	7.46	8.12	3.67	4
8	MTATP8P2	GTEX-17EVP	6.35	5.16	5.31	5.44	4
9	MTATP8P2	GTEX-YECK	4.21	4.04	3.53	4.55	4
10	PSMA6	GTEX-12WSI	4.23	3.92	5.39	3.03	4
11	RP11-1228E12.1	GTEX-1KANB	6.18	4.08	9.69	6.63	4
12	RRP7BP	GTEX-1AX9I	7.49	5.45	5.8	3.94	4
13	TMEM145	GTEX-1JMLX	3.68	7.15	7.98	7.96	4
14	TRBV23-1	GTEX-1A8FM	6.51	8.63	3.15	4.82	4
15	TRBV3-1	GTEX-1E2YA	5.04	3.64	3.34	4.64	4
16	TRMT2B	GTEX-1CB4J	6.56	5.74	5.58	5.29	4

## Exercise: High and low expression events

- Create a dataset of all high and low expression instances ( $Z > 3$  or  $< -3$ ) in any tissue (?abs may be helpful)

## Exercise: High and low expression events

- Create a dataset of all high and low expression instances ( $Z > 3$  or  $< -3$ ) in any tissue (?abs may be helpful)

```
filter(gtex_data, abs(Blood) > 3 | abs(Heart) > 3 | abs(Lung) > 3 | abs(Liver) > 3)
# A tibble: 10,171 x 7
```

	Gene	Ind	Blood	Heart	Lung	Liver	NTissues
	<chr>	<chr>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
1	A2ML1	GTEX-14E7W	0.55	-0.63	3.7	1.6	4
2	A2ML1	GTEX-1GF9V	-0.04	1.44	-0.53	3.65	4
3	A3GALT2	GTEX-11TUW	0.15	-3.41	0.96	-0.83	4
4	A3GALT2	GTEX-1BAJH	-0.7	0.91	0.16	3.61	4
5	A3GALT2	GTEX-XBEC	1.03	3.25	0.87	1.51	4
6	A3GALT2	GTEX-ZF29	-3.61	-2.02	-2.1	0.23	4
7	A3GALT2	GTEX-ZTPG	0.94	0.5	-1.23	3.22	4
8	AAMDC	GTEX-12WSG	-0.49	-3.38	-2.62	-1.67	4
9	AAMDC	GTEX-1JMLX	0.65	0.69	0.15	3.43	4
10	AANAT	GTEX-ZTPG	1.02	0.02	0.55	3.78	4

```
# ... with 10,161 more rows
```

## Exercise: getting rid of NAs

- Filter out any rows where the value for `Heart` is missing (value is `NA`)



## Exercise: getting rid of NAs

- Filter out any rows where the value for `Heart` is missing (value is NA)

```
filter(gtex_data, !is.na(Heart))
# A tibble: 383,941 x 7
   Gene   Ind      Blood Heart  Lung Liver NTissues
   <chr> <chr>    <dbl> <dbl> <dbl> <dbl>   <dbl>
1 A2ML1 GTEX-11DXZ -0.14 -1.08 NA    -0.66     3
2 A2ML1 GTEX-11GSP -0.5  0.53  0.76 -0.1     4
3 A2ML1 GTEX-11NUK -0.08 -0.4  -0.26 -0.13    4
4 A2ML1 GTEX-11NV4 -0.37  0.11 -0.42 -0.61    4
5 A2ML1 GTEX-11TT1  0.3  -1.11  0.59 -0.12    4
6 A2ML1 GTEX-11TUV  0.02 -0.47  0.29 -0.66    4
7 A2ML1 GTEX-11ZUS -1.07 -0.41  0.67  0.06    4
8 A2ML1 GTEX-11ZVC -0.27 -0.51  0.13 -0.75    4
9 A2ML1 GTEX-1212Z -0.3  0.53  0.1  -0.48    4
10 A2ML1 GTEX-12696 -0.11  0.24  0.96  0.72    4
# ... with 383,931 more rows
```

# Filtering by row number

- Use `row_number()` to filter specific rows. This is more useful once you have sorted the data in a particular order, which we will soon see how to do.

```
filter(gtex_data, row_number() <= 3)
# A tibble: 3 x 7
  Gene   Ind          Blood Heart  Lung Liver NTissues
  <chr> <chr>      <dbl> <dbl> <dbl> <dbl>    <dbl>
1 A2ML1 GTEX-11DXZ -0.14 -1.08 NA     -0.66      3
2 A2ML1 GTEX-11GSP -0.5  0.53  0.76 -0.1      4
3 A2ML1 GTEX-11NUK -0.08 -0.4  -0.26 -0.13     4
```

# Sampling rows

- You can use `sample_n()` to get `n` randomly selected rows if you don't have a particular condition you would like to filter on.

```
sample_n(gtex_data, 5)
# A tibble: 5 x 7
  Gene      Ind      Blood Heart  Lung Liver NTissues
  <chr>    <chr>    <dbl> <dbl> <dbl> <dbl>   <dbl>
1 SMARCA2  GTEX-14BIL -0.01  0.2   3.9   0.85    4
2 RP11-779018.2 GTEX-131XE -0.59 -0.48  0.74  0.29    4
3 RP11-338I21.1 GTEX-1MJIX -1.59  0.03  2.15 -1.63    4
4 GSDMB    GTEX-ZPU1  -0.05  0.23 -0.58 -2.32    4
5 ATL1     GTEX-WZTO  -1.21 -0.09 -1.59  0.97    4
```

- `sample_frac()` is similar
- Do `?sample_n()` to see how you can sample with replacement or with weights



# Arrange rows with arrange()

- `arrange()` takes a data frame and a column, and sorts the rows by the values in that column (ascending order).
- Again, the first argument is the data frame and the other arguments tell the function what to do with it

```
arrange(gtex_data, Blood)
# A tibble: 389,922 x 7
  Gene      Ind      Blood Heart  Lung Liver NTissues
  <chr>    <chr>    <dbl> <dbl> <dbl> <dbl>    <dbl>
1 HBA2     GTEX-11DXZ -9.44  -1.52  -1.44  -2.15     4
2 MTATP6P1 GTEX-1KD5A -9.18 -10.1  -10.3  -9.52     4
3 RP11-46D6.1 GTEX-14E1K -7.83  -3.94  -5.22  -4.49     4
4 CYTH3     GTEX-11NV4 -6.63  -0.6   -0.37  -1.32     4
5 TRG-AS1   GTEX-11NV4 -6.47   2.39  -0.6   -0.22     4
6 SMG1P1    GTEX-11ZUS -6.26  -1.68  -1.41  -0.31     4
7 ZBTB10    GTEX-VUSG  -6.13   0.77   0.51  -0.67     4
8 RPS29     GTEX-1B8L1 -5.84  -0.8   -0.46  -0.17     4
9 GHITM     GTEX-WK11  -5.7   -7.24  -7.37  -4.06     4
10 ZNF2      GTEX-VUSG  -5.62   1.52   0.61   0.13     4
# ... with 389,912 more rows
```

# Arrange can sort by more than one column

- This is useful if there is a tie in sorting by the first column.

```
arrange(gtex_data, NTissues, Blood)
# A tibble: 389,922 x 7
  Gene      Ind      Blood Heart  Lung  Liver NTissues
  <chr>    <chr>    <dbl> <dbl> <dbl> <dbl>   <dbl>
1 HEATR1   GTEX-1EWIQ -1.63  NA    NA    0.49    2
2 FOXO1    GTEX-1BAJH -1.58  NA    NA   -0.25    2
3 UCN      GTEX-12WSI -1.57  NA    NA   -0.48    2
4 GPR171   GTEX-132NY -1.53  NA    NA   -1.03    2
5 UCN      GTEX-WFON  -1.46  NA    NA   -0.15    2
6 KIAA1614 GTEX-12WSI -1.35  NA    NA   -0.46    2
7 ENTPD1-AS1 GTEX-11NUK -1.28  NA    NA   -0.54    2
8 TOP3B    GTEX-1A32A -1.28  NA    NA   -0.76    2
9 AOA      GTEX-11GSP -1.27  NA    NA    0.41    2
10 PRRX2    GTEX-1A8FM -1.13  -1.2  NA   NA      2
# ... with 389,912 more rows
```

## Use the desc function to arrange by descending values

```
arrange(gtex_data, desc(Blood))  
# A tibble: 389,922 x 7  
  Gene      Ind      Blood Heart  Lung Liver NTissues  
  <chr>    <chr>    <dbl> <dbl> <dbl> <dbl>    <dbl>  
1 REN      GTEX-U8XE  18.9 -0.57 NA      0.09      3  
2 KLK3     GTEX-147F4 15.7 -0.74 -0.44 -0.02      4  
3 DNASE2B  GTEX-12696 14.4 -0.82 -0.92  0.35      4  
4 GAPDHP33 GTEX-UPK5   13.8  1.52 -1.48 -1.84      4  
5 DCSTAMP  GTEX-12696 13.6 NA    -0.57 -0.91      3  
6 AC012358.7 GTEX-VUSG  13.6 -1.43  1.22 -0.39      4  
7 KCNT1    GTEX-1KANB  13.5  3.14  0.62 -0.37      4  
8 FFAR4    GTEX-12696 12.9 -0.96 -0.67  0.18      4  
9 NAPSA    GTEX-1CB4J  12.3 -0.29 -0.44 -0.14      4  
10 DIAPH2-AS1 GTEX-VUSG  12.2 -0.33  1.18  0.67      4  
# ... with 389,912 more rows
```

## Exercise: top 5 high expression instances

Use `arrange()` and `filter()` to get the data for the 5 individual-gene pairs with the most extreme expression changes in blood



## Exercise: top 5 high expression instances

Use `arrange()` and `filter()` to get the data for the 5 individual-gene pairs with the most extreme expression changes in blood

```
filter(arrange(gt看_data, desc(abs(Blood))), row_number()<=5) # "nesting" the  
calls to filter and arrange  
# A tibble: 5 x 7  
  Gene      Ind      Blood Heart  Lung Liver NTissues  
  <chr>   <chr>   <dbl> <dbl> <dbl> <dbl>   <dbl>  
1 REN     GTEX-U8XE 18.9 -0.57 NA      0.09     3  
2 KLK3    GTEX-147F4 15.7 -0.74 -0.44 -0.02     4  
3 DNASE2B GTEX-12696 14.4 -0.82 -0.92 0.35     4  
4 GAPDHP33 GTEX-UPK5 13.8 1.52 -1.48 -1.84     4  
5 DCSTAMP GTEX-12696 13.6 NA      -0.57 -0.91     3
```

## Exercise: top 5 high expression instances

Use `arrange()` and `filter()` to get the data for the 5 individual-gene pairs with the most extreme expression changes in blood

```
filter(arrange(gtex_data, desc(abs(Blood))), row_number()<=5) # "nesting" the
calls to filter and arrange
# A tibble: 5 x 7
  Gene      Ind      Blood Heart  Lung Liver NTissues
  <chr>    <chr>    <dbl> <dbl> <dbl> <dbl>    <dbl>
1 REN      GTEX-U8XE  18.9 -0.57 NA      0.09      3
2 KLK3     GTEX-147F4 15.7 -0.74 -0.44 -0.02      4
3 DNASE2B  GTEX-12696 14.4 -0.82 -0.92  0.35      4
4 GAPDHP33 GTEX-UPK5  13.8  1.52 -1.48 -1.84      4
5 DCSTAMP  GTEX-12696 13.6 NA    -0.57 -0.91      3
```

or

```
gtex_by_blood = arrange(gtex_data, desc(abs(Blood))) # using a temporary variable
filter(gtex_by_blood, row_number()<=5)
# A tibble: 5 x 7
  Gene      Ind      Blood Heart  Lung Liver NTissues
  <chr>    <chr>    <dbl> <dbl> <dbl> <dbl>    <dbl>
1 REN      GTEX-U8XE  18.9 -0.57 NA      0.09      3
2 KLK3     GTEX-147F4 15.7 -0.74 -0.44 -0.02      4
3 DNASE2B  GTEX-12696 14.4 -0.82 -0.92  0.35      4
4 GAPDHP33 GTEX-UPK5  13.8  1.52 -1.48 -1.84      4
5 DCSTAMP  GTEX-12696 13.6 NA    -0.57 -0.91      3
```



## Select columns with select()

```
select(gtex_data, Gene, Ind, Blood)
# A tibble: 389,922 x 3
   Gene   Ind      Blood
   <chr> <chr>    <dbl>
1 A2ML1 GTEX-11DXZ -0.14
2 A2ML1 GTEX-11GSP -0.5
3 A2ML1 GTEX-11NUK -0.08
4 A2ML1 GTEX-11NV4 -0.37
5 A2ML1 GTEX-11TT1 0.3
6 A2ML1 GTEX-11TUV 0.02
7 A2ML1 GTEX-11ZUS -1.07
8 A2ML1 GTEX-11ZVC -0.27
9 A2ML1 GTEX-1212Z -0.3
10 A2ML1 GTEX-12696 -0.11
# ... with 389,912 more rows
```

- The select function will return a subset of the tibble, using only the requested columns in the order specified.

# Select columns with select()

- `select()` can also be used with handy helpers like `starts_with()` and `contains()`

```
select(gtex_data, starts_with("L"))  
# A tibble: 389,922 x 2  
  Lung Liver  
  <dbl> <dbl>  
1 NA    -0.66  
2  0.76 -0.1  
3 -0.26 -0.13  
4 -0.42 -0.61  
5  0.59 -0.12  
6  0.29 -0.66  
7  0.67  0.06  
8  0.13 -0.75  
9  0.1   -0.48  
10 0.96  0.72  
# ... with 389,912 more rows
```

- Use `?select` to see all the possibilities

## Select columns with select()

```
select(gtex_data, contains("N"))
# A tibble: 389,922 x 4
  Gene   Ind      Lung NTissues
  <chr> <chr>    <dbl>    <dbl>
1 A2ML1 GTEX-11DXZ NA         3
2 A2ML1 GTEX-11GSP  0.76       4
3 A2ML1 GTEX-11NUK -0.26       4
4 A2ML1 GTEX-11NV4 -0.42       4
5 A2ML1 GTEX-11TT1  0.59       4
6 A2ML1 GTEX-11TUV  0.29       4
7 A2ML1 GTEX-11ZUS  0.67       4
8 A2ML1 GTEX-11ZVC  0.13       4
9 A2ML1 GTEX-1212Z  0.1        4
10 A2ML1 GTEX-12696  0.96       4
# ... with 389,912 more rows
```

- The quotes around the letter "N" make it a string. If we did not do this, R would think it was looking for a variable called N and not just the plain letter.
- We don't have to quote the names of columns (like Ind) because the tidyverse functions know that we are working within the dataframe and thus treat the column names like they are variables in their own right

# select() subsets columns by name

- select() can also be used to select everything **except for** certain columns

```
select(gtex_data, -starts_with("L"), -Ind)
# A tibble: 389,922 x 4
  Gene   Blood Heart NTissues
  <chr> <dbl> <dbl>    <dbl>
1 A2ML1 -0.14 -1.08      3
2 A2ML1 -0.5  0.53      4
3 A2ML1 -0.08 -0.4      4
4 A2ML1 -0.37 0.11      4
5 A2ML1  0.3 -1.11      4
6 A2ML1  0.02 -0.47      4
7 A2ML1 -1.07 -0.41      4
8 A2ML1 -0.27 -0.51      4
9 A2ML1 -0.3  0.53      4
10 A2ML1 -0.11 0.24      4
# ... with 389,912 more rows
```

# select() subsets columns by name

- or even to select only columns that match a certain condition

```
select(gtex_data, where(is.numeric))  
# A tibble: 389,922 x 5  
  Blood Heart  Lung Liver NTissues  
  <dbl> <dbl> <dbl> <dbl>    <dbl>  
1 -0.14 -1.08 NA      -0.66      3  
2 -0.5   0.53  0.76 -0.1       4  
3 -0.08 -0.4   -0.26 -0.13      4  
4 -0.37  0.11 -0.42 -0.61      4  
5  0.3   -1.11  0.59 -0.12      4  
6  0.02 -0.47  0.29 -0.66      4  
7 -1.07 -0.41  0.67  0.06      4  
8 -0.27 -0.51  0.13 -0.75      4  
9 -0.3   0.53  0.1   -0.48      4  
10 -0.11  0.24  0.96  0.72      4  
# ... with 389,912 more rows
```



# pull() is a friend of select()

- `select()` has a friend called `pull()` which returns a vector instead of a (one-column) data frame

```
select(gtex_data, Gene)
# A tibble: 389,922 x 1
  Gene
  <chr>
1 A2ML1
2 A2ML1
3 A2ML1
4 A2ML1
5 A2ML1
6 A2ML1
7 A2ML1
8 A2ML1
9 A2ML1
10 A2ML1
# ... with 389,912 more rows
```

```
pull(gtex_data, Gene)
[1] "A2ML1"      "A2ML1"      "A2ML1"
[4] "A2ML1"      "A2ML1"      "A2ML1"
[7] "A2ML1"      "A2ML1"      "A2ML1"
[10] "A2ML1"      "A2ML1"      "A2ML1"
[13] "A2ML1"      "A2ML1"      "A2ML1"
...
```

# Rename column names with rename()

- `select()` can be used to rename variables, but it drops all variables not selected

```
select(gtex_data, number_tissues = NTissues)
# A tibble: 389,922 x 1
  number_tissues
      <dbl>
1             3
2             4
3             4
...
```

- `rename()` is better suited for this because it keeps all the columns

```
rename(gtex_data, number_tissues = NTissues)
# A tibble: 389,922 x 7
  Gene   Ind          Blood Heart  Lung Liver number_tissues
  <chr> <chr>      <dbl> <dbl> <dbl> <dbl>      <dbl>
1 A2ML1 GTEX-11DXZ -0.14 -1.08 NA    -0.66          3
2 A2ML1 GTEX-11GSP -0.5   0.53  0.76 -0.1           4
3 A2ML1 GTEX-11NUK -0.08 -0.4  -0.26 -0.13          4
...
```

**Note:** `mutate()`, can also change a column name (more on `mutate()` soon)

## Exercise: select and filter

- Create a one-column dataframe of the heart expression Z-scores (`Heart`) of all individuals with data present (i.e. not `NA`) for gene `WDR34` (`Gene`) in the `gtex_data` dataset.

## Exercise: select and filter

- Create a one-column dataframe of the heart expression Z-scores (`Heart`) of all individuals with data present (i.e. not NA) for gene `WDR34` (`Gene`) in the `gtex_data` dataset.

```
select(filter(gtex_data, Gene == "WDR34", !is.na(Heart)), Heart)
# A tibble: 78 x 1
  Heart
  <dbl>
1  0.78
2  0.94
3  0.29
4  0.17
5 -0.6
6  0.46
7 -0.97
...

```

## Exercise: select and filter

- Create a one-column dataframe of the heart expression Z-scores (`Heart`) of all individuals with data present (i.e. not NA) for gene `WDR34` (`Gene`) in the `gtex_data` dataset.

```
select(filter(gtex_data, Gene == "WDR34", !is.na(Heart)), Heart)
# A tibble: 78 x 1
  Heart
  <dbl>
1  0.78
2  0.94
3  0.29
4  0.17
5 -0.6
6  0.46
7 -0.97
...

```

- What is wrong with this?

```
filter(select(gtex_data, Heart), Gene == "WDR34")
```

## Exercise: select text columns

- Use `select` to subset the `gtex_data` dataframe to just those columns that contain text data.
- Can you do this programmatically without specifying the names of each of the desired columns?
- Which base R function will help you determine if a column is textual or not? Use whatever tools you want to find out.

## Exercise: select text columns

- Use `select` to subset the `gtex_data` dataframe to just those columns that contain text data.
- Can you do this programmatically without specifying the names of each of the desired columns?
- Which base R function will help you determine if a column is textual or not? Use whatever tools you want to find out.

```
select(gtex_data, where(is.character))  
# A tibble: 389,922 x 2  
  Gene   Ind  
  <chr> <chr>  
1 A2ML1 GTEX-11DXZ  
2 A2ML1 GTEX-11GSP  
3 A2ML1 GTEX-11NUK  
4 A2ML1 GTEX-11NV4  
5 A2ML1 GTEX-11TT1  
6 A2ML1 GTEX-11TUV  
7 A2ML1 GTEX-11ZUS  
8 A2ML1 GTEX-11ZVC  
9 A2ML1 GTEX-1212Z  
10 A2ML1 GTEX-12696  
# ... with 389,912 more rows
```





## Add new variables with mutate()

```
mutate(gtex_data, abs_blood = abs(Blood))
# A tibble: 389,922 x 8
  Gene   Ind      Blood Heart  Lung Liver NTissues abs_blood
  <chr> <chr>    <dbl> <dbl> <dbl> <dbl>   <dbl>    <dbl>
1 A2ML1 GTEX-11DXZ -0.14 -1.08 NA     -0.66     3      0.14
2 A2ML1 GTEX-11GSP -0.5  0.53  0.76 -0.1     4      0.5
...
```

- This uses `mutate()` to add a new column to which is the absolute value of `Blood`.
- The thing on the left of the `=` is a new name that you make up which you would like the new column to be called
- The expression on the right of the `=` defines what will go into the new column
- **Warning!** If the new variable name already exists, `mutate()` will overwrite the existing one

```
mutate(gtex_data, Blood = Blood *1000)
# A tibble: 389,922 x 7
  Gene   Ind      Blood Heart  Lung Liver NTissues
  <chr> <chr>    <dbl> <dbl> <dbl> <dbl>   <dbl>
1 A2ML1 GTEX-11DXZ -140 -1.08 NA     -0.66     3
2 A2ML1 GTEX-11GSP -500  0.53  0.76 -0.1     4
...
```

# mutate() can create multiple new columns at once

- `mutate()` can create multiple columns at the same time and use multiple columns to define a single new one

```
mutate(gtex_data, # the newlines make it more readable
      abs_blood = abs(Blood),
      abs_heart = abs(Heart),
      blood_heart_dif = abs_blood - abs_heart
    )
# A tibble: 389,922 x 10
   Gene   Ind      Blood Heart  Lung Liver NTissues abs_blood abs_heart
   <chr> <chr>    <dbl> <dbl> <dbl> <dbl>   <dbl>    <dbl>    <dbl>
1 A2ML1 GTEX-11DXZ -0.14 -1.08 NA    -0.66     3      0.14     1.08
2 A2ML1 GTEX-11GSP -0.5  0.53  0.76 -0.1     4      0.5      0.53
3 A2ML1 GTEX-11NUK -0.08 -0.4  -0.26 -0.13    4      0.08     0.4
4 A2ML1 GTEX-11NV4 -0.37 0.11 -0.42 -0.61    4      0.37     0.11
5 A2ML1 GTEX-11TT1  0.3  -1.11  0.59 -0.12    4      0.3      1.11
6 A2ML1 GTEX-11TUV  0.02 -0.47  0.29 -0.66    4      0.02     0.47
7 A2ML1 GTEX-11ZUS -1.07 -0.41  0.67  0.06    4      1.07     0.41
8 A2ML1 GTEX-11ZVC -0.27 -0.51  0.13 -0.75    4      0.27     0.51
9 A2ML1 GTEX-1212Z -0.3  0.53  0.1  -0.48    4      0.3      0.53
10 A2ML1 GTEX-12696 -0.11 0.24  0.96  0.72    4      0.11     0.24
# ... with 389,912 more rows, and 1 more variable: blood_heart_dif <dbl>
```

- Note that we have also used two columns simultaneously (`Blood` and `Heart`) to create a new column)

# mutate() for data type conversion

- Data is sometimes given to you in a form that makes it difficult to do operations on

```
df = tibble(number = c("1", "2", "3"))
df
# A tibble: 3 x 1
  number
  <chr>
1 1
2 2
3 3
mutate(df, number_plus_1 = number + 1)
Error: Problem with `mutate()` column `number_plus_1`.
i `number_plus_1 = number + 1`.
x non-numeric argument to binary operator
```

- mutate() is also useful for converting data types, in this case text to numbers

```
mutate(df, number = as.numeric(number))
# A tibble: 3 x 1
  number
  <dbl>
1 1
2 2
3 3
```

## Exercise: mutate()

I want to identify genes that have large average expression changes across blood and liver. Can you compute the average of blood and liver expression changes across all gene-individual pairs? Compute the average manually (i.e. don't use the mean function).

## Exercise: mutate()

I want to identify genes that have large average expression changes across blood and liver. Can you compute the average of blood and liver expression changes across all gene-individual pairs? Compute the average manually (i.e. don't use the mean function).

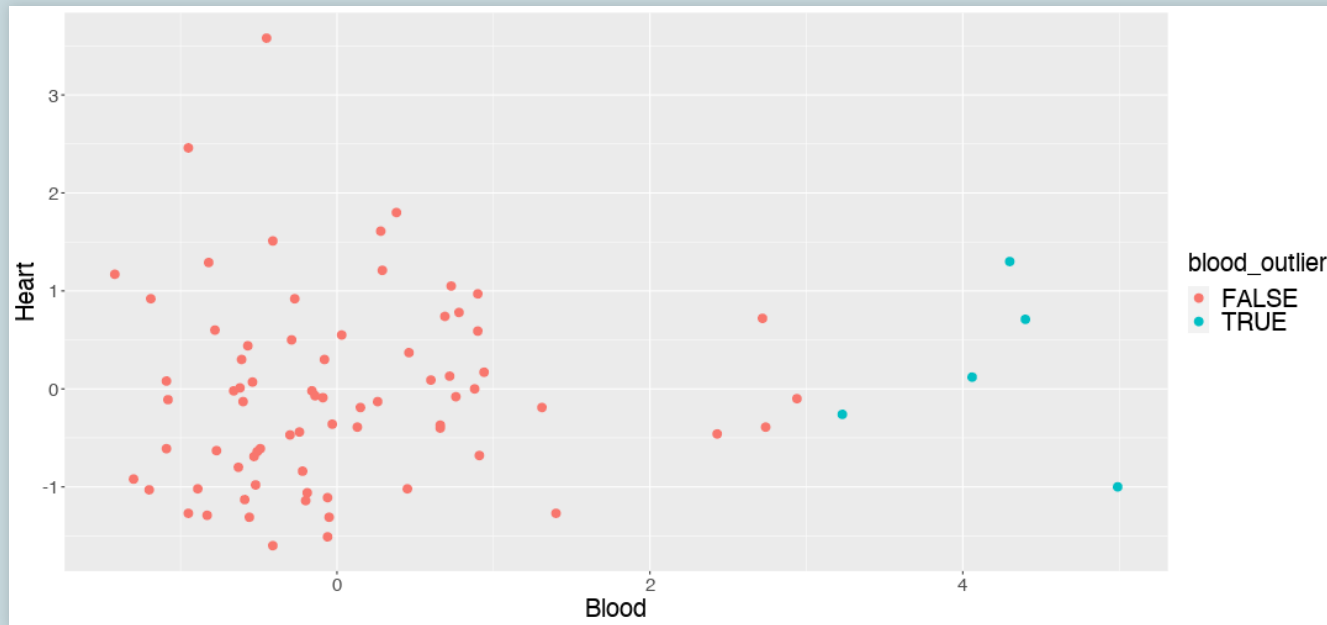
```
mutate(gtex_data, avg_blood_liver = (Blood+Liver)/2)
# A tibble: 389,922 x 8
  Gene   Ind      Blood Heart  Lung Liver NTissues avg_blood_liver
  <chr> <chr>    <dbl> <dbl> <dbl> <dbl>   <dbl>         <dbl>
1 A2ML1 GTEX-11DXZ -0.14 -1.08 NA    -0.66     3          -0.4
2 A2ML1 GTEX-11GSP -0.5  0.53  0.76 -0.1      4          -0.3
3 A2ML1 GTEX-11NUK -0.08 -0.4  -0.26 -0.13     4         -0.105
4 A2ML1 GTEX-11NV4 -0.37  0.11 -0.42 -0.61     4         -0.49
5 A2ML1 GTEX-11TT1  0.3  -1.11  0.59 -0.12     4           0.09
6 A2ML1 GTEX-11TUV  0.02 -0.47  0.29 -0.66     4         -0.32
7 A2ML1 GTEX-11ZUS -1.07 -0.41  0.67  0.06     4        -0.505
8 A2ML1 GTEX-11ZVC -0.27 -0.51  0.13 -0.75     4         -0.51
9 A2ML1 GTEX-1212Z -0.3  0.53  0.1  -0.48     4         -0.39
10 A2ML1 GTEX-12696 -0.11  0.24  0.96  0.72     4          0.305
# ... with 389,912 more rows
```

## Exercise: mutate() and ggplot

Filter `gtex_data` to only include measurements of the MYL1 gene. Then, use `mutate` to mark which gene-individual pairs have outlier MYL1 expression in blood, defined as  $Z > 3$  or  $Z < -3$ . Then, produce a plot showing blood Z-scores vs heart Z-scores and color the blood gene expression outliers in a different color than the other points.

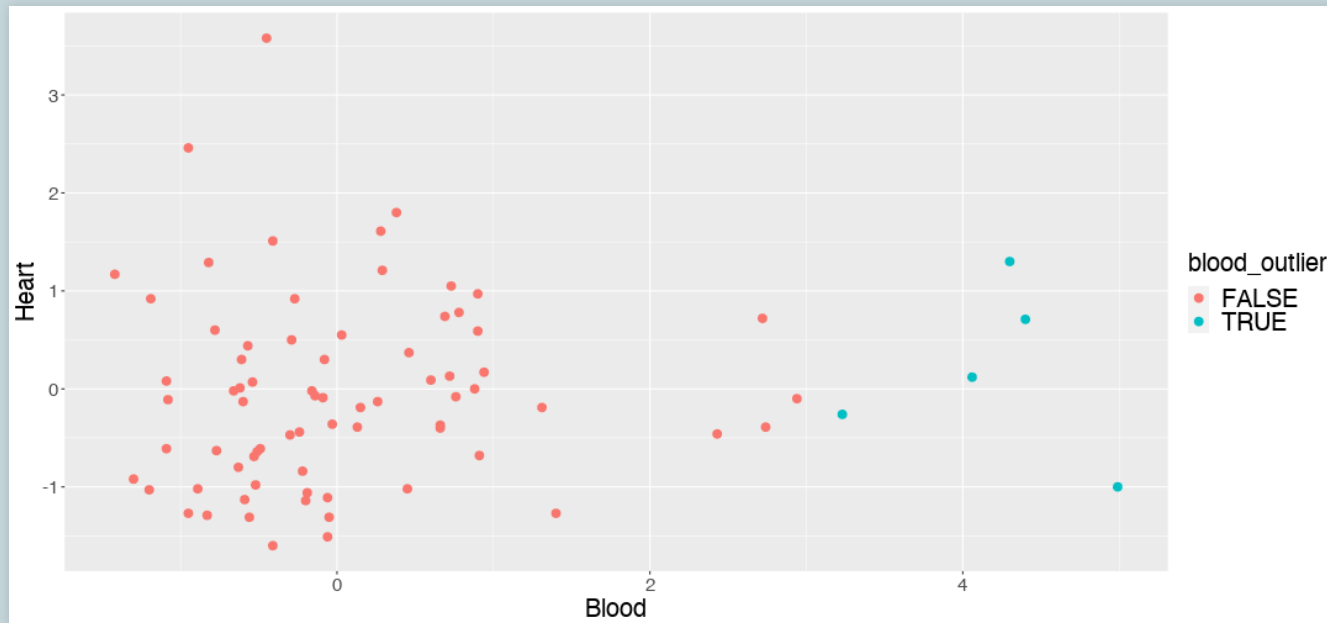
## Exercise: mutate() and ggplot

Filter `gtex_data` to only include measurements of the MYL1 gene. Then, use `mutate` to mark which gene-individual pairs have outlier MYL1 expression in blood, defined as  $Z > 3$  or  $Z < -3$ . Then, produce a plot showing blood Z-scores vs heart Z-scores and color the blood gene expression outliers in a different color than the other points.



## Exercise: mutate() and ggplot

Filter `gtex_data` to only include measurements of the MYL1 gene. Then, use `mutate` to mark which gene-individual pairs have outlier MYL1 expression in blood, defined as  $Z > 3$  or  $Z < -3$ . Then, produce a plot showing blood Z-scores vs heart Z-scores and color the blood gene expression outliers in a different color than the other points.



```
gene_data = filter(gtex_data, Gene == 'MYL1')
blood_outliers = mutate(gene_data, blood_outlier = abs(Blood)>3)
ggplot(blood_outliers) +
  geom_point(aes(x=Blood, y=Heart, color=blood_outlier))
```



## Exercise: putting it together

I am interested in identifying individuals that have a large change in gene expression change for any gene between lung tissue and blood tissue, with higher expression in lung.

1. Produce a list of top 10 individual-gene pairs arranged by the expression change for lung compared to blood
2. Only consider individual-gene pairs measured in all four tissues
3. In the output, just show the gene, individual, lung expression, blood expression, and lung-blood differences

## Exercise: putting it together

I am interested in identifying individuals that have a large change in gene expression change for any gene between lung tissue and blood tissue, with higher expression in lung.

1. Produce a list of top 10 individual-gene pairs arranged by the expression change for lung compared to blood
2. Only consider individual-gene pairs measured in all four tissues
3. In the output, just show the gene, individual, lung expression, blood expression, and lung-blood differences

```
gtex_data_no_change = filter(gtex_data, NTissues == 4)
gtex_data_ratio = mutate(gtex_data_no_change, lung_blood_dif = Lung - Blood)
sorted = arrange(gtex_data_ratio, desc(lung_blood_dif))
top_10 = filter(sorted, row_number() <= 10)
select(top_10, Gene, Ind, Lung, Blood, lung_blood_dif)
# A tibble: 10 x 5
```

	Gene <chr>	Ind <chr>	Lung <dbl>	Blood <dbl>	lung_blood_dif <dbl>
1	TMEM151A	GTEX-ZEX8	12.0	-0.77	12.8
2	DSG3	GTEX-14PJO	10.3	-0.04	10.4
3	KNG1	GTEX-12WSD	11.6	1.74	9.82
4	NPIPA8	GTEX-XXEK	9.23	0.03	9.2
5	ALOXE3	GTEX-YFC4	7.5	-1.32	8.82
6	ETNPPL	GTEX-1CB4J	8.47	-0.31	8.78
7	EGR4	GTEX-13FTZ	7.79	-0.75	8.54
8	IGHV2-26	GTEX-YECK	7.82	-0.63	8.45
9	LINC00342	GTEX-1A32A	6.01	-2.16	8.17
10	RP11-1260E13.4	GTEX-11ZVC	6.96	-1.17	8.13



# Why pipe?

- In our last exercise, we used a number of different function applications to arrive at our answer. Shown below, we used temporary variables to keep our code clean.

```
gtex_data_no_change = filter(gtex_data, NTissues == 4)
gtex_data_ratio = mutate(gtex_data_no_change, lung_blood_dif = Lung - Blood)
sorted = arrange(gtex_data_ratio, desc(lung_blood_dif))
top_10 = filter(sorted, row_number() <= 10)
select(top_10, Gene, Ind, Lung, Blood, lung_blood_dif)
```

# Why pipe?

- In our last exercise, we used a number of different function applications to arrive at our answer. Shown below, we used temporary variables to keep our code clean.

```
gtex_data_no_change = filter(gtex_data, NTissues == 4)
gtex_data_ratio = mutate(gtex_data_no_change, lung_blood_dif = Lung - Blood)
sorted = arrange(gtex_data_ratio, desc(lung_blood_dif))
top_10 = filter(sorted, row_number() <= 10)
select(top_10, Gene, Ind, Lung, Blood, lung_blood_dif)
```

- Compare that to the same code using nested calls (instead of storing in temporary variables):

```
select(
  filter(
    arrange(
      mutate(
        filter(
          gtex_data, NTissues == 4,
          lung_blood_dif = Lung - Blood),
        desc(lung_blood_dif)),
      row_number() <= 10),
    Gene, Ind, Lung, Blood, lung_blood_dif
)
```

- What makes either of these hard to read or understand?

# The pipe operator

- Tidyverse solves these problems with the pipe operator `%>%`

```
gtex_data %>%  
  filter(NTissues == 4) %>%  
  mutate(lung_blood_dif = Lung - Blood) %>%  
  arrange(desc(lung_blood_dif)) %>%  
  filter(row_number() <= 10) %>%  
  select(Gene, Ind, Lung, Blood, lung_blood_dif)
```

# The pipe operator

- Tidyverse solves these problems with the pipe operator `%>%`

```
gtex_data %>%  
  filter(NTissues == 4) %>%  
  mutate(lung_blood_dif = Lung - Blood) %>%  
  arrange(desc(lung_blood_dif)) %>%  
  filter(row_number() <= 10) %>%  
  select(Gene, Ind, Lung, Blood, lung_blood_dif)
```

- How does this compare with our code before? What do you notice?

```
gtex_data_no_change = filter(gtex_data, NTissues == 4)  
gtex_data_ratio = mutate(gtex_data_no_change, lung_blood_dif = Lung - Blood)  
sorted = arrange(gtex_data_ratio, desc(lung_blood_dif))  
top_10 = filter(sorted, row_number() <= 10)  
select(top_10, Gene, Ind, Lung, Blood, lung_blood_dif)
```

# Pipe details: What happens to an object when it gets "piped in"?

When `df1` is piped into `fun(x)` (`fun` is just some fake function)

```
df1 %>% fun(x)
```

is converted into:

```
fun(df1, x)
```

- That is: the thing being piped in is used as the *first* argument of `fun`.
- The tidyverse functions are consistently designed so that the first argument is a data frame, and the result is a data frame, so you can push a dataframe all the way through a series of functions



# Pipe details: What objects can be piped?

- The pipe works for all variables and functions (not just tidyverse functions)

## Piping with an array

```
c(1, 44, 21, 0, -4) %>%  
  sum() # instead of sum(c(1, 44, 21, 0, -4))  
[1] 62
```

## Piping with a scalar

```
1 %>% `+`(1) # `+` is just a function that takes two arguments!  
[1] 2
```

## Piping with a data frame

```
tibble(name = c("Petunia", "Rose", "Daisy", "Marigold", "Arabidopsis"),  
       age = c(10, 54, 21, 99, 96)) %>%  
  filter(age > 30)  
# A tibble: 3 x 2  
  name      age  
  <chr>    <dbl>  
1 Rose      54  
2 Marigold  99  
3 Arabidopsis 96
```

# Piping to another position

- The pipe typically pipes into the first argument of a function, but you can use `.` to represent the object you're piping into the function

```
# install.packages("slider")
library(slider)
mean %>%
  slide_vec(1:10, ., .before=2)
[1] 1.0 1.5 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0
```

- Also notice how I've piped in a *function* to a function! (yes, functions are just objects like anything else in R)
- More about this in the functional programming section

## Exercise: Pipe to ggplot

- Run this code to see what it does. Then rewrite it using the pipe operator and get it to produce the same output.

```
gene_data = filter(gtex_data, Gene == 'MYBL2')
outliers = mutate(gene_data, blood_outlier = abs(Blood) > 2)
ggplot(outliers) +
  geom_bar(aes(x=blood_outlier)) +
  scale_x_discrete("Class", labels=c("Other", "Outlier")) +
  ggtitle("How many individuals have outlier MYBL2 expression in blood?")
```

## Exercise: Pipe to ggplot

- Run this code to see what it does. Then rewrite it using the pipe operator and get it to produce the same output.

```
gene_data = filter(gtex_data, Gene == 'MYBL2')
outliers = mutate(gene_data, blood_outlier = abs(Blood) > 2)
ggplot(outliers) +
  geom_bar(aes(x=blood_outlier)) +
  scale_x_discrete("Class", labels=c("Other", "Outlier")) +
  ggtitle("How many individuals have outlier MYBL2 expression in blood?")
```

```
gtex_data %>%
  filter(Gene == 'MYBL2') %>%
  mutate(blood_outlier = abs(Blood) > 2) %>%
  ggplot() +
  geom_bar(aes(x=blood_outlier)) +
  scale_x_discrete("Class", labels=c("Other", "Outlier")) +
  ggtitle("How many individuals have outlier MYBL2 expression in blood?")
```



## Grouped summaries with summarise()

```
summarize(gtex_data, tissue_avg=mean(NTissues))  
# A tibble: 1 x 1  
  tissue_avg  
    <dbl>  
1      3.97
```

- `summarize()` boils down the data frame according to the conditions it gets. In this case, it creates a data frame with a single column called `tissue_avg` that contains the mean of the `NTissues` column
- As with `mutate()`, the name on the left of the `=` is something you make up that you would like the new column to be named.
- `mutate()` transforms columns into new columns of the same length, but `summarize()` collapses down the data frame into a single row
- Summaries are more useful when you apply them to subgroups of the data, which we will soon see how to do.

# Grouped summaries with summarise()

- note that you can also pass in multiple conditions that operate on multiple columns at the same time

```
gtex_data %>%
  summarize( # newlines not necessary, again just increase clarity
    tissue_avg = mean(NTissues),
    blood_max = max(Blood, na.rm=T),
    blood_lung_dif_min = min(Blood - Lung, na.rm=T)
  )
# A tibble: 1 x 3
  tissue_avg blood_max blood_lung_dif_min
    <dbl>      <dbl>      <dbl>
1     3.97     18.9      -12.8
```

# Grouped summaries with summarise()

- Summaries are more useful when you apply them to subgroups of the data

```
gtex_data %>%  
  group_by(Gene) %>%  
  summarize(max_blood = max(Blood))  
# A tibble: 4,999 x 2  
  Gene                max_blood  
  <chr>              <dbl>  
1 A2ML1                2.08  
2 A3GALT2              2.77  
3 A4GALT              2.78  
4 AAMDC                NA  
5 AANAT                1.71  
6 AAR2                2.52  
7 AARSD1              1.89  
8 AB019441.29         2.31  
9 ABC7-42389800N19.1  1.98  
10 ABCA5               2.3  
# ... with 4,989 more rows
```



## Multiple columns can be used to group the data simultaneously

```
gtex_data %>%  
  group_by(Gene, Ind) %>%  
  summarize(max_blood = max(Blood))  
# A tibble: 389,922 x 3  
# Groups:   Gene [4,999]  
  Gene   Ind      max_blood  
  <chr> <chr>      <dbl>  
1 A2ML1 GTEX-11DXZ    -0.14  
2 A2ML1 GTEX-11GSP    -0.5  
3 A2ML1 GTEX-11NUK    -0.08  
4 A2ML1 GTEX-11NV4    -0.37  
5 A2ML1 GTEX-11TT1     0.3  
6 A2ML1 GTEX-11TUV     0.02  
7 A2ML1 GTEX-11ZUS    -1.07  
8 A2ML1 GTEX-11ZVC    -0.27  
9 A2ML1 GTEX-1212Z    -0.3  
10 A2ML1 GTEX-12696   -0.11  
# ... with 389,912 more rows
```

- The result has the summary value for each unique combination of the grouping variables

# Computing the number of rows in each group

- The `n()` function counts the number of rows in each group:

```
gtex_data %>%
  filter(!is.na(Blood)) %>%
  group_by(Gene) %>%
  summarize(how_many = n())
# A tibble: 4,999 x 2
   Gene                how_many
   <chr>                <int>
1 A2ML1                  78
2 A3GALT2                78
3 A4GALT                78
4 AAMDC                  77
5 AANAT                  78
6 AAR2                   78
7 AARSD1                 78
8 AB019441.29            78
9 ABC7-42389800N19.1     78
10 ABCA5                  78
# ... with 4,989 more rows
```

- You can also use `count()`, which is just a shorthand for the same thing

```
gtex_data %>%
  filter(!is.na(Blood)) %>%
  group_by(Gene) %>%
  count()
```

# Computing the number of distinct elements in a column, per group

- `n_distinct()` counts the number of unique elements in a column

```
gtex_data %>%
  group_by(Ind) %>%
  summarize(n_genes = n_distinct(Gene))
# A tibble: 78 x 2
  Ind          n_genes
  <chr>        <int>
1 GTEX-11DXZ    4999
2 GTEX-11GSP    4999
3 GTEX-11NUK    4999
4 GTEX-11NV4    4999
5 GTEX-11TT1    4999
6 GTEX-11TUV    4999
7 GTEX-11ZUS    4999
8 GTEX-11ZVC    4999
9 GTEX-1212Z    4999
10 GTEX-12696    4999
# ... with 68 more rows
```

## Exercise: top expression per tissue

- Ignoring NAs, what is the highest liver expression value seen for each gene in the `gtex_data` dataset?
- What about the lowest?

## Exercise: top expression per tissue

- Ignoring NAs, what is the highest liver expression value seen for each gene in the `gtex_data` dataset?
- What about the lowest?

```
gtex_data %>%
  group_by(Gene) %>%
  summarize(
    max_liver = max(Liver, na.rm=T),
    min_liver = min(Liver, na.rm=T)
  )
# A tibble: 4,999 x 3
  Gene                max_liver min_liver
  <chr>                <dbl>    <dbl>
1 A2ML1                3.65     -1.94
2 A3GALT2              3.61     -1.3
3 A4GALT              2.22     -1.76
4 AAMDC               3.43     -2.62
5 AANAT               3.78     -2.22
6 AAR2               2.32     -3.23
7 AARSD1             2.77     -2.75
8 AB019441.29        3.36     -1.53
9 ABC7-42389800N19.1 2.51     -3.2
10 ABCA5              3.27     -3.27
# ... with 4,989 more rows
```

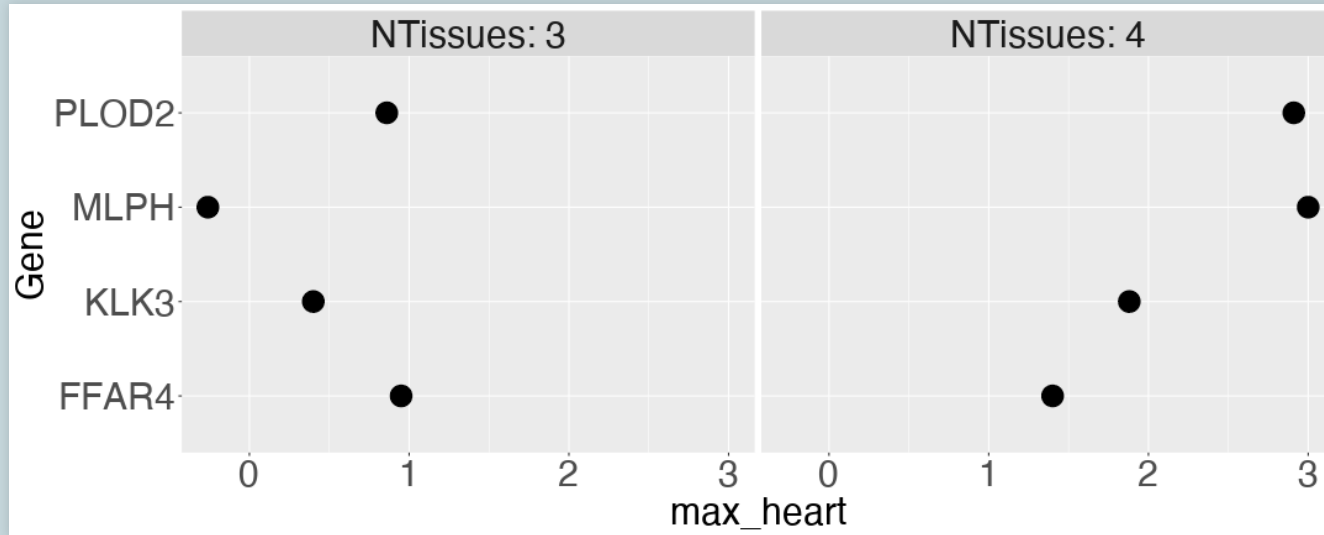
# Exercise: summarize and plot

Recreate this plot.



# Exercise: summarize and plot

Recreate this plot.



```
gtex_data %>%  
  filter(Gene %in% c('FFAR4', 'KLK3', 'PLOD2', 'MLPH')) %>%  
  group_by(Gene, NTissues) %>%  
  summarize(max_heart = max(Heart ,na.rm=T)) %>%  
  
ggplot() +  
  geom_point(aes(y=Gene, x=max_heart)) +  
  facet_grid(. ~ NTissues , labeller = label_both)
```





# Filtering grouped data

- `filter()` is aware of grouping. When used on a grouped dataset, it applies the filtering condition separately in each group

```
gtex_data %>%
  group_by(Gene) %>%
  filter(NTissues == max(NTissues))
# A tibble: 376,883 x 7
# Groups:   Gene [4,999]
  Gene   Ind      Blood Heart  Lung Liver NTissues
  <chr> <chr>      <dbl> <dbl> <dbl> <dbl> <dbl>
1 A2ML1 GTEX-11GSP -0.5   0.53  0.76 -0.1   4
2 A2ML1 GTEX-11NUK -0.08 -0.4  -0.26 -0.13  4
3 A2ML1 GTEX-11NV4 -0.37  0.11 -0.42 -0.61  4
4 A2ML1 GTEX-11TT1  0.3   -1.11  0.59 -0.12  4
5 A2ML1 GTEX-11TUW  0.02 -0.47  0.29 -0.66  4
6 A2ML1 GTEX-11ZUS -1.07 -0.41  0.67  0.06  4
7 A2ML1 GTEX-11ZVC -0.27 -0.51  0.13 -0.75  4
8 A2ML1 GTEX-1212Z -0.3   0.53  0.1  -0.48  4
9 A2ML1 GTEX-12696 -0.11  0.24  0.96  0.72  4
10 A2ML1 GTEX-12WSD  0.53  0.36  0.2   0.51  4
# ... with 376,873 more rows
```

- Why do we get back multiple rows per class?
- This is an extremely convenient idiom for finding the rows that minimize or maximize a condition

## Exercise: Max expression change in blood and lung

Which are the individual pairs that have both the max blood expression change *and* max lung expression change among all individuals with measurements for the same gene?

```
gtex_data %>%
  group_by(Gene) %>%
  filter(Blood == max(Blood), Lung==max(Lung))
# A tibble: 64 x 7
# Groups:   Gene [64]
  Gene      Ind      Blood Heart  Lung Liver NTissues
  <chr>    <chr>    <dbl> <dbl> <dbl> <dbl>   <dbl>
1 A4GALT   GTEX-12696  2.78 -1.02  2.31 -0.23     4
2 ABHD1    GTEX-VUSG   6.33  0.41  2.04 -0.04     4
3 AL162151.3 GTEX-WZTO   2.37 -0.19  4.23 -1.22     4
4 ANKRD36B  GTEX-12WSD  2.72  0.74  2.66  1.22     4
5 APOA1     GTEX-12WSD  5.45 NA     7      0.67     3
6 C14orf119 GTEX-11ZUS  2.51  0.76  1.85 -0.99     4
7 CD1D      GTEX-1B996  3.05  2.85  2.78  2.1      4
8 CTB-131B5.2 GTEX-1GN73  6.29 -1.17  5.51 -0.96     4
9 CTC-448F2.6 GTEX-131YS  4.1   0.75  2.67 -0.24     4
10 EVC       GTEX-UPK5   4.31  0.21  2.6   -0.61     4
# ... with 54 more rows
```

# Mutating grouped data

- `mutate()` is aware of grouping. When used on a grouped dataset, it applies the mutation separately in each group

```
gtex_data %>%
  group_by(Gene) %>%
  mutate(blood_diff_from_min = Blood - min(Blood)) %>%
  select(Gene, Ind, Blood, blood_diff_from_min)
# A tibble: 389,922 x 4
# Groups:   Gene [4,999]
   Gene   Ind      Blood blood_diff_from_min
  <chr> <chr>    <dbl>          <dbl>
1 A2ML1 GTEX-11DXZ -0.14          1.26
2 A2ML1 GTEX-11GSP -0.5           0.9
3 A2ML1 GTEX-11NUK -0.08          1.32
4 A2ML1 GTEX-11NV4 -0.37          1.03
5 A2ML1 GTEX-11TT1  0.3           1.7
6 A2ML1 GTEX-11TUV  0.02          1.42
7 A2ML1 GTEX-11ZUS -1.07          0.33
8 A2ML1 GTEX-11ZVC -0.27          1.13
9 A2ML1 GTEX-1212Z -0.3           1.1
10 A2ML1 GTEX-12696 -0.11          1.29
# ... with 389,912 more rows
```

- As always, `mutate` does not change the number of rows in the dataset

# Data Transformation with dplyr : : CHEAT SHEET



dplyr functions work with pipes and expect **tidy data**. In tidy data:



Each **variable** is in its own **column**



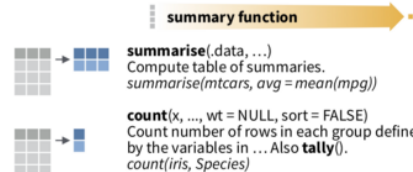
Each **observation**, or **case**, is in its own **row**



**x %>% f(y)** becomes **f(x, y)**

## Summarise Cases

These apply **summary functions** to columns to create a new table of summary statistics. Summary functions take vectors as input and return one value (see back).



### VARIATIONS

**summarise\_all()** - Apply funs to every column.  
**summarise\_at()** - Apply funs to specific columns.  
**summarise\_if()** - Apply funs to all cols of one type.

## Group Cases

Use **group\_by()** to create a "grouped" copy of a table. dplyr functions will manipulate each "group" separately and then combine the results.



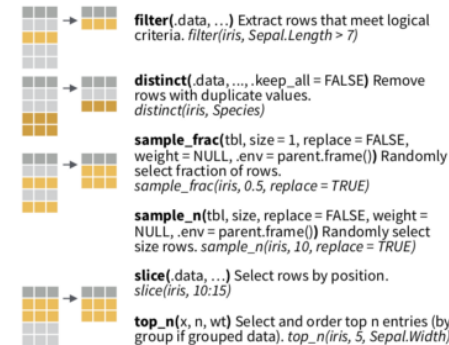
**group\_by(data, ..., add = FALSE)**  
Returns copy of table grouped by ...  
`g_iris <- group_by(iris, Species)`

**ungroup(x, ...)**  
Returns ungrouped copy of table.  
`ungroup(g_iris)`

## Manipulate Cases

### EXTRACT CASES

Row functions return a subset of rows as a new table.

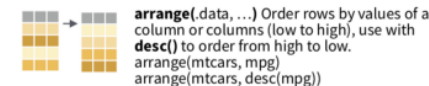


### Logical and boolean operators to use with filter()

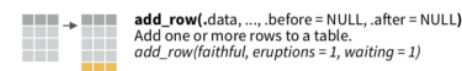
<	<=	is.na()	%in%		xor()
>	>=	!is.na()	!	&	

See **?base::logic** and **?Comparison** for help.

### ARRANGE CASES



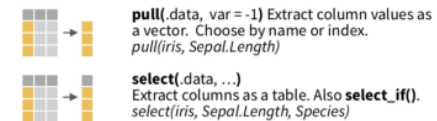
### ADD CASES



## Manipulate Variables

### EXTRACT VARIABLES

Column functions return a set of columns as a new vector or table.

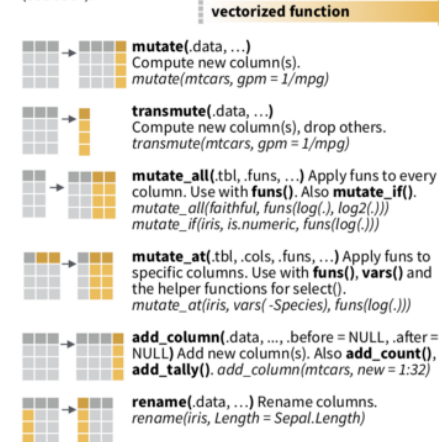


Use these helpers with **select()**,  
e.g. `select(iris, starts_with("Sepal"))`

<b>contains(match)</b>	<b>num_range(prefix, range)</b>	; e.g. <code>mpg:cyl</code>
<b>ends_with(match)</b>	<b>one_of(...)</b>	; e.g. <code>-Species</code>
<b>matches(match)</b>	<b>starts_with(match)</b>	

### MAKE NEW VARIABLES

These apply **vectorized functions** to columns. Vectorized funs take vectors as input and return vectors of the same length as output (see back).



RStudio® is a trademark of RStudio, Inc. • CC BY SA RStudio • info@rstudio.com • 844-448-1212 • rstudio.com • Learn more with `browseVignettes(package = c("dplyr", "tidyverse"))` • dplyr 0.7.0 • tidyverse 1.2.0 • Updated: 2017-03

Source: Rstudio Cheat Sheets. Download the [full dplyr cheat sheet here](#).