

R session 2: Customizing R Graphics and Data Manipulations

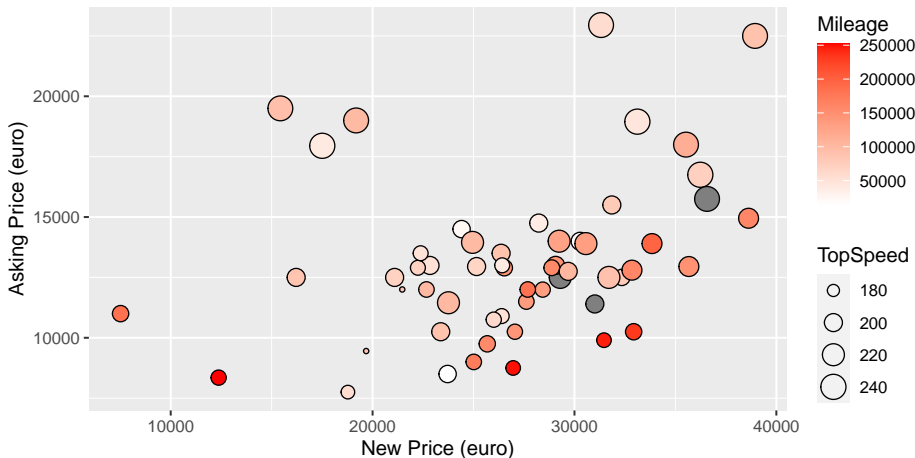
Alina Ferecatu
Rotterdam School of Management
Erasmus University

9/24/2020

Target

Asking Price versus New Price

56 second-hand VW Golfs from Marktplaats.nl



->After session 2 (+ statistics and data manipulations)

Today's lecture

- Customizing ggplot2 graphics
- Data manipulations using dplyr

Recap

Documents and software

Have the **latest version** of:

- R: <https://CRAN.R-project.org>
- RStudio Desktop: <https://www.rstudio.com>
- Installation instructions on Canvas

Software requirements

Package tidyverse (for ggplot2 and dplyr), and haven:

```
library("tidyverse")  
library("haven")
```

Optionally, also colourpicker and ggthemes:

```
library("colourpicker")  
library("ggthemes")
```

Data sets: - Prestige.RData: Prestige of occupations in Canada - vwggolf.RData:
Dutch ads of second-hand 2009 Volkswagen Golfs

An R session

- Create a new script file: File -> New file -> R Script
- Save script file: Keyboard shortcut: *Ctrl / Cmd + S*
- Do not store the objects (workspace) you created
- Execute the line in which your cursor is with *Ctrl / Cmd + Enter*

Loading data

File → Open File ... and select the R data file in the dialog

Path relative to the current working directory:

If the file is not in the working directory, specify the full path:

```
load("~/Documents/CMR/session_1/Prestige.RData")
```

→ No need for full path if file is in the working directory

Always use / and not \ (Windows!)

Basic plotting with package ggplot2

Add together two basic elements:

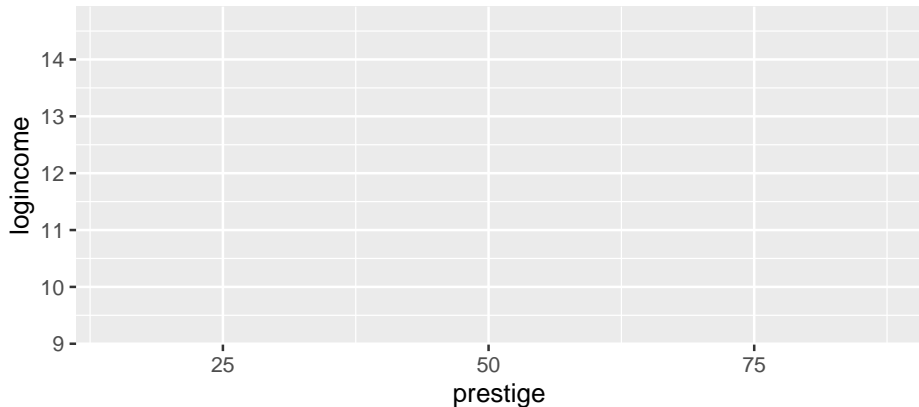
- ❶ Scaffolding defined by `ggplot()`
 - Selects the data set
 - Defines the variables to be used (the aesthetic mapping): function `aes()`
- ❷ Any number of visual representations of the data, known as `geoms`
 - Define the visual representation (the geometric objects): function family `geom_x()`
 - Different elements are added to the plot using the `+` operator

Load the package:

```
library("ggplot2")
```

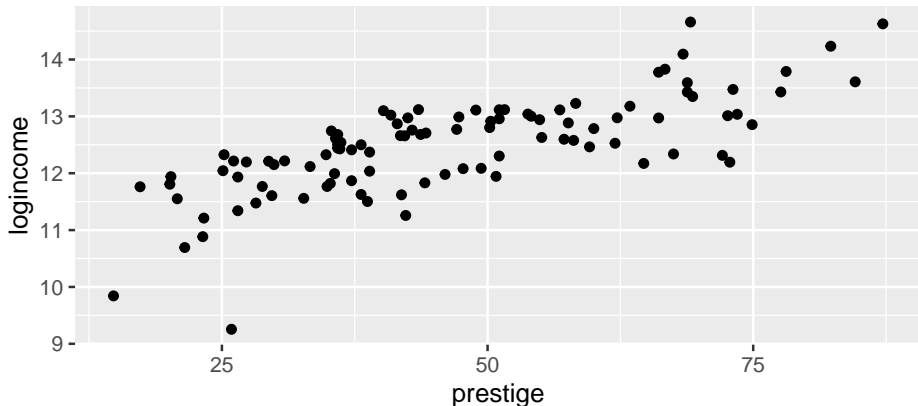
Scatterplot: Scaffolding

```
ggplot(Prestige, aes(x = prestige, y = logincome))
```



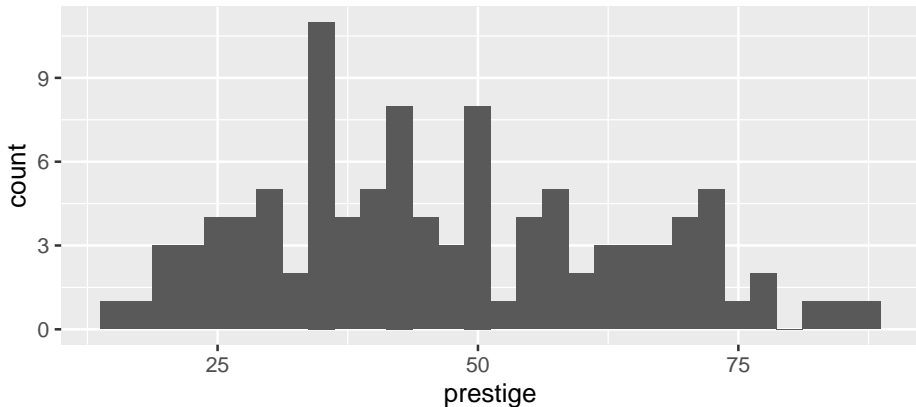
Scatterplot: Scaffolding + points

```
ggplot(Prestige, aes(x = prestige, y = logincome)) +  
  geom_point()
```



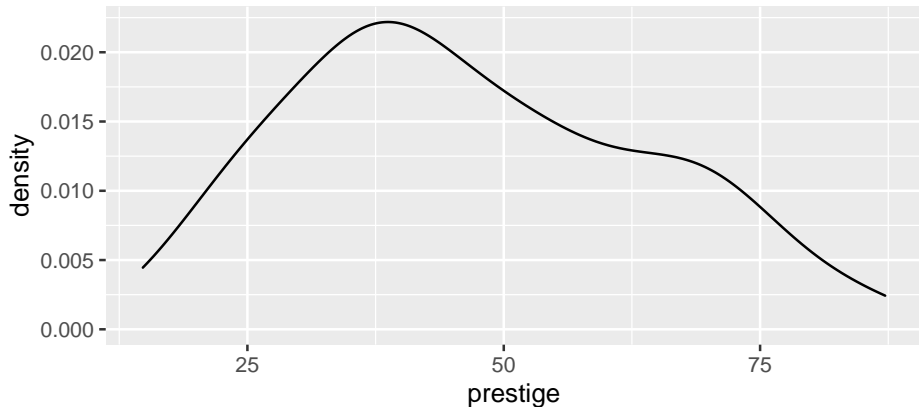
Histogram

```
ggplot(Prestige, aes(x = prestige)) +  
  geom_histogram()
```



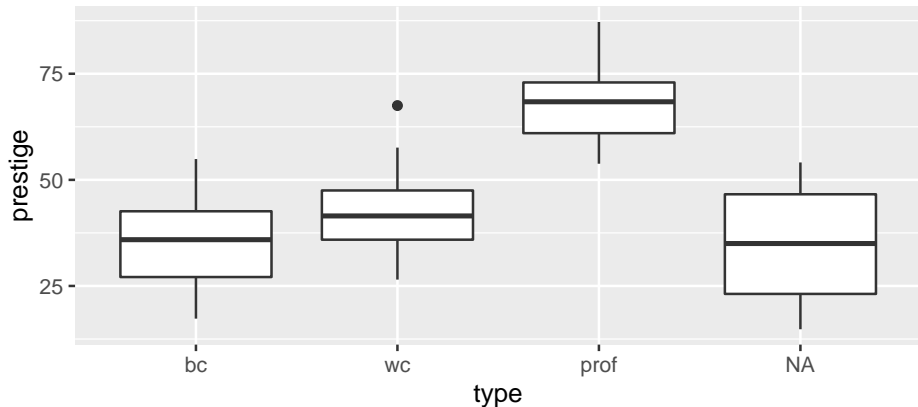
Density plot

```
ggplot(Prestige, aes(x = prestige)) +  
  geom_density()
```



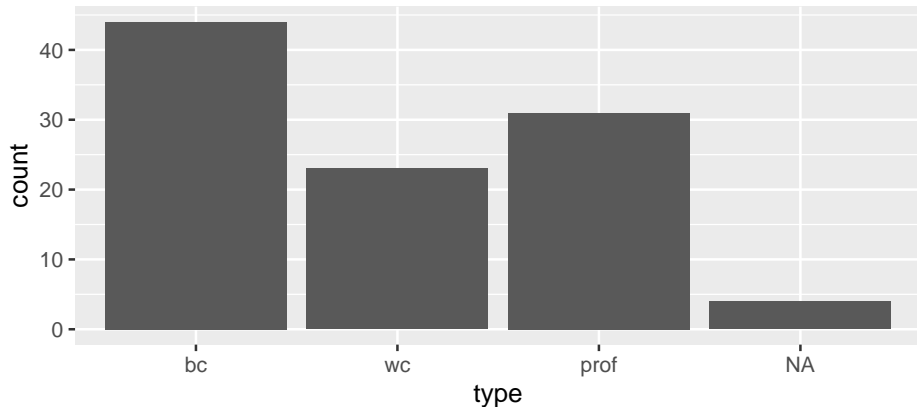
Conditional boxplot

```
ggplot(Prestige, aes(x = type, y = prestige)) + geom_boxplot()
```



Barplot

```
ggplot(Prestige, aes(x = type)) + geom_bar()
```



Some geoms

For a complete list of geoms, click [here](#). Important ones include:

`geom_point()`: Points

`geom_line()`: Lines / time series

`geom_[h/v]line()`: Horizontal or vertical line

`geom_bar()` Bars

`geom_boxplot()` Box and whiskers plot

`geom_density()` Density estimate

`geom_smooth()` Fitted regression line

`geom_text/label` Text

`geom_tile()` Rectangles for heat maps

→ Use appropriate geoms!

Customizing ggplot2 Graphics

VW Golf data

Load the data set:

```
load("~/Documents/CMR/session_2/vwgolf.RData")
```

View the data in RStudio:

```
View(vwgolf)
```

Print the dimensions:

```
dim(vwgolf)
```

Univariate summary statistics:

```
summary(vwgolf)
```

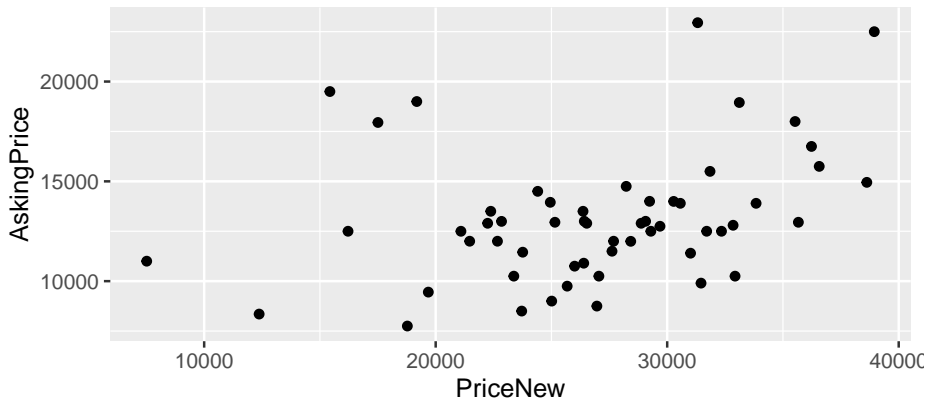
Exercise 1

Create a new R script, and do the following:

- 1 Create a scatterplot of Mileage against AskingPrice
- 2 Create a scatterplot of Mileage against PriceNew minus AskingPrice
- 3 Create a histogram and density plot of Mileage
- 4 Create boxplots of Mileage conditional on Fuel

Scatterplot

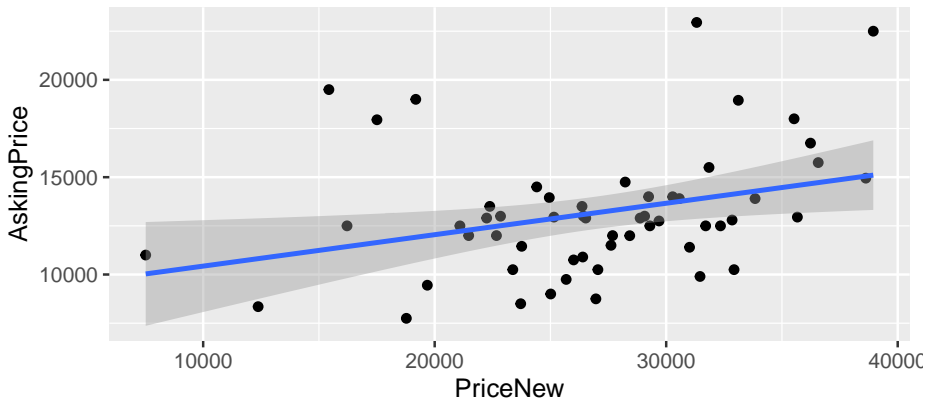
```
ggplot(vwggolf, aes(x = PriceNew, y = AskingPrice)) + geom_point()
```



Adding multiple geoms

Multiple layers can be added to a plot, such as a regression line:

```
ggplot(vwggolf, aes(x = PriceNew, y = AskingPrice)) +  
  geom_point() + geom_smooth(method = "lm")
```



Arguments for Fine-tuning

Finetuning can be done for:

colour Colour for points/lines (alternatively, color)

fill Fill colour for areas

alpha Transparency of colours

shape Symbol for points

linetype Type of line

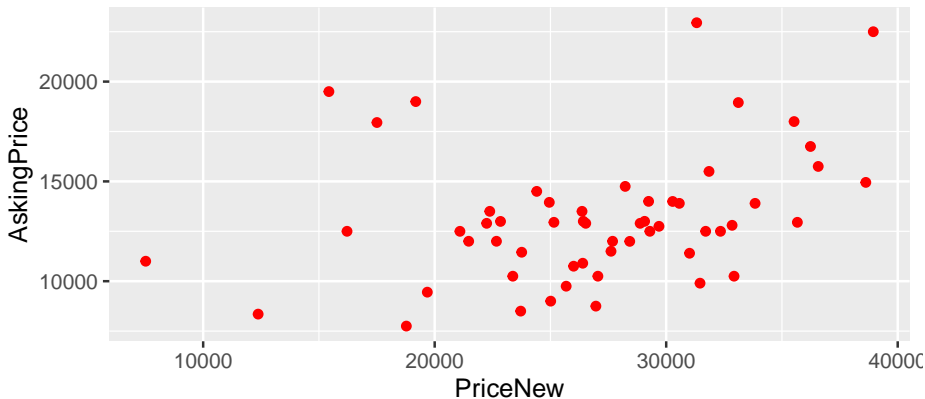
size Size of points/lines

If same setting should be used for all points/lines/areas:

—> Best to supply those arguments to `geom_ xxx()`

Scatterplot with Colour

```
ggplot(vwgolf, aes(x = PriceNew, y = AskingPrice)) +  
geom_point(colour = "red")
```

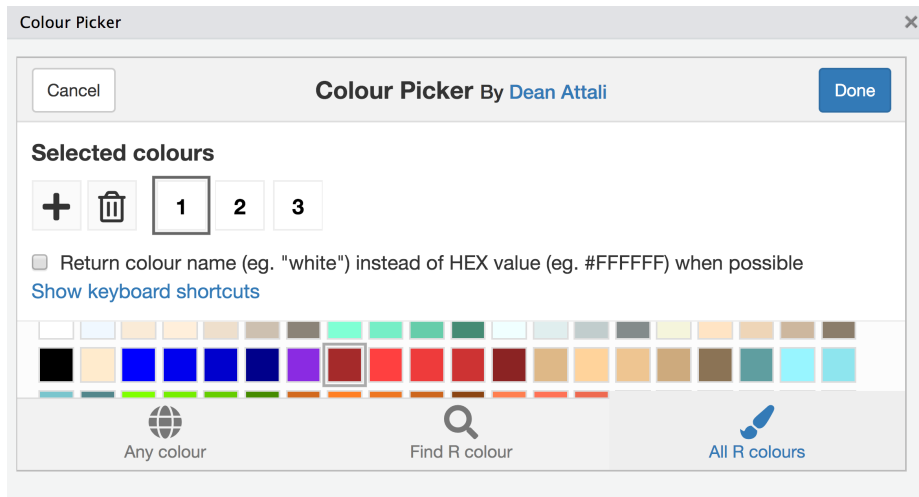


Colors

```
colours()
```

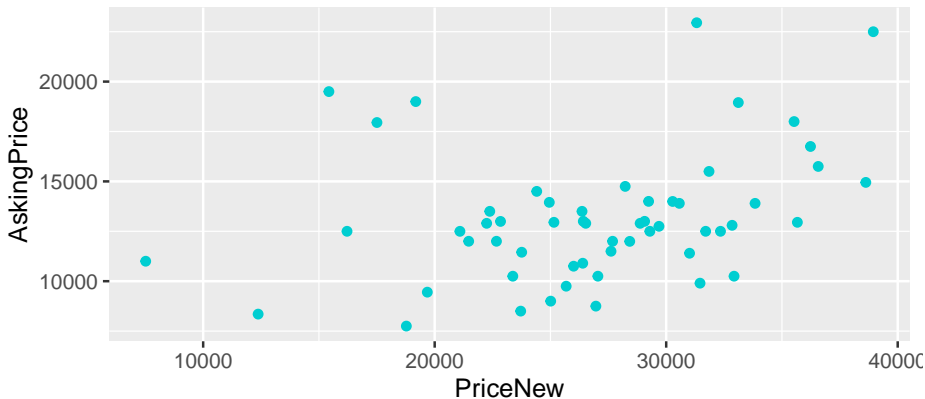

Colour Picker

Package colourpicker contains a nice Colour Picker, which will appear in RStudio under Addins after installation:



Scatterplot with Colour

```
ggplot(vwgolf, aes(x = PriceNew, y = AskingPrice)) +  
geom_point(colour = "#00CED1")
```



Kernel Density Plot with Colour

```
ggplot(vwgf, aes(x = AskingPrice)) +  
  geom_density(fill = "orangered2")
```



Plot Symbols

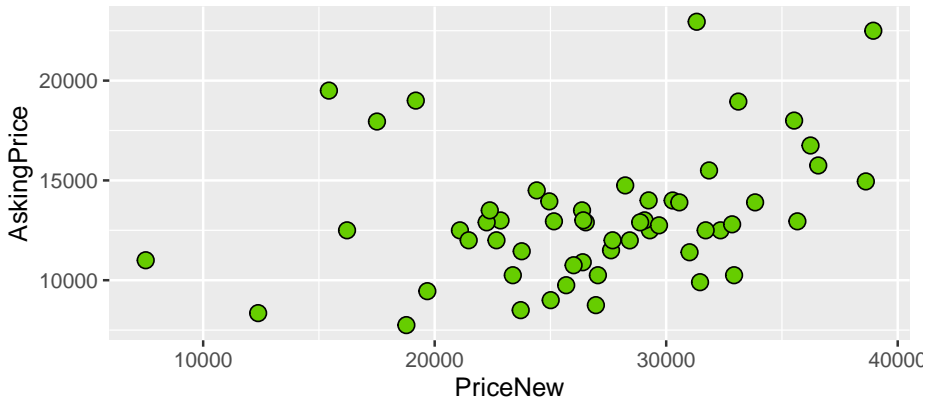


Figure 2: Symbols

—>Specify fill colour for symbols 21—25 with parameter fill

Adjusted Scatterplot

```
ggplot(vwgolf, aes(x = PriceNew, y = AskingPrice)) +  
  geom_point(colour = "black", fill = "chartreuse3", shape = 21,  
            size = 3)
```



Linetypes

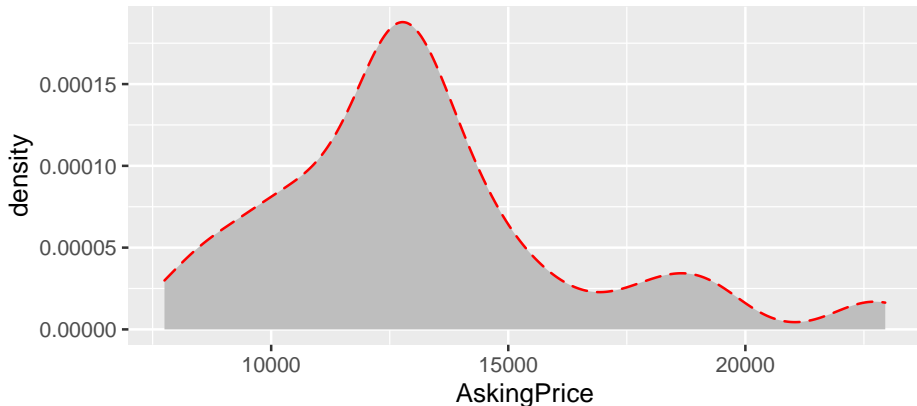


Figure 3: Symbols

—> Specify line type by index or character string

Kernel Density Plot: Linetype

```
ggplot(vwggolf, aes(x = AskingPrice)) +  
geom_density(fill = "grey", linetype = 5, colour = "red")
```



Exercise 2

Adjust your plots from the previous exercises as indicated:

- ❶ Create a scatterplot of Mileage against AskingPrice
 - Increase the point size to 3. Set the plotting symbol to 22. Use a purple colour for filling the symbols.
- ❷ Create a histogram of Mileage
 - Change the type of line to dashed, and the colour of the line to red.
 - Also change the fill colour to red. Set the fill colour to be partly transparent using $\alpha = 0.5$.

Mapping Parameters to Aesthetics

Mapping graphical parameters to variables adds more information to a plot Colour / shape / etc. can be mapped to depend on the value of a variable

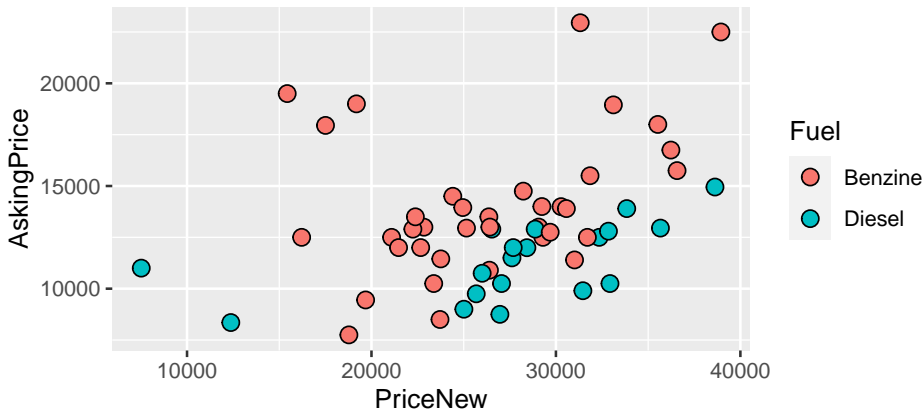
Specify such mappings in the aesthetic mapping in `aes()`

The scale `xxx()` functions determine how different colours / shapes / etc. are assigned

—> Powerful, but don't add too much information

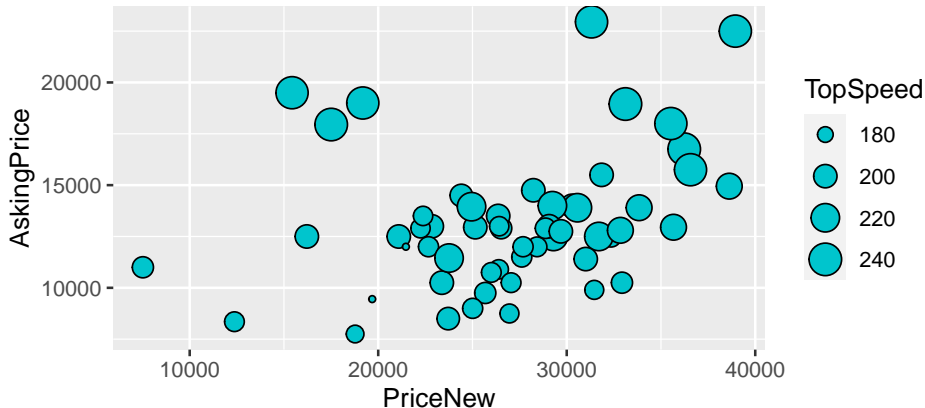
Scatterplot: Colour Mapping

```
ggplot(vwgolf, aes(x = PriceNew, y = AskingPrice, fill = Fuel)) +  
geom_point(shape = 21, size = 3)
```



Scatterplot: Size Mapping

```
ggplot(vwgf,
  aes(x = PriceNew, y = AskingPrice, size = TopSpeed)) +
  geom_point(shape = 21, fill = "turquoise3")
```



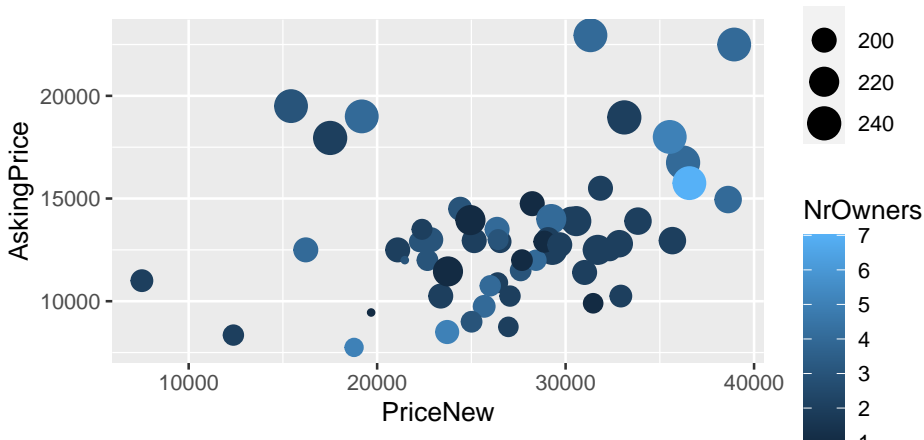
Scatterplot: Colour Mapping

```
ggplot(vwggolf,  
  aes(x = PriceNew, y = AskingPrice, colour = NrOwners)) +  
  geom_point(size = 4)
```



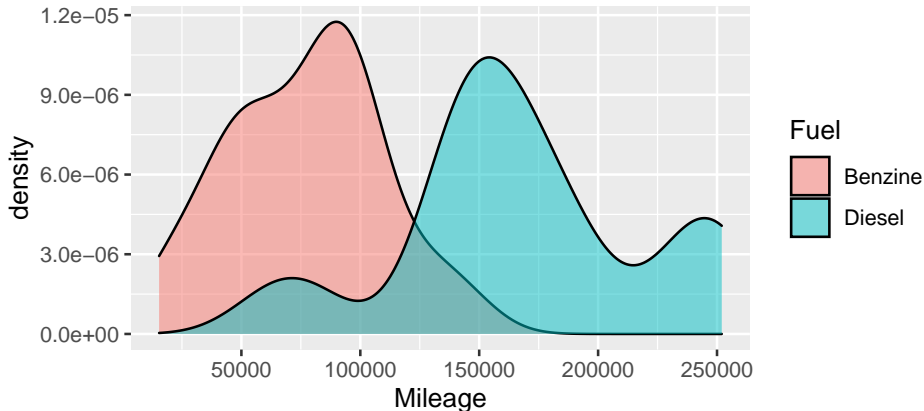
Scatterplot: Colour and Size Mappings

```
ggplot(vwggolf, aes(x = PriceNew, y = AskingPrice,  
  colour = NrOwners, size = TopSpeed)) +  
geom_point()
```



Kernel Density Plot: Fill

```
ggplot(vwgf, aes(x = Mileage, fill = Fuel)) +  
geom_density(alpha = 0.5)
```



Exercise 3

- 1 Plot PriceNew against AskingPrice using a scatterplot. Let the size of the points depend on TopSpeed. Use shape 21, and let the fill colour depend on Mileage.
- 2 Create a conditional boxplot of PriceNew conditional on Colour. Map the fill colour of the boxes to Colour.

Custom Scales

—> Built-in scales are used to obtain default values

Custom discrete scales supplied through function family `scale_xxx_manual()`:

`scale_colour_manual()` - Colour for points/lines

`scale_fill_manual()` - Fill colour for areas

`scale_alpha_manual()` - Transparency of colours

`scale_shape_manual()` - Symbol for points

`scale_linetype_manual()` - Type of line

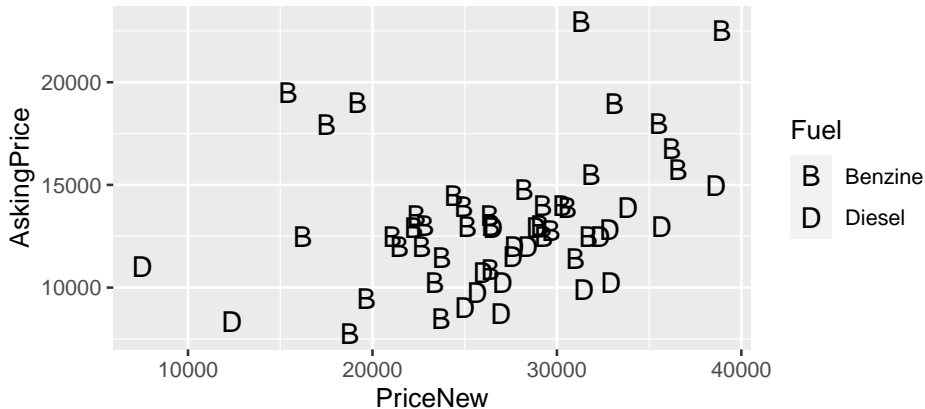
`scale_size_manual()` - Size of points/lines

—> Custom values are supplied via argument values

—> Scales are added to a plot with the `+` operator

Scatterplot: Manual Shapes

```
ggplot(vwgf, aes(x = PriceNew, y = AskingPrice, shape = Fuel)) +  
geom_point(size = 4) + scale_shape_manual(values = c("B", "D"))
```



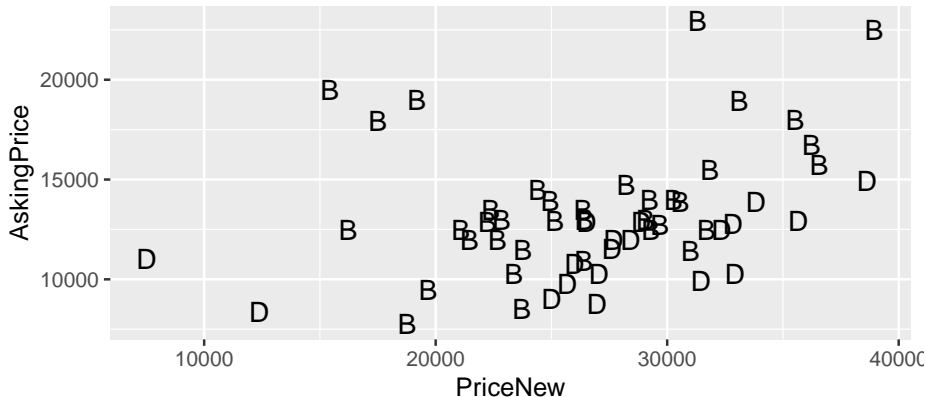
Removing Legends

For some plots, the legend might not provide new information:

—> Can be removed by setting the argument `show.legend` to `FALSE` in the `geom`

Scatterplot: Manual Shapes

```
ggplot(vwgfolf, aes(x = PriceNew, y = AskingPrice, shape = Fuel)) +  
  geom_point(size = 4, show.legend = FALSE) +  
  scale_shape_manual(values = c("B", "D"))
```



Scatterplot: Manual Colours

```
ggplot(vwggolf,  
  aes(x = PriceNew, y = AskingPrice, colour = Imported)) +  
  geom_point() +  
  scale_colour_manual(values = c("#1333E6", "#EB934C"))
```



Continuous Colour Scales

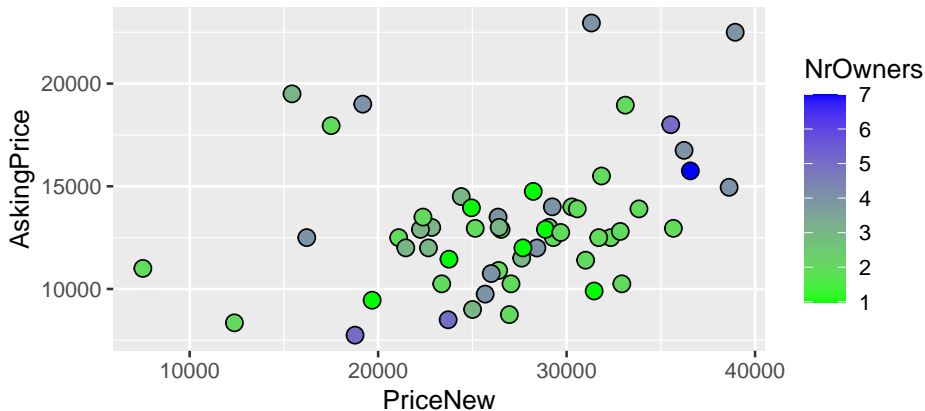
For scales based on continuous variables:

Sequential colour scales are more appropriate

For example, `scale_colour_gradient()` from dark to light

Scatterplot: Manual Colours

```
ggplot(vwggolf,  
  aes(x = PriceNew, y = AskingPrice, fill = NrOwners)) +  
  geom_point(shape = 21, size = 3) +  
  scale_fill_gradient(low = "green", high = "blue")
```



Picking Colours

Some colours are better at conveying the information than others

- > Choose carefully, e.g., using packages such as `colorspace`, or the `ggplot2` defaults
- > Never use red and green together

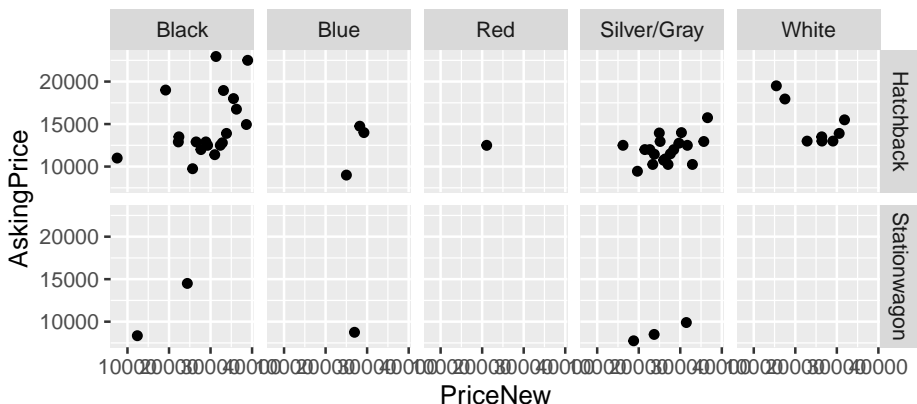
Exercise 4

- 1 Create a scatterplot of PriceNew against AskingPrice. Let the point size depend on TopSpeed and the colour depend on Mileage. Use shape 21 with a fill.
- 2 Change the scale of the fill colour to range from white to red. Store the plot as an object named price plot.
- 3 Why are there grey dots in the plot?

Facets

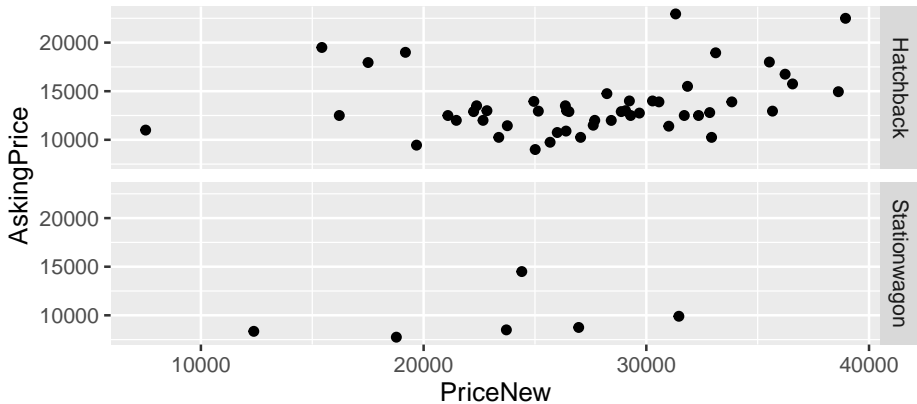
`facet_grid()` constructs multiple panels or facets:

```
ggplot(vwggolf, aes(x = PriceNew, y = AskingPrice)) +  
  geom_point() + facet_grid(Class ~ Colour)
```



You can use single or multiple variables to define the facets

```
ggplot(vwgolf, aes(x = PriceNew, y = AskingPrice)) +  
geom_point() + facet_grid(Class ~ .)
```



More Options

Axis limits:

Use `lims()` with arguments `x` and `y`

Titles, subtitles, and axis titles:

Use `labs()` with the arguments `title`, `subtitle`, `x` or `y`

Switch horizontal and vertical axes:

Use `coord flip()`

Fix the aspect ratio:

Use `coord fixed()` (Important for representing distances)

—> Simply add to plot object with the `+` operator

Exercise 5

- 1 Add an appropriate title and subtitle to your price plot from Exercise 4.
- 2 Add neater axis labels.
- 3 Use an aspect ratio of one.

Themes

The `theme()` function gives control over other aspects of the plot:

—> See `?theme` for an extensive list of options

—> Setting these options makes use of the element `xxx()` functions (such as `element_blank()`)

You can:

Use the default theme, or

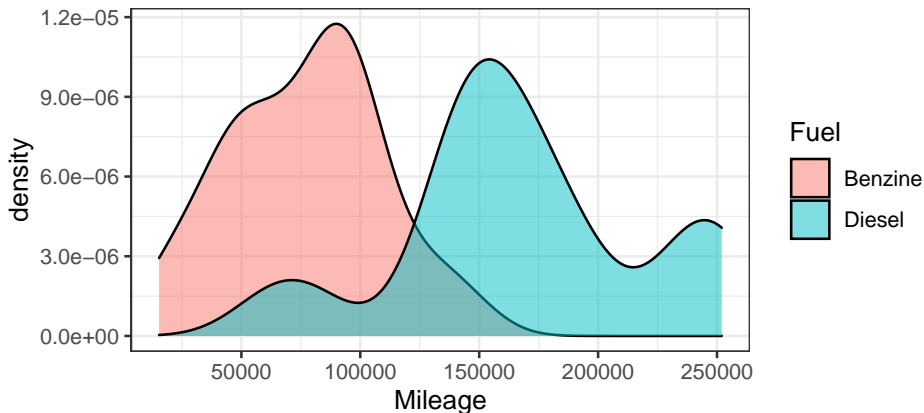
Tweak aspects of the plot to taste, or

Use a built-in theme function, such as `theme_bw()` or `theme_classic()`, or

Even develop your own theme function.

Printer-friendly theme

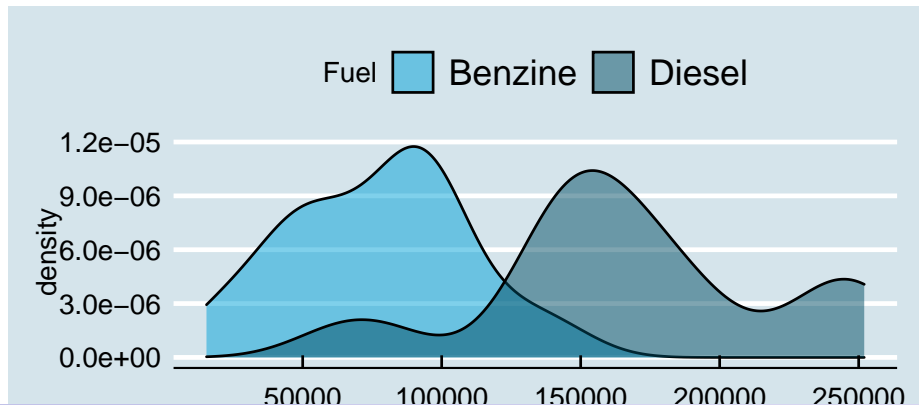
```
ggplot(vwgf, aes(x = Mileage, fill = Fuel)) +  
  geom_density(alpha = 0.5) + theme_bw()
```



Additional themes

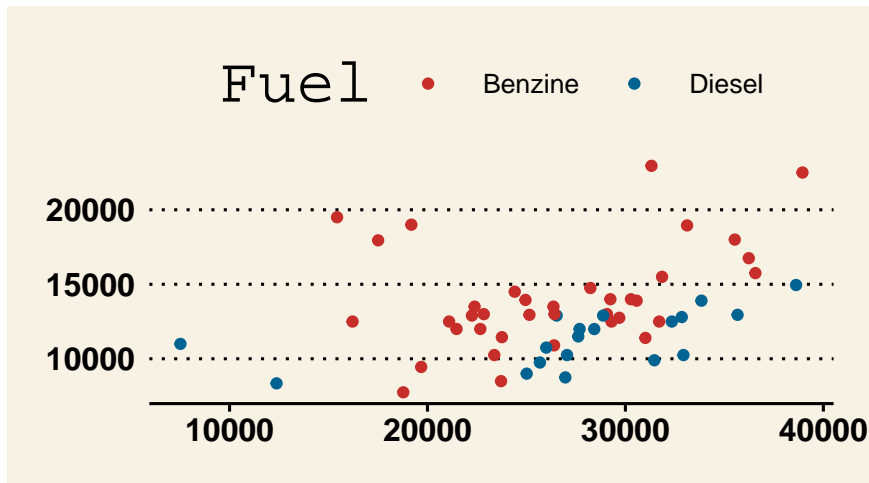
Packages like ggthemes provide additional themes and colour schemes:

```
library(ggthemes)
ggplot(vwggolf, aes(x = Mileage, fill = Fuel)) +
  geom_density(alpha = 0.5) + theme_economist() +
  scale_fill_economist()
```



Wall Street Journal theme and colours:

```
ggplot(vwggolf, aes(x = PriceNew, y = AskingPrice, colour = Fuel)) +  
geom_point() + theme_ws() + scale_colour_ws()
```



Aside: Embedding R graphics in documents

- 1 In the Plots tab of the lower right panel, click Export —> Copy to Clipboard. . .
- 2 In the dialog, change width and height if necessary and click *Copy Plot*
- 3 In the document, paste the plot from clipboard

Data Manipulation with dplyr

dplyr functions

What to do if the data is not exactly in the form you want?

The dplyr package makes it easy to:

- Pick observations by value with `filter()`
- Reorder the rows using `arrange()`
- Pick variables by name using `select()`
- Create new variables from existing ones using `mutate()`
- Collapse many values to a summary statistic using `summarize()`

Philosophy

`filter()`, `arrange()`, `select()`, `mutate()` and `summarize()` all work similarly:

- 1 First argument is a data frame
- 2 Subsequent arguments tell R what to do with the data
- 3 The result is a modified data frame

—> All of these can do group-specific computations using `group by()`

Subsets of observations

—> Function `filter()` finds observations where a specified condition is true and drops all other observations

Keep only the stationwagons

```
filter(vwgf, Class == "Stationwagon")
```

Keep only the black cars

```
filter(vwgf, Colour == "Black")
```

Keep only the black stationwagons

```
filter(vwgf, Colour == "Black", Class == "Stationwagon")
```

Comparisons

Operator/ function **Operation** Example

`==` **exactly equal** `x == y`

`!=` **not equal** `x != y`

`<=` **less or equal** `x <= y`

`>=` **greater or equal** `x >= y`

`<` **less** `x < y`

`>` **greater** `x > y`

`is.na()` **is missing** `is.na(x)`

Basic logic

Operator/ function **Operation** Example

! **not** !is.na(x)

& **and** (x > 0) & (x < 1)

| **or** (x < 0) | (x > 1)

More filtering rows

Keep only cars with Mileage at most 50 000 km

```
filter(vwgfolf, Mileage <= 50000)
```

—> Note that missing values (NA) are excluded

Display cars with missing values on Mileage:

```
filter(vwgfolf, is.na(Mileage))
```

Display cars with Mileage at most 50 000 km, or with Mileage missing:

```
filter(vwgfolf, Mileage <= 50000 | is.na(Mileage))
```


Selecting subsets of variables

—> Function `select()` keeps the specified variables (in that order)

Example: Drop all variables after Fuel

These are all equivalent:

```
vw_num <- select(vwggolf, Version, Class, Transmission, Fuel)
vw_num <- select(vwggolf, Version:Fuel)
vw_num <- select(vwggolf, -(Colour:DaysAPK))
head(vw_num)
```

Rearranging columns, selection based on names

Put AskingPrice and PriceNew first, then everything() else:

```
select(vwggolf, AskingPrice, PriceNew, everything())
```

Select variables with names starting with "Weight"

```
select(vwggolf, starts_with("Weight"))
```

—> Also ends with(), contains(), matches() etc. (see ?starts_with)

Sorting rows with arrange()

—> Rows can be sorted based on one or more variables using arrange()

Sort by AskingPrice:

```
arrange(vwggolf, AskingPrice)
```

—> By default, sorted in ascending order

Use desc() to get descending order:

```
arrange(vwggolf, desc(AskingPrice))
```

Using multiple variables:

```
arrange(vwggolf, Fuel, desc(AskingPrice))
```

Adding or modifying columns

—> Add new or modify existing columns using `mutate()` Add difference between AskingPrice and PriceNew:

```
mutate(vwggolf, PriceDifference = PriceNew - AskingPrice)
```

Transform Mileage to be in 1000's of kilometres:

```
mutate(vwggolf, Mileage = Mileage / 1000)
```

You can do multiple mutations at once:

```
mutate(vwggolf, PriceDifference = PriceNew - AskingPrice,  
Mileage = Mileage / 1000)
```

Summarise

—> Use `summarize()` to collapse many variables into summary statistics

Specify a name for the summary statistic, and an expression giving the value:

```
summarize(vwgolf,  
  NumberOfCars = length(AskingPrice),  
  MeanPrice = mean(AskingPrice),  
  MinPrice = min(AskingPrice),  
  MaxPrice = max(AskingPrice),  
  SDPrice = sd(AskingPrice))
```

Conditional computations

- > Use function `group by()` to make all subsequent computations conditional on one or more grouping variables
- > Use `ungroup()` to undo the grouping behaviour

Summarize AskingPrice per level of Colour:

```
summarize(group_by(vwgf, Colour),  
  NumberOfCars = length(AskingPrice),  
  MeanPrice = mean(AskingPrice),  
  MinPrice = min(AskingPrice),  
  MaxPrice = max(AskingPrice),  
  SDPrice = sd(AskingPrice))
```

Putting it all together

—> The power of dplyr comes from combining multiple operations in succession

- 1 Retain only diesel cars
- 2 Then select only some variables of interest
- 3 Add a new variable called PriceDifference
- 4 Then summarize by PriceDifference and Mileage

In one expression:

```
summarize(
  group_by(
    mutate(
      select(
        filter(vwgolf, Fuel == "Diesel"),
        AskingPrice, PriceNew, Mileage, Colour
      ),
      PriceDifference = PriceNew - AskingPrice
    ),
    Colour
  ),
  MeanPriceDifference = mean(PriceDifference),
  MeanMileage = mean(Mileage)
)
```

—> Alternative 1: Store result from intermediate steps

—> Alternative 2: Use the pipe operator `%>%` to chain the commands from left to right

Chaining with pipes

The function call

```
f(x, y)
```

is the same as

```
x %>% f(y)
```

—>Ctrl/Cmd + Shift + M in RStudio inserts a pipe operator

Piped data manipulation pipeline

Example revisited:

```
vwgolf %>%  
filter(Fuel == "Diesel") %>%  
select(AskingPrice, PriceNew, Mileage, Colour) %>%  
mutate(PriceDifference = PriceNew - AskingPrice) %>%  
group_by(Colour) %>%  
summarize(MeanPriceDifference = mean(PriceDifference),  
MeanMileage = mean(Mileage))
```

Store the final results for reuse if needed:

```
vwgolf_summary <- vwgolf %>%  
  filter(Fuel == "Diesel") %>%  
  select(AskingPrice, PriceNew, Mileage, Colour) %>%  
  mutate(PriceDifference = PriceNew - AskingPrice) %>%  
  group_by(Colour) %>%  
  summarize(MeanPriceDifference = mean(PriceDifference),  
    MeanMileage = mean(Mileage))
```

Conclusions

What we've learned

ggplot2:

- Provides top-class graphics capabilities

- Consistent grammar for lots of plots

- Many options for customization

dplyr:

- Many facilities for constructing recipes for data manipulation

- Useful for exploring and summarizing data

- Useful for preprocessing data before plotting

Acknowledgements

Thank you Pieter Schoonees and Andreas Alfons
for contributing to these materials!

R extra-credit assignment and hackathon

R extra-credit assignment

Create a new R script (that is, a .R file) in RStudio.

Replicate all analyses posted for the CMR sessions 2, 3, 4, and 5, in the *.html* files.

It is important that your R script contains no errors and is easy to read.

Remember to include code that loads the required packages.

Include comments (lines that start with a *#*) in this text file so that it is clear which code replicates which analysis.

Save your work under a file name containing
Yourname_studentNumber_Rassignment.R

Remember to resave regularly.

Submit the file (one file for all sessions) on Canvas, under *R extra-credit assignment*

Deadline: Oct 8 2020 at 1PM, before the workshop

R hackathon

Did you spot an error in the code running analyses for the CMR sessions 2, 3, 4, and 5?

- Add a comment starting with `#ERROR` before the analysis
- Propose a correction, and be very specific for which analysis you are proposing the correction

Do you have a more elegant solution to replicate an analysis?

- Add a comment starting with `#SUGGESTION` before the analysis
- Suggest new code for the analysis, and be very specific for which analysis you are making the suggestion

To participate, add your name in the CMR sign-up document, on the sheet named *R hackathon*

R hackathon winner and prize

The student reporting most errors and suggestions (#ERROR+#SUGGESTION) wins the R hackathon.

