



UITS RT WORKSHOPS

# Introduction to Information Visualization

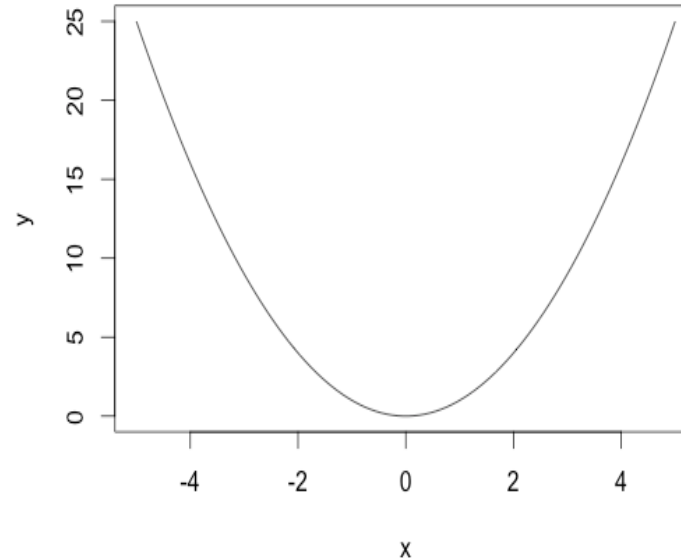
INDIANA UNIVERSITY

Jefferson Davis, Research Analytics

# Statistical Graphics

# Statistical Graphics

x	$y=x^2$
0.0	0.00
0.1	0.01
0.2	0.04
0.3	0.09
0.4	0.16
0.5	0.25
0.6	0.36
0.7	0.49
0.8	0.64
0.9	0.81
1.0	1.00
1.1	1.21
1.2	1.44
1.3	1.69

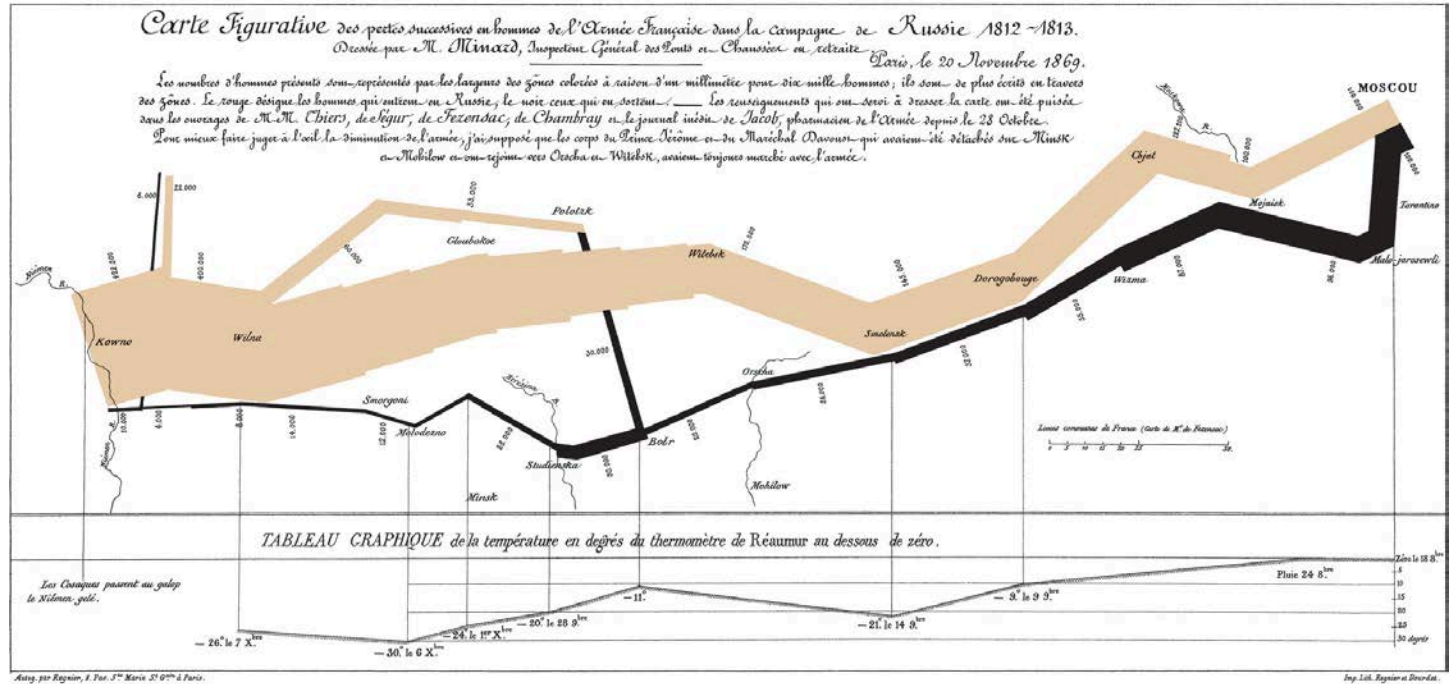


# Statistical Graphics

Graphics can convey meaning without displaying any particular quantitative data.



# Statistical Graphics



# Statistical Graphics



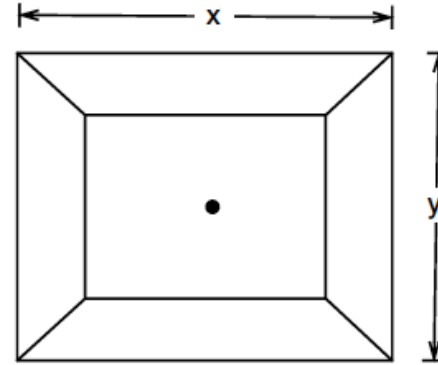
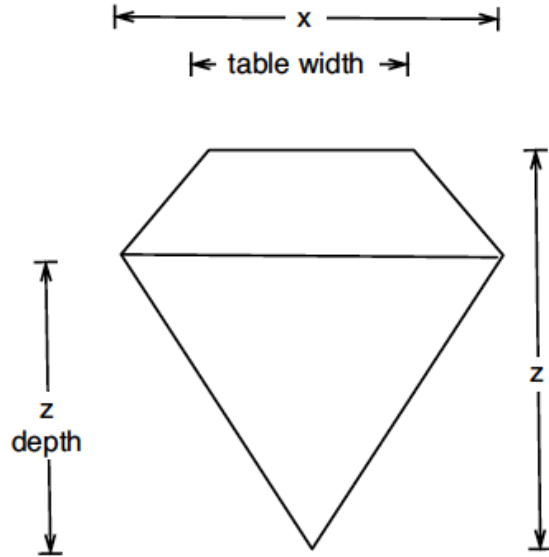
# The dataset

## The diamond dataset

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	4.2	4.23	2.63
0.31	Good	J	SI2	63.3	58	335	4.34	4.35	2.75
0.24	Very Good	J	VVS2	62.8	57	336	3.94	3.96	2.48



# The dataset



$$\text{depth} = z \text{ depth} / z * 100$$
$$\text{table} = \text{table width} / x * 100$$



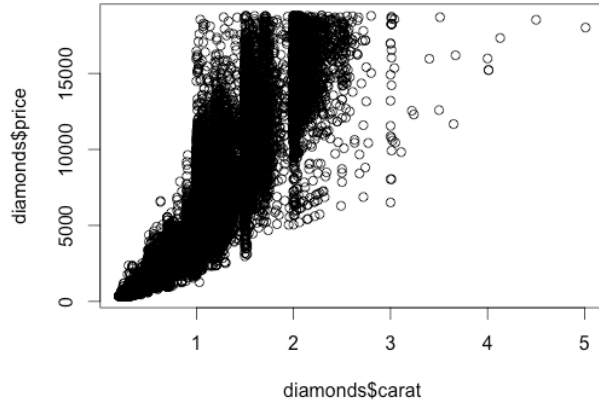
# The dataset

```
library(ggplot2)
head(diamonds)[,1:4]
# A tibble: 6 × 4
  carat    cut  color clarity
  <dbl>  <ord> <ord>  <ord>
1  0.23   Ideal    E     SI2
2  0.21  Premium    E     SI1
3  0.23    Good    E     VS1
4  0.29  Premium    I     VS2
5  0.31    Good    J     SI2
6  0.24 Very Good    J     VVS2
```

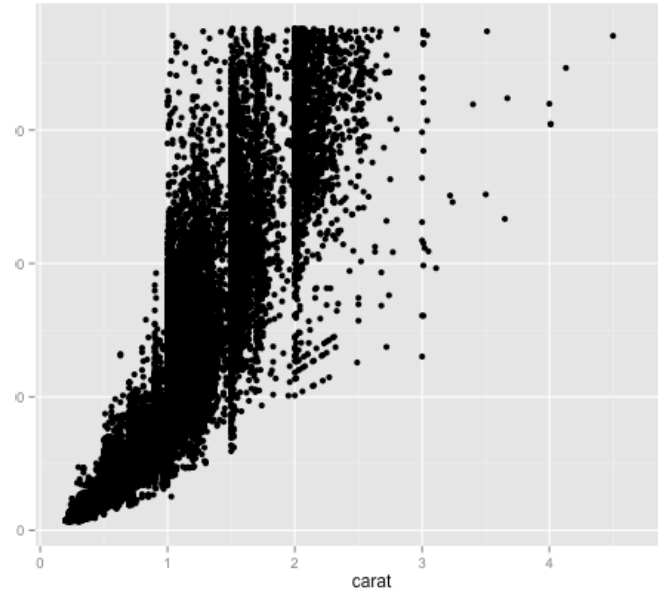


# qplot()

```
plot(diamonds$carat,  
     diamonds$price)
```

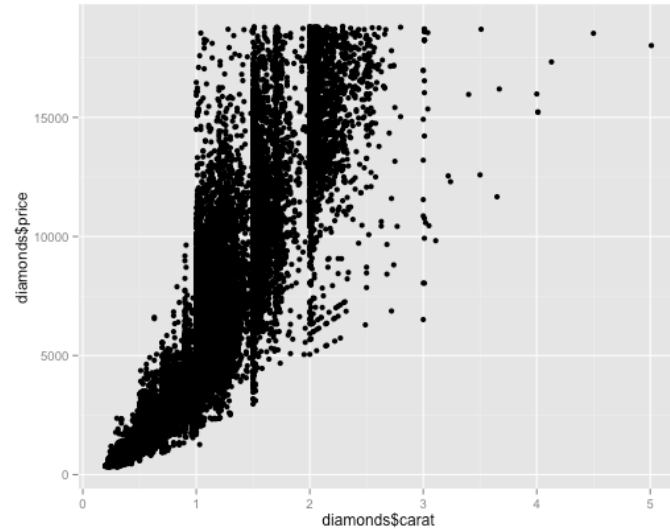


```
qplot(carat, price,  
      data = diamonds)
```



# qplot()

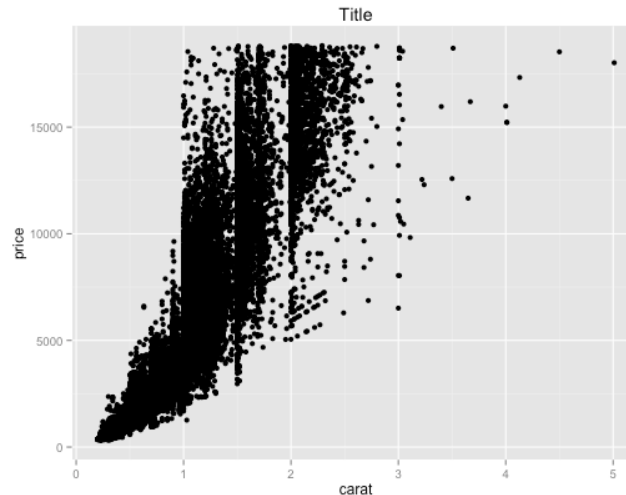
```
qplot(diamonds$carat,  
diamonds$price)
```



# qplot()

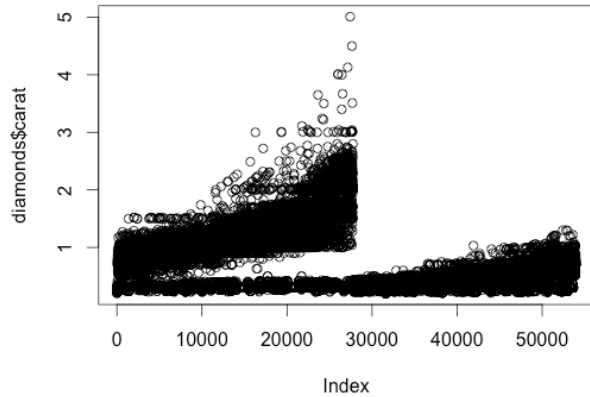
If we add a title we will  
redraw

```
qplot(carat,  
price,  
data=diamonds) +  
labs(title = "Title")
```

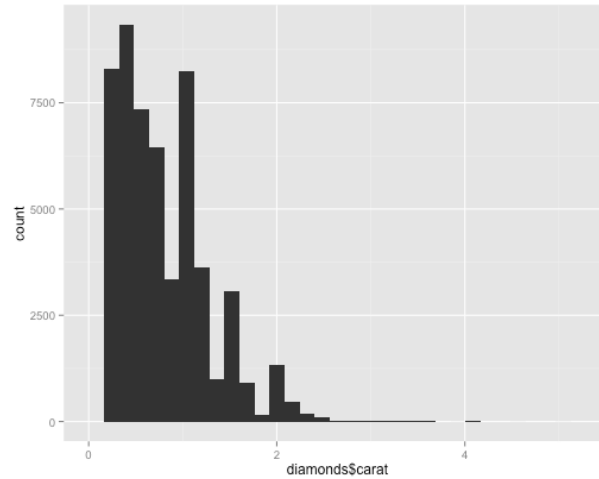


# qplot()

`plot(diamonds$carat)`

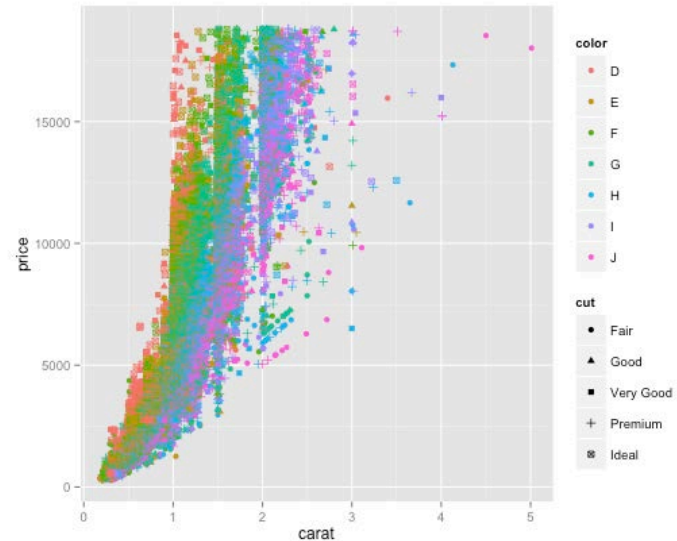


`qplot(diamonds$carat)`



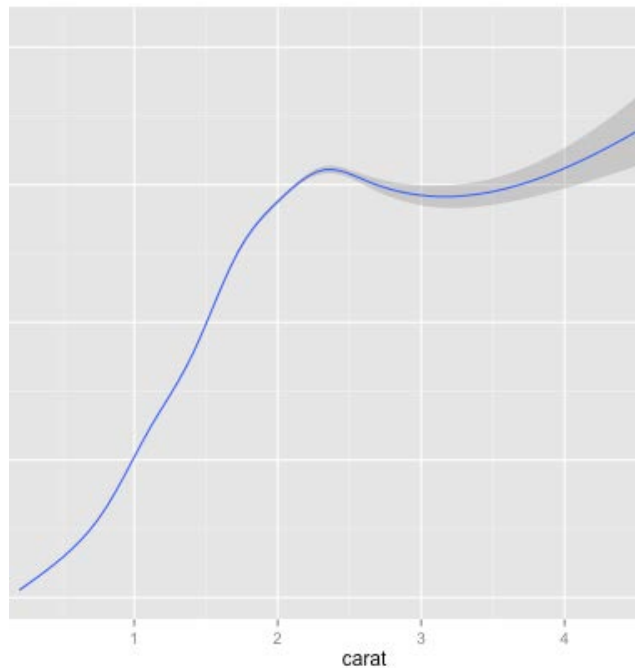
# qplot()

```
qplot(carat, price,  
data = diamonds,  
shape = cut,  
color = color)
```



# qplot()

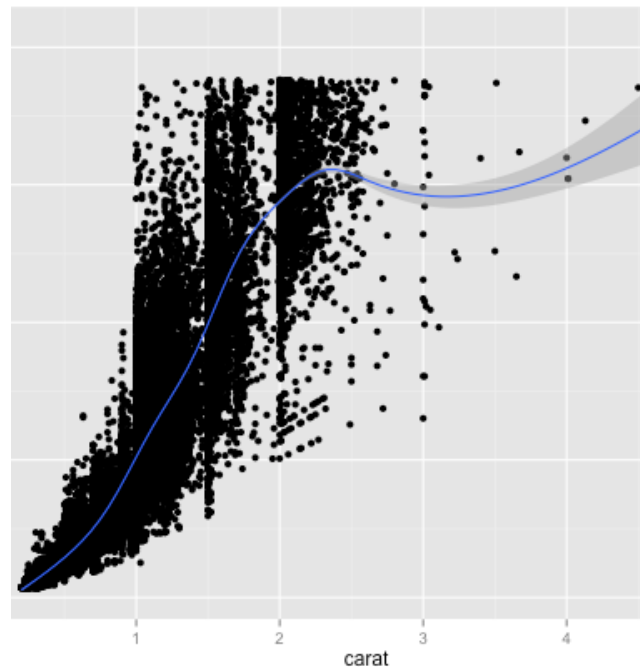
```
qplot(carat, price,  
data = diamonds,  
geom = c("smooth"))
```



# qplot()

```
qplot(carat, price,  
data = diamonds,  
geom =  
c("point", "smooth"))
```

```
qplot(carat, price,  
data = diamonds) +  
geom_smooth()
```



# Basic ggplot2

Any layer will have the following:

- Data
- Aesthetic mappings e.g. diamond cut-> shape
- Geometrical objects. Points, bars, polygons, etc.
- Scales
- Coordinate system
- Faceting system



# The ggplot2 syntax

We'll start by assigning a ggplot to a variable

```
p <- ggplot(data = diamonds, aes(x = carat))
```

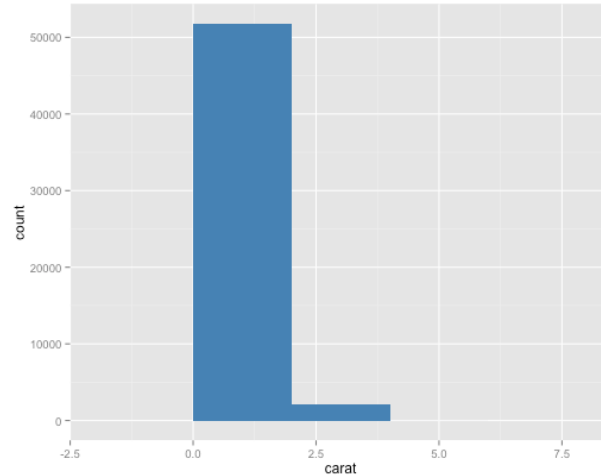
```
p <- ggplot(diamonds, aes(carat))
```



# The ggplot2 syntax

p +

```
geom_histogram(  
  fill = "steelblue",  
  binwidth = 2)
```



# The ggplot2 syntax

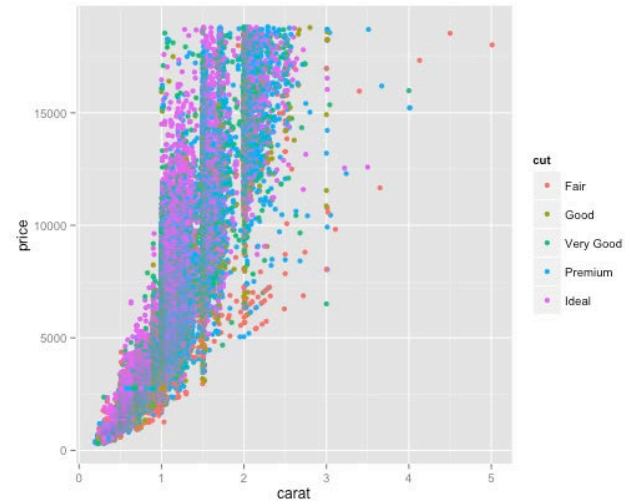
geom	default stat	Aesthetics (required in <b>bold</b> )
blank	identity	no parameters
text	identity	<b>x, y, label</b> , size, color, alpha, hjust, vjust, parse
point	identity	<b>x, y</b> , size, shape, color, fill, alpha, na.rm
bar	bin	<b>x</b> , y, size, linetype, color, fill, alpha



# Scales

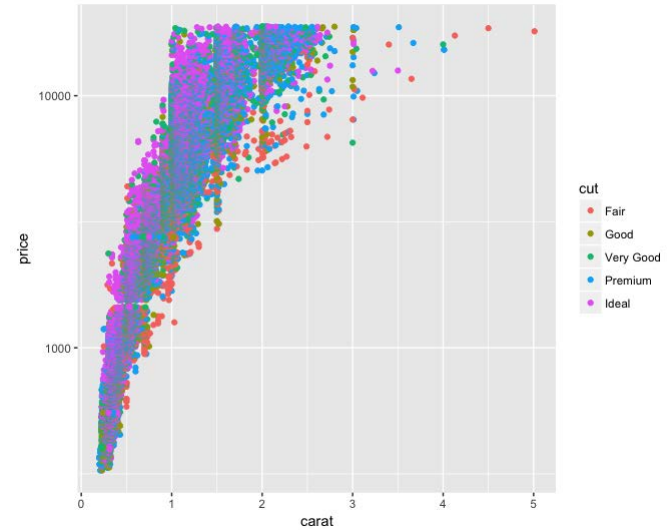
```
p <- ggplot(diamonds, aes(carat, price,  
  color = cut))  
  + geom_point()
```

p



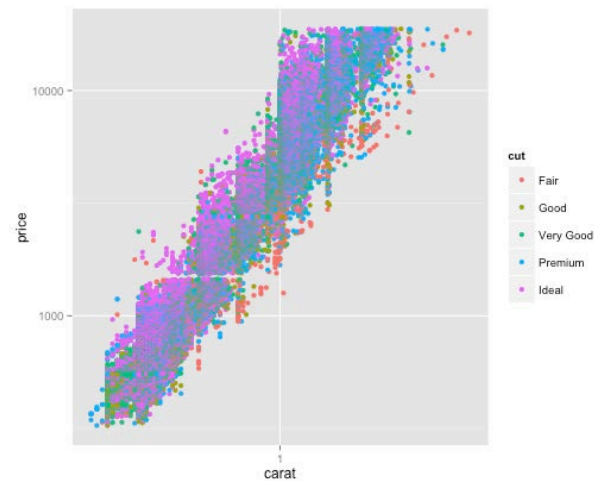
# Scales

```
p + scale_y_log10()
```



# Scales

```
p + scale_x_log10() + scale_y_log10()
```



# Scales

The color space in ggplot is hcl. In this space we have the following

Hue: an angle between 0 and 360 for color

Chroma: the “purity” of the color. At 0 you have grey. The maximum depends on luminance

Luminance: brightness. Black at 0 and white at 1.

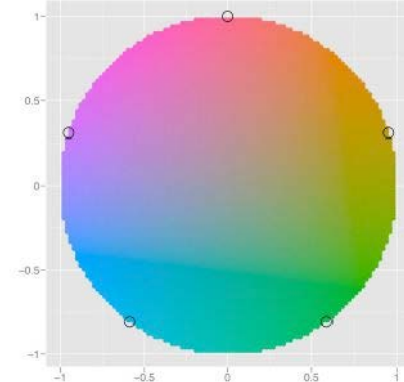
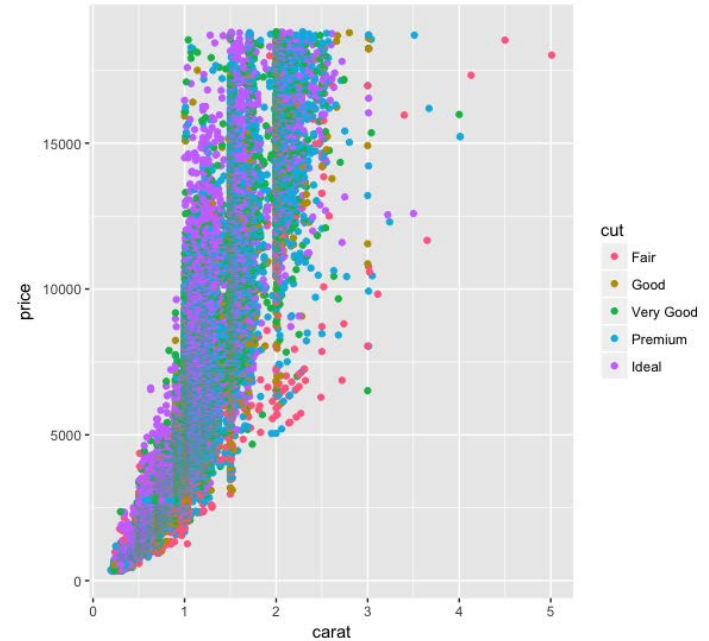


Fig. 3.4: A colour wheel illustrating the choice of five equally spaced colours. This is the default scale for discrete variables.

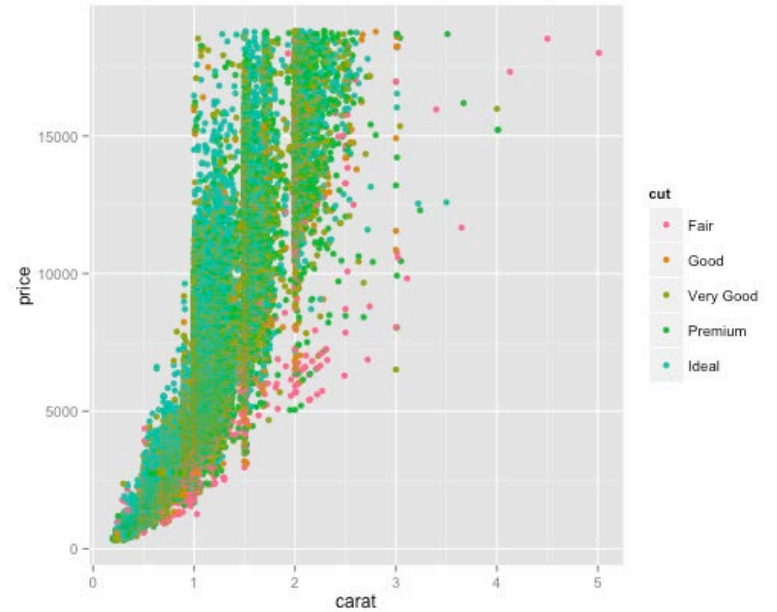
# Scales

```
p + scale_color_discrete(h=c(0,360))
```



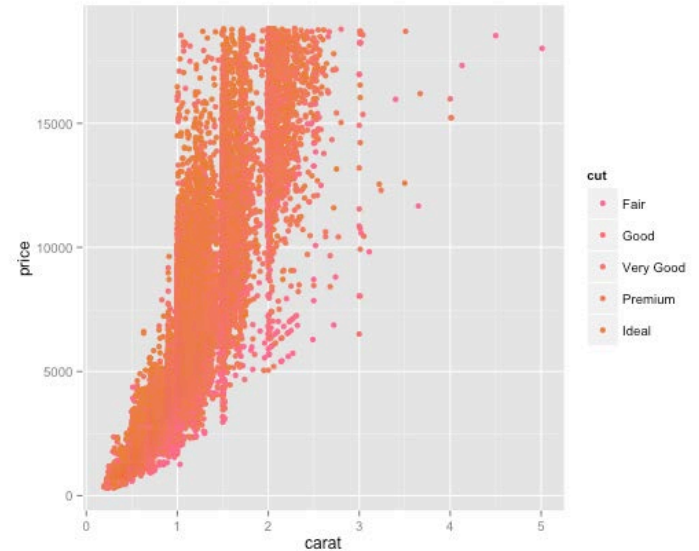
# Scales

```
p + scale_color_discrete(h = c(0,180))
```



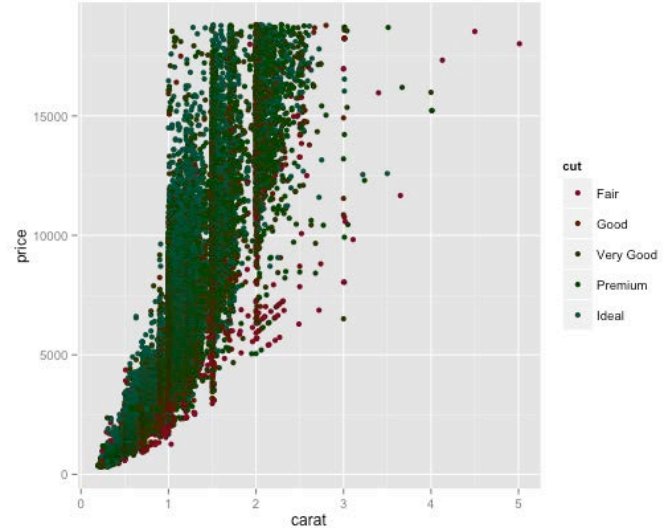
# Scales

```
p + scale_color_discrete(h = c(0, 30))
```



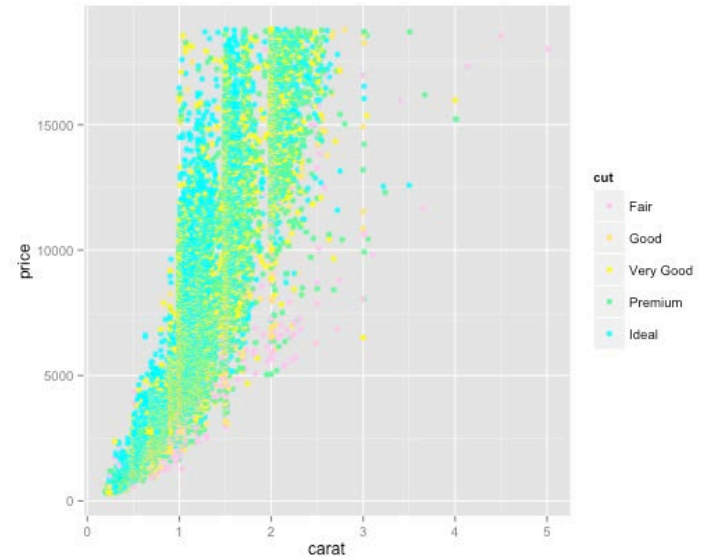
# Scales

```
p + scale_color_discrete(h = c(0, 180) ,  
  c = 100, l = 20)
```



# Scales

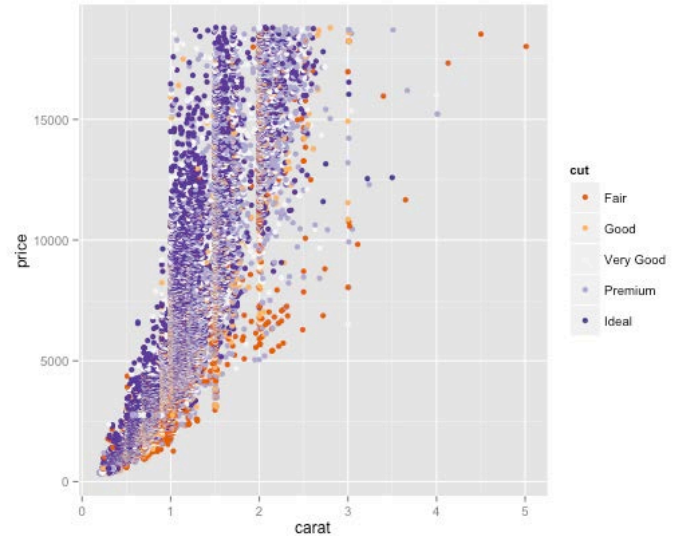
```
p + scale_color_discrete(h = c(0, 180),  
c = 100, l = 100)
```



# Scales

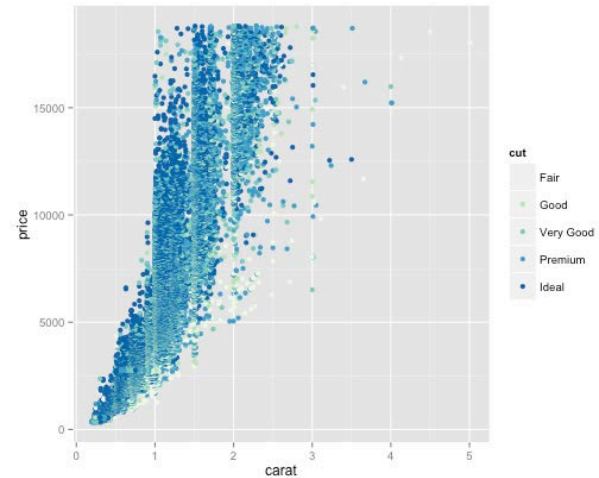
Cynthia Brewer's color palettes are available.

```
p + scale_color_brewer(type = "div",  
palette = 4)
```



# Scales

```
p + scale_color_brewer(type = "seq",  
  palette = 4)
```

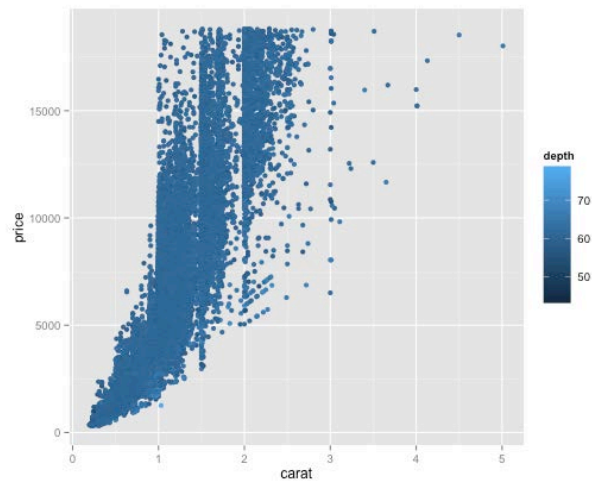


# Scales

What about color for continuous values?

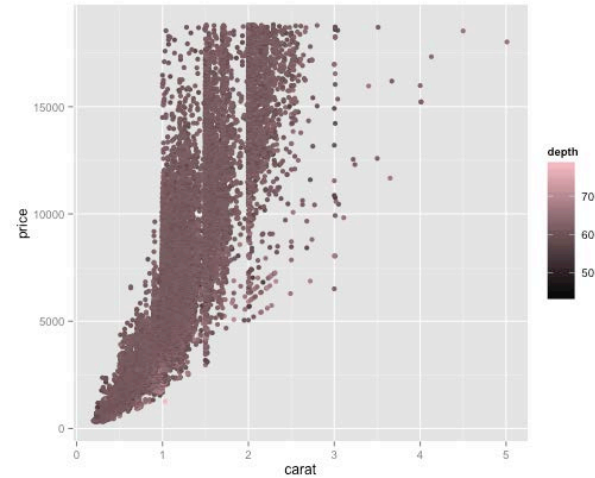
```
q <- ggplot(diamonds, aes(carat, price,  
  color = depth))  
  
+ geom_point()
```

q



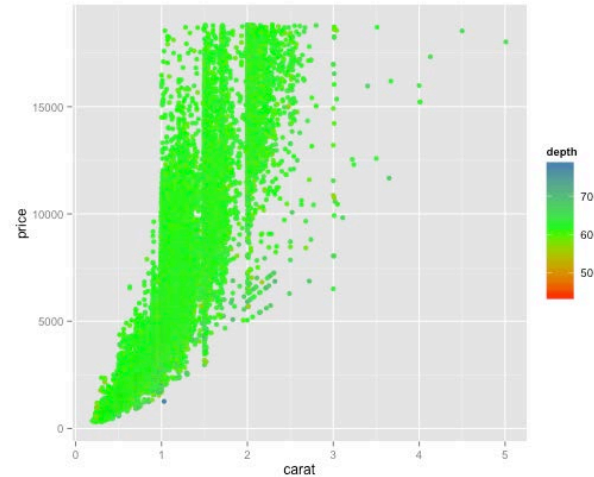
# Scales

```
q + scale_color_continuous(low = "black",  
  high = "pink"))
```



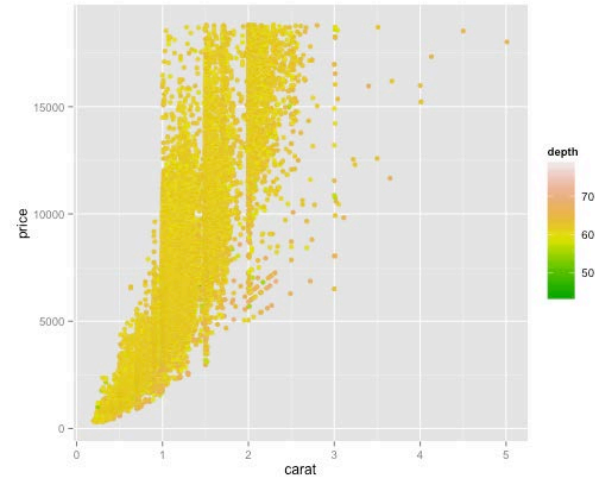
# Scales

```
q + scale_color_gradientn(colors = c("red",  
  "green", "steel blue"))
```



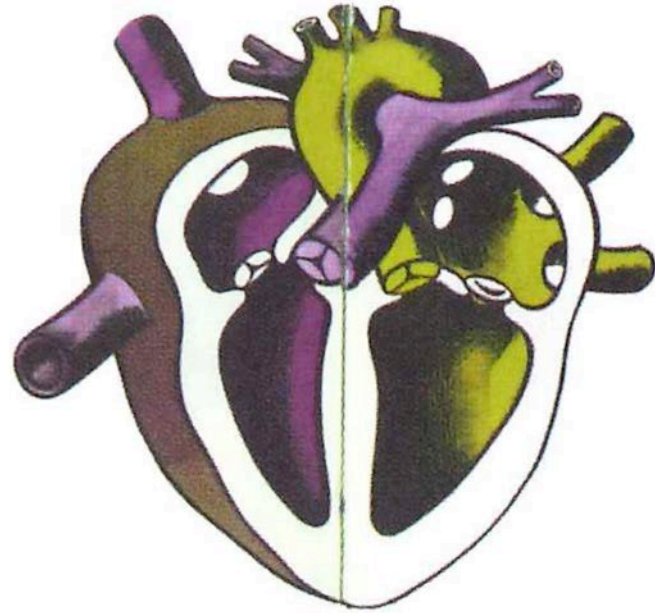
# Scales

```
q + scale_color_gradientn(colors =  
terrain.colors(10))
```



# Scales

Try not to be too cute with the colors.



# Overplotting

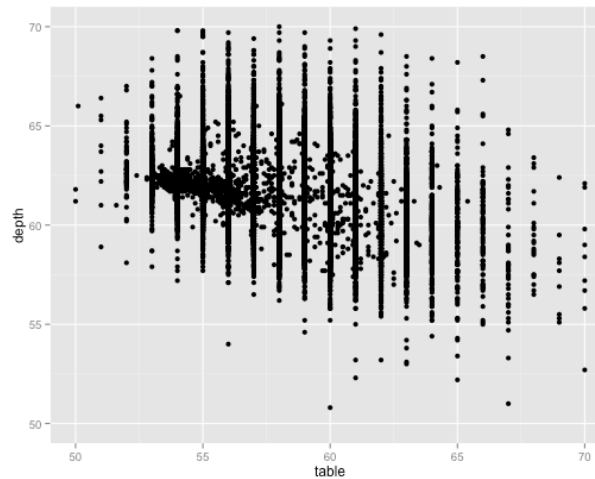
How to deal with overplotting?

```
td <- ggplot(diamonds, aes(table, depth))
```

```
+ xlim(50, 70)
```

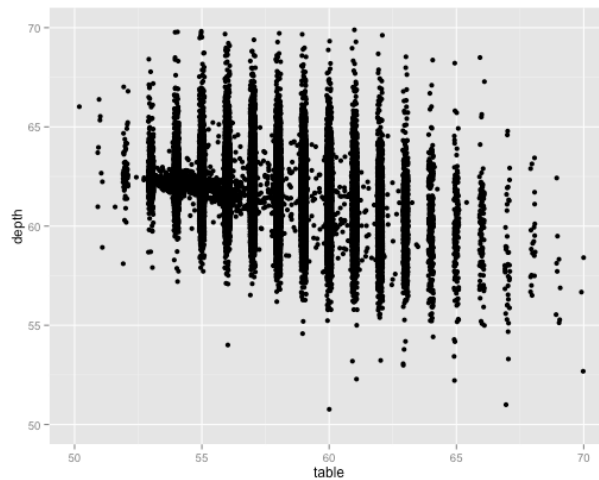
```
+ ylim(50, 70)
```

```
td + geom_point()
```



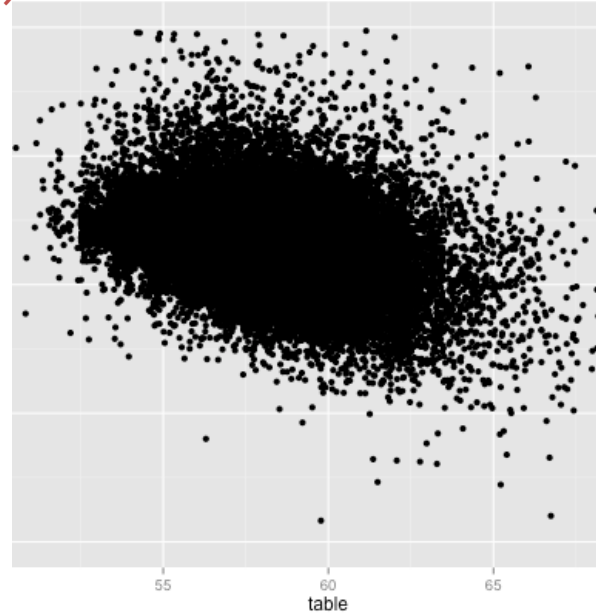
# Overplotting

```
td + geom_jitter(position =  
  position_jitter(width = .1))
```



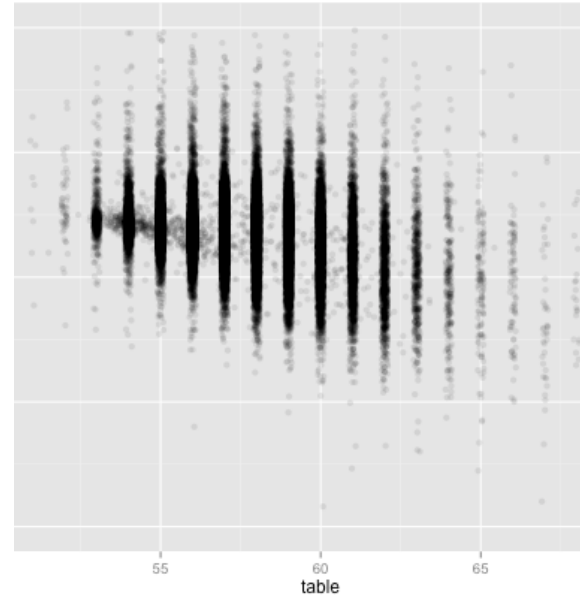
# Overplotting

```
td + geom_jitter(position =  
position_jitter(width = .5));
```



# Overplotting

```
td + geom_jitter(position = position_jitter(  
width = .1),  
alpha = .1 )
```

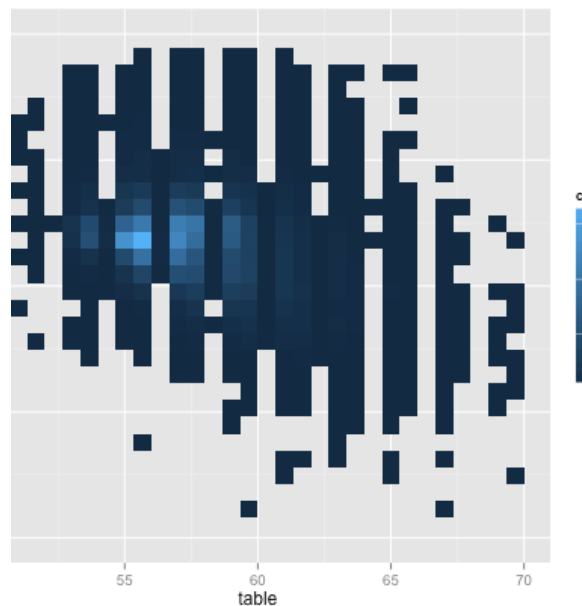


# Overplotting

```
td + stat_bin2d()
```

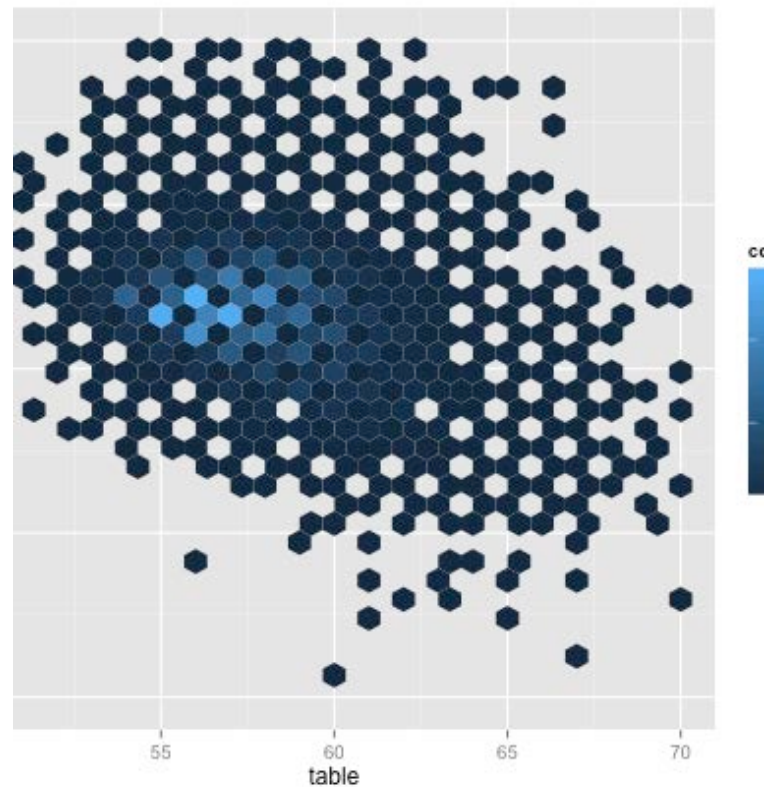
```
#This is the same as
```

```
#td + geom_bin2d()
```



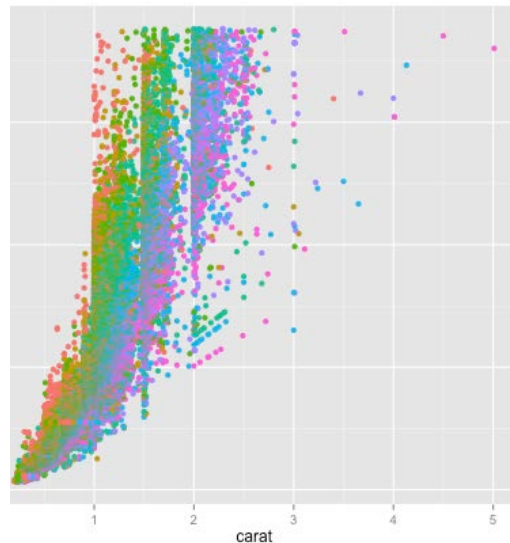
# Overplotting

```
td + stat_binhex()
```



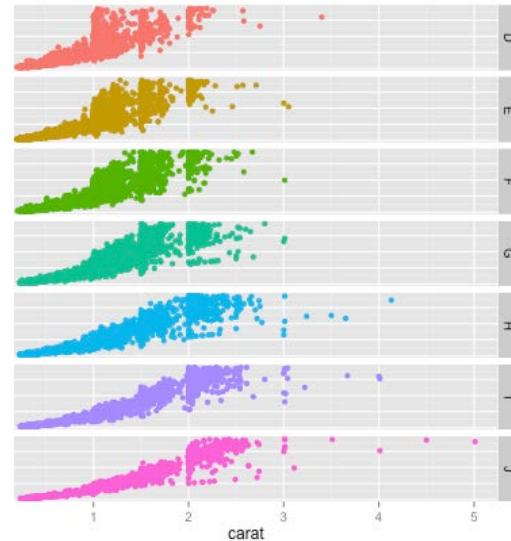
# Overplotting/Faceting

```
qplot(carat, price, data = diamonds,  
       color = color)
```



# Overplotting/Faceting

```
ggplot(carat, price,  
data = diamonds,  
color = color) +  
facet_grid(color~.)
```

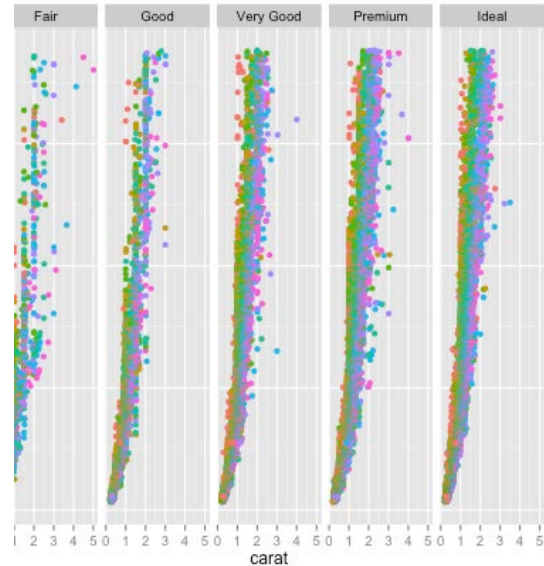


# Overplotting/Faceting

```
qplot(carat, price,
```

```
data = diamonds,
```

```
color = color) +  
facet_grid(. ~ cut)
```

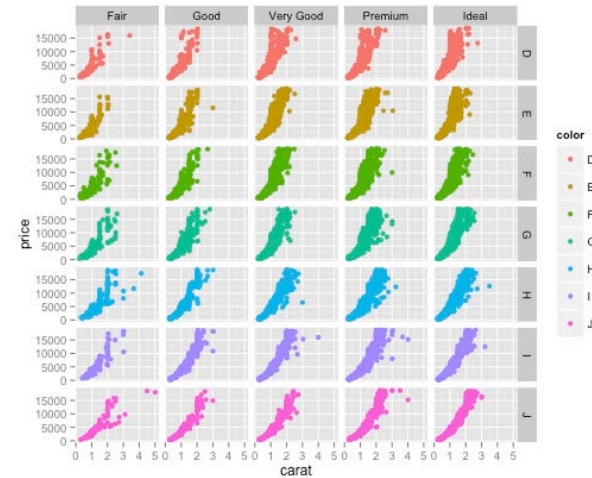


# Overplotting/Faceting

```
qplot(carat, price,
```

```
data = diamonds,
```

```
color = color) + facet_grid(color ~ cut)
```



# Revisiting Minard

While Tufte says Minard's Napoleon graph tracks six dimensions, Wilkinson points out there is a grouping variable so we have seven values with aesthetic mappings.

```
troops <- read.table("troops.txt", header=TRUE)
```

```
cities <- read.table("cities.txt", header=TRUE)
```



# Revisiting Minard

head(troops)

	long	lat	survivors	direction	group
1	24.0	54.9	340000	A	1
2	24.5	55.0	340000	A	1
3	25.5	54.5	340000	A	1
4	26.0	54.7	320000	A	1
5	27.0	54.8	300000	A	1
6	28.0	54.9	280000	A	1



# Revisiting Minard

```
head(cities)
```

```
   long  lat   city
1 24.0 55.0  Kowno
2 25.3 54.7  Wilna
3 26.4 54.4  Smorgoni
4 26.8 54.3  Moiodexno
5 27.7 55.2  Gloubokoe
6 27.6 53.9  Minsk
```



# Revisiting Minard

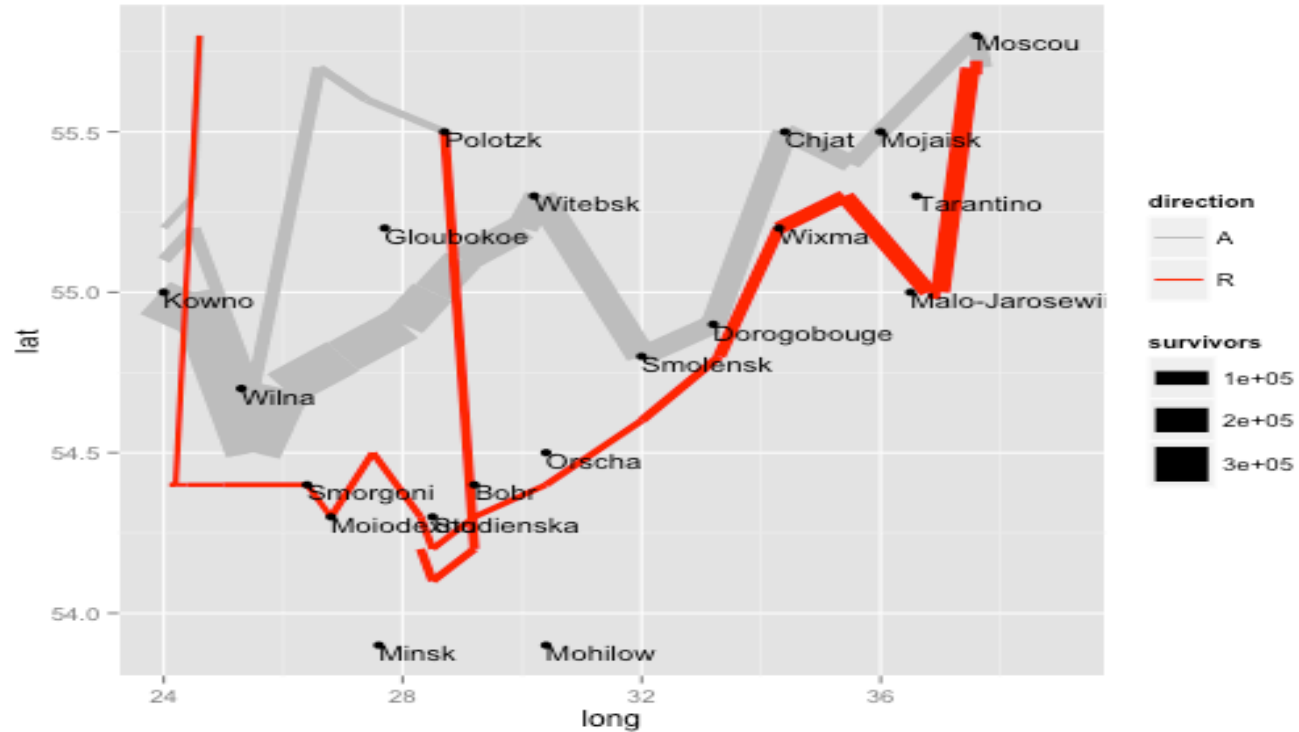
```
#Adapted from code by Hadely Wickham
ggplot(cities, aes(x = long, y = lat)) +

  geom_path(aes(size = survivors, color = direction,
group = group), data = troops) +
  geom_point() +
  geom_text(aes(label = city), hjust=0, vjust=1,
size=4) +

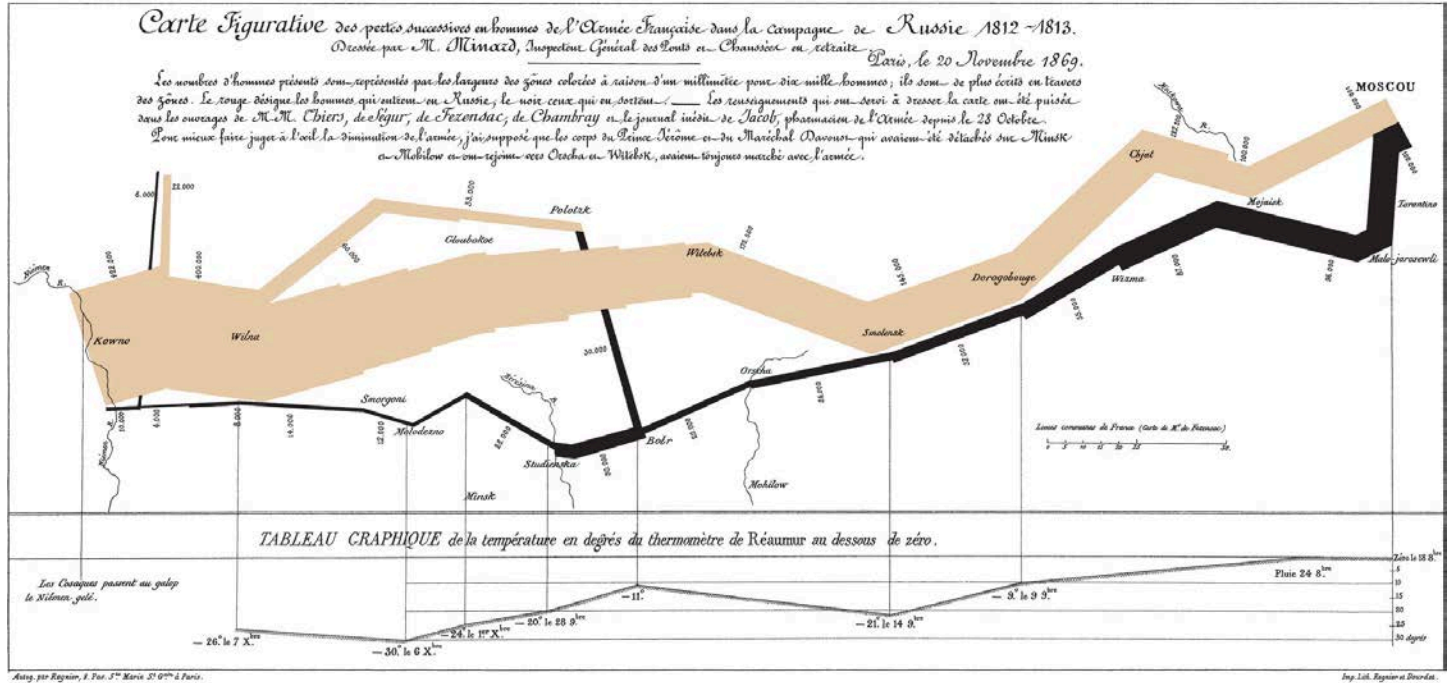
  scale_size(range = c(1, 10)) +
  scale_colour_manual(values = c("grey", "red")) +
  scale_x_continuous(limits = c(24, 39))
```



# Revisiting Minard



# Revisiting Minard



David Reagan, Advanced Visualization Lab

# Web tools for Information Visualization

# Why build something for the web?

- Accessibility
- Let users interact with results
- Let users do their own exploration or analysis

*“Overview first, zoom and filter, then details-on-demand”*

Shneiderman, 1996



# Can't I do that with Shiny?

- If you use R, then you probably can
- If you use Python, try Dash or Bokeh



# When to use Shiny

- Comfortable in the R ecosystem, not interested in learning new tools
- Need the power of the server, not worried about having to run Shiny server
- Don't need anything too fancy



# Prepare data for the web

- Export flat files in JSON or CSV
- Find import library for your data type (SheetJS, ...)
- Query database



# Mr. Data Converter

I will convert your Excel data into one of several web-friendly formats, including HTML, JSON and XML.

Fork me on [github](#).

## SETTINGS

Delimiter:  Auto •  Comma •  Tab

Decimal Sign:  Dot •  Comma

First row is the header

Transform:  downcase •  upcase •  none

Include white space in output

Indent with:  tabs •  spaces

Input CSV or tab-delimited data. Using Excel? Simply copy and paste. No data on hand? [Use sample](#)

NAME	VALUE	COLOR	DATE
Alan	12	blue	Sep. 25, 2009
Shan	13	"green blue"	Sep. 27, 2009
John	45	orange	Sep. 29, 2009
Minna	27	teal	Sep. 30, 2009

Output as **JSON - Properties**

```
[{"NAME": "Alan", "VALUE": 12, "COLOR": "blue", "DATE": "Sep. 25, 2009"}, {"NAME": "Shan", "VALUE": 13, "COLOR": "green\tblue", "DATE": "Sep. 27, 2009"}, {"NAME": "John", "VALUE": 45, "COLOR": "orange", "DATE": "Sep. 29, 2009"}, {"NAME": "Minna", "VALUE": 27, "COLOR": "teal", "DATE": "Sep. 30, 2009"}]
```



## D3: Data-Driven Documents

- JavaScript library for manipulating web documents using data
- Uses open standards: HTML, CSS, SVG
- Typically renders SVG, but can render to HTML canvas for performance



# An interactive example

- Bind data to standard web elements
- Update data to update existing elements, add new ones, delete old ones
- Maintain object constancy with smooth transitions
- <https://bl.ocks.org/mbostock/raw/3808234/>



# Example: Define data

```
var alphabet = "abcdefghijklmnopqrstuvwxyz".split("");
```



# Example: Update data on timer

```
// Grab a random sample of letters from the alphabet, in alphabetical order.
d3.interval(function() {
  update(d3.shuffle(alphabet)
    .slice(0, Math.floor(Math.random() * 26))
    .sort());
}, 1500);
```



# Example: Create transition

```
var t = d3.transition()  
    .duration(750);
```



## Example: Update selection with new data

```
// JOIN new data with old elements.  
var text = g.selectAll("text")  
  .data(data, function(d) { return d; });
```



# Example: Remove old elements

```
// EXIT old elements not present in new data.  
text.exit()  
  .attr("class", "exit")  
.transition(t)  
  .attr("y", 60)  
  .style("fill-opacity", 1e-6)  
  .remove();
```



## Example: Update old elements

```
// UPDATE old elements present in new data.
text.attr("class", "update")
    .attr("y", 0)
    .style("fill-opacity", 1)
    .transition(t)
    .attr("x", function(d, i) { return i * 32; });
```



# Example: Create new elements

```
// ENTER new elements present in new data.
text.enter().append("text")
  .attr("class", "enter")
  .attr("dy", ".35em")
  .attr("y", -60)
  .attr("x", function(d, i) { return i * 32; })
  .style("fill-opacity", 1e-6)
  .text(function(d) { return d; })
.transition(t)
  .attr("y", 0)
  .style("fill-opacity", 1);
```



# A more InfoVis-y version of the same idea

1. <https://bost.ocks.org/mike/nations/>
2. <https://romsson.github.io/dragit/example/nations.html>



# But how do I make graphs?

- Bar: `append('rect')`
- Line: `append('path')`
- Scatter: `append('circle')`
- **Everything by hand!**



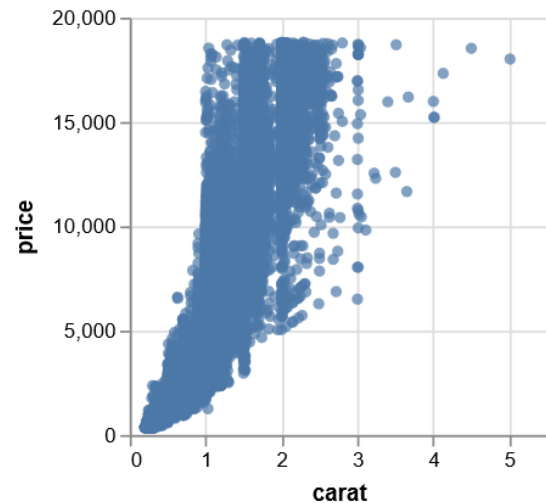
# Vega and Vega-Lite

- “Vega is a visualization grammar”
- “Vega-Lite is a high-level grammar of interactive graphics”
- Both use a JSON specifications to map data to properties of graphical marks
- Vega-Lite can be compiled to Vega
- Both use D3 underneath



# Vega-Lite: Scatter

```
1 {  
2   "$schema": "https://vega.github.io/schema/vega-lite/v2.json",  
3   "data": {"url":  
4     "https://vincentarelbundock.github.io/Rdatasets/csv/ggplot2/diamonds.csv"},  
5   "mark": "circle",  
6   "encoding": {  
7     "x": {"field": "carat", "type": "quantitative"},  
8     "y": {"field": "price", "type": "quantitative"}  
9   }  
10 }
```



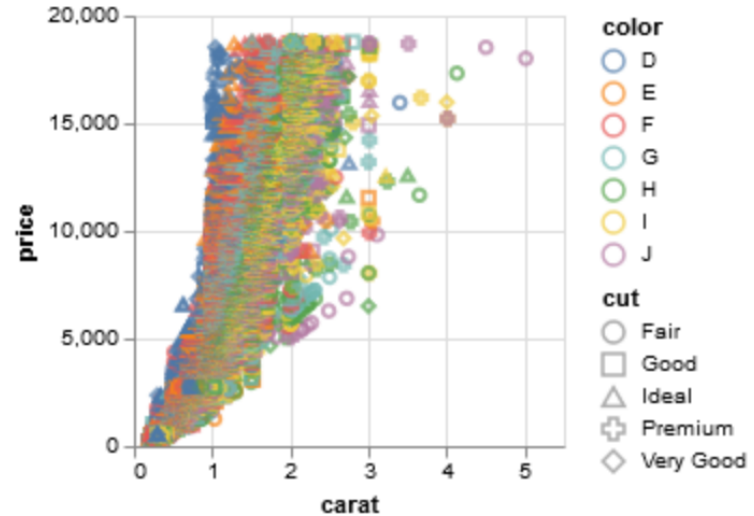
# Vega-Lite: Scatter

```

{
  "$schema":
  "https://vega.github.io/schema/vega-lite/v2.json",
  "description": "A scatterplot showing horsepower
  and miles per gallons.",
  "data": {"url":
  "https://vincentarelbundock.github.io/Rdatasets/csv/
  ggplot2/diamonds.csv"},
  "mark": "point",
  "encoding": {
    "x": {"field": "carat", "type": "quantitative"},
    "y": {"field": "price", "type": "quantitative"},
    "color": {"field": "color", "type": "nominal"},
    "shape": {"field": "cut", "type": "nominal"}
  }
}

```

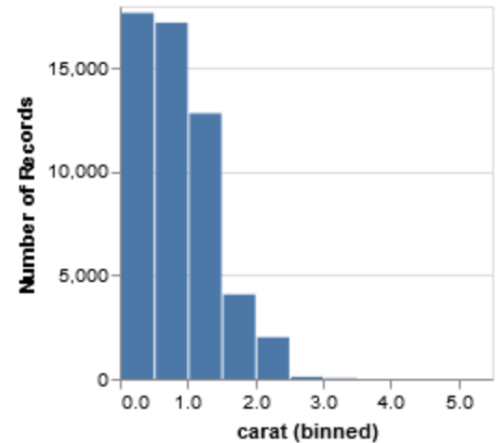
Format



# Vega-Lite: Histogram

```
{
  "$schema":
  "https://vega.github.io/schema/vega-lite/v2.json",
  "data": {"url":
  "https://vincentarelbundock.github.io/Rdatasets/csv/ggplot2/diamonds.csv"},
  "mark": "bar",
  "encoding": {
    "x": {
      "bin": true,
      "field": "carat",
      "type": "quantitative"
    },
    "y": {
      "aggregate": "count",
      "type": "quantitative"
    }
  }
}
```

Format



Vega-Lite version 2.3.0

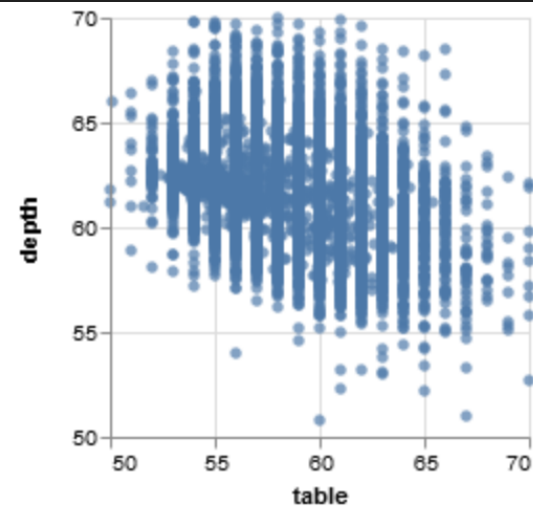
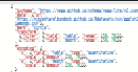
Parse: auto

No Tooltips



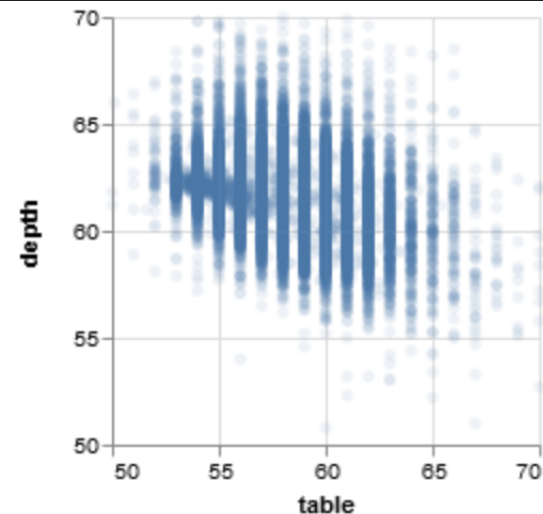
# Vega-Lite: Scatter with filter

```
{
  "$schema": "https://vega.github.io/schema/vega-lite/v2.json",
  "data": {"url":
    "https://vincentarelbundock.github.io/Rdatasets/csv/ggplot2/diamonds.csv"},
  "mark": "circle",
  "transform": [
    {"filter": {"field": "table", "range": [50, 70]}},
    {"filter": {"field": "depth", "range": [50, 70]}}
  ],
  "encoding": {
    "x": {"field": "table", "type": "quantitative",
      "scale": {"domain": [50, 70]}},
    "y": {"field": "depth", "type": "quantitative",
      "scale": {"domain": [50, 70]}}
  }
}
```



# Vega-Lite: Scatter with lower opacity

```
{
  "$schema": "https://vega.github.io/schema/vega-lite/v2.json",
  "data": {"url":
    "https://vincentarelbundock.github.io/Rdatasets/csv/ggplot2/diamonds.csv"},
  "mark": {"type": "circle", "opacity": 0.1},
  "transform": [
    {"filter": {"field": "table", "range": [50, 70]}},
    {"filter": {"field": "depth", "range": [50, 70]}}
  ],
  "encoding": {
    "x": {"field": "table", "type": "quantitative",
      "scale": {"domain": [50, 70]}},
    "y": {"field": "depth", "type": "quantitative",
      "scale": {"domain": [50, 70]}}
  }
}
```



# Vega-Lite: Scatter with bins

```

"mark": {"type": "rect"},
"transform": [
  {"filter": {"field": "table", "range": [50, 70]}},
  {"filter": {"field": "depth", "range": [50, 70]}}
],
"encoding": {
  "x": {"field": "table", "type": "quantitative",
    "scale": {"domain": [50, 70]}, "bin":
    {"maxbins": 40}},
  "y": {"field": "depth", "type": "quantitative",
    "scale": {"domain": [50, 70]}, "bin":
    {"maxbins": 40}},
  "color": {"aggregate": "count",
    "type": "quantitative"}
}

```

