GENOME 569A

Class 6: Making figures

R is a deeply flawed language

R is designed to be very flexible. This flexibility makes code extremely slow and hard to understand, even for experts.

R's greatest strength is the diverse collection of packages. These packages don't interoperate well. And the "standard library" of functions and tools is weak, slow, confusing, and internally inconsistent

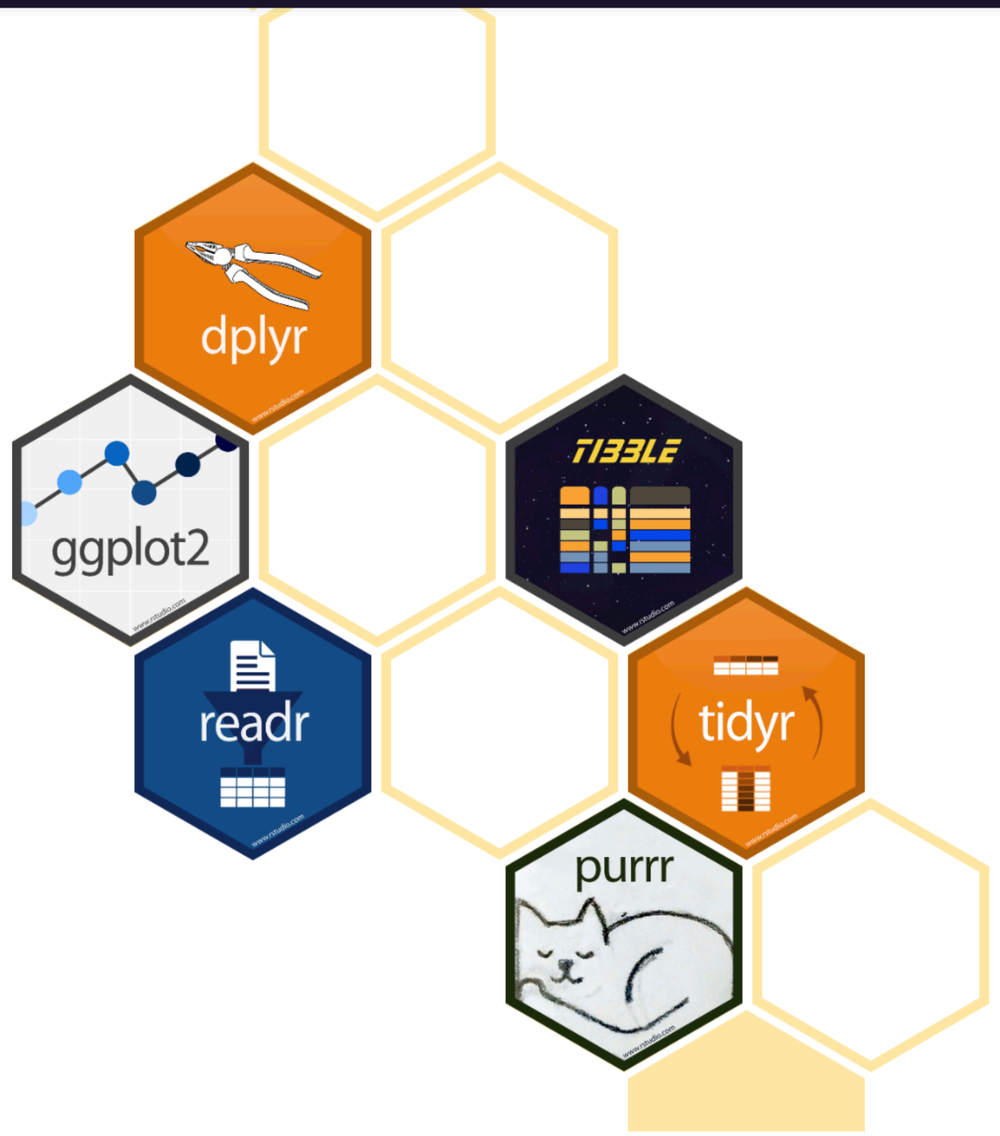
Basic, simple things that nearly every programmer expects to work a certain way (e.g. for loops) work strangely and badly.

Memory management is difficult to impossible, threading is inefficient and buggy, and writing fast packages usually requires that you write code in C/C++ or FORTRAN and wrap that code in R.

However, we lack viable alternatives, so it remains widely used in science.

I also have complaints about Python, Matlab, Mathematica, C++, and other languages. I'm a blast at parties.

Tidyverse



R packages for data science

The tidyverse is an opinionated **collection of R packages** designed for data science. All packages share an underlying design philosophy, grammar, and data structures.

Install the complete tidyverse with:

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

O'REILLY®

R for Data Science

VISUALIZE, MODEL, TRANSFORM, TIDY, AND IMPORT DATA

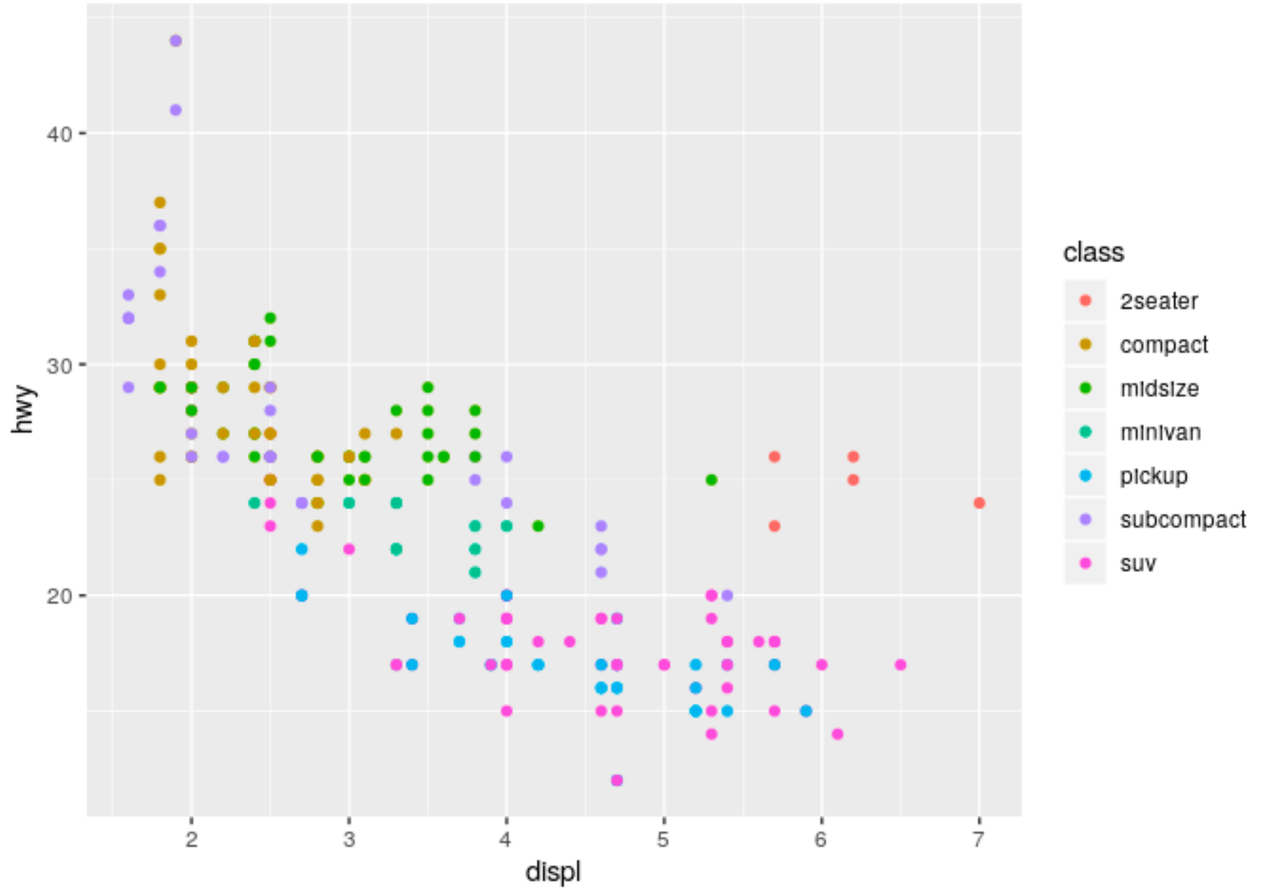
Hadley Wickham &
Garrett Grolemund

ggplot2

Problem: Generate plots using code instead of by hand

Solution: Grammar of graphics plotting

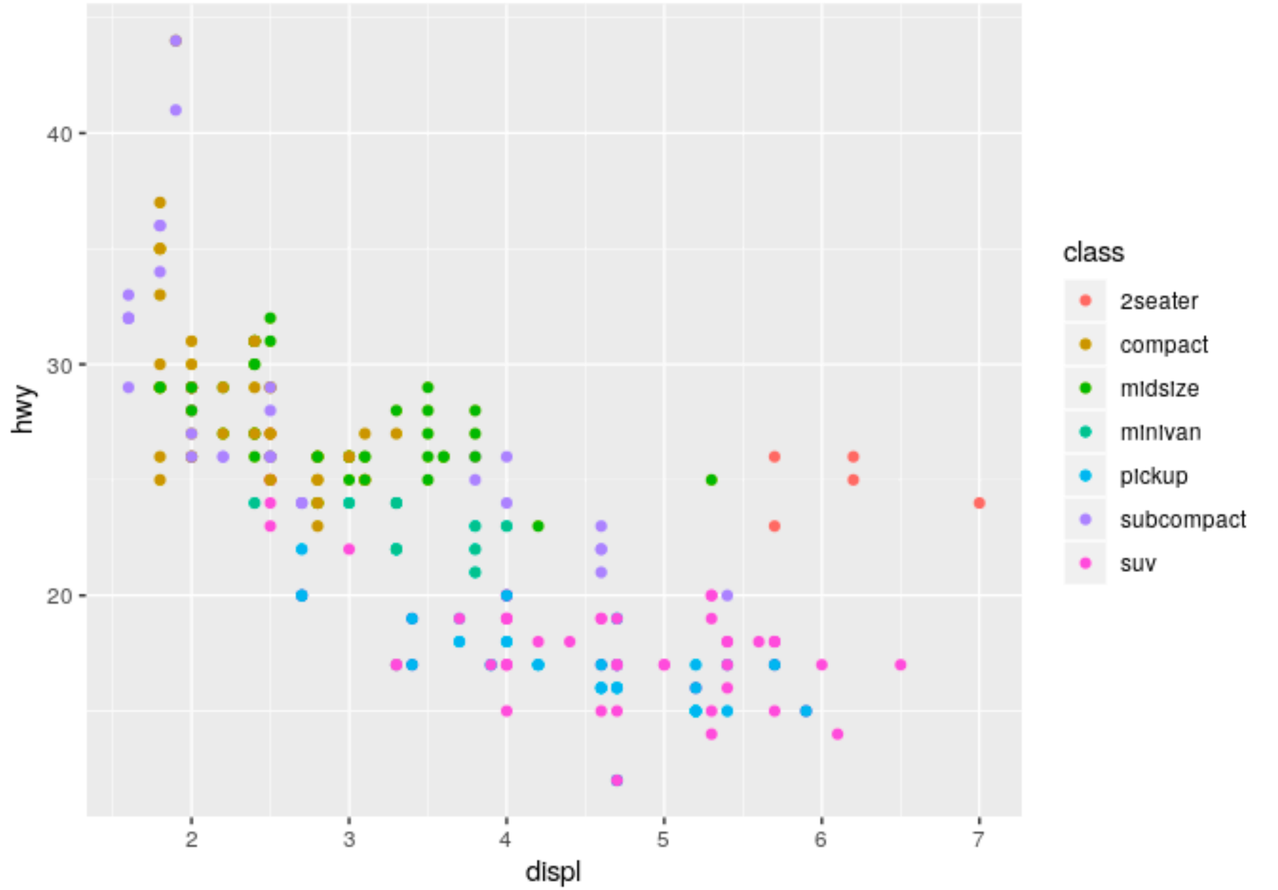
```
ggplot(mpg, aes(displ, hwy, colour = class)) +
  geom_point()
```



A simple ggplot2 example

`ggplot()` establishes coordinates.
Takes a data frame as input.

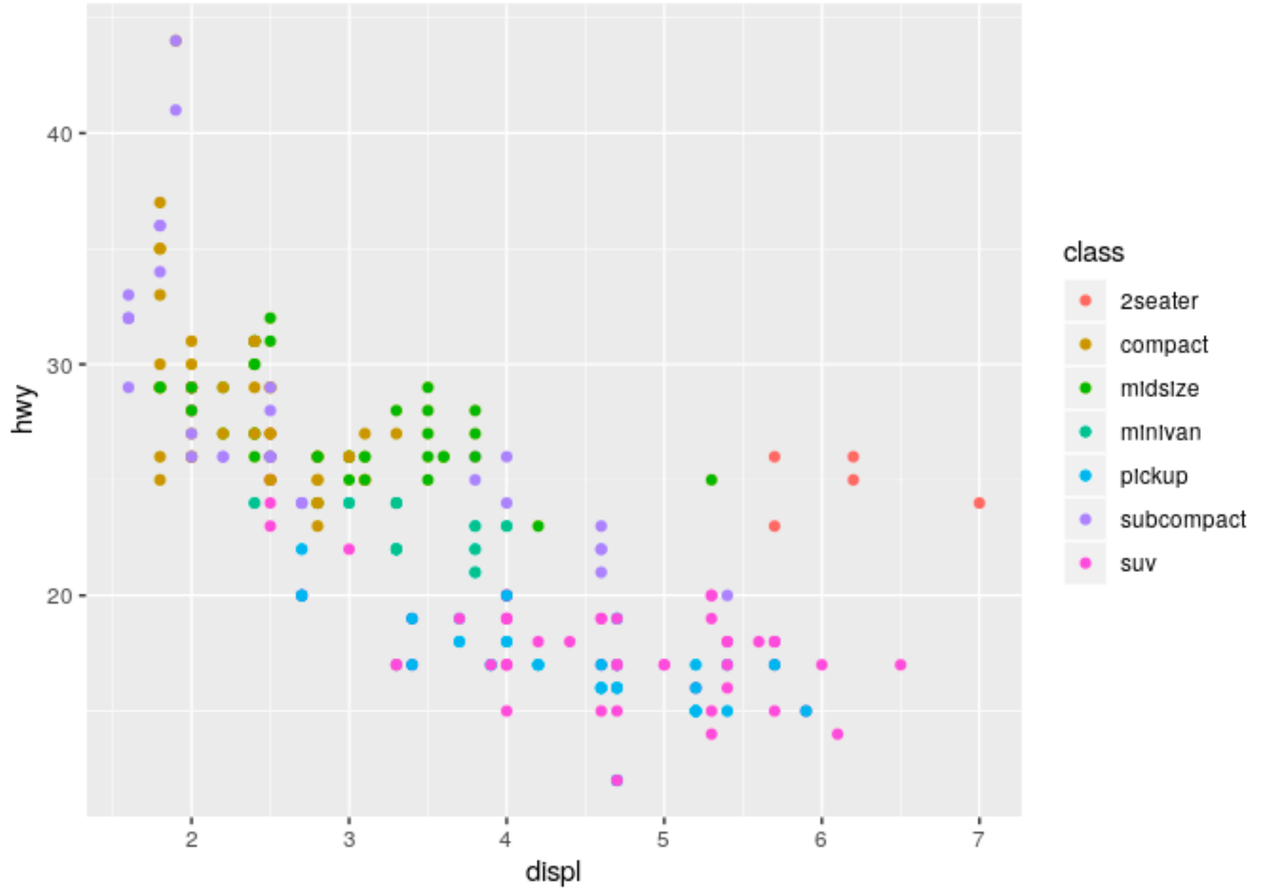
```
ggplot(mpg, aes(displ, hwy, colour = class)) +
  geom_point()
```



A simple ggplot2 example

```
ggplot(mpg, aes(displ, hwy, colour = class)) +
  geom_point()
```

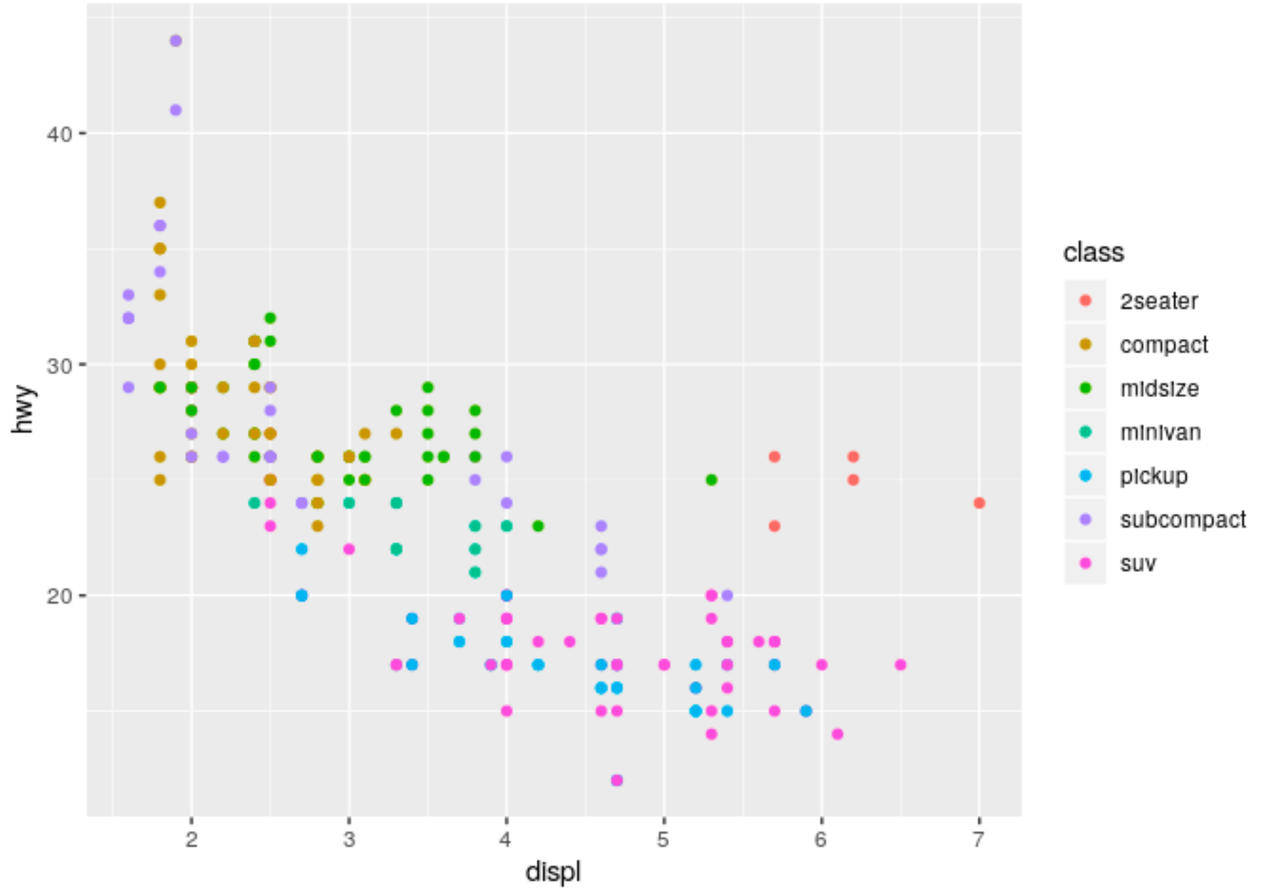
`aes()` maps variables in the data frame to *aesthetics* in the plot



A simple ggplot2 example

```
ggplot(mpg, aes(displ, hwy, colour = class)) +
  geom_point()
```

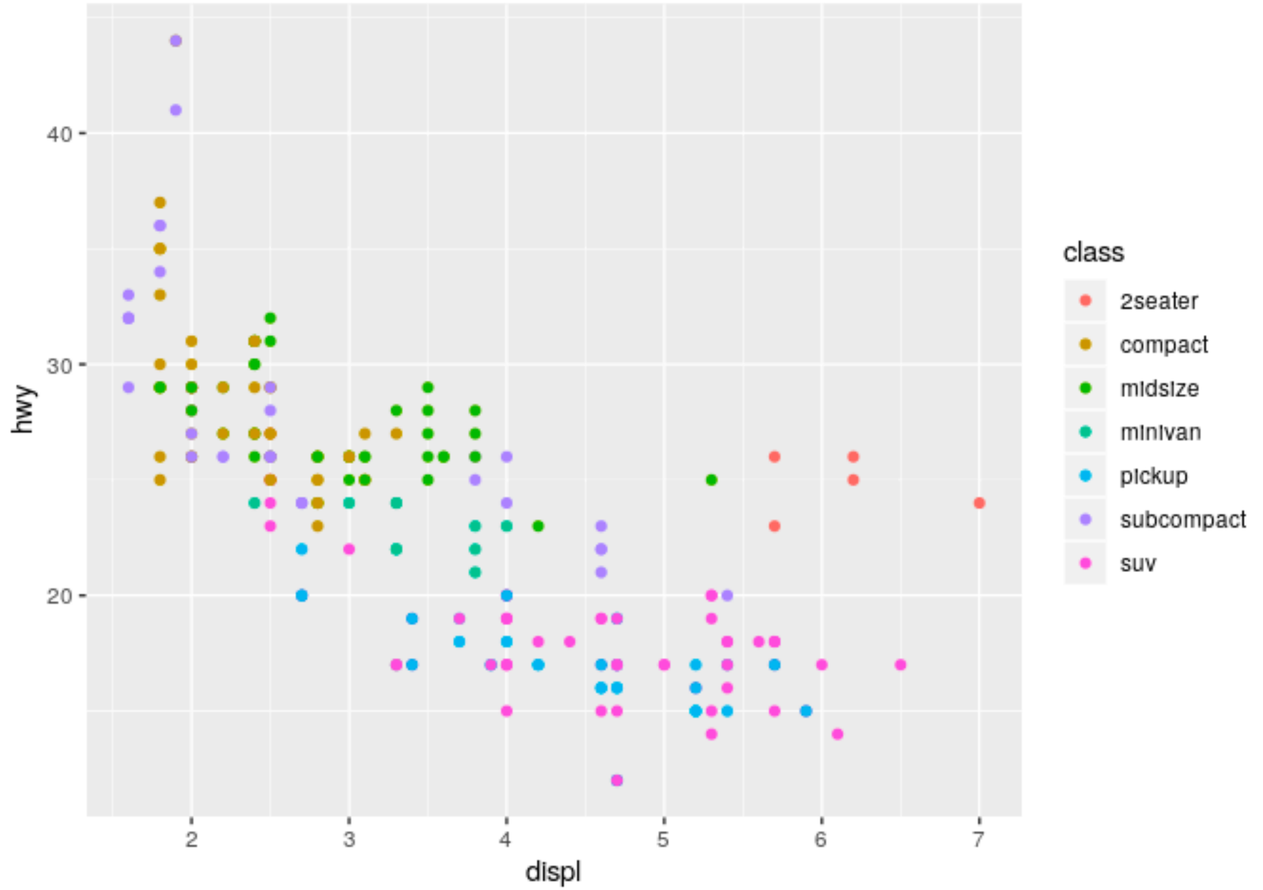
X Y



A simple ggplot2 example

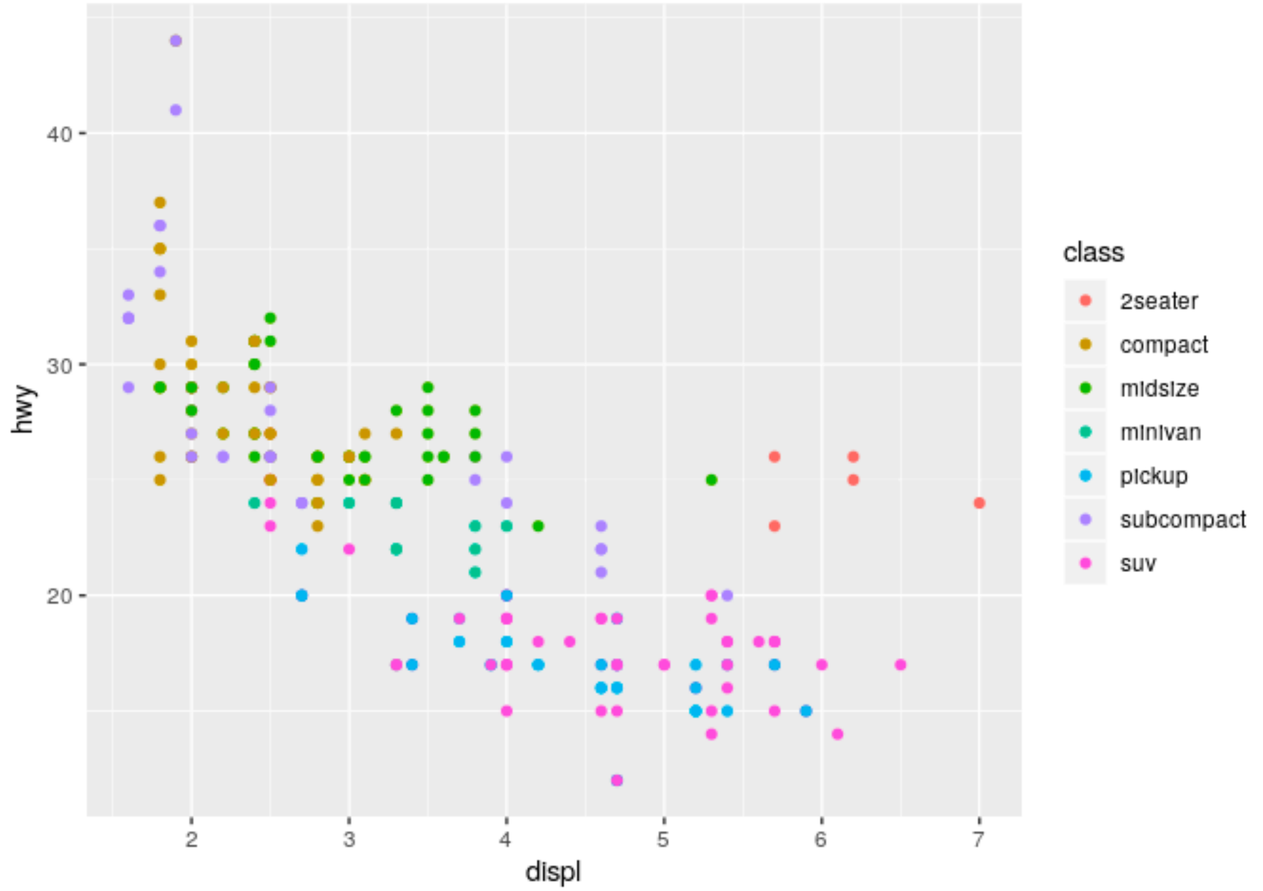
```
ggplot(mpg, aes(displ, hwy, colour = class)) +
  geom_point()
```

`geom_point()` adds a scatter plot as a new layer



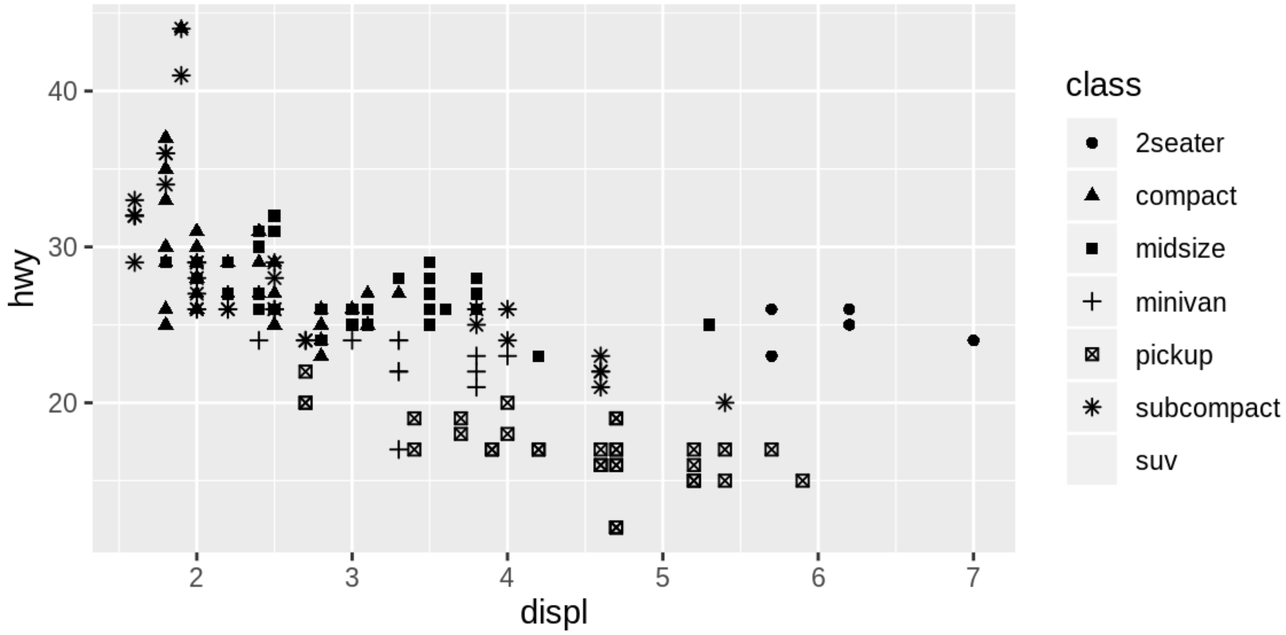
A simple ggplot2 example

```
ggplot(mpg, aes(displ, hwy, colour = class)) +
  geom_point()
```



A simple ggplot2 example

```
ggplot(mpg, aes(displ, hwy, shape = class)) +
  geom_point()
```



The “mpg” data set

```
mpg
#> # A tibble: 234 x 11
#>   manufacturer model displ  year   cyl trans      drv      cty   hwy fl    class
#>   <chr>        <chr> <dbl> <int> <int> <chr>      <chr> <int> <int> <chr> <chr>
#> 1 audi         a4      1.8  1999     4 auto(l5)   f        18    29 p     compa…
#> 2 audi         a4      1.8  1999     4 manual(m5) f        21    29 p     compa…
#> 3 audi         a4      2    2008     4 manual(m6) f        20    31 p     compa…
#> 4 audi         a4      2    2008     4 auto(av)   f        21    30 p     compa…
#> 5 audi         a4      2.8  1999     6 auto(l5)   f        16    26 p     compa…
#> 6 audi         a4      2.8  1999     6 manual(m5) f        18    26 p     compa…
#> # … with 228 more rows
```

Exercise set 1

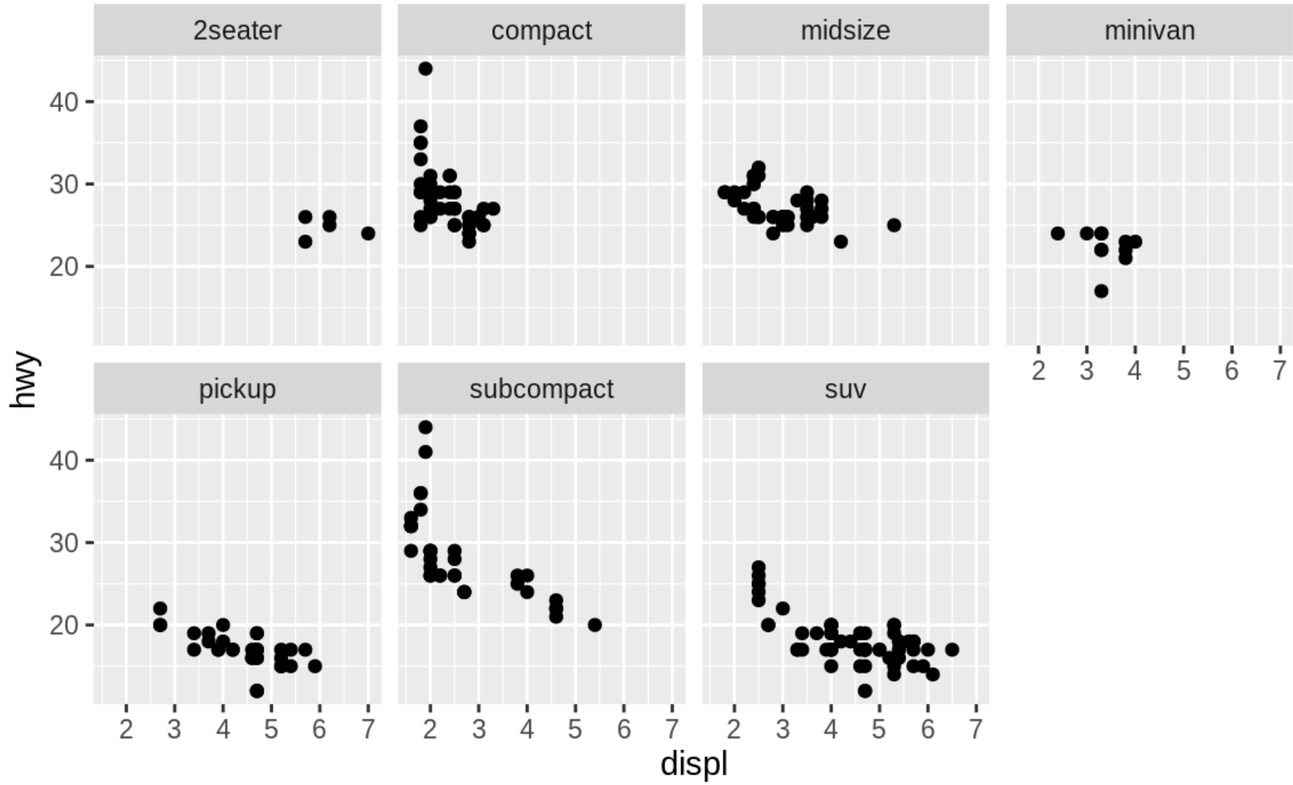
1. Run `ggplot(data = mpg)`. What do you see?

2. Make a scatterplot of `hwy` vs `cyl`.

3. What happens if you make a scatterplot of `class` vs `drv`? Why is the plot not useful?

“Faceting” reveals patterns in subsets of your data

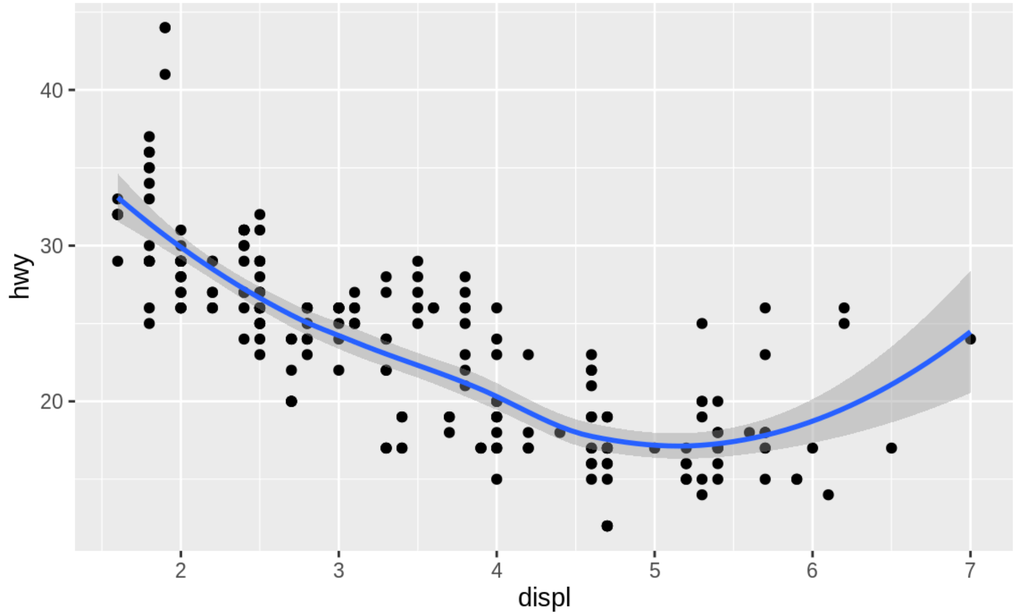
```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy)) +
  facet_wrap(~ class, nrow = 2)
```



Adding trend lines

```
ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) +
  geom_point() +
  geom_smooth()
```

This adds the trend line as a second layer with the same aesthetic mapping as the first



Exercise set 2

Run this code in your head and predict what the output will look like. Then, run the code in R and check your predictions.

```
ggplot(data = mpg, mapping = aes(x = displ, y = hwy, color = drv)) +
  geom_point() +
  geom_smooth(se = FALSE)
```

Exercise set 2

Will these two graphs look different? Why/why not?

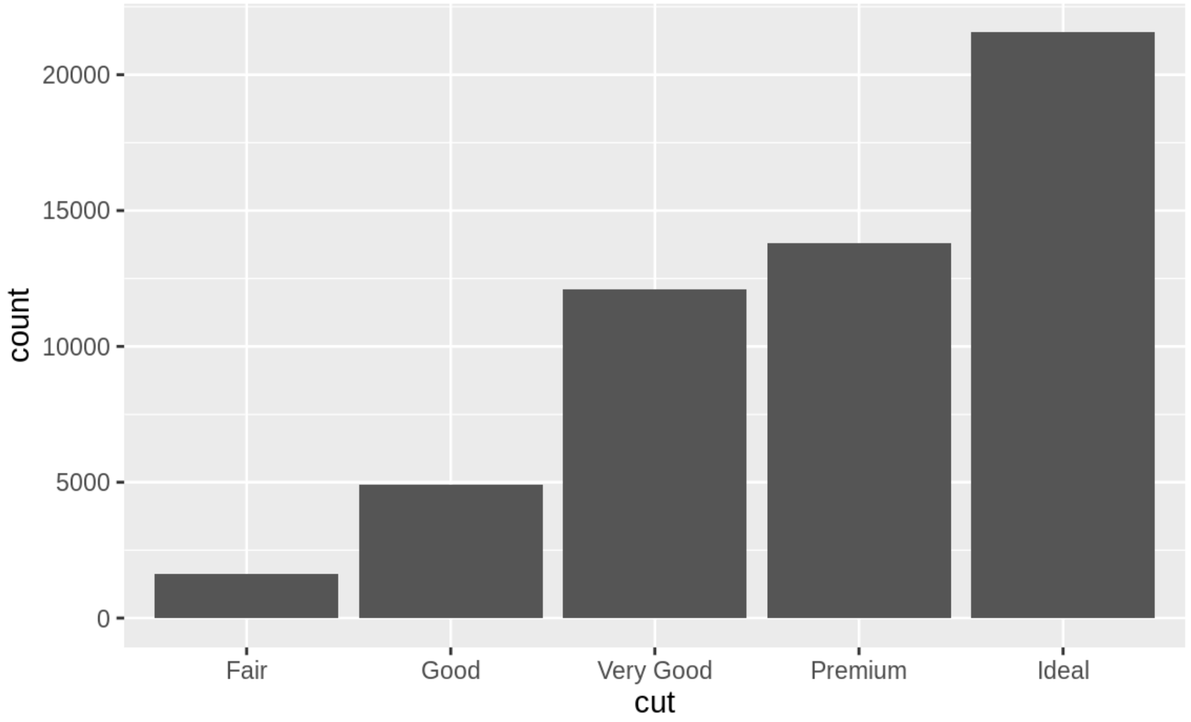
```
ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) +
  geom_point() +
  geom_smooth()

ggplot() +
  geom_point(data = mpg, mapping = aes(x = displ, y = hwy)) +
  geom_smooth(data = mpg, mapping = aes(x = displ, y = hwy))
```

Many plots apply statistical transformations or summaries to the input data

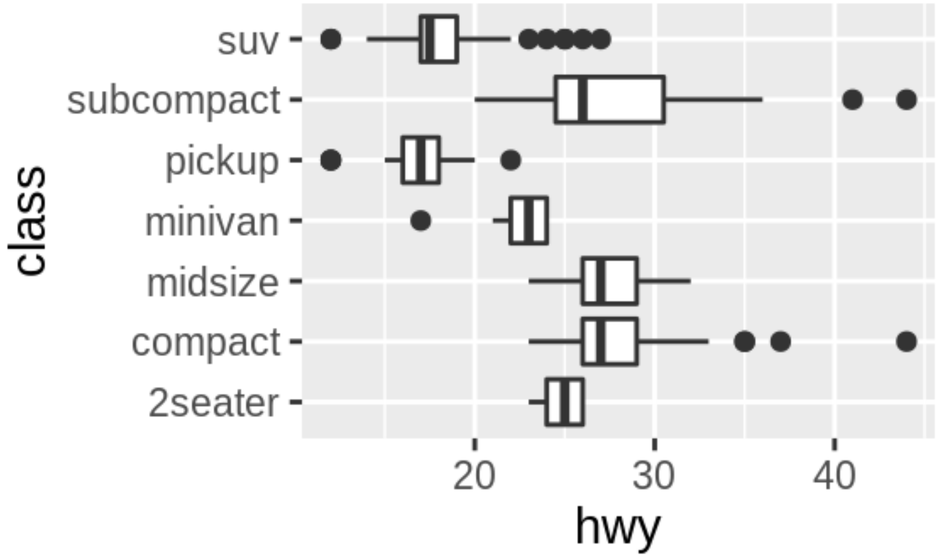
```
ggplot(data = diamonds) +
  geom_bar(mapping = aes(x = cut))
```

```
# A tibble: 53,940 x 10
   carat cut       color clarity depth table price     x     y     z
   <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
 1 0.23  Ideal     E     SI2      61.5    55   326  3.95  3.98  2.43
 2 0.21  Premium   E     SI1      59.8    61   326  3.89  3.84  2.31
 3 0.23  Good      E     VS1      56.9    65   327  4.05  4.07  2.31
 4 0.290 Premium   I     VS2      62.4    58   334  4.2   4.23  2.63
 5 0.31  Good      J     SI2      63.3    58   335  4.34  4.35  2.75
 6 0.24  Very Good J     VVS2     62.8    57   336  3.94  3.96  2.48
 7 0.24  Very Good I     VVS1     62.3    57   336  3.95  3.98  2.47
 8 0.26  Very Good H     SI1      61.9    55   337  4.07  4.11  2.53
 9 0.22  Fair      E     VS2      65.1    61   337  3.87  3.78  2.49
10 0.23  Very Good H     VS1      59.4    61   338  4     4.05  2.39
# … with 53,930 more rows
```



Many plots apply statistical transformations or summaries to the input data

```
ggplot(data = mpg, mapping = aes(x = class, y = hwy)) +
  geom_boxplot() +
  coord_flip()
```



Many plots apply statistical transformations or summaries to the input data

1. **`geom_bar()`** begins with the **`diamonds`** data set

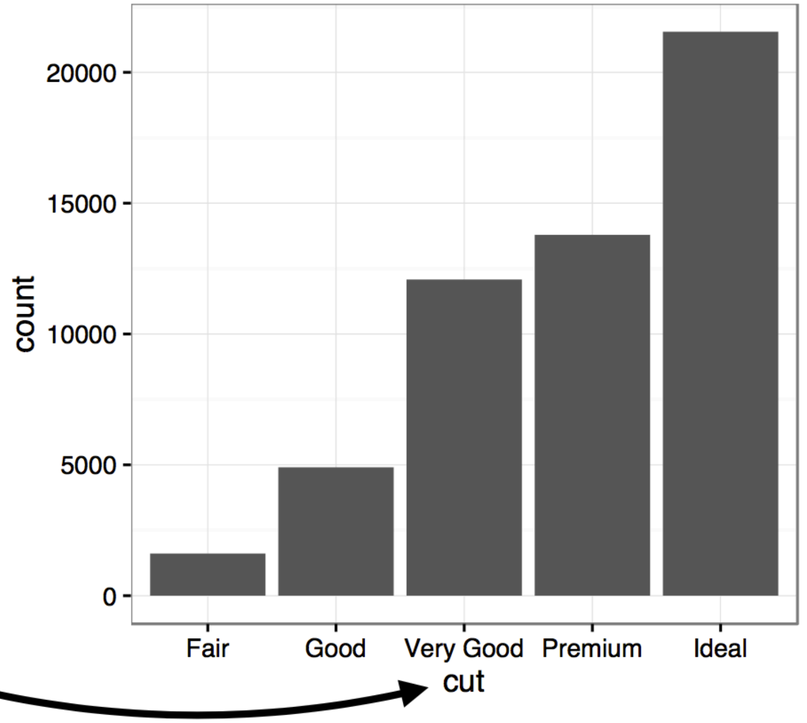
carat	cut	color	clarity	depth	table	price	x	y	z
0.23	Ideal	E	SI2	61.5	55	326	3.95	3.98	2.43
0.21	Premium	E	SI1	59.8	61	326	3.89	3.84	2.31
0.23	Good	E	VS1	56.9	65	327	4.05	4.07	2.31
0.29	Premium	I	VS2	62.4	58	334	4.20	4.23	2.63
0.31	Good	J	SI2	63.3	58	335	4.34	4.35	2.75
…	…	…	…	…	…	…	…	…	…

2. **`geom_bar()`** transforms the data with the "count" stat, which returns a data set of cut values and counts.

stat_count()

cut	count	prop
Fair	1610	1
Good	4906	1
Very Good	12082	1
Premium	13791	1
Ideal	21551	1

3. **`geom_bar()`** uses the transformed data to build the plot. cut is mapped to the x axis, count is mapped to the y axis.

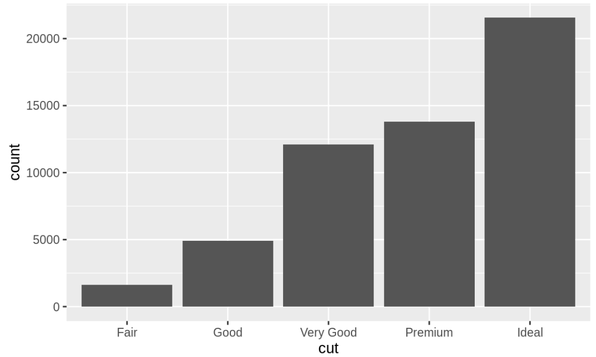


All plots apply statistical transformations or summaries to the input data

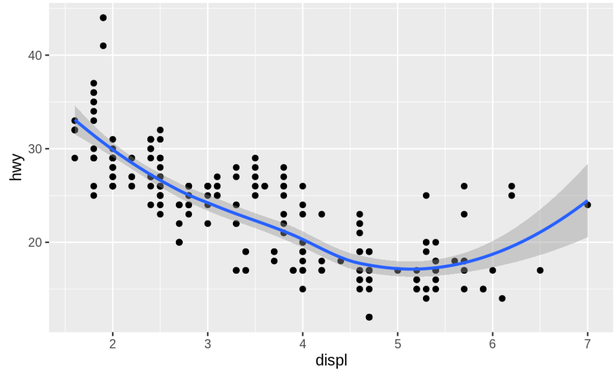
Every plot type has a default statistical transformation. For some, it's the identity.

Every statistical transformation has a default type of plot.

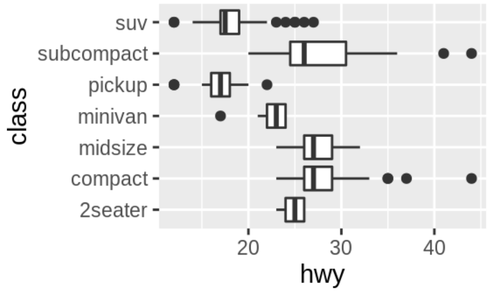
bar charts, histograms, and frequency polygons bin your data and then plot bin counts, the number of points that fall in each bin.



Smoothers fit a model to your data and then plot predictions from the model



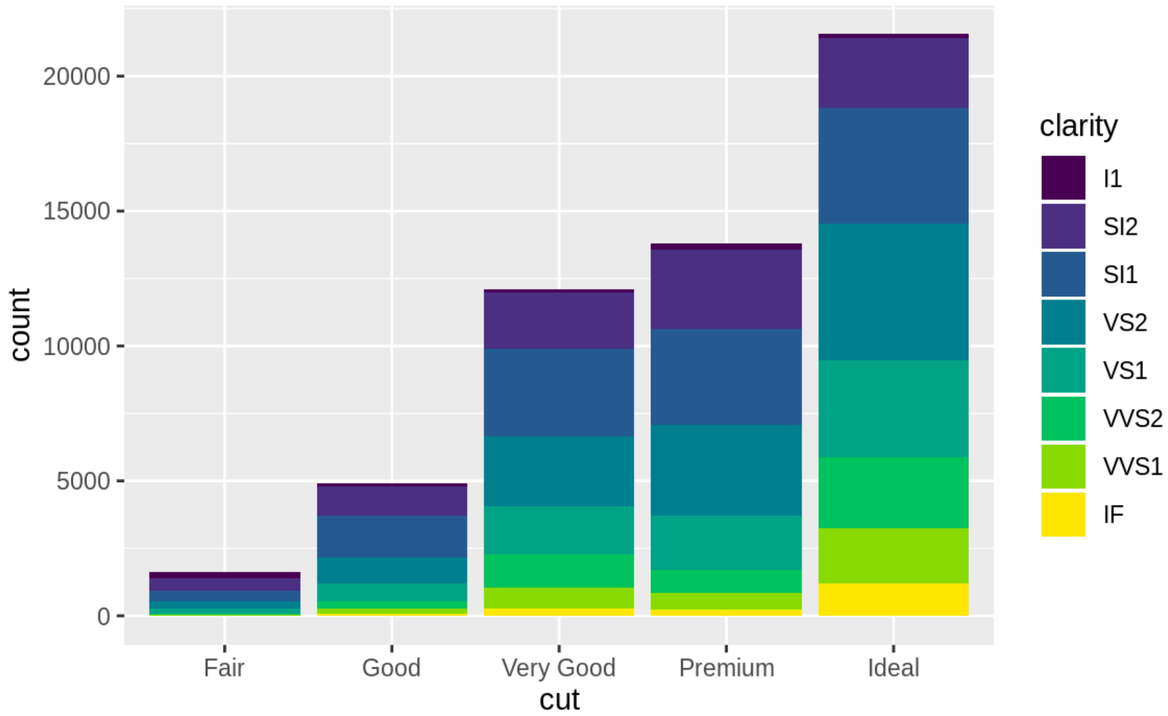
Boxplots compute a robust summary of the distribution and then display a specially formatted box



You can apply position adjustments to plots

```
ggplot(data = diamonds) +
  geom_bar(mapping = aes(x = cut, fill = clarity))
```

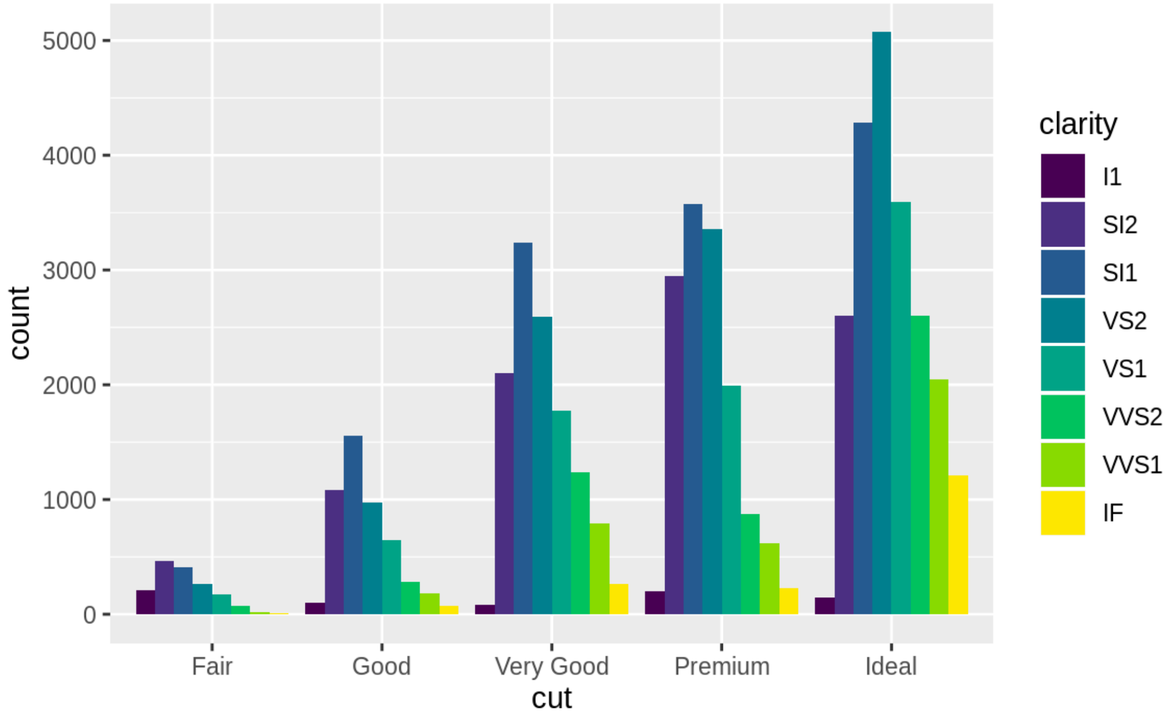
Mapping `clarity` to the "fill" aesthetic causes the plot to break out the rows by this variable and then **stack** the bars. Stacking is a **position adjustment**.



You can apply position adjustments to plots

```
ggplot(data = diamonds) +
  geom_bar(mapping = aes(x = cut, fill = clarity),
           position = "dodge")
```

You can change the type of position adjustment that gets applied.



Exercise set 3

1. Turn a stacked bar chart into a pie chart using `coord_polar()`.

2. What's the difference between `coord_quickmap()` and `coord_map()`?

3. What does the plot below tell you about the relationship between city and highway mpg? Why is `coord_fixed()` important? What does `geom_abline()` do?

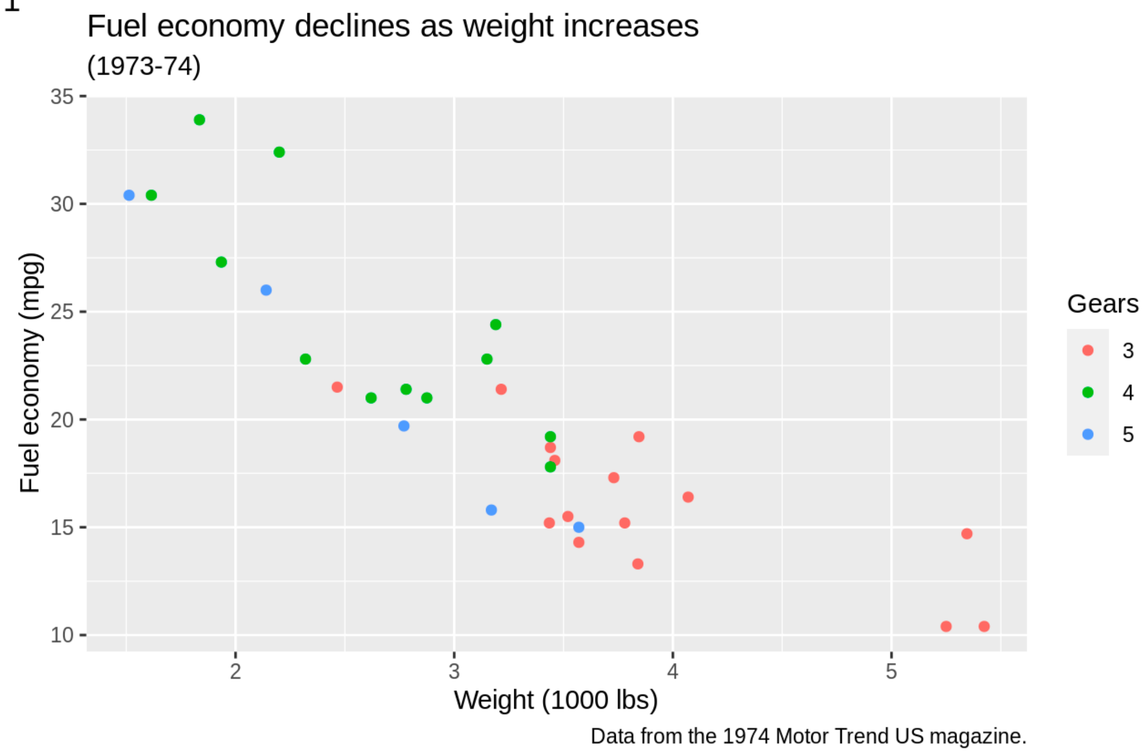
```
ggplot(data = mpg, mapping = aes(x = cty, y = hwy)) +
geom_point() +
geom_abline() +
coord_fixed()
```

ggplot() returns a plot object that can be modified later

```
p1 <- ggplot(mtcars2) +
geom_point(aes(x = wt, y = mpg, colour = gear)) +
labs(title = "Fuel economy declines as weight increases",
     subtitle = "(1973-74)",
     caption = "Data from the 1974 Motor Trend US magazine.",
     tag = "Figure 1",
     x = "Weight (1000 lbs)",
     y = "Fuel economy (mpg)",
     colour = “Gears")

p1
```

Figure 1

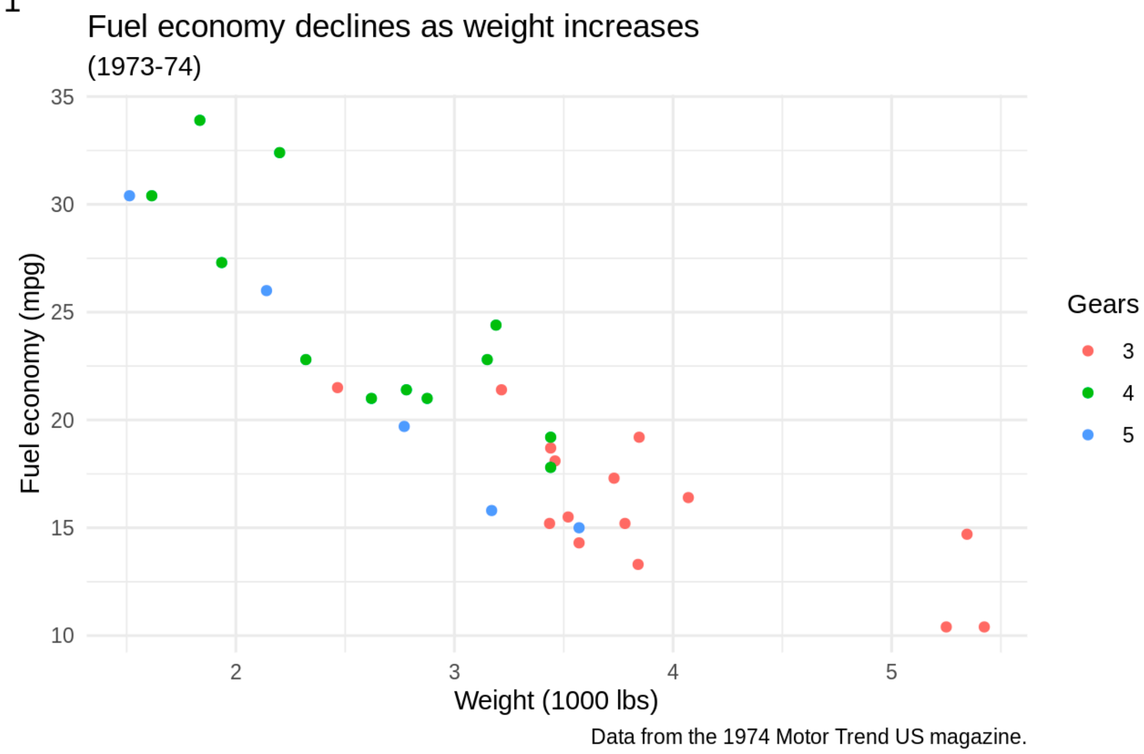


ggplot() returns a plot object that can be modified later

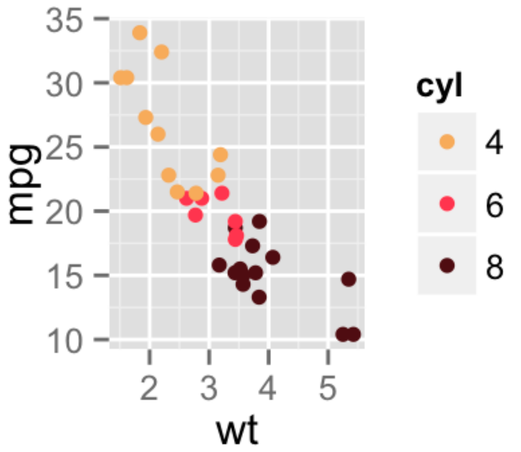
```
p1 <- ggplot(mtcars2) +
geom_point(aes(x = wt, y = mpg, colour = gear)) +
labs(title = "Fuel economy declines as weight increases",
     subtitle = "(1973-74)",
     caption = "Data from the 1974 Motor Trend US magazine
     tag = "Figure 1",
     x = "Weight (1000 lbs)",
     y = "Fuel economy (mpg)",
     colour = “Gears")

p1 + theme_minimal()
```

Figure 1



There are many ggplot2 themes

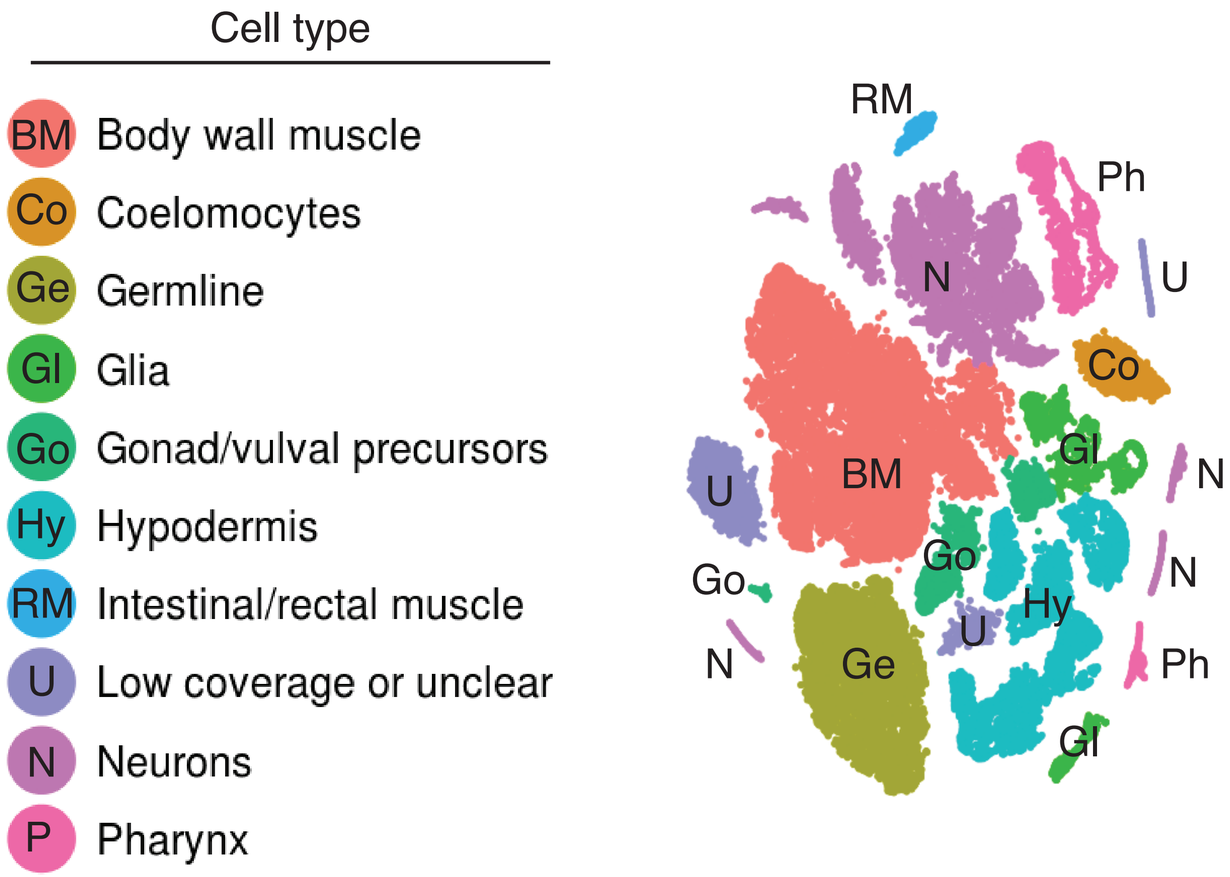



```
library(wesanderson)
# Box plot
bp+scale_fill_manual(values=wes_palette(n=3, name="GrandBudapest"))
# Scatter plot
sp+scale_color_manual(values=wes_palette(n=3, name="GrandBudapest"))
```

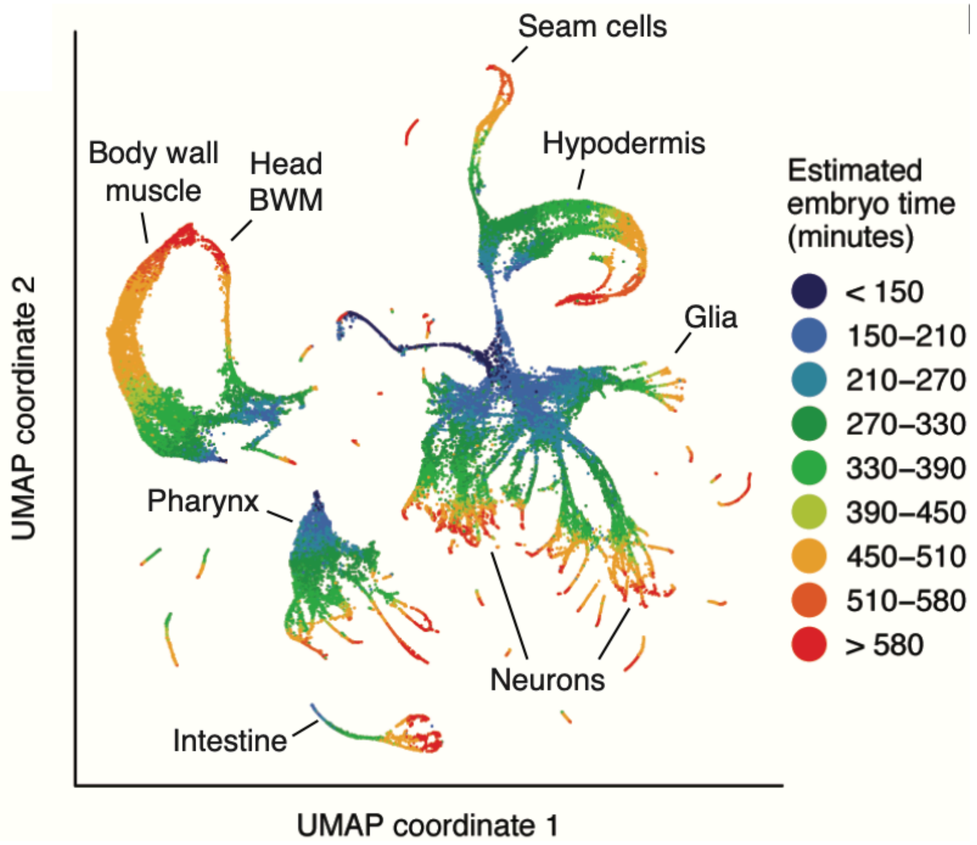
Problem: Organize cells according to type & developmental maturity

Monocle 3

Problem: Organize cells according to type & developmental maturity

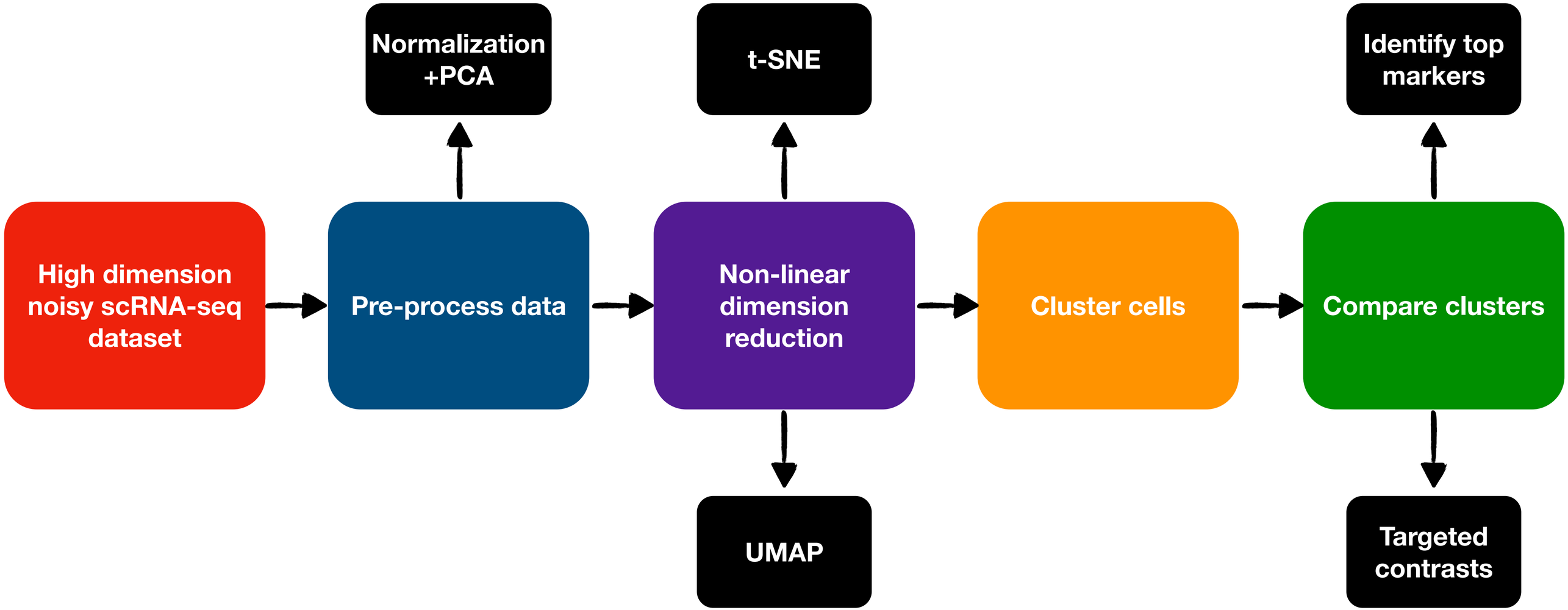


Cao & Packer et al, Science 2017



Packer et al, Science 2019

Workflow steps



Software & installation

- Monocle3 runs in the R statistical computing environment
- Needs R version 3.5 or higher
- To install from the Trapnell Lab GitHub

```
devtools :: install_github('cole-trapnell-lab/monocle3')
```

- To test the installation

```
library(monocle3)
```

Import your own data

- To create your own cds object, use gene x cell matrix (mat), gene data frame (gene_meta) and cell data frame (cell_meta):

```
cds <- new_cell_dataset(mat, cell_meta, gene_meta)
```

- Monocle3 can also import data from 10x experiments directly into cds objects

```
cds <- load_cellranger_data("cell_ranger_output")
```

Accessor functions for cds

- **exprs/counts**: A numeric matrix of expression values, where rows are genes and columns are cells

- **pData/colData**: An object where rows are cells and columns are cell attributes such as cell type, culture condition, etc

- **fData/rowData**: An object where rows are features (e.g. genes) and columns are gene attributes such as biotype, gc content, etc

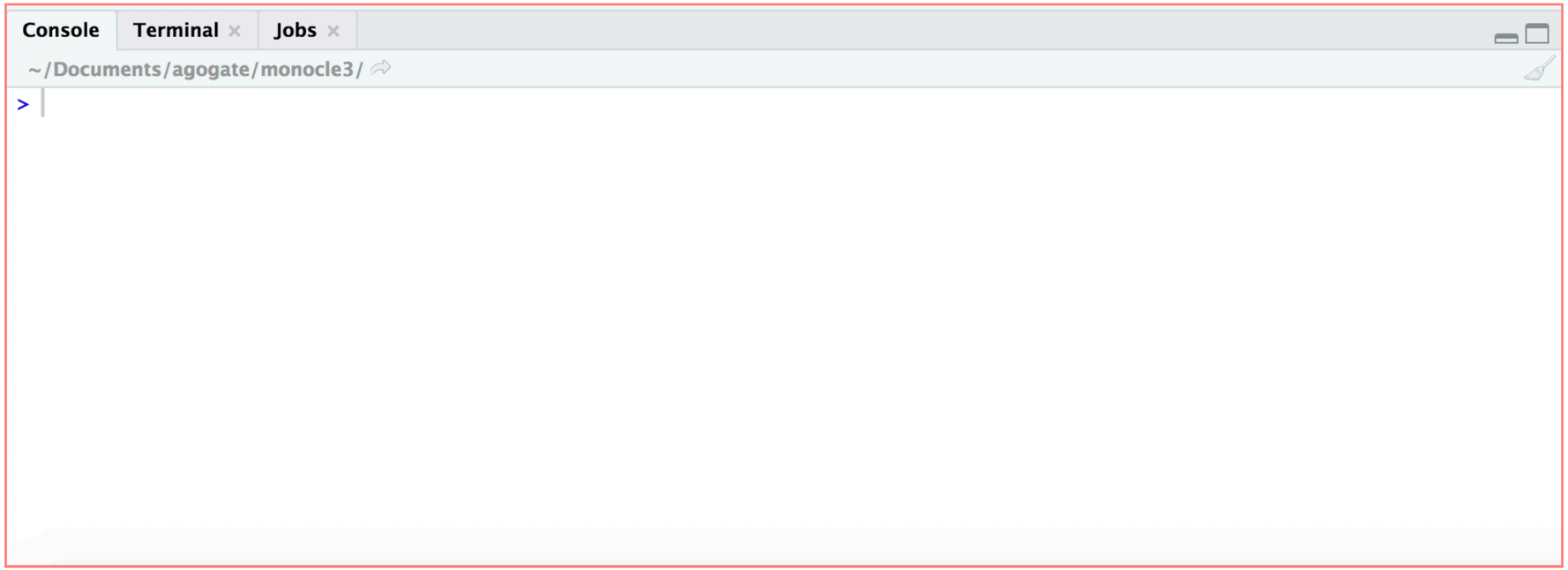
`colData(cds)` `rowData(cds)` `counts(cds)`

`colData(cds)`

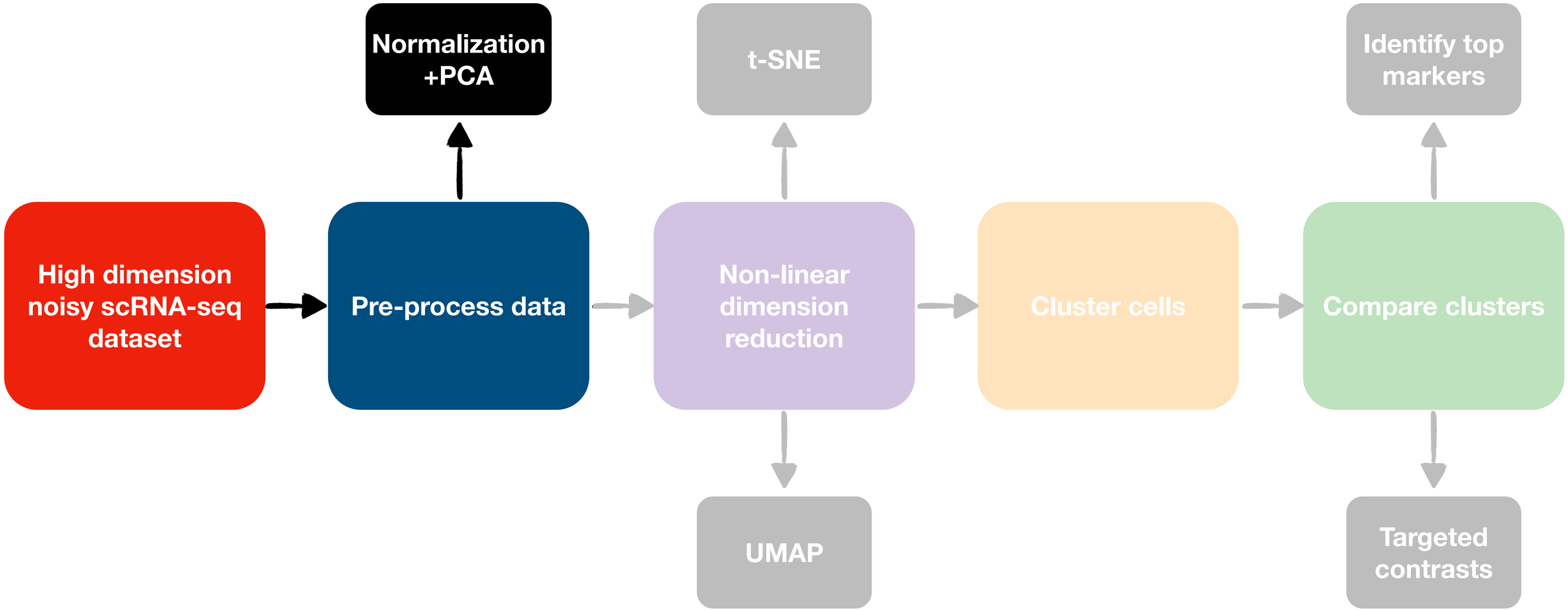
	cell_type	culture_cond	Size_Factor
cell_1			
cell_2			
cell_3			
cell_n			

`rowData(cds)`

	biotype	gc_content
gene_1		
gene_2		
gene_3		
gene_n		



Pre-process & batch

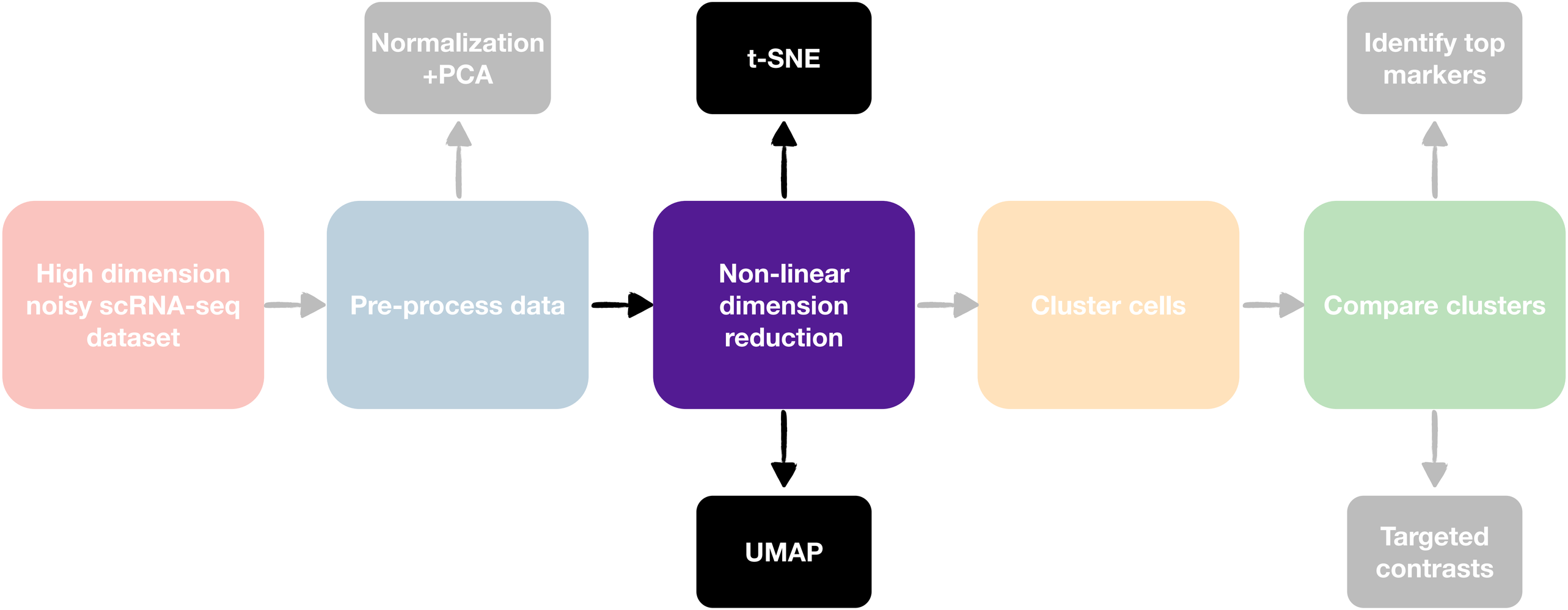


Preprocess the data

- Preprocess data with initial dimensionality reduction

```
cds <- preprocess_cds(cds, num_dim = 100)
```

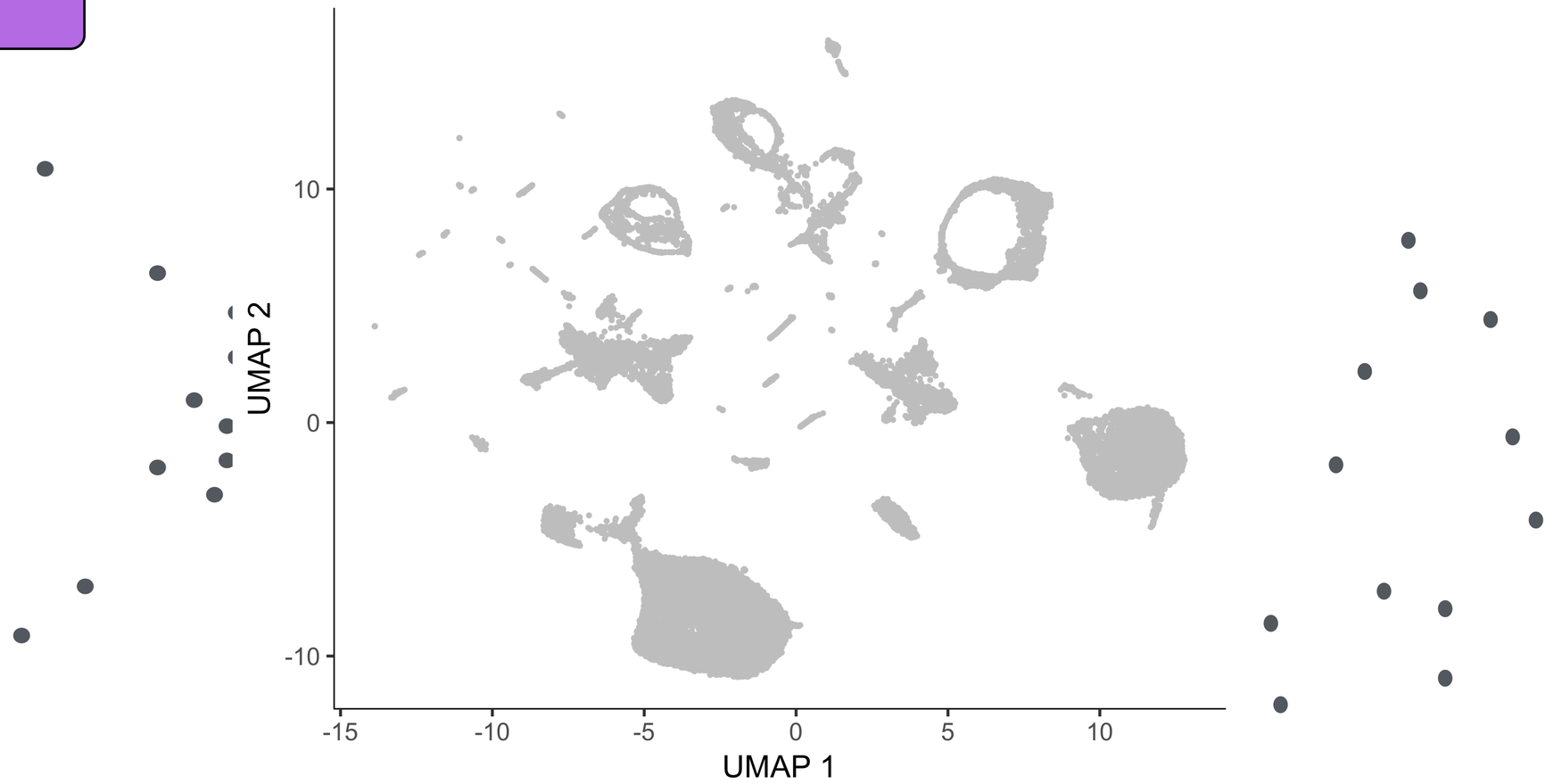
Reduce dimension



Pre-process

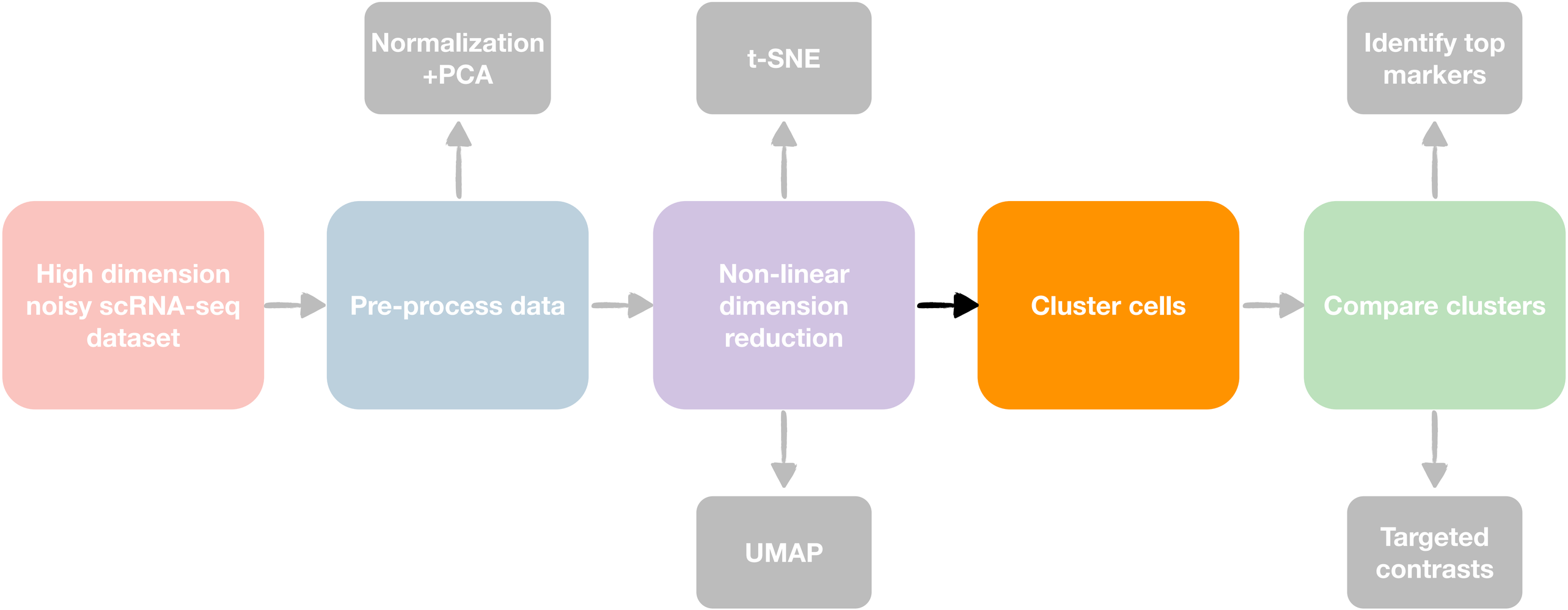
Reduce dimension

Reduce dimensions




```
cds <- reduce_dimension(cds)
plot_cells(cds)
```

Cluster cells



Next we cluster the cells

- The cluster_cells function of monocle3 allows users to group similar cells according to global expression profiles
- In addition, partitions (super-clusters) are calculated for dividing distinct trajectories
- You can access values using the following:

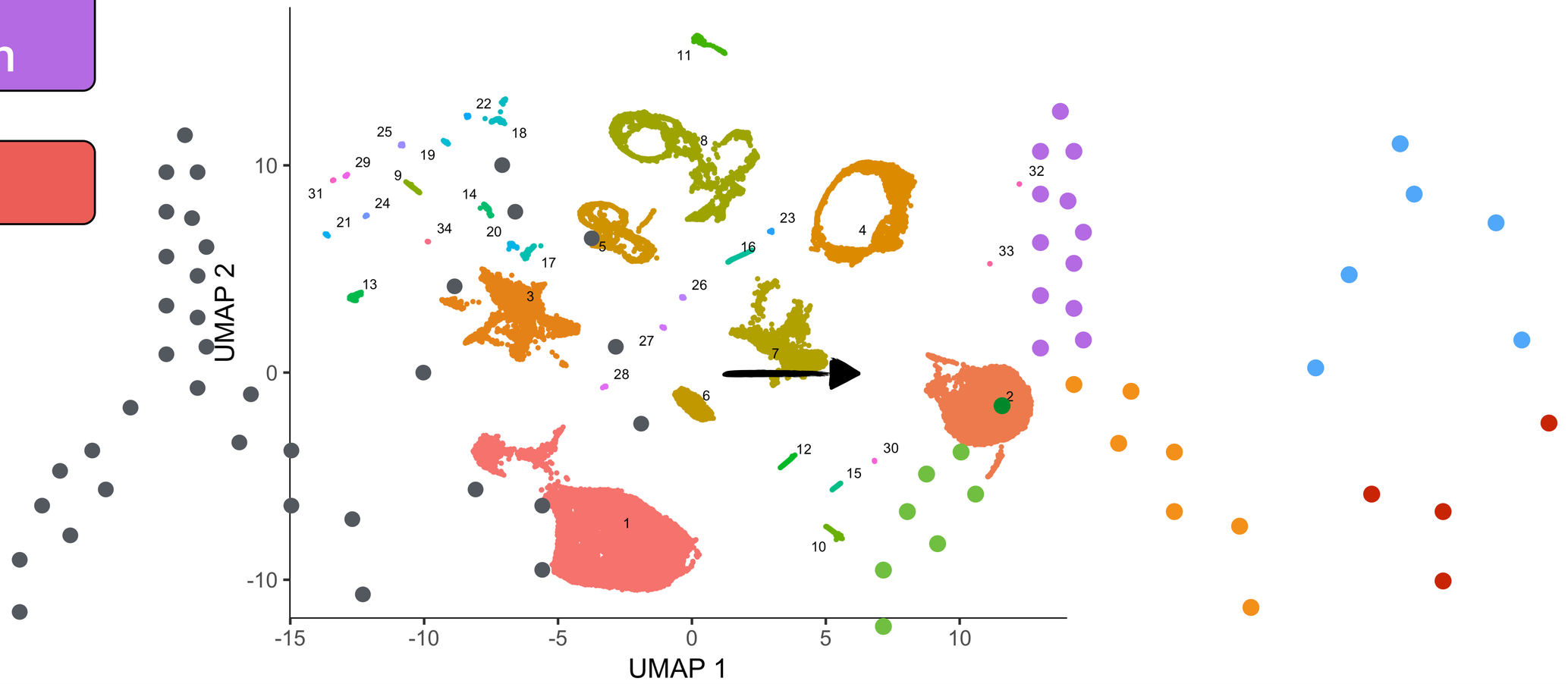
```
cds <- cluster_cells(cds)
head(partitions(cds, reduction_method = "UMAP"))
head(clusters(cds, reduction_method = "UMAP"))
```

Pre-process

Reduce dimension

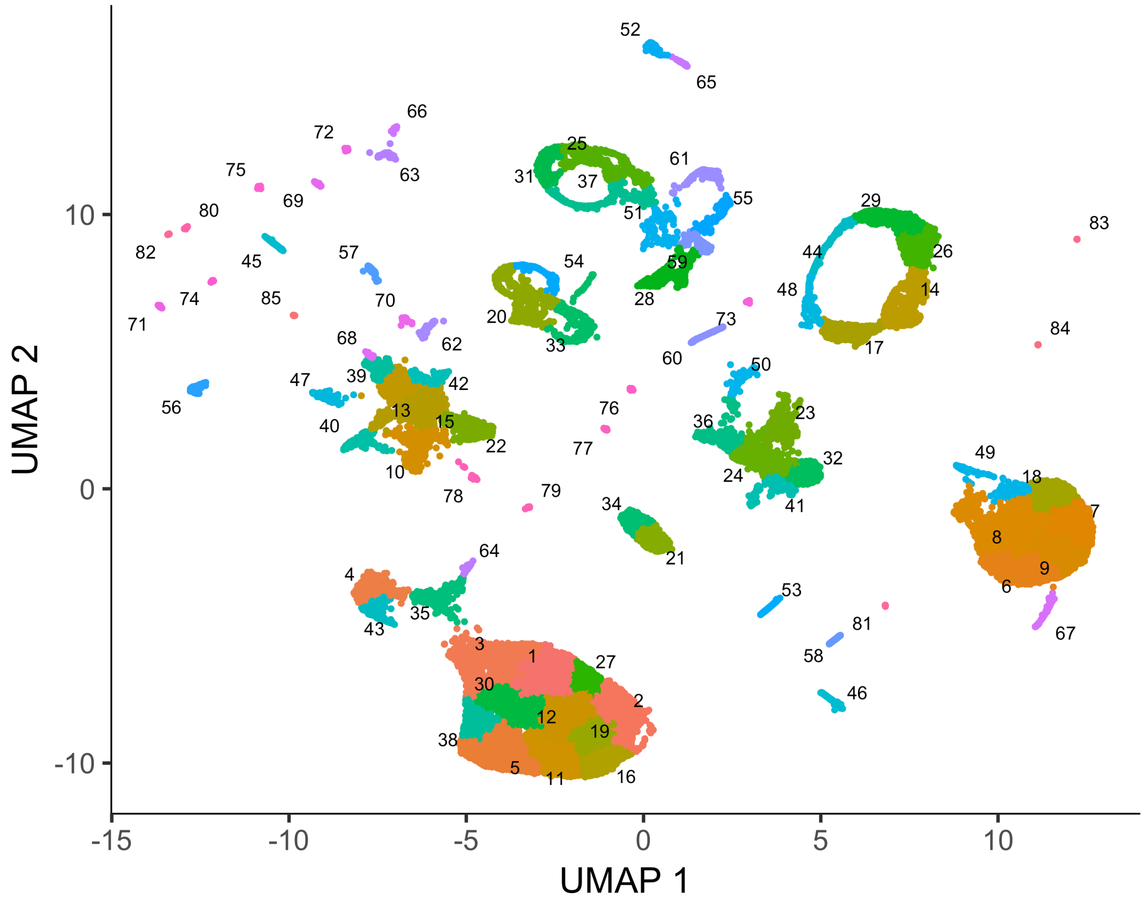
Cluster

Group cells by partitions



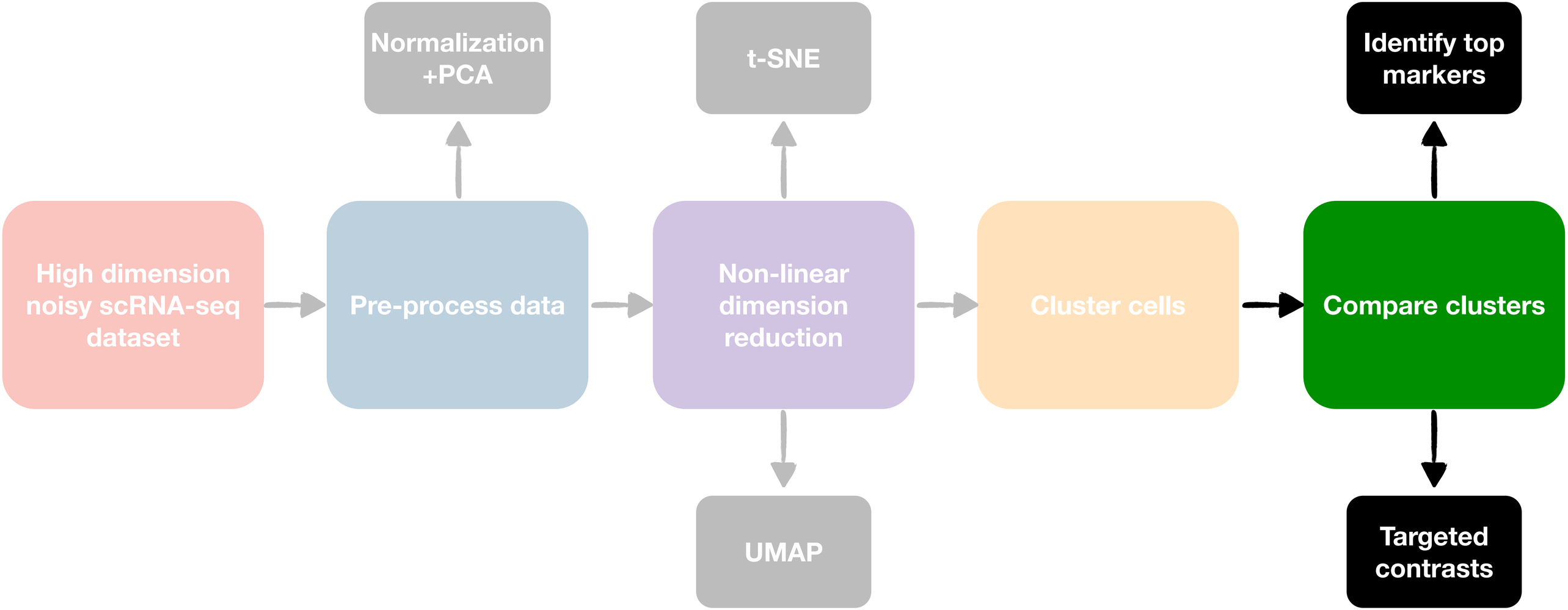

```
cds <- cluster_cells(cds)
plot_cells(cds, color_cells_by="partition",
group_cells_by="partition")
```

Group cells by clusters




```
cds <- cluster_cells(cds)
plot_cells(cds)
```

Compare clusters



THE END

For now.