Intro to R

Data Visualization

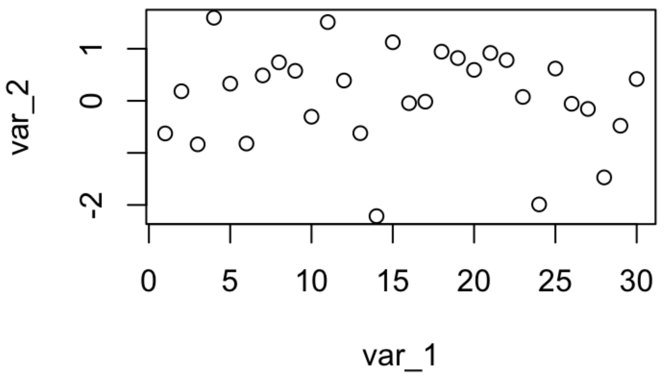
Overview

- Making plots with base R
- Making plots with `ggplot2` package
  - Main concept: “linking” data with the plot elements
  - First plot
  - Specifying plot layers: line, point, boxplot, …
  - Customizing plot look: title, axis labels, background colors, …
  - Combining multiple plots
  - Saving a plot to file

Making plots with base R

- Pros: the fastest way to make a plot
- Cons: limits for plot customization

```r
var_1 <- seq(from = 1, to = 30)
var_2 <- rnorm(30)

plot(x = var_1, y = var_2)
```



Making plots with base R

Often, you want to plot the data straight from a data frame

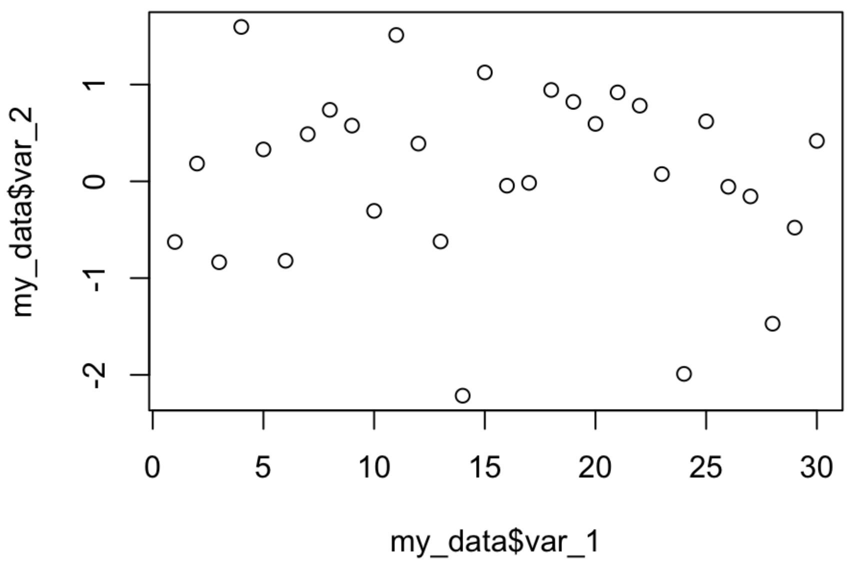
```
# creating a data frame from the vectors we had
my_data = data.frame(var_1, var_2)
my_data
```

```
   var_1       var_2
1      1 -0.62645381
2      2  0.18364332
3      3 -0.83562861
4      4  1.59528080
5      5  0.32950777
6      6 -0.82046838
7      7  0.48742905
8      8  0.73832471
9      9  0.57578135
10    10 -0.30538839
11    11  1.51178117
12    12  0.38984324
13    13 -0.62124058
14    14 -2.21469989
15    15  1.12493092
16    16 -0.04493361
17    17 -0.01619026
18    18  0.94383621
19    19  0.82122120
20    20  0.59390132
21    21  0.91897737
22    22  0.78213630
```

Making plots with base R

Often, you want to plot the data straight from a data frame

```
# use $ to access column from a data set
plot(x = my_data$var_1, y = my_data$var_2)
```

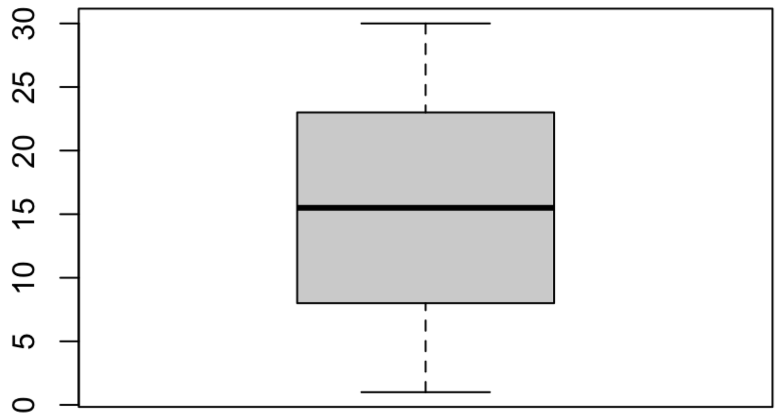


Making plots with base R

`plot()` is one of the functions from base R to produce a graphic.

We also have `hist()`, `boxplot()` and many others. They may vary in usage, e.g. in number of arguments we typically need to supply

```r
# use $ to access column from a data set
boxplot(x = my_data$var_1)
```



ggplot2

- A package for producing graphics - gg = Grammar of graphics
- Created by Hadley Wickham in 2005
- Belongs to “Tidyverse” family of packages
- *“Make a ggplot”* = Make a plot with the use of ggplot2 package
- based on the idea of **layering** plot objects on top of one another

ggplot2

- Pros: extremely powerful/flexible – allows combining multiple plot elements together, allows high customization of a look, many resources online
- Cons: ggplot2-specific “grammar of graphic” of constructing a plot
- ggplot2 gallery

Tidy data

To make graphics using `ggplot2`, our data needs to be in a **tidy** format

Tidy data:

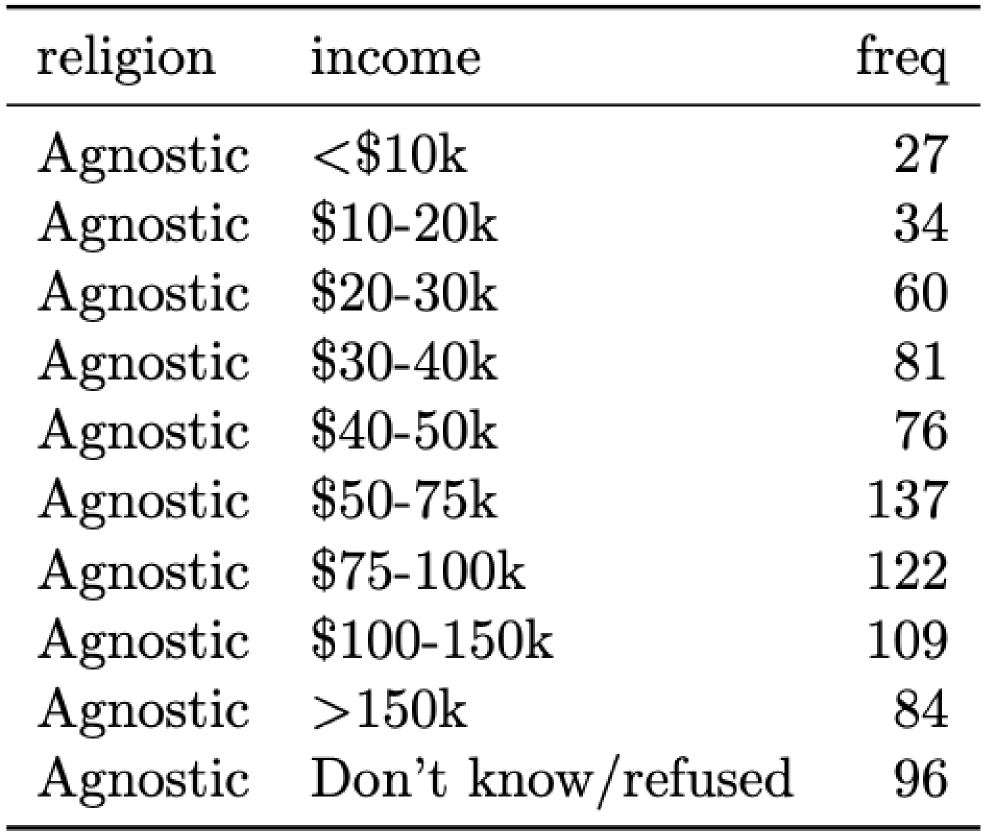
1. Each variable forms a column.
2. Each observation forms a row.

Messy data:

- Column headers are values, not variable names.
- Multiple variables are stored in one column.
- Variables are stored in both rows and columns.

Tidy data: example

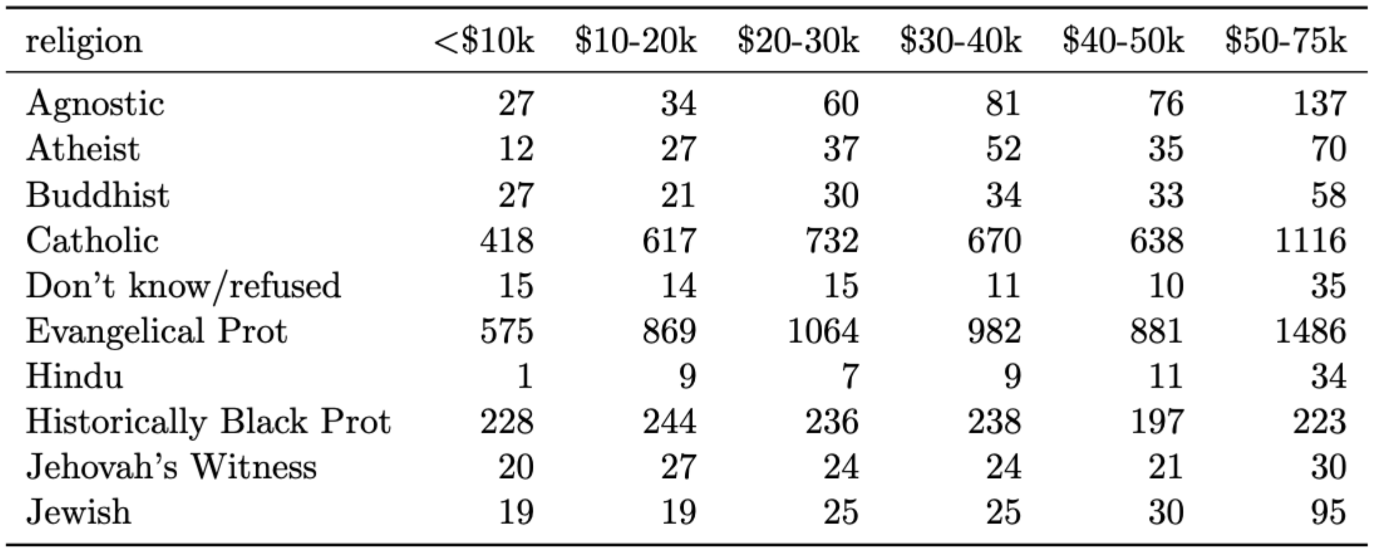
Each variable forms a column. Each observation forms a row.



religion	income	freq
Agnostic	<$10k	27
Agnostic	$10-20k	34
Agnostic	$20-30k	60
Agnostic	$30-40k	81
Agnostic	$40-50k	76
Agnostic	$50-75k	137
Agnostic	$75-100k	122
Agnostic	$100-150k	109
Agnostic	>150k	84
Agnostic	Don't know/refused	96

Messy data: example

Column headers are values, not variable names



religion	<$10k	$10-20k	$20-30k	$30-40k	$40-50k	$50-75k
Agnostic	27	34	60	81	76	137
Atheist	12	27	37	52	35	70
Buddhist	27	21	30	34	33	58
Catholic	418	617	732	670	638	1116
Don't know/refused	15	14	15	11	10	35
Evangelical Prot	575	869	1064	982	881	1486
Hindu	1	9	7	9	11	34
Historically Black Prot	228	244	236	238	197	223
Jehovah's Witness	20	27	24	24	21	30
Jewish	19	19	25	25	30	95

Table 4: The first ten rows of data on income and religion from the Pew Forum. Three columns, `$75-100k`, `$100-150k` and `>150k`, have been omitted

Read more about tidy data and see other examples: Tidy Data tutorial by Hadley Wickham

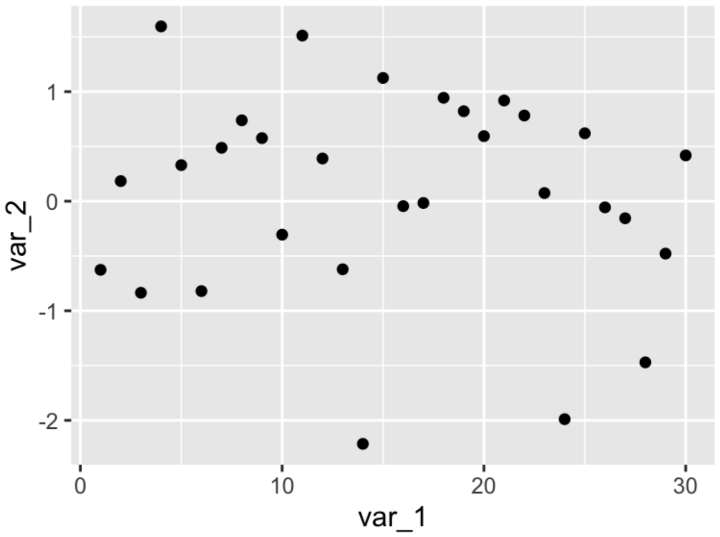
It's also helpful to have data in long format!!!

First plot with `ggplot2` package

```
head(my_data, 3)    # to recall how "my_data" data frame looks like
```

```
  var_1      var_2
1     1 -0.6264538
2     2  0.1836433
3     3 -0.8356286
```

```
library(ggplot2)

ggplot(my_data, aes(x = var_1, y = var_2)) +
  geom_point()
```



First plot with `ggplot2` package: unpacking

```
ggplot(my_data, mapping = aes(x = var_1, y = var_2)) +
  geom_point()
```

In the code first line, we use `ggplot()` function. It takes two arguments:

- **Data frame** (here: `my_data`)
- **Aesthetic mapping** (here: `mapping = aes(x = var_1, y = var_2)`) that describes how variables in our data are mapped to elements of the plot

  Here, can read: *take variable named `var_1` from a data frame `my_data` and use it for `x` axis; take variable named `var_2` from a data frame `my_data` and use it for `y` axis*

First plot with `ggplot2` package: unpacking

```
ggplot(my_data, mapping = aes(x = var_1, y = var_2)) +
  geom_point()
```

In the code second line, we have definition of a **layer** of a plot. There is only one layer here, `+ geom_point()`

- Here, can read: *add points to the plot (use data as provided by the aesthetic mapping)*

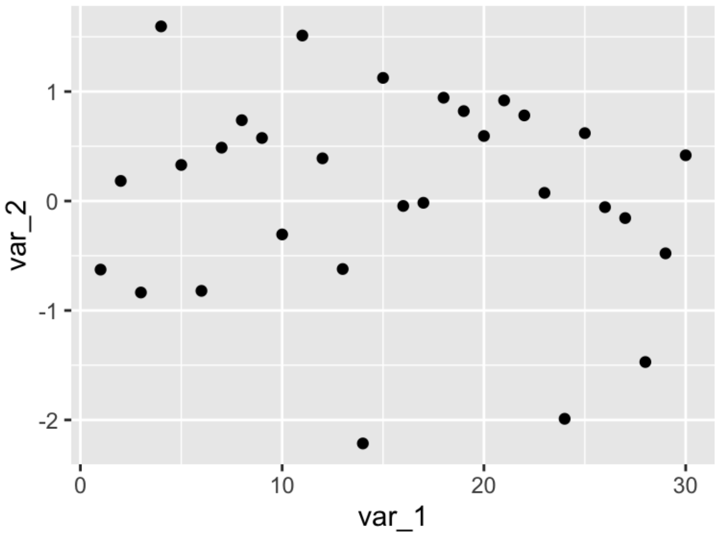
Specifying plot layers

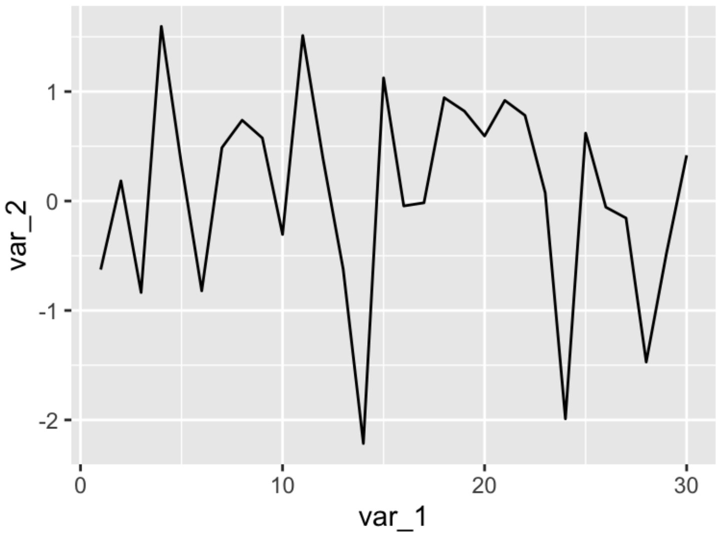
There are many to choose from, to list just a few:

- `geom_point()` – points (we have seen)
- `geom_line()` – lines to connect observations
- `geom_boxplot()`
- `geom_histogram()`
- `geom_bar()`
- `geom_col()`
- `geom_errorbar()`
- `geom_density()`
- `geom_tile()` – blocks filled with color

Specifying plot layers: examples

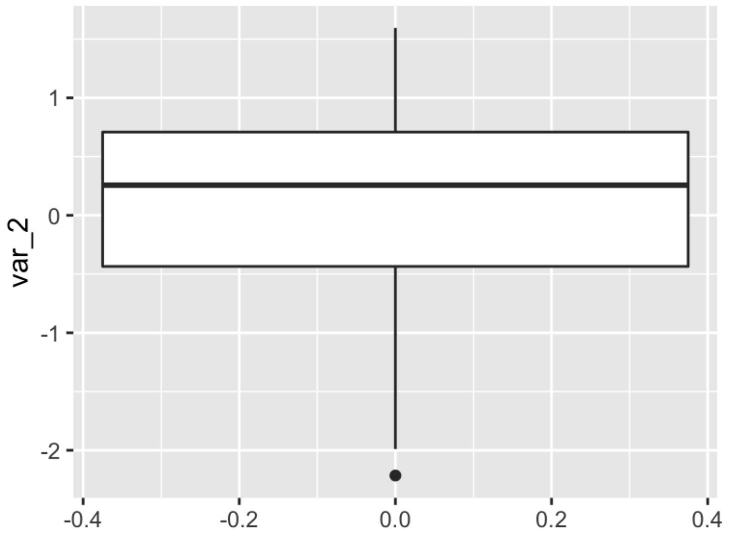
```r
plt1 <-
  ggplot(my_data, aes(x = var_1, y = var_2)) +
  geom_point()

plt2 <-
  ggplot(my_data, aes(x = var_1, y = var_2)) +
  geom_line()

plt1; plt2 # to have 2 plots printed next to each other on a slide
```

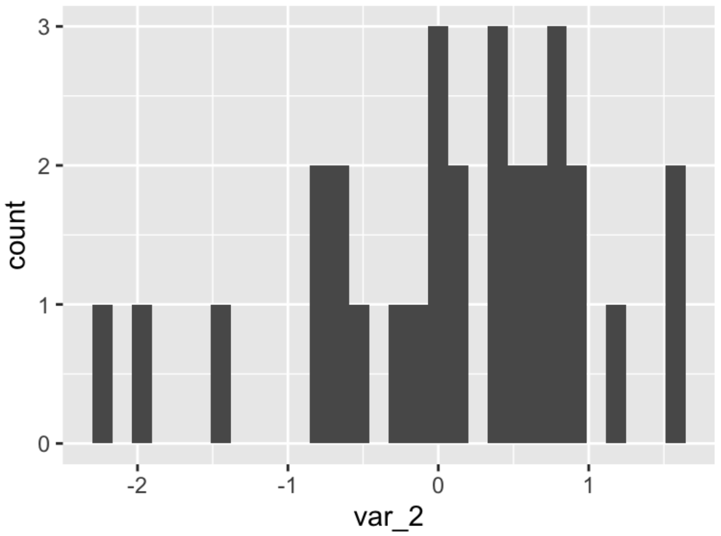




Specifying plot layers: examples

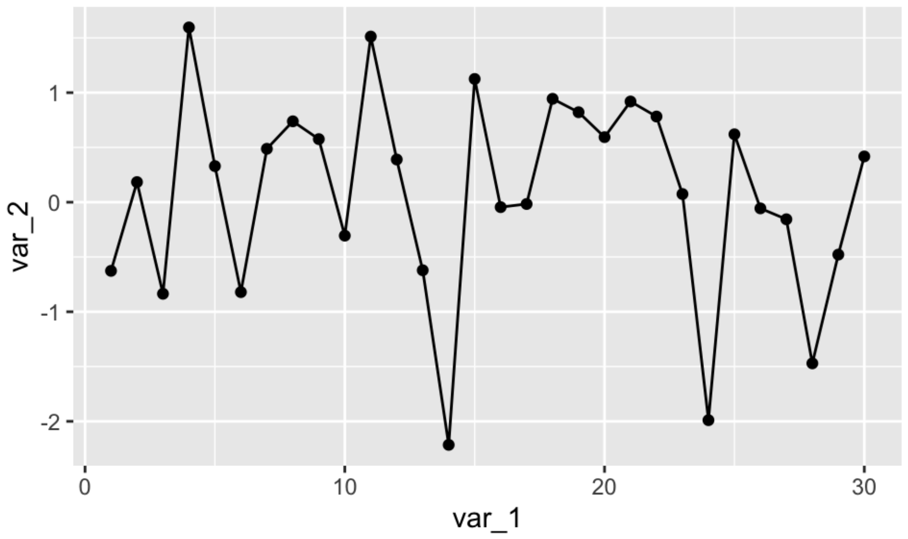
```r
plt3 <-
  ggplot(my_data, aes(y = var_2)) +
  geom_boxplot()

plt4 <-
  ggplot(my_data, aes(x = var_2)) +
  geom_histogram()

plt3; plt4 # to have 2 plots printed next to each other on a slide
```





Specifying plot layers: combining multiple layers

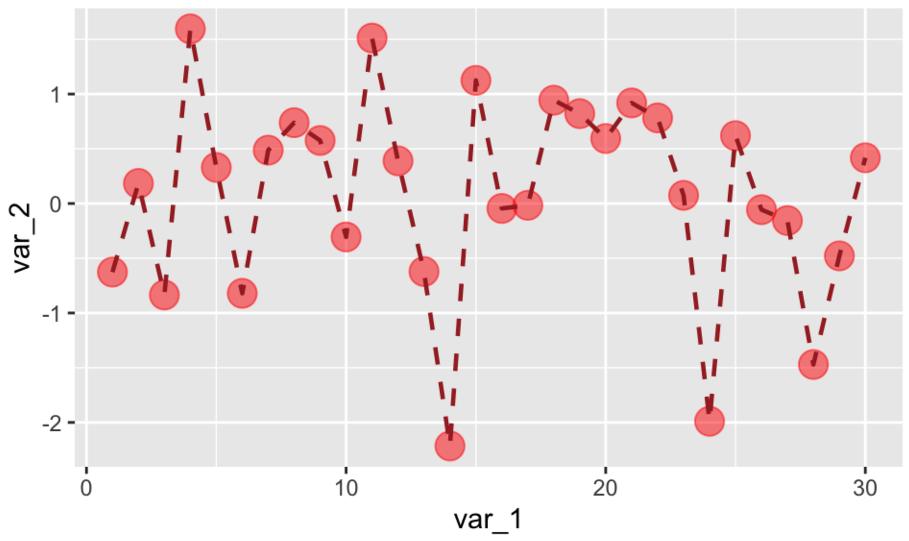
```
ggplot(my_data, aes(x = var_1, y = var_2)) +
  geom_point() +
  geom_line()
```



Customize the look of the plot

You can change look of each layer separately.

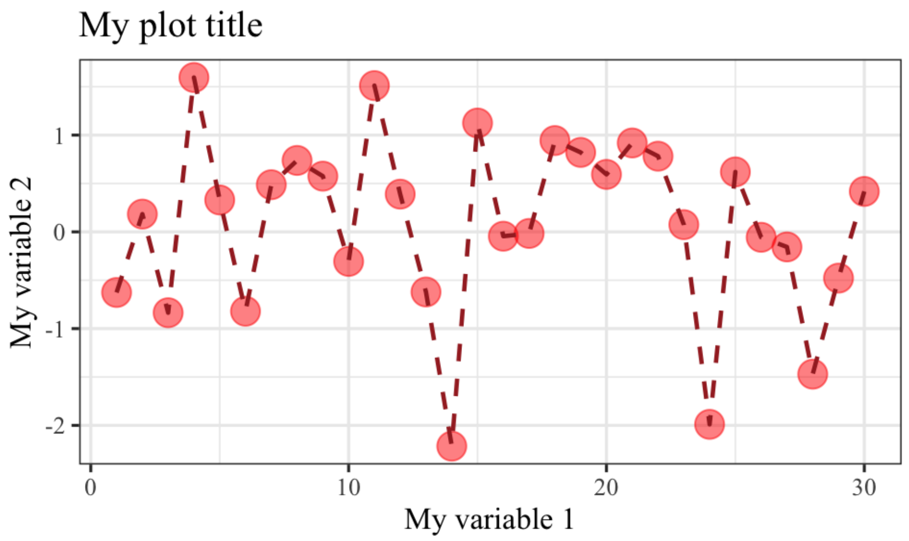
```
ggplot(my_data, aes(x = var_1, y = var_2)) +
  geom_point(size = 5, color = "red", alpha = 0.5) +
  geom_line(size = 0.8, color = "brown", linetype = 2)
```



Customize the look of the plot

You can change look of each layer separately. You can change the look of whole plot-specific elements, too.

```r
ggplot(my_data, aes(x = var_1, y = var_2)) +
  geom_point(size = 5, color = "red", alpha = 0.5) +
  geom_line(size = 0.8, color = "brown", linetype = 2) +
  labs(x = "My variable 1", y = "My variable 2",
       title = "My plot title") +
  theme_bw(base_size = 12, base_family = "Times New Roman")
```



Lab 1

Lab document: http://jhudatascience.org//intro_to_r/Data_Visualization/lab/Data_Visualization_Lab.Rm

Group and/or color by variable's values

We generate some data frame for the purpose of demonstration

- 20 different items (e.g. products in a store)
- of 2 different categories (e.g. pasta, rice)
- 100 price values collected over time for each item

```
# create 4 vectors: 2x character class and 2x numeric class
item_categ <- as.vector(sapply(1:20, function(i) rep(sample(c("pasta", "rice")
item_ID  <- rep(seq(from = 1, to = 20), each = 100)
item_ID <- paste0("ID_", item_ID)
observation_time  <- rep(seq(from = 1, to = 100), times = 20)
item_price <- as.vector(replicate(20, cumsum(rnorm(100))))
item_price <- item_price + abs(min(item_price)) + 1

# use 4 vectors to create data frame with 4 columns
df  <- data.frame(item_ID, item_categ, observation_time, item_price)
```

Group and/or color by variable's values

```
head(df, 3)
```

```
  item_ID item_categ observation_time item_price
1    ID_1      pasta                1   22.53215
2    ID_1      pasta                2   22.27878
3    ID_1      pasta                3   22.97575
```

```
tail(df, 3)
```

```
     item_ID item_categ observation_time item_price
1998   ID_20       rice               98   16.71701
1999   ID_20       rice               99   16.25430
2000   ID_20       rice              100   17.29740
```

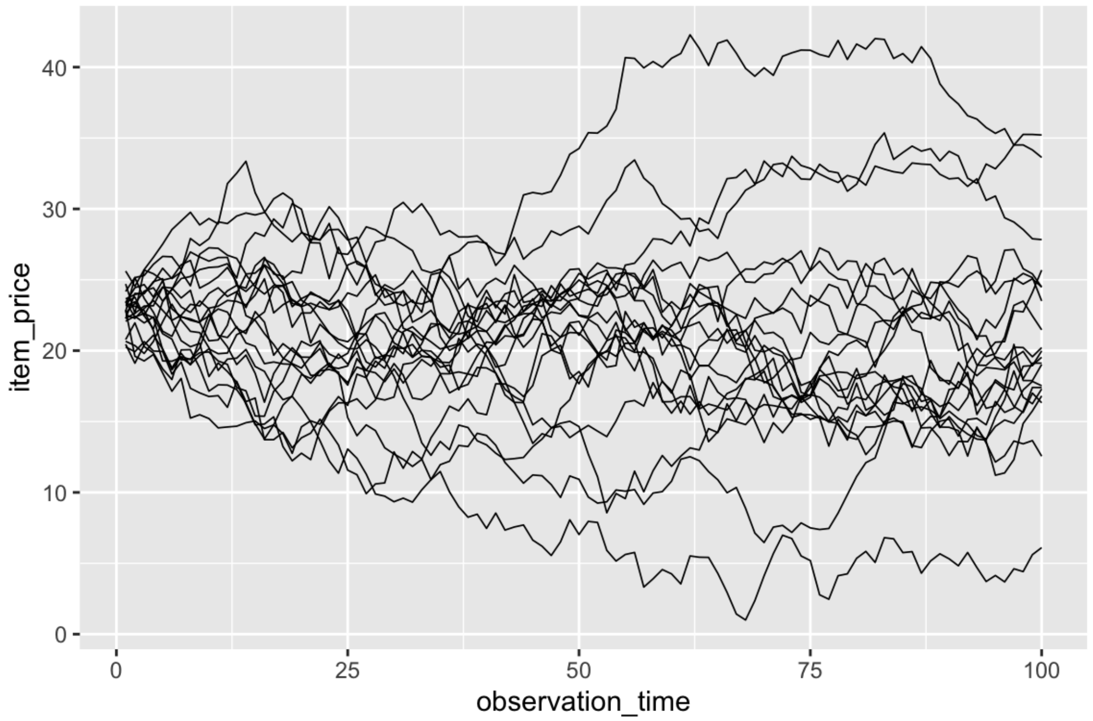
```
str(df)
```

```
'data.frame':   2000 obs. of  4 variables:
 $ item_ID         : chr  "ID_1" "ID_1" "ID_1" "ID_1" ...
 $ item_categ      : chr  "pasta" "pasta" "pasta" "pasta" ...
 $ observation_time: int  1 2 3 4 5 6 7 8 9 10 ...
 $ item_price      : num  22.5 22.3 23 23.5 22.8 ...
```

Group and/or color by variable’s values

You can use `group` element in a mapping to indicate that each `item_ID` will have a separate price line (more generally: a separate layer element)

```
ggplot(df, aes(x = observation_time, y = item_price, group = item_ID)) +
  geom_line(size = 0.3)
```

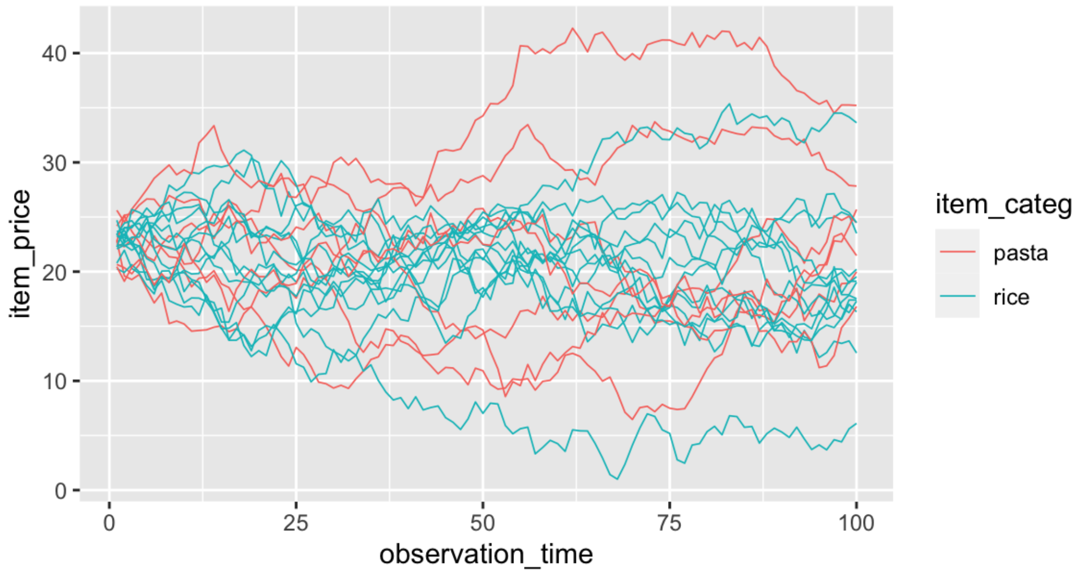


Group and/or color by variable's values

You can use `color` element in a mapping to indicate that each `item_categ` will have different color used.

Colors palette is selected by default (and can be modified). Legend position, legend title etc. have a default look (and can be modified).

```
ggplot(df, aes(x = observation_time, y = item_price, group = item_ID,
               color = item_categ)) +
  geom_line(size = 0.3)
```

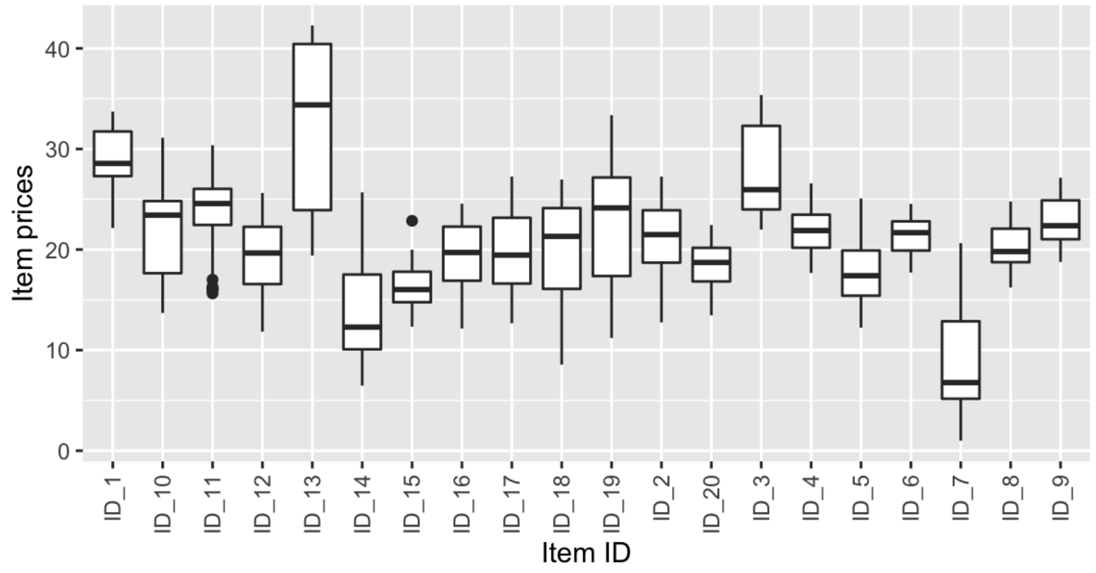


Group and/or color by variable's values

Here, we use boxplot instead of lines.

Note how aesthetic mappings are defined now: `aes(x = item_ID, y = item_price)`.

```
ggplot(df, aes(x = item_ID, y = item_price)) +
  geom_boxplot() +
  labs(x = "Item ID", y = "Item prices") +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust = 1))
```

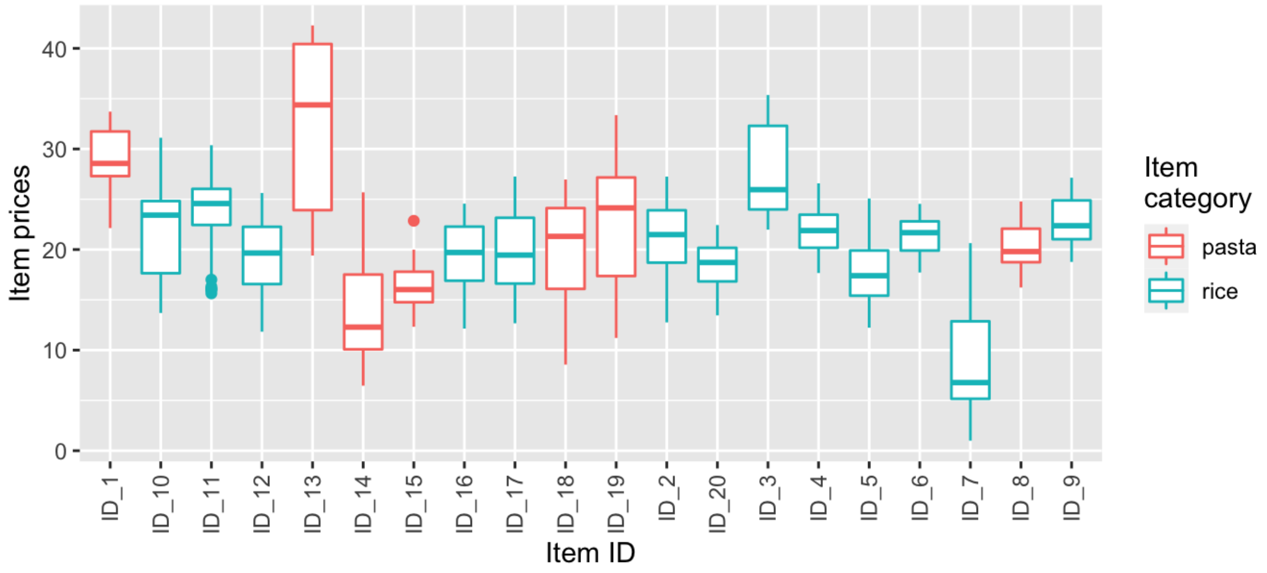


Group and/or color by variable's values

We use `color` element in mappings to indicate that each `item_categ` will have different **color of boxplot box** used.

We also use `color = "Item\ncategory"` to change name of legend.

```r
ggplot(df, aes(x = item_ID, y = item_price, color = item_categ)) +
  geom_boxplot() +
  labs(x = "Item ID", y = "Item prices", color = "Item\ncategory") +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust = 1))
```

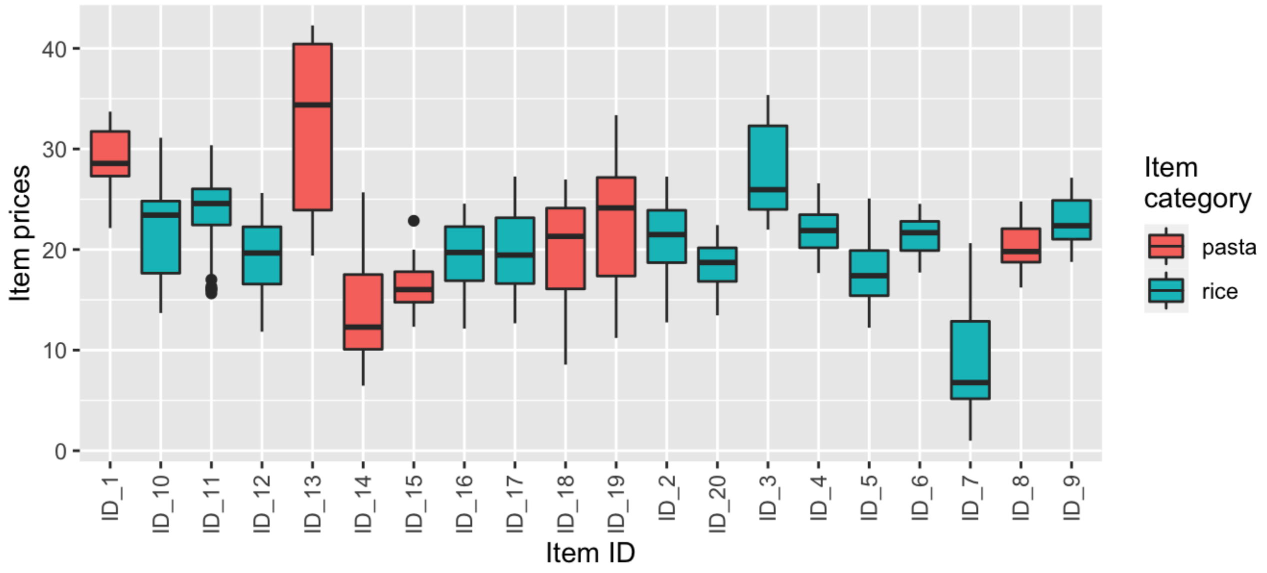


Group and/or color by variable's values

We use `fill` element in mappings to indicate that each `item_categ` will have different **color of boxplot filling** used.

We also use `fill = "Item\ncategory"` to change name of legend.

```r
ggplot(df, aes(x = item_ID, y = item_price, fill = item_categ)) +
  geom_boxplot() +
  labs(x = "Item ID", y = "Item prices", fill = "Item\ncategory") +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust = 1))
```



Group and/or color by variable's values

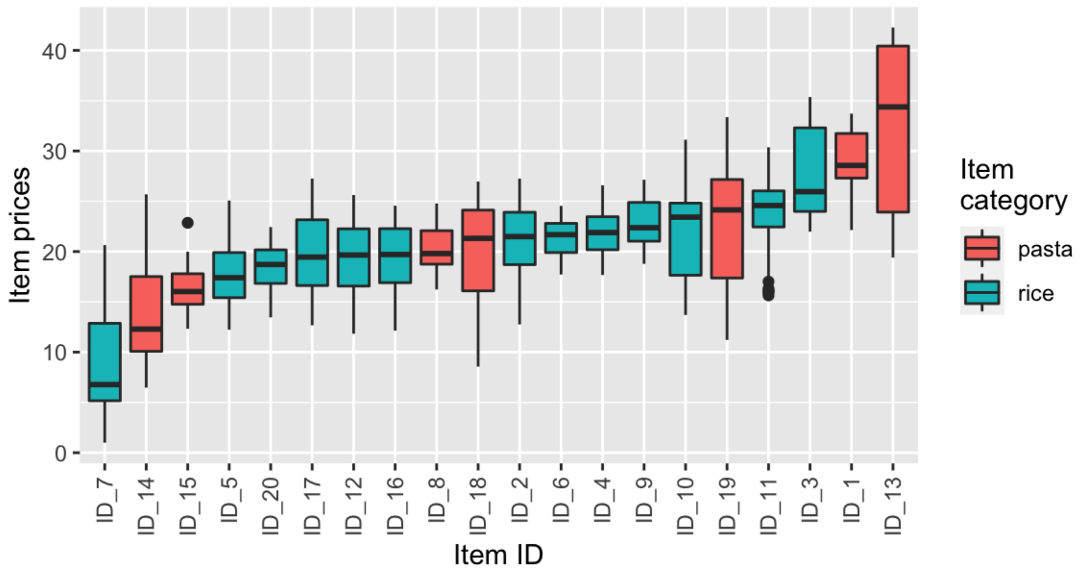
Let's make some tweaks to item ID (`item_ID`) ordering by creating a factor version with a certain order of factor levels

```
item_ID_levels <-
  df %>%
  group_by(item_ID) %>%
  summarise(item_price_median = median(item_price)) %>%
  arrange(item_price_median) %>%
  pull(item_ID)

df <-
  df %>%
  mutate(item_ID_factor = factor(item_ID, levels = item_ID_levels))
```

Group and/or color by variable's values

Same as 2 slides ago, but we replaced `item_ID` with `item_ID_factor` in mappings definition (`aes()`).

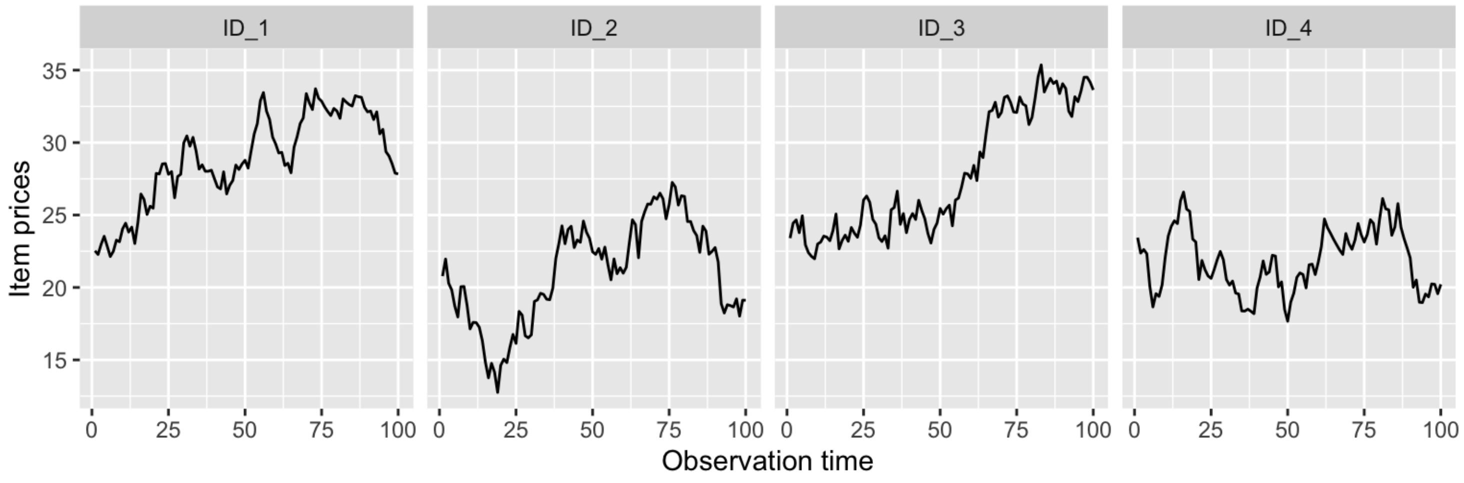
```r
ggplot(df, aes(x = item_ID_factor, y = item_price, fill = item_categ)) +
  geom_boxplot() +
  labs(x = "Item ID", y = "Item prices", fill = "Item\ncategory") +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust = 1))
```



Split plot into panels by variable's values

We define data subset to keep only 4 (out of 20) item IDs. We use `facet_grid(. ~ item_ID)` to split the plot into panels where each product ID has a separate panel

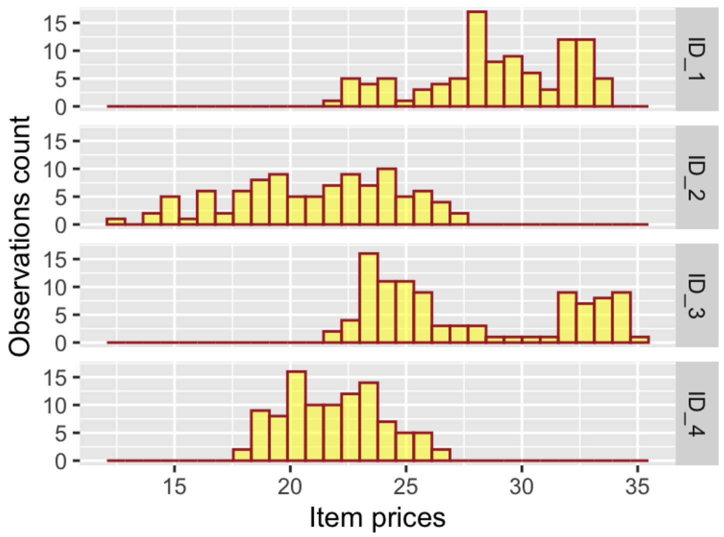
```
df_subset <- df %>%
  filter(item_ID %in% c("ID_1", "ID_2", "ID_3", "ID_4"))

ggplot(df_subset, aes(x = observation_time, y = item_price)) +
  geom_line() +
  labs(x = "Observation time", y = "Item prices") +
  facet_grid(. ~ item_ID)
```



Split plot into panels by variable's values

We define data subset to keep only 4 (out of 20) item IDs. We use `facet_grid(item_ID ~ .)` to split the plot into panels where each product ID has a separate panel

```
ggplot(df_subset, aes(x = item_price)) +
  geom_histogram(fill = "yellow", color = "brown", alpha = 0.5) +
  labs(x = "Item prices", y = "Observations count") +
  facet_grid(item_ID ~ .)
```



Saving a ggplot to file

A few options:

- RStudio > Plots > Export > Save as image / Save as PDF
- RStudio > Plots > Zoom > [right mouse click on the plot] > Save image as
- In the code

```
plot_FINAL <-
  ggplot(df_subset, aes(x = item_price)) +
  geom_histogram(fill = "yellow", color = "brown", alpha = 0.5) +
  labs(x = "Item prices", y = "Observations count") +
  facet_grid(item_ID ~ .)

ggsave(filename = "very_important_plot.png",  # will save in working director
       plot = plot_FINAL,
       width = 6, height = 3.5)               # by default in inch
```

Lab 2

Lab document: http://jhudatascience.org//intro_to_r/Data_Visualization/lab/Data_Visualization_Lab.Rm