Introduction to the tidyverse

Using slides adapted from R4All 2017 Beckerman, Petchey, Childs and Cooper

Welcome to the tidyverse (tidyverse.org)

The tidyverse

Components

The tidyverse is a collection of R packages that share common philosophies and are designed to w ite is a work-in-progress guide to the tidyverse and its packages.

What can the tidyverse do?

- Reading in data with readr
- Data manipulation using dplyr
- Tidying data with tidyr
- Plotting with ggplot2
- etc etc.
- All bundled together for easy installation and loading (yay!)

dplyr

each dplyr function does ONE thing, very very quickly and effectively.

each dplyr function works on dataframes and produces a new dataframe

Now work with dealing-with-data-dplyr.Rmd

- Work along with me (to save you having to type everything). There are periodic practicals.

Viewing your data

tbl_df()
glimpse()

vs. str()

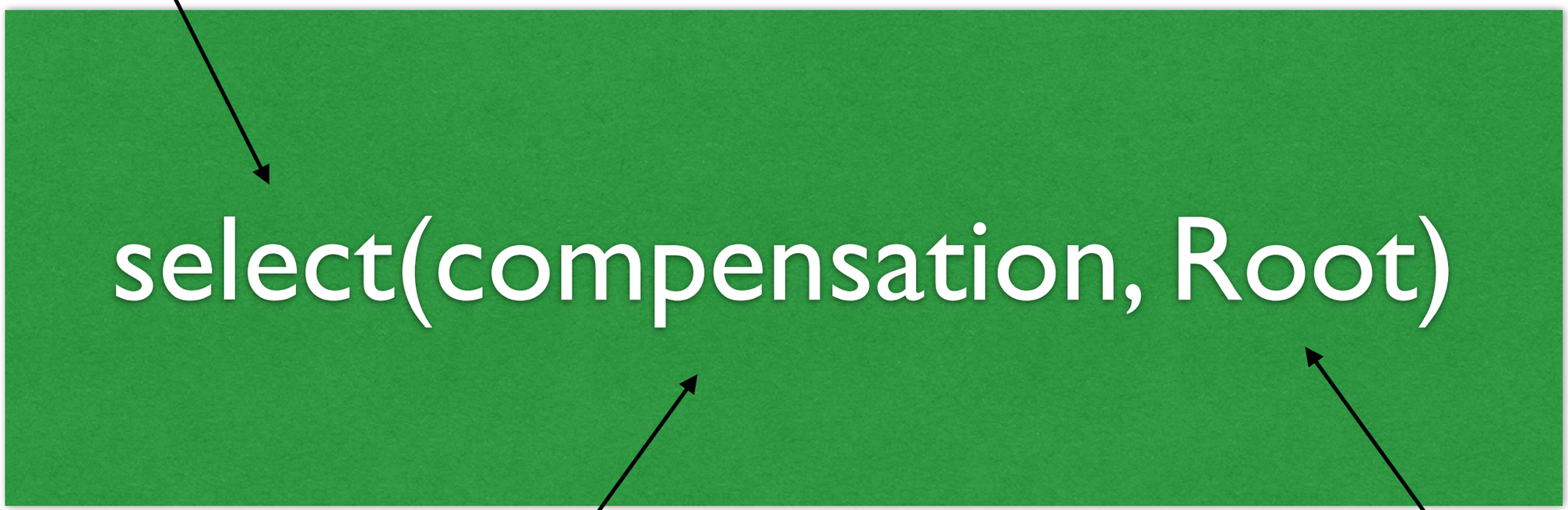
subsetting

select()

slice()

Get a column

dplyr function



name of the data frame

name of the column

Get more than one column

```
select(compensation, Root, Fruit)
```

Drop a column

```
select(compensation, -Root)
```

Helper functions

There are several helper functions that work with `select` to simplify common variable selection tasks:

- `starts_with("xyz")`: every name that starts with `"xyz"`
- `ends_with("xyz")`: every name that ends with `"xyz"`
- `contains("xyz")`: every name that contains `"xyz"`
- `matches("xyz")`: every name that matches `"xyz"`
- `one_of(names)`: every name that appears in `names` (character vector).

```
select(compensation, ends_with('oot'))
```

```
select(compensation, -ends_with('oot'))
```

Select rows

```
slice(compensation, 2)
slice(compensation, 2:5)
slice(compensation, c(2, 5))
```

To use the modified datasets you need to assign them to something…

```
comp2 <- select(compensation, -Root)
```

- Choose any name you want but note that R will overwrite things in R. So avoid using the same name as the original data frame.
- Here compensation is all the data, comp2 is the data without the Root column.

Practical

1. Create a dataframe containing only the Root column
2. Create a dataframe containing only the 8th row
3. Create a dataframe containing only the 8th row of the Fruit column. There are THREE fundamentally different ways of doing this, if you work out one way, try and find the other two...

```r
# Get the 8th row
slice(compensation, 8)
# Get the Fruit column
select(compensation, Fruit)
# To get the 8th row of the Fruit column
slice(select(compensation, Fruit), 8)
# OR
select(slice(compensation, 8), Fruit)
# OR
FruitOnly <- select(compensation, Fruit)
slice(FruitOnly, 8)
# OR pipe
compensation %>%
select(Fruit) %>%
slice(8)
```

piping %>%

- Piping is a magical, fast way to do multiple things to a dataset.
- Start with the dataset name (remember all dplyr functions start with the data name)
- Then add required functions separated by %>%

name of the data frame

piping %>%

```
compensation %>%
 select(Fruit) %>%
slice(8)
```

Note that because we have the name of the data frame first we don’t need to specify it for select or slice etc.

Subsetting

filter()

Subsetting

```
compensation %>%
  filter(Root <= 10)
```

Logical Operations

- `x < y`: is x less than y?
- `x > y`: is x greater than y?
- `x <= y`: is x less than or equal to y?
- `x >= y`: is x greater than or equal to y?
- `x == y`: is x equal to y?
- `x != y`: is x not equal to y?

- Try some of these out!

Practical

1. Create a new data frame containing only rows where the Fruit values > 80.

2. Create a new dataframe containing only the Ungrazed rows.

3. Using two `dplyr` functions, and piping, create a dataframe containing the **Root** values where Fruit is greater than 80.

Creating new variables

mutate()

Creating new variables

```
compensation %>%
  mutate(Root_cm = Root/100)
```

Rearranging data

arrange()

Rearranging data

```
compensation %>%
  arrange(Fruit)
```

Use the help files to arrange compensation with Grazing in reverse alphabetical order

Renaming variables

rename()

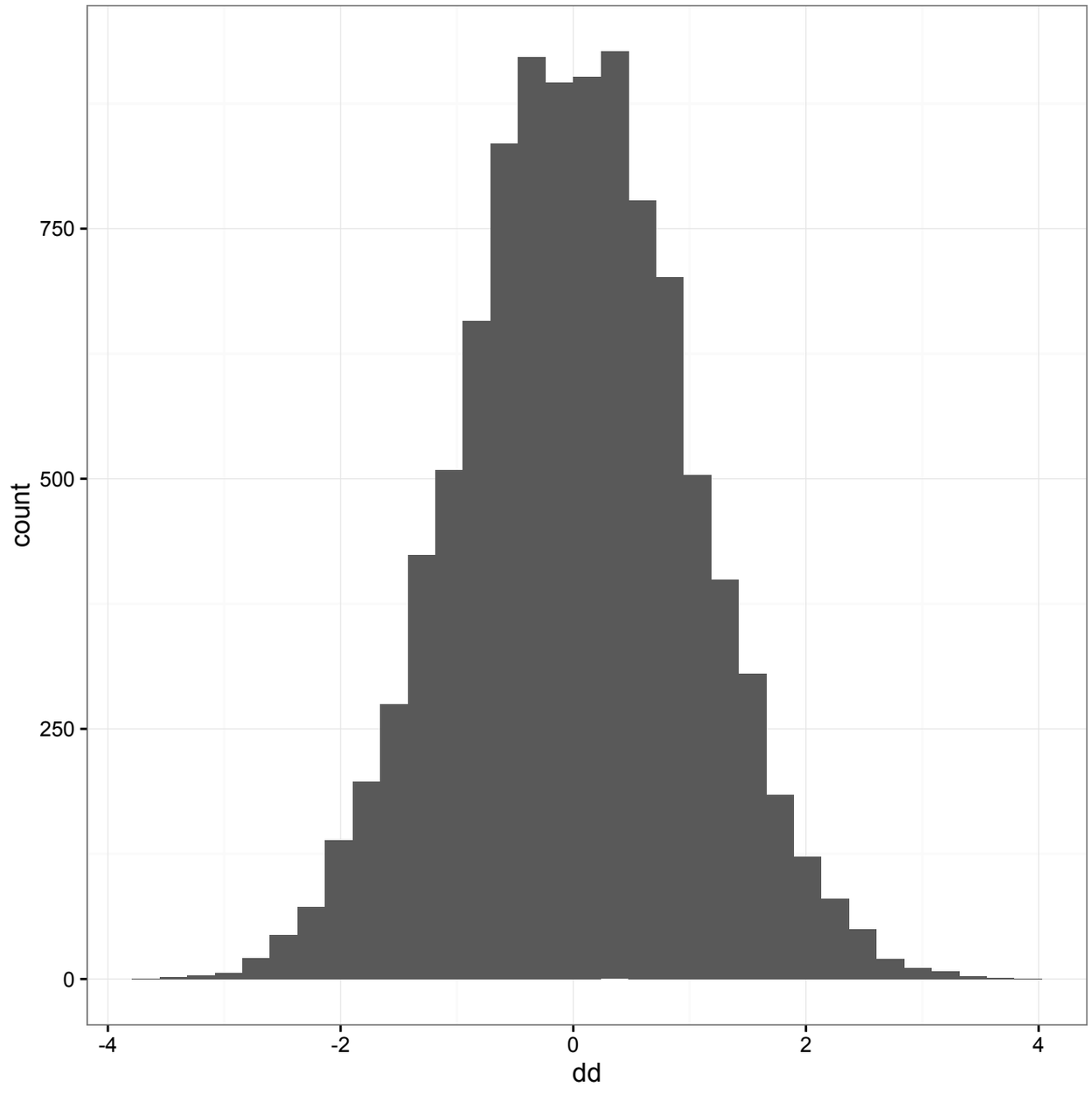
Renaming variables

```
compensation %>%
  rename(FruityMcFruitFace = Fruit)
```

tbl_df()
glimpse()
select()
slice()
filter()
mutate()
rename()
arrange()

Summarising The Data

- Mean
- SD
- Sample Size
- Standard Error



What functions in R can we use to get the following?

- mean
- median
- standard deviation
- variance
- sample size
- standard error (you might need to look up the formula on Google - there is no inbuilt function in R)

with ALL dplyr functions,

FIRST provide the name of the data frame

summarise

name of the data frame

```
compensation %>%
summarise(meanF = mean(Fruit))
```

label - this can be anything you like. To remind you what the number is

function

mean
sd
length
sum

Practical

1. What would you need to do if you wanted to use these numbers in later calculations/graphs?

2. Calculate the mean and standard deviation of the Fruit column and save it for later use.

summarise

```
summary.data <-
  compensation %>%
    summarise(meanFruit = mean(Fruit),
              sdFruit = sd(Fruit))
```

summarise by groups

```
compensation %>%
  group_by(Grazing) %>%
  summarise(meanF = mean(Fruit))
```

grouping variable

summarise by groups

```
compensation %>%
  group_by(Grazing) %>%
  summarise(meanF = mean(Fruit))
```

We can also string together lots of dplyr functions…

```
compensation %>%
  mutate(FRratio = Fruit/Root) %>%
  group_by(Grazing) %>%
  summarise(
    meanFRrat = mean(FRratio))
```

Practical

- Create a new dataframe, with a name of your choice, that contains the mean and standard error of the Fruit column, for the Grazed and Ungrazed groups.

tbl_df()
glimpse()
select()
slice()
filter()
mutate()
rename()
arrange()

summarise()
group_by()

Saving to a file

```
write.csv(
    file = “tidy-data.csv”,
        newcompensation)
```

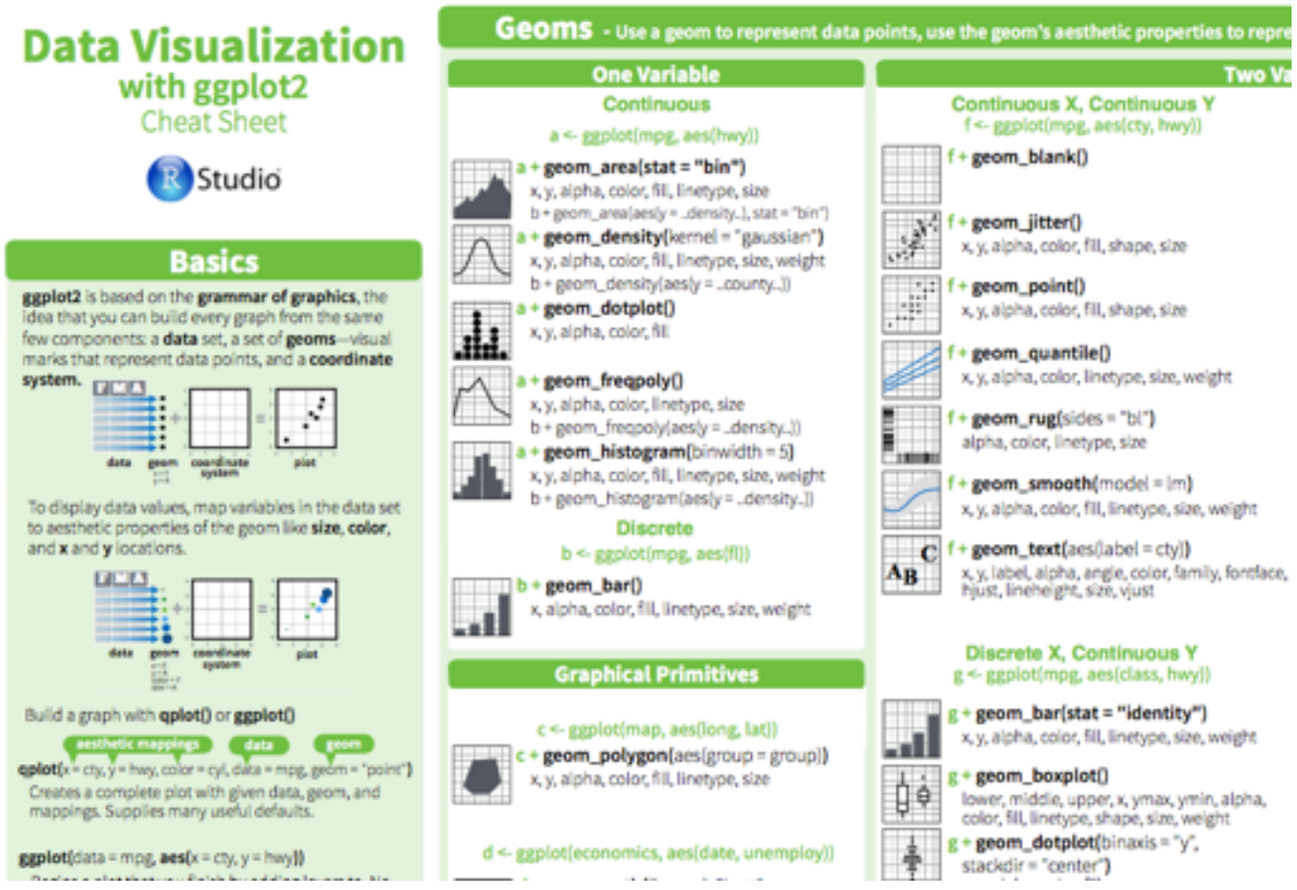
Making fancy figures with ggplot2

with ALL ggplot usage,

FIRST provide the name of the data frame

Info online is immense

- http://docs.ggplot2.org/current/index.html
- http://www.cookbook-r.com/Graphs/



Why use `ggplot2`?

Roughly speaking, there are three commonly used plotting frameworks in R.

- base graphics: available as soon as you open R
- `lattice` package: create **trellis** graphs
- `ggplot2` implements a **grammar of graphics**.

Advantages of using `ggplot2`

- Consistent and intuitive framework for plotting
- Fleixble enough to make every plot you will need
- Works well with `dplyr`

Disadvantages of using `ggplot2`

- You have to learn "the grammar" to use it well
- Vast package, can be intimidating
- More than one way to do things

data frame

aesthetic components

geometric objects

data frame

dataframe!

aesthetic components

Like the X and Y axes

geometric objects

Like the points and lines or bars

Key concepts

You need to wrap your head around a few ideas to start using `ggplot2` effectively:

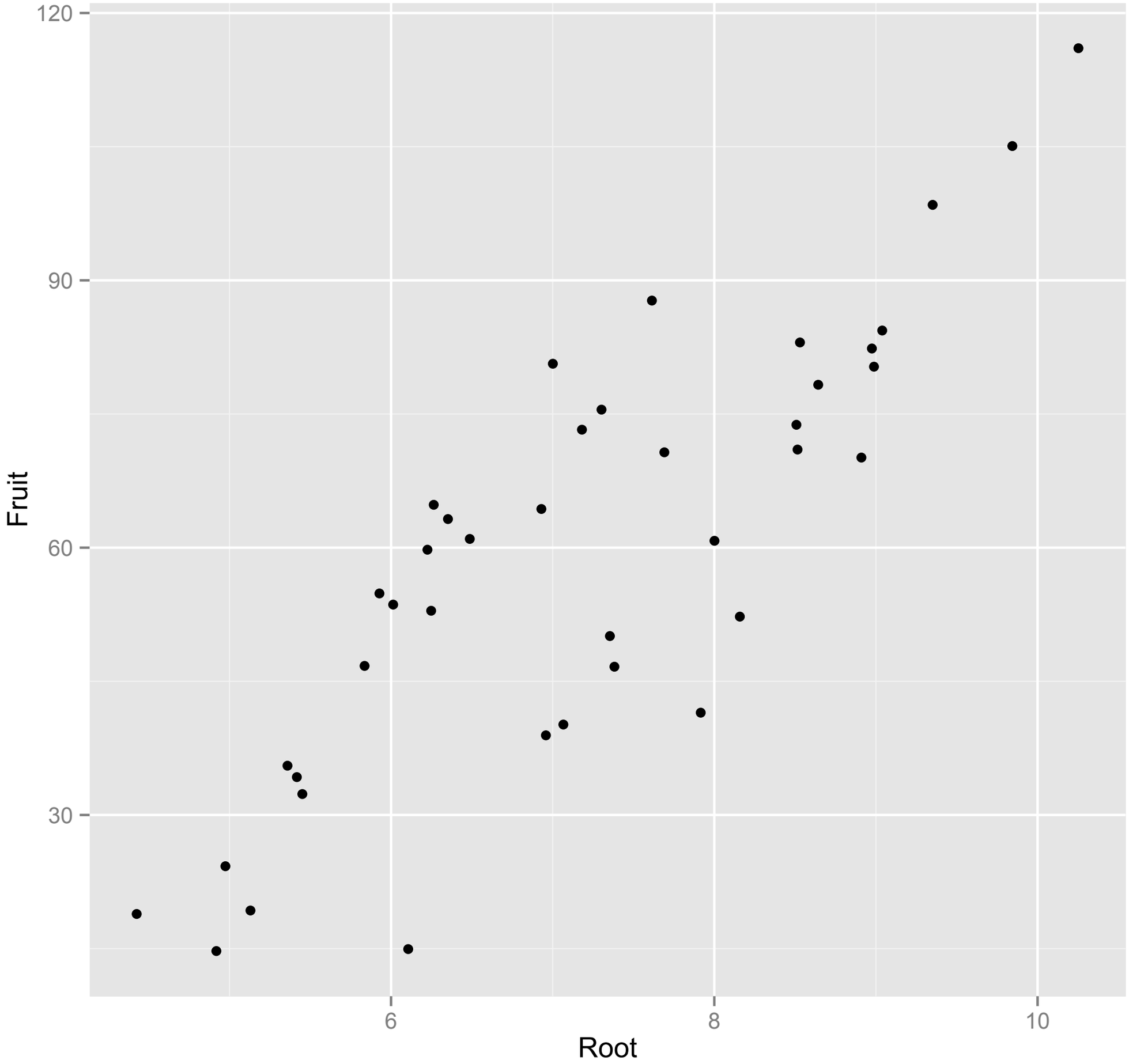
- **layers**: We build ggplot2 objects by adding different kinds of layers together. In practise we do this with the `+` operator. We can do this in a stepwise way, only plotting the object when we are ready
- **aesthetics**: The word aesthetics refers to **the information** in a plot. For example, which variables are associated with the x and y axes? We specify this using the `aes` function.
- **geometric objects**: Geometric objects ("geoms") determine how the information is displayed. For example, will it be a scatter plot or a bar plot? We specify geoms using functions beginning with `geom_`.

First Argument is the dataset

Second Argument are the 'aesthetics': what we want ggplot to use to construct the axes of the plot

```
ggplot(data.frame, aes(x = , y = ))+
geom_point()
```

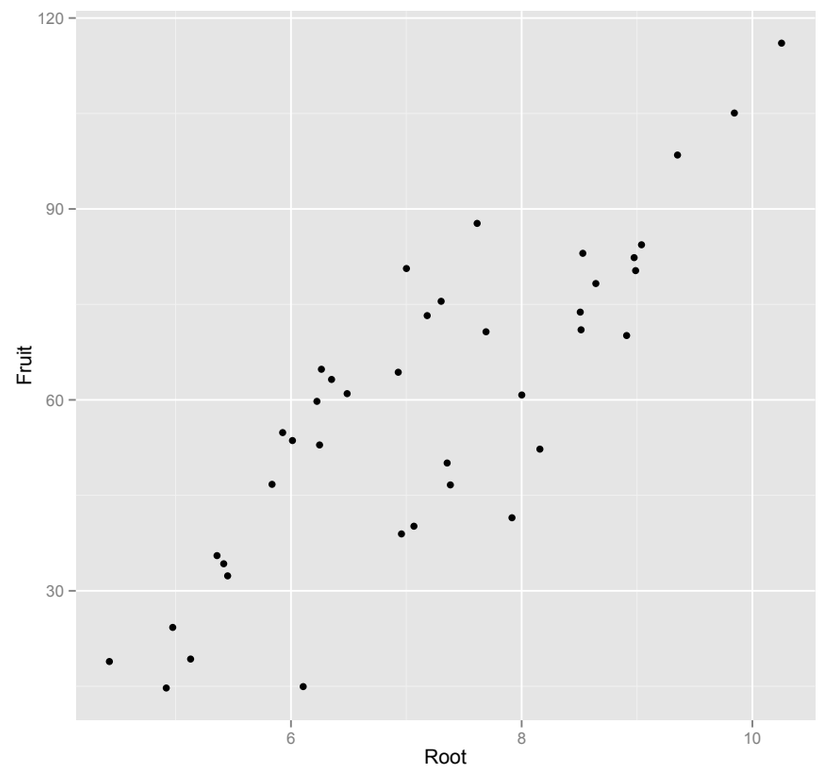
The First '**layer**' to add to the plot


120
90
60
30
Fruit
6
8
10
Root

Practical

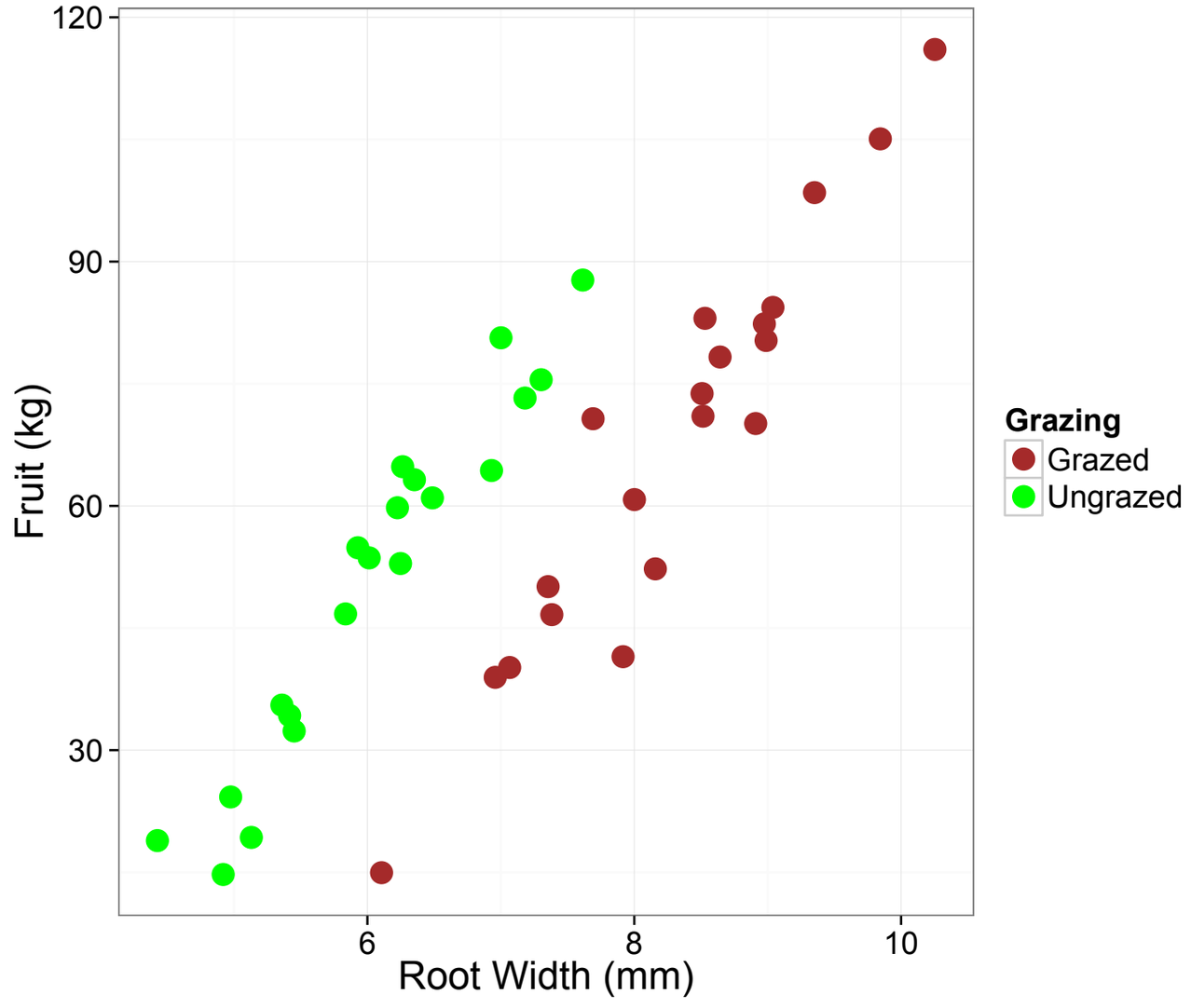
Prepare a new script: plottingExercises
Annotate the start of it
load dplyr, ggplot2 & gridExtra libraries
read in (again) the compensation.csv data
reproduce this figure

```
ggplot(data.frame, aes(x = , y = )) +
  geom_point()
```



What do we want to change?

- Hate the grey grid
- axis labels need modifying
- axis ranges
- point character and colours



theme_bw

xlab(), ylab(), xlim(), ylim()

ggtitle()

labs()

scale_x OR scale_y

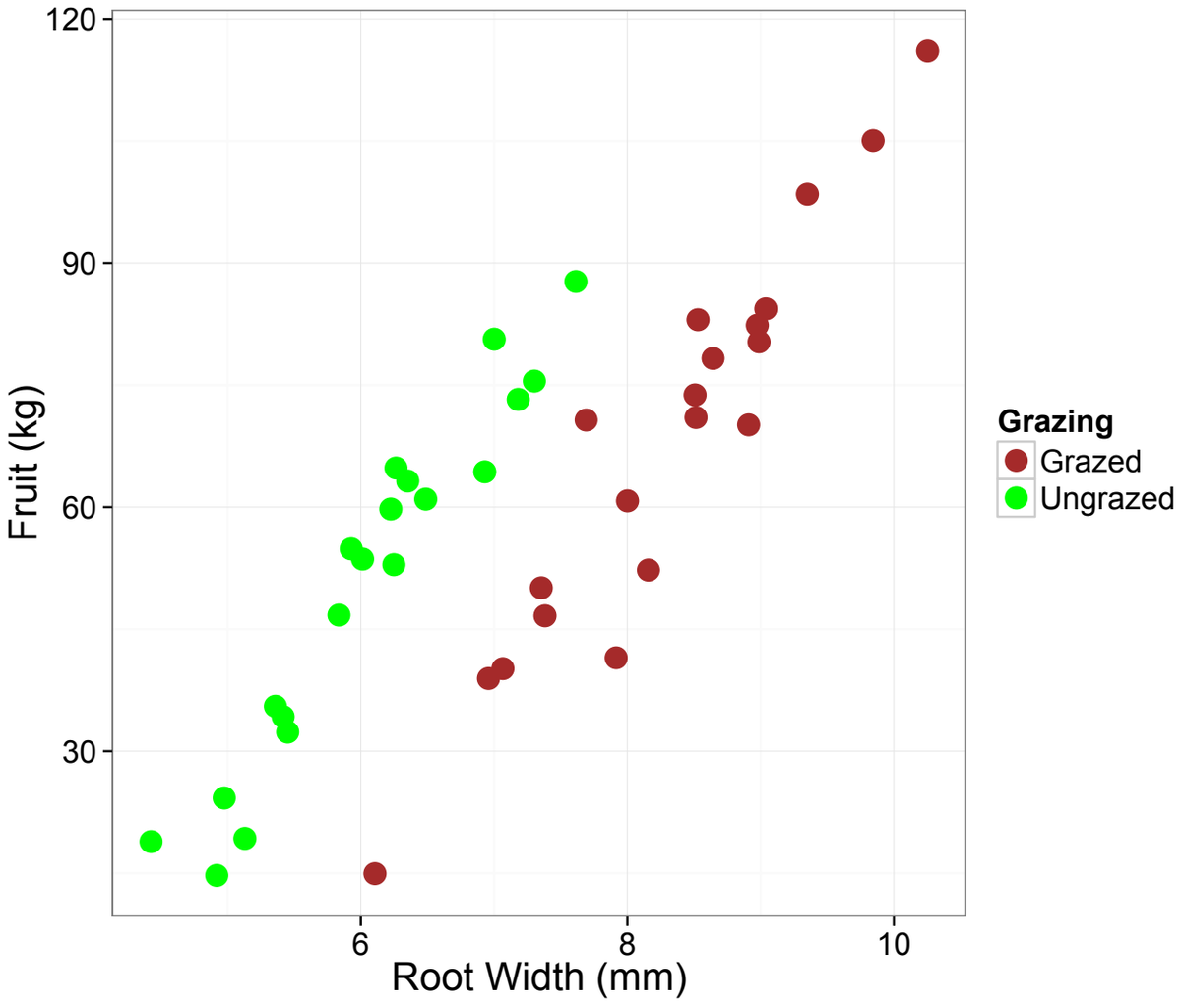
```
scale_x_continuous(
    limits = c(),
    breaks = c(),
    labels = c())

scale_y_continuous()
```

scale_colour_

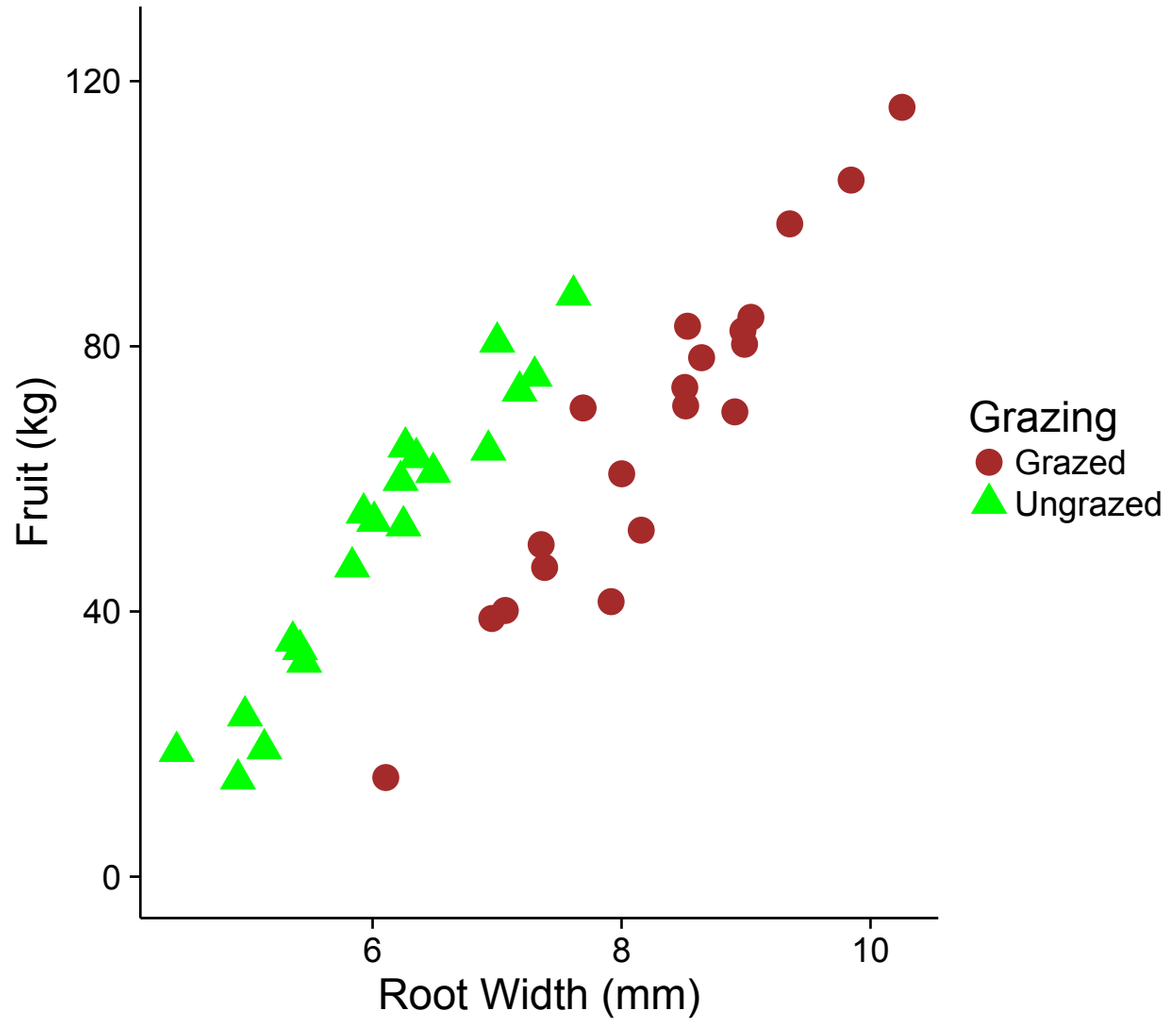
```
scale_colour_manual(values = c(   ))

scale_fill_manual(values = c(   ))
```

theme()

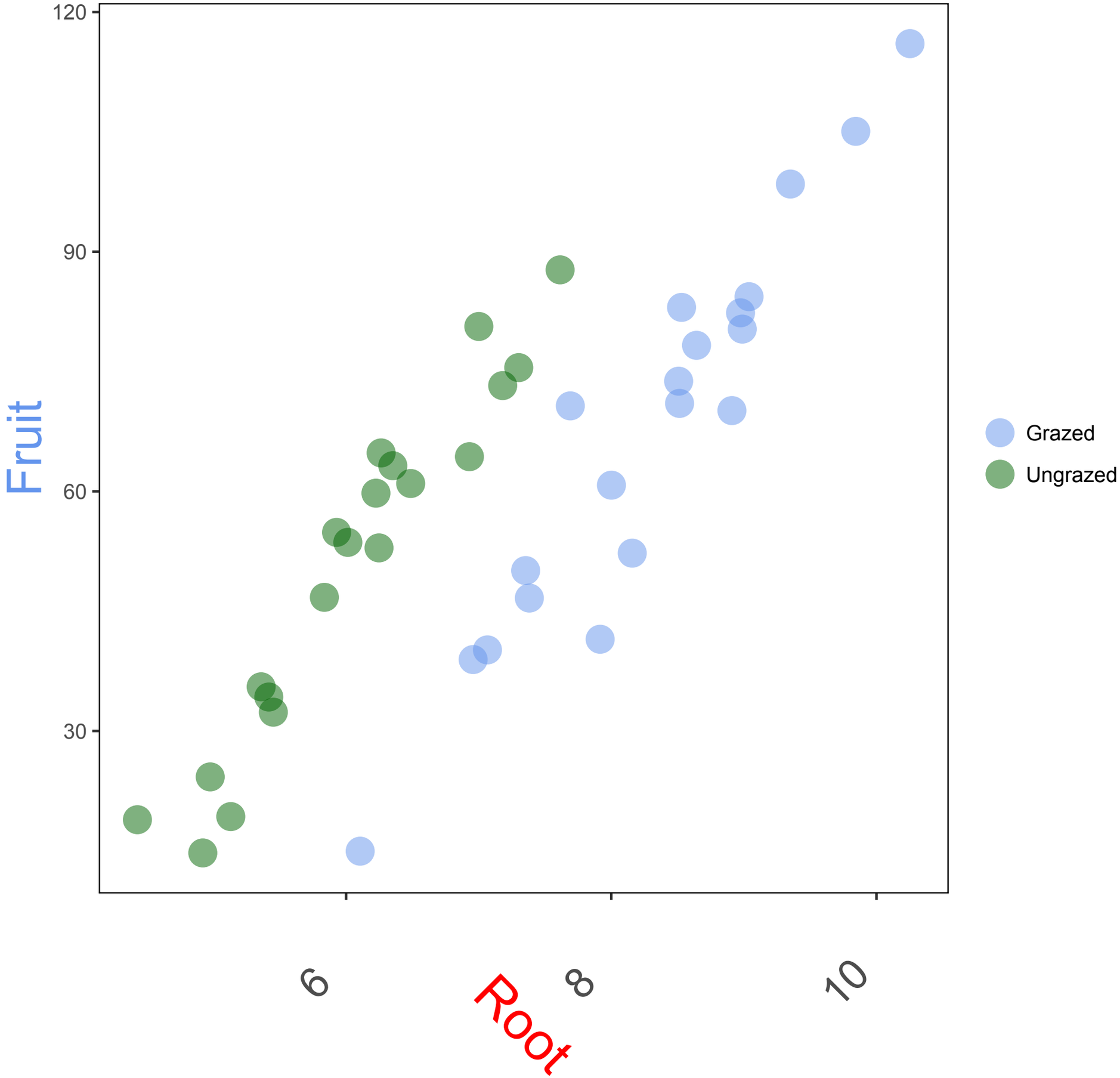
colours
theme_bw()



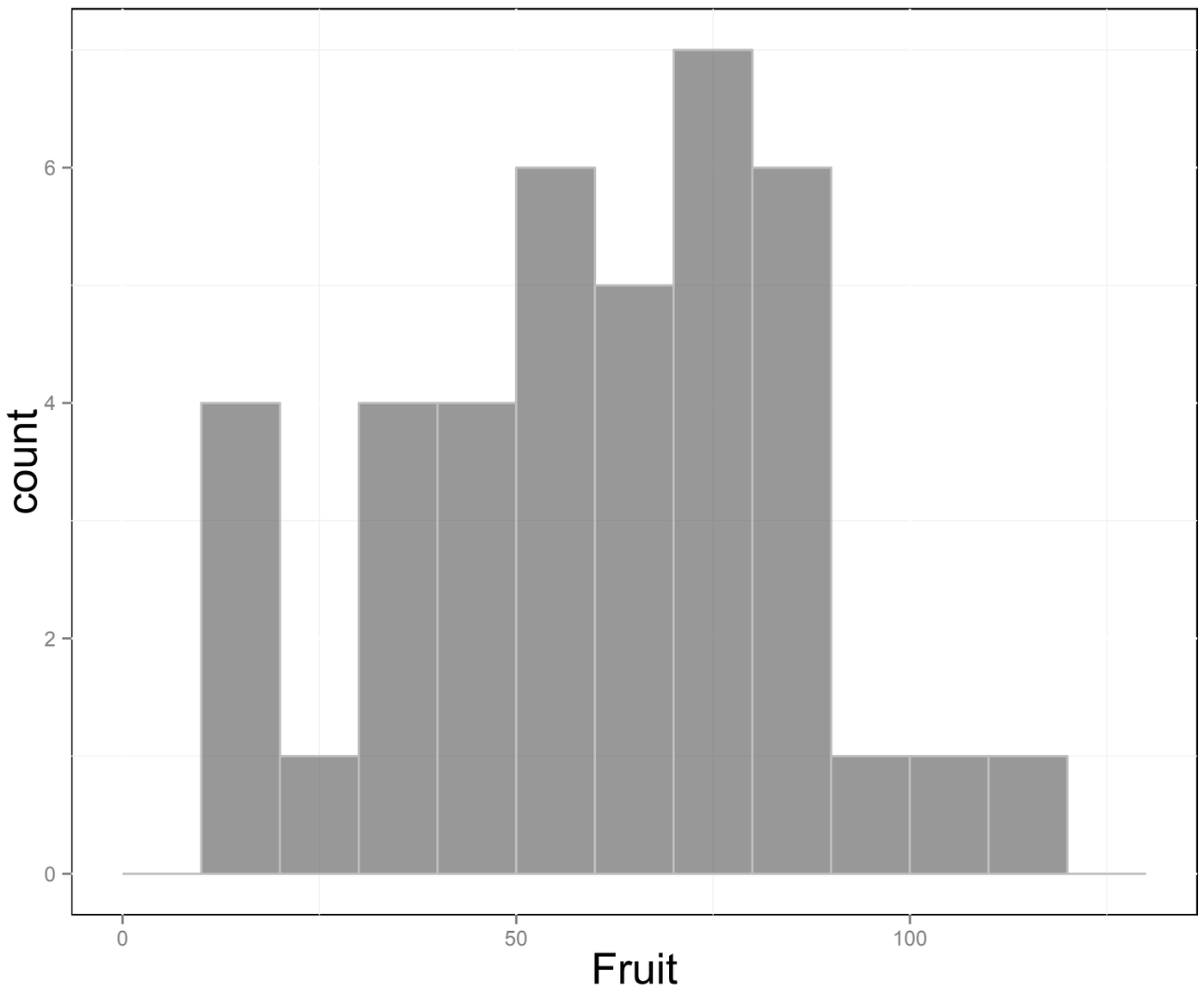
colours
shapes
theme_classic()



```r
ggplot(compensation, aes(x=Root, y = Fruit, colour = Grazing))+
geom_point(size = 5, alpha = 0.5)+
scale_colour_manual(values = c("cornflowerblue","darkgreen")) +
scale_x_continuous(breaks = 2:10)+
ggtitle("a")+
theme(
  # get rid of grid and colour background
  # fill with nothing, black border
  panel.background = element_rect(fill = NA, colour = 'black'),
  panel.grid = element_blank(),

  # adjust the x & y axis details
  axis.title.x=element_text(size=rel(2), angle = -45, colour = 'red'),
  axis.title.y=element_text(size=20, colour = 'cornflowerblue'),

  # adjust the axis tick label size, rotation, and position
  axis.text.x = element_text(size = rel(2), angle = 45, vjust = -1),

  # get rid of the legend title (make it blank)
  legend.title=element_blank(),

  # get rid of boxes around points in legend
  legend.key=element_rect(fill=NA),

  # title justify left
  plot.title = element_text(hjust = 0))
```


120
90
60
30
Fruit
6
8
10
Root
Grazed
Ungrazed

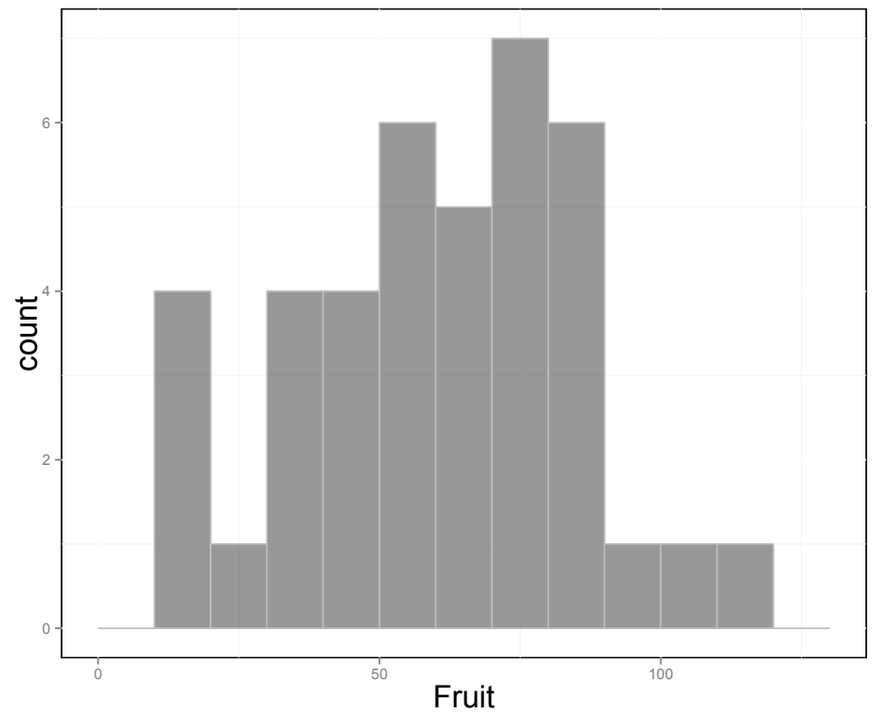
geom_histogram



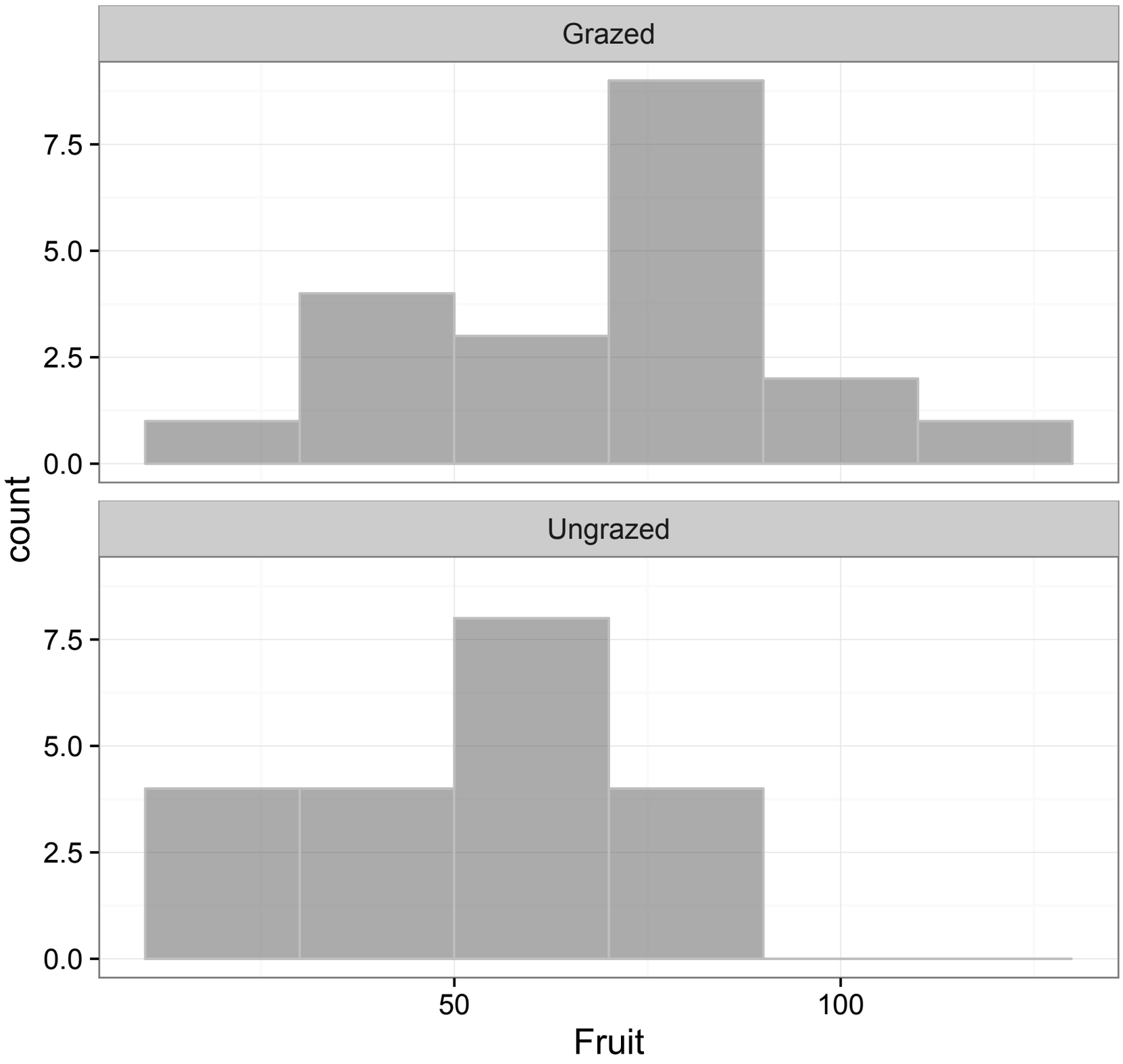
Practical:

Modify this to make the histogram….

```
ggplot(data.frame, aes(x = , y = )) +
 geom_point()
```

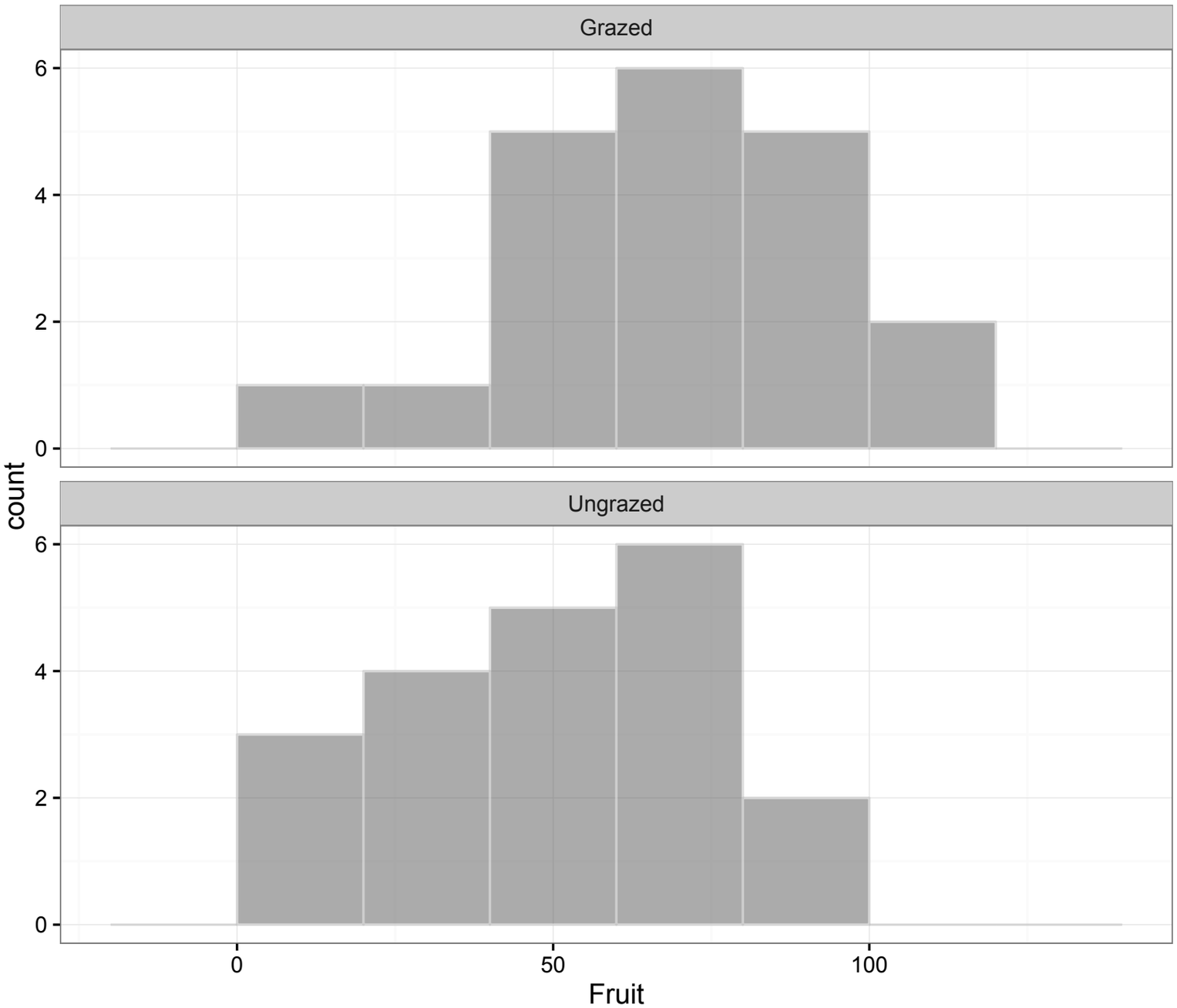


histograms by grazing



+facet_wrap(~
Grazing)

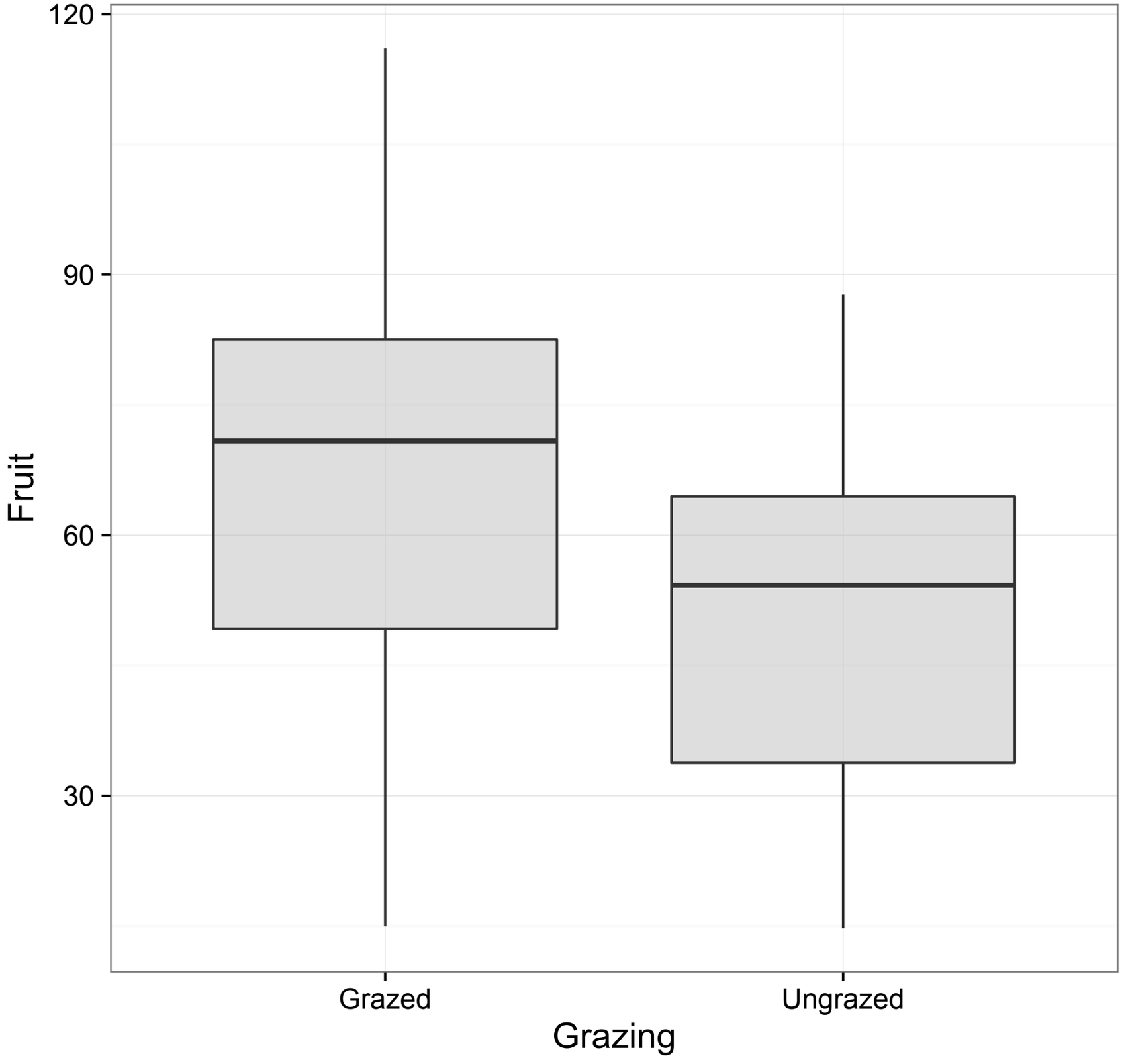
histograms by grazing



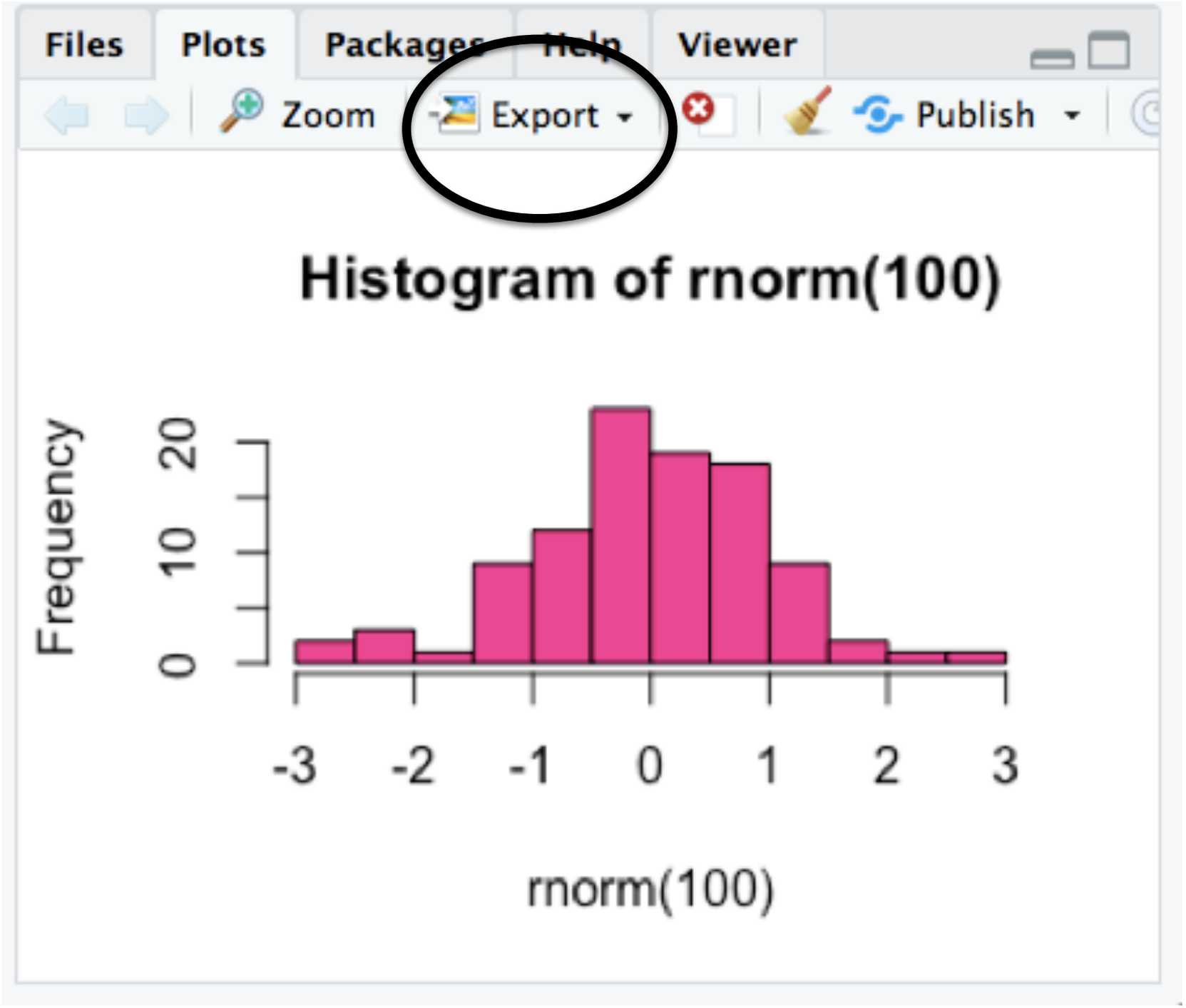

```
# histograms and facets
ggplot(compensation, aes(x=Fruit, col = "grey", alpha = 0.5))+
    geom_histogram(binwidth = 20)+
    facet_wrap(~Grazing, ncol = 1)
```

boxplot by Grazing?

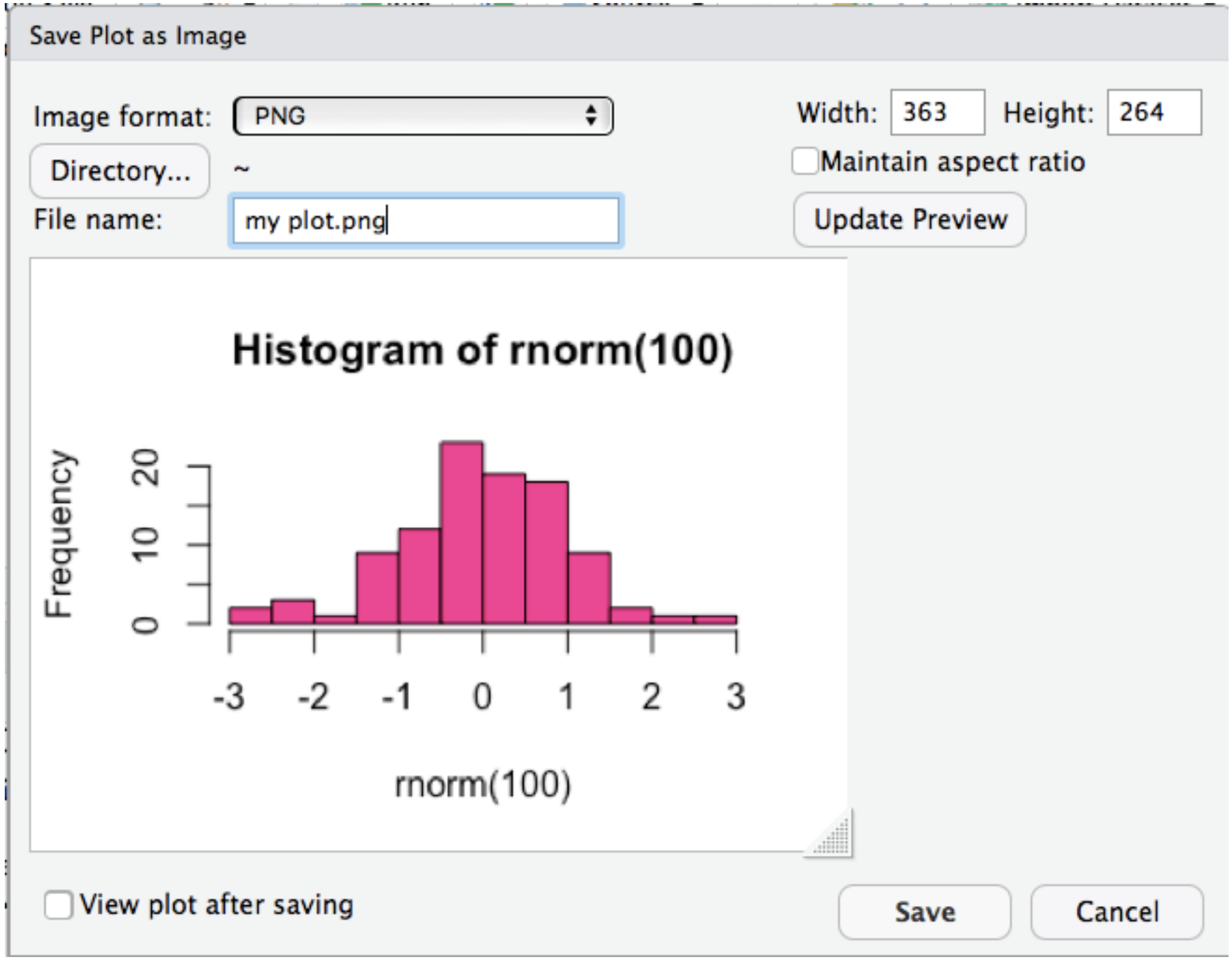
geom_?????


120
90
60
30
Fruit
Grazed
Ungrazed
Grazing

Saving graphs…



Saving graphs…



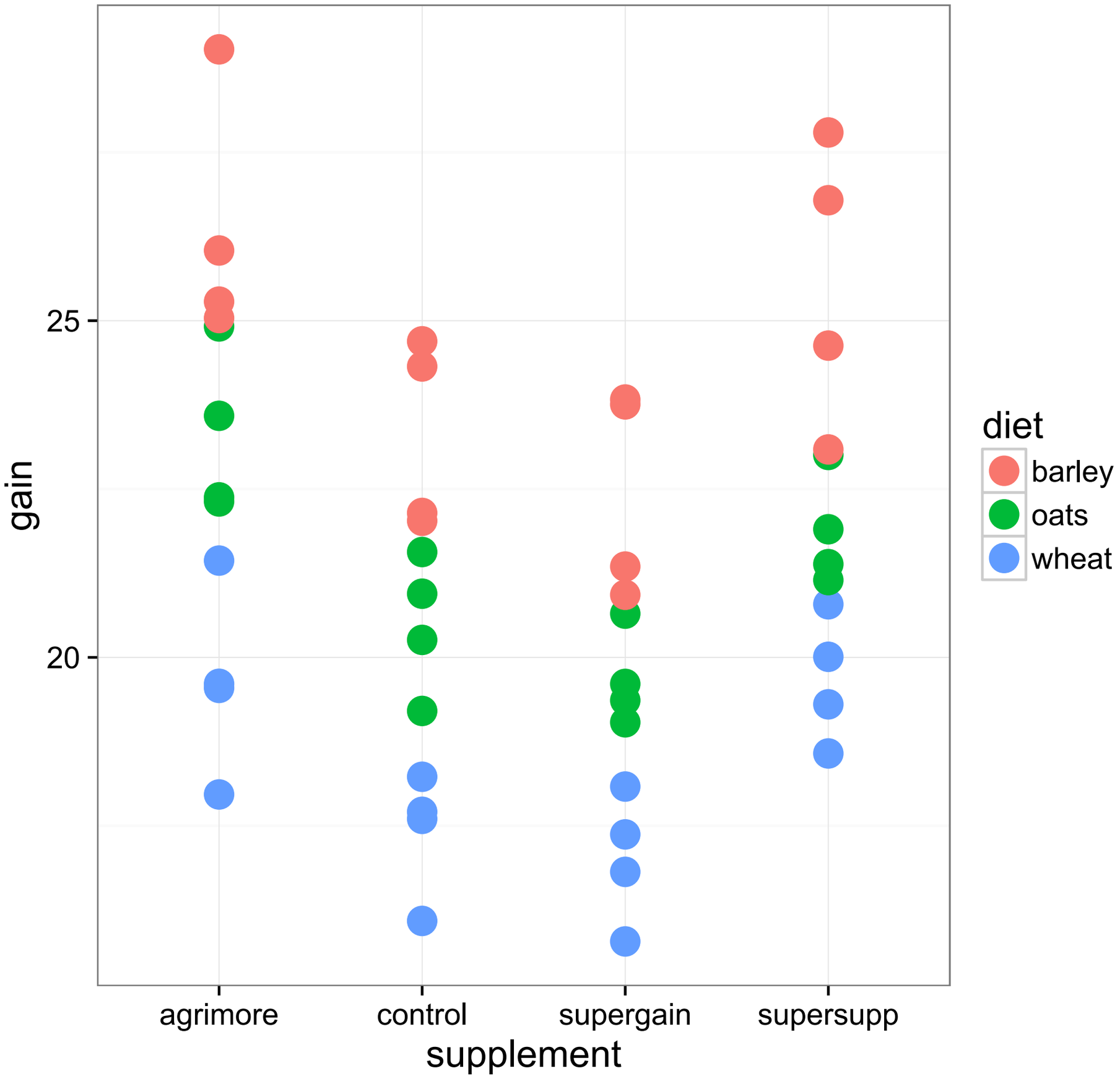
dplyr + ggplot2

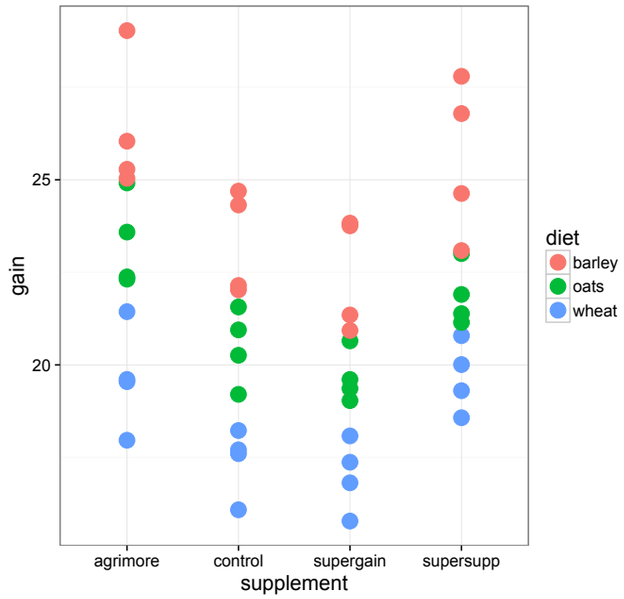
Practical

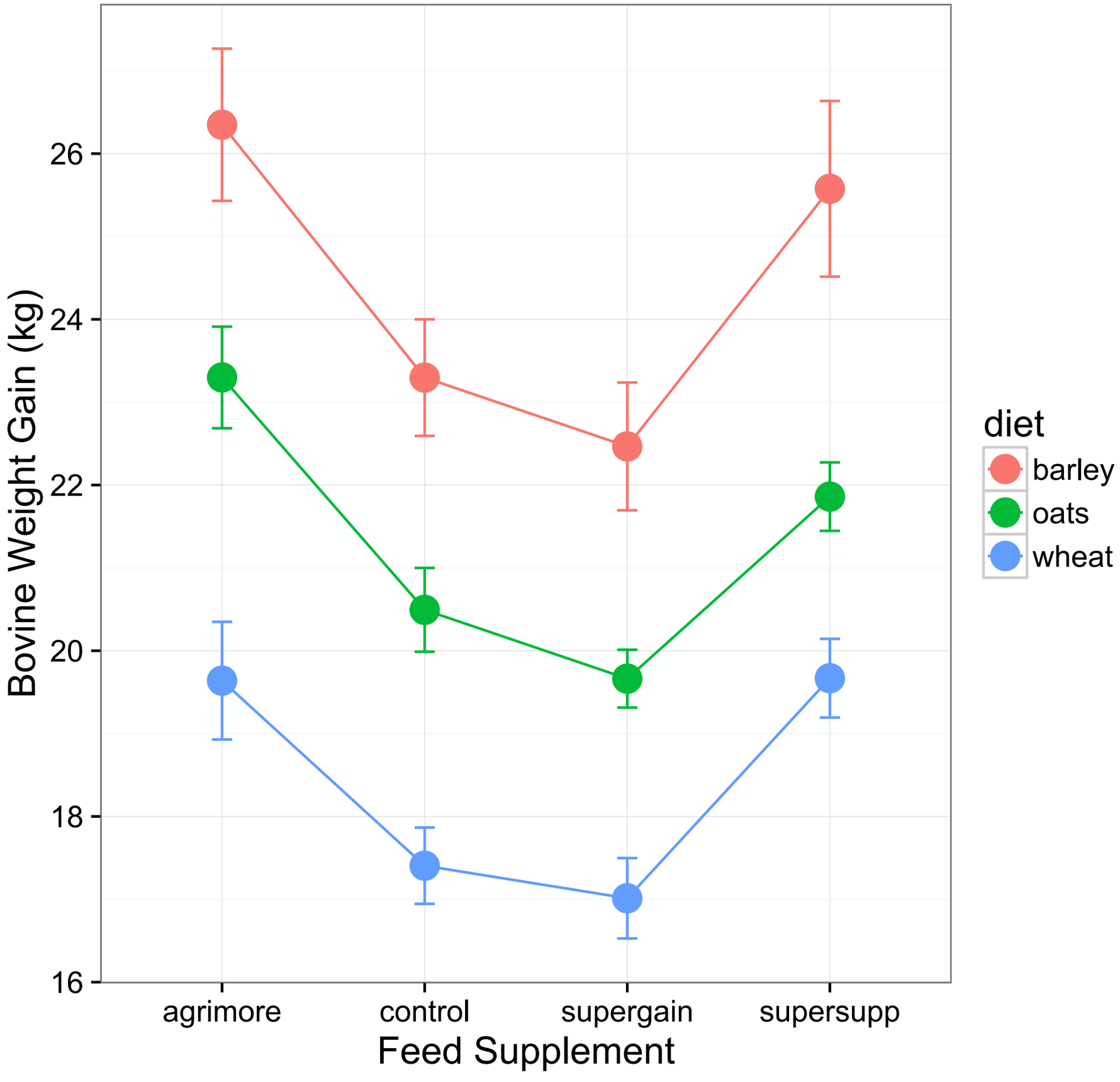
1. import the growth.csv data
2. explore the data - bovine weight gain

    2.1.tbl_df, str, glimpse

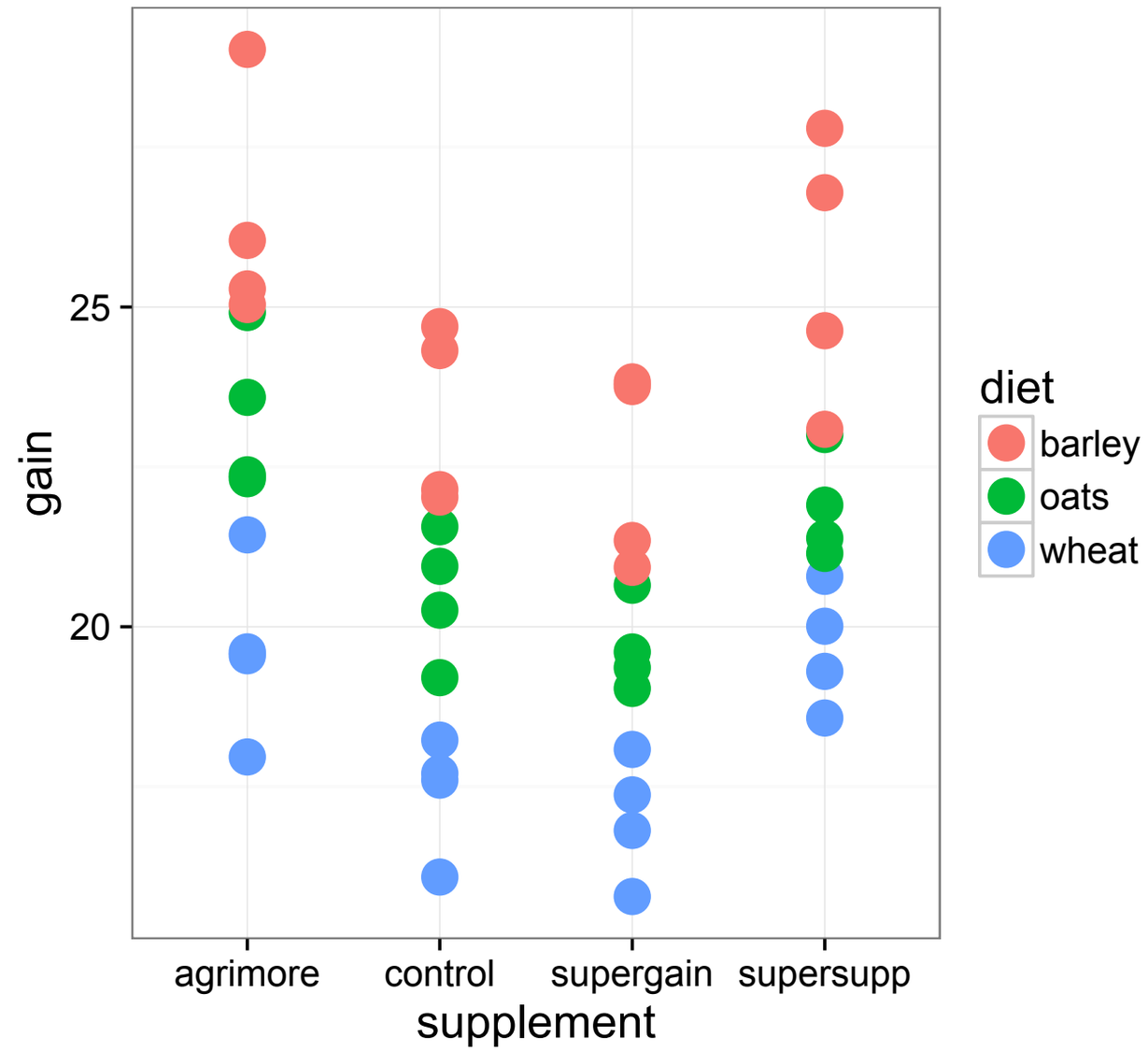
3. Make the following Graph

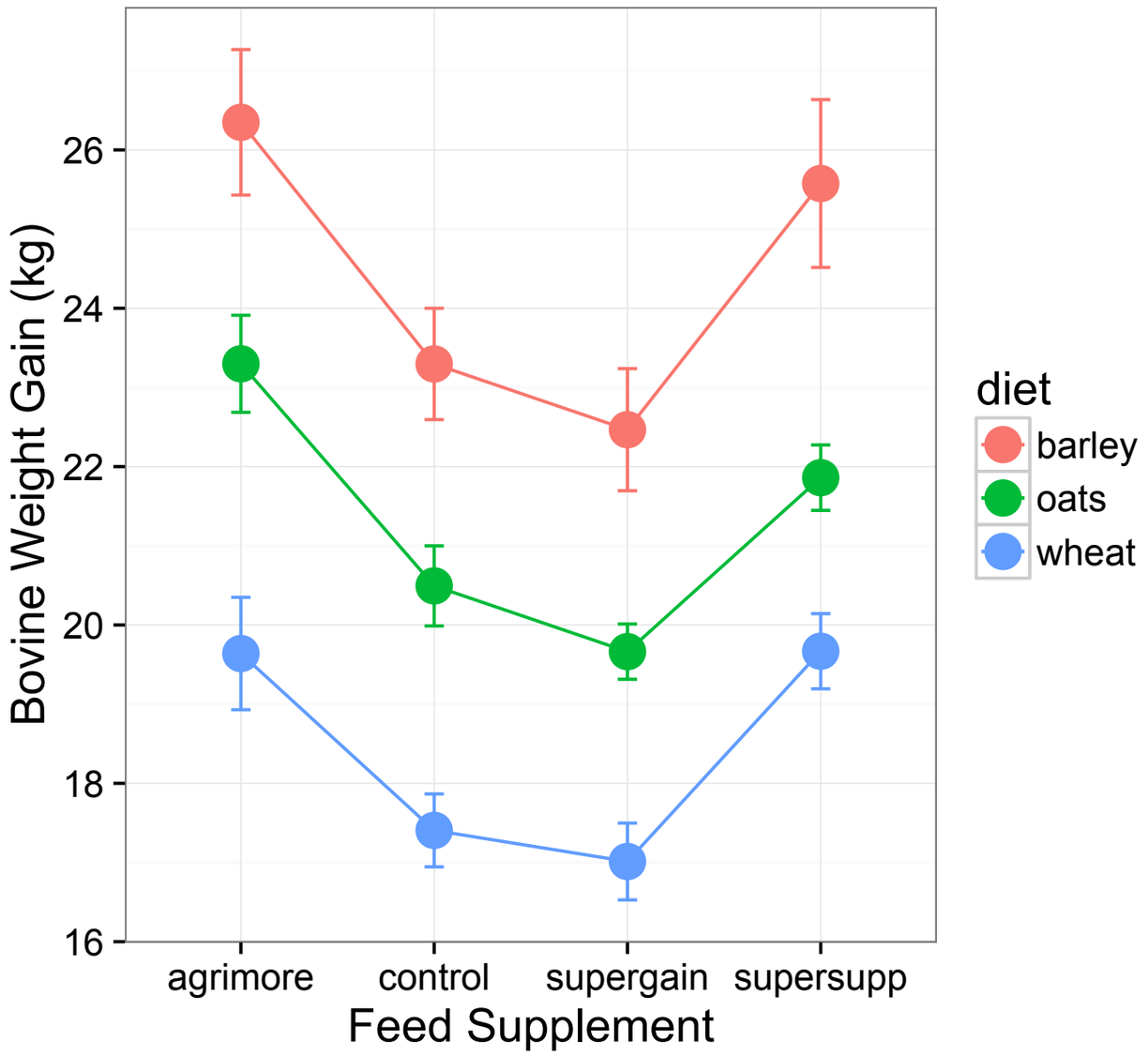

25
20
gain
agrimore
control
supergain
supersupp
supplement
diet
barley
oats
wheat


gain
25
20
agrimore
control
supergain
supersupp
supplement
diet
barley
oats
wheat


Bovine Weight Gain (kg)
26
24
22
20
18
16
agrimore
control
supergain
supersupp
Feed Supplement
diet
barley
oats
wheat

```r
# Now, lets use dplyr to create a summary data frame
# means and se's

sumGrow<-summarise(group_by(grow, supplement, diet),
 meanGrow = mean(gain),
 seGrow = sd(gain)/sqrt(n()))

# NOW... lets plot these data
# y is now meanGrow
# groups!!!

ggplot(sumGrow, aes(x = supplement, y = meanGrow, group = diet, colour = diet))+
 geom_point(size = 5)+
 geom_line()+
 geom_errorbar(aes(ymin = meanGrow - seGrow, ymax = meanGrow + seGrow), width = 0.1)+
 ylab("Bovine Weight Gain (kg)")+
 xlab("Feed Supplement")+
 theme_bw(base_size = 15)
```

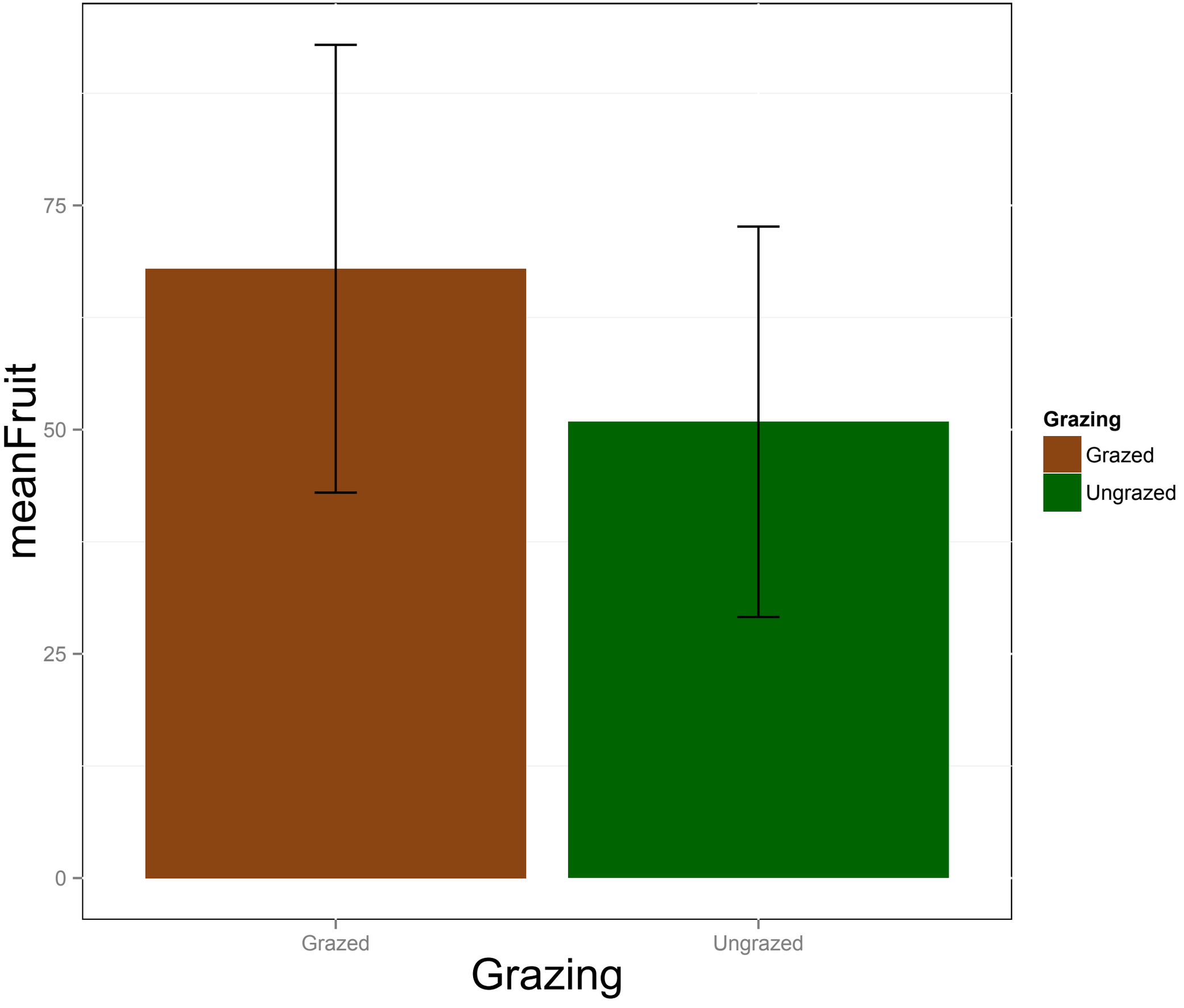

25
20
gain
agrimore
control
supergain
supersupp
supplement
diet
barley
oats
wheat


26
24
22
20
18
16
Bovine Weight Gain (kg)
agrimore
control
supergain
supersupp
Feed Supplement
diet
barley
oats
wheat

Kill the bar charts!!!

PLOS Biology: Beyond Bar and Line Graphs: Time for a New Data Presentation Paradigm

compensation.csv


75
50
25
0
meanFruit
Grazed
Ungrazed
Grazing
Grazing
Grazed
Ungrazed

calculate mean and standard error for each group + lower and upper bounds of bars using summarise()

Step 1

```
# barplot with errorbars
sumDat<-summarise(group_by(compensation, Grazing),
   meanFruit = mean(Fruit),
   seFruit = sqrt(var(Fruit)/n()),
   nFruit = length(Fruit),
   lwr = meanFruit - sdFruit,
   upr = meanFruit + sdFruit)
```

you can do
calculations
WITHIN summarise

Step 2

This is something we will use to build the bars…
WHAT IS IT?

```
limits<-aes(ymin = lwr, ymax = upr)
```

Step 3

The "bars" layer

```
ggplot(sumDat, aes(x=Grazing, y=meanFruit, fill=Grazing))+
   geom_bar(stat = "identity")
```

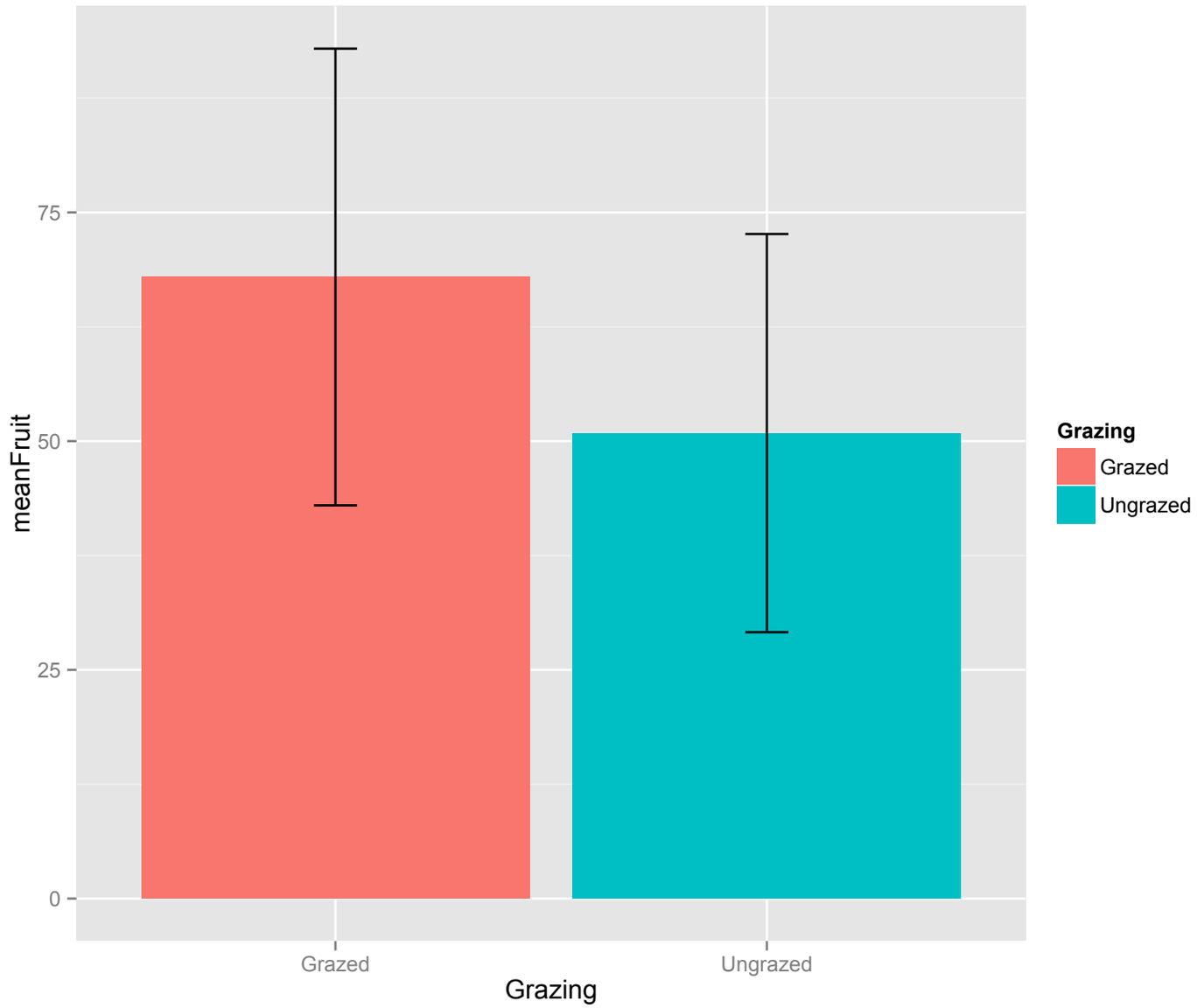
Step 4

The "error bars" layer

```
ggplot(sumDat, aes(x=Grazing, y=meanFruit, fill=Grazing))+
   geom_bar(stat = "identity")+
   geom_errorbar(aes(ymin = lwr, ymax = upr), width = 0.1)
```

Step 5

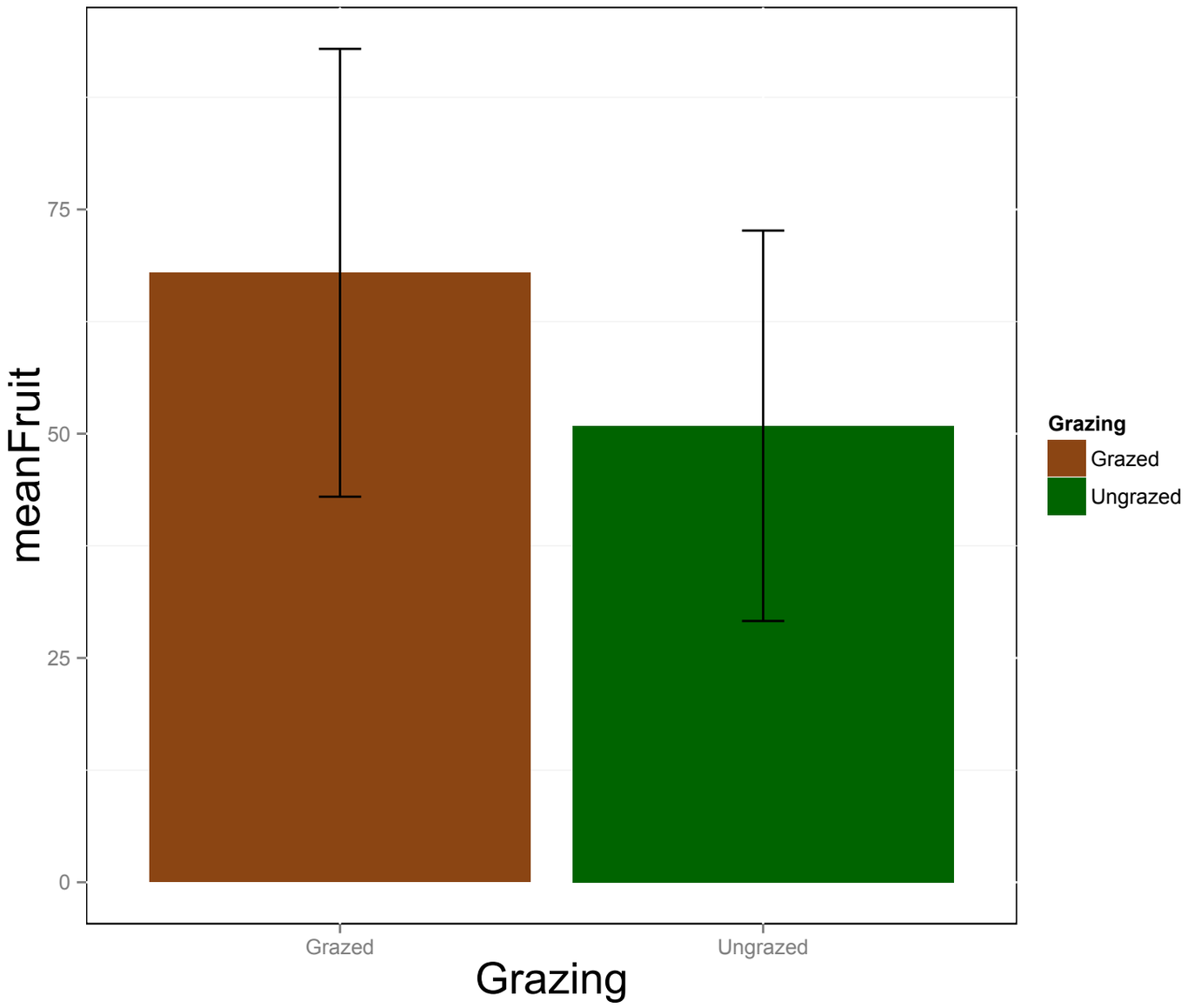
The "result"



Step 6
The Customising

```
ggplot(sumDat, aes(x=Grazing, y=meanFruit, fill=Grazing))+
   geom_bar(stat = "identity")+
   geom_errorbar(limits, width = 0.1)+
   scale_fill_manual(values = c("chocolate4","darkgreen"))+
   theme(panel.background = element_rect(fill = "white",
   colour = "black"),
    axis.title.x=element_text(size=20),
    axis.title.y=element_text(size=20))
```

Step 6
The Customising

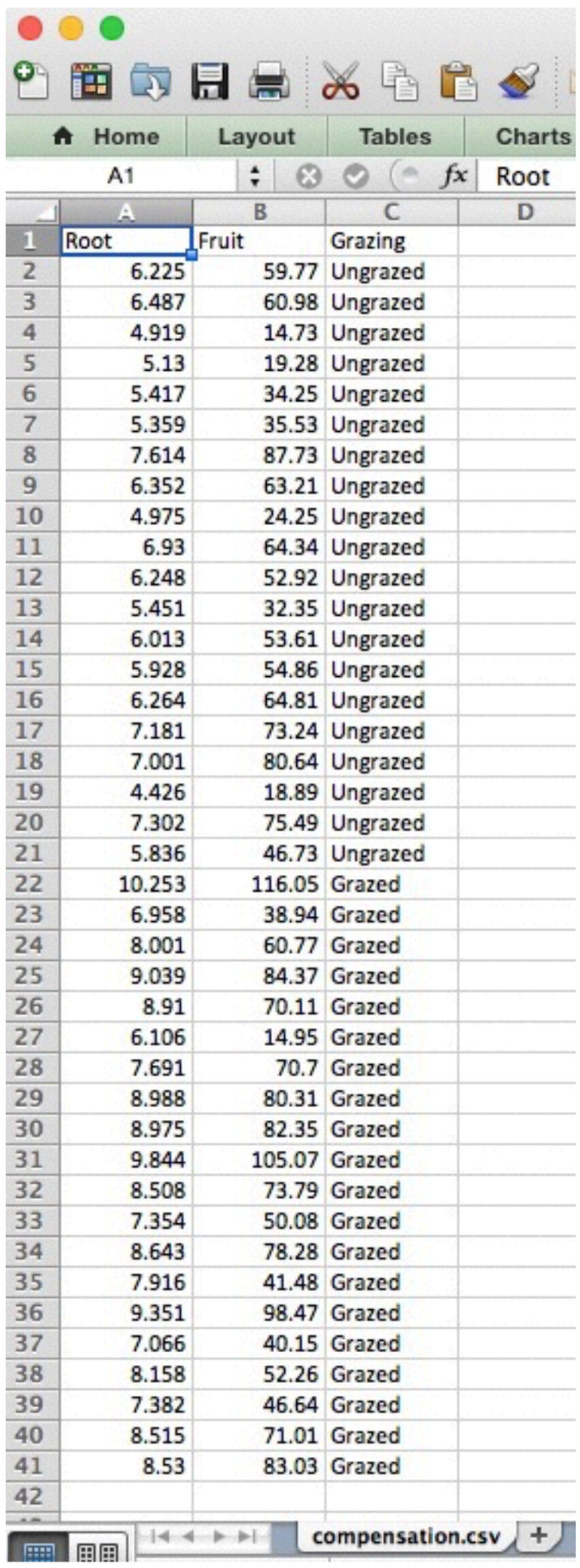


Quick note on the tidyverse

- dplyr, tidyr, lubridate, ggplot2 and others were built by Hadley Wickham and colleagues.
- All have a very similar structure e.g. the data frame is always first, they work well with %>%.
- Most other packages do not work like this!
- When using other packages or base R remember that the position of the data frame will change etc.

Tidying data with tidyr and lubridate

R4AII 2016



Home | Layout | Tables | Charts

A1 | Root

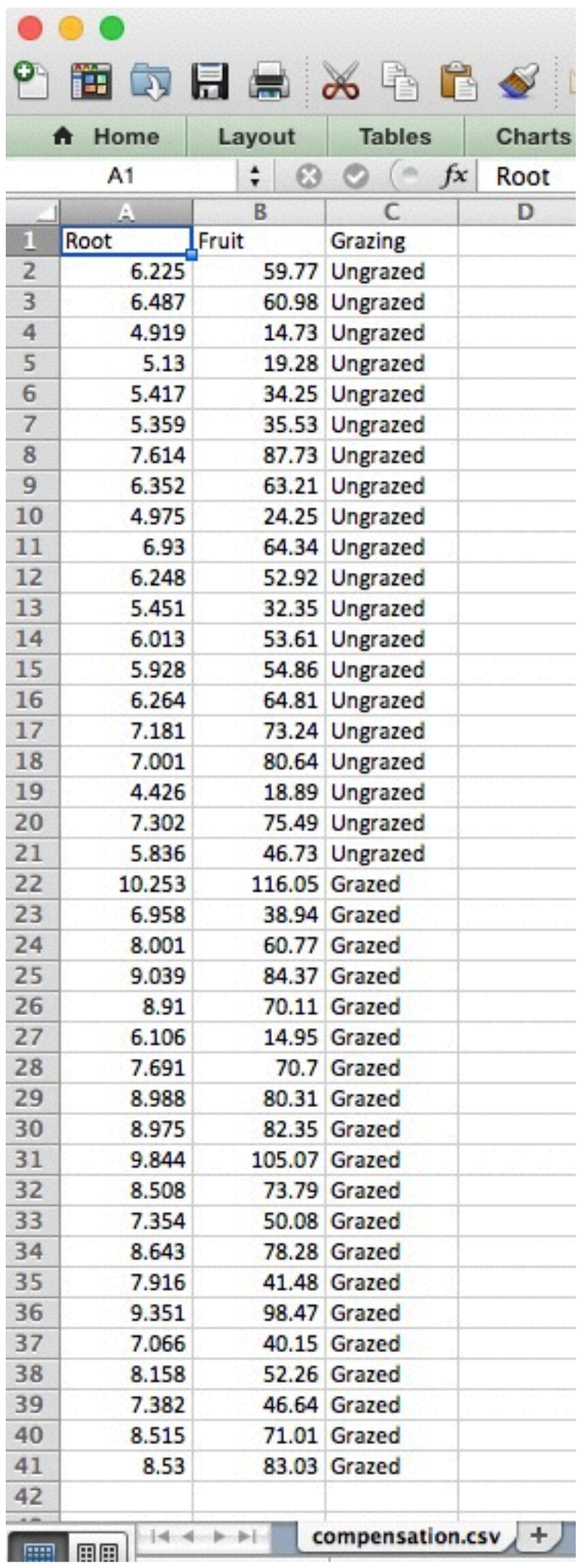
	A	B	C	D
1	Root	Fruit	Grazing	
2	6.225	59.77	Ungrazed	
3	6.487	60.98	Ungrazed	
4	4.919	14.73	Ungrazed	
5	5.13	19.28	Ungrazed	
6	5.417	34.25	Ungrazed	
7	5.359	35.53	Ungrazed	
8	7.614	87.73	Ungrazed	
9	6.352	63.21	Ungrazed	
10	4.975	24.25	Ungrazed	
11	6.93	64.34	Ungrazed	
12	6.248	52.92	Ungrazed	
13	5.451	32.35	Ungrazed	
14	6.013	53.61	Ungrazed	
15	5.928	54.86	Ungrazed	
16	6.264	64.81	Ungrazed	
17	7.181	73.24	Ungrazed	
18	7.001	80.64	Ungrazed	
19	4.426	18.89	Ungrazed	
20	7.302	75.49	Ungrazed	
21	5.836	46.73	Ungrazed	
22	10.253	116.05	Grazed	
23	6.958	38.94	Grazed	
24	8.001	60.77	Grazed	
25	9.039	84.37	Grazed	
26	8.91	70.11	Grazed	
27	6.106	14.95	Grazed	
28	7.691	70.7	Grazed	
29	8.988	80.31	Grazed	
30	8.975	82.35	Grazed	
31	9.844	105.07	Grazed	
32	8.508	73.79	Grazed	
33	7.354	50.08	Grazed	
34	8.643	78.28	Grazed	
35	7.916	41.48	Grazed	
36	9.351	98.47	Grazed	
37	7.066	40.15	Grazed	
38	8.158	52.26	Grazed	
39	7.382	46.64	Grazed	
40	8.515	71.01	Grazed	
41	8.53	83.03	Grazed	
42				

compensation.csv

Data in our examples are tidy…

Real data are often not tidy!

	A	B	C	D	E	F
1	Grazed			Ungrazed		
2	Root	Fruit		Root	Fruit	
3	10.253	116.05		6.225	59.77	
4	6.958	38.94		6.487	60.98	
5	8.001	60.77		4.919	14.73	
6	9.039	84.37		5.13	19.28	
7	8.91	70.11		5.417	34.25	
8	6.106	14.95		5.359	35.53	
9	7.691	70.7		7.614	87.73	
10	8.988	80.31		6.352	63.21	
11	8.975	82.35		4.975	24.25	
12	9.844	105.07		6.93	64.34	
13	8.508	73.79		6.248	52.92	
14	7.354	50.08		5.451	32.35	
15	8.643	78.28		6.013	53.61	
16	7.916	41.48		5.928	54.86	
17	9.351	98.47		6.264	64.81	
18	7.066	40.15		7.181	73.24	
19	8.158	52.26		7.001	80.64	
20	7.382	46.64		4.426	18.89	
21	8.515	71.01		7.302	75.49	
22	8.53	83.03		5.836	46.73	
23						



	A	B	C	D
1	Root	Fruit	Grazing	
2	6.225	59.77	Ungrazed	
3	6.487	60.98	Ungrazed	
4	4.919	14.73	Ungrazed	
5	5.13	19.28	Ungrazed	
6	5.417	34.25	Ungrazed	
7	5.359	35.53	Ungrazed	
8	7.614	87.73	Ungrazed	
9	6.352	63.21	Ungrazed	
10	4.975	24.25	Ungrazed	
11	6.93	64.34	Ungrazed	
12	6.248	52.92	Ungrazed	
13	5.451	32.35	Ungrazed	
14	6.013	53.61	Ungrazed	
15	5.928	54.86	Ungrazed	
16	6.264	64.81	Ungrazed	
17	7.181	73.24	Ungrazed	
18	7.001	80.64	Ungrazed	
19	4.426	18.89	Ungrazed	
20	7.302	75.49	Ungrazed	
21	5.836	46.73	Ungrazed	
22	10.253	116.05	Grazed	
23	6.958	38.94	Grazed	
24	8.001	60.77	Grazed	
25	9.039	84.37	Grazed	
26	8.91	70.11	Grazed	
27	6.106	14.95	Grazed	
28	7.691	70.7	Grazed	
29	8.988	80.31	Grazed	
30	8.975	82.35	Grazed	
31	9.844	105.07	Grazed	
32	8.508	73.79	Grazed	
33	7.354	50.08	Grazed	
34	8.643	78.28	Grazed	
35	7.916	41.48	Grazed	
36	9.351	98.47	Grazed	
37	7.066	40.15	Grazed	
38	8.158	52.26	Grazed	
39	7.382	46.64	Grazed	
40	8.515	71.01	Grazed	
41	8.53	83.03	Grazed	
42				

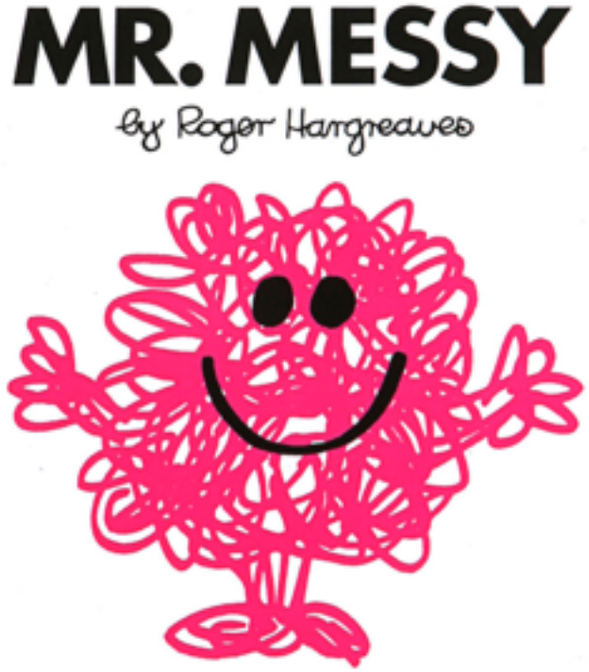
What is tidy data?

- Data that is formatted so that doing further analyses etc. with it is easy.

1. Each variable forms a column.
2. Each observation forms a row.
3. Each type of observational unit forms a table.

What is messy data?

1. Column headers are values, not variable names.
2. Multiple variables are stored in one column.
3. Variables are stored in both rows and columns.
4. Multiple types of observational units are stored in the same table.
5. A single observational unit is stored in multiple tables.



Home | Layout | Tables | Charts

A1 | Root

	A	B	C	D
1	Root	Fruit	Grazing	
2	6.225	59.77	Ungrazed	
3	6.487	60.98	Ungrazed	
4	4.919	14.73	Ungrazed	
5	5.13	19.28	Ungrazed	
6	5.417	34.25	Ungrazed	
7	5.359	35.53	Ungrazed	
8	7.614	87.73	Ungrazed	
9	6.352	63.21	Ungrazed	
10	4.975	24.25	Ungrazed	
11	6.93	64.34	Ungrazed	
12	6.248	52.92	Ungrazed	
13	5.451	32.35	Ungrazed	
14	6.013	53.61	Ungrazed	
15	5.928	54.86	Ungrazed	
16	6.264	64.81	Ungrazed	
17	7.181	73.24	Ungrazed	
18	7.001	80.64	Ungrazed	
19	4.426	18.89	Ungrazed	
20	7.302	75.49	Ungrazed	
21	5.836	46.73	Ungrazed	
22	10.253	116.05	Grazed	
23	6.958	38.94	Grazed	
24	8.001	60.77	Grazed	
25	9.039	84.37	Grazed	
26	8.91	70.11	Grazed	
27	6.106	14.95	Grazed	
28	7.691	70.7	Grazed	
29	8.988	80.31	Grazed	
30	8.975	82.35	Grazed	
31	9.844	105.07	Grazed	
32	8.508	73.79	Grazed	
33	7.354	50.08	Grazed	
34	8.643	78.28	Grazed	
35	7.916	41.48	Grazed	
36	9.351	98.47	Grazed	
37	7.066	40.15	Grazed	
38	8.158	52.26	Grazed	
39	7.382	46.64	Grazed	
40	8.515	71.01	Grazed	
41	8.53	83.03	Grazed	
42				

compensation.csv

Home | Layout | Tables | Charts | SmartArt | Formulas | Data

C31

	A	B	C	D	E	F	G
1	Ungrazed				Grazed		
2	Root	Fruit	Grazing		Root	Fruit	Grazing
3	6.225	59.77	Ungrazed		10.253	116.05	Grazed
4	6.487	60.98	Ungrazed		6.958	38.94	Grazed
5	4.919	14.73	Ungrazed		8.001	60.77	Grazed
6	5.13	19.28	Ungrazed		9.039	84.37	Grazed
7	5.417	34.25	Ungrazed		8.91	70.11	Grazed
8	5.359	35.53	Ungrazed		6.106	14.95	Grazed
9	7.614	87.73	Ungrazed		7.691	70.7	Grazed
10	6.352	63.21	Ungrazed		8.988	80.31	Grazed
11	4.975	24.25	Ungrazed		8.975	82.35	Grazed
12	6.93	64.34	Ungrazed		9.844	105.07	Grazed
13	6.248	52.92	Ungrazed		8.508	73.79	Grazed
14	5.451	32.35	Ungrazed		7.354	50.08	Grazed
15	6.013	53.61	Ungrazed		8.643	78.28	Grazed
16	5.928	54.86	Ungrazed		7.916	41.48	Grazed
17	6.264	64.81	Ungrazed		9.351	98.47	Grazed
18	7.181	73.24	Ungrazed		7.066	40.15	Grazed
19	7.001	80.64	Ungrazed		8.158	52.26	Grazed
20	4.426	18.89	Ungrazed		7.382	46.64	Grazed
21	7.302	75.49	Ungrazed		8.515	71.01	Grazed
22	5.836	46.73	Ungrazed		8.53	83.03	Grazed
23							
24							

Why bother tidying?

- Make analyses, graphs, and data manipulation much easier
- Data will already be in the correct format for use with many packages, including dplyr and ggplot2

http://blog.rstudio.org/2014/12/08/tidyr-0-2-0/

Now work with dealing-with-data-tidyr.Rmd

- Work along with me (to save you having to type everything). There are periodic practicals.

We can use tidyr to tidy messy data

1. Read in the **two** museum datasets: museum1.csv and museum2.csv
2. Use dplyr to look at the data. Try and identify what we need to tidy!

Issue #1: multiple columns for one variable

Species <chr>	MammalTower <int>	SpiritCollection <int>	Warehouse <int>
Niffler	10	0	6
Occamy	1	9	1
Kelpie	13	3	1
Demiguise	12	1	5
Dragon	1	1	4
Hinkypuff	1	6	3

Gather columns together

gather()

with ALL tidyr usage,

FIRST provide the name of the data frame

Gather columns together

new column name for grouping variable

```
museum1 %>%
  gather(Location, Number,
         MammalTower:Warehouse)
```

names of the columns to gather

new column name for values

Remember: to use the modified datasets you need to assign them to something…

```
tidy.museum <-
  museum1 %>%
    gather(Location, Number,
           MammalTower:Warehouse)
```

What if we want to split our columns?

Species <chr>	Location <chr>	Number <int>
Niffler	MammalTower	10
Occamy	MammalTower	1
Kelpie	MammalTower	13
Demiguise	MammalTower	12
Dragon	MammalTower	1
Hinkypuff	MammalTower	1
Murtlap	MammalTower	1
Phoenix	MammalTower	3
Thunderbird	MammalTower	1
Thestral	MammalTower	7

Spread columns (usually for multivariate analyses)

spread()

Spread columns

column to spread

```
tidy.museum %>%
spread(Location, Number)
```

names of column with the observations in it

Practical

- Tidy `museum1` and arrange with `Species` in reverse alphabetical order (clue - this will require a `dplyr` function).

Issue #2: multiple variables in one column

museum2.csv

SpecimenID <chr>	Species <chr>	CollectionDate <chr>	Status <chr>
NHM.10.1.6.7	Niffler	12/03/1901	male juvenile
NHM.10.1.6.8	Occamy	15/08/1992	female adult
NHM.10.1.6.8	Occamy	15/08/1992	female adult
NHM.10.1.6.9	Kelpie	14/12/1902	male adult
NHM.10.1.6.10	Demiguise	01/01/1900	male adult
NHM.10.1.6.11	Dragon	08/09/1954	male adult

Separate columns

```
tidy.museum2 <-
  museum2 %>%
  separate(Status,
  c("Sex","Age"), sep = " ")
```

column to separate

names of the new columns.
c = combine
names must be in “”

what pattern tells you when to separate.
(here it’s a space)

Other types of separators…

- “ ” - space
- “_” - underscore
- “.” - period
- “,” - comma
- “\t” - tab
- Many many more using regular expressions (see ?regex for more examples)

What if we want to combine columns instead?

Unite columns

name of new united column

```
tidy.museum2 %>%
  unite(Sex_Age,
  c(Sex, Age), sep = "_")
```

names columns to unite - these do not need "" because they are column names

how to separate the entries. here an underscore (_)

gather()
separate()

vs.

spread()
unite()

Tidying data also involves extra data cleaning

- Identifying missing values (use NA rather than blanks).
- Fixing dates and times (see next slides).
- Making sure data are of the correct type (see next slides)
- Correcting typos.
- Model based cleaning to identify incorrect values.

Times and dates - a special kind of messy…

1. Use dplyr to sort museum2 by CollectionDate
2. What do you notice?
3. What type of variable does R think CollectionDate is (hint - use tbl_df)

Remember to install and load the lubridate library… it isn't part of the tidyverse

```
library(lubridate)
```

Converting (parsing) dates and times so R reads them correctly

```
dmy("01/12/1998")
dmy("01-12-1998")
dmy("01.12.1998")
```

day month year

It is possible to parse any kind of date and time

- dmy()
- ymd()
- mdy()
- hms()
- ymd_hms()
- now() - gives the date and time now!

Create a new variable using dplyr and lubridate

```
new.museum2 <-
  museum2 %>%
  mutate(DateCollected =
           dmy(CollectionDate)) %>%
  select(-CollectionDate)
```

Sort new.museum2 by DateCollected. What kind of variable is it now?

If you use dates and times a lot check out:
http://www.noamross.net/blog/2014/2/10/using-times-and-dates-in-r---presentation-code.html

Practical

- Using museum2, make a new dataframe called tidy.museum, tidied as much as possible using the tools you've just learned.

Ideal workflow

- Import your raw data. NEVER change this in Excel.*
- Clean/tidy it using tidyr, lubridate etc. You then have a script so you can easily repeat the tidying in the future!
- Use this tidied data for all further analyses.

*We'd recommend checking your data before importing for typos etc.

```
write_csv(dataset_name,
path = "filename.csv")
```

organising your data?

There are **many** other things you can do to wrangle data with the tidyverse

- remove duplicate rows (?distinct)
- randomly sample data (?sample_n)
- merge data frames together (?join)
- join columns and rows to data frames (?bind)
- set operations (?setdiff, ?union, ?intersect)

Data Wrangling
with dplyr and tidyr
Cheat Sheet

RStudio

Tidy Data - A foundation for wrangling in R

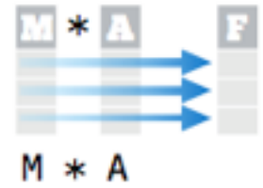
In a tidy data set:

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

&

Each **observation** is saved in its own **row**

Tidy data complements R's **vectorized operations**. R will automatically preserve observations as you manipulate variables. No other format works as intuitively with R.



Syntax - Helpful conventions for wrangling

dplyr::**tbl_df(iris)**

Converts data to tbl class. tbl's are easier to examine than data frames. R displays only the data that fits onscreen:

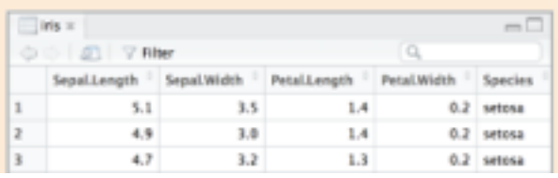
```
Source: local data frame [150 x 5]

   Sepal.Length Sepal.Width Petal.Length
1           5.1         3.5          1.4
2           4.9         3.0          1.4
3           4.7         3.2          1.3
4           4.6         3.1          1.5
5           5.0         3.6          1.4
..          ...         ...          ...
Variables not shown: Petal.Width (dbl),
  Species (fctr)
```

dplyr::**glimpse(iris)**

Information dense summary of tbl data.

utils::**View(iris)**

View data set in spreadsheet-like display (note capital V).



Reshaping Data - Change the layout of a data set

tidyr::**gather(cases, "year", "n", 2:4)**

Gather columns into rows.

tidyr::**spread(pollution, size, amount)**

Spread rows into columns.

tidyr::**separate(storms, date, c("y", "m", "d"))**

Separate one column into several.

tidyr::**unite(data, col, ..., sep)**

Unite several columns into one.

dplyr::**data_frame(a = 1:3, b = 4:6)**

Combine vectors into data frame (optimized).

dplyr::**arrange(mtcars, mpg)**

Order rows by values of a column (low to high).

dplyr::**arrange(mtcars, desc(mpg))**

Order rows by values of a column (high to low).

dplyr::**rename(tb, y = year)**

Rename the columns of a data frame.

Subset Observations (Rows)

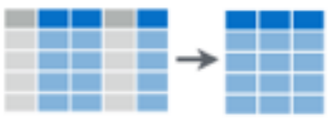
dplyr::**filter(iris, Sepal.Length > 7)**

Extract rows that meet logical criteria.

dplyr::**distinct(iris)**

Remove duplicate rows

Subset Variables (Columns)



dplyr::**select(iris, Sepal.Width, Petal.Length, Species)**

Select columns by name or helper function.

Helper functions for select - ?select

Quick note on the Hadley-Verse

- dplyr, tidyr, lubridate, ggplot2 and others were built by Hadley Wickham.
- All have a very similar structure e.g. the data frame is always first, they work well with %>%.
- Most other packages do not work like this!
- When using other packages or base R remember that the position of the data frame will change etc.