

Chapter 2 Workshop

Table of contents

Dataset Prestige	3
Exercise 2.1	4
Exercise 2.2	7
Exercise 2.3	12
Exercise 2.4	13
Exercise 2.5	14
Exercise 2.6	15
Exercise 2.7	16
Exercise 2.8	17
Exercise 2.9	19
Exercise 2.10	26
Exercise 2.11	27
Exercise 2.12	29

Dataset Prestige

We will be using a well-known dataset called **Prestige** from the **car** R package. This dataset deals with prestige ratings of Canadian occupations. The **Prestige** dataset has 102 rows and 6 columns. Each row (or ‘observation’) is an occupation.

This data frame contains the following columns:

- **education** - Average education of occupational incumbents, years, in 1971.
- **income** - Average income of incumbents, dollars, in 1971.
- **women** - Percentage of incumbents who are women.
- **prestige** - Pineo-Porter prestige score for occupation, from a social survey conducted in the mid-1960s.
- **census** - Canadian Census occupational code.
- **type** - Type of occupation. A factor with levels: bc, Blue Collar; prof, Professional, Managerial, and Technical; wc, White Collar. (includes four missing values).

First we’ll load the data. The dataset sits in the **car** package, so you need to load the **car** package first.

```
library(car)
data(Prestige)
```

Exercise 2.1

Draw a bar chart for `type`:

```
library(tidyverse)

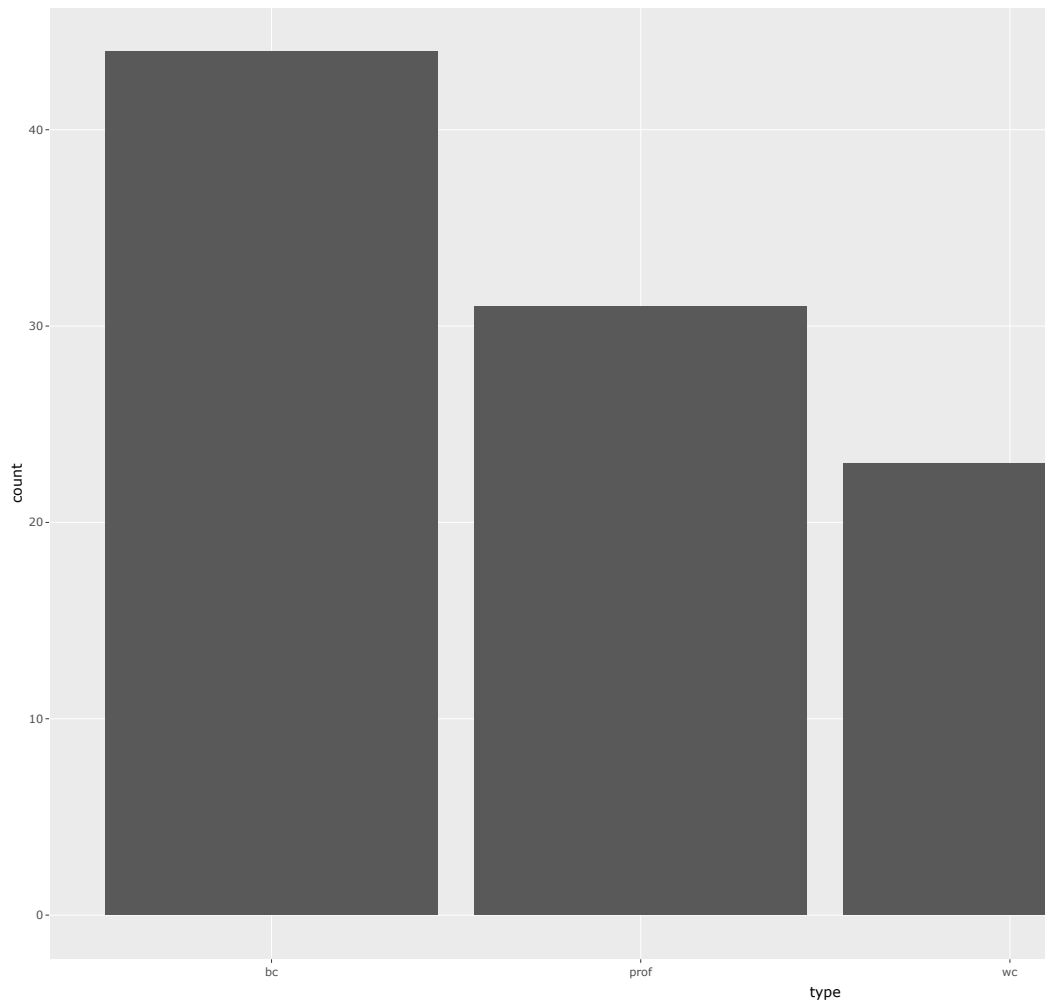
p <- Prestige |>
  ggplot() +
  aes(type) +
  geom_bar()

p
```

Or with `plotly` (which works for HTML, not for PDF)

```
library(plotly)

ggplotly(p)
```



Or with old-style R plot

```
# or  
library(car)  
barplot(table(Prestige$type))
```

Exercise 2.2

Draw a histogram of `prestige`.

Below demonstrates the flexibility of `ggplot` code. You can specify the `data` argument by piping it into `ggplot`, or by putting it as an argument to `ggplot` or a `geom_`. Likewise, the `mapping` or `aes` information, which determines which variables are used where, can be added as an extra line or specified inside the `ggplot` or `geom_` function.

All of these methods give the same result.

```
Prestige |>
  ggplot() +
  aes(x = prestige) +
  geom_histogram(bins=10)

ggplot(
  data = Prestige,
  mapping = aes(x = prestige)
) +
  geom_histogram(bins=10)

ggplot(Prestige) +
  aes(x = prestige) +
  geom_histogram(bins=10)

ggplot() +
  geom_histogram(
    data = Prestige,
    mapping = aes(x = prestige),
    bins = 10
  )

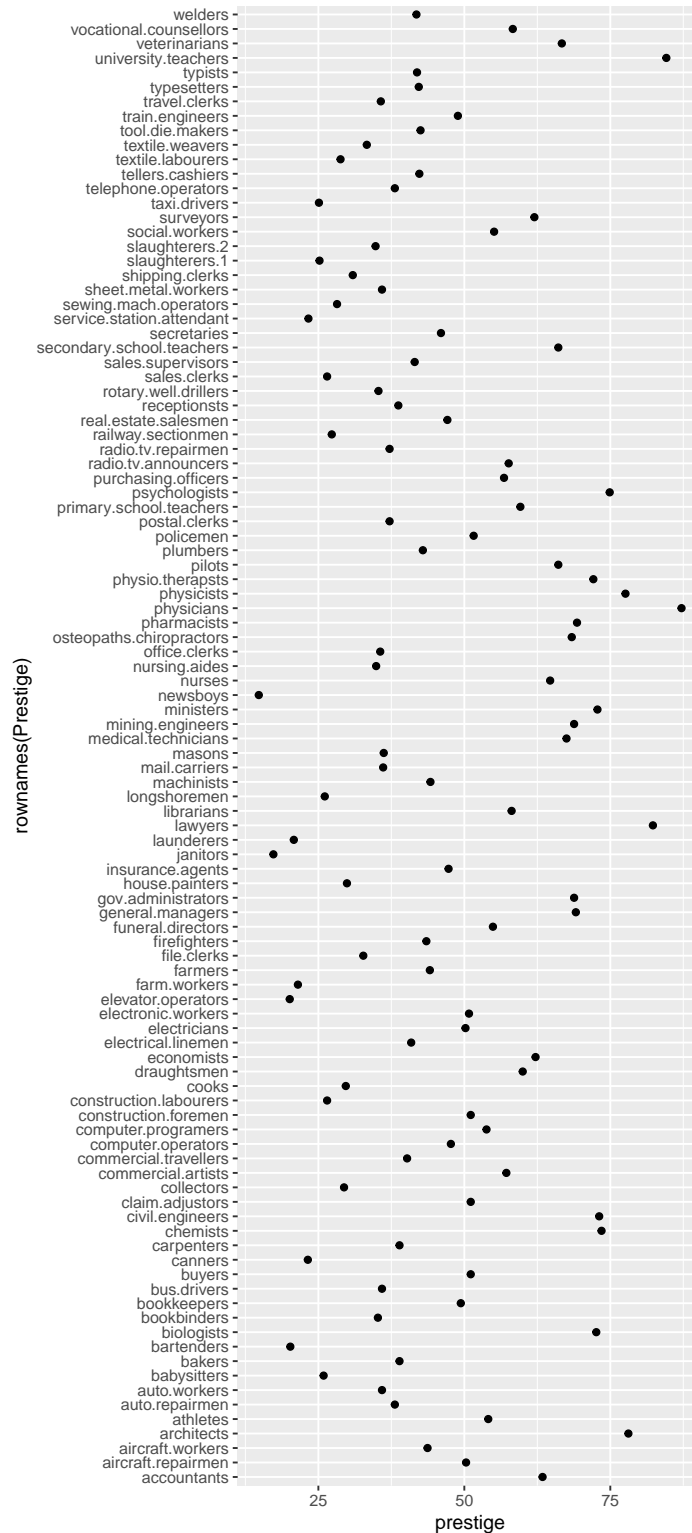
# or
# library(plotly)
```

```
# p <- Prestige |>
#   ggplot() +
#     aes(prestige) +
#     geom_histogram(bins=10)
#
# ggplotly(p)

# or
# hist(Prestige$prestige)
```

We can display this information as a dot plot.

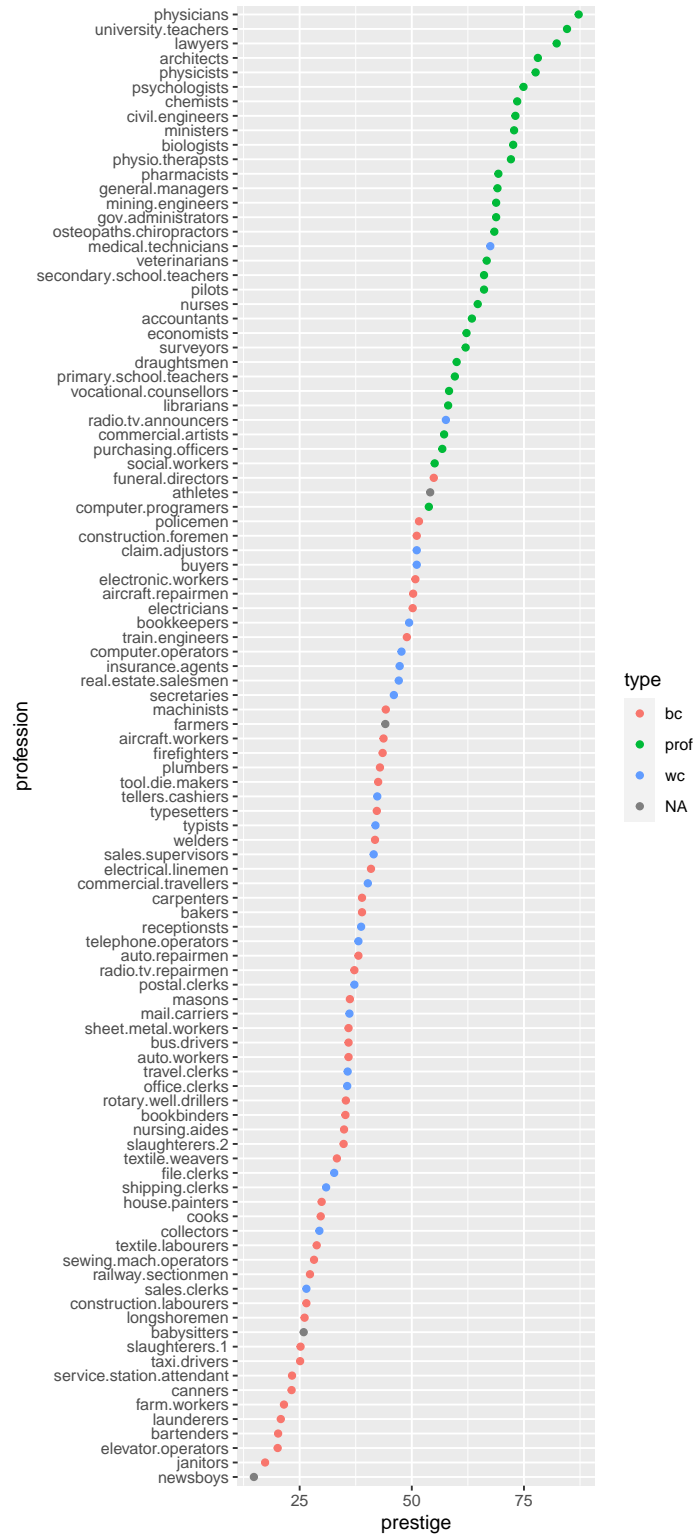
```
Prestige |>
  ggplot() +
  aes(x = rownames(Prestige), y = prestige) +
  geom_point() +
  coord_flip()
```

What a mess!

We can tidy it up by ordering the professions on the plot according to `prestige`. First, we move the professions from rownames to a variable. Then, we `fct_reorder` the professions using `prestige`. Then, the resulting data gets piped into `ggplot`.

```
Prestige |>
  rownames_to_column(var = "profession") |>
  mutate(
    profession = fct_reorder(profession, prestige)
  ) |>
  ggplot() +
  aes(x = profession, y = prestige, colour = type) +
  geom_point() +
  coord_flip()
```



Exercise 2.3

Obtain some summary statistics for `prestige`. There are a few options for this.

```
summary(Prestige)

library(psych)

describe(Prestige)

describeBy(education + income + women + prestige ~ type,
           data = Prestige)
```

Exercise 2.4

Make a boxplot of `prestige ~ type`:

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

# or
# library(plotly)
# p <- Prestige |> ggplot() +
#   aes(y=prestige, x=type) + geom_boxplot()
# ggplotly(p)

# or
# library(lattice)
# bwplot(prestige ~ type, data=Prestige)

# as violin plots
Prestige |>
  ggplot() +
  aes(y=prestige, x=type) +
  geom_violin()

# Or put it all together
Prestige |>
  ggplot() +
  aes(y=prestige, x=type) +
  geom_violin() +
  geom_boxplot(col = 2, alpha = .2) +
  geom_jitter(alpha = .2, width = .2, height = 0, colour = 4)
```

Exercise 2.5

Obtain the Empirical Cumulative Distribution Function (ECDF) graphs of `prestige ~ type`:

```
Prestige |>
  ggplot() +
  aes(prestige, colour=type) +
  stat_ecdf()
```

```
Prestige |>
  ggplot() +
  aes(prestige) +
  stat_ecdf() +
  facet_wrap(~type)
```

```
# or
library(latticeExtra)
ecdfplot(~ prestige | type, data = Prestige)
```

```
Prestige |>
  ggplot() +
  aes(
    x = prestige, # these aes settings are used
    col = type    # by both geoms
  ) +
  geom_density(
    aes(fill = type), # the 'fill' aes goes here because
    alpha = .2        # geom_rug doesn't use 'fill'
  ) +
  geom_rug()
```

Exercise 2.6

Obtain the {0.05, 0.1, 0.25, 0.5, 0.75, 0.9, 0.95} quantiles of `prestige`:

```
pr <- c(0.01, 0.05, 0.1, 0.25, 0.5, 0.75, 0.9, 0.95, 0.99)

Prestige |>
  summarise(
    probs = pr,
    quants = quantile(prestige, pr)
  )

# or simply
quantile(Prestige$prestige, pr)
```

Exercise 2.7

Obtain the scatter plot (with and without marginal boxplots) **prestige vs. education** :

```
library(ggExtra)

p1 <- Prestige |>
  ggplot() +
  aes(x = education, y = prestige) +
  geom_point() +
  geom_smooth(col = "red") +
  geom_smooth(method = "lm", se = FALSE)

ggMarginal(p1, type="boxplot")

library(car)

scatterplot(education ~ prestige, data = Prestige)
```

The later plot will show prediction interval ribbon while the first plot will show the confidence interval ribbon.

Exercise 2.8

Obtain the bubble or balloon plot **prestige vs. education vs. income** (income forming the bubble size):

```
library(ggplot2)

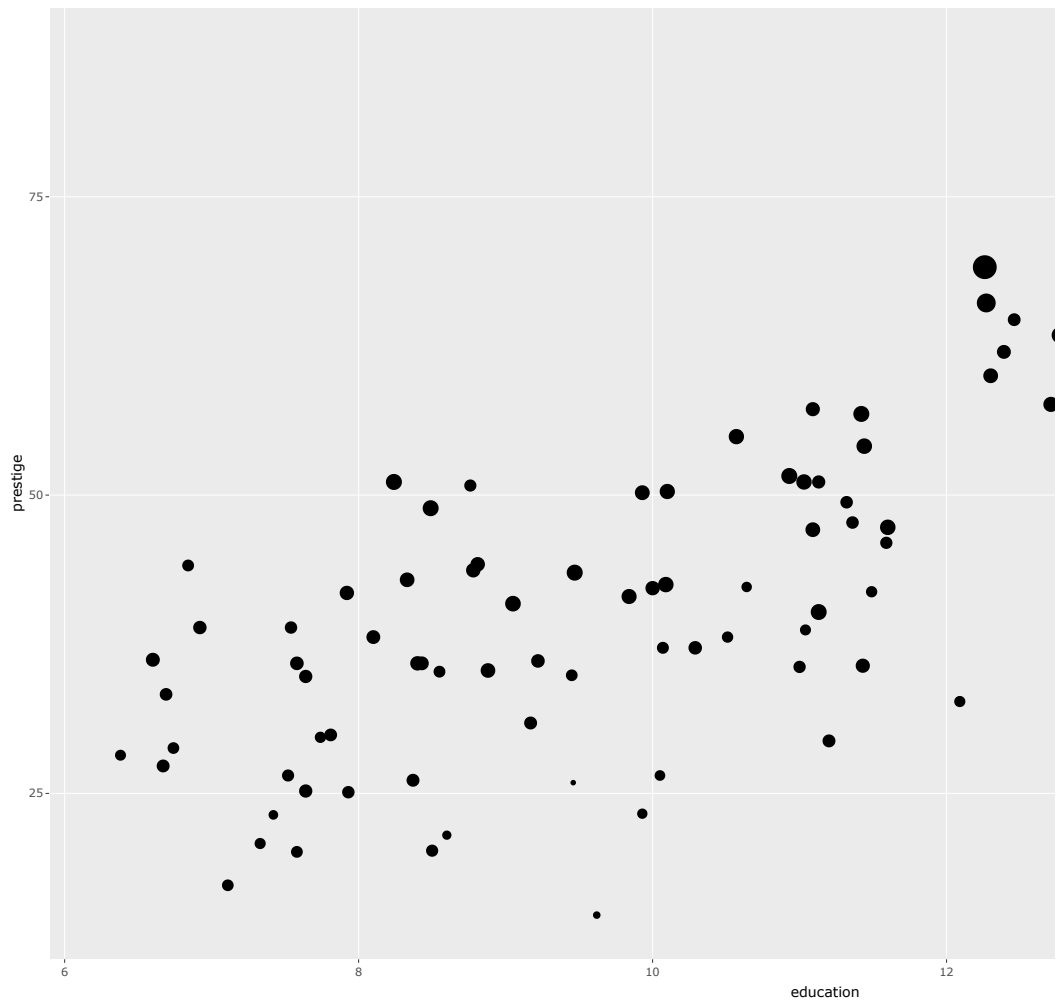
Prestige |>
  ggplot() +
  aes(x = education, y = prestige, size = income) +
  geom_point()

# or

library(plotly)

p <- Prestige |>
  ggplot() +
  aes(x = education, y = prestige, size = income) +
  geom_point()

ggplotly(p)
```



Exercise 2.9

Obtain the contour plot **prestige vs. education vs. income** :

```
library(plotly)

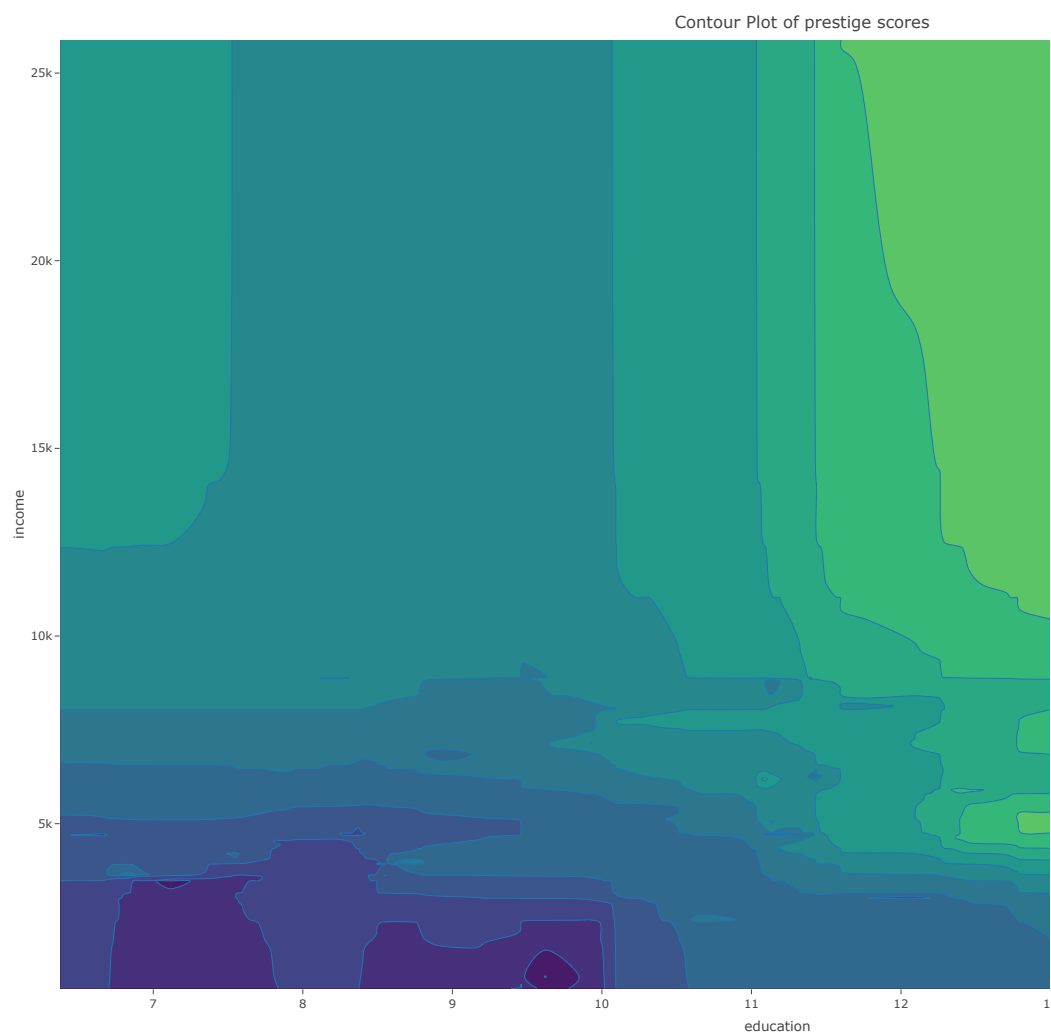
plot_ly(type = 'contour',
        x = Prestige$education,
        y = Prestige$income,
        z = Prestige$prestige)
```



To add axes labels and titles, try-

```
library(plotly)

plot_ly(
  Prestige,
  type = 'contour',
  x = Prestige$education,
  y = Prestige$income,
  z = Prestige$prestige
) |> layout(
  title = 'Contour Plot of prestige scores',
  xaxis = list(title = 'education'),
  yaxis = list(title = 'income')
)
```



We can also define our own function for the contour approximation.

```
library(modelr)

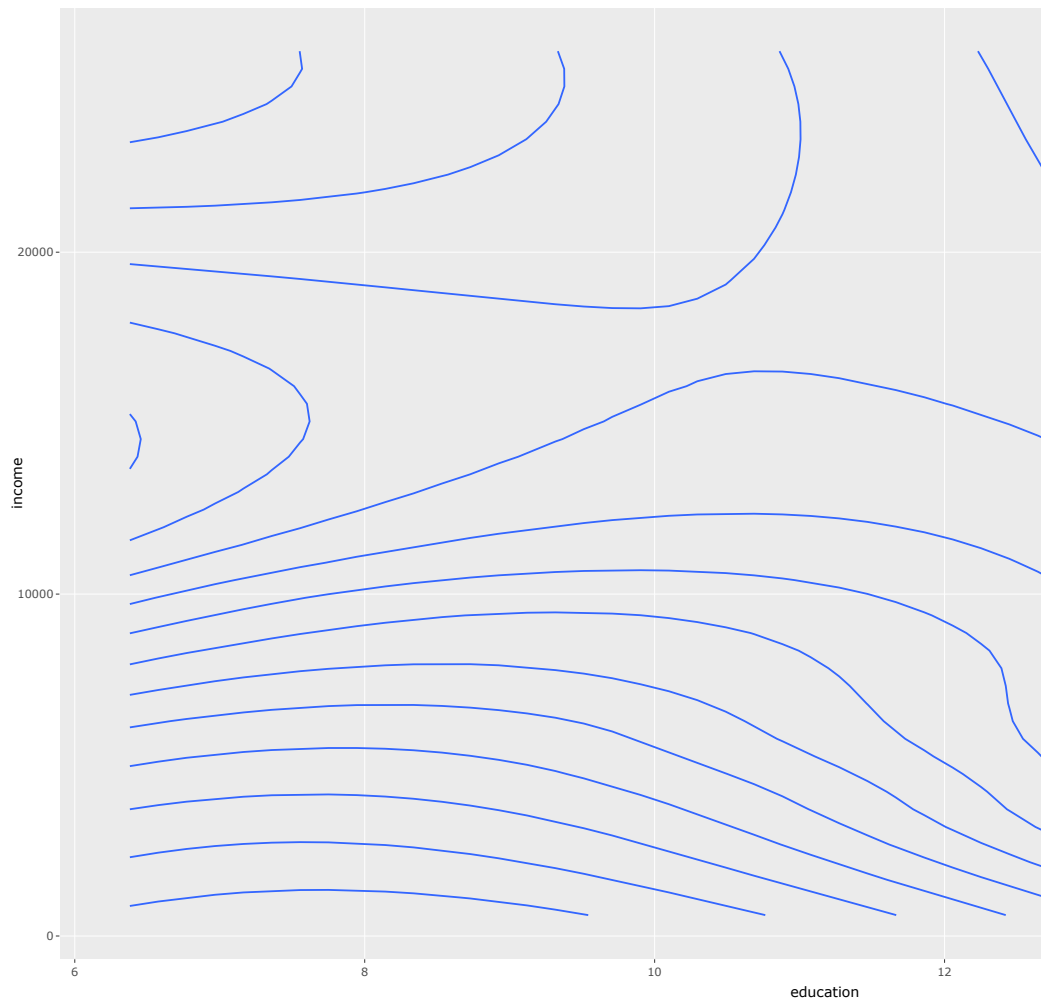
# make a smooth model
y.m = loess(prestige ~ education * income, data = Prestige)

# make a regular grid of all combinations of education and income
mygrid <- Prestige |>
  data_grid(
    education = seq_range(education, 50),
    income = seq_range(income, 50)
  ) |>
  # add predicted prestige using the smooth model
  add_predictions(y.m, var = "predicted prestige")

# make ggplot contour plot
p <- mygrid |>
  ggplot() +
  aes(x = education, y = income, z = `predicted prestige`) +
  geom_contour()

p

# make a plotly version
library(plotly)
ggplotly(p)
```




```

# filled contour ggplot
mygrid |>
  ggplot() +
  aes(x=education, y=income, z=`predicted prestige`) +
  stat_contour_filled()

# or the older-style lattice graphs

library(lattice)

contourplot(`predicted prestige` ~ education * income,
            data = mygrid,
            cuts = 10, region = TRUE,
            xlab = "education ", ylab = "income ")

wireframe(`predicted prestige` ~ education * income,
          data = mygrid,
          cuts = 10, region = TRUE,
          xlab = "education ", ylab = "income ")

levelplot(`predicted prestige` ~ education * income,
          data = mygrid,
          cuts = 10, region = TRUE,
          xlab = "education ", ylab = "income ")

cloud(`predicted prestige` ~ income * education,
      data = mygrid)

```

Exercise 2.10

Obtain the 3-D plot **prestige vs. education vs. income** :

```
library(car)

scatter3d(prestige ~ education + income,
          data = Prestige)
```

Exercise 2.11

Create `prestige ~ education | type` graphs. That is, `prestige ~ education` grouped by `type` as colours and/or panels.

```
Prestige |>
  ggplot() +
  aes(x = education, y = prestige, color = type) +
  geom_point() +
  facet_wrap(~ type)

# or
# library(plotly)
#
# p <- Prestige |>
#   ggplot() +
#   aes(x = education, y = prestige, color = type) +
#   geom_point() +
#   facet_wrap(~ type)
#
# ggplotly(p)

p <- Prestige |>
  ggplot() +
  aes(x = education, y = prestige, color = type) +
  geom_point()

p

# OR
#
# library(plotly)
# ggplotly(p)
```

```
scatterplot(prestige ~ education | type,  
            data=Prestige)
```

```
xyplot(prestige ~ education | type,  
        auto.key = TRUE,  
        data = Prestige)
```

```
xyplot(prestige ~ education,  
        group = type,  
        auto.key = TRUE,  
        data = Prestige)
```

Exercise 2.12

Time-series data EDA based on RBNZ house sales data.

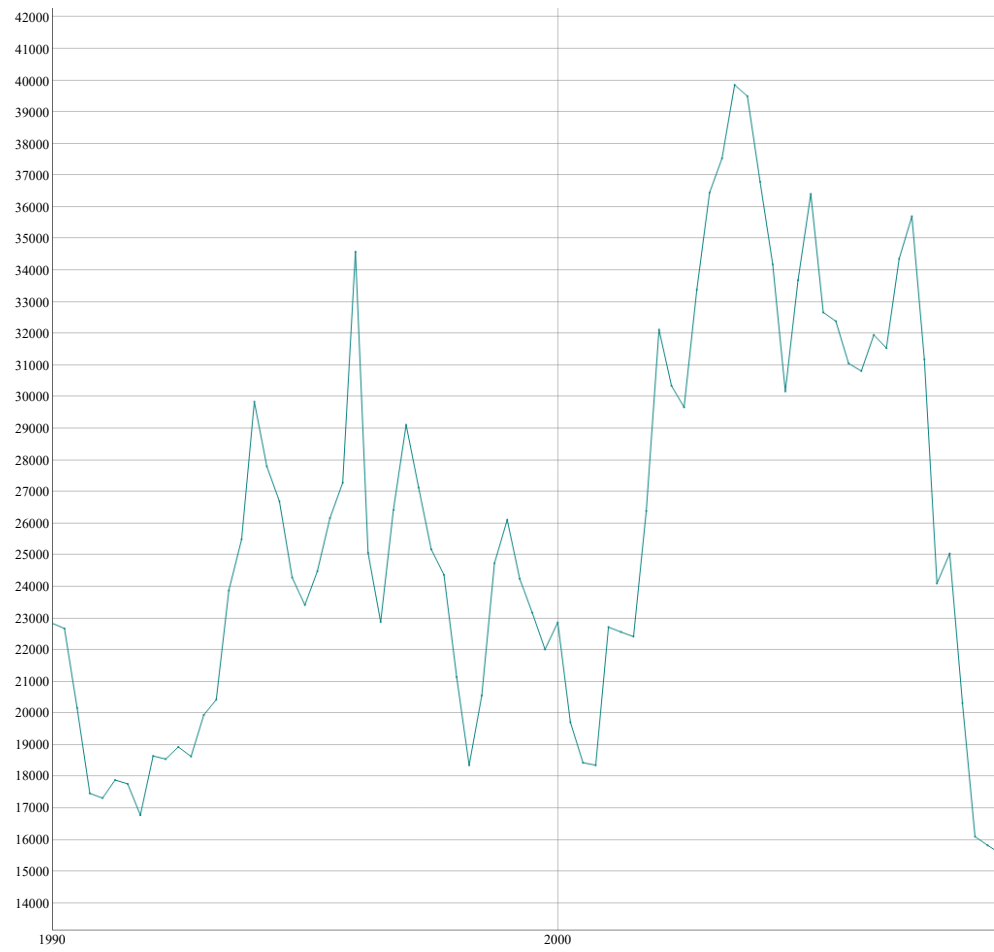
```
url1 <- "https://www.massey.ac.nz/~anhsmith/data/housesales.RData"  
download.file(url = url1, destfile = "housesales.RData")  
load("housesales.RData")
```

```
library(forecast)  
autoplot(housesales)
```

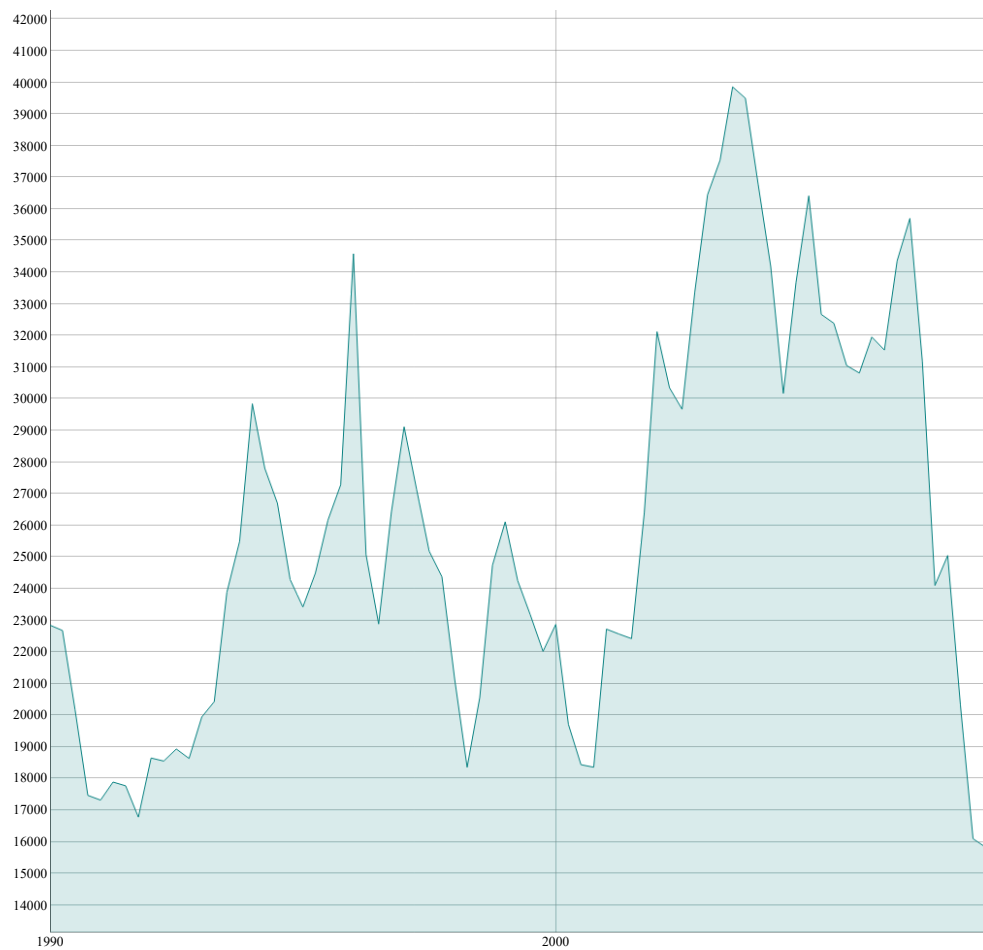
```
# Dynamic graphing  
# https://rstudio.github.io/dygraphs/index.html
```

```
library(dygraphs)
```

```
dygraph(housesales) |>  
  dyOptions(drawPoints = TRUE)
```



```
dygraph(housesales) |>  
  dyOptions(fillGraph=TRUE)
```




```
ggseasonplot(housesales)
```

```
ggsubseriesplot(housesales)
```

Series Decomposition

```
library(tidyverse)
```

```
housesales |>  
  decompose(type="additive") |>  
  forecast::autoplot() +  
  ggtitle("")
```

```
housesales |>  
  decompose(type="multiplicative") |>  
  forecast::autoplot() +  
  ggtitle("")
```

lag & ACF plots

```
gglagplot(housesales)
```

```
gglagplot(housesales, seasonal=FALSE, lag=1)
```

```
ggAcf(housesales)
```

```
ggPacf(housesales)
```

```
ggtsdisplay(housesales)
```

More graphing examples are [here](#) (R code file).