Contents

Part I

Part: Regression models

1 Linear regression

Regression models are the workhorse of data science. They are the most well described, practical and theoretically understood models in statistics. A data scientist well versed in regression models will be able to solve an incredible array of problems.

Benefits of regression models are the **simplicity**, **parsimony** and **intrepretability**. This is unlike machine learning algorithms, which often sacrifice interpretability for improved prediction performance or automation.

1.1 A famous motivating example

In the Victorian era, Sir Francis Galton showed that

> 'when dealing with the transmission of stature from parents to children, the average height of the two parents, … is all we need care to know about them' (1886)

A still relevant example:

Predicting height: the Victorian approach beats modern genomics

Predicting human height by Victorian and genomic

Research questions:

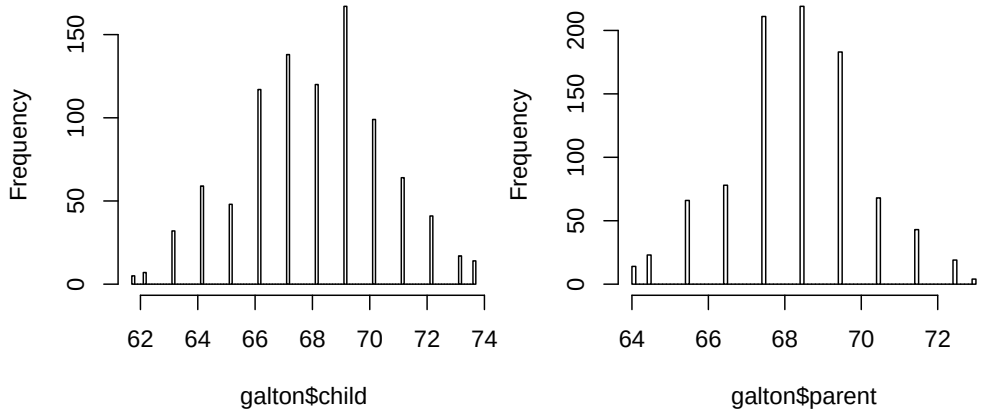
- To use the parents' heights to predict childrens' heights.
- To try to find a parsimonious, easily described mean relationship between parent and children's heights.
- To investigate the variation in childrens' heights that appears unrelated to parents' heights (residual variation).
- To figure out how/whether and what assumptions are needed to generalize findings beyond the data in question.

- Why do children of very tall parents tend to be tall, but a little shorter than their parents and why children of very short parents tend to be short, but a little taller than their parents? (This is a famous question called **Regression to the mean**.)

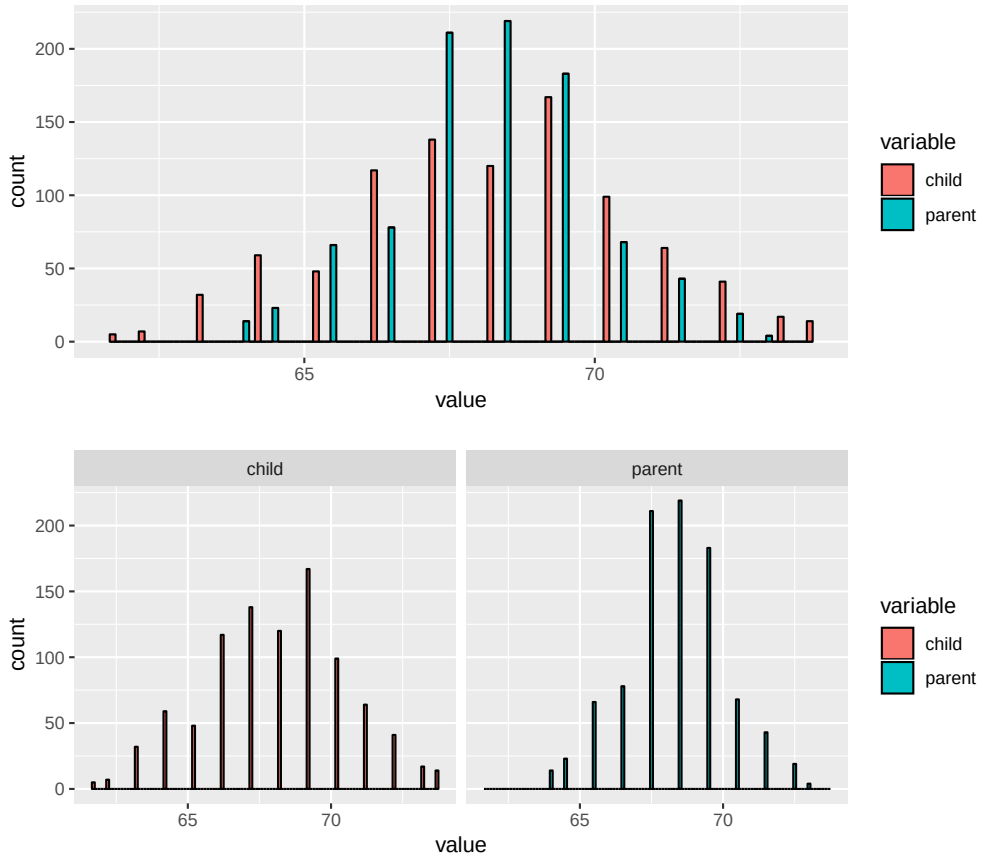
Galton's Data

- Data used by Francis Galton in 1885.
- Galton was a statistician who invented the term and concepts of regression and correlation (and was the cousin of Charles Darwin).

```
library(UsingR); data(galton)
hist(galton$child,breaks=100,main="")
hist(galton$parent,breaks=100,main="")
```

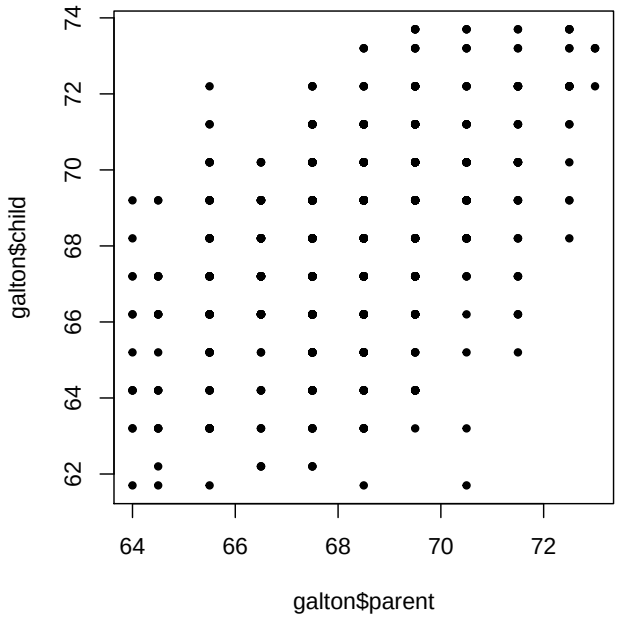


```
library(ggplot2); library(reshape)
long <- melt(galton)
head(long)
#>   variable value
#> 1    child  61.7
#> 2    child  61.7
#> 3    child  61.7
#> 4    child  61.7
#> 5    child  61.7
#> 6    child  62.2
g <- ggplot(long, aes(x=value, fill=variable))
g1 <- g + geom_histogram(colour="black", binwidth = .1)
g2 <- g1 + facet_grid(.~variable)
g1; g2
```

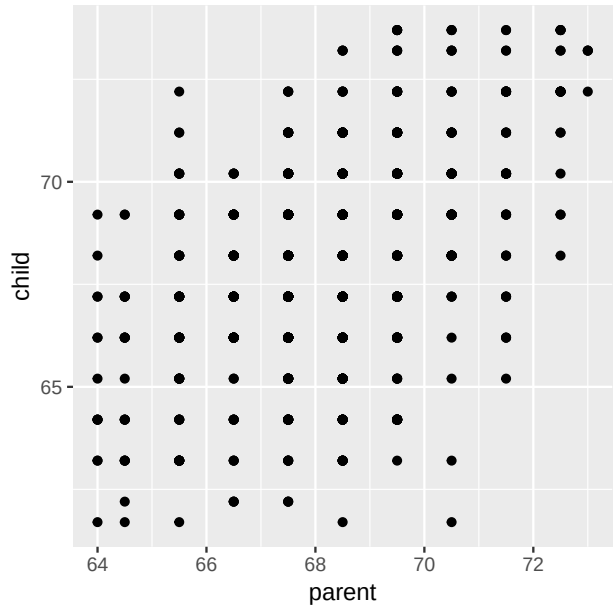


Comparing childrens' heights and their parents'heights

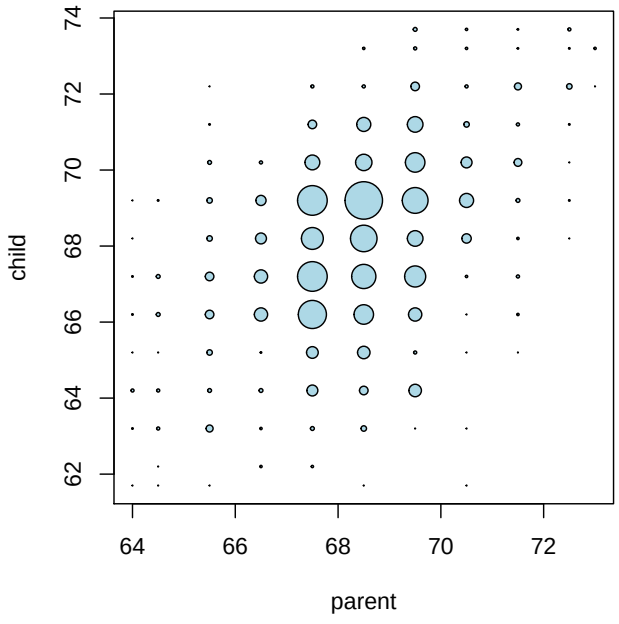
```
plot(galton$parent,galton$child, pch=20)
```



```
ggplot(galton, aes(x = parent, y = child)) + geom_point()
```



Size of point represents number of points at that (X, Y) combination



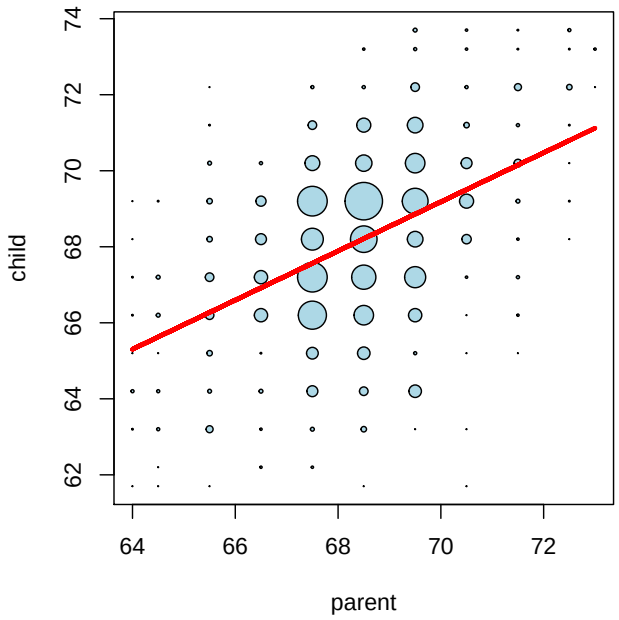
Regression through the origin

- Suppose that X_i are the parents' heights.

- Consider picking the slope β that minimizes

$$\sum_{i=1}^{n}(Y_i - X_i\beta)^2$$

Visualizing the best fit line



1.2 Least squares estimation of regression lines

Ordinary least squares (OLS) is the workhorse of statistics. It gives a way of taking complicated outcomes and explaining behavior using linearity.

The simplest application of OLS is fitting a line through some data.

1.2.1 Fitting the best line

- Let Y_i be the i^{th} child's height and X_i be the i^{th} (average over the pair of) parents' heights.
- Consider finding the best line
  - Child's Height = β_0 + Parent's Height β_1
- Use least squares

$$\sum_{i=1}^{n} Y_i - (\beta_0 + \beta_1 X_i)^2$$

- Minimize LS.

1.2.1.1 Linear least squares

- The least squares model fit to the line $Y = \beta_0 + \beta_1 X$ through the data pairs (X_i, Y_i) with Y_i as the outcome obtains the line $Y = \hat{\beta}_0 + \hat{\beta}_1 X$ where

$$\hat{\beta}_1 = Cor(Y, X)\frac{Sd(Y)}{Sd(X)} \quad \hat{\beta}_0 = \bar{Y} - \hat{\beta}_1 \bar{X}$$

- The line passes through the point $(\bar{X}, \bar{Y})$

Revisiting Galton's data

```
y <- galton$child
x <- galton$parent
beta1 <- cor(y, x) *  sd(y) / sd(x)
beta0 <- mean(y) - beta1 * mean(x)
rbind(c(beta0, beta1), coef(lm(y ~ x)))
#>      (Intercept)         x
#> [1,]    23.94153 0.6462906
#> [2,]    23.94153 0.6462906
cor(y, x)
#> [1] 0.4587624
```

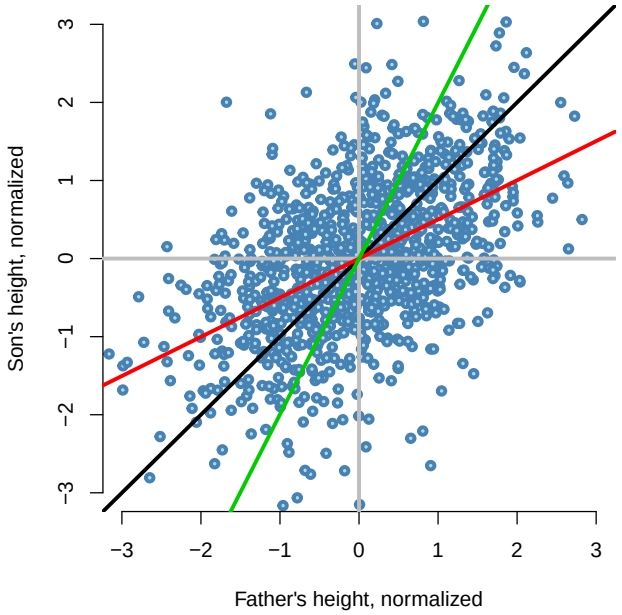
1.2.1.2 Regression to the mean

Why is it that the children of tall (short) parents tend to be tall (short), but not as tall (short) as their parents?

We can try this with anything that is **measured with error**. Why do the best performers on hard exams always do a little worse on the next hard exam?

These phenomena are all examples of so-called **regression to the mean** that was invented by Francis Galton in the paper *Regression towards mediocrity in hereditary stature* The Journal of the Anthropological Institute of Great Britain and Ireland , Vol. 15, (1886). The idea served as a foundation for the discovery of linear regression.

```
abline(h = 0, col=8); abline(v = 0, col=8) # reference lines for no relathionship
abline(0, 1) # if there were perfect correlation
abline(0, rho, col=2) # father predicts son
abline(0, 1 / rho, col=3) # son predicts father, son on vertical axis
```



- If you had to predict a son's normalized height, it would be $Cor(Y, X) * X_i$
- If you had to predict a father's normalized height, it would be $Cor(Y, X) * Y_i$
- Multiplication by this correlation shrinks toward 0 (regression toward the mean)
- If the correlation is 1 there is no regression to the mean (if father's height perfectly determine's child's height and vice versa)

1.3 Statistical linear regression

Up to this point, we've only considered estimation. Estimation is useful, but we also need to know how to extend our estimates to a population. This is the process of **statistical inference**.

Our approach to statistical inference will be through a **statistical model**. At the bare minimum, we need a few distributional assumptions on the errors. However, we'll focus on full model assumptions under Gaussianity.

1.3.1 Basic regression model with additive Gaussian errors

- Least squares is an estimation tool, how do we do inference?
- Consider developing a probabilistic model for linear regression

$$Y_i = \beta_0 + \beta_1 X_i + \epsilon_i$$

- Here the ϵ_i are assumed iid $N(0, \sigma^2)$.
- Note, $E[Y_i \mid X_i = x_i] = \mu_i = \beta_0 + \beta_1 x_i$
- Note, $Var(Y_i \mid X_i = x_i) = \sigma^2$.
- Likelihood equivalent model specification is that the Y_i are independent $N(\mu_i, \sigma^2)$.
- The **least squares estimate** for $\mu_i = \beta_0 + \beta_1 x_i$ **is exactly the maximimum likelihood estimate**.

Recap

- Model $Y_i = \mu_i + \epsilon_i = \beta_0 + \beta_1 X_i + \epsilon_i$ where ϵ_i are iid $N(0, \sigma^2)$
- $E[Y \mid X = x] = \beta_0 + \beta_1 x$
- $Var(Y \mid X = x) = \sigma^2$
- ML estimates of β_0 and β_1 are the LS estimates

$$\hat{\beta}_1 = Cor(Y, X)\frac{Sd(Y)}{Sd(X)} \quad \hat{\beta}_0 = \bar{Y} - \hat{\beta}_1 \bar{X}$$

1.3.2 Interpreting regression coefficients

- β_0 is the expected value of the response when the predictor X is 0
  - Note, this isn't always of interest, for example when $X = 0$ is impossible or far outside of the range of data.
  - Consider that

$$\begin{aligned} Y_i = \beta_0 + \beta_1 X_i + \epsilon_i &= \beta_0 + a\beta_1 + \beta_1(X_i - a) + \epsilon_i \\ &= \tilde{\beta}_0 + \beta_1(X_i - a) + \epsilon_i \end{aligned}$$

    So, shifting X values by a value a changes the intercept, but not the slope.
  - Often a is set to $\bar{X}$ so that the intercept is interpretted as ???

- β_1 is the expected change in response for a 1 unit change in the predictor
  - Consider the impact of changing the units of X.

$$\begin{aligned} Y_i = \beta_0 + \beta_1 X_i + \epsilon_i &= \beta_0 + \frac{\beta_1}{a}(X_i a) + \epsilon_i \\ &= \beta_0 + \tilde{\beta}_1(X_i a) + \epsilon_i \end{aligned}$$

    Therefore, multiplication of X by a factor a results in dividing the coefficient by a factor of a.
  - Example: X is height in m and Y is weight in kg. Then β_1 is kg/m. Converting X to cm implies multiplying X by $100cm/m$. To get β_1 in the right units, we have to divide by $100cm/m$ to get it to have the right units.

1.3.3 Using regression for prediction

- If we would like to guess the outcome at a particular value of the predictor, say X, the regression model guesses

$$\hat{\beta}_0 + \hat{\beta}_1 X$$

- Note that at the observed value of Xs, we obtain the predictions

$$\hat{\mu}_i = \hat{Y}_i = \hat{\beta}_0 + \hat{\beta}_1 X_i$$

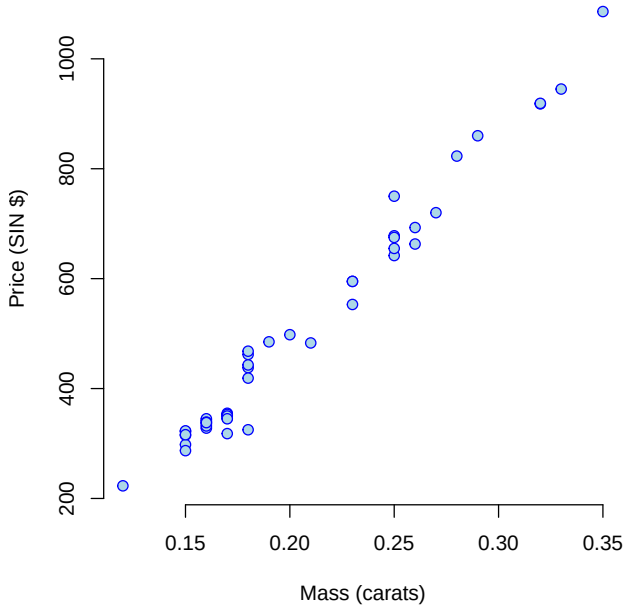
- Remember that least squares minimizes $\sum_{i=1}^{n}(Y_i - \mu_i)^2$ for μ_i expressed as points on a line

---

Example: `diamond` data set from `UsingR`

Data is diamond prices (Signapore dollars) and diamond weight in carats (standard measure of diamond mass, 0.2 g).

Plotting the data: use `plot(...) or ggplot(...)`



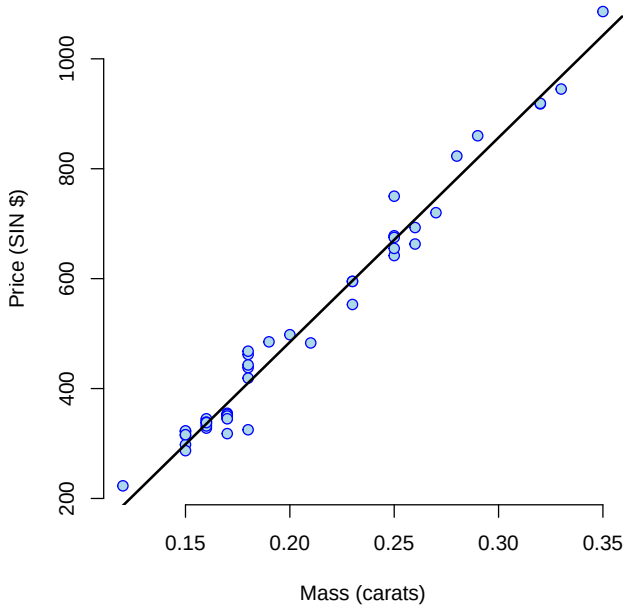
Fitting the linear regression model

```
fit <- lm(price ~ carat, data = diamond)
coef(fit)
#> (Intercept)       carat
#>   -259.6259   3721.0249
```

- We estimate an expected 3721.02 (SIN) dollar increase in price for ...
- The intercept -259.63 is the ...

Plotting the fitted regression line and data

```
abline(lm(price ~ carat, data = diamond))
```



- Getting a more interpretable intercept

```
fit2 <- lm(price ~ I(carat - mean(carat)), data = diamond)
coef(fit2)
#>             (Intercept) I(carat - mean(carat))
#>                500.0833              3721.0249
```

Thus $500.1 is the expected price for the average sized diamond of the data (0.2041667 carats).

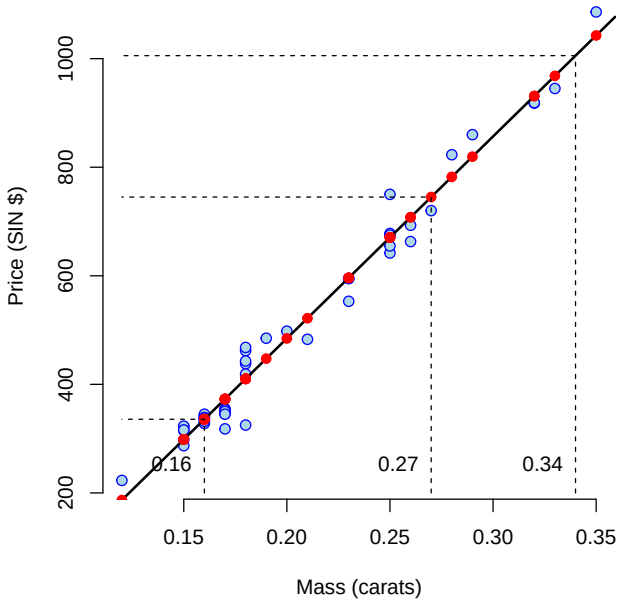
- Changing scale
  - A one carat increase in a diamond is pretty big, what about changing units to 1/10th of a carat?
  - We expect a 372.102 (SIN) dollar change in price for every 1/10th of a carat increase in mass of diamond.
  - Showing that it's the same if we rescale the Xs and refit

```
fit3 <- lm(price ~ I(carat * 10), data = diamond)
coef(fit3)
#>   (Intercept) I(carat * 10)
#>     -259.6259      372.1025
```

Predicting the price of a diamond

```
newx <- c(0.16, 0.27, 0.34)
coef(fit)[1] + coef(fit)[2] * newx
#> [1]  335.7381  745.0508 1005.5225
predict(fit, newdata = data.frame(carat = newx))
#>         1         2         3
#>  335.7381  745.0508 1005.5225
```

Predicted values at the observed Xs (red) and at the new Xs (lines)



1.4 Residuals

Residuals represent variation left unexplained by our model.

We emphasize the difference between residuals (observable errors from the estimated coefficients) and **errors** (unobservable true errors from the known coefficients).
In a sense, the residuals are estimates of the errors.

Recap

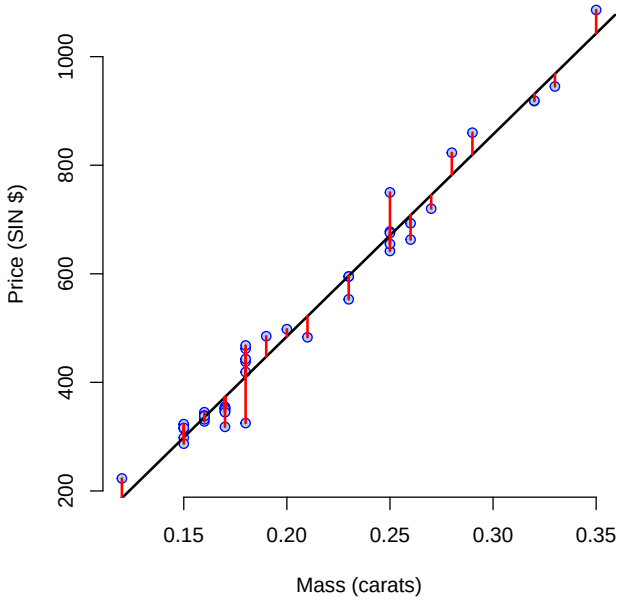
- Model $Y_i = \beta_0 + \beta_1 X_i + \epsilon_i$ where $\epsilon_i \sim N(0, \sigma^2)$.
- Observed outcome i is Y_i at predictor value X_i
- Predicted outcome i is $\hat{Y}_i$ at predictor value X_i given by $\hat{Y}_i = \hat{\beta}_0 + \hat{\beta}_1 X_i$
- Residual $e_i = Y_i - \hat{Y}_i$ (he vertical distance between the observed data point and the regression line)
- Least squares minimizes $\sum_{i=1}^{n} e_i^2$
- The e_i can be thought of as estimates of the ϵ_i.

Properties of the residuals

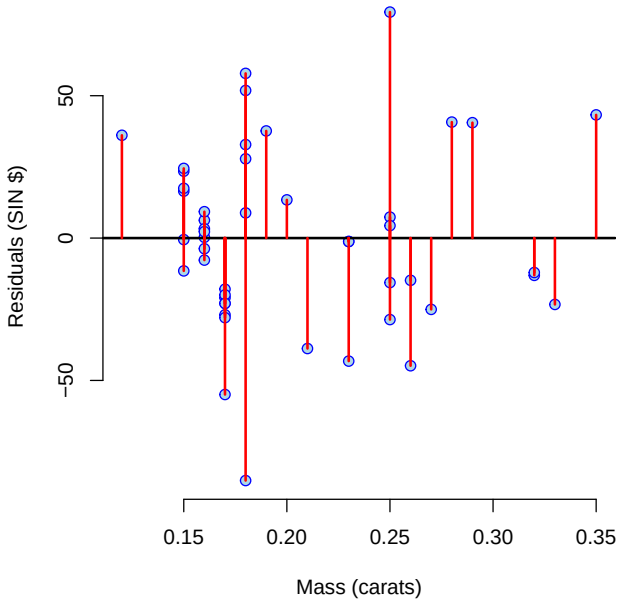
- $E[e_i] = 0$.
- If an intercept is included, $\sum_{i=1}^{n} e_i = 0$
- If a regressor variable, X_i, is included in the model $\sum_{i=1}^{n} e_i X_i = 0$.
- One differentiates *residual variation* (variation after removing the predictor) from *systematic variation* (variation explained by the regression model).
- *Residual plots* highlight poor model fit.

Diamond example

Residuals are the signed length of the red lines

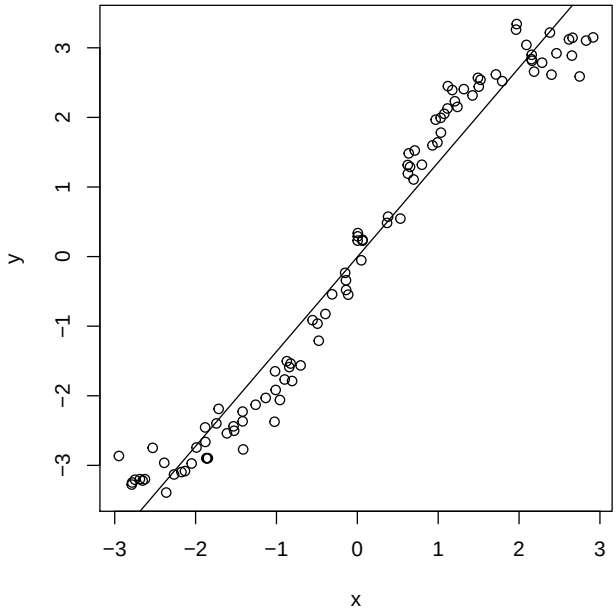


1.4.1 Residuals versus X

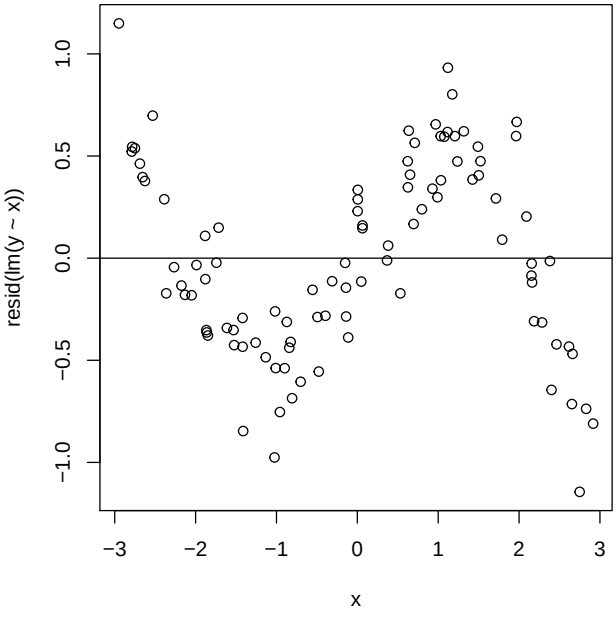


1.4.1.1 Non-linear data

```
x <- runif(100, -3, 3); y <- x + sin(x) + rnorm(100, sd = .2);
plot(x, y); abline(lm(y ~ x))
```

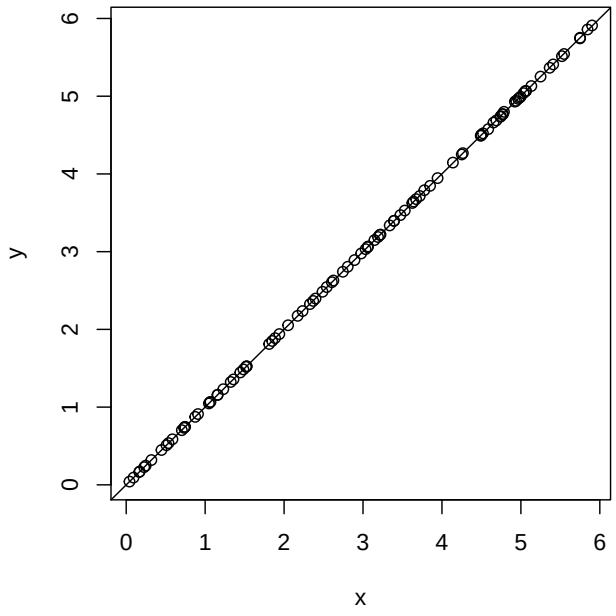



```
plot(x, resid(lm(y ~ x)));
abline(h = 0)
```

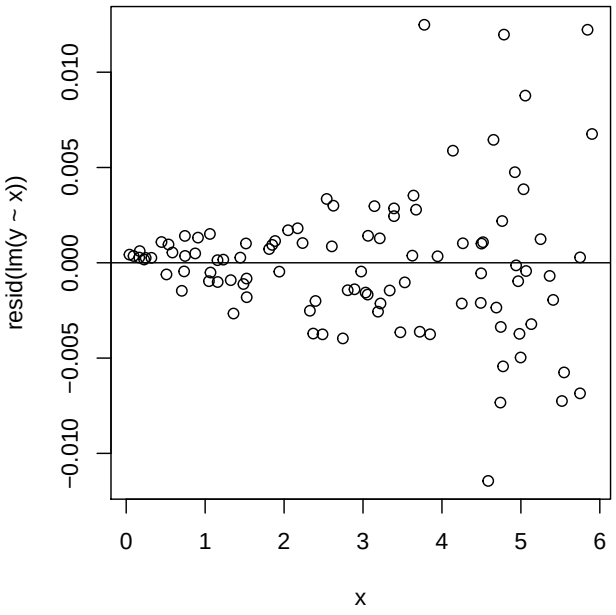


1.4.2 Heteroskedasticity

```
x <- runif(100, 0, 6); y <- x + rnorm(100,  mean = 0, sd = .001 * x);
plot(x, y); abline(lm(y ~ x))
```




```
plot(x, resid(lm(y ~ x)));
abline(h = 0)
```



1.4.3 Estimating residual variation

- Model $Y_i = \beta_0 + \beta_1 X_i + \epsilon_i$ where $\epsilon_i \sim N(0, \sigma^2)$.
- The ML estimate of σ^2 is $\frac{1}{n}\sum_{i=1}^n e_i^2$, the average squared residual.
- Most people use

$$\hat{\sigma}^2 = \frac{1}{n-2}\sum_{i=1}^n e_i^2.$$

- The $n-2$ instead of n is so that $E[\hat{\sigma}^2] = \sigma^2$

Diamond example

```
y <- diamond$price; x <- diamond$carat; n <- length(y)
fit <- lm(y ~ x)
summary(fit)$sigma
#> [1] 31.84052
sqrt(sum(resid(fit)^2) / (n - 2))
#> [1] 31.84052
```

1.4.4 Summarizing variation

$$\sum_{i=1}^n (Y_i - \bar{Y})^2 = \sum_{i=1}^n (Y_i - \hat{Y}_i)^2 + \sum_{i=1}^n (\hat{Y}_i - \bar{Y})^2$$

Or

Total Variation = Residual Variation + Regression Variation

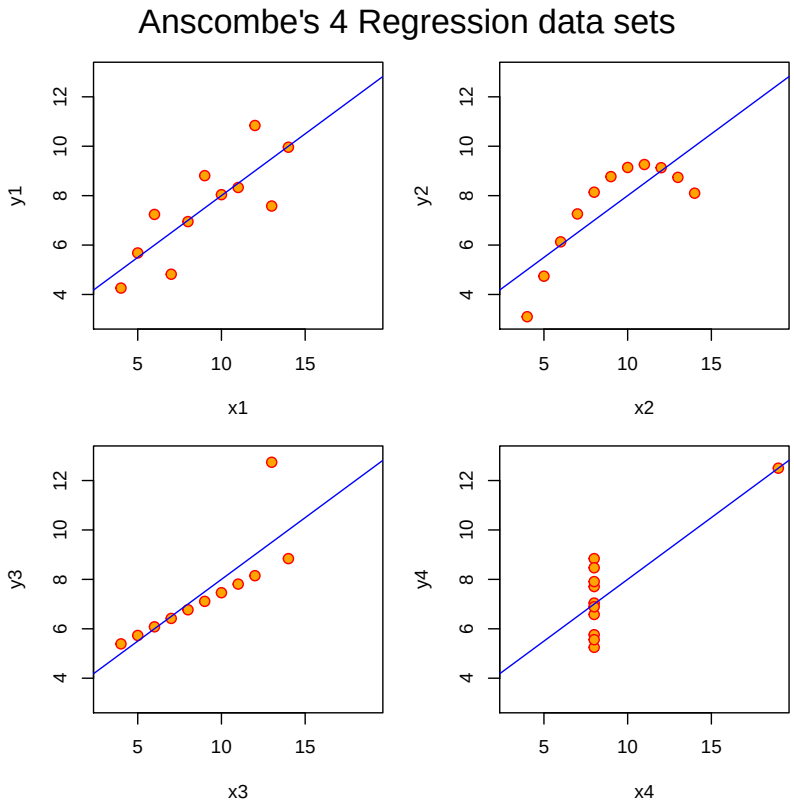
Define the percent of total varation described by the model as

$$R^2 = \frac{\sum_{i=1}^n (\hat{Y}_i - \bar{Y})^2}{\sum_{i=1}^n (Y_i - \bar{Y})^2} = 1 - \frac{\sum_{i=1}^n (Y_i - \hat{Y}_i)^2}{\sum_{i=1}^n (Y_i - \bar{Y})^2}$$

Some facts about R^2

- R^2 is the percentage of variation explained by the regression model.
- $0 \leq R^2 \leq 1$
- R^2 is the sample correlation squared (in simple regression).
- R^2 can be a misleading summary of model fit.
  - Deleting data can inflate R^2.
  - Adding terms to a regression model always increases R^2.
- Do `example(anscombe)` to see the following data.
  - Basically same mean and variance of X and Y.
  - Identical correlations (hence same R^2).
  - Same linear regression relationship.

```
data(anscombe); example(anscombe)
```



1.5 Inference in regression

Inference is the process of drawing conclusions about a population using a sample.

We'll cover inference in regression where we make some Gaussian assumptions about the errors.

Recall the model

- Consider the model
$$Y_i = \beta_0 + \beta_1 X_i + \epsilon_i$$
- $\epsilon \sim N(0, \sigma^2)$.
- *We assume that you've seen confidence intervals and hypothesis tests before!*
- We assume that the true model is known.
- $\hat{\beta}_0 = \bar{Y} - \hat{\beta}_1 \bar{X}$
- $\hat{\beta}_1 = Cor(Y, X)\frac{Sd(Y)}{Sd(X)}$.

1.5.1 Inference on parameters

It is easy to obtain the standard errors (conditioned on X)

- $\sigma_{\hat{\beta}_1} = var(\hat{\beta}_1) = \frac{\sigma^2}{\sum_{i=1}^{n}(X_i - \bar{X})^2}$
- $\sigma_{\hat{\beta}_0} = var(\hat{\beta}_0) = \left(\frac{1}{n} + \frac{\bar{X}^2}{\sum_{i=1}^{n}(X_i - \bar{X})^2}\right)\sigma^2$

In practice σ is replaced by its estimate.

It can be demonstrated that under iid Gaussian errors

$$\frac{\hat{\beta}_j - \beta_j}{\hat{\sigma}_{\hat{\beta}_j}}$$

follows a t distribution with $n-2$ degrees of freedom and a normal distribution for large n.

This result above can be used to create **confidence intervals** and perform **hypothesis tests**.

Example diamond data set

```
y <- diamond$price; x <- diamond$carat; n <- length(y)
fit <- lm(y ~ x);

summary(fit)$coefficients
#>              Estimate Std. Error   t value     Pr(>|t|)
#> (Intercept) -259.6259   17.31886 -14.99094 2.523271e-19
#> x           3721.0249   81.78588  45.49715 6.751260e-40
```

1.5.2 Prediction of outcomes

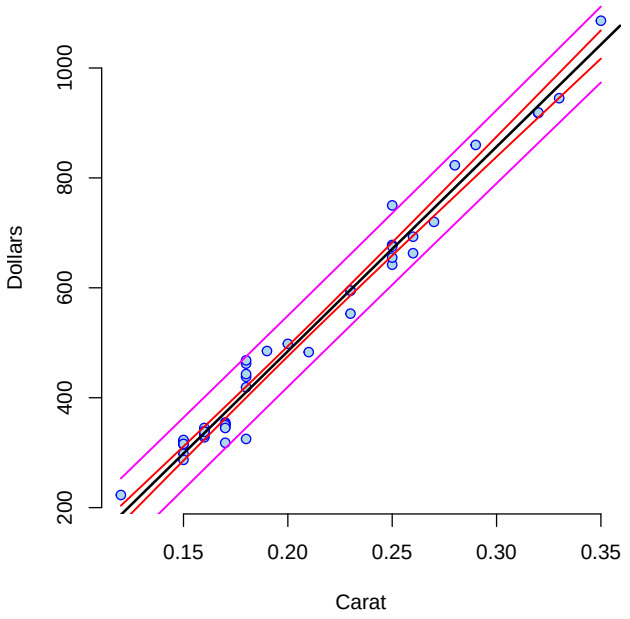
- Consider predicting Y at a value of X
- The obvious estimate for prediction at point x_0 is

$$\hat{\beta}_0 + \hat{\beta}_1 x_0$$

- We want to create a **prediction interval**, so we need a standard error
- There's a distinction between intervals for the regression line at point x_0 and the prediction of what a y would be at point x_0.
  - *se* of line at x_0: $\hat{\sigma}\sqrt{\frac{1}{n} + \frac{(x_0 - \bar{X})^2}{\sum_{i=1}^{n}(X_i - \bar{X})^2}}$
  - *se* of prediction interval at x_0, $\hat{\sigma}\sqrt{1 + \frac{1}{n} + \frac{(x_0 - \bar{X})^2}{\sum_{i=1}^{n}(X_i - \bar{X})^2}}$

Plotting the prediction intervals

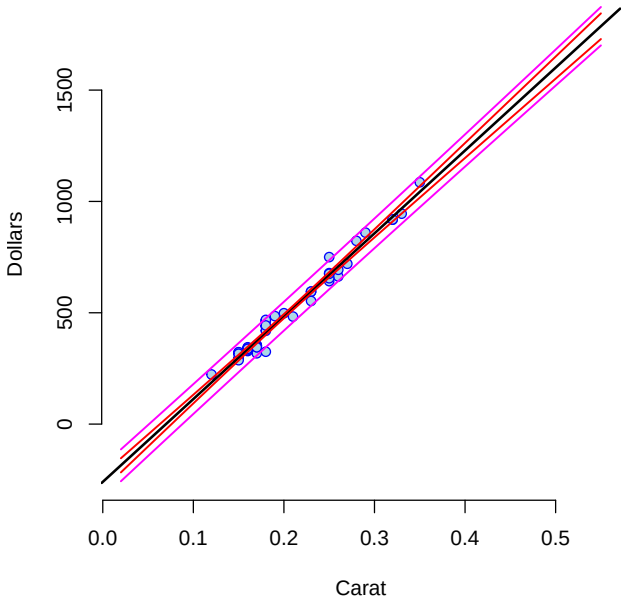
```
xVals <- seq(min(x), max(x), by = .01)
newdata <- data.frame(x = xVals)

p1 <- predict(fit, newdata, interval = ("confidence"))
p2 <- predict(fit, newdata, interval = ("prediction"))

plot(x, y)
abline(fit)
lines(xVals, p1[,2]); lines(xVals, p1[,3])
lines(xVals, p2[,2]); lines(xVals, p2[,3])
```



Discussion

- Both intervals have varying widths.
  - Least width at the mean of the Xs.
- We are quite confident in the regression line, so that interval is very narrow.
  - If we knew β_0 and β_1 this interval would have zero width.
- The prediction interval must incorporate the variabilibity in the data around the line.
  - Even if we knew β_0 and β_1 this interval would still have width.

What about extrapolation?



1.6 Multiple regression

We now extend linear regression so that our models can contain more variables.

A natural first approach is to assume additive effects, basically extending our line to a plane, or generalized version of a plane as we add more variables.

Multiple/multivariable regression represents one of the most widely used and successful methods in statistics.

One use of multivariable regression is for **adjustment**, that is trying to look at the relationship of a predictor and a response, while having, at some level, **accounted for** other variables (which can distort, or confound, the relationship between two others).

Another use is for **prediction**. To get a good prediction model you may need to incoporate lots of regressors.

- How do we pick which predictors to include?
- Problem of overfitting

1.6.1 The linear model

- The general linear model extends simple linear regression by adding terms linearly into the model.

$$Y_i = \beta_1 X_{1i} + \beta_2 X_{2i} + \ldots + \beta_p X_{pi} + \epsilon_i = \sum_{k=1}^{p} X_{ik}\beta_j + \epsilon_i$$

- Here $X_{1i} = 1$ typically, so that an intercept is included.
- Least squares (and hence ML estimates under iid Gaussianity of the errors) minimizes

$$\sum_{i=1}^{n} \left(Y_i - \sum_{k=1}^{p} X_{ki}\beta_j \right)^2$$

- Note, the important **linearity** is linearity in the coefficients. Thus

$$Y_i = \beta_1 X_{1i}^2 + \beta_2 X_{2i}^2 + \ldots + \beta_p X_{pi}^2 + \epsilon_i$$

  is still a linear model. (We've just squared the elements of the predictor variables.)

1.6.1.1 LS estimates

The least squares estimate for a coefficient of a multivariate regression model is exactly the regression (through the origin) estimate having removed the linear relationships with the other regressors from both the regressor and outcome.

In this sense, multivariate regression "**adjusts**" a coefficient for the linear impact of the other variables.

Think of it as follows. If we want an adjusted relationship between y and x, keep taking residuals over confounders and do regression through the origin.

Interpretation of the coefficient

$$E[Y|X_1 = x_1, \ldots, X_p = x_p] = \sum_{k=1}^{p} x_k \beta_k$$

So that

$$E[Y|X_1 = x_1 + 1, \ldots, X_p = x_p] - E[Y|X_1 = x_1, \ldots, X_p = x_p]$$
$$= (x_1 + 1)\beta_1 + \sum_{k=2}^{p} x_k \beta_k - \sum_{k=1}^{p} x_k \beta_k = \beta_1$$

So that the interpretation of a multivariate regression coefficient is the expected change in the response per unit change in the regressor, holding all of the other regressors fixed.

Fitted values, residuals and residual variation

All of our SLR quantities can be extended to linear models

- Model $Y_i = \sum_{k=1}^{p} X_{ik}\beta_k + \epsilon_i$ where $\epsilon_i \sim N(0, \sigma^2)$
- Fitted responses $\hat{Y}_i = \sum_{k=1}^{p} X_{ik}\hat{\beta}_k$
- Residuals $e_i = Y_i - \hat{Y}_i$
- Variance estimate $\hat{\sigma}^2 = \frac{1}{n-p}\sum_{i=1}^{n} e_i^2$
- Get predicted responses at new values, $x_1, \ldots, x_p$, plugging them into the model $\sum_{k=1}^{p} x_k \hat{\beta}_k$

Inference

- Coefficients have standard errors, $\hat{\sigma}_{\hat{\beta}_k}$, and $(\hat{\beta}_k - \beta_k)/\hat{\sigma}_{\hat{\beta}_k} \sim t_{n-p}$.
- Predicted responses have standard errors and we can calculate predicted and expected response intervals.

Linear models

- Linear models are the most important applied statistical and machine learning techniqe, *by far*.
- Some amazing things that you can accomplish with linear models
  - Flexibly fit complicated functions.
  - Fit factor variables as predictors.
  - Uncover complex multivariate relationships with the response.
  - Build accurate prediction models.

1.6.2 Residuals again

We can generalize this idea to the vertical distances between the observed data and the fitted surface in multivariable settings.

(to be cont'd)

1.7 Regression with R

1.7.1 Formula structure

Regression models can be used to study how the expected value of a *dependent variable* changes as *independent variables* change.

In R, regression formulas take this structure:

```
## Generic code
[response variable] ~ [indep. var. 1] + [indep. var. 2] + ...
```

Notice that a tilde, `~`, is used to separate the independent and dependent variables and that a plus sign, `+`, is used to join independent variables. This format mimics the statistical notation:

$$Y_i \sim X_1 + X_2 + X_3$$

There are some conventions that can be used in R formulas. Common ones include:

Convention	Meaning
`I()`	evaluate the formula inside `I()` before fitting (e.g., `I(x1 + x2)`)
`:`	fit the interaction between `x1` and `x2` variables
`*`	fit the main effects and interaction for both variables (e.g., `x1*x2` equals `x1 + x2 + x1:x2`)
`.`	include as independent variables all variables other than the response (e.g., `y ~ .`)
`1`	intercept (e.g., `y ~ 1` for an intercept-only model)
`-`	do not include a variable in the dataframe as an independent variables (e.g., `y ~ . - x1`); usually used in conjunction with `.` or `1`

1.7.2 Linear models

To fit a linear model, you can use the function `lm()`. This function is part of the `stats` package, which comes installed with base R. In this function, you can use the `data` option to specify the dataframe from which to get the vectors.

```
library(faraway); data("nepali")
library(dplyr)
nepali %>% head
#>       id sex   wt    ht mage lit died alive age
#> 1 120011   1 12.8  91.2   35   0    2     5  41
#> 2 120011   1 12.8  93.9   35   0    2     5  45
#> 3 120011   1 13.1  95.2   35   0    2     5  49
#> 4 120011   1 13.8  96.9   35   0    2     5  53
#> 5 120011   1   NA    NA   35   0    2     5  57
#> 6 120012   2 14.9 103.9   35   0    2     5  57
nepali <- nepali %>%
```

```
            select(id, sex, wt, ht, age) %>%
            mutate(id = factor(id), sex = factor(sex, levels = c(1, 2), labels = c("Male", "Female"))) %
            distinct(id, .keep_all = TRUE)
```

```
mod_a <- lm(wt ~ ht, data = nepali)
```

This previous call fits the model:

$$Y_i = \beta_0 + \beta_1 X_{1,i} + \epsilon_i$$

where:

- Y_i : weight of child i
- $X_{1,i}$: height of child i

If you run the `lm` function without saving it as an object, R will fit the regression and print out the function call and the estimated model coefficients:

```
lm(wt ~ ht, data = nepali)
#>
#> Call:
#> lm(formula = wt ~ ht, data = nepali)
#>
#> Coefficients:
#> (Intercept)           ht
#>     -8.6948       0.2351
```

However, to be able to use the model later for things like predictions and model assessments, you should save the output of the function as an R object:

```
mod_a <- lm(wt ~ ht, data = nepali)
```

This object has a special class, `lm`:

```
class(mod_a)
#> [1] "lm"
```

This class is a special type of list object. If you use `is.list` to check, you can confirm that this object is a list:

```
is.list(mod_a)
#> [1] TRUE
```

There are a number of functions that you can apply to an `lm` object. These include:

Function	Description
`summary`	Get a variety of information on the model, including coefficients and p-values for the coefficients
`coefficients`	Pull out just the coefficients for a model
`fitted`	Get the fitted values from the model (for the data used to fit the model)
`plot`	Create plots to help assess model assumptions
`residuals`	Get the model residuals

For example, you can get the coefficients from the model by running:

```
coefficients(mod_a)
#> (Intercept)          ht
#>   -8.694768    0.235050
```

The estimated coefficient for the intercept is always given under the name "(Intercept)". Estimated coefficients for independent variables are given based on their column names in the original data ("ht" here, for β_1, or the estimated increase in expected weight for a one unit increase in height).

You can use the output from a `coefficients` call to plot a regression line based on the model fit on top of points showing the original data (Figure 1).

```
library(ggplot2)
mod_coef <- coefficients(mod_a)
ggplot(nepali, aes(x = ht, y = wt)) +
  geom_point(size = 0.2)  +
  geom_abline(aes(intercept = mod_coef[1], slope = mod_coef[2]), col = "blue") +
  xlab("Height (cm)") + ylab("Weight (kg)")
```

You can also add a linear regression line to a scatterplot by adding the geom `geom_smooth` using the argument `method = "lm"`.

You can use the function `residuals` on an `lm` object to pull out the residuals from the model fit:

```
head(residuals(mod_a))
#>           1           2           3           4           5           6
#>  0.05820415 -0.82693141 -0.08223993  0.48644436 -0.46920621 -0.06405608
```

The result of a `residuals` call is a vector with one element for each of the non-missing observations (rows) in the dataframe you used to fit the model.

You can also use the shorter functions `coef` and `resid` as alternatives.

The `summary` function returns different output depending on the type of object that is input to the function. If you input a regression model object, the function gives you a lot of information, including

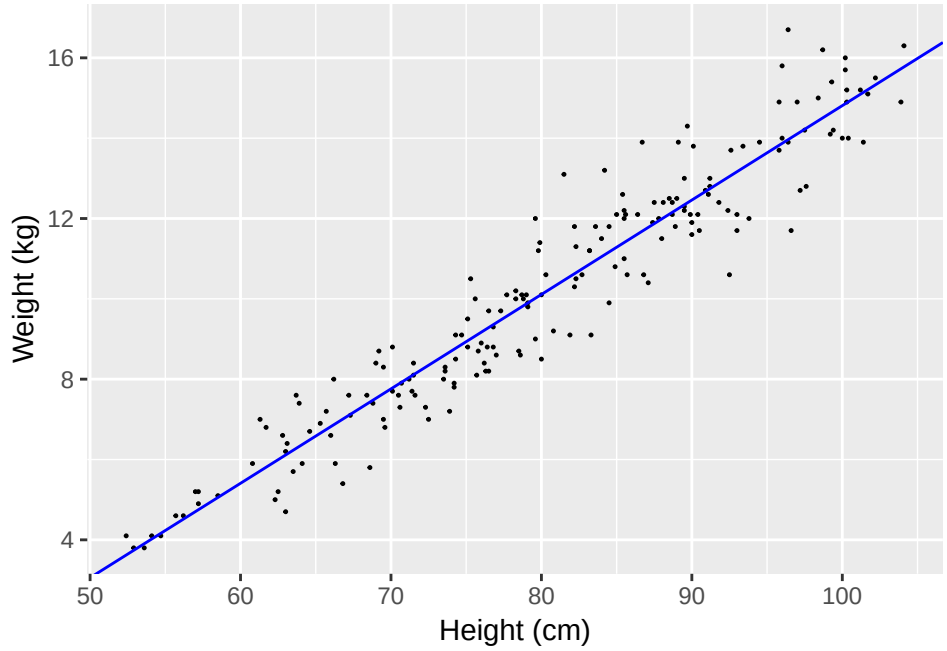


Figure 1: Example of using the output from a coefficients call to add a regression line to a scatterplot.

(1) basic summary statistics for the residuals (to meet model assumptions, the median should be around zero and the absolute values fairly similar for the first and third quantiles),
(2) coefficient estimates, standard errors, and p-values,
(3) some model summary statistics, including residual standard error, degrees of freedom, number of missing observations, and F-statistic.

```
summary(mod_a)
#>
#> Call:
#> lm(formula = wt ~ ht, data = nepali)
#>
#> Residuals:
#>      Min       1Q   Median       3Q      Max
#> -2.44736 -0.55708  0.01925  0.49941  2.73594
#>
#> Coefficients:
#>              Estimate Std. Error t value Pr(>|t|)
#> (Intercept) -8.694768   0.427398  -20.34   <2e-16 ***
#> ht           0.235050   0.005257   44.71   <2e-16 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 0.9017 on 183 degrees of freedom
#>   (15 observations deleted due to missingness)
#> Multiple R-squared:  0.9161, Adjusted R-squared:  0.9157
#> F-statistic:  1999 on 1 and 183 DF,  p-value: < 2.2e-16
```

The object returned by the `summary()` function when it is applied to an `lm` object is a list.

With any list, you can use the `names` function to get the names of all of the different elements of the object:

```
names(summary(mod_a))
#>  [1] "call"          "terms"         "residuals"     "coefficients"
#>  [5] "aliased"       "sigma"         "df"            "r.squared"
#>  [9] "adj.r.squared" "fstatistic"    "cov.unscaled"  "na.action"
```

You can use the `$` operator to pull out any element of the list. For example:

```
summary(mod_a)$coefficients
#>              Estimate  Std. Error   t value      Pr(>|t|)
#> (Intercept) -8.694768 0.427397843 -20.34350  7.424640e-49
#> ht           0.235050 0.005256822  44.71334 1.962647e-100
```

The `plot` function will give different output depending on the class of the object that you input. For an `lm` object, you get a number of useful diagnostic plots that will help you check regression assumptions (Figure 2):

```
plot(mod_a)
```

You can also use binary variables or factors as independent variables in regression models. For example, in the `nepali` dataset, `sex` is a factor variable with the levels "Male" and "Female". You can fit a linear model of weight regressed on sex for this data with the call:

```
mod_b <- lm(wt ~ sex, data = nepali)
```

This call fits the model:

$$Y_i = \beta_0 + \beta_1 X_{1,i} + \epsilon_i$$

where $X_{1,i}$: sex of child i, where 0 = male and 1 = female.

Here are the estimated coefficients from fitting this model:

```
summary(mod_b)$coefficients
#>              Estimate Std. Error   t value     Pr(>|t|)
#> (Intercept) 10.497980  0.3110957 33.745177 1.704550e-80
#> sexFemale   -0.674724  0.4562792 -1.478752 1.409257e-01
```

By default, the first factor level is used as the baseline level.

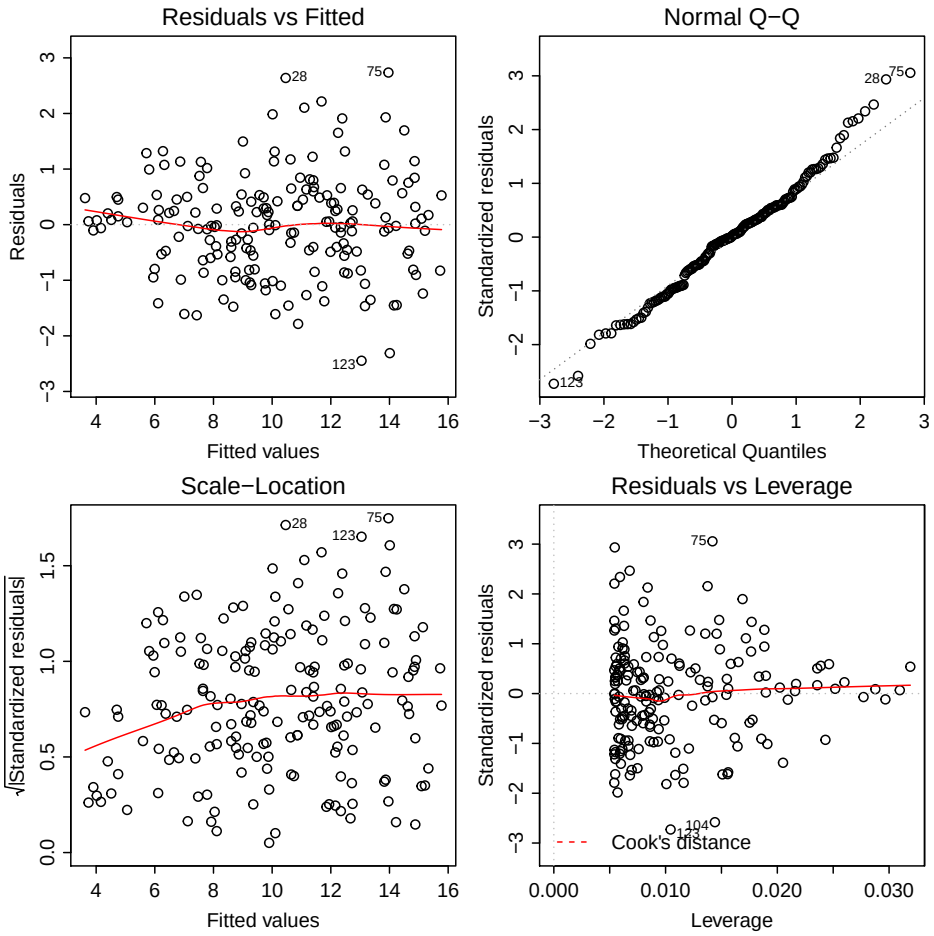


Figure 2: Example output from running the plot function with an lm object as the input.

For example, the model fit above tells us that the estimated mean weight of males is 10.5, while the estimated mean weight of females is 10.5 + -0.7 = 9.8.

If you would prefer that a different level of the factor be the baseline, you can do that by using the `levels` argument in the `factor` function. For example:

```
nepali_reset <- nepali %>%
  mutate(sex = factor(sex, levels = c("Female", "Male")))
mod_b_reset <- lm(wt ~ sex, data = nepali_reset)
summary(mod_b_reset)$coef
#>             Estimate Std. Error   t value     Pr(>|t|)
#> (Intercept) 9.823256  0.3337816 29.430189 2.626719e-71
#> sexMale     0.674724  0.4562792  1.478752 1.409257e-01
```

Check that coef. estimates are the same as above.

1.7.3 Generalized linear models (GLMs)

You can fit a variety of models, including linear models, logistic models, and Poisson models, using generalized linear models (GLMs).

For linear models, the only difference between `lm` and `glm` are the mechanics of how they estimate the model coefficients (`lm` uses least squares while `glm` uses maximum likelihood). You will (almost always) get exactly the same estimated coefficients.

For example, here is the code to fit a linear regression model for weight regressed on height from the `nepali` dataset:

```
mod_c <- glm(wt ~ ht, data = nepali)
```

This call fits the same regression model estimated earlier with the `lm` function. You can see that the two methods give exactly the same coefficient estimates:

```
coef(mod_c)
#> (Intercept)          ht
#>   -8.694768    0.235050
coef(mod_a)
#> (Intercept)          ht
#>   -8.694768    0.235050
```

Unlike the `lm` function, however, the `glm` function also allows you to fit other model types, including logistic and Poisson models. You can specify the model type using the `family` argument to the `glm` call:

Model type	'family' argument
Linear	'family = gaussian(link = 'identity')'
Logistic	'family = binomial(link = 'logit')'
Poisson	'family = poisson(link = 'log')'

1.7.4 References– statistics in R

One great book to find out more about using R for basic statistics is:

- Introductory Statistics with R

If you want all the details about fitting linear models and GLMs in R, Julian Faraway's books are fantastic. He has one on linear models and one on extensions including logistic and Poisson models:

- Linear Models with R
- Extending the Linear Model with R

1.8 In-course exercise