Prelims Stats: Rdemo-2 TT 2026

Old Faithful data

`https://en.wikipedia.org/wiki/Old_Faithful`

Old Faithful is a geyser in Yellowstone National Park, USA. To look at the first few rows of the dataset `faithful`, which is built-in to R, use

```
head(faithful)
```

The two columns in the dataset are:

- `eruptions`, the duration of eruptions of the geyser (in minutes) – take this as explanatory variable x
- `waiting`, the waiting time until the next eruption of the geyser (in minutes) – take this as response variable y. [To see the help page for the dataset which gives these details, use `?faithful`]

Suppose we are interested in using the duration of the current eruption (i.e. `eruptions`) to predict the time until next eruption takes place (i.e. `waiting`).

Plot the data, fit a simple linear regression, and draw the regression line on the plot:

```
plot(waiting ~ eruptions, data=faithful)
fit1 <- lm(waiting ~ eruptions, data=faithful)
fit1

# add the fitted line to the plot
abline(fit1, col="red")
```

The fitted line is $y = 33.47 + 10.73x$. If the most recent eruption lasted 4 minutes, how long does the model predict until the next eruption?

The function `fitted()` extracts the fitted values $\widehat{y}_i$. The function `resid()` extracts the residuals e_i. The function `rstandard()` extracts the residuals r_i, sometimes called "standardised" residuals, which are called "studentised" residuals in the synopses and in JWHT (2013).

Residual plots:

```
plot(resid(fit1) ~ fitted(fit1))
# plot(resid(fit1) ~ eruptions, data=faithful)
# the same plot but different horizontal scale

# similar plot but use the standardised/studentised residuals
plot(rstandard(fit1) ~ fitted(fit1))
```

Auto data

The `Auto` dataset used in lectures is part of the `ISLR` package. To install this package, use

```
install.packages("ISLR")
```

If you are using RStudio this should download and install the package in a few seconds. [If you are using R, not RStudio, then you may get pop-up box asking you to choose a mirror site to download the package from – choose e.g. "UK (Bristol)" and then the package should download and install.]

Load the package and look at the first few rows of the dataset:

```
library(ISLR)
head(Auto)
```

Plot the data, fit a straight line regression, and draw the regression line on the plot:

```
plot(mpg ~ horsepower, data=Auto)
fit2 <- lm(mpg ~ horsepower, data=Auto)
```

```
fit2
abline(fit2, col="red")
```

The curved nature of the following residual plot indicates that a straight line model isn't really appropriate:

```
plot(rstandard(fit2) ~ fitted(fit2))
```

Try a model of the form $y = \beta_0 + \beta_1 x + \beta_2 x^2 + \epsilon$:

```
fit3 <- lm(mpg ~ horsepower + I(horsepower^2), data=Auto)
fit3
```

Note: for the quadratic term to be included as we intend, the identity function `I()` is needed above. Note also: when using the `lm()` function above, we have specified the linear term `horsepower` and the quadratic term `I(horsepower^2)`, and R automatically includes an intercept term. [We can omit an intercept term by using a "-1", say using `lm(y ~ -1 + x)`, but we rarely want to do this.]

Plot the data, the fitted straight line, and the fitted quadratic:

```
plot(mpg ~ horsepower, data=Auto)
abline(fit2, col="red")
curve(56.90 - 0.4662*x + 0.0012*x^2, col="blue", add=TRUE)
```

Repeat the previous residual plot but for the quadratic model to see how/if the quadratic term has improved the behaviour of the residuals:

```
plot(rstandard(fit3) ~ fitted(fit3))
```

Forbes data

Install the package `MASS`, load it, and look at the `forbes` dataset:

```
install.packages("MASS")
library(MASS)
forbes
?forbes
```

The 19th-century physicist James Forbes had a theory that suggested that log(pressure) (i.e. the log of `pres`) was linearly related to temperature (`bp`).

Try a linear model fit – before plotting the fitted line, does a linear model look like a good idea?

```
plot(log(pres) ~ bp, data=forbes)
fit4 <- lm(log(pres) ~ bp, data=forbes)
fit4
abline(fit4, col="red")
```

Once the fitted line is included the problem should be apparent. The following plot makes the problem point (data point #12) even more obvious:

```
plot(rstandard(fit4) ~ fitted(fit4))
```

Olympics data

The following will read-in data (from Rogers and Girolami, 2012) of winning times in the 100 metres at Olympics in 2008 and earlier. Try fitting a linear model to each dataset and hence predicting the winning times for 2012 and 2016.

You should be able to cut-and-paste this into R:

```
femaleyear <- c(1928, 1932, 1936, 1948, 1952, 1956, 1960, 1964, 1968,
                1972, 1976, 1980, 1984, 1988, 1992, 1996, 2000, 2004, 2008)

female100 <- c(12.2, 11.9, 11.5, 11.9, 11.5, 11.5, 11, 11.4, 11, 11.07,
```

```
              11.08, 11.06, 10.97, 10.54, 10.82, 10.94, 11.12, 10.93, 10.78)

maleyear <- c(1896, 1900, 1904, 1906, 1908, 1912, 1920, 1924, 1928,
              1932, 1936, 1948, 1952, 1956, 1960, 1964, 1968, 1972,
              1976, 1980, 1984, 1988, 1992, 1996, 2000, 2004, 2008)

male100 <- c(12, 11, 11, 11.2, 10.8, 10.8, 10.8, 10.6, 10.8,
             10.3, 10.3, 10.3, 10.4, 10.5, 10.2, 10, 9.95, 10.14,
             10.06, 10.25, 9.99, 9.92, 9.96, 9.84, 9.87, 9.85, 9.69)

plot(female100 ~ femaleyear)
plot(male100 ~ maleyear)
```

Python

In each case you will first need to read-in the data. Download `faithful.txt`, `Auto.txt`, `forbes.txt`, `olympicsf.txt`, `olympicsm.txt` from `http://www.stats.ox.ac.uk/~laws/data/`

```
import os
path = [customise for your setting]

                                    os.chdir(os.path.join(path, folder))

import pandas as pd
import numpy as np
import statsmodels.formula.api as smf
import matplotlib.pyplot as plt


#### Old Faithful data ####
faithful = pd.read_table('faithful.txt', delimiter = ' ')
faithful.head()
y = faithful['waiting']
x = faithful['eruptions']

plt.scatter(x, y)
plt.xlabel("No. of Eruptions")
plt.ylabel("Waiting Time (min)")
plt.show()

# find coefficients b of linear model
X = np.column_stack((np.ones(len(x)), x))
b = np.linalg.lstsq(X, y)[0]
print(b[0], b[1])

fit1 = b[0] + b[1]*x

# alternative using numpy.polyfit
slope, intercept = np.polyfit(x, y, 1)
print(slope, intercept)

fit1a = intercept + slope*x

# alternative using statsmodels.formula.api.ols
lin_reg = smf.ols(formula = "waiting ~ eruptions", data = faithful).fit()
print(lin_reg.summary())

# extract standardised/studentised residual
out = lin_reg.outlier_test()
```

```
y_resid = out['student_resid']

# plot data and simple linear regression fit
plt.scatter(x, y)
plt.xlabel("No. of Eruptions")
plt.ylabel("Waiting Time (min)")
plt.plot(x, fit1a, color = 'green')
plt.plot(x, fit1, color = 'red')
plt.show()

# calculate residuals
res1 = y - fit1

# residual plot
plt.scatter(fit1, res1)
plt.xlabel("Fitted Waiting Time (min)")
plt.ylabel("Residual")
plt.show()

# same plot with different horizontal scale
plt.scatter(x, res1)
plt.xlabel("No. of Eruptions")
plt.ylabel("Residual")
plt.show()

# similar plot using the standardised/studentised residuals
plt.scatter(fit1, y_resid)
plt.axhline(y = 0, color = 'black', linestyle = '--')
plt.xlabel('Fitted Waiting Time (min)')
plt.ylabel('Studentised Residuals')
plt.show()

#### Auto data ####
Auto = pd.read_table("auto.txt", delimiter = ' ')
Auto.head()
y = Auto['mpg']
x = Auto['horsepower']

plt.scatter(x, y)
plt.xlabel("Horsepower")
plt.ylabel("mpg")
plt.show()

# find coefficients of linear model
X = np.column_stack((np.ones(len(x)), x))
b = np.linalg.lstsq(X, y)[0]
fit2 = b[0] + b[1]*x

# alternative using numpy.polyfit
slope, intercept = np.polyfit(x, y, 1)
fit2a = intercept + slope*x

# plot data and linear regression fit
plt.scatter(x, y)
plt.xlabel("Horsepower")
plt.ylabel("mpg")
plt.plot(x, fit2a, color = 'green')
plt.plot(x, fit2, color = 'red')
plt.show()
```

```
# calculate and plot studentised residuals
lin_reg = smf.ols(formula = "mpg ~ horsepower", data = Auto).fit()
# print(lin_reg.summary())
out2 = lin_reg.outlier_test()
y_resid = out2['student_resid']

plt.scatter(fit2, y_resid)
plt.axhline(y = 0, color = 'black', linestyle = '--')
plt.xlabel("Fitted mpg")
plt.ylabel("Studentised Residuals")
plt.show()

# curved nature of residual plot above suggests linear fit is not appropriate!

# find coefficients of quadratic model
X = np.column_stack((np.ones(len(x)), x, x**2))
b = np.linalg.lstsq(X, y)[0]
fit3 = b[0] + b[1]*x + b[2]*(x**2)
# NB: x**2 = pow(x,2) = np.square(x)

# alternative with numpy.polyfit
coefs = np.polyfit(x, y, 2)
poly = np.poly1d(coefs)
fit3a = poly(x)

# calculate residuals
res3 = y - fit3

# plot data, fitted straight line, and fitted quadratic
plt.scatter(x, y)
plt.plot(x, fit2, color = 'red', label='linear') # straight line fit
plt.plot(x[np.argsort(x)], fit3a[np.argsort(x)], color='green') # quadratic fit
plt.plot(x[np.argsort(x)], fit3[np.argsort(x)], color='black',  label='quadratic') # quadratic fit
plt.xlabel("Horsepower")
plt.ylabel("mpg")
plt.legend()
plt.show()

# repeat previous residual plots with quadratic fit:
# has it improved the behaviour of the residuals? If so, how?

# residual plot
plt.scatter(fit3, res3)
plt.xlabel("Fitted mpg")
plt.ylabel("Residual")
plt.show()

# extract standardised/studentised residual
lin_reg = smf.ols(formula = "mpg ~ horsepower + I(horsepower**2)", data=Auto).fit()
print(lin_reg.summary())
out3 = lin_reg.outlier_test()
y_resid = out3['student_resid']

# studentised residual plot
plt.scatter(fit3, y_resid)
plt.axhline(y = 0, color = 'black', linestyle = '--')
plt.xlabel("Fitted mpg")
plt.ylabel("Studentised Residuals")
```

```
plt.show()

#### Forbes data ####
forbes = pd.read_table("forbes.txt", delimiter = ' ')
forbes.head()
y = forbes['pres'] # pressure
x = forbes['bp']   # temperature

plt.scatter(x, np.log(y))
plt.xlabel("Temp. (bp)")
plt.ylabel("log(Pressure)")
plt.show()

X = np.column_stack((np.ones(len(x)), x))
b = np.linalg.lstsq(X, np.log(y))[0]
fit4 = b[0] + b[1]*x

# alternative using numpy.polyfit
slope, intercept = np.polyfit(x, np.log(y), 1)
fit4a = intercept + slope*x

plt.scatter(x, np.log(y))
plt.plot(x, fit4a, color = 'green')
plt.plot(x, fit4, color = 'red')
plt.xlabel("Temperature (bp)")
plt.ylabel("log(Pressure)")
plt.show()

# calculate and visualise the studentised residuals
lin_reg = smf.ols(formula = "I(np.log(pres)) ~ bp", data=forbes).fit()
# print(lin_reg.summary())
out4 = lin_reg.outlier_test()
y_resid = out4['student_resid']

plt.scatter(fit4, y_resid)
plt.axhline(y = 0, color = 'black', linestyle = '--')
plt.xlabel("Fitted log(Pressure)")
plt.ylabel("Studentised Residuals")
plt.show()

#### Olympics data ####
olympicsf = pd.read_table("olympicsf.txt", delimiter = ' ')
olympicsm = pd.read_table("olympicsm.txt", delimiter = ' ')

y_f = olympicsf['female100']
x_f = olympicsf['year'] # women's athletics debuted in Summer 1928

y_m = olympicsm['male100']
x_m = olympicsm['year']

X_f = np.column_stack((np.ones(len(x_f)), x_f))
b_f = np.linalg.lstsq(X_f, y_f)[0]

X_m = np.column_stack((np.ones(len(x_m)), x_m))
b_m = np.linalg.lstsq(X_m, y_m)[0]

fit_f = b_f[0] + b_f[1]*x_f
fit_m = b_m[0] + b_m[1]*x_m
```

```
plt.scatter(x_f, y_f)
plt.scatter(x_m, y_m)
plt.plot(x_f, fit_f, label = "women")
plt.plot(x_m, fit_m, label = "men")
plt.xlabel("Year")
plt.ylabel("100m Winning Time (s)")
plt.legend()
plt.show()
```