Exploratory Data Analysis with R

Interactive Data Visualization

Xuemao Zhang
East Stroudsburg University

November 4, 2022

Outline

- Overview
- `ggplot2` and `plotly`
- Data visualization with `plotly`
  - *Pie/donut charts*
  - *Bars and histograms*
  - *Density and histogram overlay*
  - *Boxplots*
  - *Violin plots*
  - *Error bars*
  - *Scatter plot*
  - *3d scatter plots*
- Buttons
- `You can disable/enable mouse click advance by pressing key 'k', when viewing the presentation.`

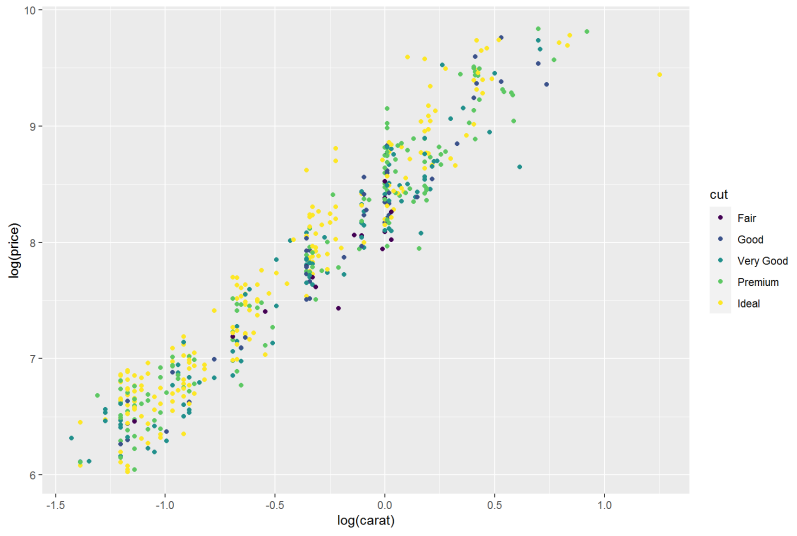
Overview

- Interactive graphics are different from animations in that the viewer has control.
- `plotly`: has come a long way in the last years, started as part of Carson Sievert's PhD thesis research. The beauty is that it builds directly onto `ggplot2`
- There are two main ways to creating a plotly object:
  - *Transforming a ggplot2 object (via `ggplotly()`) into a plotly object*
  - *Directly initializing a plotly object with `plot_ly()/plot_geo()/plot_mapbox()`. They **do not** follow the grammar of graphics.*

plotly and ggplot2

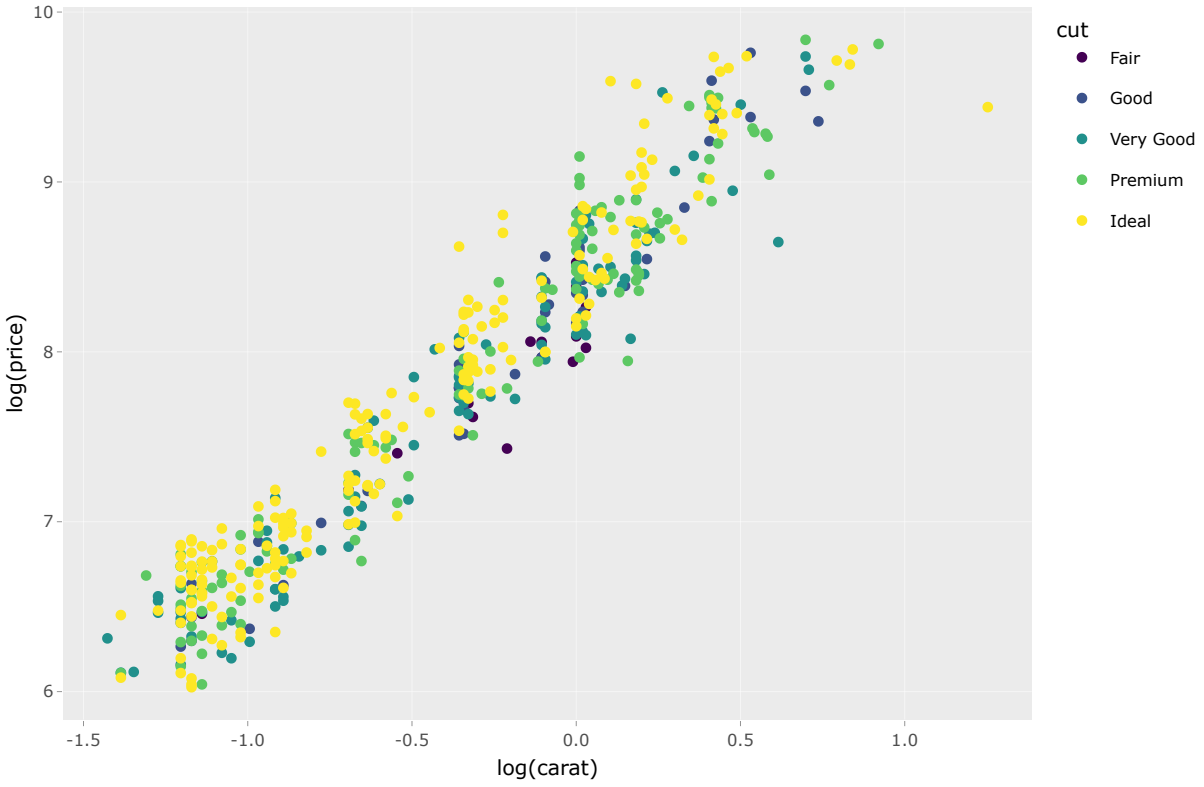
- Calling plotly::ggplotly() after bringing a ggplot up will coerce it into an interactive plot. It may fail sometimes.

```
library(plotly);
library(dplyr);
library(ggplot2);
data(diamonds, package = "ggplot2");
set.seed(37);
diamonds1 = sample_n(diamonds,size=500);
p <- ggplot(diamonds1, aes(x = log(carat), y = log(price),color=cut))+
  geom_point();
p
```



plotly and ggplot2

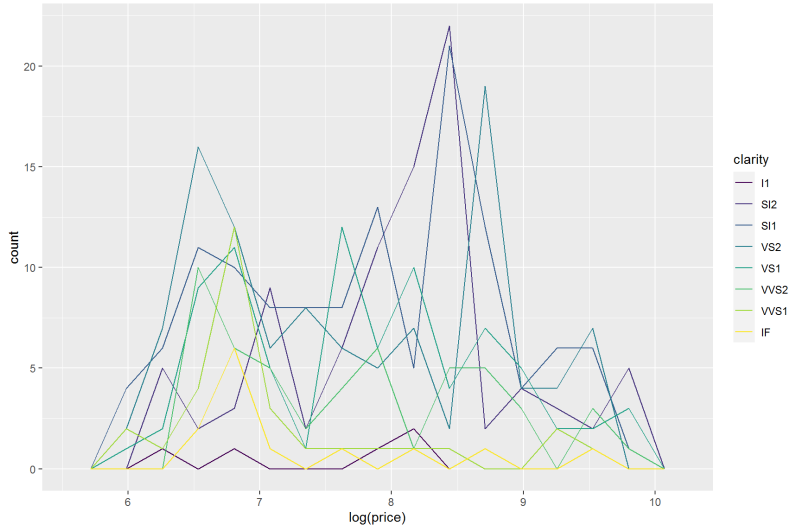
```
ggplotly(p);
```



plotly and ggplot2

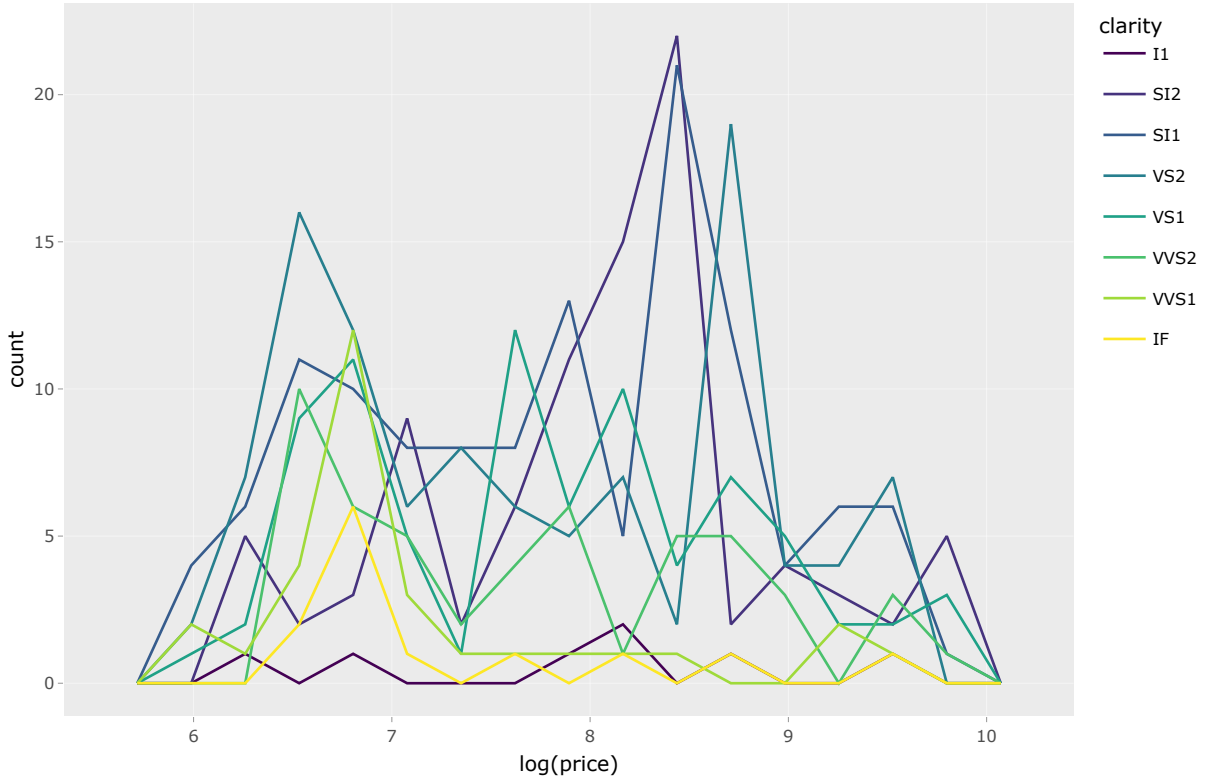
- ggplot2::geom_freqpoly() is used to visualise the distribution of a single continuous variable by dividing the x axis into bins and counting the number of observations in each bin
- geom_freqpoly() produces a frequency polygon for each level of that variable

```
p <- ggplot(diamonds1, aes(x = log(price), color = clarity)) +
    geom_freqpoly(bins=15);
p
```



plotly and ggplot2

```
ggplotly(p);
```

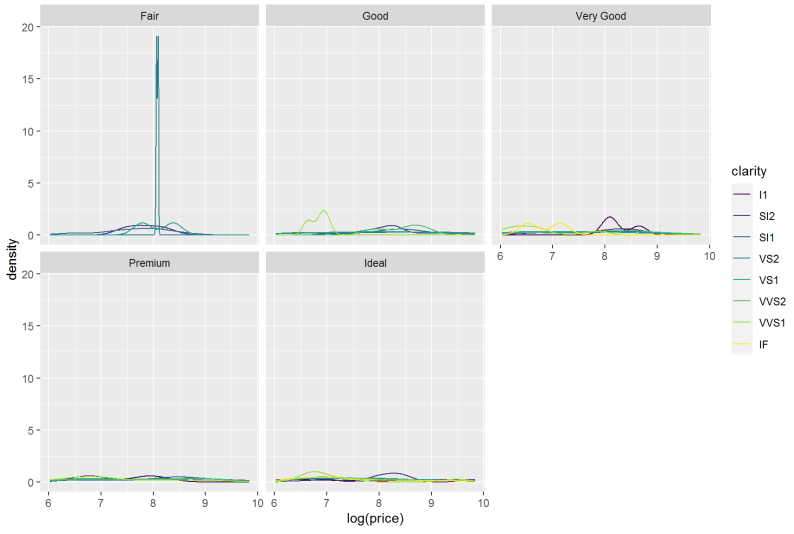


plotly and ggplot2

- stat = “density” is used to display relative rather than absolute frequencies.

```
p <- ggplot(diamonds1, aes(x = log(price), color = clarity)) +
    geom_freqpoly(stat = "density") +
    facet_wrap(~cut);
p
```

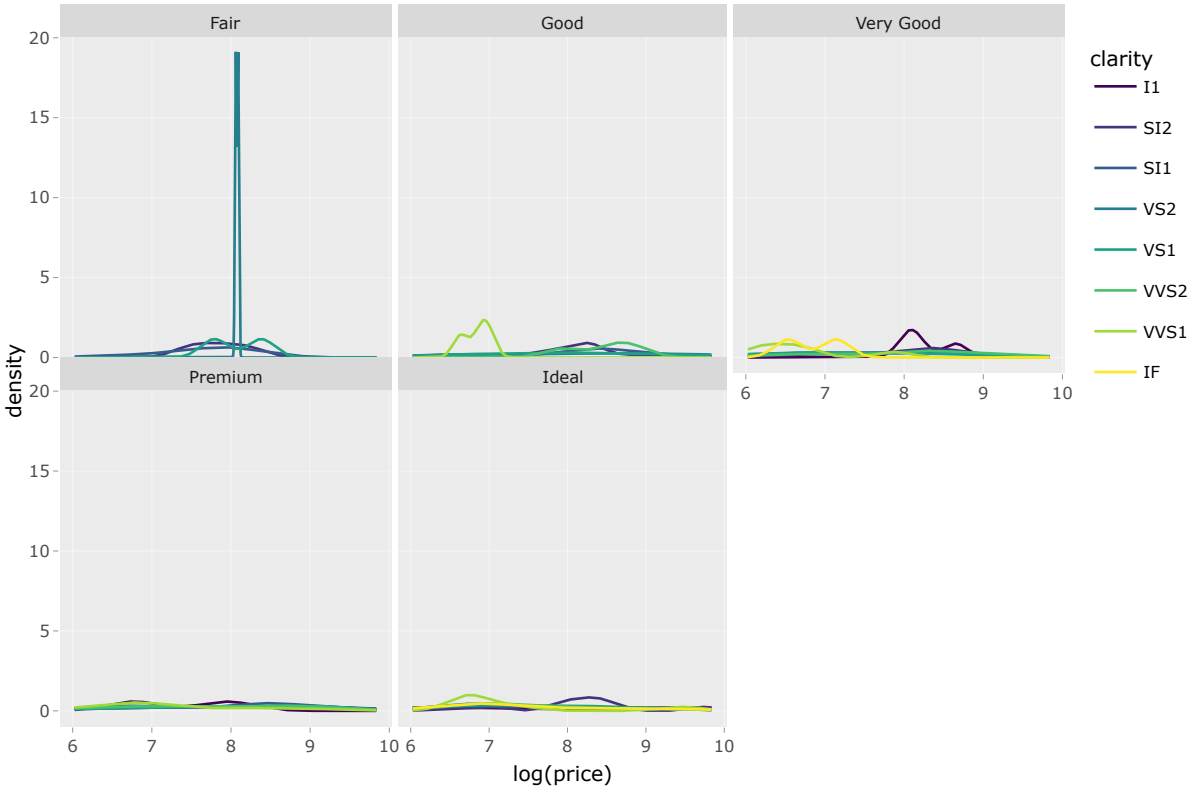
```
## Warning: Groups with fewer than two data points have been dropped.
```



plotly and ggplot2

```
ggplotly(p);
```

```
## Warning: Groups with fewer than two data points have been dropped.
```



plotly and ggplot2

- Scatterplot(ly) matrix

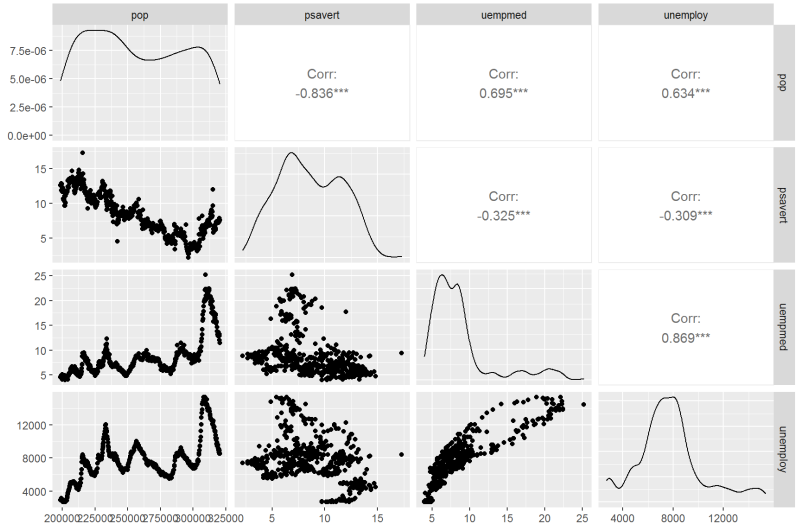
```
library(GGally);
data(economics, package = "ggplot2");
dim(economics);
```

```
## [1] 574   6
```

```
str(economics);
```

```
## spec_tbl_df [574 x 6] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ date    : Date[1:574], format: "1967-07-01" "1967-08-01" ...
##  $ pce     : num [1:574] 507 510 516 512 517 ...
##  $ pop     : num [1:574] 198712 198911 199113 199311 199498 ...
##  $ psavert : num [1:574] 12.6 12.6 11.9 12.9 12.8 11.8 11.7 12.3 11.7 12.3 ...
##  $ uempmed : num [1:574] 4.5 4.7 4.6 4.9 4.7 4.8 5.1 4.5 4.1 4.6 ...
##  $ unemploy: num [1:574] 2944 2945 2958 3143 3066 ...
```

```
p <- ggpairs(economics[,3:6]);
p
```

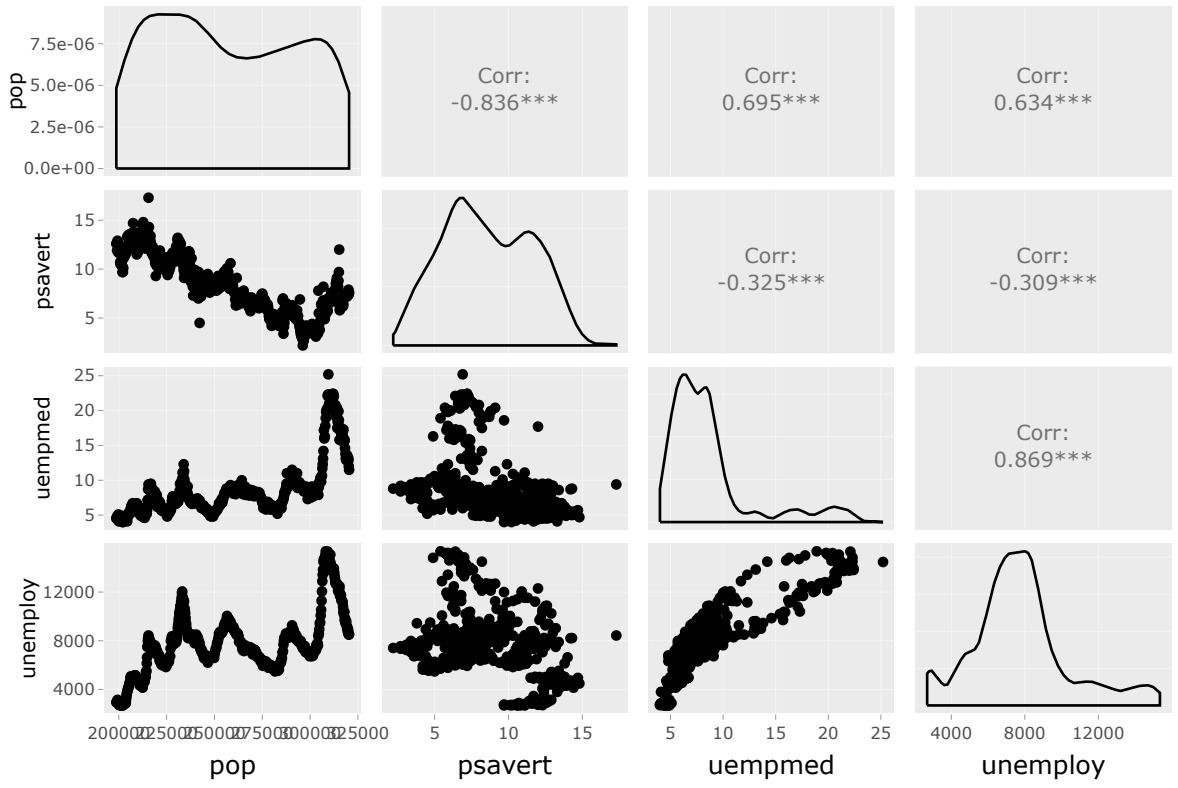


plotly and ggplot2

- Scatterplot(ly) matrix

```
ggplotly(p)
```

```
## Warning: Can only have one: highlight

## Warning: Can only have one: highlight

## Warning: Can only have one: highlight
```



Data visualization with `plotly`

- Plotly is both a commercial service and open source product for creating high end interactive visualizations. The plotly package allows you to create plotly interactive graphs from within R.

- Any graph made with the plotly R package is powered by the JavaScript library plotly.js((MIT licensed)). The `plot_ly()` function provides a 'direct' interface to plotly.js with some additional abstractions to help reduce typing.

- The `plot_ly()` function has numerous arguments that are unique to the R package (e.g., color, stroke, span, symbol, linetype, etc) and make it easier to encode data variables.

  - *This function maps R objects to plotly.js. It provides abstractions for doing common things (e.g. mapping data values to fill colors (via color) or creating animations (via frame)) and sets some different defaults to make the interface feel more 'R-like' (i.e., closer to `plot()` and `ggplot2::qplot()`).*

Data visualization with `plotly`

- A plotly.js figure contains one (or more) trace(s), and every trace has a type. The trace type scatter is great for drawing low-level geometries (e.g., points, lines, text, and polygons) and provides the foundation for many add_*() functions (e.g., add_markers(), add_lines(), add_paths(), add_segments(), add_ribbons(), add_area(), and add_polygons()) as well as many ggplotly() charts.
- The pipe operator `%>%` is used in the package.
- The types of traces are similar as `geom_*` in ggplot2. See R Figure Reference: image Traces.
- We follow the book Interactive web-based data visualization with R, plotly, and shiny to show some interactive data visualizations.

Data visualization with `plotly`

- Plotly's graph description places attributes into two categories: traces (which describe a single series of data in a graph) and layout attributes that apply to the rest of the chart, like the title, xaxis, or annotations).

```
library(plotly)

p <- plot_ly(economics,
         type = "scatter",       # all "scatter" attributes:
         https://plotly.com/r/reference/#scatter
         mode = "markers",       # the drawing mode for this scatter
         trace
         x = ~date,              # more about scatter's "x":
         /r/reference/#scatter-x
         y = ~uempmed,           # more about scatter's "y":
         /r/reference/#scatter-y
         name = "unemployment",  # more about scatter's "name":
         /r/reference/#scatter-name
         marker = list(          # marker is a named list, valid keys:
         /r/reference/#scatter-marker
         color="#264E86"        # more about marker's "color" attribute:
         /r/reference/#scatter-marker-color
                                 # see https://htmlcolorcodes.com/ for
         colors
               )) %>%

  add_trace(x = ~date,                                          #
         scatter's "x": /r/reference/#scatter-x
            y = ~fitted((loess(uempmed ~ as.numeric(date)))),  #
         scatter's "y": /r/reference/#scatter-y
            mode = 'lines+markers',                             #
         scatter's "y": /r/reference/#scatter-mode
            line = list(                                        # line is
         a named list, valid keys: /r/reference/#scatter-line
            color = "#5E88FC",                                  # line's
         "color": /r/reference/#scatter-line-color
            dash = "dashed"                                    # line's
         "dash" property: /r/reference/#scatter-line-dash
               )
     ) %>%
```

```
layout(                         # all of layout's properties:
       /r/reference/#layout
       title = "Unemployment", # layout's title: /r/reference/#layout-
       title
       xaxis = list(            # layout's xaxis is a named list. List
       of valid keys: /r/reference/#layout-xaxis
          title = "Time",       # xaxis's title: /r/reference/#layout-
       xaxis-title
          showgrid = F),        # xaxis's showgrid:
       /r/reference/#layout-xaxis-showgrid
       yaxis = list(            # layout's yaxis is a named list. List
       of valid keys: /r/reference/#layout-yaxis
           title = "uidx")      # yaxis's title: /r/reference/#layout-
       yaxis-title
     )
```

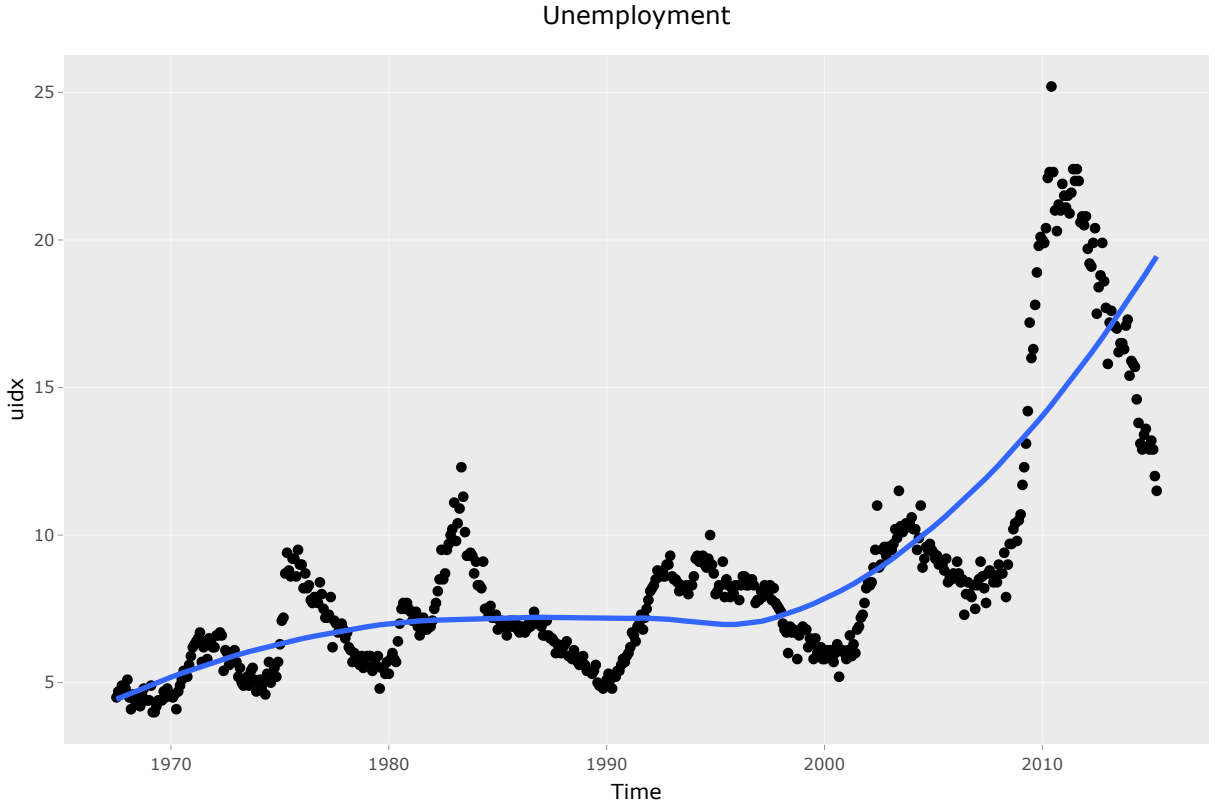
Data visualization with `plotly`

```
p
```

Data visualization with `plotly`

- using ggplot2

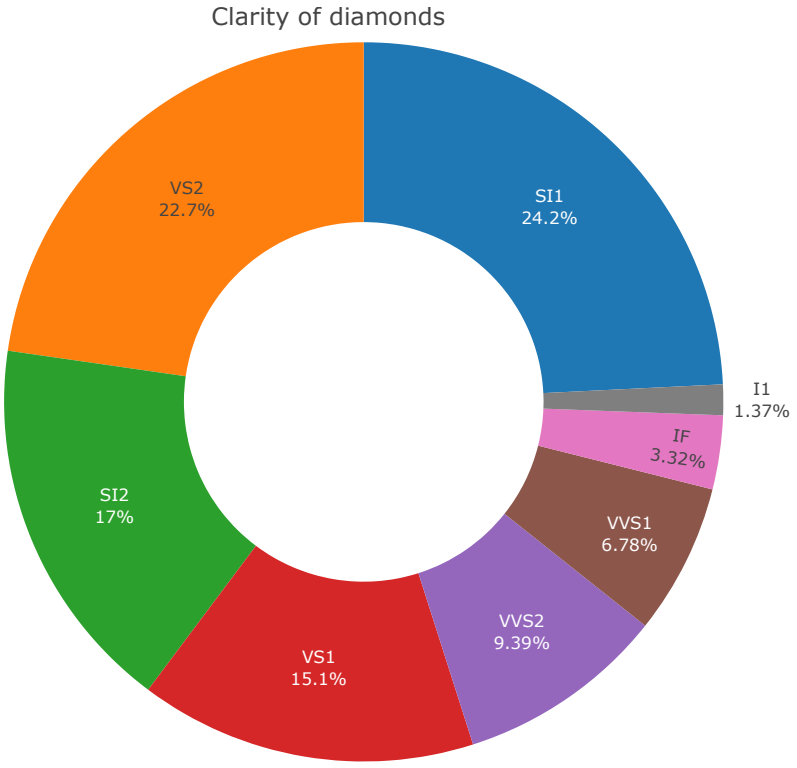
```
p=ggplot(data=economics, aes(x=date, y=uempmed) )+
  geom_point()+
  geom_smooth(formula=y~x, method="loess", se=F)+
  ggtitle("Unemployment")+
  theme(plot.title=element_text(hjust = 0.5))+#center the title
  labs(x="Time", y="uidx");
ggplotly(p)
```



Data visualization with `plotly`

Pie/donut charts

```
diamonds%>%group_by(clarity)%>%summarize(freq = n())%>% #
  plot_ly(labels = ~clarity, values = ~freq)%>%
  add_pie(hole = 0.5, text=~clarity)%>%
  layout(title = "Clarity of diamonds", showlegend =F);
```



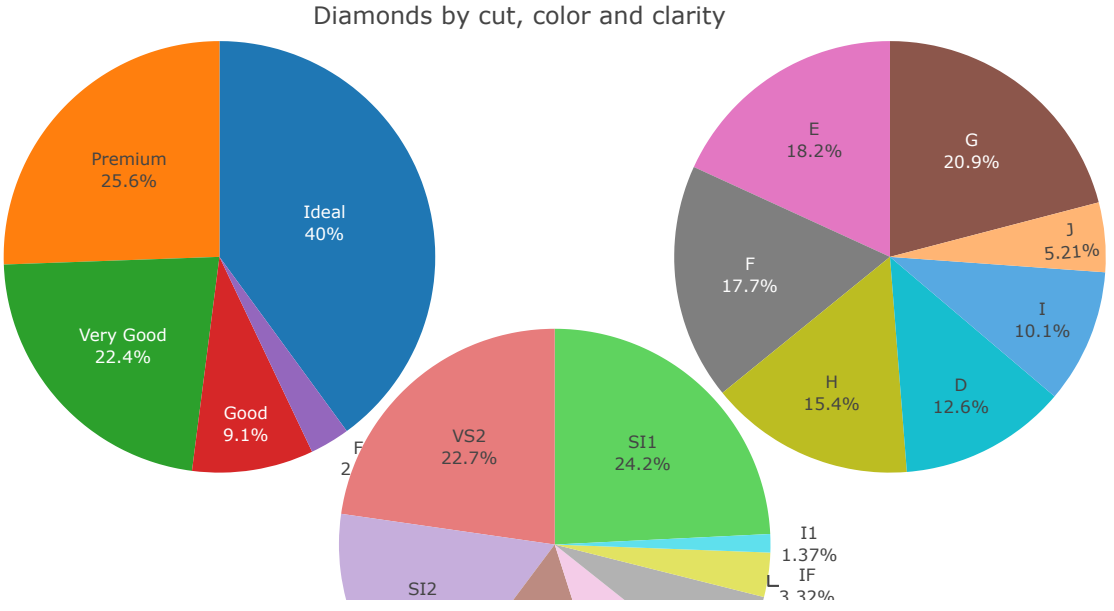
- Polar coordinates are not yet supported by ggplotly(). So we cannot make a ggplot2 pie chart interactive.

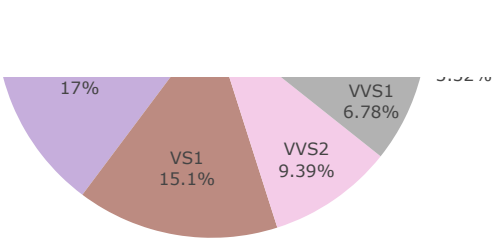
Data visualization with plotly

Pie/donut charts

- Pie charts with subplots

```
diamonds%>%group_by(cut)%>%summarize(freq = n())->data1;
diamonds%>%group_by(color)%>%summarize(freq = n())->data2;
diamonds%>%group_by(clarity)%>%summarize(freq = n())->data3;

fig <- plot_ly();
fig<-fig %>%add_pie(data=data1, labels = ~cut, values = ~freq,
         text=~cut, domain = list(x = c(0, 0.4), y = c(0.4, 1)));

fig<-fig %>%add_pie(data=data2, labels = ~color, values =
         ~freq,text=~color, domain = list(x = c(0.6, 1), y = c(0.4,
         1)));

fig<-fig %>%add_pie(data=data3, labels = ~clarity, values = ~freq,
         text=~clarity, domain = list(x = c(0.25, 0.75), y = c(0,
         0.6)));

fig %>% layout(title = "Diamonds by cut, color and clarity", showlegend
         = F)%>%
  add_annotations( x=c(0.2, 0.5, 0.8), y=-0.05, text = c("Cut",
         "Color","Clarity"),
    font = list(size = 15),
    xref = "paper", yref = "paper", xanchor = "center", showarrow =
         FALSE); # https://plotly.com/r/reference/layout/annotations/
```





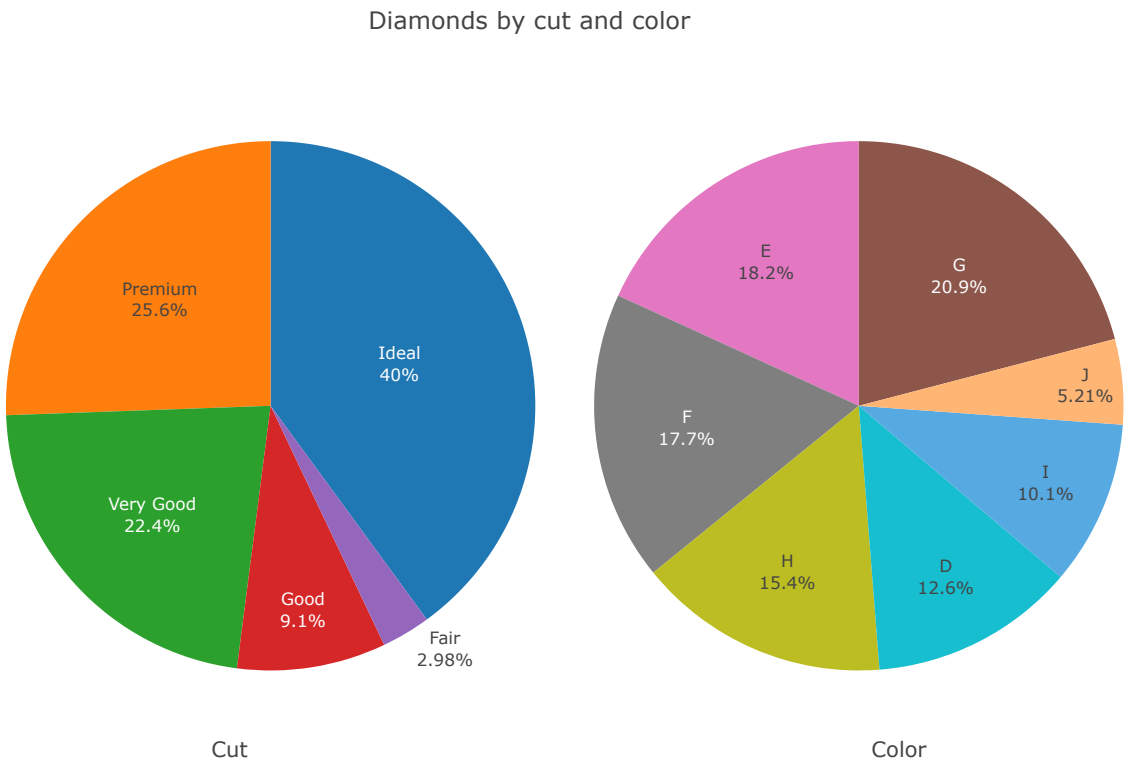
Cut Color Clarity

Data visualization with plotly

Pie/donut charts

- Subplots Using Grid

```
diamonds%>%group_by(cut)%>%summarize(freq = n())->data1;
diamonds%>%group_by(color)%>%summarize(freq = n())->data2;

fig <- plot_ly();
fig<-fig %>%add_pie(data=data1, labels = ~cut, values = ~freq,
         text=~cut, domain = list(row = 0, column = 0));

fig<-fig %>%add_pie(data=data2, labels = ~color, values =
         ~freq,text=~color, domain = list(row = 0, column = 1));

fig %>% layout(title = "Diamonds by cut and color", showlegend = F,
         grid=list(rows=1, columns=2)
)%>%
  add_annotations( x=c(0.2, 0.8), y=0, text = c("Cut", "Color"),
    font = list(size = 15),
    xref = "paper", yref = "paper", xanchor = "center", showarrow =
         FALSE);
```

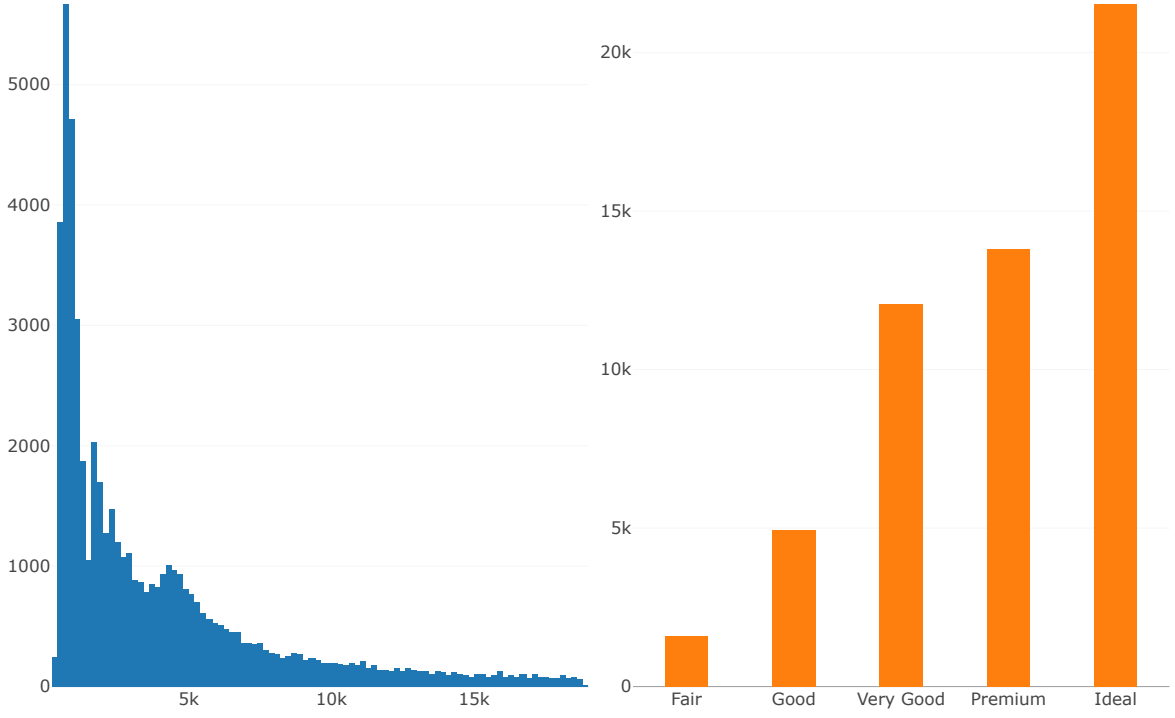


Data visualization with `plotly`

Bars and histograms

- The `add_bars()` and `add_histogram()` functions wrap the bar and histogram plotly.js trace types.
- The `subplot()` function provides a flexible interface for merging plotly objects into a single object (i.e., view).

```
library(dplyr)
p1 <- plot_ly(diamonds, x = ~price) %>% add_histogram();

p2 <- diamonds %>% group_by(cut)%>%summarise(freq=n())%>%
  plot_ly(x = ~cut, y = ~freq)%>%add_bars(width = 0.4);

fig <- subplot(p1, p2);
fig%>%hide_legend();  #removing the legend due to the merging
```

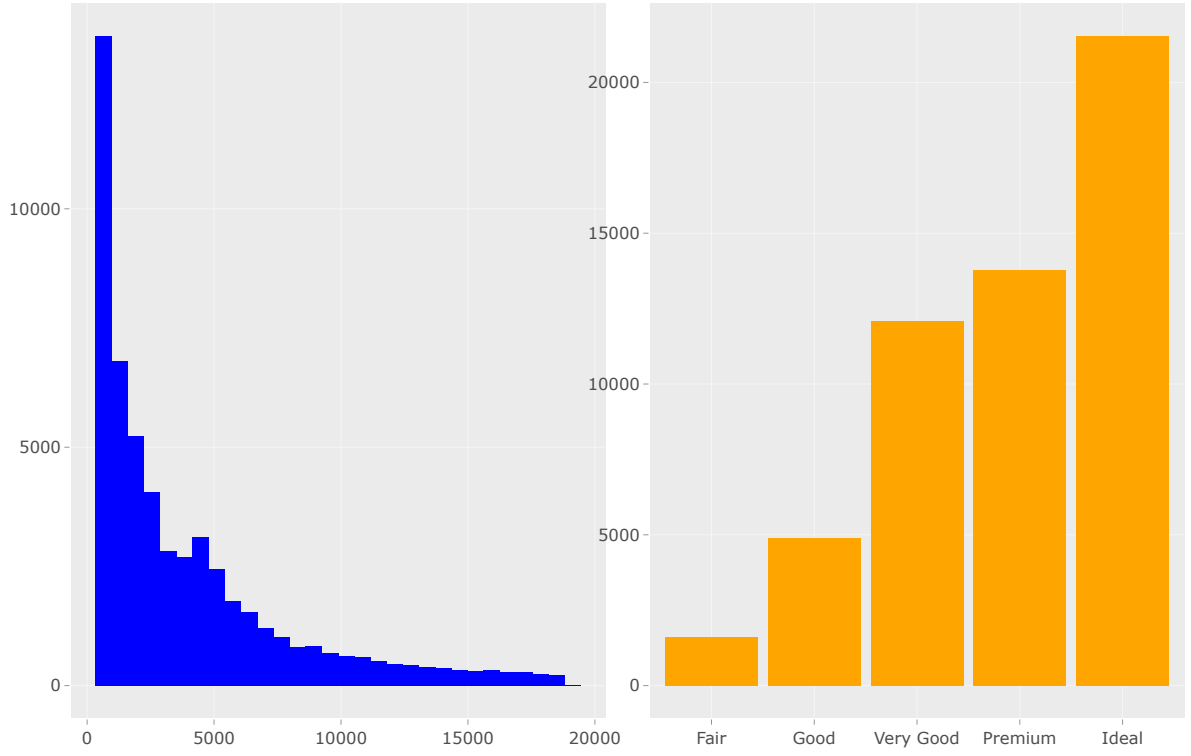


Data visualization with `plotly`

Bars and histograms

- using ggplot2

```
p1 <- ggplot(diamonds, aes(x = price)) +
  geom_histogram(fill='blue');
p2 <- ggplot(diamonds, aes(x = cut))+
  geom_bar(fill="orange");
ply1=ggplotly(p1)
ply2=ggplotly(p2)
subplot(ply1, ply2)
```

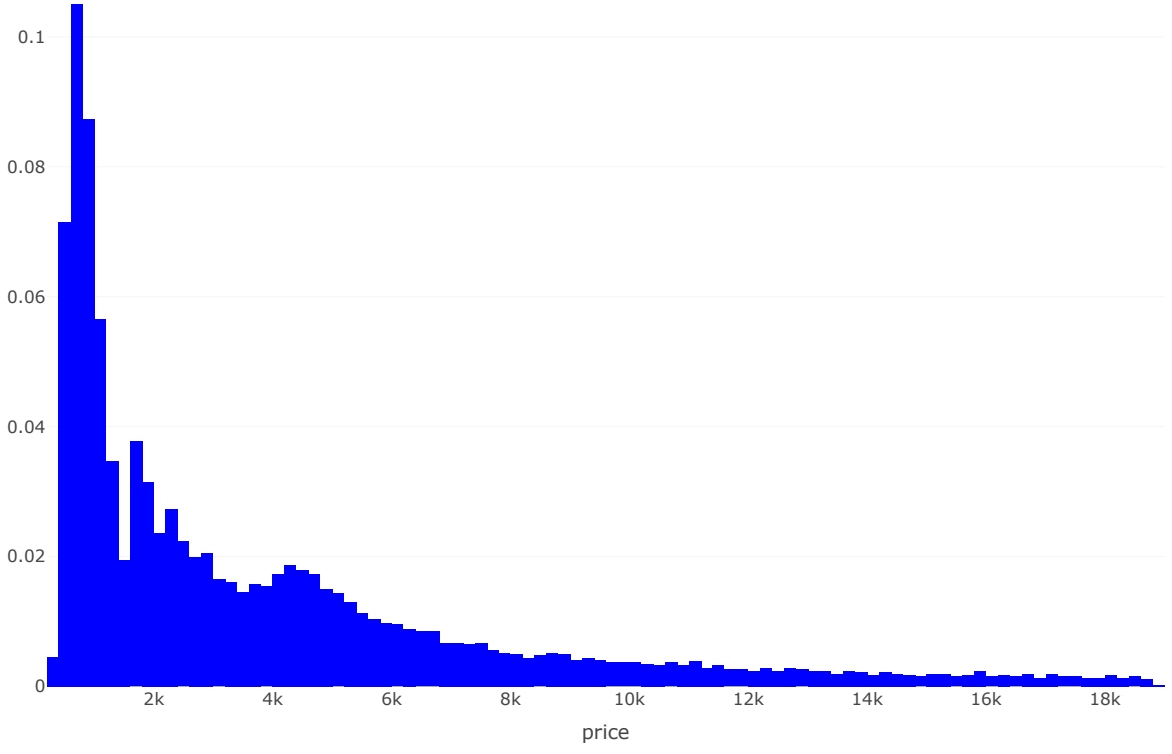


Data visualization with `plotly`

Bars and histograms

- Probability (relative frequency) histogram

```
plot_ly(diamonds, x = ~price, type = "histogram",histnorm =
          "probability", marker = list(color = "blue"));
```

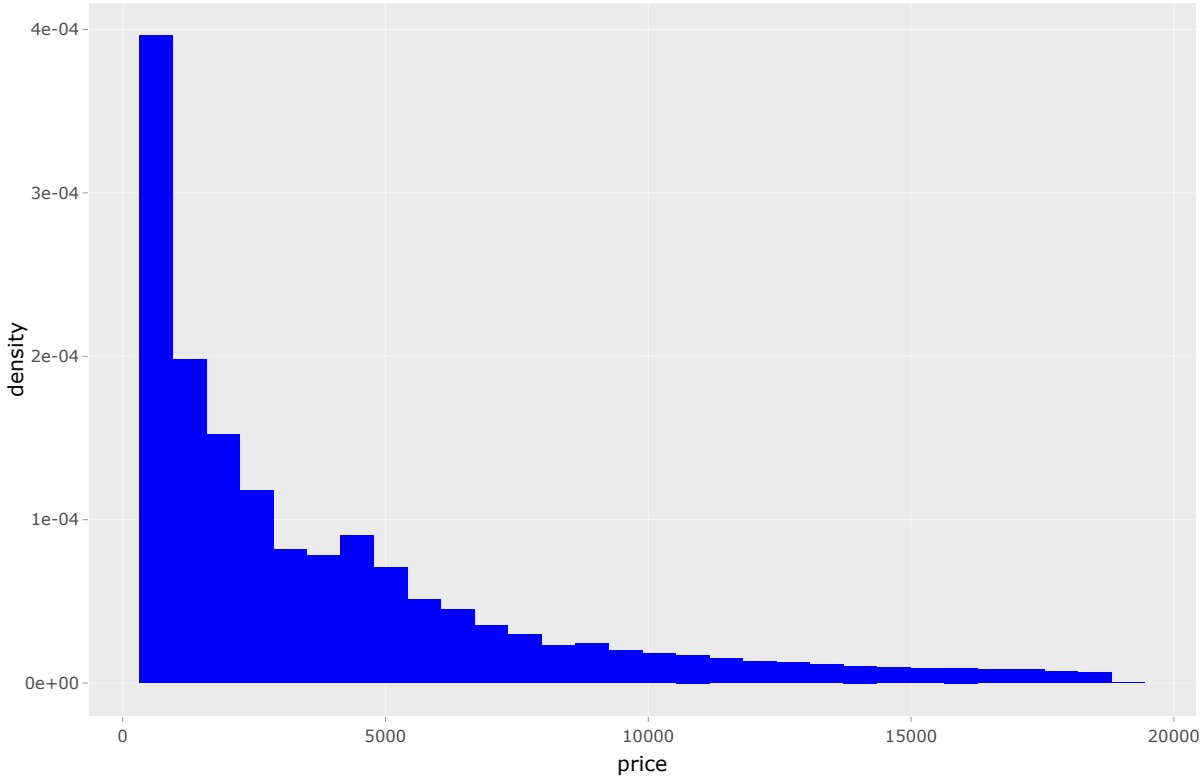


Data visualization with `plotly`

Bars and histograms

- using ggplot2

```
p=ggplot(diamonds, aes(x = price)) +
  geom_histogram(aes(y= ..density..), fill='blue');
ggplotly(p)
```



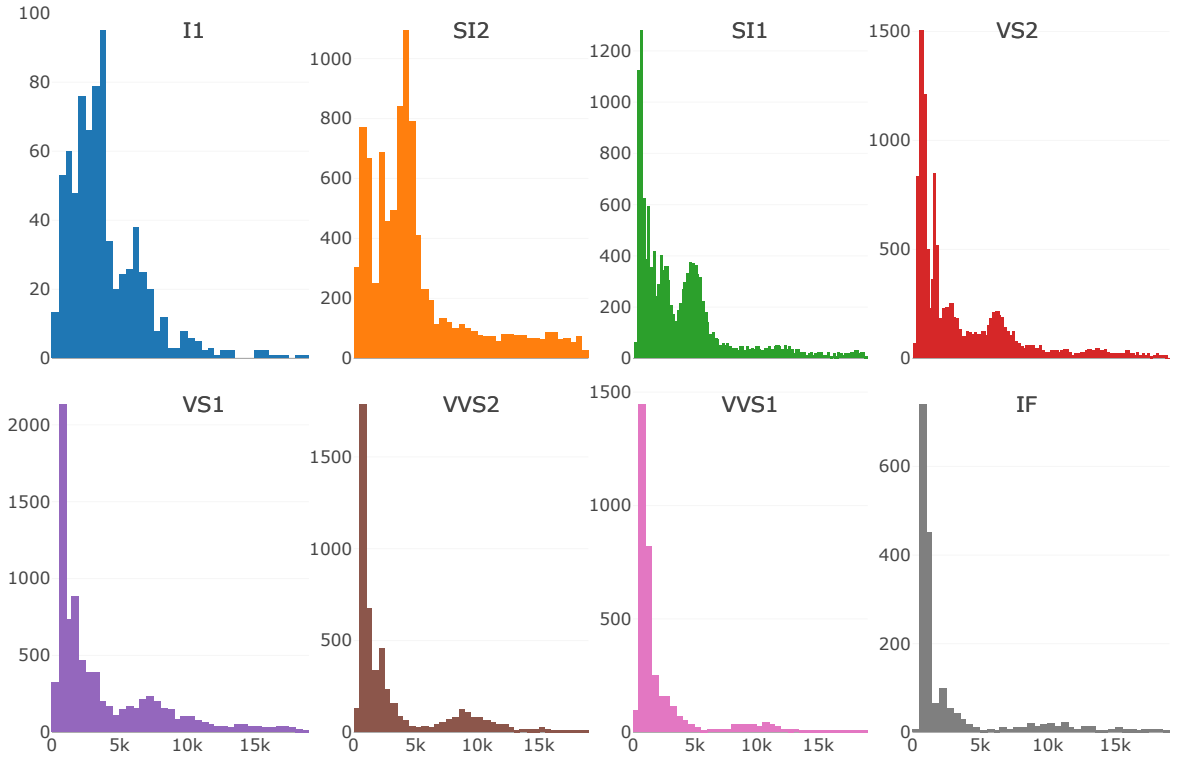
Data visualization with `plotly`

Bars and histograms

- display of diamond price by diamond clarity (like faceting method in ggplot2)

```
one_plot <- function(d){
  plot_ly(d, x = ~price) %>%
    add_histogram()%>%
    add_annotations(text=~unique(clarity), x = 0.5, y = 1,
      xref = "paper", yref = "paper", xanchor = "middle",
    yanchor = "top",showarrow = FALSE,
    font = list(size = 15, face="bold") )
  # https://plotly.com/r/reference/layout/annotations/
      }
```

```
diamonds%>%split(.$clarity)%>% #`split` divides the data by the factor
        variable
lapply(one_plot)%>%  # `lapply` applies the function
  subplot(nrows = 2, shareX = T, titleX = F) %>% # shareX specifies
        sharing xlab or not
  hide_legend()
```

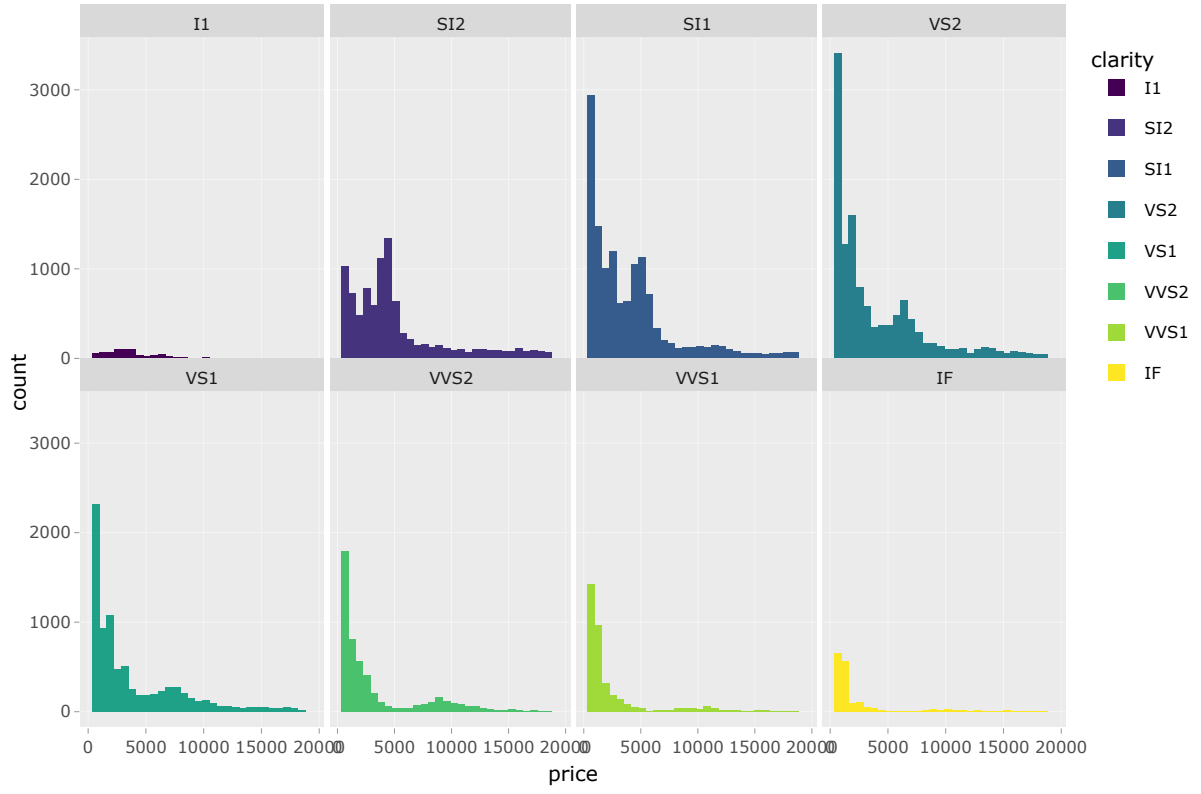


Data visualization with `plotly`

Bars and histograms

- display of diamond price by diamond clarity (using ggplot2)

```
p=ggplot(diamonds, aes(x = price,fill=clarity)) +
  geom_histogram()+
  facet_wrap(~clarity, nrow=2, ncol=4);
ggplotly(p);
```

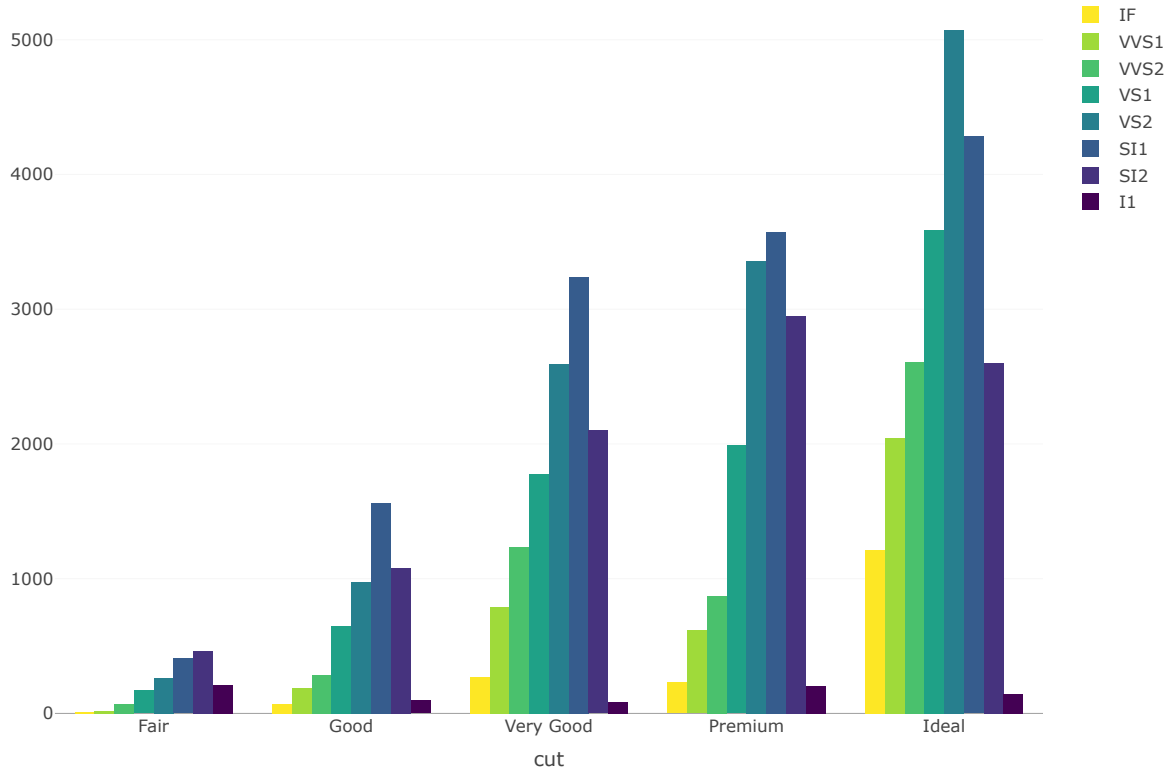


Data visualization with `plotly`

Bars and histograms

- Grouped Bar Charts

```
plot_ly(diamonds, x = ~cut, color = ~clarity) %>%
  add_histogram(); #both variables are categorical
```

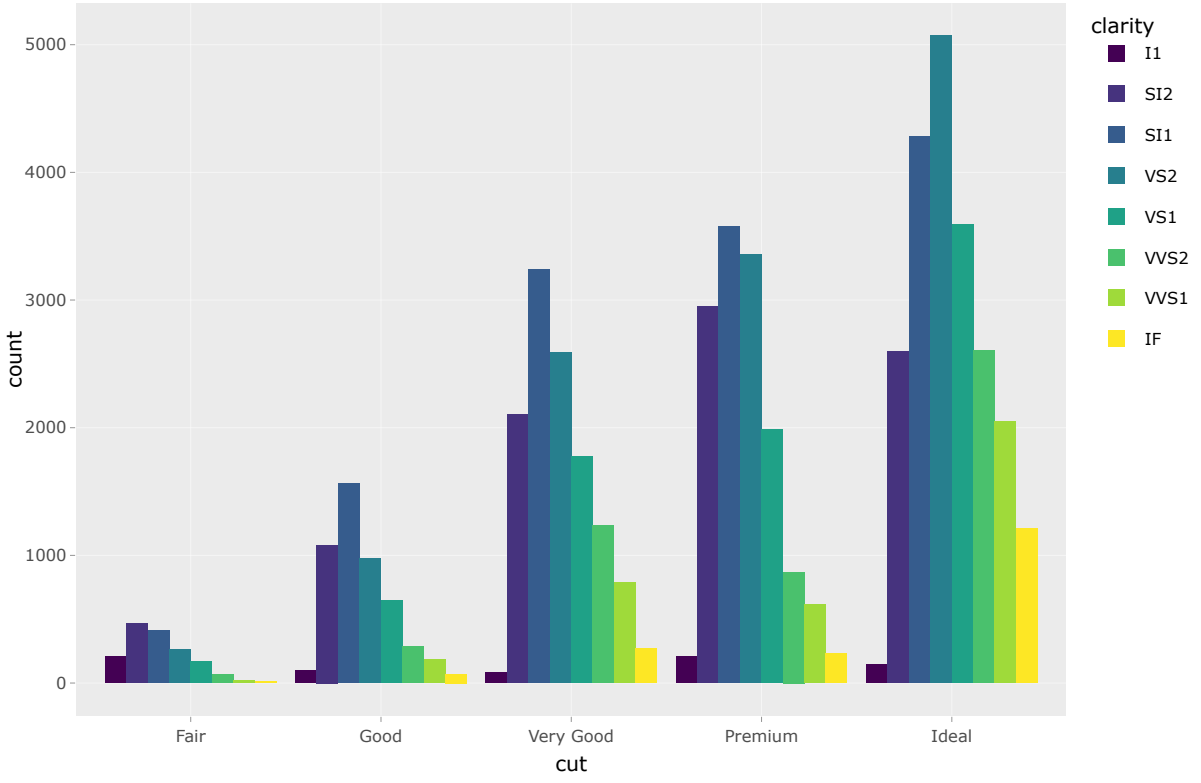


Data visualization with `plotly`

Bars and histograms

- Grouped Bar Charts using ggplot2

```
p=ggplot(diamonds, aes(x = cut, fill = clarity)) +
  geom_bar(position="dodge", stat = "count");  #position="stack"
ggplotly(p)
```

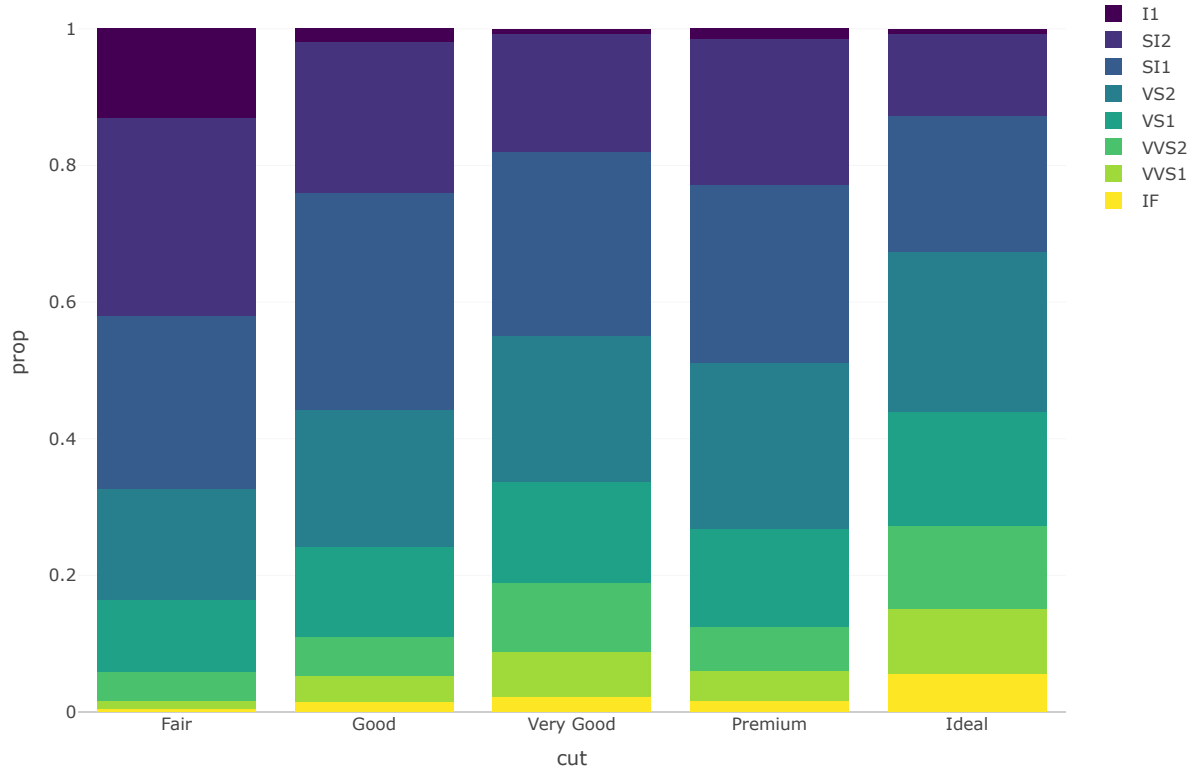


Data visualization with `plotly`

Bars and histograms

- Grouped Bar Charts
  - *we show the relative frequency of diamonds by clarity, given a cut.*

```
#diamonds%>%mutate(clarity=fct_rev(clarity));
diamonds%>%count(cut, clarity)%>%
         group_by(cut)%>%mutate(prop=n/sum(n))%>%
plot_ly(x = ~cut, y=~prop, color = ~clarity)%>%
  add_bars()%>%layout(barmode = "stack");
```

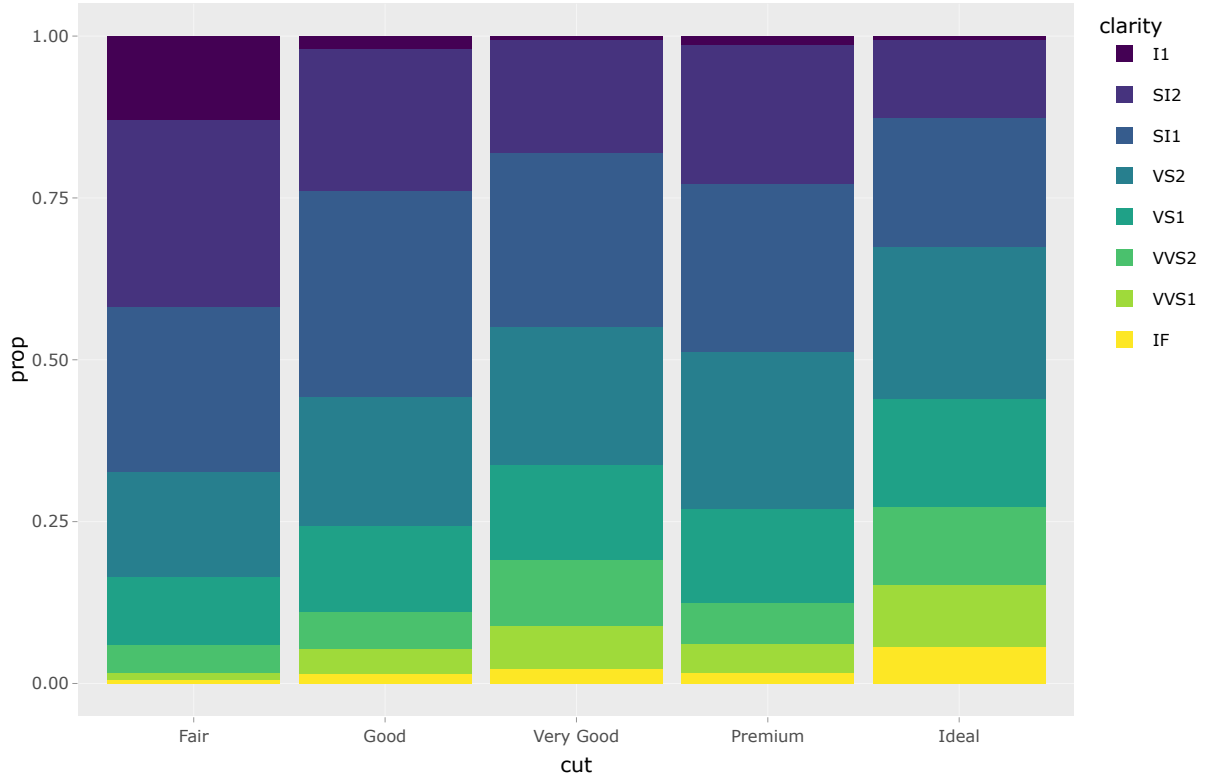


Data visualization with `plotly`

Bars and histograms

- Grouped Bar Charts using ggplot2 (relative frequency chart)

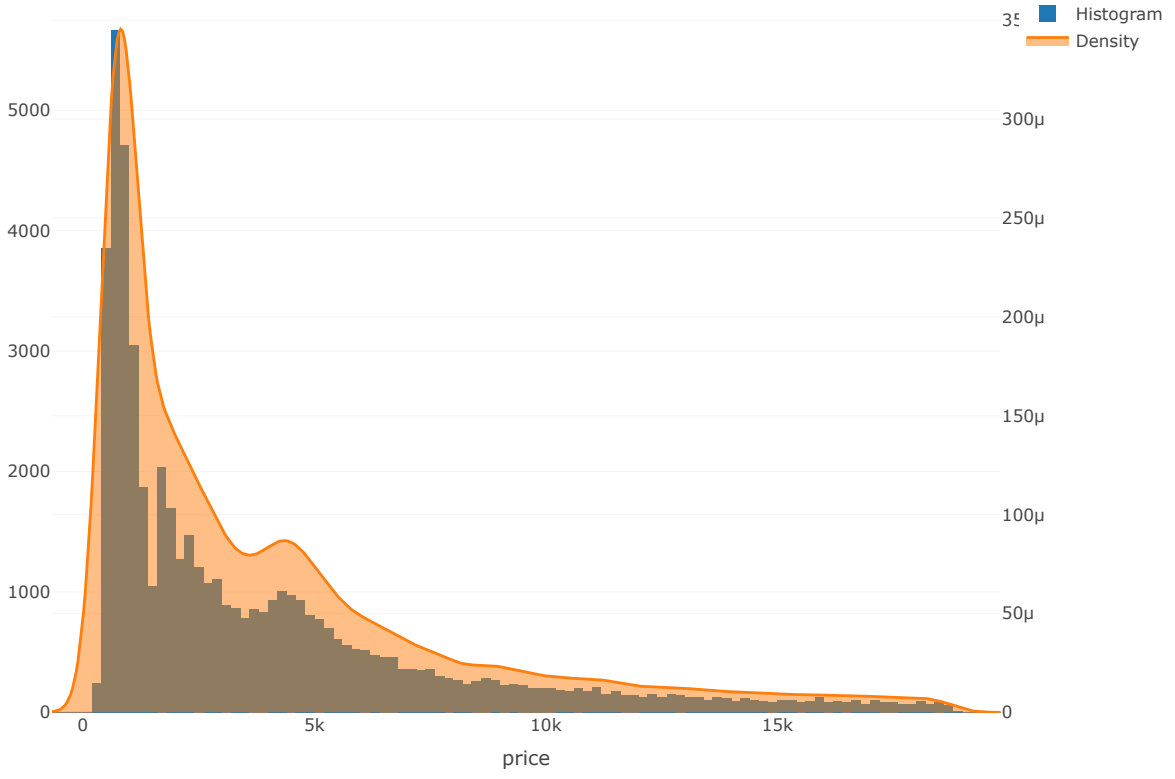
```
p=diamonds%>%count(cut, clarity)%>%
        group_by(cut)%>%mutate(prop=n/sum(n))%>%
ggplot(aes(x = cut, y=prop, fill = clarity)) +
  geom_bar(position="stack", stat = "identity");
ggplotly(p)
```



Data visualization with `plotly`

Density and histogram overlay

```
fit <- density(diamonds$price); #density estimation
diamonds%>%plot_ly(x =~price,type = "histogram", name = "Histogram")%>%
  add_trace(x = fit$x, y = fit$y, type= "scatter",mode = "lines", fill =
         "tozeroy", #Sets the area to fill with a solid color; "tozeroy"
         fill to  y=0: https://plotly.com/r/reference/scatter/#scatter-
         stackgroup
        yaxis = "y2",name =  "Density")%>%
  layout(yaxis2 = list(overlaying = "y", side = "right"));
```

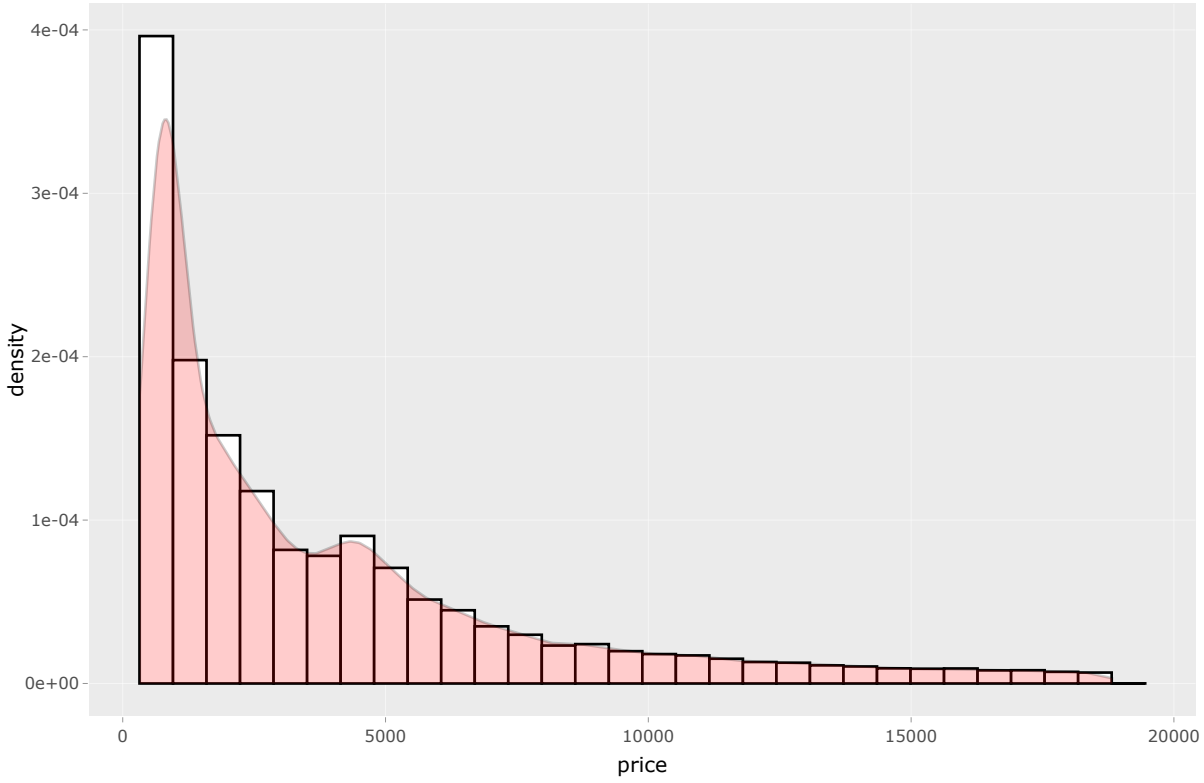


Data visualization with `plotly`

Density and histogram overlay

- using ggplot2

```
p=ggplot(data=diamonds, aes(x=price)) +
geom_histogram(aes(y= ..density..), color="black",fill="white")+
#Histogram with density instead of count on y-axis
geom_density(alpha=.2, fill="red");
ggplotly(p)
```

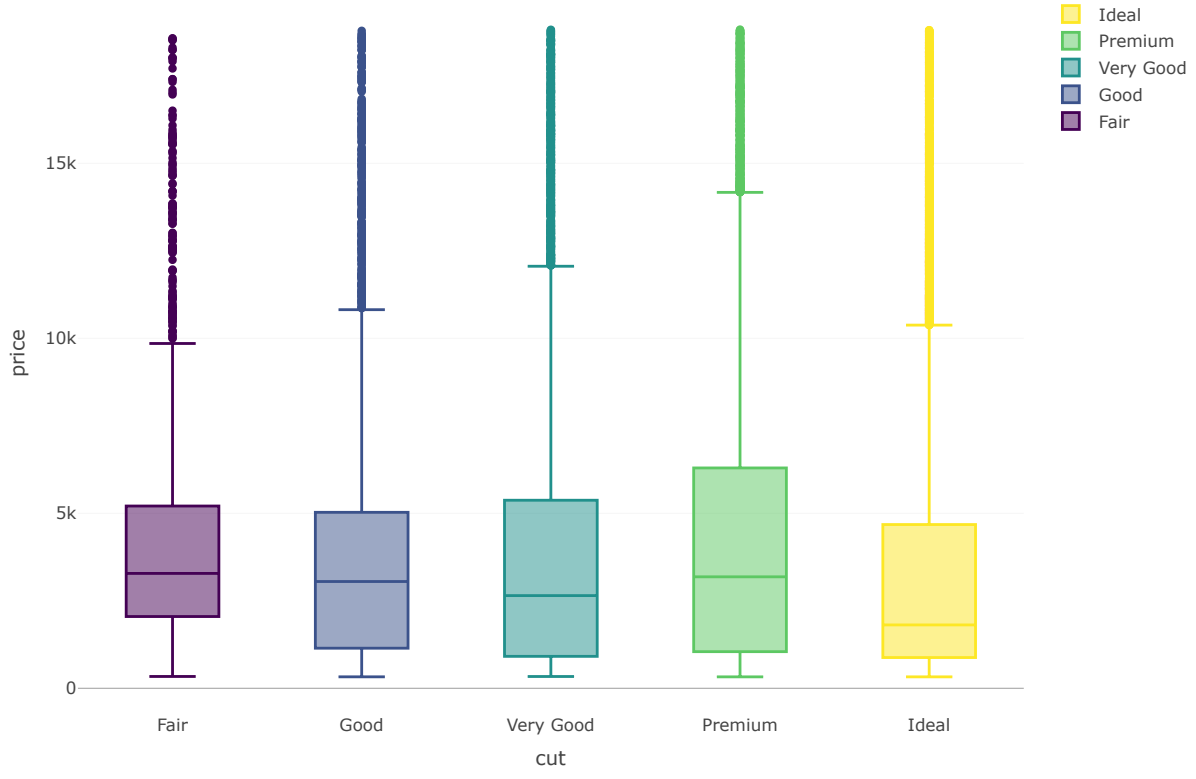


Data visualization with `plotly`

Boxplots

- Boxplots of `price` by factor `cut`

```
plot_ly(diamonds, x=~cut, y=~price, color=~cut)%>%
  add_boxplot()
```

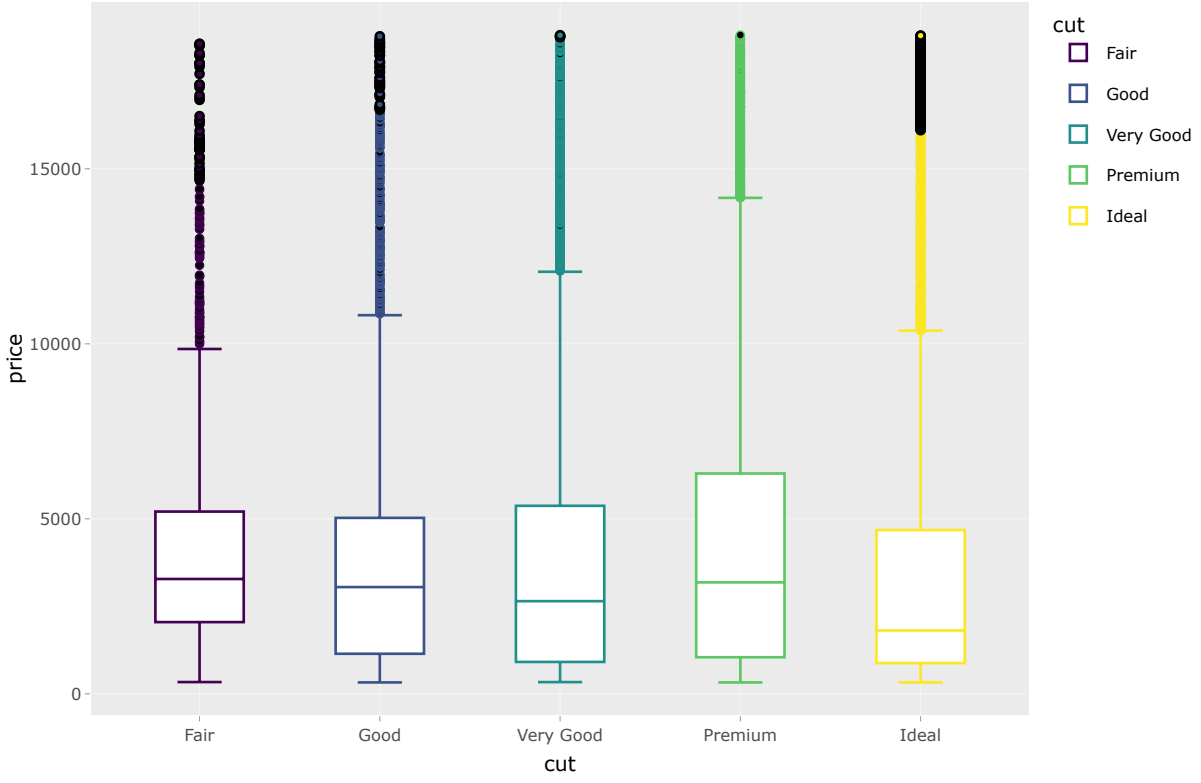


Data visualization with `plotly`

Boxplots

- Boxplots of `price` by factor `cut` using ggplot2

```
p=ggplot(diamonds, aes(x=cut, y=price, color=cut))+
  geom_boxplot();
ggplotly(p)
```

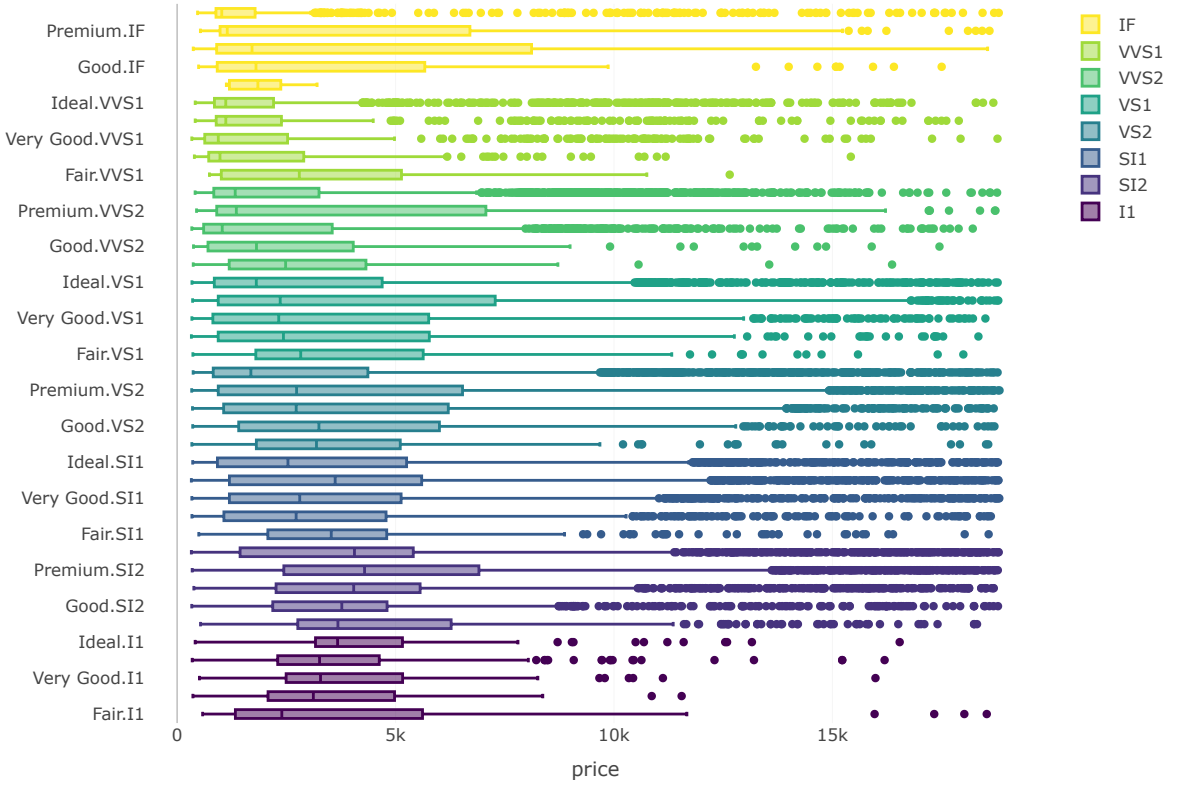


Data visualization with `plotly`

Boxplots

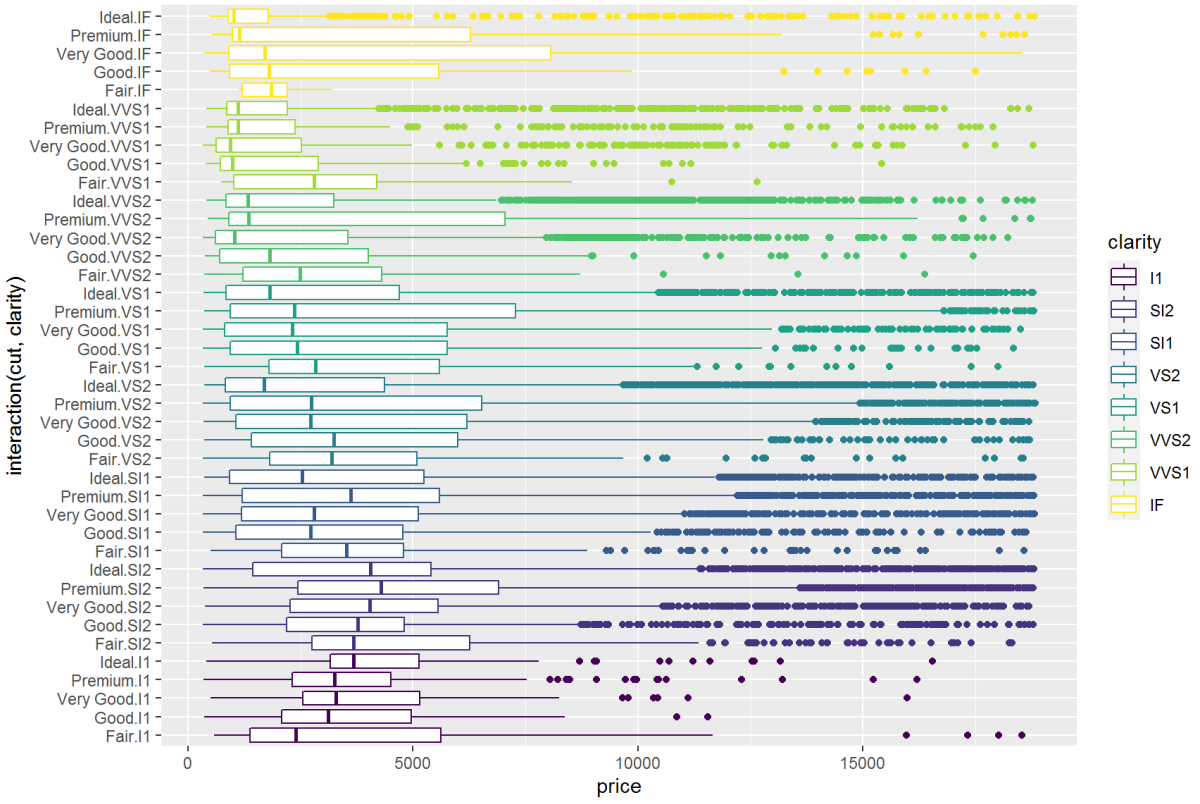
- Boxplots of `price` by factor `cut` and `clarity`

```
plot_ly(diamonds, x = ~price, y = ~interaction(cut, clarity)) %>%
  add_boxplot(color = ~clarity)%>%
layout(yaxis = list(title = "")) #remove the ylab
```



- ggplot2 does not do a good job. But the static graph looks good

```
p=ggplot(diamonds, aes(x=price, y=interaction(cut, clarity), color =
         clarity) )+
  geom_boxplot();
p;
```


Ideal.IF
Premium.IF
Very Good.IF
Good.IF
Fair.IF
Ideal.VVS1
Premium.VVS1
Very Good.VVS1
Good.VVS1
Fair.VVS1
Ideal.VVS2
Premium.VVS2
Very Good.VVS2
Good.VVS2
Fair.VVS2
Ideal.VS1
Premium.VS1
Very Good.VS1
Good.VS1
Fair.VS1
Ideal.VS2
Premium.VS2
Very Good.VS2
Good.VS2
Fair.VS2
Ideal.SI1
Premium.SI1
Very Good.SI1
Good.SI1
Fair.SI1
Ideal.SI2
Premium.SI2
Very Good.SI2
Good.SI2
Fair.SI2
Ideal.I1
Premium.I1
Very Good.I1
Good.I1
Fair.I1
interaction(cut, clarity)
0
5000
10000
15000
price
clarity
I1
SI2
SI1
VS2
VS1
VVS2
VVS1
IF

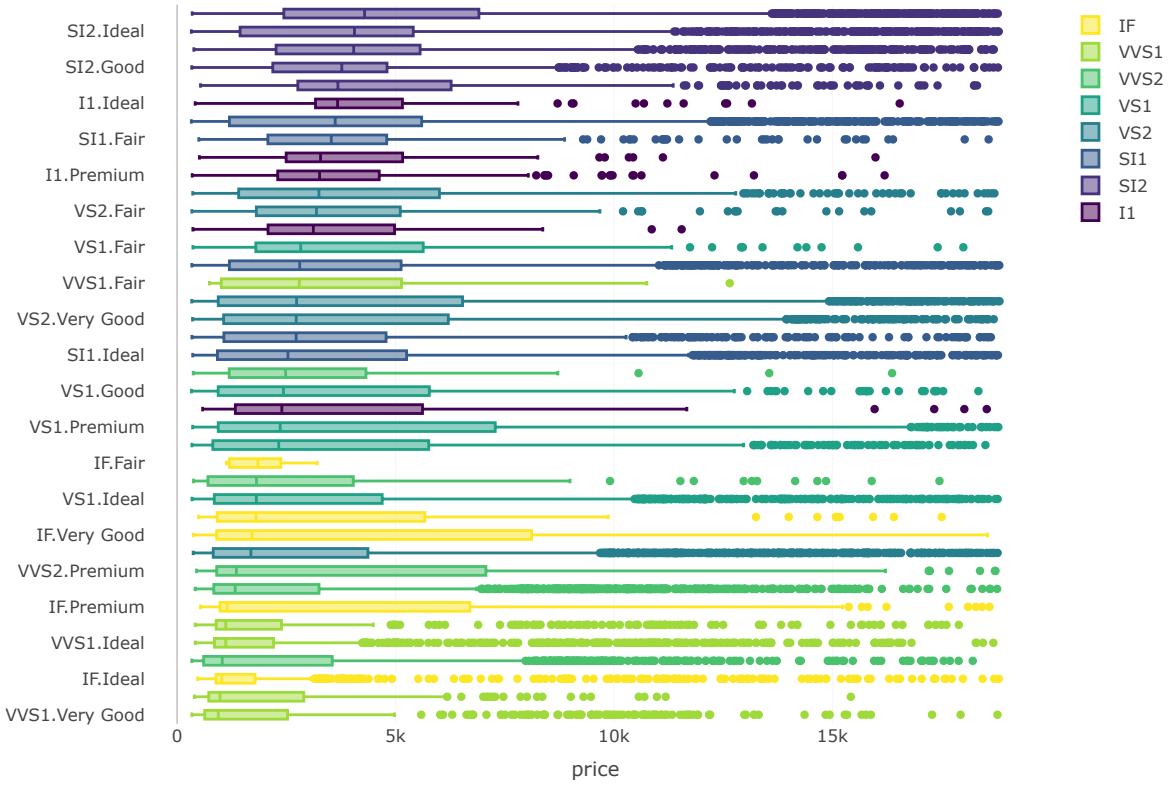
```
#ggplotly(p)
```

Data visualization with `plotly`

Boxplots

- Sort the boxplots by medians

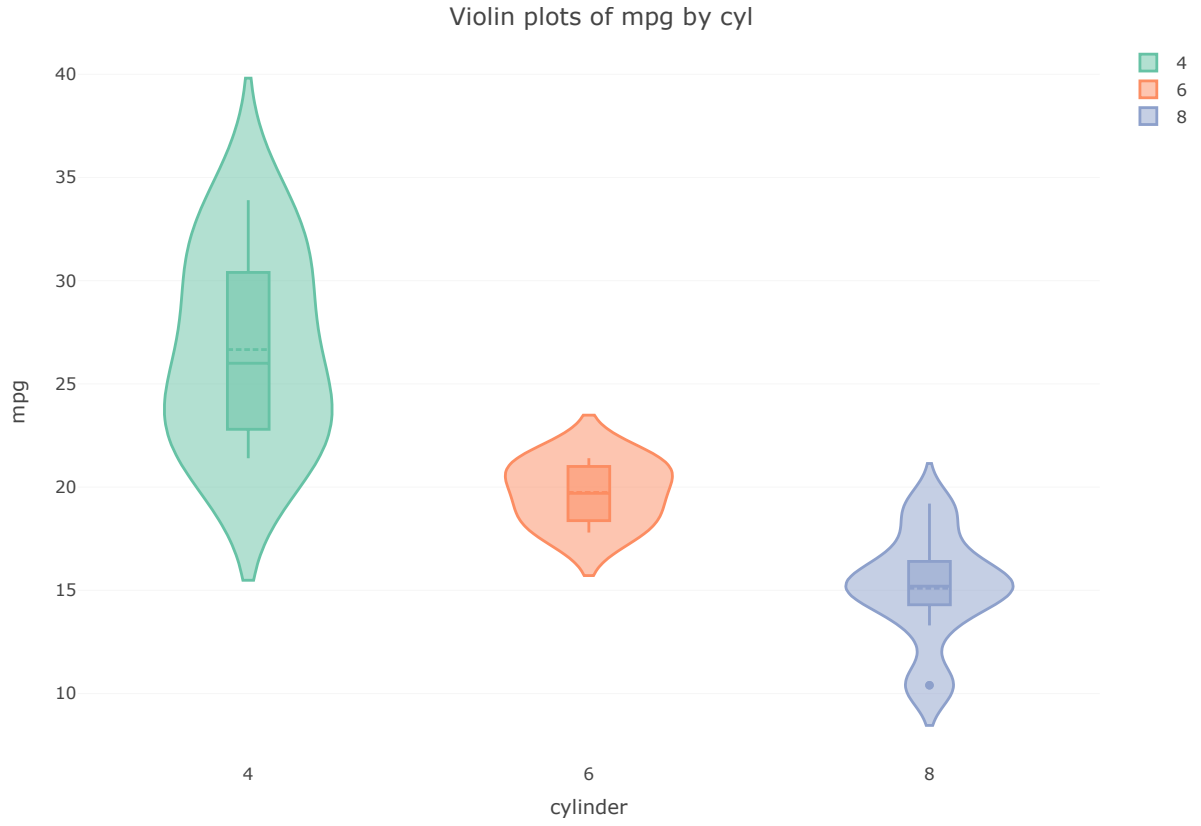
```
diamonds=diamonds%>%mutate(cc = interaction(clarity, cut));
# interaction levels sorted by median price
lvls <- diamonds%>%group_by(cc)%>%summarise(m =
          median(price))%>%arrange(m) %>% #order the medians
  pull(cc); #extract the column cc

plot_ly(diamonds, x=~price, y=~factor(cc, lvls))%>% # relevel the factor
          cc
  add_boxplot(color=~clarity)%>%
  layout(yaxis=list(title=""));
```



Data visualization with `plotly`

Violin plots

```
mtcars$cyl=factor(mtcars$cyl);
mtcars%>%plot_ly(y=~mpg, type='violin',color=~cyl, #plot by factor cyl
          box = list(visible = T), #box in the boxplot
          meanline = list(visible = T) #average value
  ) %>%
  layout( title = "Violin plots of mpg by cyl",
    xaxis = list( title = "cylinder")
    );
```

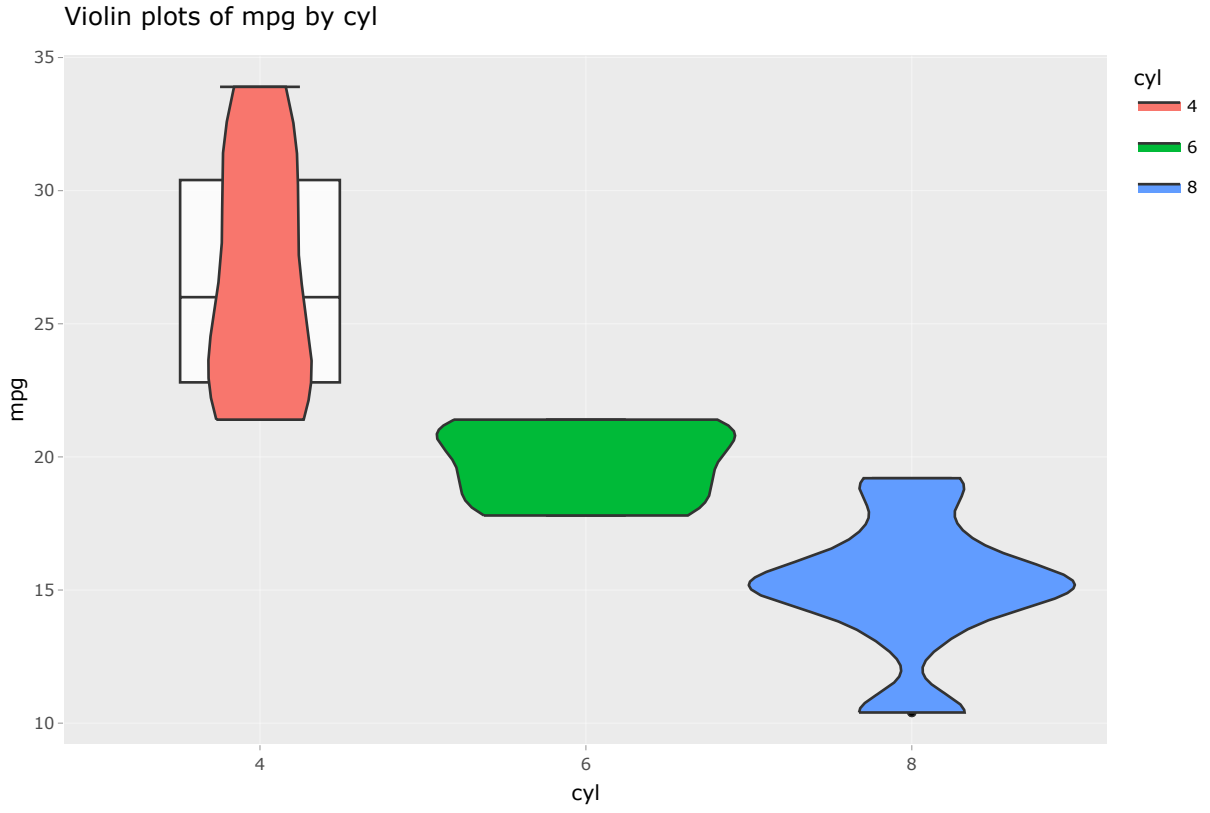


Data visualization with `plotly`

Violin plots

- using ggplot2
  - *Not all boxes are shown* 😢

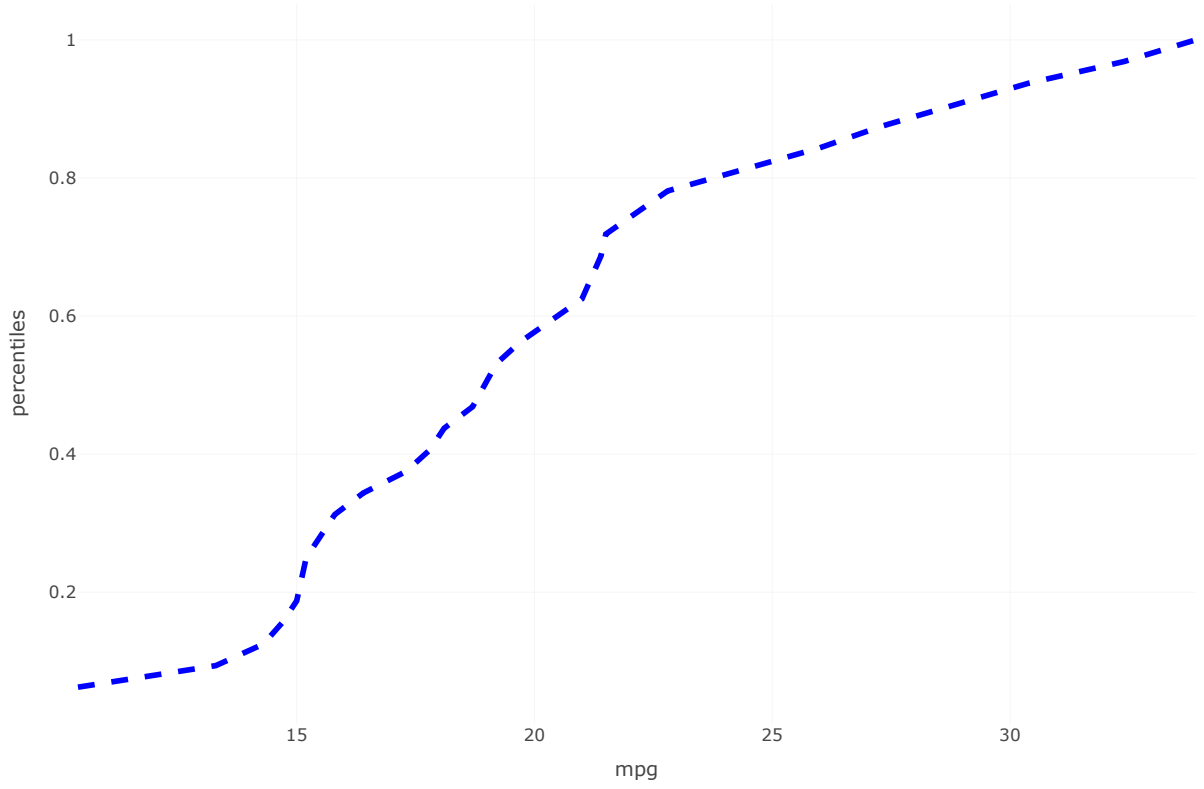
```
p=ggplot(data=mtcars, aes(x=cyl,y=mpg) ) +
  geom_violin(aes(fill=cyl))+
  geom_boxplot(width = 0.1, alpha=0.8)+
  labs(title = "Violin plots of mpg by cyl");
ggplotly(p);
```



Data visualization with `plotly`

Cumulative Frequencies

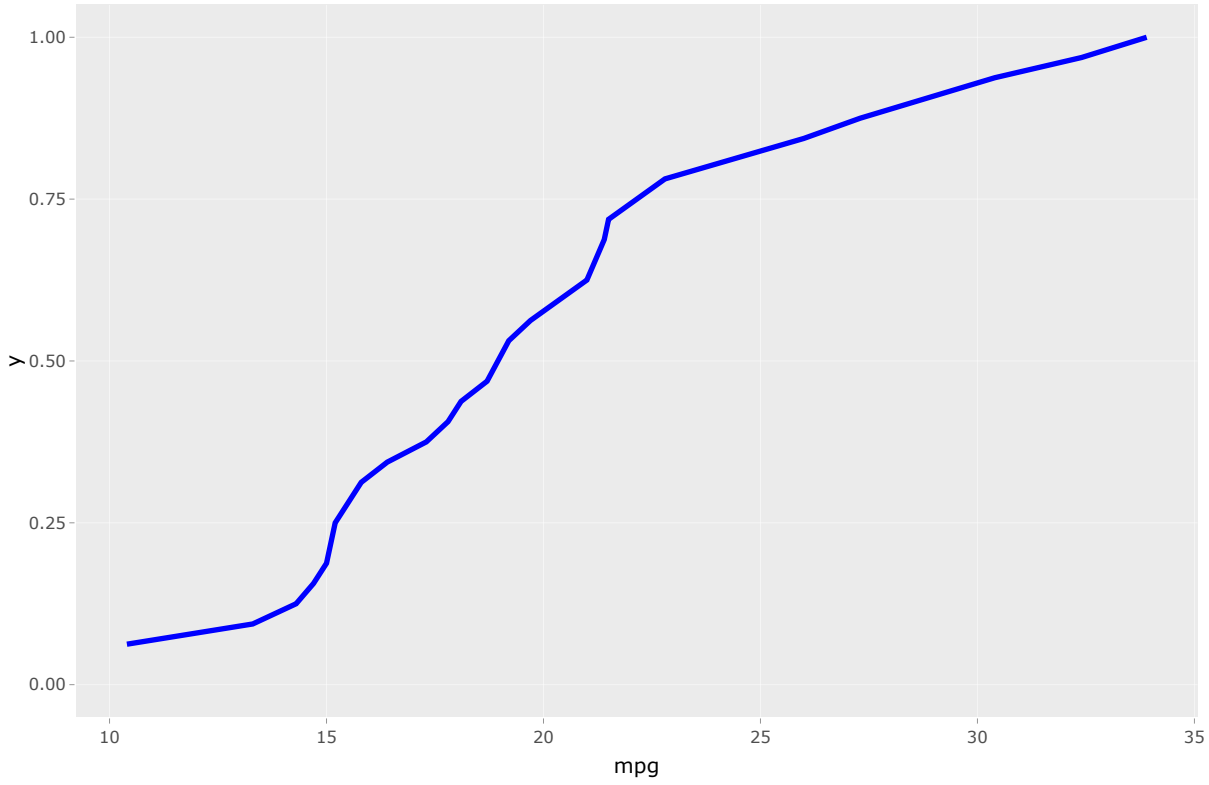
```
mtcars %>%arrange(mpg)->mtcars1;
Fn=ecdf(mtcars1$mpg); # ecdf returns a *function*
mtcars1<-mtcars1%>%mutate(percentiles=Fn(mpg));
#Fn(mtcars1$mpg) #returns the percentiles
mtcars1%>%plot_ly(x = ~mpg, y = ~percentiles,  type = 'scatter', mode =
          'lines', name="cdf",
    line = list(width = 4, color="blue",dash = "dash"));
```



Data visualization with `plotly`

Cumulative Frequencies using ggplot2

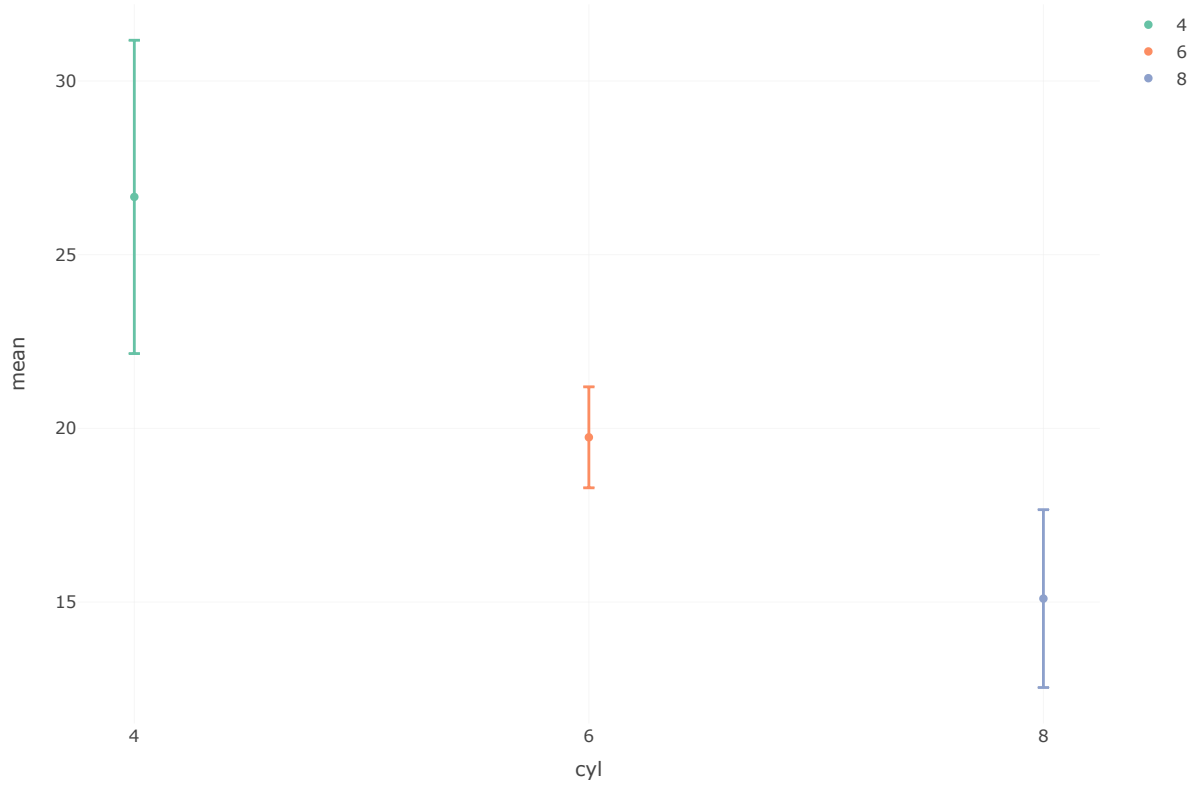
```
p=ggplot(mtcars, aes(x=mpg)) +
stat_ecdf(geom = "line", color="blue", size=1);
ggplotly(p);
```



Data visualization with `plotly`

Error bars

```
mtcars%>%mutate(cyl=factor(cyl))%>%group_by(cyl)%>%summarise(mean=mean(mpg),
          sd=sd(mpg)) %>%
  plot_ly(x=~cyl, y=~mean, color = ~cyl, type = "scatter",
          mode="markers", error_y = ~list(array = sd) );
```

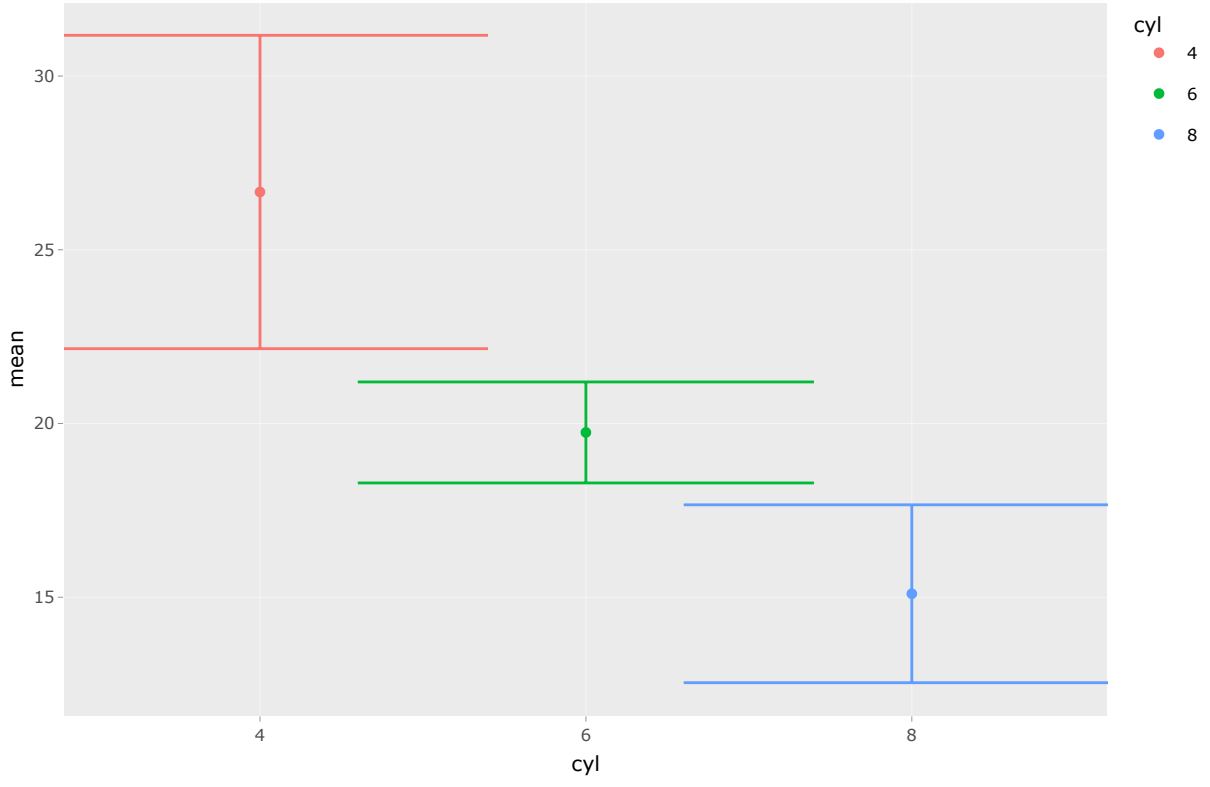


Data visualization with `plotly`

Error bars

- using ggplot2

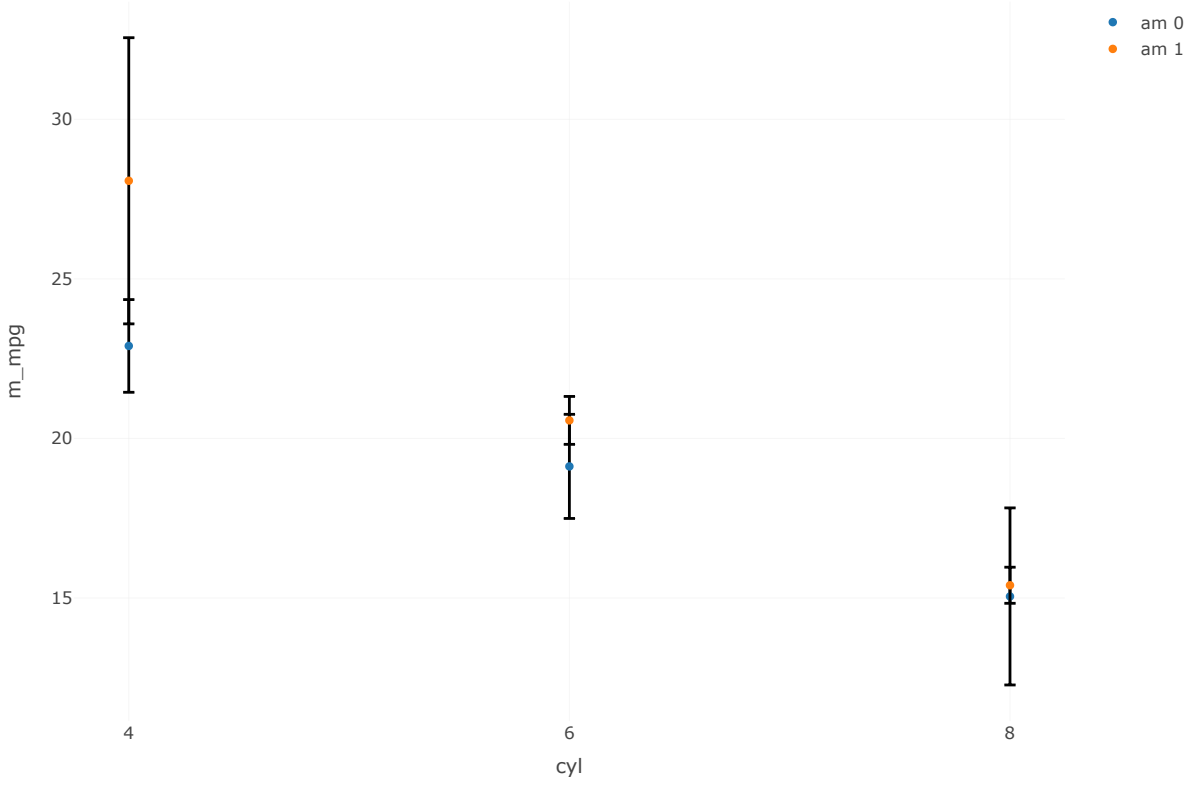
```
p<-mtcars%>%mutate(cyl=factor(cyl))%>%
  group_by(cyl)%>%summarise(mean=mean(mpg),sd=sd(mpg))%>%
  ggplot(aes(x=cyl,y=mean, color=cyl)) +
  geom_point()+
  geom_errorbar(aes(ymin=mean-sd, ymax=mean+sd));
ggplotly(p);
```



Data visualization with `plotly`

Error bars

```
mtcars2<-mtcars%>%mutate(cyl=factor(cyl), am=factor(am))%>%
    group_by(cyl,am)%>%summarise(m_mpg=mean(mpg),sd_mpg=sd(mpg));
plot_ly(data=mtcars2[which(mtcars2$am == '0'),], x=~cyl, y=~m_mpg,
        type = 'scatter', mode = 'markers', name = 'am 0',
        error_y = ~list(array = sd_mpg,color = '#000000')) %>%
  add_trace(data=mtcars2[which(mtcars2$am == '1'),], name = 'am 1');
```

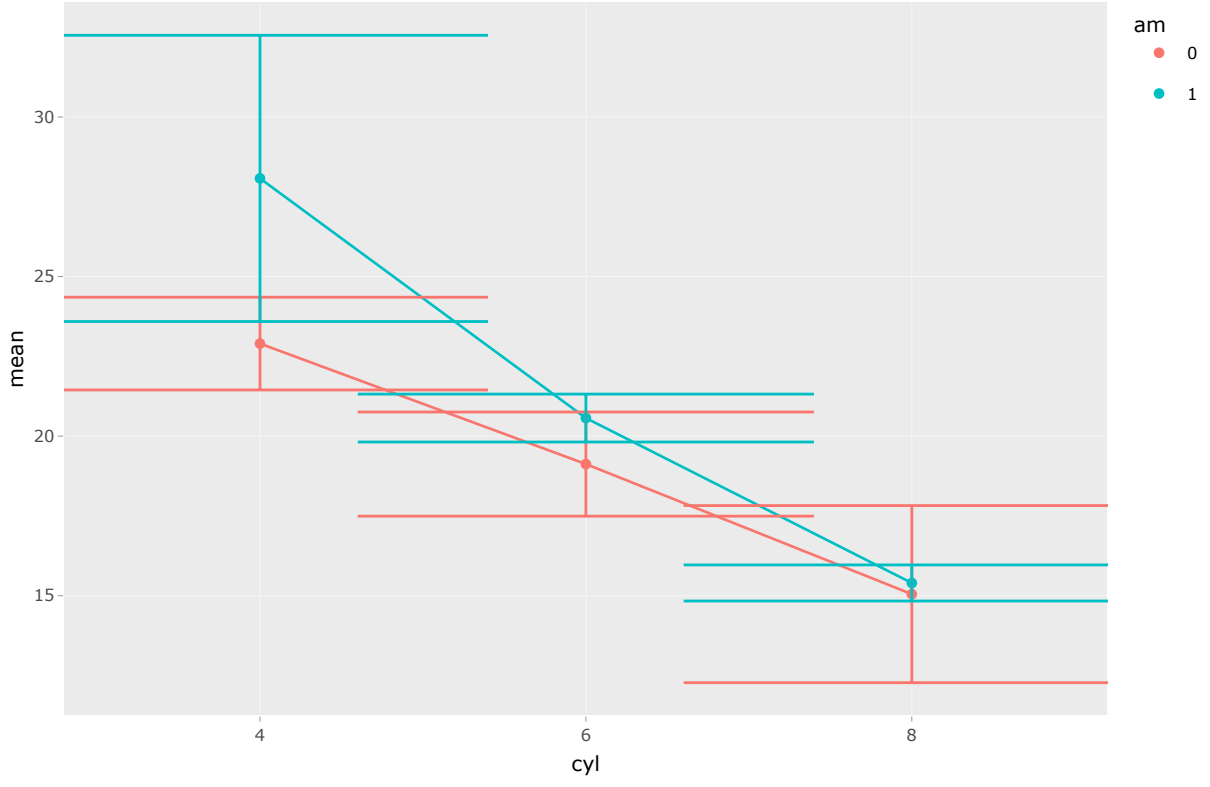


Data visualization with `plotly`

Error bars

- using ggplot2

```
p=mtcars%>%mutate(cyl=factor(cyl), am=factor(am))%>%
    group_by(cyl,am)%>%summarise(mean=mean(mpg),sd=sd(mpg))%>%
  ggplot(aes(x=cyl,y=mean, color=am)) + geom_point()+
geom_line(aes(group=am))+
geom_errorbar(aes(ymin=mean-sd, ymax=mean+sd));
ggplotly(p);
```

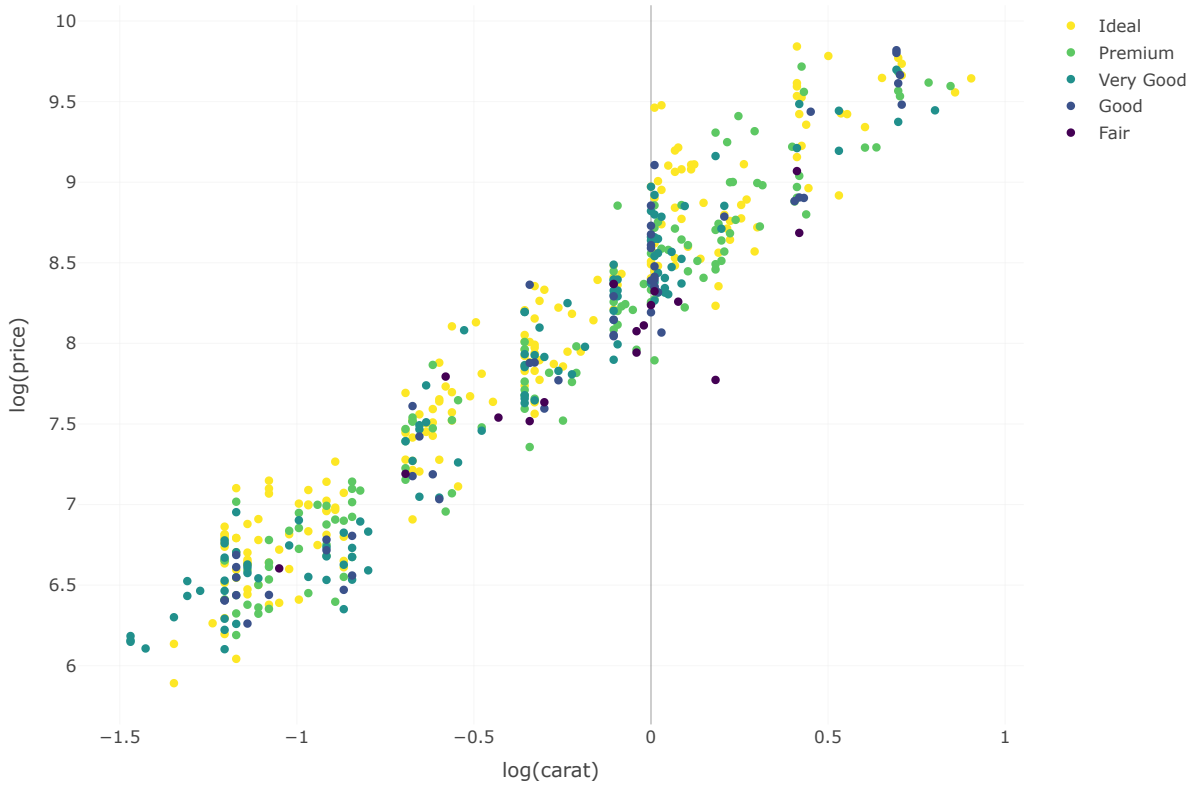


Data visualization with `plotly`

Scatter plot

- For scatter plot matrix, please see https://plotly.com/r/splom/
- Please see our first plotly example

```
data(diamonds, package = "ggplot2");
set.seed(26);
diamonds2 = sample_n(diamonds,size=500);
diamonds2%>%plot_ly(x = ~log(carat), y = ~log(price),color=~cut,
                    type = "scatter",mode = "markers");
```

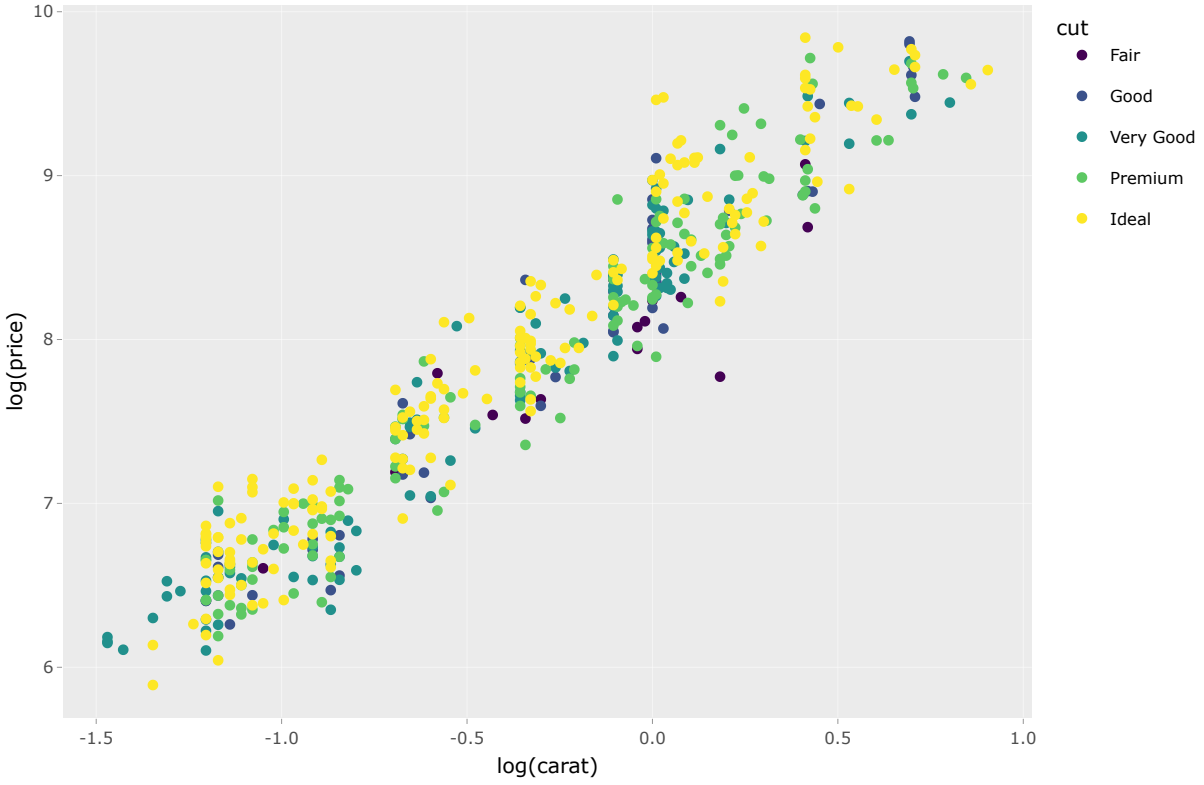


Data visualization with `plotly`

Scatter plot using ggplot2

- We have seen this

```
p <- ggplot(diamonds2, aes(x = log(carat), y = log(price),color=cut))+
  geom_point();
ggplotly(p);
```



Data visualization with `plotly`

Scatter plot

- 3d scatter plots

```
mtcars%>%mutate(cyl=factor(cyl))%>%
  plot_ly(x = ~mpg, y = ~disp, z = ~cyl) %>%
  add_markers(color = ~cyl)
```

Data visualization with `plotly`

3d scatter plots

```
mtcars%>%mutate(cyl=factor(cyl))%>%
  plot_ly(x = ~mpg, y = ~disp, z = ~wt,color = ~cyl) %>%
  add_markers();
```

Data visualization with `plotly`

3d scatter plots

```
mtcars%>%mutate(cyl=factor(cyl))%>%
  plot_ly(x = ~mpg, y = ~disp, z = ~wt, color = ~cyl) %>%
  add_markers()%>%add_lines();
```

Buttons

Dropdown events

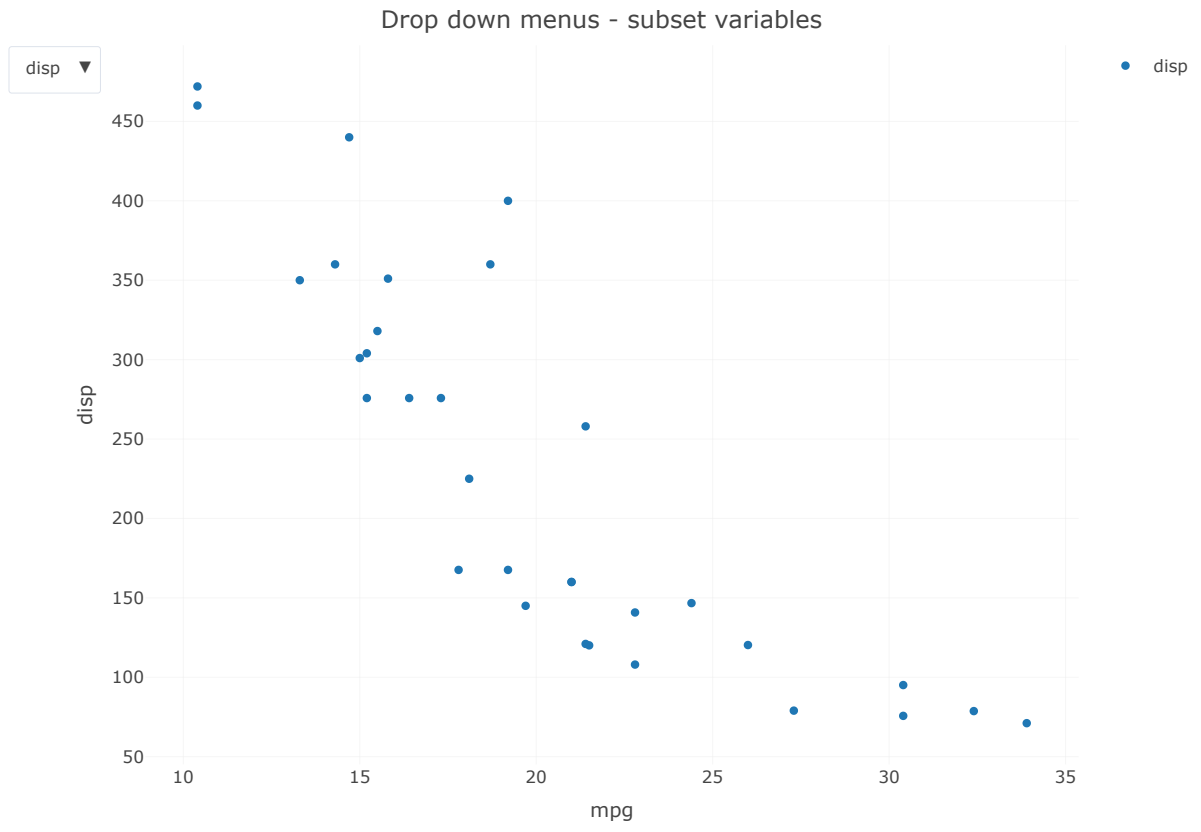
- The updatemenu method determines which plotly.js function will be used to modify the chart. There are 4 possible methods:
  - *`restyle`: modify data or data attributes*
  - *`update`: modify data and layout attributes*
  - *`relayout`: modify layout attributes*
  - *`animate`: start or pause an animation (only available offline)*

```
p=mtcars%>%plot_ly(x = ~mpg, y = ~disp,
             name='disp', type='scatter', mode='markers') %>%
  add_trace(y = ~hp, name = 'hp', type='scatter', mode='markers',
            visible=FALSE) %>%
  add_trace(y = ~wt, name = 'wt', type='scatter', mode='markers',
            visible=FALSE) %>%
 layout(
   title = "Drop down menus - subset variables",
   yaxis = list(title = "disp"),
             updatemenus = list(
             list(
               type= 'dropdownlist',
           buttons = list(
           list(method = "update",
                args = list(list(visible = list(TRUE, FALSE, FALSE)),
                            list(yaxis = list(title = "disp"))),
                label = "disp"),
           list(method = "update",
                args = list(list(visible = list(FALSE, TRUE, FALSE)),
                            list(yaxis = list(title = "hp"))),
                label = "hp"),
           list(method = "update",
                args = list(list(visible = list(FALSE, FALSE, TRUE)),
                            list(yaxis = list(title = "wt"))),
                label = "wt")
           )
         )
       )
    )
```

Buttons

Dropdown events

p

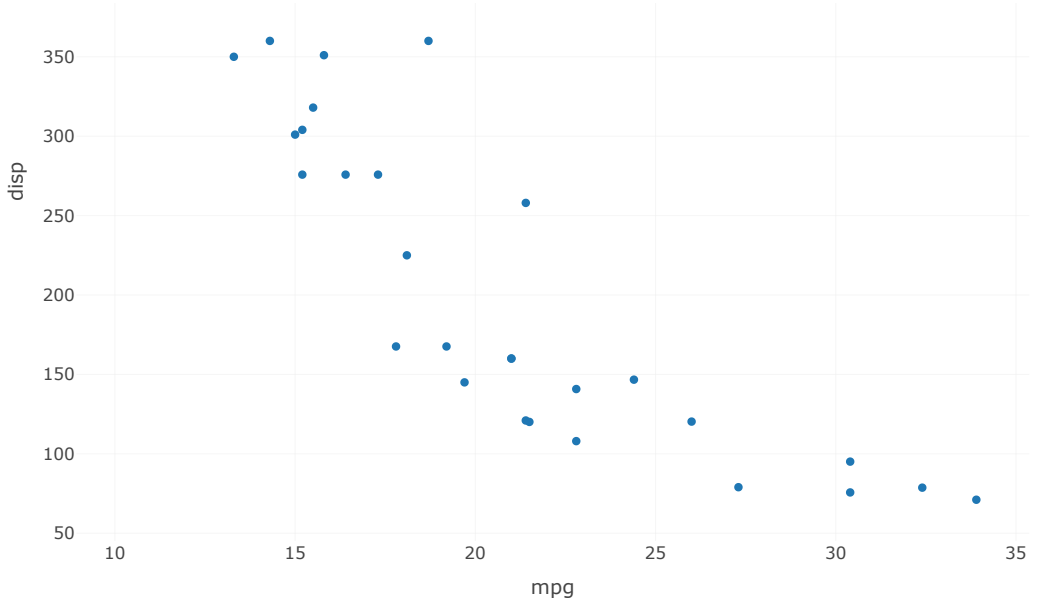


Buttons

Dropdown events

- Try to change type `type= 'buttons',`
- Read https://plotly.com/r/custom-buttons/ for more.

```
mtcars%>%plot_ly(x = ~mpg, y = ~disp,
             name='disp', type='scatter', mode='markers') %>%
  add_trace(y = ~hp, name = 'hp', type='scatter', mode='markers',
            visible=FALSE) %>%
  add_trace(y = ~wt, name = 'wt', type='scatter', mode='markers',
            visible=FALSE) %>%
 layout(
   title = "subset variables",
   yaxis = list(title = "disp"),
            updatemenus = list(
            list(
              type= 'buttons',
           buttons = list(
           list(method = "update",
                args = list(list(visible = list(TRUE, FALSE, FALSE)),
                            list(yaxis = list(title = "disp"))),
                label = "disp"),
           list(method = "update",
                args = list(list(visible = list(FALSE, TRUE, FALSE)),
                            list(yaxis = list(title = "hp"))),
                label = "hp"),
           list(method = "update",
                args = list(list(visible = list(FALSE, FALSE, TRUE)),
                            list(yaxis = list(title = "wt"))),
                label = "wt")
           )
         )
       )
    )
```


disp
350
300
250
200
150
100
50
10
15
20
25
30
35
mpg

License