4. Model fitting

Thomas Lumley
Ken Rice

UW Biostatistics

Liège, September 2009

Regression commands

Two of the most important `R` commands;

- `lm()`: fits **L**inear **M**odels
- `glm()`: fits **G**eneralized **L**inear **M**odels

(If you've used SAS, its `glm` is **not** the same as `R`'s)

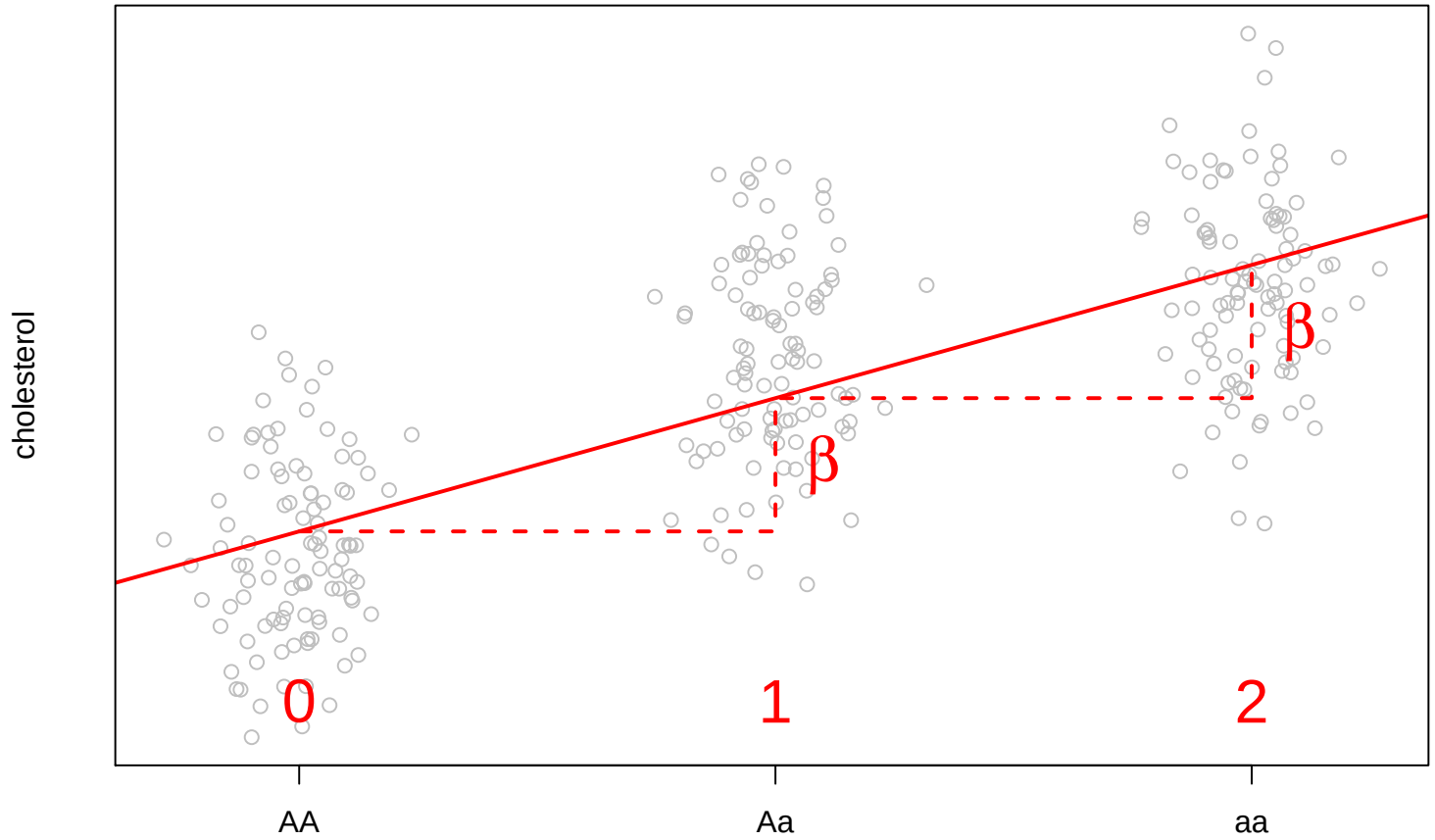
'Linear Regression' and 'Logistic Regression' are special cases.

There's a lot to learn here – entire graduate courses! – so the help files are huge. How are `lm()`, `glm()` used in genetics?

Linear regression, with SNPs

Many analyses fit the 'additive model'

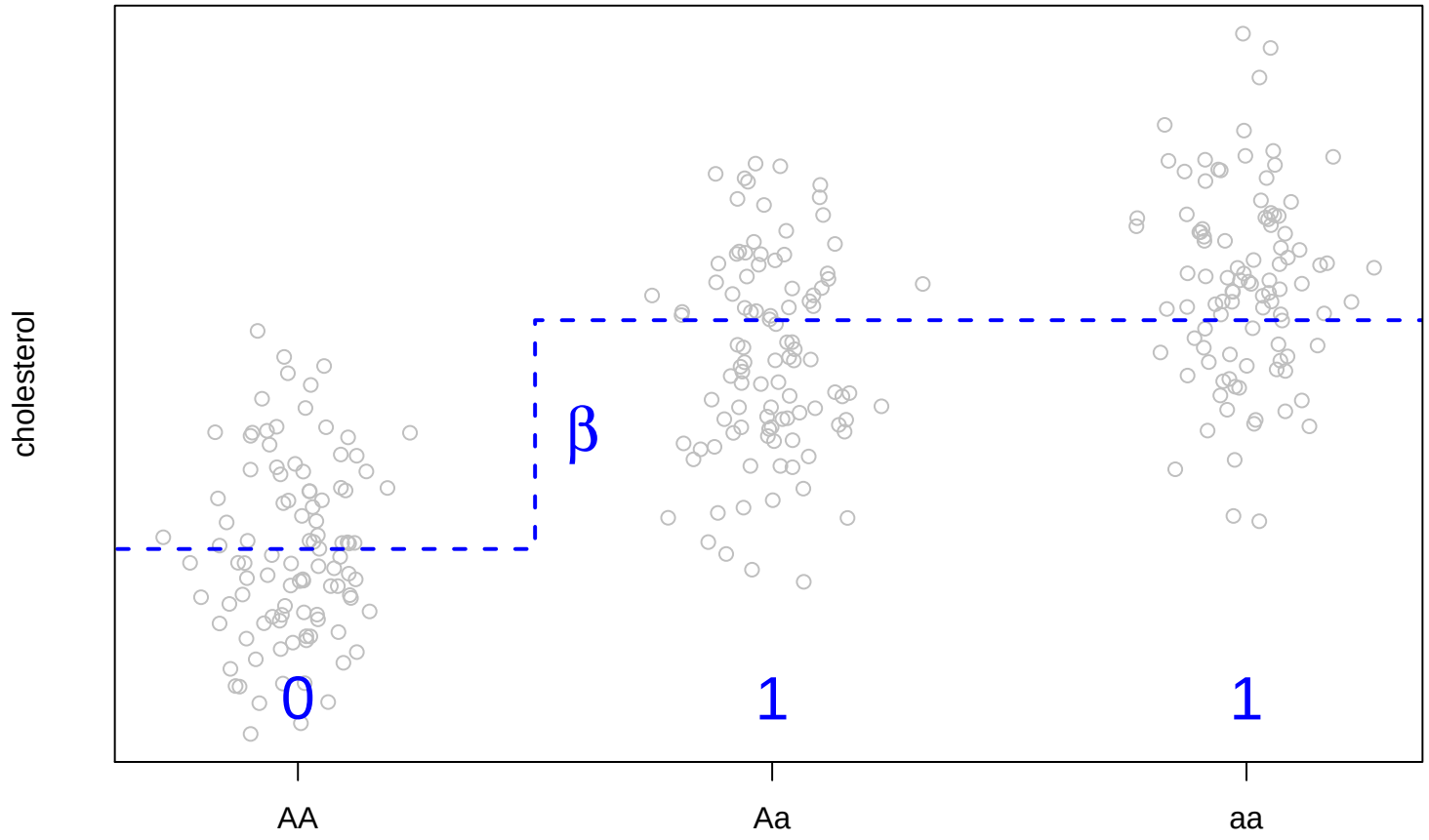
$$y = \beta_0 + \beta \times \#\text{minor alleles}$$



Linear regression, with SNPs

An alternative is the 'dominant model';

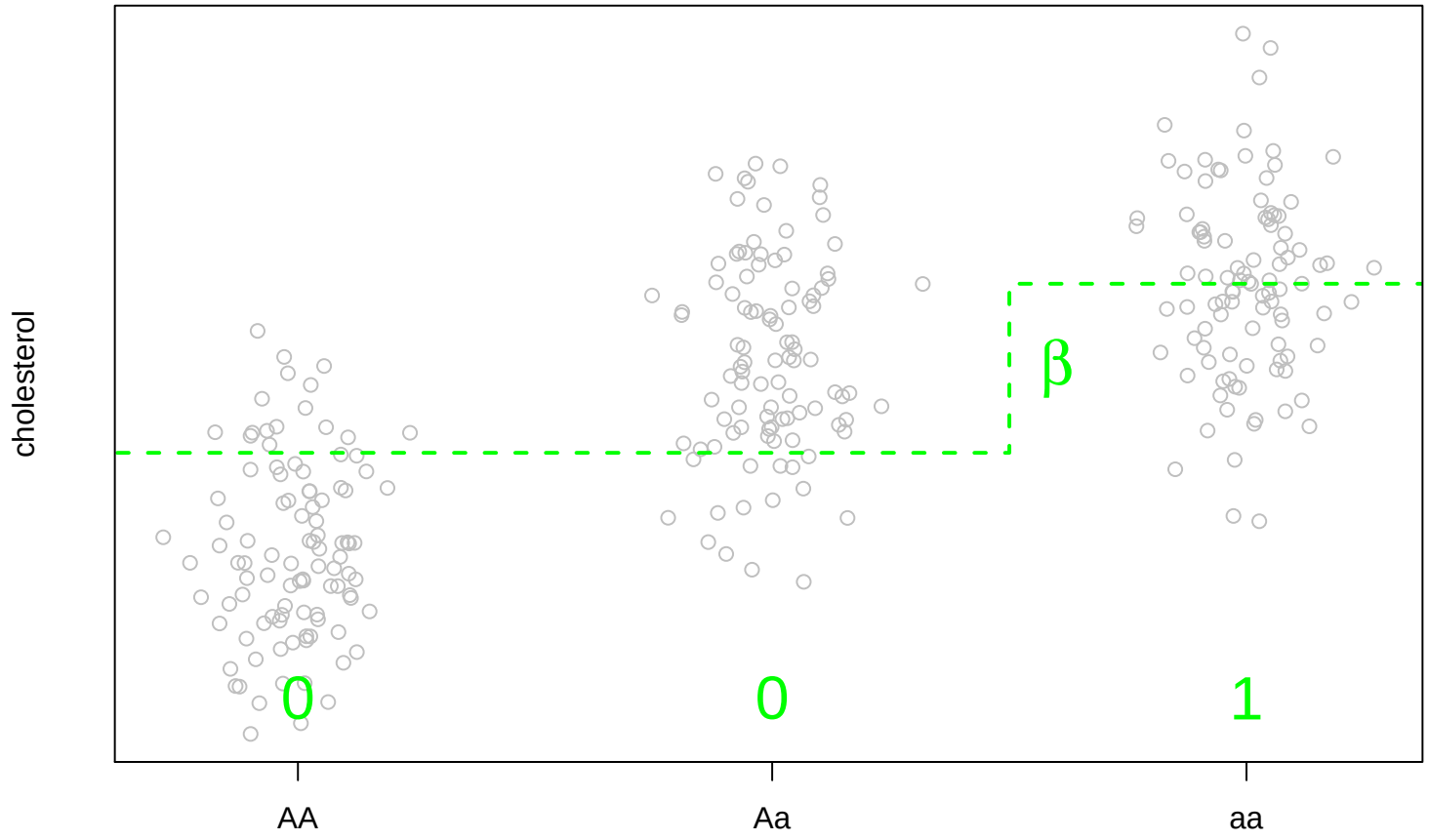
$$y = \beta_0 + \beta \times (G \neq AA)$$



Linear regression, with SNPs

or the ‘recessive model’;

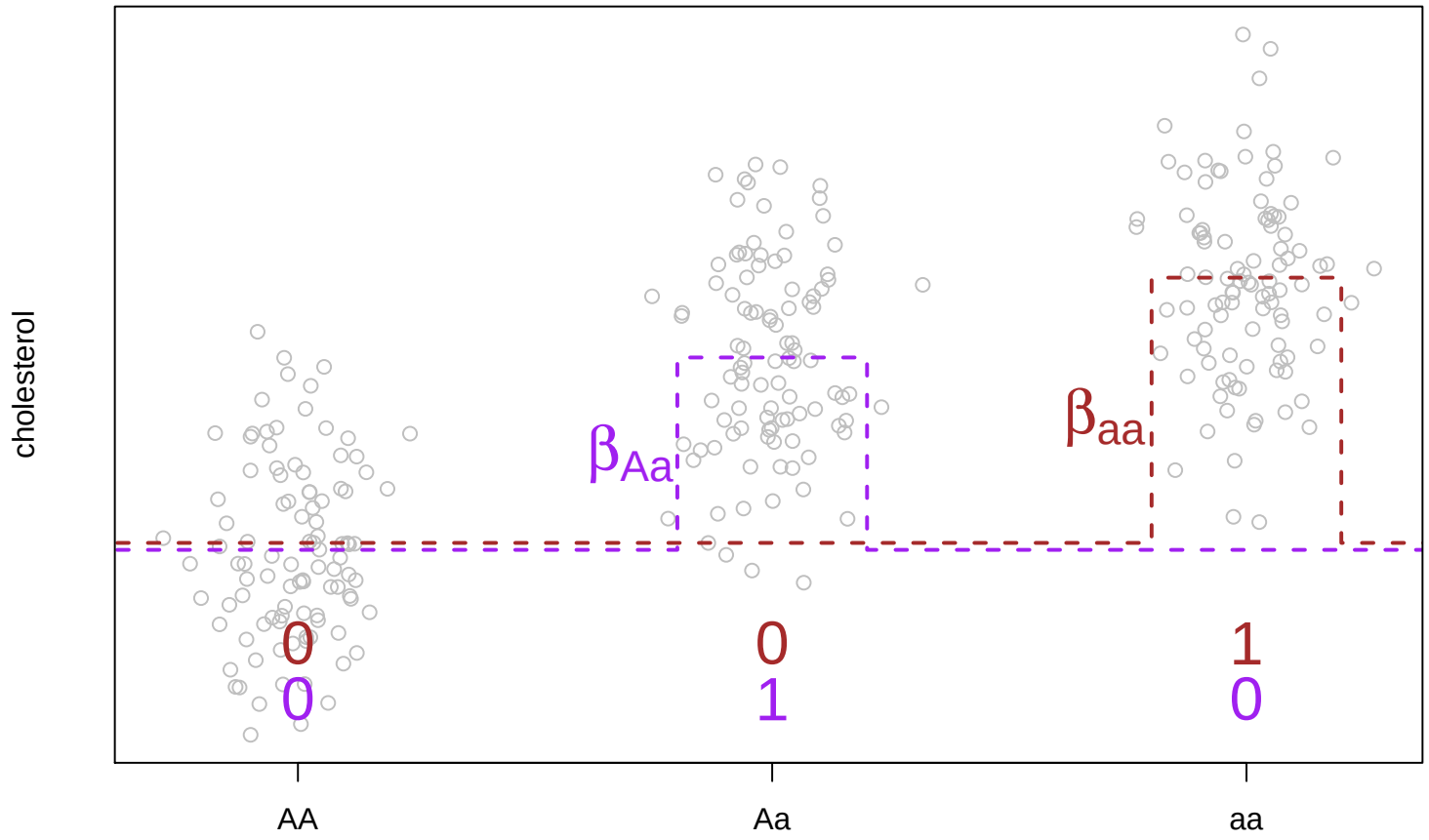
$$y = \beta_0 + \beta \times (G == AA)$$



Linear regression, with SNPs

Finally, the 'two degrees of freedom model';

$$y = \beta_0 + \beta_{Aa} \times (G == Aa) + \beta_{aa} \times (G == aa)$$



Use of `lm()` in genetics

The `lm()` command fits all of these, in the same way. Formally,

`lm(outcome` $\sim$ `genetic.predictor, [...] )`

estimates the association between `outcome` and `predictor`

The **optional** arguments [...] might be

- `data = my.data` – your dataset
- `subset = race=="CEPH"` – use partial data
- `weights =` – for advanced analyses

Use of `lm()` in genetics

How to make the `genetic.predictor` variable? Note that when `R` meets `FALSE` or `TRUE` in a 'math' setting, it will **coerce** them to be zero or one. So `1 + 2*TRUE` is 3, `TRUE + 2*FALSE` is 1, etc

Suppose you had genotypes stored in vector `g`, as character strings `"AA"`/`"Aa"`/`"aa"`. You might use these commands;

Chosen Model	`genetic.predictor <-`
Additive	`(g=="Aa") + 2*(g=="aa")`
Dominant	`(g=="Aa") | (g=="aa")`
Recessive	`g=="aa"`
2 degrees of freedom	`factor(g)`

- There are many other ways to do this!
  Use `table(g, genetic.predictor)` to check what you did
- Often, genotypes may be stored as 0/1/2. This is easier to work with in `R` – but makes it harder to decide if A/C/G/T is the risk allele

`lm()`: Estimates, Intervals, p-values

`lm()` produces **point estimates** for your model;

```
> genetic.predictor <- (g=="Aa") + 2*(g=="aa") #using additive model
> my.lm <- lm( cholesterol ~ genetic.predictor )
> my.lm
Call:
lm(formula = cholesterol ~ genetic.predictor)
Coefficients:
(Intercept)      predictor
     0.2104        0.9507
```

– also available via `my.lm$coefficients` or `coef(my.lm)`.

The `coefficients` in the output tell you the **additive increase** in `outcome` associated with a **one-unit** difference in the `genetic predictor`.

The coefficient for `predictor` is in units of `cholesterol` per 'a' allele

`lm()`: Estimates, Intervals, p-values

You will also want **confidence intervals**;

```
> confint.default(my.lm)
                 2.5 %    97.5 %
(Intercept) 0.08391672 0.3368275
predictor   0.85279147 1.0486953
```

Remember to **round these numbers** to an appropriate number of significant figures! (2 or 3 is usually enough)

We are **seldom** interested in the `Intercept`

`lm()`: Estimates, Intervals, p-values

Two-sided **p-values** are also available;

```
> summary(my.lm)
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.21037    0.06426   3.274  0.00119 **
predictor    0.95074    0.04977  19.101  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

In this data, we have **strong evidence** of an **additive effect** of the minor allele on cholesterol

`summary(my.lm)` gives **many** other details – ignore for now

Confidence intervals are just `Estimate` $\pm$ 2$\times$`Std.Error`

Use of `glm()` in genetics

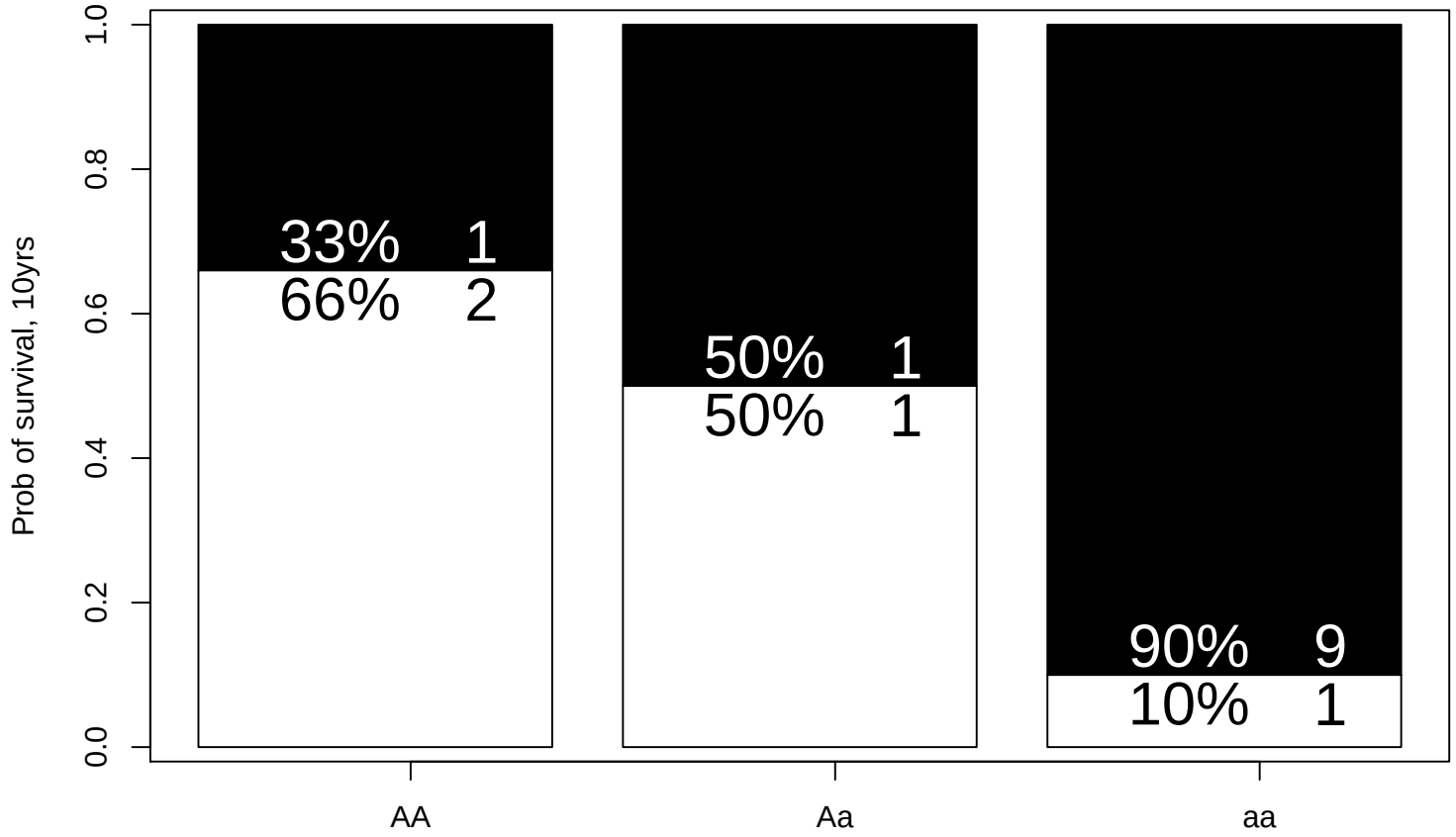
Logistic regression is the 'default' analysis for **binary outcomes**

Outcome	Type	Regression	Scale
Cholesterol Blood Pressure BMI	Continuous	Linear	Difference in Outcome
Death Stroke BMI>30	Binary	Logistic	Ratio of odds

What are **odds**? Really just **probability**...

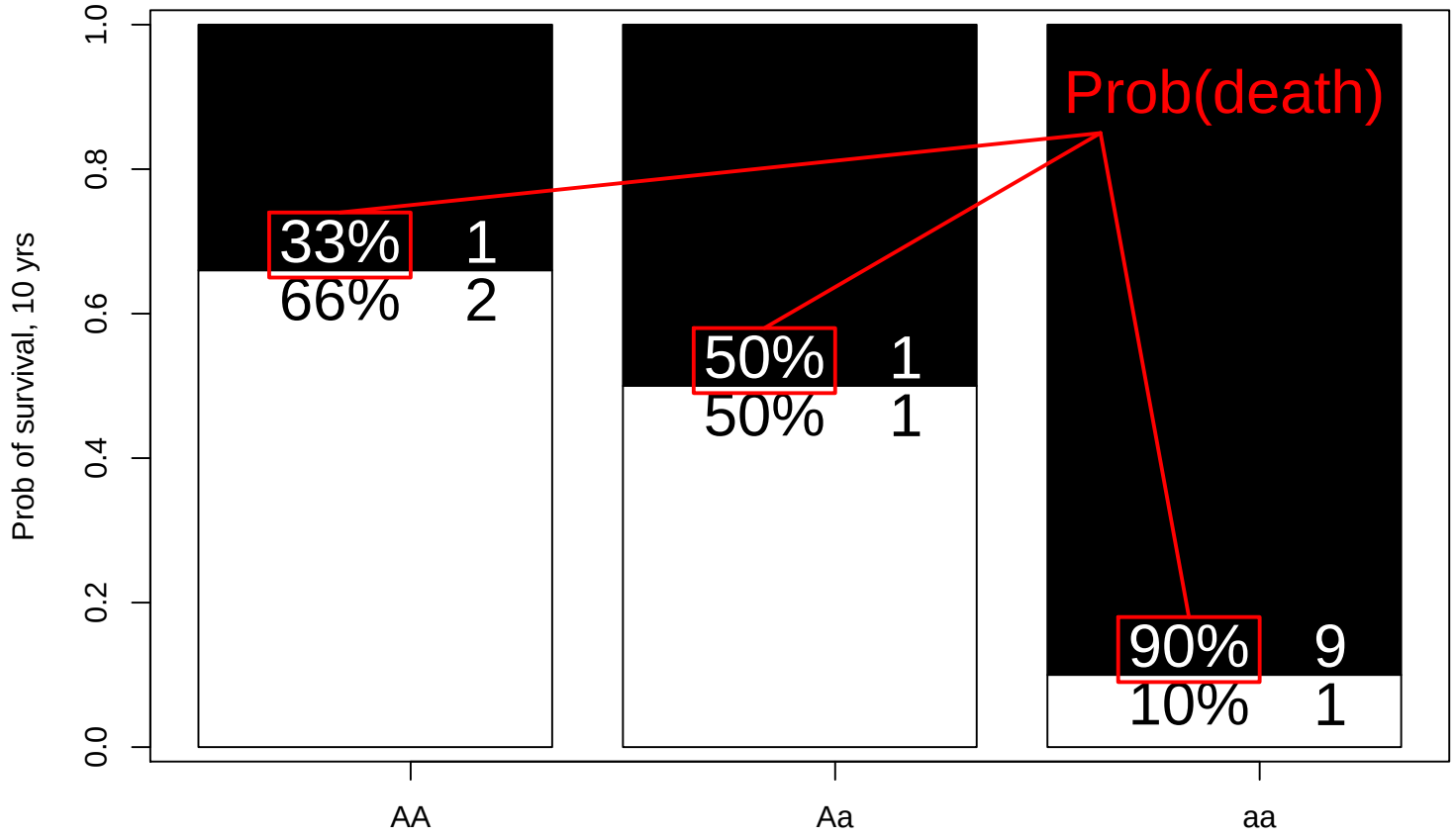
Use of `glm()` in genetics

Odds are a [gambling-friendly] measure of chance;



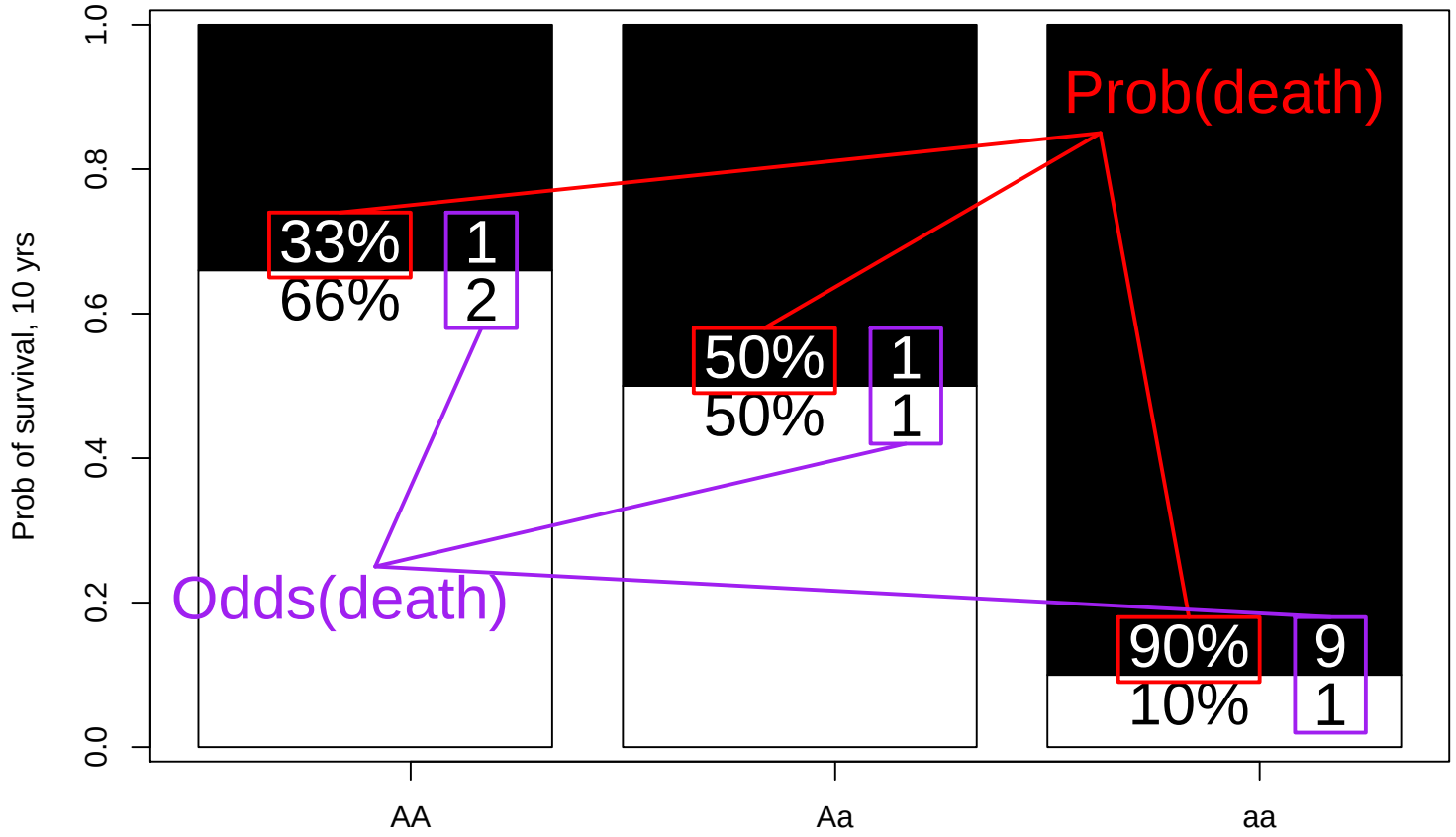
Use of `glm()` in genetics

Odds are a [gambling-friendly] measure of chance;



Use of `glm()` in genetics

Odds are a [gambling-friendly] measure of chance;



– so what are **odds ratios**?

Use of `glm()` in genetics

Using the data from the bar charts;

```
> genpred2 <- factor(g) # the 2df model
> glm1 <- glm( dead10yrs ~ genpred2, family=binomial)
> coef(glm1)
     pred2Aa      pred2aa
      0.6931       2.8904
```

These are **log odds ratio** estimates; to transform to OR, use $e^{0.6931} = 2, e^{2.8904} = 18$

They are given **relative to the baseline group** – 'AA' in this case

Don't forget the `family=binomial` argument!

Use of `glm()` in genetics

Confidence intervals, p-values as with `lm()`, **for the log odds ratios**;

```
> confint.default(glm1)
                2.5 %     97.5 %
genpred2Aa     0.1201986  1.2660957
genpred2aa     2.1148912  3.6658523
```

```
> summary(glm1)
            Estimate Std. Error z value Pr(>|z|)
genpred2Aa      0.6931     0.2923   2.371  0.01773 *
genpred2aa      2.8904     0.3957   7.305 2.77e-13 ***
```

Use `exp()` to get odds ratio estimates, intervals; p-values are **scale-independent**

The formula syntax

We fit `lm(y~ genetic.predictor)` and `glm(y~ genpred2)`. To see how phenotype depends on *several* covariates, we specify e.g.

```
y ~ genotype.pred + age + sex
```

– formally, this gives *multivariate regression*; the `genotype.pred` coefficients reflect the genotype effects *adjusted for age and sex*

- Separate covariates with '+'. This is *not* addition!
- For now, make predictor variables first, then do regression. It's possible to do everything in one step, but use of e.g. '+' will confuse `R` – unless you're careful.
- For keen people; in the formula syntax, `*` indicates that interactions should be fitted, `I()` insulates mathematical operations, `-1` removes the intercept... see `?formula`
- For *very* keen people; `vcovHC()` in the `sandwich` package provides 'robust' standard errors; `coeftest()` in the `lmtest` package uses them to give 'robust' tests.