Graphics in R

Programming in R for Data Science

Anders Stockmarr, Kasper Kristensen, Anders Nielsen

Overview

base The original/default graphics system in **R** .

- Example:

  ```
  > demo(graphics)
  ```

- Highly customizable.
- Complex plots require many code lines.

lattice Shorter syntax for complex (e.g. multipanel) plots. Less custumizable than **base**.

- Example:

  ```
  > library(lattice)
  > demo(lattice)
  ```

ggplot2 By Hadley Wickham; builds on the same ideas as **lattice**.

- gg = "grammar of graphics"
- Example:

  ```
  > library(ggplot2)
  > example(qplot)
  ```

This presentation is about **ggplot2**.

ggplot2

- Basic plotting function: `ggplot()`. Used for advanced plots.
- Wrapper that resembles `plot()` from the basic graphics system: `qplot()`. Used for 'quick' plots. Syntax resembles that of `plot()`.
- Grammar of graphics:
  - All plots are objects. You build them incrementally. Use the operator + to add to an existing plot.
  - Layer: Aestetics (**aes**): Defines how the data are mapped.
  - Layer: Geometric objects (**geom**): Points, lines, polygens, etc.
  - Layer: Coordinate system objects (**coord**).

Example: Diamond data

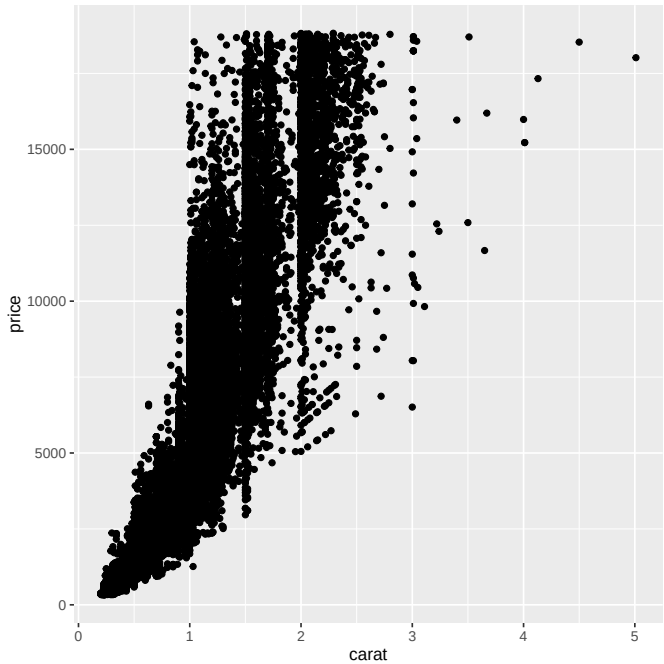
- Load the ggplot2 package and take a look at the diamond data:

```
> library(ggplot2)
> head(diamonds)
```

```
  carat       cut color clarity depth table price    x    y    z
1  0.23     Ideal     E     SI2  61.5    55   326 3.95 3.98 2.43
2  0.21   Premium     E     SI1  59.8    61   326 3.89 3.84 2.31
3  0.23      Good     E     VS1  56.9    65   327 4.05 4.07 2.31
4  0.29   Premium     I     VS2  62.4    58   334 4.20 4.23 2.63
5  0.31      Good     J     SI2  63.3    58   335 4.34 4.35 2.75
6  0.24 Very Good     J    VVS2  62.8    57   336 3.94 3.96 2.48
```

- Quick plot

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



Modifying the quick plot

- With `qplot()`, it is easy to work with:

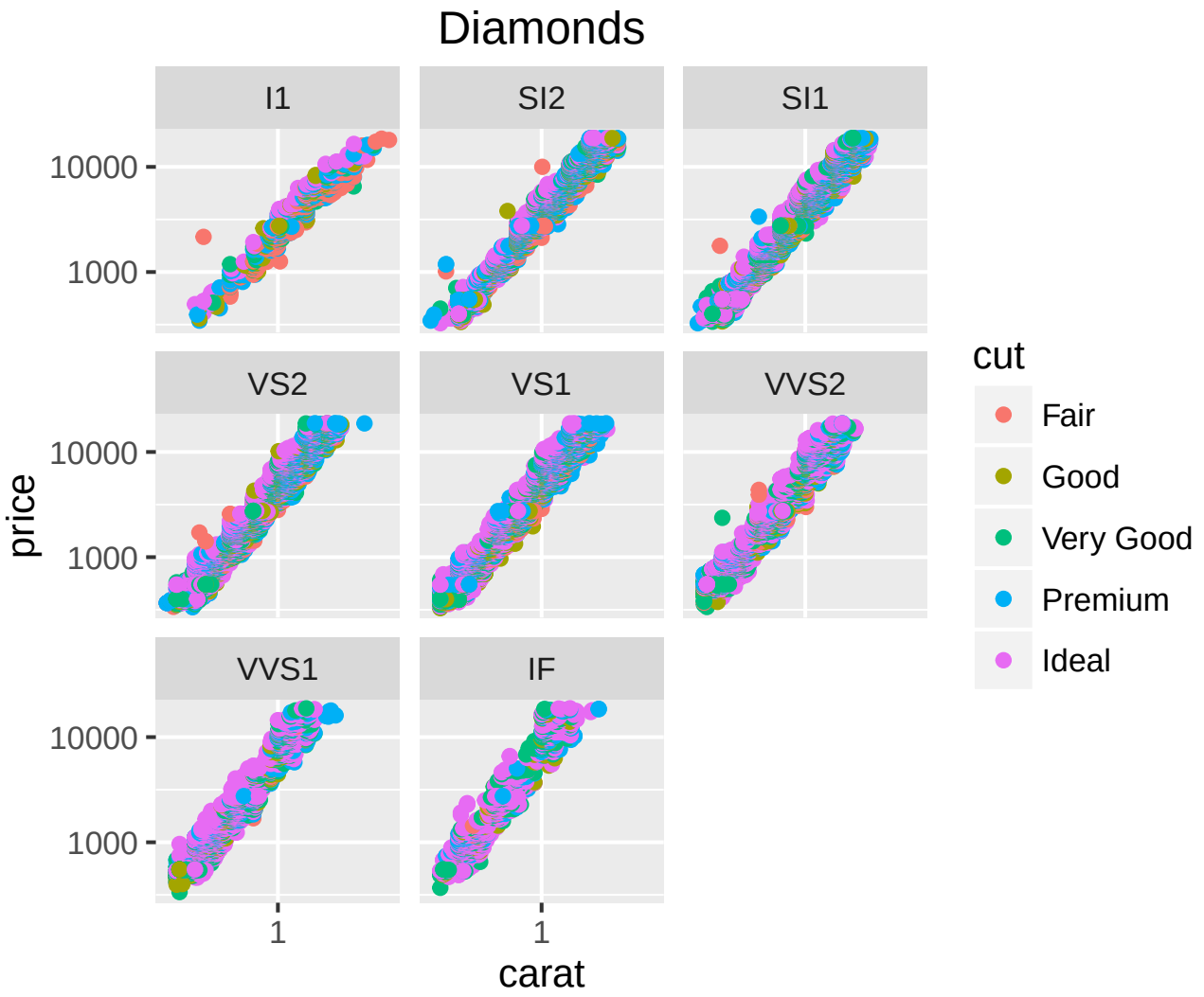
color	Color each point according to a variable in the dataset, and add a corresponding legend.
log	Log-transform one or both axes.
facets	Split in a multi-panel plot according to a group variable.
main	Add a title

- Let's try modifying the previous plot by adding
  - $color = cut$
  - $log = "xy"$
  - $facets = \sim clarity$
  - $main = "Diamonds"$

  one-by-one

Modified plot

```
> qplot(carat,
+       price,
+       data=diamonds,
+       color = cut,
+       log="xy",
+       facets=~clarity,
+       main="Diamonds")
```



Incremental plot construction

- **qplot** is good for a start. However, in order to take full advantage of ggplot2, we must know what the plot is built of and how to modify the parts.
- The quick plot **qplot(carat, price, data=diamonds)** can be built incrementally by
  - Define an empty plot object:

    ```
    > p <- ggplot(data = diamonds)
    ```
  - Update the plot with the variables to plot:

    ```
    > p <- p + aes(x = carat, y = price)
    ```
  - Add a 'geom' object to the plot defining the type of the plot:

    ```
    > p <- p + geom_point()
    ```
  - Display the object thereby triggering the actual plot:

    ```
    > p
    ```
- We can use the '+' operator to modify the plot 'p'. Lets see some examples in the following:

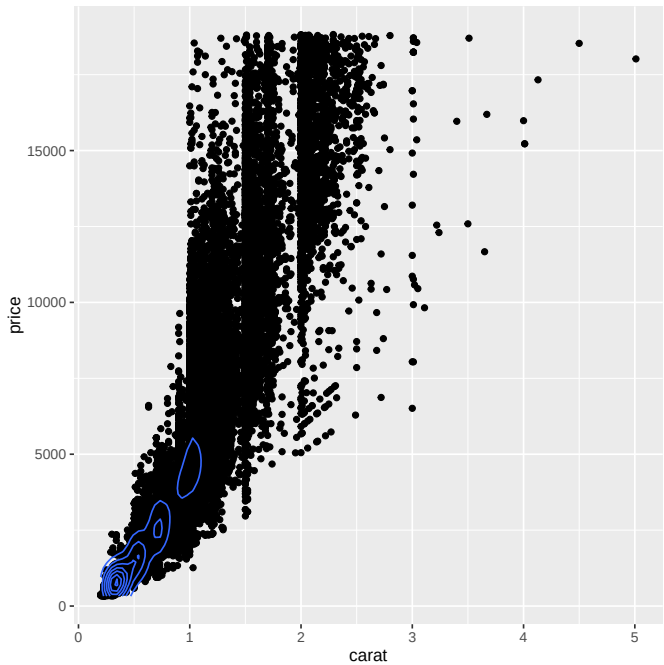
Change the plot type (geom)

- Get an overview of possible **geom**s at `http://docs.ggplot2.org`.
- You can also look at the examples in the documentation:

  ```
  > example(geom_boxplot)
  > example(geom_polygon)
  > example(geom_raster)
  ```

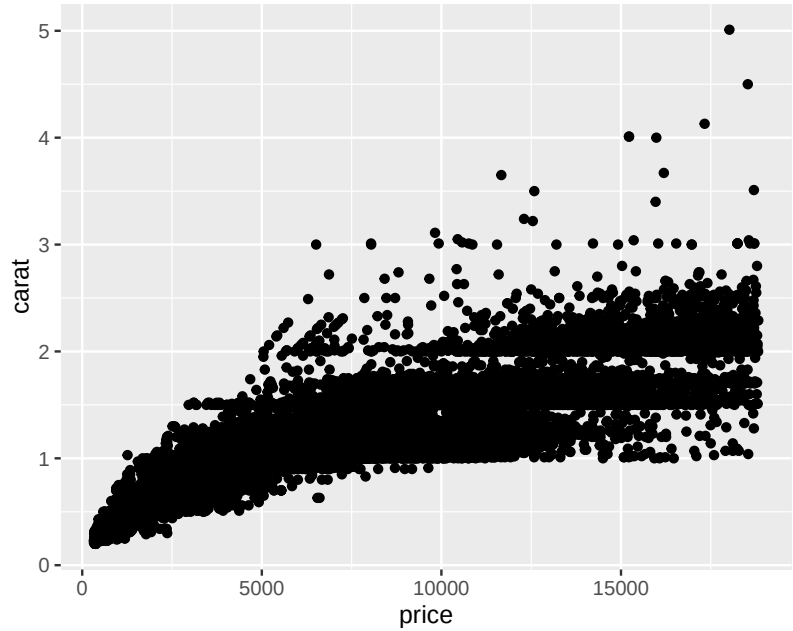
- Example: add a 2D density on top:

  ```
  > p + geom_density2d()
  ```

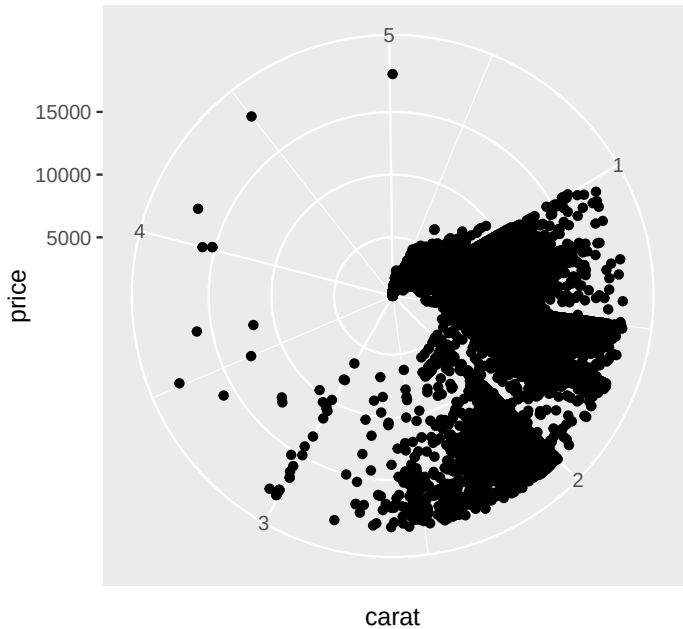


Change the coordinate transformations

```
> p + coord_flip()
```



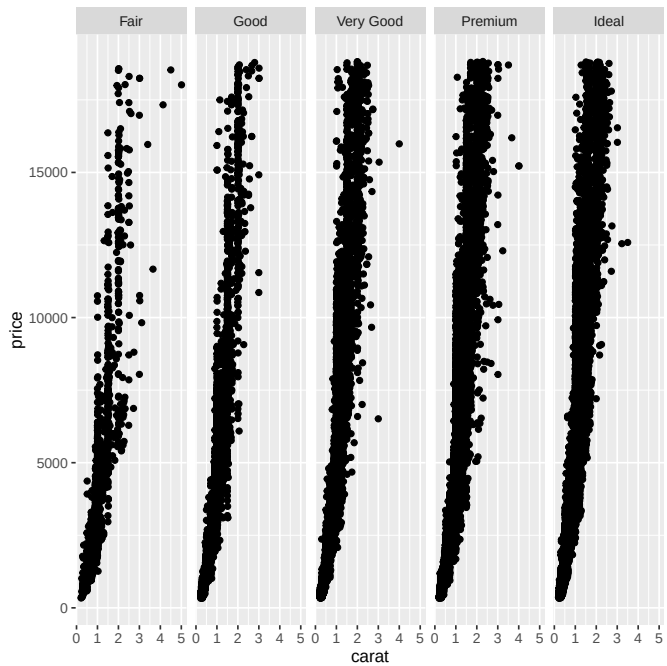

```
> p + coord_polar()
```



Change to multiplanel display

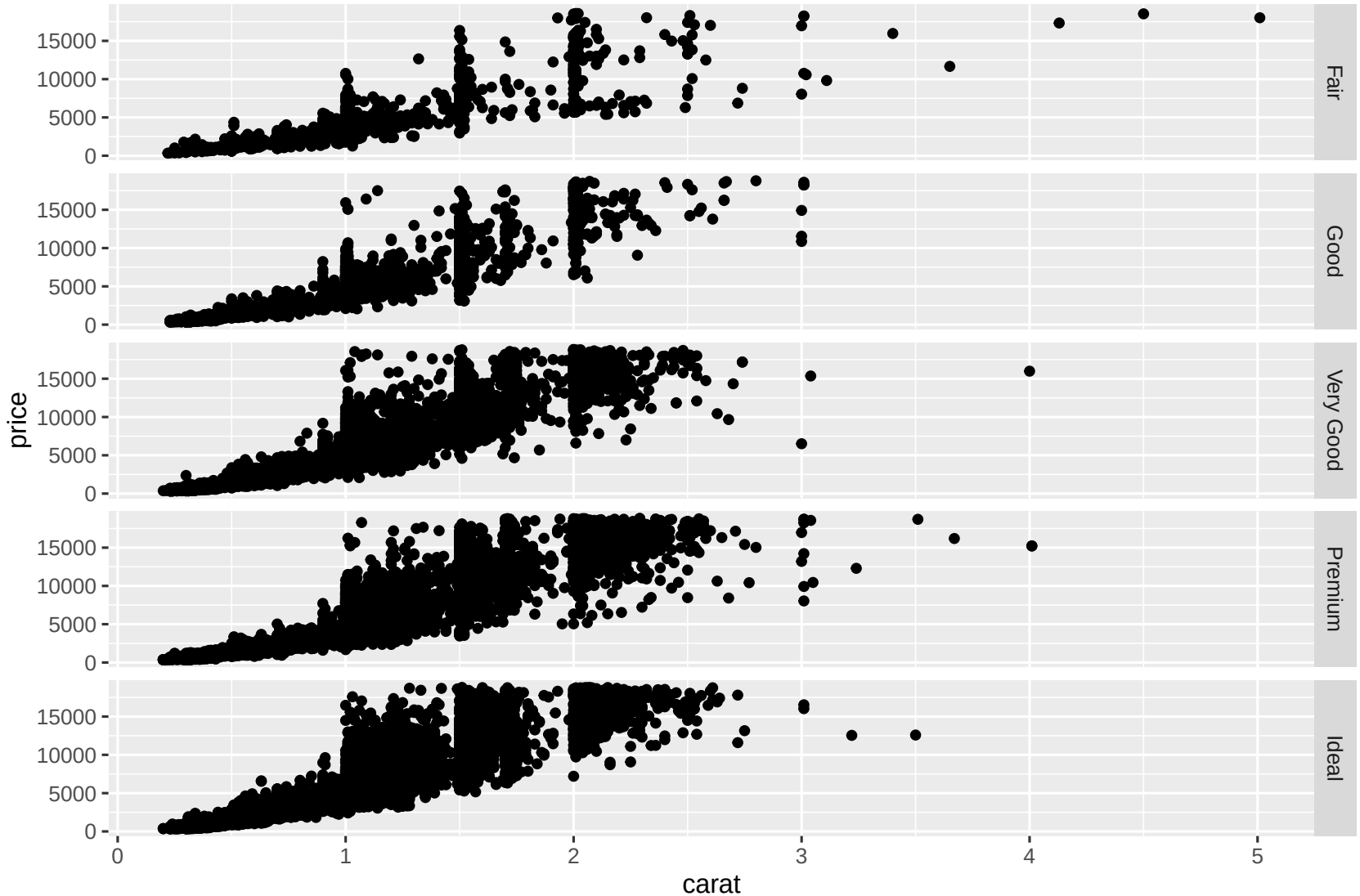
- Add a **facet_grid** to split the plot in multiple panels.
- A facet_grid takes a formula as input.
- Example:

```
> p + facet_grid(. ~ cut)
```



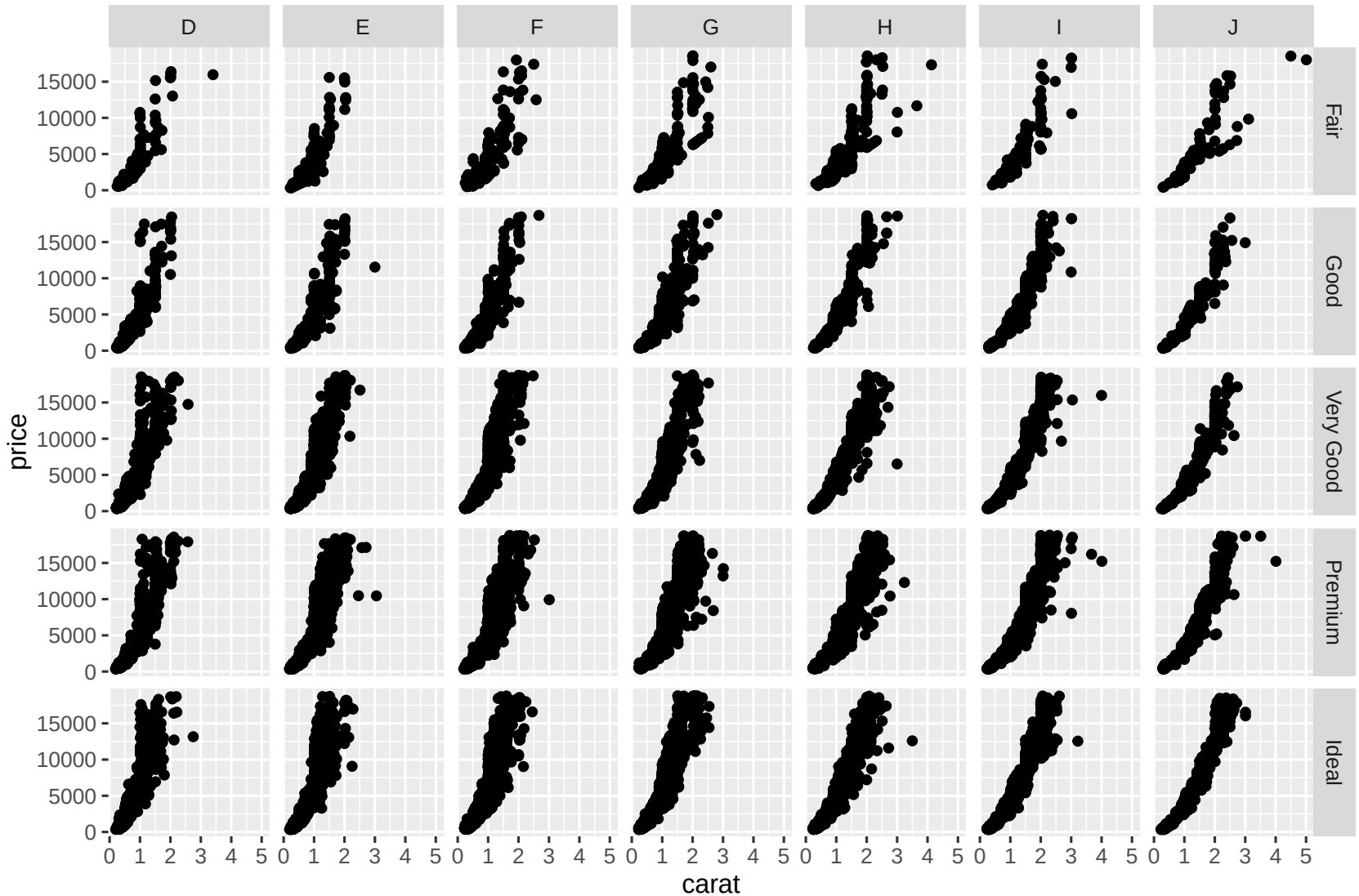
Change to multiplanel display - other facets

```
> p + facet_grid(cut ~ .)
```



Change to multiplanel display - other facets

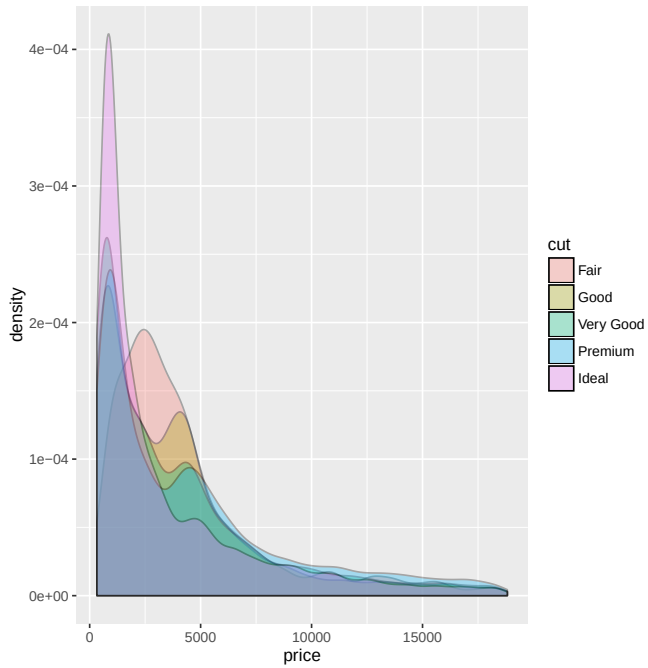
```
> p + facet_grid(cut ~ color)
```



Density plot and alpha blending

- Attributes (colour, shape, fill, linetype etc) have automatically become grouping variables.
- Note the specification of transparancy through the alpha argument: *alpha blending*.

```
> ggplot(diamonds) +
+     aes(price, fill=cut) +
+     geom_density(alpha=.3)
```



Application: Maps

- the ggmap package: Interfacing ggplot2 and RGoogleMaps.

- Two steps in making a map with ggmap:
  1. download raster dta for the map;
  2. create the map with ggmap(), and overlay it with layers of geoms etc.

- Downloading raster data: Specify: a) location of center; b) the zoom factor.
- : Location specification for map downloads: Two ways.
  1. location/address:

     ```
     > myLocation<-"University of Washington"
     ```
  2. lat/long:

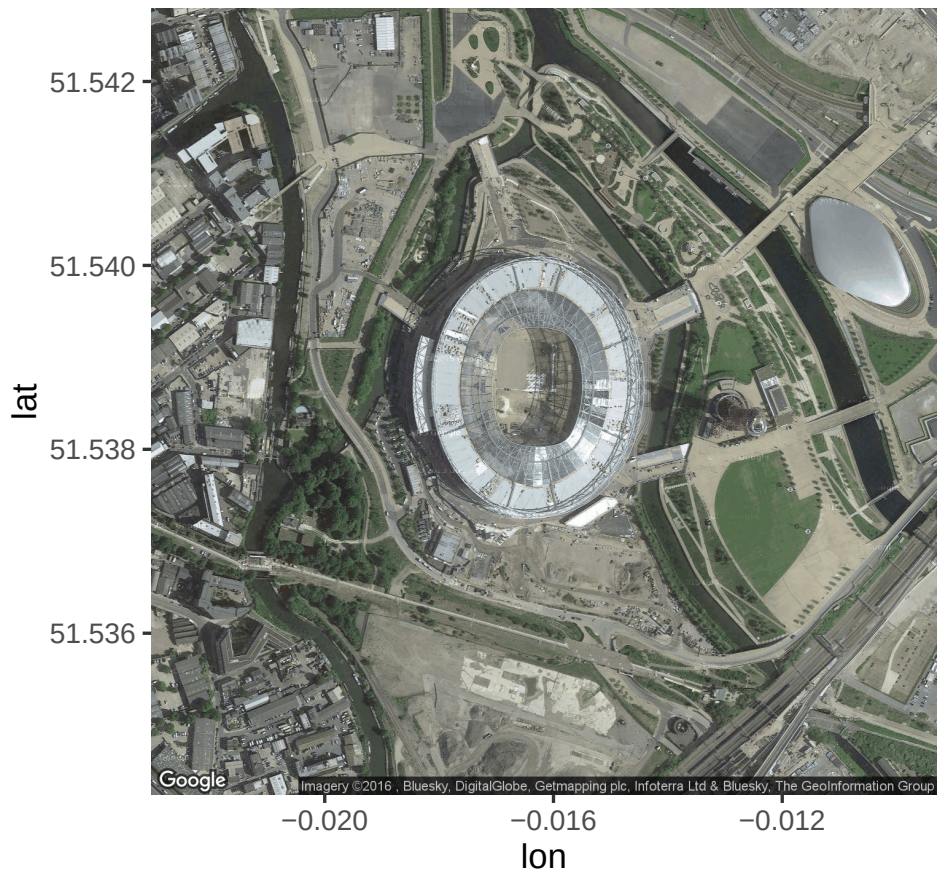
     ```
     > myLocation<-c(lon= -106.4407, lat = 31.76788)
     ```
- both methods will give the same results when specifying the center for the map to download.

- zoom factor: 3 = continent, 10=city, 21=building.

A first map: The London Olympic Stadium

- Download data with the `get_map()` function, plot with `ggmap()`:

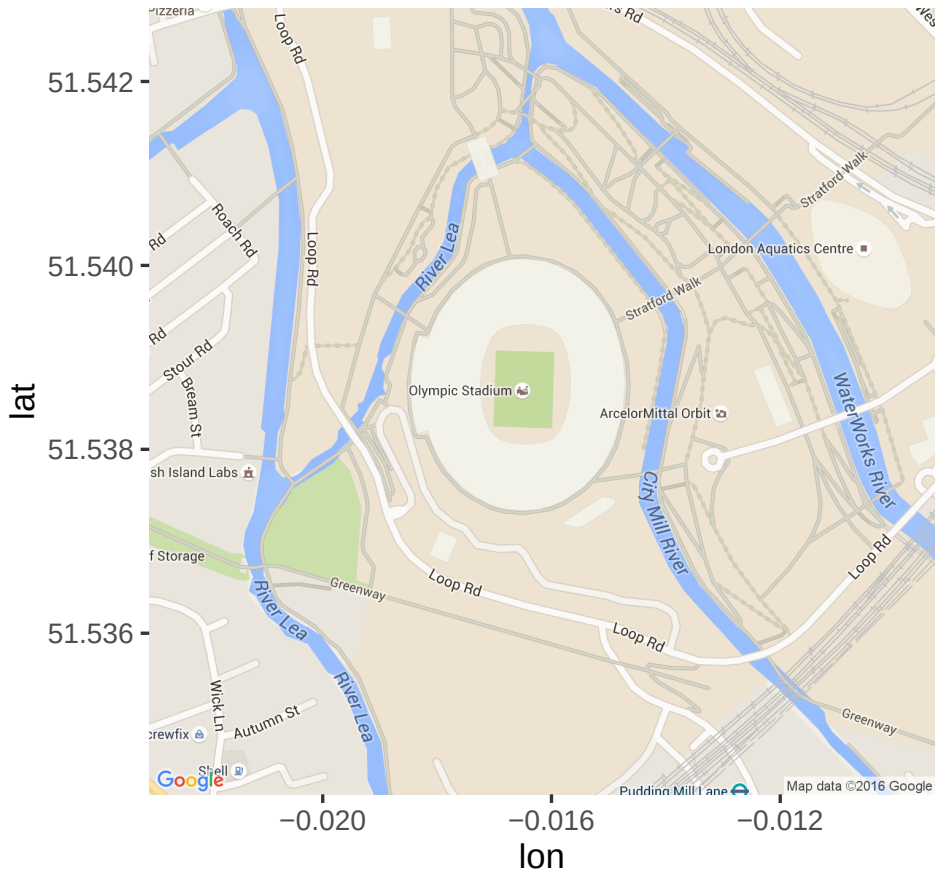
```
> mapData1 <- get_map(location = c(lon = -0.016179, lat = 51.538525),
+                             color = "color",source = "google",
+                             maptype = "satellite",zoom = 16)
> ggmap(mapData1,extent = "panel",ylab = "Latitude",xlab = "Longitude")
```



The London Olympic Stadium - same but different

- Different map type:

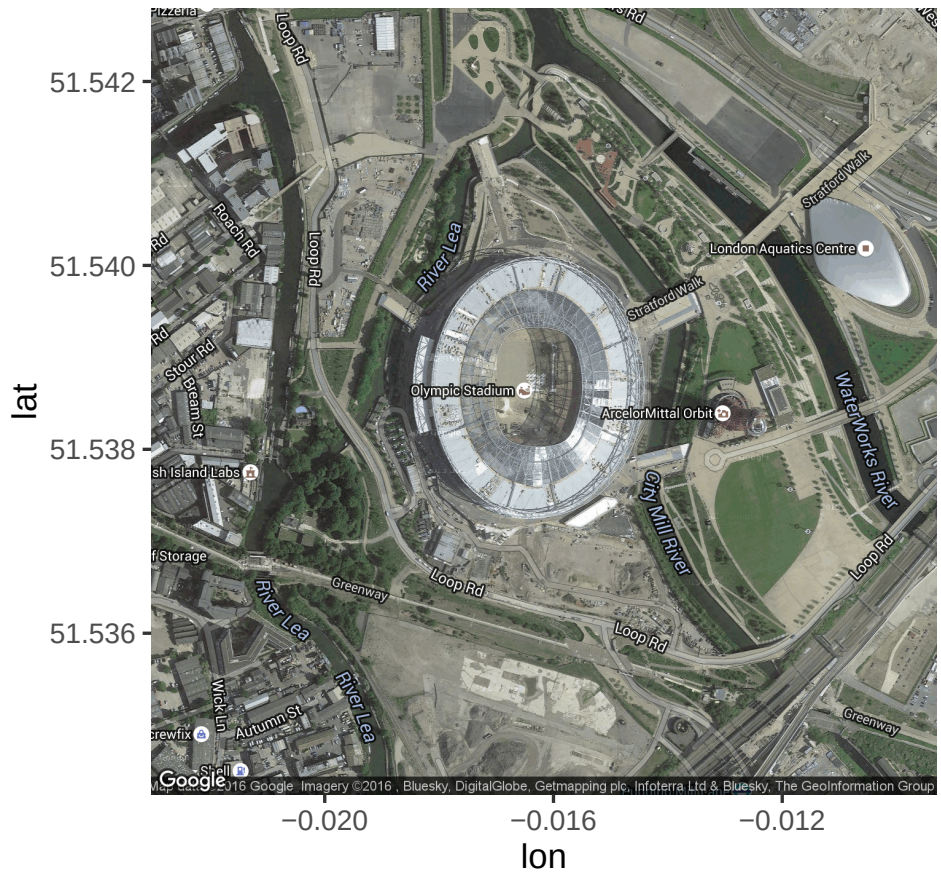
```
> mapData <- get_map(location = c(lon = -0.016179, lat = 51.538525),
+                            color = "color",source = "google",
+                            maptype = "roadmap",zoom = 16)
> ggmap(mapData,extent = "panel",ylab = "Latitude",xlab = "Longitude")
```



The London Olympic Stadium - same but hybrid

- Different map type:

```
> mapData <- get_map(location = c(lon = -0.016179, lat = 51.538525),
+                            color = "color",source = "google",
+                            maptype = "hybrid",zoom = 16)
> ggmap(mapData,extent = "panel",ylab = "Latitude",xlab = "Longitude")
```



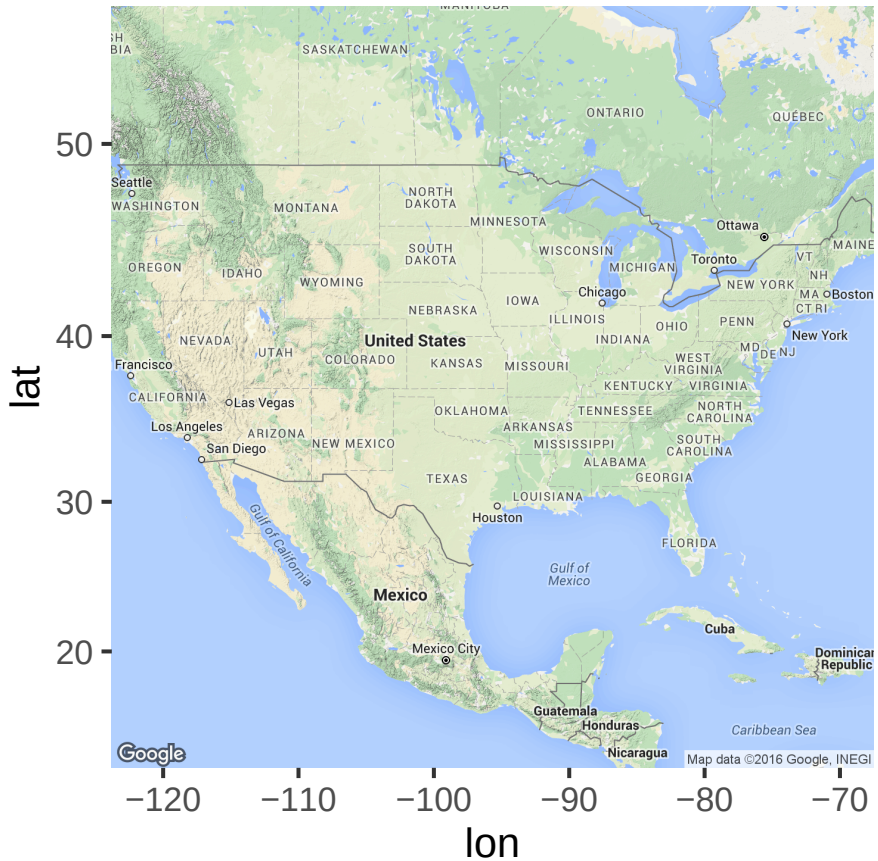
Overlaying maps

- Geographic coordinates obtained with the geocode() function:

```
> geocode("University of Washington")
        lon      lat
1 -106.4407 31.76788
```

- A map of the USA: Lets overlay this map with data.

```
> usa_center <- geocode("United States")
> USA <- ggmap(get_map(location=usa_center, source="google",zoom=4),
+              extent="panel")
> USA
```



Fatal vehicle accidents in the USA 2012

- mv_collisions data:

```
> head(mv_collisions)

        state collisions
1     Alabama         78
2     Arizona        145
3    Arkansas         46
4  California        722
5    Colorado         77
6 Connecticut         40
```

- Getting the geocoordinates with geocode():

```
> for (i in 1:nrow(mv_collisions)) {
+   latlon = geocode(mv_collisions$state[i])
+   mv_collisions$lon[i] = as.numeric(latlon[1])
+   mv_collisions$lat[i] = as.numeric(latlon[2])
+ }
```

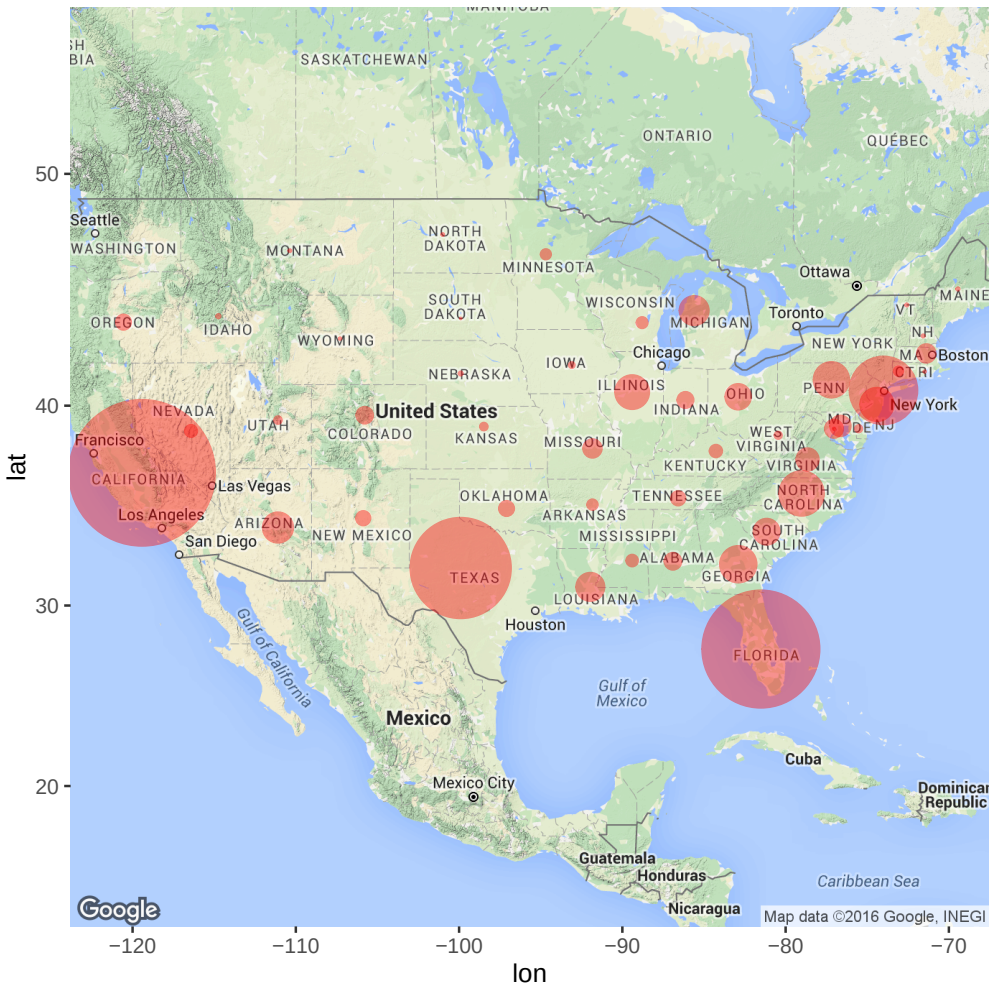
- Getting the map:

```
> usa_center = geocode("United States")
> USA <- ggmap(get_map(location=usa_center,zoom=4), extent="panel")
```

Fatal vehicle accidents in the USA 2012

- Overlaying the data:

```
> circle_scale<-0.04
> USA + geom_point(aes(x=lon, y=lat), data=mv_collisions, col="red",
+                  alpha=0.4, size=mv_collisions$collisions*circle_scale)
```



Credits

- Original coding of the fatal motor vehicle collision example: Sean Lorenz.

- `ggmap`:
  D. Kahle and H. Wickham (2013): ggmap: Spatial Visualization with ggplot2.The R Journal, 5(1), 144-161. URL: `http://journal.r-project.org/archive/2013-1/kahle-wickham.pdf`