Topics in Multivariate Analysis

APSTA GE-2004

Lecture 7 - Multivariate Normal Distribution and Regression

3/8/2022

Outline

Welcome! Today, we'll cover the following:

- Multivariate normal distribution
- Multilevel regression
- Principal component analysis

Reading:

RAOS Ch 3; Ch 5; Ch 11.8
Estimating Generalized (Non-)Linear Models with Group-Specific Terms with rstanarm by Gabry and Goodrich
RAOS Appendix A and B

One-dimensional Gaussian distribution

One-dimensional Gaussian distribution

First principles about the process that generates Y_i:

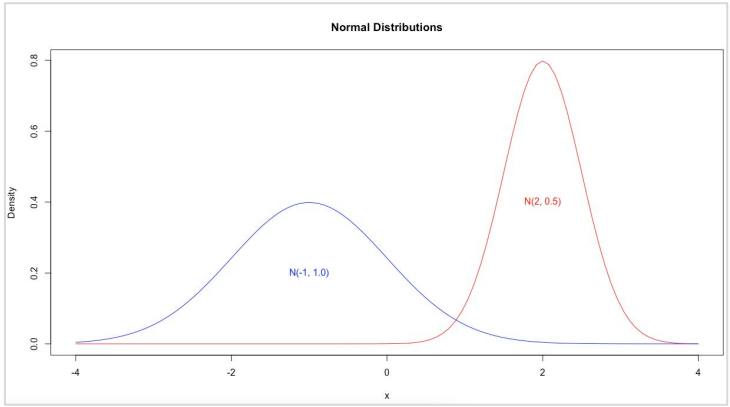
- Many different first principles
  - The Central Limit Theorem
  - Normal by addition
  - Normal by multiplication
  - Normal by log-multiplication
- The univariate normal density (with mean μ_i, variance σ^2):

  $$N(y_i \mid \mu_i, \sigma^2) = (2\pi\sigma^2)^{-1/2} \exp(-(y_i - \mu_i)^2 / 2\sigma^2)$$

- Regression notation:

  $Y_i \sim N(\mu_i, \sigma^2)$ stochastic

  $\mu_i = x_i\beta$ systematic



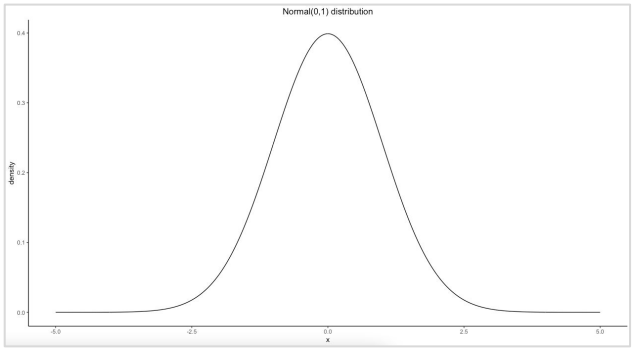
Multivariate Gaussian distribution

Univariate and Multivariate Gaussian distributions

Univariate (aka 1-dimensional) normal distribution

Notation: $N(\mu, \sigma^2)$
where μ is the mean and σ is the standard deviation

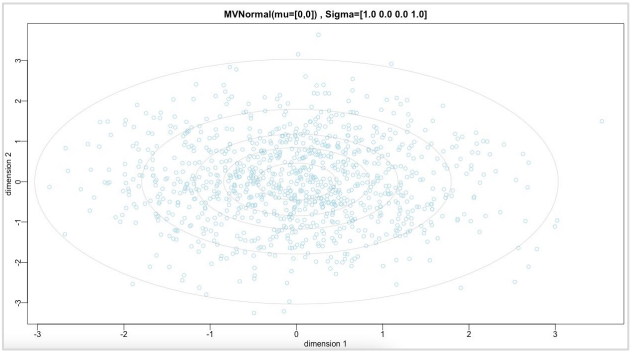
e.g. N(0, 1)



Multivariate (aka k-dimensional) normal distribution

Notation: $N_k(\boldsymbol{\mu}, \boldsymbol{\Sigma})$
where $\boldsymbol{\mu}$ is *a vector of means* and $\boldsymbol{\Sigma}$ is *a $k \times k$ covariance matrix*

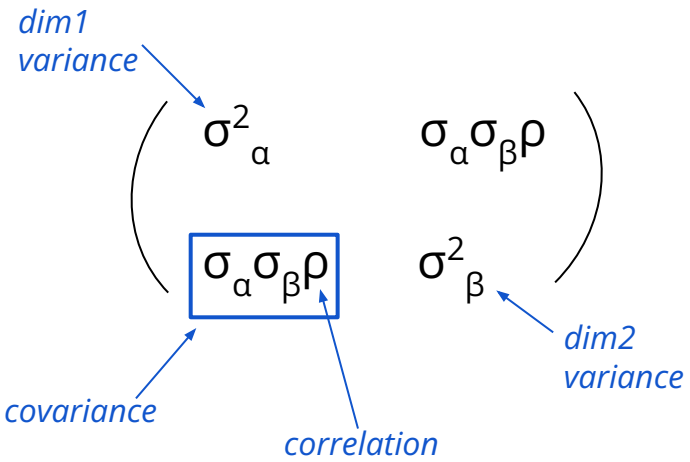
e.g. N_2([0,0], [1 0
0 1])

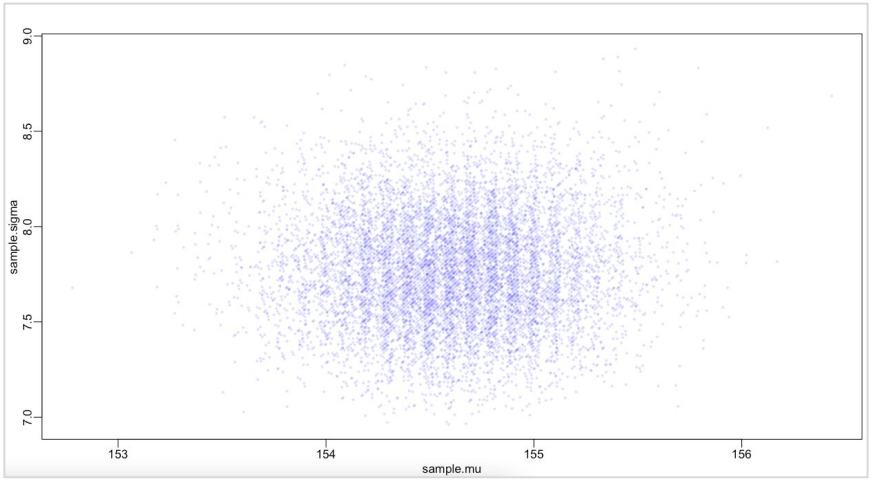


Multivariate Gaussian distribution

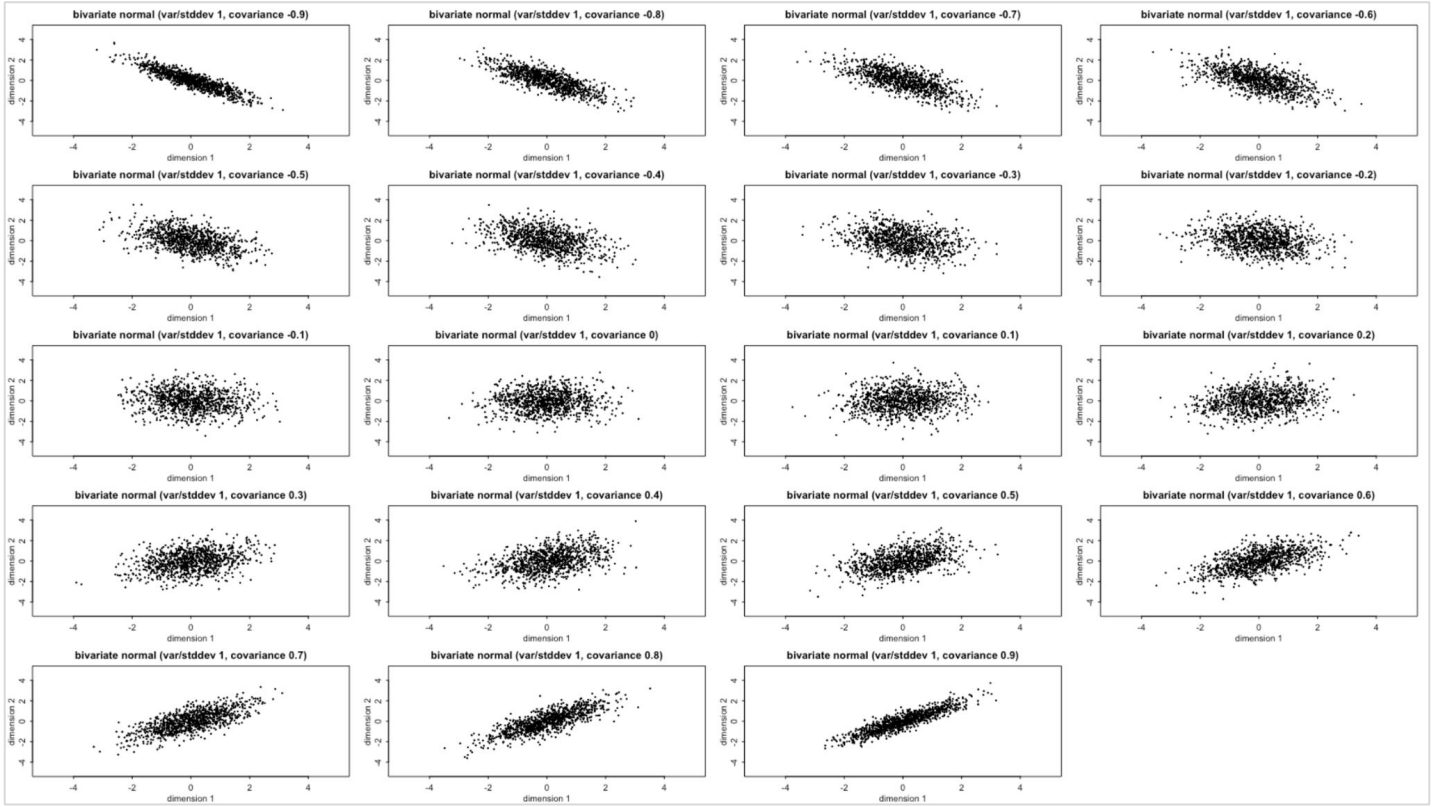
e.g. 2-dimensional Gaussian distribution:

- $\boldsymbol{\mu}$ – vector of means
- $\boldsymbol{\Sigma}$ – variance-covariance matrix

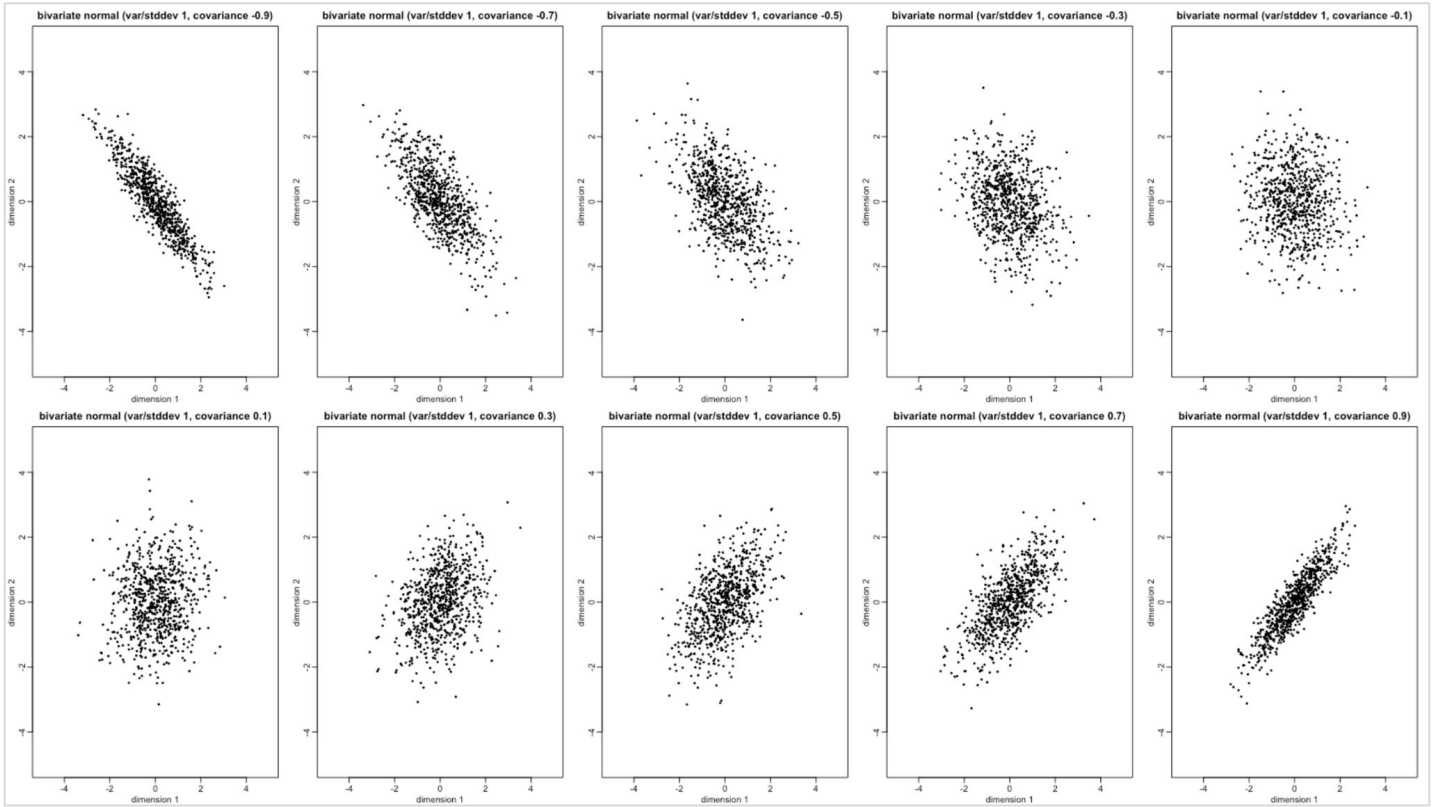




2-dimensional normal distribution with covariances ranging from −0.9 to 0.9 by 0.1



2-dimensional normal distribution with covariances ranging from −0.9 to 0.9 by 0.2



5-dimensional multivariate normal distribution with correlations 0.8

Two key properties of multivariate normal distributions:

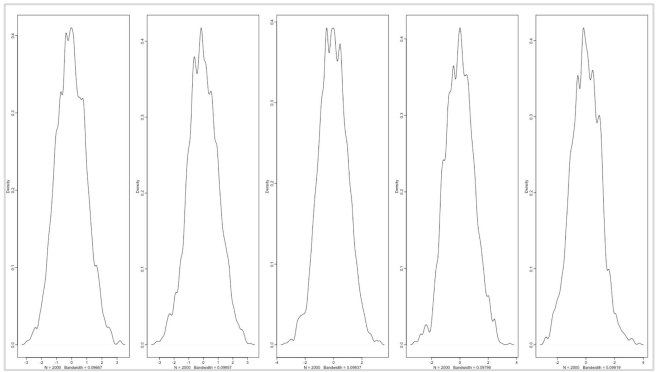
1. If $\mathbf{y} \sim \text{Normal}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, then each element of **y** is normally distributed
2. If $\mathbf{y} \sim \text{Normal}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, then y_i and y_j, any two elements of **y**, are independent if, and only if, they are uncorrelated, that is, $\text{Cov}(y_i, y_j) = 0$

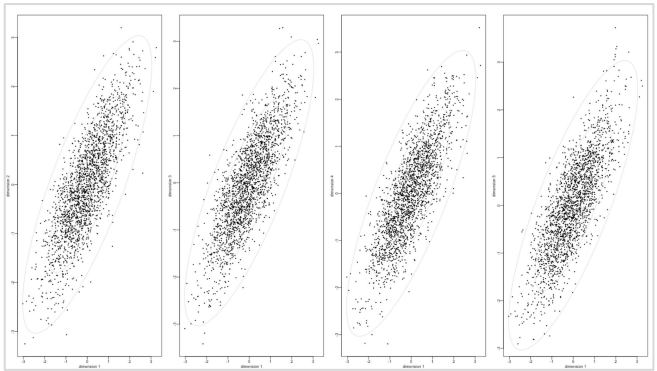
Variance-Covariance Matrix

$$\text{Var}(\mathbf{y}) = \begin{bmatrix} \sigma_1^2 & \sigma_{12} & \cdots & \sigma_{1n} \\ \sigma_{21} & \sigma_2^2 & \cdots & \sigma_{2n} \\ \vdots & & & \\ \sigma_{n1} & \sigma_{n2} & \cdots & \sigma_n^2 \end{bmatrix}$$

- Variances of each element of **y** down its diagonal
- Covariance terms in the off diagonals

```
> n <- 2000
> k <- 5
> rho <- 0.8
>
> Sigma <- rho*array(1, c(k,k)) + (1-rho)*diag(k)
> Sigma
     [,1] [,2] [,3] [,4] [,5]
[1,]  1.0  0.8  0.8  0.8  0.8
[2,]  0.8  1.0  0.8  0.8  0.8
[3,]  0.8  0.8  1.0  0.8  0.8
[4,]  0.8  0.8  0.8  1.0  0.8
[5,]  0.8  0.8  0.8  0.8  1.0
>
> X <- mvrnorm(n=n, mu=rep(0,k), Sigma=Sigma)
>
> dim(X)
[1] 2000    5
>
> head(X)
           [,1]       [,2]      [,3]       [,4]      [,5]
[1,] 0.53631891 -0.4985182 1.4602025  1.4238147 0.9105752
[2,] 0.03527085  0.3930214 0.3262860 -0.3117836 0.5679978
[3,] 1.67035609  1.5246306 1.4576932  2.4971745 2.4896613
[4,] 0.93192381  1.6683864 1.1673334  2.2209627 1.6542479
[5,] 1.60522374  0.5147985 0.8237284  1.3878868 0.6998603
[6,] 1.49169592  1.7794343 1.4819433  0.9460341 1.9337277
```





Variance, covariance, correlation

We define **covariance** ($\sigma_\alpha \sigma_\beta \varrho_{\alpha\beta}$) using three parameters:

- standard deviation of the first variable, e.g. σ_α
- standard deviation of the second variable, e.g. σ_β
- correlation between the two variables, e.g. $\varrho_{\alpha\beta}$

variance is a special case of covariance, i.e. the covariance of a variable with itself:

$$\text{var}(x) = \text{cov}(x,x) = E(x^2) - E(x)^2 = E(x^2) \text{ ; when } E(x) = 0$$

correlation is a rescaled covariance, so that the minimum is –1 and the maximum is 1:

$$\varrho_{xy} = \text{cov}(x,y) \,/\, \sqrt{\text{var}(x)\,\text{var}(y)}$$

$$\text{cov}(x,y) = \sqrt{[\,\text{var}(x)\,\text{var}(y)\,]}\,\varrho_{xy} = \sigma_x \sigma_y \varrho_{xy}$$

```
> # variance, covariance, correlation
> sigma_a <- 1     # std dev in intercepts
> sigma_b <- 0.5   # std dev in slopes
> rho <- 0.7       # correlation between intercepts and slopes
>
> # covariance between a and b
> cov_ab <- sigma_a * sigma_b * rho
> cov_ab
[1] 0.35
>
> # variance of a
> var_a <- sigma_a * sigma_a
> var_a
[1] 1
>
> # variance of b
> var_b <- sigma_b * sigma_b
> var_b
[1] 0.25
>
> # correlation between a and b
> cor_ab <- cov_ab / sqrt(var_a * var_b)
> cor_ab
[1] 0.7
>
> # variance-covariance matrix
> Sigma <- matrix( c(sigma_a^2, cov_ab, cov_ab, sigma_b^2) , ncol=2 )
> Sigma
     [,1] [,2]
[1,] 1.00 0.35
[2,] 0.35 0.25
>
> # correlation matrix
> cov2cor(Sigma)
     [,1] [,2]
[1,]  1.0  0.7
[2,]  0.7  1.0
```

Multivariate Gaussian distribution

mu and sigma

Gaussian model of adult height

The data contained in data(Howell1) are partial census data for the Dobe area !Kung San, compiled from interviews conducted by Nancy Howell in the late 1960s

For now, we filter down to individuals of age 18 or greater to focus on adults since height is strongly correlated with age, before adulthood

Our goal is to model these values using a Gaussian distribution. But which Gaussian distribution? There are an infinite number of them, with an infinite number of different means and standard deviations

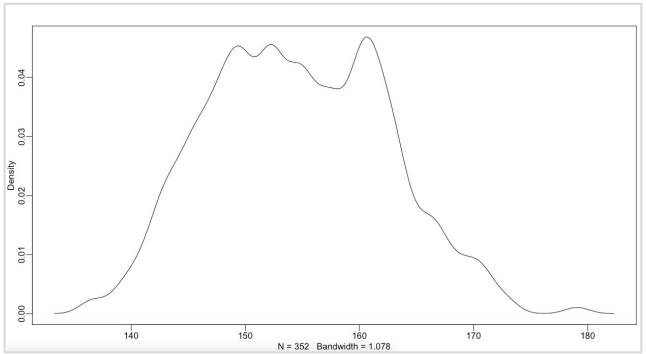
Let's write down the general model and compute the plausibility of each combination of μ and σ:

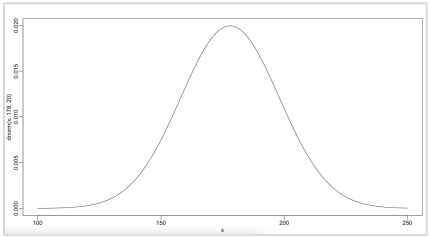
$h_i \sim \text{Normal}(\mu, \sigma)$ [likelihood]

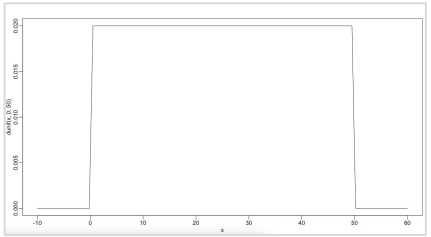
$\mu \sim \text{Normal}(178, 20)$ [μ prior]

$\sigma \sim \text{Uniform}(0, 50)$ [σ prior]

In most cases, priors are specified independently for each parameter, which amounts to assuming $\Pr(\mu,\sigma) = \Pr(\mu)\Pr(\sigma)$







Gaussian model of adult height

Now let us fit this model in rethinking and rstanarm code:

$h_i \sim \text{Normal}(\mu,\sigma)$ [height ~dnorm(mu,sigma)]

$\mu \sim \text{Normal}(178,20)$ [mu ~dnorm(178,20)]

$\sigma \sim \text{Uniform}(0,50)$ [sigma ~dunif(0,50)]

```
> library(rethinking)
> data(Howell1)
> d <- Howell1
> d2 <- d[ d$age >= 18 , ]
>
> m4.1 <- quap( alist(
+   height ~ dnorm( mu , sigma ) ,
+   mu ~ dnorm( 178 , 20 ) ,
+   sigma ~ dunif( 0 , 50 )
+ ) , data=d2 )
>
> precis(m4.1)
        mean   sd  5.5%  94.5%
mu    154.61 0.41 153.95 155.27
sigma   7.73 0.29   7.27   8.20
```

The summaries of the posterior distribution provide Gaussian approximations for each parameter's *marginal* distribution:

Mu (μ):
The plausibility of each value of μ, after averaging over the plausibility of each value of σ, is given by a Gaussian distribution with mean 154.6 and standard deviation 0.4.

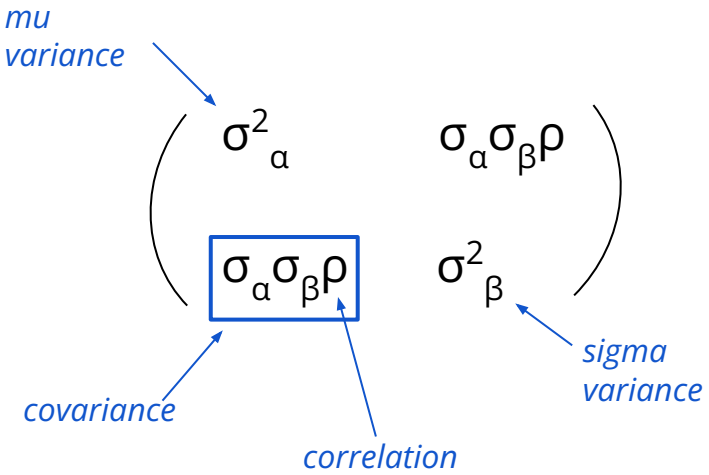
Sigma (σ):
The plausibility of each value of σ, after averaging over the plausibility of each value of μ, is given by a Gaussian distribution with mean 7.7 and standard deviation 0.3.

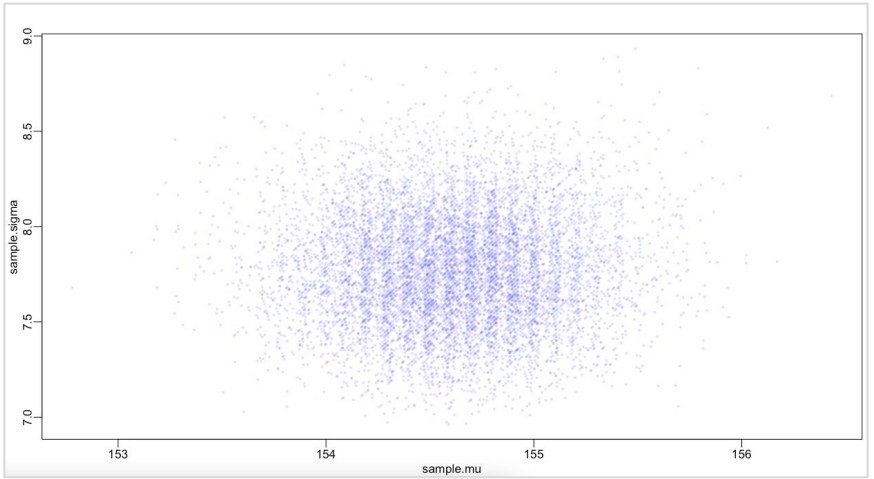
```
> b4.1 <- stan_glm(height ~ 1, data = d2,
+                  prior_intercept = normal( location = c(178) , scale = c(20) ) ,
+                  prior_aux = NULL, refresh = 0)
> summary(b4.1)

Model Info:
 function:     stan_glm
 family:       gaussian [identity]
 formula:      height ~ 1
 algorithm:    sampling
 sample:       4000 (posterior sample size)
 priors:       see help('prior_summary')
 observations: 352
 predictors:   1

Estimates:
              mean   sd   10%   50%   90%
(Intercept) 154.6    0.4 154.1 154.6 155.1
sigma         7.8    0.3   7.4   7.8   8.2
```

Quadratic approximation with more than one parameter (μ and σ) is a multivariate Gaussian distribution

2-dimensional Gaussian distribution:

- vector of means
- variance-covariance matrix

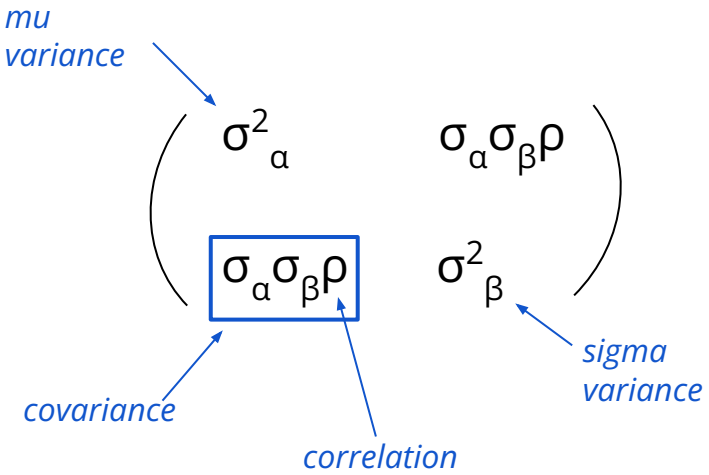


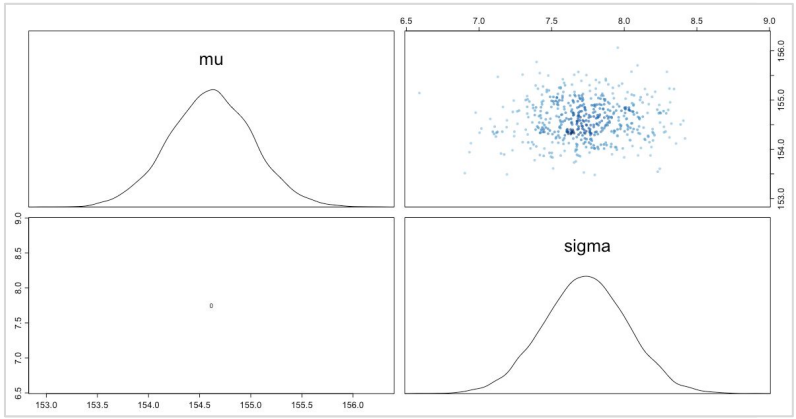


Quadratic approximation with more than one parameter (μ and σ) is a multivariate Gaussian distribution

2-dimensional Gaussian distribution:

- vector of means
- variance-covariance matrix





rethinking

```
> # variance-covariance matrix
> vcov( m4.1 )
                mu        sigma
mu    0.1697030293 0.0002206564
sigma 0.0002206564 0.0848600539
>
> # a vector of variances for the parameters
> diag( vcov( m4.1 ) )
       mu      sigma
0.16970303 0.08486005
>
> # a correlation matrix that tells how changes
> # in any parameter lead to correlated changes in the others
> cov2cor( vcov( m4.1 ) )
                mu       sigma
mu    1.000000000 0.001838739
sigma 0.001838739 1.000000000
```

rstanarm

```
> # extract posterior distribution as a matrix
> b4.1_post <- as.matrix(b4.1)
>
> # variance-covariance matrix
> var(b4.1_post)
             (Intercept)        sigma
(Intercept)  0.171280086 -0.003716241
sigma       -0.003716241  0.090056166
>
> # a vector of variances for the parameters
> diag( var(b4.1_post) )
(Intercept)       sigma
 0.17128009  0.09005617
>
> # a correlation matrix that tells how changes
> # in any parameter lead to correlated changes in the others
> cov2cor( var(b4.1_post) )
             (Intercept)        sigma
(Intercept)  1.00000000 -0.02992221
sigma       -0.02992221  1.00000000
```

Sample from multivariate Gaussian distribution

Now let us sample vectors of values from a multivariate Gaussian distribution, with its vector of means and variance-covariance matrix based on the model's fitted values:

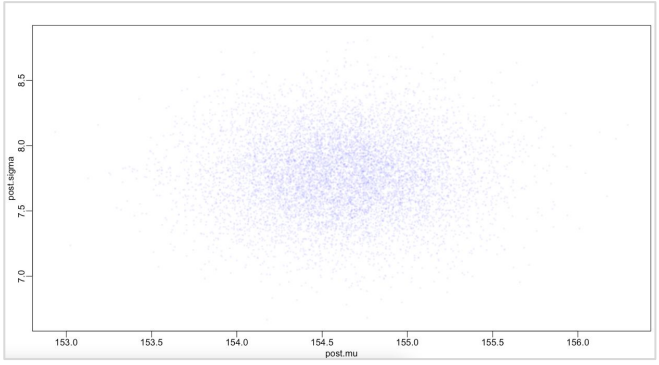
$$\mathbf{X} \sim \text{Normal}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$$

$$\boldsymbol{\mu} = [\, 154.6 \,,\, 7.7 \,]^{\mathsf{T}}$$

$$\boldsymbol{\Sigma} = \begin{bmatrix} 0.1697 & 0.0002 \\ 0.0002 & 0.0849 \end{bmatrix}$$

These samples preserve the covariance between μ and σ

This hardly matters right now, because μ and σ don't covary in this model. But once we add a predictor variable, covariance will matter a lot



rethinking

```
> # vector of means
> coef(m4.1)
       mu      sigma
154.607013   7.731331
>
> # variance-covariance matrix
> vcov(m4.1)
                mu        sigma
mu    0.1697395278 0.0002177895
sigma 0.0002177895 0.0849057177
>
> # draw samples from a multi-dimensional Gaussian distribution
> # with means and variance-covariance matrix based on the model
> library(rethinking)
> post <- extract.samples( m4.1 , n=1e4 )
> head(post)
       mu    sigma
1 155.4759 7.646574
2 154.5805 7.997083
3 153.8489 7.352140
4 154.3988 8.271771
5 154.7664 7.476361
6 154.8337 7.848426
>
> # plot the samples from the multi-dimensional Gaussian distribution
> jitt <- runif(n = dim(post)[1], min = -0.075, max = 0.075)
> plot( post$mu + jitt , post$sigma + jitt , cex=0.5 ,
+       col=col.alpha(rangi2,0.1) ,
+       xlab="post.mu" , ylab="post.sigma" , pch=16 )
```

rstanarm

```
> b4.1_post <- as.matrix(b4.1)
> colMeans(b4.1_post)
(Intercept)       sigma
 154.612373    7.775306
>
> # variance-covariance matrix
> var(b4.1_post)
             (Intercept)         sigma
(Intercept)  0.1697857572 -0.0001626524
sigma       -0.0001626524  0.0833826418
>
> # draw samples from a multi-dimensional Gaussian distribution
> # with means and variance-covariance matrix based on the model
> library(MASS)
> post <- mvrnorm( n=1e4 , mu=colMeans(b4.1_post) , Sigma=var(b4.1_post) )
> head(post)
     (Intercept)    sigma
[1,]    155.1986 7.845216
[2,]    154.7201 6.970247
[3,]    154.2390 8.342064
[4,]    155.3467 7.776605
[5,]    154.3182 7.776134
[6,]    155.0219 7.945837
>
> # plot the samples from the multi-dimensional Gaussian distribution
> jitt <- runif(n = dim(post)[1], min = -0.075, max = 0.075)
> plot( post[,"(Intercept)"] + jitt , post[,"sigma"] + jitt , cex=0.5 ,
+       col=col.alpha(rangi2,0.1) ,
+       xlab="post.mu" , ylab="post.sigma" , pch=16 )
```

Multivariate Gaussian distribution

alpha and beta (centered)

Gaussian model of adult height based on centered weight

Now let's look at how **height** in these Kalahari foragers (the outcome variable) *covaries* with centered **weight** (the predictor variable)

How do we get weight into a Gaussian model of height? Let x be the name for the column of weight measurements. Let the average of the x values be xbar

Now we have a predictor variable x, which is a list of measures of the same length as height, h. To get weight into the model, we define the mean μ as a function of the values of x

Let's write down the general model and compute the plausibility of each combination of α , β , and σ:

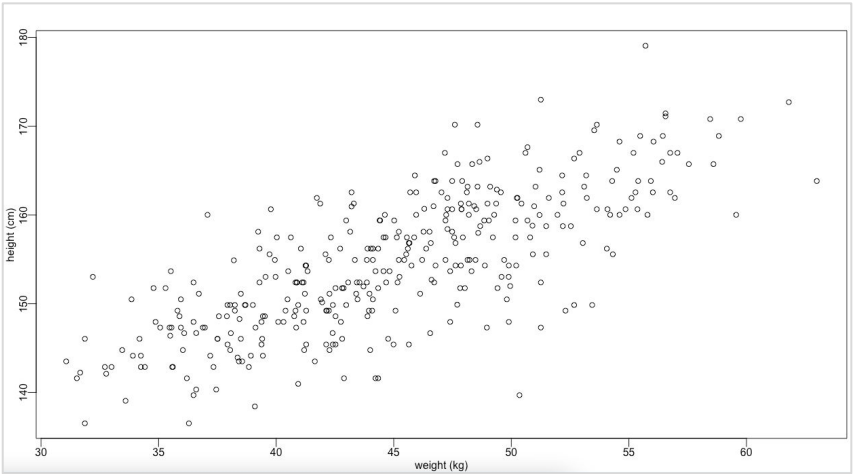
$h_i \sim \text{Normal}(\mu_i, \sigma)$ [likelihood]

$\mu_i = \alpha + \beta(x_i - \text{xbar})$ [linear model]

$\alpha \sim \text{Normal}(178, 20)$ [α prior]

$\beta \sim \text{Normal}(0, 10)$ [β prior]

$\sigma \sim \text{Uniform}(0, 50)$ [σ prior]



Gaussian model of adult height based on centered weight

Now let us fit this model in rethinking and rstanarm code:

$h_i \sim \text{Normal}(\mu_i, \sigma)$ [height ~dnorm(mu,sigma)]

$\mu_i = \alpha + \beta(x_i - \text{x\_bar})$ [mu ← a + b*(weight – xbar)]

$\alpha \sim \text{Normal}(178, 20)$ [a ~dnorm(178, 20)]

$\beta \sim \text{Normal}(0, 10)$ [b ~dnorm(0, 1)]

$\sigma \sim \text{Uniform}(0, 50)$ [sigma ~dunif(0,50)]

The summaries of the posterior distribution provide Gaussian approximations for each parameter's *marginal* distribution:

Intercept (α):
The plausibility of each value of α, after averaging over the plausibility of each value of β and σ, is given by a Gaussian distribution with mean **154.6** and standard deviation **0.3**.

Slope (β):
The plausibility of each value of β, after averaging over the plausibility of each value of α and σ, is given by a Gaussian distribution with mean **0.9**

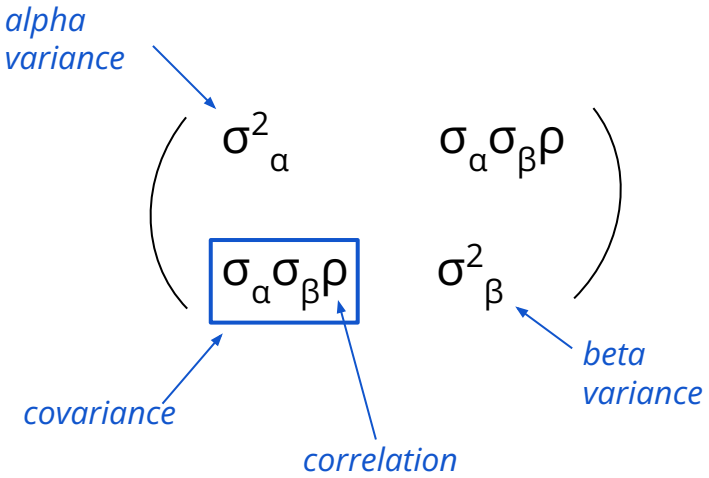
```
> library(rethinking)
> data(Howell1)
> d <- Howell1
> d2 <- d[ d$age >= 18 , ]
> xbar <- mean(d2$weight)
>
> m4.3 <- quap( alist(
+   height ~ dnorm( mu , sigma ) ,
+   mu <- a + b*( weight - xbar ) ,
+   a ~ dnorm( 178 , 20 ) ,
+   b ~ dnorm( 0 , 1 ) ,
+   sigma ~ dunif( 0 , 50 )
+ ) , data=d2 )
>
> precis(m4.3)
        mean   sd  5.5%  94.5%
a     154.60 0.27 154.17 155.03
b       0.90 0.04   0.84   0.97
sigma   5.07 0.19   4.77   5.38
```

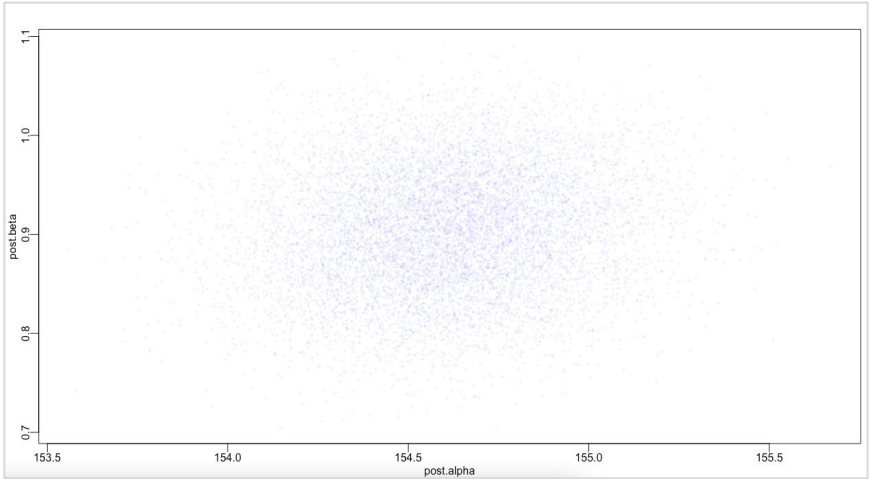
```
> weight_c <- d2$weight - mean(d2$weight)
>
> b4.3 <- stan_glm(height ~ 1 + weight_c, data = d2,
+                  prior_intercept = normal( location = c(178) , scale = c(20) ) ,
+                  prior = normal( location = c(0) , scale = c(1) ) ,
+                  prior_aux = NULL, refresh = 0)
>
> summary(b4.3)

Model Info:
 function:     stan_glm
 family:       gaussian [identity]
 formula:      height ~ 1 + weight_c
 algorithm:    sampling
 sample:       4000 (posterior sample size)
 priors:       see help('prior_summary')
 observations: 352
 predictors:   2

Estimates:
              mean   sd   10%   50%   90%
(Intercept) 154.6    0.3 154.2 154.6 154.9
weight_c      0.9    0.0   0.9   0.9   1.0
sigma         5.1    0.2   4.9   5.1   5.3
```

Quadratic approximation with more than one parameter (α and β) is a multivariate Gaussian distribution

2-dimensional Gaussian distribution:

- vector of means
- variance-covariance matrix

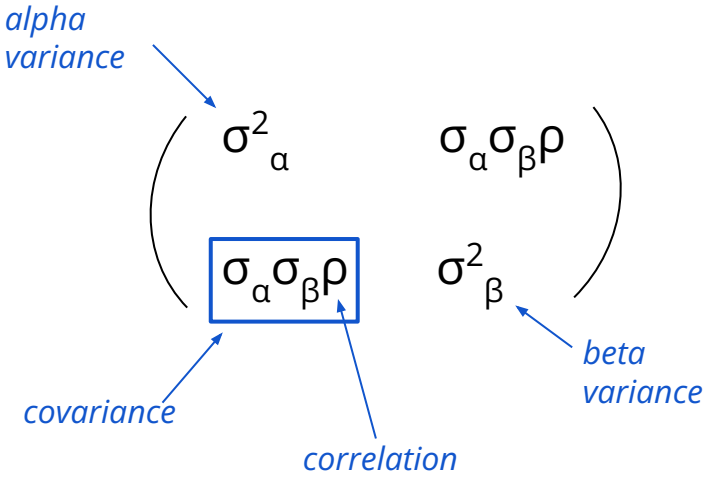


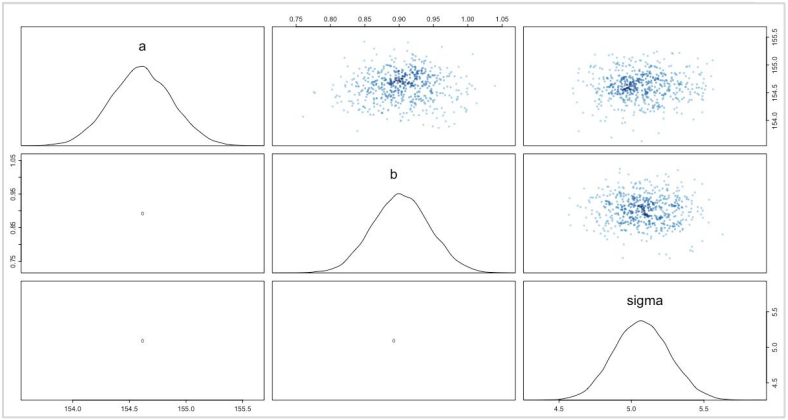


Quadratic approximation with more than one parameter (α and β) is a multivariate Gaussian distribution

2-dimensional Gaussian distribution:

- vector of means
- variance-covariance matrix





rethinking

```
> # variance-covariance matrix
> round( vcov( m4.3 ) , 3)
          a     b sigma
a     0.073 0.000 0.000
b     0.000 0.002 0.000
sigma 0.000 0.000 0.037
>
> # a vector of variances for the parameters
> round( diag( vcov( m4.3 ) ) , 3 )
    a     b sigma
0.073 0.002 0.037
>
> # a correlation matrix that tells how changes
> # in any parameter lead to correlated changes in the others
> round( cov2cor( vcov( m4.3 ) ) , 3 )
          a      b  sigma
a     1.000  0.000  0.001
b     0.000  1.000 -0.003
sigma 0.001 -0.003  1.000
```

rstanarm

```
> # extract posterior distribution as a matrix
> b4.3_post <- as.matrix(b4.3)
>
> # variance-covariance matrix
> round( var(b4.3_post) , 3 )
             (Intercept) weight_c  sigma
(Intercept)        0.075    0.000 -0.001
weight_c           0.000    0.002  0.000
sigma             -0.001    0.000  0.036
>
> # a vector of variances for the parameters
> round( diag( var(b4.3_post) ) , 4 )
(Intercept)    weight_c       sigma
     0.0748      0.0017      0.0364
>
> # a correlation matrix that tells how changes
> # in any parameter lead to correlated changes in the others
> round( cov2cor( var(b4.3_post) ) , 4 )
             (Intercept) weight_c   sigma
(Intercept)       1.0000  -0.0090 -0.0103
weight_c         -0.0090   1.0000  0.0214
sigma            -0.0103   0.0214  1.0000
```

Sample from multivariate Gaussian distribution

Now let us sample vectors of values from a multivariate Gaussian distribution, with its vector of means and variance-covariance matrix based on the model's fitted values:

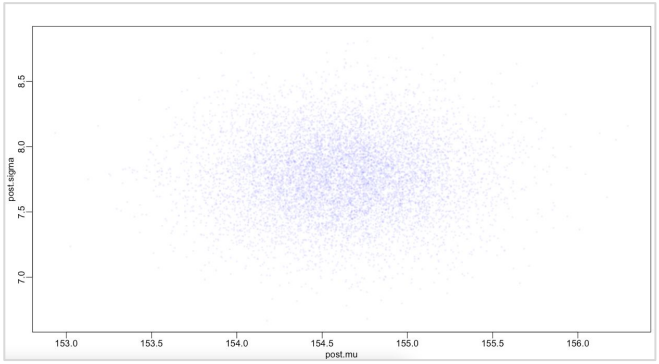
$$\mathbf{X} \sim \text{Normal}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$$

$$\boldsymbol{\mu} = [\, 154.6 \,,\, 0.9 \,,\, 5.1 \,]^{T}$$

$$\boldsymbol{\Sigma} = \begin{matrix} 0.073 & 0.000 & 0.000 \\ 0.000 & 0.002 & 0.000 \\ 0.000 & 0.000 & 0.037 \end{matrix}$$

These samples preserve the covariance between α and β

This hardly matters right now, because α and β don't covary in this model. But once we *uncenter* the predictor variable, covariance will matter a lot



rethinking

```
> # vector of means
> round( coef(m4.3) , 2)
     a      b  sigma
154.60   0.90   5.07
>
> # variance-covariance matrix
> round( vcov(m4.3) , 3 )
            a     b sigma
a       0.073 0.000 0.000
b       0.000 0.002 0.000
sigma   0.000 0.000 0.037
>
> # draw samples from a multi-dimensional Gaussian distribution
> # with means and variance-covariance matrix based on the model
> library(rethinking)
> post <- extract.samples( m4.3 , n=1e4 )
> head(post)
         a         b    sigma
1 154.4015 0.8864317 5.372173
2 154.6159 0.8480212 5.277300
3 154.7560 0.8885167 5.195976
4 154.8498 0.9130208 5.327713
5 154.4406 0.8906902 5.216519
6 154.8106 0.9306013 5.126488
>
> # plot the samples from the multi-dimensional Gaussian distribution
> jitt <- runif(n = dim(post)[1], min = -0.075, max = 0.075)
> plot( post$a + jitt , post$b + jitt , cex=0.5 ,
+       col=col.alpha(rangi2,0.1) ,
+       xlab="post.alpha" , ylab="post.beta" , pch=16 )
```

rstanarm

```
> b4.3_post <- as.matrix(b4.3)
> colMeans(b4.3_post)
(Intercept)    weight_c       sigma
154.5993152   0.9040244   5.1033971
>
> # variance-covariance matrix
> round( var(b4.3_post) , 3 )
            (Intercept) weight_c  sigma
(Intercept)       0.075    0.000 -0.001
weight_c          0.000    0.002  0.000
sigma            -0.001    0.000  0.036
>
> # draw samples from a multi-dimensional Gaussian distribution
> # with means and variance-covariance matrix based on the model
> library(MASS)
> post <- mvrnorm( n=1e4 , mu=colMeans(b4.3_post) , Sigma=var(b4.3_post) )
> head(post)
     (Intercept)  weight_c    sigma
[1,]    154.3244 0.8753303 5.341038
[2,]    154.1357 0.9537212 5.332868
[3,]    154.2045 0.9109135 5.384641
[4,]    154.3314 0.8611746 5.191895
[5,]    154.5664 0.9325227 5.136812
[6,]    154.0234 0.9158578 5.053935
> # plot the samples from the multi-dimensional Gaussian distribution
> jitt <- runif(n = dim(post)[1], min = -0.075, max = 0.075)
> plot( post[,"(Intercept)"] + jitt , post[,"weight_c"] + jitt , cex=0.5 ,
+       col=col.alpha(rangi2,0.1) ,
+       xlab="post.alpha" , ylab="post.beta" , pch=16 )
```

Multivariate Gaussian distribution

alpha and beta

Gaussian model of adult height based on weight

Now let us fit this model in rethinking and rstanarm code:

$h_i \sim \text{Normal}(\mu_i, \sigma)$	[height ~dnorm(mu,sigma)]
$\mu_i = \alpha + \beta x_i$	[mu ← a + b*weight]
$\alpha \sim \text{Normal}(178, 20)$	[a ~dnorm(178, 20)]
$\beta \sim \text{Normal}(0, 10)$	[b ~dnorm(0, 1)]
$\sigma \sim \text{Uniform}(0, 50)$	[sigma ~dunif(0,50)]

```
> m4M7 <- quap(
+   alist(
+     height ~ dnorm( mu , sigma ) ,
+     mu <- a + b * weight ,
+     a ~ dnorm( 178 , 20 ) ,
+     b ~ dnorm( 0 , 1 ) ,
+     sigma ~ dunif( 0 , 50 )
+ ) , data=d2 )
>
> precis(m4M7)
        mean   sd  5.5%  94.5%
a     114.53 1.90 111.50 117.56
b       0.89 0.04   0.82   0.96
sigma   5.07 0.19   4.77   5.38
```

```
> b4M7 <- stan_glm(height ~ 1 + weight, data = d2,
+                  prior_intercept = normal( location = c(178) , scale = c(20) ) ,
+                  prior = normal( location = c(0) , scale = c(1) ) ,
+                  prior_aux = NULL, refresh = 0)
>
> summary(b4M7)

Model Info:
 function:     stan_glm
 family:       gaussian [identity]
 formula:      height ~ 1 + weight
 algorithm:    sampling
 sample:       4000 (posterior sample size)
 priors:       see help('prior_summary')
 observations: 352
 predictors:   2

Estimates:
              mean   sd   10%   50%   90%
(Intercept) 114.0    1.9 111.6 113.9 116.3
weight        0.9    0.0   0.8   0.9   1.0
sigma         5.1    0.2   4.9   5.1   5.4
```

The summaries of the posterior distribution provide Gaussian approximations for each parameter's *marginal* distribution:

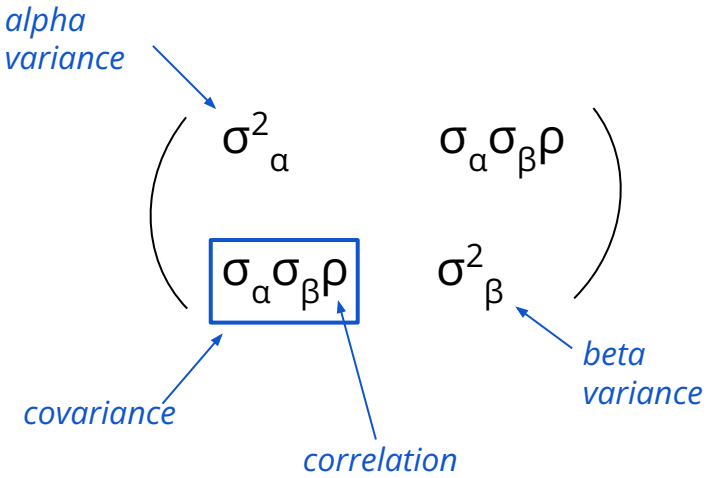
Intercept (α):
The plausibility of each value of α, after averaging over the plausibility of each value of β and σ, is given by a Gaussian distribution with mean **114.5** and standard deviation **1.9**.

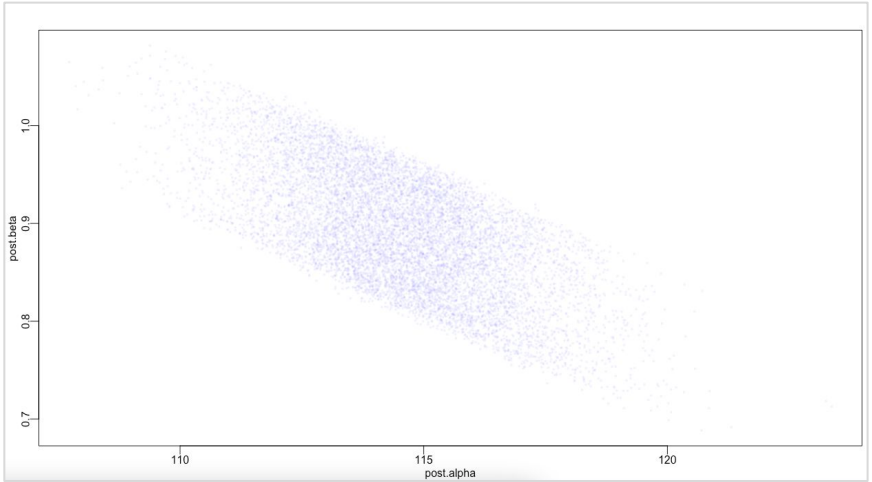
Slope (β):
The plausibility of each value of β, after averaging over the plausibility of each value of α and σ, is given by a Gaussian distribution with mean **0.9** and standard deviation **0.04**.

Quadratic approximation with more than one parameter (α and β) is a multivariate Gaussian distribution

2-dimensional Gaussian distribution:

- vector of means
- variance-covariance matrix

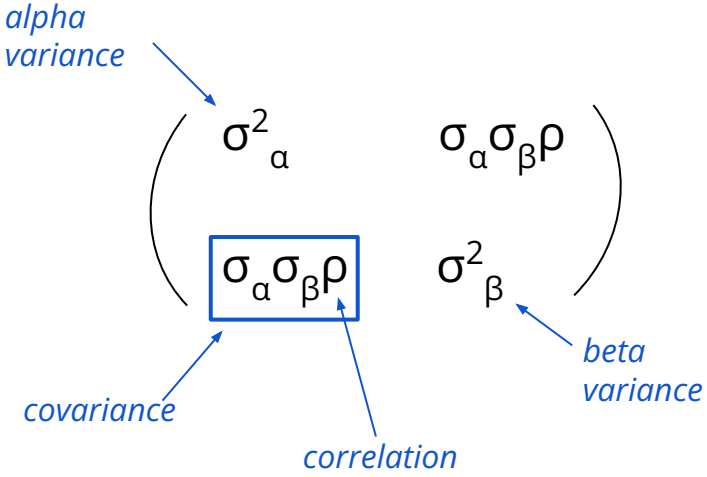


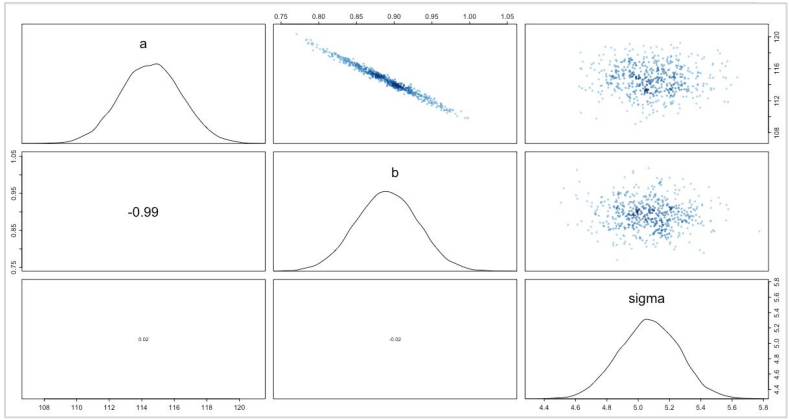


Quadratic approximation with more than one parameter (α and β) is a multivariate Gaussian distribution

2-dimensional Gaussian distribution:

- vector of means
- variance-covariance matrix





rethinking

```
> # variance-covariance matrix
> round( vcov( m4M7 ) , 3)
          a      b sigma
a     3.596 -0.078 0.009
b    -0.078  0.002 0.000
sigma 0.009  0.000 0.037
>
> # a vector of variances for the parameters
> round( diag( vcov( m4M7 ) ) , 3 )
    a     b sigma
3.596 0.002 0.037
>
> # a correlation matrix that tells how changes
> # in any parameter lead to correlated changes in the others
> round( cov2cor( vcov( m4M7 ) ) , 3 )
          a      b  sigma
a     1.000 -0.990  0.025
b    -0.990  1.000 -0.025
sigma 0.025 -0.025  1.000
```

rstanarm

```
> # extract posterior distribution as a matrix
> b4M7_post <- as.matrix(b4M7)
>
> # variance-covariance matrix
> round( var(b4M7_post) , 3 )
             (Intercept) weight sigma
(Intercept)        3.549 -0.077 0.001
weight            -0.077  0.002 0.000
sigma              0.001  0.000 0.038
>
> # a vector of variances for the parameters
> round( diag( var(b4M7_post) ) , 4 )
(Intercept)      weight       sigma
     3.5490      0.0017      0.0385
>
> # a correlation matrix that tells how changes
> # in any parameter lead to correlated changes in the others
> round( cov2cor( var(b4M7_post) ) , 4 )
             (Intercept)  weight   sigma
(Intercept)       1.0000 -0.9897  0.0022
weight           -0.9897  1.0000 -0.0068
sigma             0.0022 -0.0068  1.0000
```

Sample from multivariate Gaussian distribution

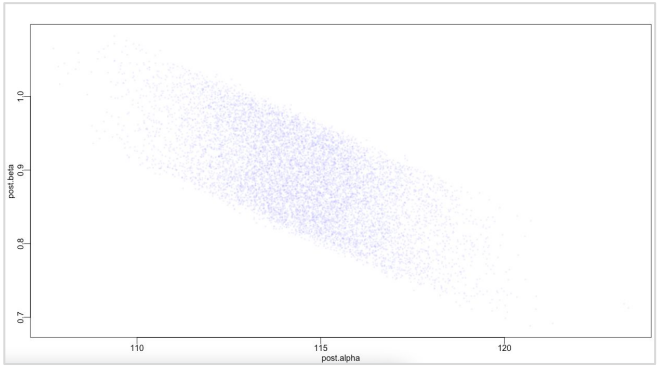
Now let us sample vectors of values from a multivariate Gaussian distribution, with its vector of means and variance-covariance matrix based on the model's fitted values:

$$\mathbf{X} \sim \text{Normal}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$$

$$\boldsymbol{\mu} = [\, 114.5 \,,\, 0.9 \,,\, 5.1 \,]^{\mathsf{T}}$$

$$\boldsymbol{\Sigma} = \begin{bmatrix} 3.596 & -0.078 & 0.009 \\ -0.078 & 0.002 & 0.000 \\ 0.009 & 0.000 & 0.037 \end{bmatrix}$$

These samples preserve the covariance between α and β



rethinking

```
> # vector of means
> round( coef(m4M7) , 2 )
     a      b  sigma
114.53   0.89   5.07
>
> # variance-covariance matrix
> round( vcov(m4M7) , 3 )
             a      b sigma
a        3.596 -0.078 0.009
b       -0.078  0.002 0.000
sigma    0.009  0.000 0.037
>
> # draw samples from a multi-dimensional Gaussian distribution
> # with means and variance-covariance matrix based on the model
> library(rethinking)
> post <- extract.samples( m4M7 , n=1e4 )
> head(post)
         a         b    sigma
1 113.4917 0.9075525 5.295032
2 115.2126 0.8789498 5.334145
3 114.6524 0.8978084 4.975320
4 111.9071 0.9473981 5.161275
5 112.2090 0.9419364 5.200980
6 113.8611 0.9026463 5.150633
>
> # plot the samples from the multi-dimensional Gaussian distribution
> jitt <- runif(n = dim(post)[1], min = -0.075, max = 0.075)
> plot( post$a + jitt , post$b + jitt , cex=0.5 ,
+       col=col.alpha(rangi2,0.1) ,
+       xlab="post.alpha" , ylab="post.beta" , pch=16 )
```

rstanarm

```
> # vector of means
> b4M7_post <- as.matrix(b4M7)
> round( colMeans(b4M7_post) , 2 )
(Intercept)      weight       sigma
     113.97        0.90        5.11
>
> # variance-covariance matrix
> round( var(b4M7_post) , 3 )
             (Intercept) weight sigma
(Intercept)        3.549 -0.077 0.001
weight            -0.077  0.002 0.000
sigma              0.001  0.000 0.038
>
> # draw samples from a multi-dimensional Gaussian distribution
> # with means and variance-covariance matrix based on the model
> library(MASS)
> post <- mvrnorm( n=1e4 , mu=colMeans(b4M7_post) , Sigma=var(b4M7_post) )
> head(post)
     (Intercept)    weight    sigma
[1,]    111.8324 0.9530824 4.919729
[2,]    117.5555 0.8289016 4.925537
[3,]    113.3016 0.9146623 5.149051
[4,]    110.3910 0.9756269 5.019545
[5,]    112.2841 0.9408032 5.228960
[6,]    110.9838 0.9587446 4.835497
>
> # plot the samples from the multi-dimensional Gaussian distribution
> jitt <- runif(n = dim(post)[1], min = -0.075, max = 0.075)
> plot( post[,"(Intercept)"] + jitt , post[,"weight"] + jitt , cex=0.5 ,
+       col=col.alpha(rangi2,0.1) ,
+       xlab="post.alpha" , ylab="post.beta" , pch=16 )
```

Multivariate Gaussian distribution

multilevel models
alpha and beta

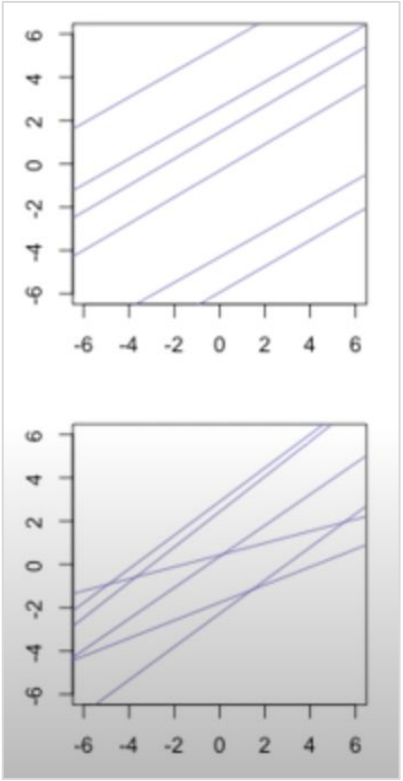
Multilevel model: varying intercepts and varying slopes

Varying ***intercepts***: means differ by cluster

Varying ***slopes***: effects of predictors vary by cluster

Any parameter can be made into a varying effect:

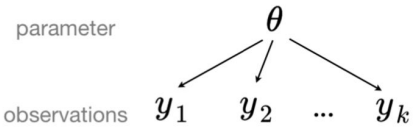
- split into vector of parameters by cluster
- define population distribution



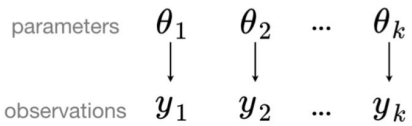
Complete pooling and no pooling are unsatisfactory options

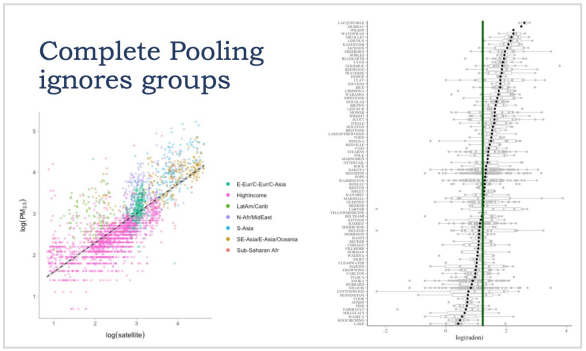
Two unsatisfactory options for handling groups

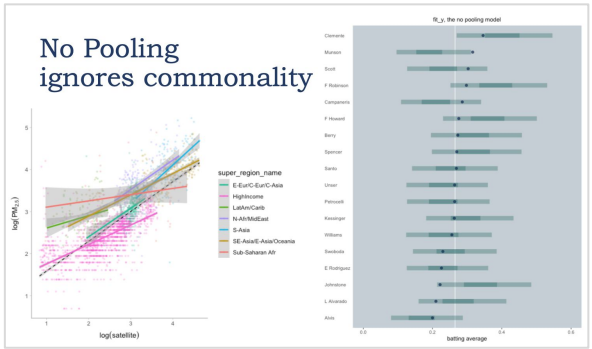
- Complete pooling



- No pooling





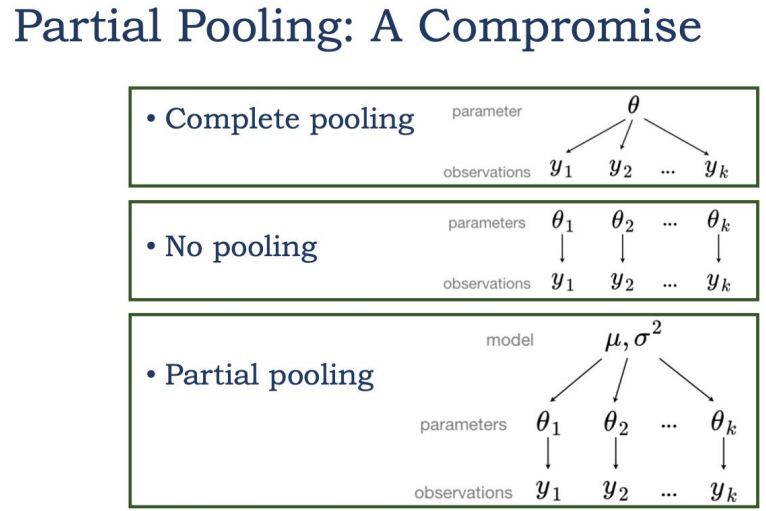


Multilevel model: population pooling

Major innovation with varying intercepts and varying slopes is **pooling across parameters (intercepts & slopes)**

Features of units have correlation structure

Learn one feature → info about other features



Cafe Robots example

Robots programmed to visit cafes, order coffee, and record wait time

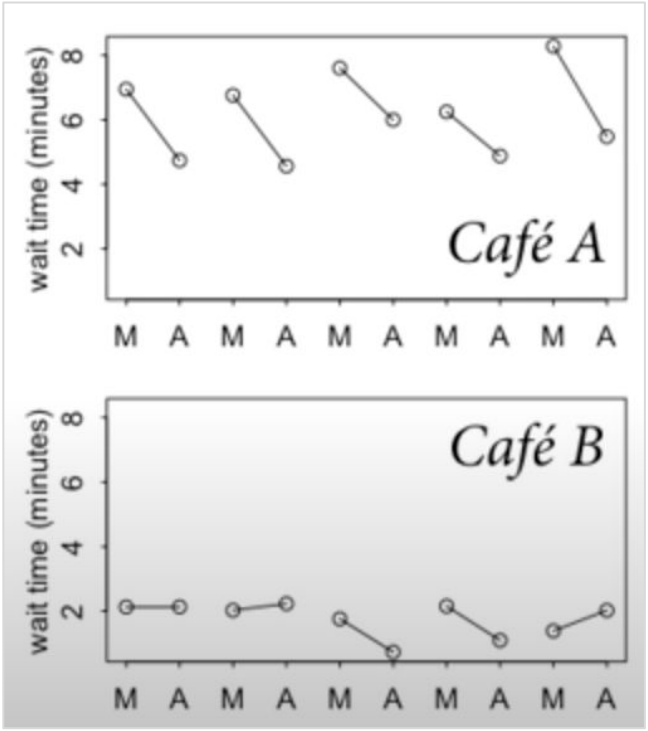
Visits in *morning* and *afternoon*

Intercepts: avg morning wait

Slopes: avg difference between afternoon and morning

Are intercepts and slopes related?

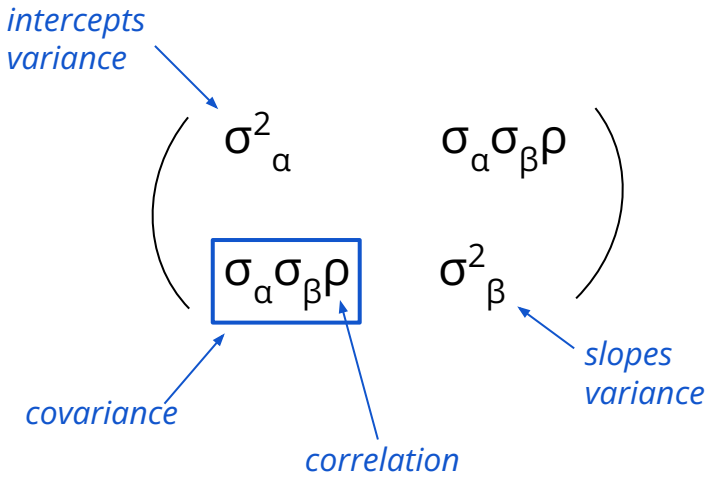
- Yes => pooling across parameter types

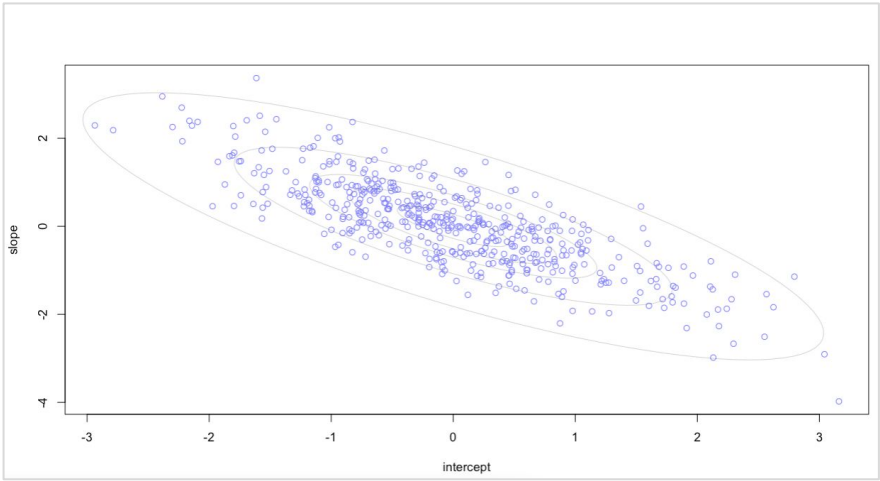


Population from Multivariate Gaussian distribution

2-dimensional Gaussian distribution:

- vector of means
- variance-covariance matrix





Covariance matrix

k-by-k covariance matrix requires estimating

- k standard deviations (or variances)
- $(k^2 - k) / 2$ correlations (for covariances)
- total of k(k+1)/2 parameters

$$S = \begin{pmatrix} \sigma^2_\alpha & \sigma_\alpha\sigma_\beta\rho \\ \sigma_\alpha\sigma_\beta\rho & \sigma^2_\beta \end{pmatrix} = \begin{pmatrix} \sigma^2_\alpha & 0 \\ 0 & \sigma_\beta \end{pmatrix} \underbrace{\begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix}}_{\mathbf{R}} \begin{pmatrix} \sigma^2_\alpha & 0 \\ 0 & \sigma_\beta \end{pmatrix}$$

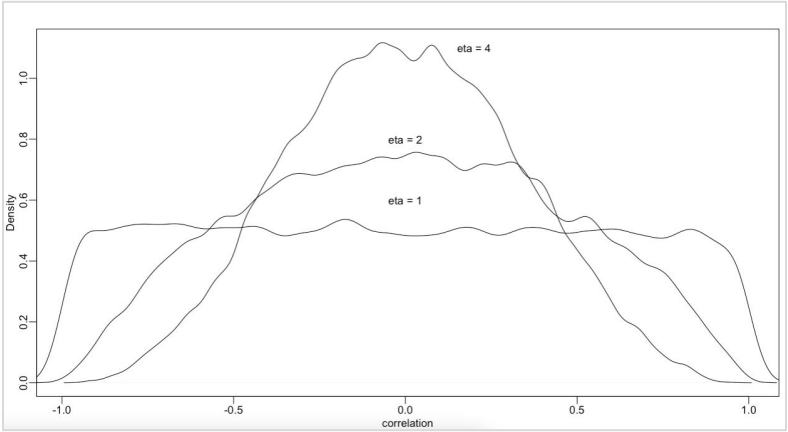


Figure: LKJcorr(eta) probability density. The plot shows the distribution of correlation coefficients extracted from random 2-by-2 correlation matrices, for three values of eta. When eta = 1, all correlations are equally plausible. As eta increases, extreme correlations become less plausible.

LKJcorr(2) defines a weakly informative prior on rho that is skeptical of extreme correlations near –1 or 1. You can think of it as a regularizing prior for correlation matrices. The distribution has a single parameter, eta, that controls how skeptical the prior is of large correlations in the matrix

Varying intercepts, varying slopes model

$$W_i \sim \text{Normal}(\mu_i, \sigma)$$

$$\mu_i = \alpha_{\text{cafe}[i]} + \beta_{\text{cafe}[i]} A_i$$

$$\begin{bmatrix} \alpha_{\text{cafe}[i]} \\ \beta_{\text{cafe}[i]} \end{bmatrix} \sim \text{MVNormal}([\alpha, \beta]^T , \mathbf{S})$$

$$S = \begin{pmatrix} \sigma_\alpha & 0 \\ 0 & \sigma_\beta \end{pmatrix} \begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix} \begin{pmatrix} \sigma_\alpha & 0 \\ 0 & \sigma_\beta \end{pmatrix}$$

$\alpha \sim \text{Normal}(5, 2)$

$\beta \sim \text{Normal}(-1, 0.5)$

$\sigma \sim \text{Exponential}(1)$

$\sigma_\alpha \sim \text{Exponential}(1)$

$\sigma_\beta \sim \text{Exponential}(1)$

$\mathbf{R} \sim \text{LKJcorr}(2)$

Varying intercepts, varying slopes model

varying intercepts

varying slopes

$$W_i \sim \text{Normal}(\mu_i, \sigma)$$

$$\mu_i = \alpha_{\text{cafe}[i]} + \beta_{\text{cafe}[i]} A_i$$

$$\begin{bmatrix} \alpha_{\text{cafe}[i]} \\ \beta_{\text{cafe}[i]} \end{bmatrix} \sim \text{MVNormal}([\alpha, \beta]^T , \mathbf{S})$$

$$\mathbf{S} = \begin{pmatrix} \sigma_\alpha & 0 \\ 0 & \sigma_\beta \end{pmatrix} \begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix} \begin{pmatrix} \sigma_\alpha & 0 \\ 0 & \sigma_\beta \end{pmatrix}$$

$\alpha \sim \text{Normal}(5, 2)$

$\beta \sim \text{Normal}(-1, 0.5)$

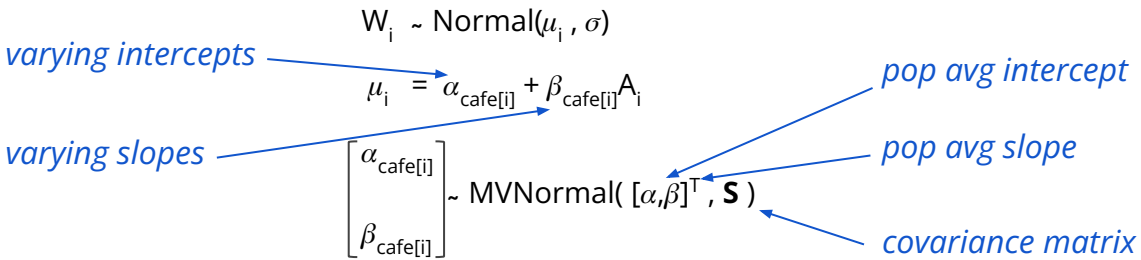
$\sigma \sim \text{Exponential}(1)$

$\sigma_\alpha \sim \text{Exponential}(1)$

$\sigma_\beta \sim \text{Exponential}(1)$

$\mathbf{R} \sim \text{LKJcorr}(2)$

Varying intercepts, varying slopes model



$\alpha \sim \text{Normal}(5, 2)$

$\beta \sim \text{Normal}(-1, 0.5)$

$\sigma \sim \text{Exponential}(1)$

$\sigma_\alpha \sim \text{Exponential}(1)$

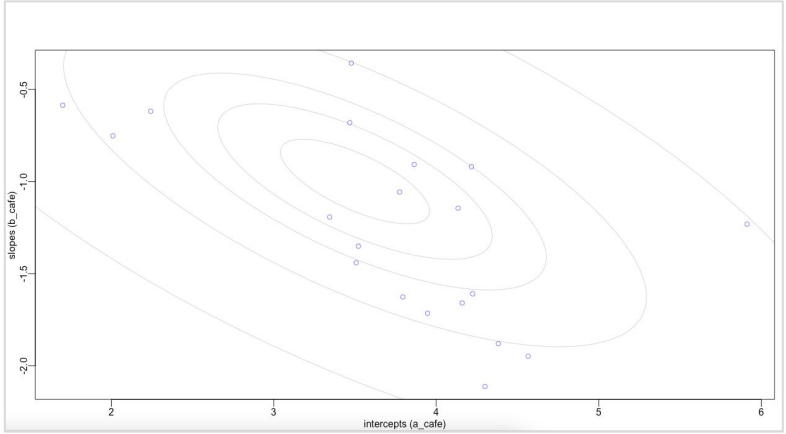
$\sigma_\beta \sim \text{Exponential}(1)$

$\mathbf{R} \sim \text{LKJcorr}(2)$

Define the population

Let's define the *average* properties of the cafes, and then sample cafes from them:

```
a ← 3.5          # average morning wait time
b ← (–1)         # average difference afternoon wait time
sigma_a ← 1      # std dev in intercepts
sigma_b ← 0.5    # std dev in slopes
rho ← (–0.7)     # correlation between intercepts and slopes
sigma ← 0.5      # std dev within cafes
```




```
> # Simulate observations
> # i.e. simulate our robot visiting
> # these cafes and collecting data
> set.seed(22)
> N_visits <- 10
> afternoon <- rep(0:1,N_visits*N_cafes/2)
> cafe_id <- rep( 1:N_cafes , each=N_visits )
> mu <- a_cafe[cafe_id] + b_cafe[cafe_id]*afternoon
> sigma <- 0.5  # std dev within cafes
> wait <- rnorm( N_visits*N_cafes , mu , sigma )
> d <- data.frame( cafe=cafe_id , afternoon=afternoon , wait=wait )
> head(d)
  cafe afternoon     wait
1    1         0 3.967893
2    1         1 3.857198
3    1         0 4.727875
4    1         1 2.761013
5    1         0 4.119483
6    1         1 3.543652
> tail(d)
    cafe afternoon     wait
195   20         0 3.569120
196   20         1 2.109806
197   20         0 3.790041
198   20         1 2.837626
199   20         0 3.784056
200   20         1 3.634059
```

Fitting complete pooling, no pooling, and multilevel partial pooling models with **rethinking**

Now let's see how to specify these regression models in R:

```
# Fit a complete pooling model with `rethinking::quap()`
m14.1_pool <- quap(
  alist(
    wait ~ dnorm( mu , sigma ),
    mu <- a + b*afternoon,
    a ~ dnorm(5, 2),
    b ~ dnorm(-1, 0.5),
    sigma ~ dexp(1)
  ) , data=d )

precis(m14.1_pool)


# Fit a no pooling model with `rethinking::quap()`
m14.1_no_pool <- quap(
  alist(
    wait ~ dnorm( mu , sigma ),
    mu <- a[cafe] + b*afternoon,
    a[cafe] ~ dnorm(5, 2),
    b ~ dnorm(-1, 0.5),
    sigma ~ dexp(1)
  ) , data=d )

precis(m14.1_no_pool)

precis(m14.1_no_pool, depth=2)
```

```
# Fit a multilevel model with `rethinking::ulam()`
m14.1 <- ulam(
  alist(
    wait ~ normal( mu , sigma ),
    mu <- a_cafe[cafe] + b_cafe[cafe]*afternoon,
    c(a_cafe,b_cafe)[cafe] ~ multi_normal( c(a,b) , Rho , sigma_cafe ),
    a ~ normal(5,2),
    b ~ normal(-1,0.5),
    sigma_cafe ~ exponential(1),
    sigma ~ exponential(1),
    Rho ~ lkjcorr(2)
  ) , data=d , chains=4 , cores=4 )

precis(m14.1)

precis(m14.1, depth=2)

precis(m14.1, depth=3)
```

```
> precis(m14.1, depth=3)
             mean   sd  5.5% 94.5% n_eff Rhat4
b_cafe[1]   -1.15 0.26 -1.58 -0.73  4920     1
b_cafe[2]   -0.90 0.27 -1.33 -0.46  3935     1
b_cafe[3]   -1.94 0.28 -2.37 -1.49  4355     1
b_cafe[4]   -1.24 0.26 -1.65 -0.82  4218     1
b_cafe[5]   -0.14 0.28 -0.57  0.30  3782     1
b_cafe[6]   -1.31 0.26 -1.71 -0.89  5053     1
b_cafe[7]   -1.02 0.26 -1.43 -0.61  3968     1
b_cafe[8]   -1.63 0.26 -2.05 -1.22  4439     1
b_cafe[9]   -1.31 0.26 -1.72 -0.90  3916     1
b_cafe[10]  -0.95 0.27 -1.38 -0.53  4639     1
b_cafe[11]  -0.43 0.27 -0.86  0.00  4636     1
b_cafe[12]  -1.19 0.27 -1.62 -0.76  4667     1
b_cafe[13]  -1.81 0.28 -2.26 -1.37  4099     1
b_cafe[14]  -0.93 0.26 -1.36 -0.52  4733     1
b_cafe[15]  -2.19 0.29 -2.66 -1.74  3182     1
b_cafe[16]  -1.04 0.26 -1.44 -0.63  4546     1
b_cafe[17]  -1.22 0.26 -1.62 -0.80  4477     1
b_cafe[18]  -1.02 0.28 -1.47 -0.58  3884     1
b_cafe[19]  -0.26 0.29 -0.72  0.20  4339     1
b_cafe[20]  -1.06 0.26 -1.48 -0.64  4867     1
a_cafe[1]    4.21 0.20  3.90  4.53  5008     1
a_cafe[2]    2.15 0.21  1.83  2.48  3928     1
a_cafe[3]    4.38 0.20  4.05  4.71  4400     1
a_cafe[4]    3.25 0.20  2.93  3.57  4124     1
a_cafe[5]    1.87 0.20  1.55  2.20  3923     1
a_cafe[6]    4.26 0.20  3.96  4.58  4965     1
a_cafe[7]    3.61 0.20  3.29  3.94  4371     1
a_cafe[8]    3.94 0.20  3.62  4.25  4692     1
a_cafe[9]    3.98 0.19  3.67  4.29  4592     1
a_cafe[10]   3.56 0.20  3.23  3.87  4494     1
a_cafe[11]   1.93 0.20  1.61  2.26  4493     1
a_cafe[12]   3.84 0.21  3.52  4.17  4751     1
a_cafe[13]   3.88 0.20  3.56  4.21  4264     1
a_cafe[14]   3.17 0.20  2.85  3.48  4520     1
a_cafe[15]   4.45 0.20  4.13  4.78  3482     1
a_cafe[16]   3.39 0.20  3.08  3.70  4817     1
a_cafe[17]   4.21 0.20  3.89  4.53  4531     1
a_cafe[18]   5.75 0.20  5.43  6.08  4050     1
a_cafe[19]   3.25 0.21  2.92  3.58  4444     1
a_cafe[20]   3.74 0.20  3.42  4.05  4842     1
a            3.65 0.22  3.31  4.00  4203     1
b           -1.14 0.14 -1.36 -0.90  3868     1
sigma_cafe[1]  0.96 0.17  0.73  1.24  3323     1
sigma_cafe[2]  0.59 0.13  0.41  0.81  2678     1
sigma        0.47 0.03  0.43  0.52  3272     1
Rho[1,1]     1.00 0.00  1.00  1.00   NaN   NaN
Rho[1,2]    -0.50 0.18 -0.76 -0.18  3701     1
Rho[2,1]    -0.50 0.18 -0.76 -0.18  3701     1
Rho[2,2]     1.00 0.00  1.00  1.00   NaN   NaN
```

Quickly fitting complete pooling, no pooling, and multilevel partial pooling models in R

Now let's see how to specify these regression models in R:

```
# Fit a complete pooling model with `lm`
arm::display( lm( wait ~ 1 + afternoon , data = d) )


# Fit a no pooling model with `lm`
arm::display( lm( wait ~ afternoon + factor(cafe_id) - 1 , data = d) )
```

```
# Fit a multilevel model with `Matrix::lmer()`
lmer14.1 <- lmer( wait ~ 1 + afternoon + (1 + afternoon | cafe_id) , data = d)
arm::display(lmer14.1)
```

```
> head(d)
  cafe afternoon     wait
1    1         0 3.967893
2    1         1 3.857198
3    1         0 4.727875
4    1         1 2.761013
5    1         0 4.119483
6    1         1 3.543652
> tail(d)
    cafe afternoon     wait
195   20         0 3.569120
196   20         1 2.109806
197   20         0 3.790041
198   20         1 2.837626
199   20         0 3.784056
200   20         1 3.634059
```

```
> # Fit a complete pooling model with `lm`
> arm::display( lm( wait ~ 1 + afternoon , data = d) )
lm(formula = wait ~ 1 + afternoon, data = d)
            coef.est coef.se
(Intercept)  3.64     0.10
afternoon   -1.14     0.13
---
n = 200, k = 2
residual sd = 0.95, R-Squared = 0.27
>
>
> # Fit a no pooling model with `lm`
> arm::display( lm( wait ~ afternoon + factor(cafe_id) - 1 , data = d) )
lm(formula = wait ~ afternoon + factor(cafe_id) - 1, data = d)
                 coef.est coef.se
afternoon        -1.14     0.08
factor(cafe_id)1  4.23     0.18
factor(cafe_id)2  2.23     0.18
factor(cafe_id)3  3.98     0.18
factor(cafe_id)4  3.18     0.18
factor(cafe_id)5  2.34     0.18
factor(cafe_id)6  4.20     0.18
factor(cafe_id)7  3.68     0.18
factor(cafe_id)8  3.70     0.18
factor(cafe_id)9  3.90     0.18
factor(cafe_id)10 3.66     0.18
factor(cafe_id)11 2.24     0.18
factor(cafe_id)12 3.82     0.18
factor(cafe_id)13 3.54     0.18
factor(cafe_id)14 3.26     0.18
factor(cafe_id)15 3.92     0.18
factor(cafe_id)16 3.43     0.18
factor(cafe_id)17 4.19     0.18
factor(cafe_id)18 5.89     0.18
factor(cafe_id)19 3.70     0.18
factor(cafe_id)20 3.78     0.18
---
n = 200, k = 21
residual sd = 0.56, R-Squared = 0.97
>
>
> # Fit a multilevel model with `Matrix::lmer()`
> lmer14.1 <- lmer( wait ~ 1 + afternoon + (1 + afternoon | cafe_id) , data = d)
> arm::display(lmer14.1)
lmer(formula = wait ~ 1 + afternoon + (1 + afternoon | cafe_id),
    data = d)
            coef.est coef.se
(Intercept)  3.64     0.22
afternoon   -1.14     0.15

Error terms:
 Groups   Name        Std.Dev. Corr
 cafe_id  (Intercept) 0.95
          afternoon   0.59     -0.63
 Residual             0.47
---
number of obs: 200, groups: cafe_id, 20
AIC = 375.7, DIC = 356.2
deviance = 360.0
```

Fitting complete pooling, no pooling, and multilevel partial pooling models with **rstanarm**

Now let's see how to specify these regression models in R:

```
# Fit a complete pooling model with `rstanarm::stan_glm()`
b14.1_pool <- stan_glm( wait ~ 1 + afternoon ,
                        data = d , refresh=0)
summary( b14.1_pool , digits=2 )
tidy(b14.1_pool)
glance(b14.1_pool)


# Fit a no pooling model with `rstanarm::stan_glm()`
b14.1_no_pool <- stan_glm( wait ~ afternoon + factor(cafe_id) - 1 ,
                           data = d , refresh=0)
summary( b14.1_no_pool , digits=2 )
tidy(b14.1_no_pool)
glance(b14.1_no_pool)
```

```
# Fit a multilevel model with `rstanarm::stan_glm()`
b14.1 <- stan_glmer( wait ~ 1 + afternoon + (1 + afternoon | cafe_id) ,
                     data = d , refresh=0)
summary( b14.1 , digits=2 )
tidy(b14.1)
glance(b14.1)
```

```
> # Fit a multilevel model with `rstanarm::stan_glm()`
> b14.1 <- stan_glmer( wait ~ 1 + afternoon + (1 + afternoon | cafe_id) ,
+                      data = d , refresh=0)
> summary( b14.1 , digits=2 )

Model Info:
 function:     stan_glmer
 family:       gaussian [identity]
 formula:      wait ~ 1 + afternoon + (1 + afternoon | cafe_id)
 algorithm:    sampling
 sample:       4000 (posterior sample size)
 priors:       see help('prior_summary')
 observations: 200
 groups:       cafe_id (20)

Estimates:
                                          mean  sd    10%   50%   90%
(Intercept)                               3.63  0.22  3.36  3.64  3.91
afternoon                                -1.14  0.16 -1.34 -1.14 -0.94
b[(Intercept) cafe_id:1]                  0.58  0.29  0.22  0.58  0.95
b[afternoon cafe_id:1]                   -0.02  0.29 -0.40 -0.02  0.35
b[(Intercept) cafe_id:2]                 -1.48  0.29 -1.84 -1.48 -1.10
b[afternoon cafe_id:2]                    0.24  0.29 -0.13  0.24  0.62
b[(Intercept) cafe_id:3]                  0.74  0.29  0.37  0.74  1.12
b[afternoon cafe_id:3]                   -0.80  0.30 -1.18 -0.80 -0.42
b[(Intercept) cafe_id:4]                 -0.39  0.28 -0.75 -0.39 -0.03
b[afternoon cafe_id:4]                   -0.09  0.29 -0.45 -0.10  0.27
b[(Intercept) cafe_id:5]                 -1.76  0.30 -2.15 -1.76 -1.36
b[afternoon cafe_id:5]                    1.01  0.31  0.61  1.01  1.42
b[(Intercept) cafe_id:6]                  0.63  0.29  0.26  0.64  1.00
b[afternoon cafe_id:6]                   -0.18  0.29 -0.55 -0.17  0.19
b[(Intercept) cafe_id:7]                 -0.02  0.29 -0.39 -0.02  0.35
b[afternoon cafe_id:7]                    0.12  0.29 -0.26  0.11  0.48
b[(Intercept) cafe_id:8]                  0.31  0.30 -0.05  0.31  0.69
b[afternoon cafe_id:8]                   -0.49  0.30 -0.89 -0.49 -0.11
b[(Intercept) cafe_id:9]                  0.35  0.30 -0.03  0.35  0.73
b[afternoon cafe_id:9]                   -0.17  0.30 -0.56 -0.17  0.21
b[(Intercept) cafe_id:10]                -0.08  0.29 -0.45 -0.08  0.28
b[afternoon cafe_id:10]                   0.19  0.29 -0.17  0.18  0.57
b[(Intercept) cafe_id:11]                -1.71  0.29 -2.08 -1.70 -1.33
b[afternoon cafe_id:11]                   0.72  0.31  0.32  0.72  1.11
b[(Intercept) cafe_id:12]                 0.20  0.29 -0.17  0.20  0.57
b[afternoon cafe_id:12]                  -0.05  0.29 -0.42 -0.05  0.33
b[(Intercept) cafe_id:13]                 0.25  0.29 -0.12  0.25  0.62
b[afternoon cafe_id:13]                  -0.67  0.30 -1.06 -0.67 -0.29
b[(Intercept) cafe_id:14]                -0.45  0.29 -0.83 -0.46 -0.08
b[afternoon cafe_id:14]                   0.19  0.30 -0.19  0.19  0.57
b[(Intercept) cafe_id:15]                 0.82  0.29  0.46  0.82  1.19
b[afternoon cafe_id:15]                  -1.06  0.31 -1.45 -1.05 -0.67
b[(Intercept) cafe_id:16]                -0.25  0.29 -0.62 -0.24  0.12
b[afternoon cafe_id:16]                   0.10  0.29 -0.27  0.10  0.47
b[(Intercept) cafe_id:17]                 0.58  0.29  0.22  0.58  0.95
b[afternoon cafe_id:17]                  -0.09  0.29 -0.46 -0.09  0.29
b[(Intercept) cafe_id:18]                 2.11  0.30  1.73  2.11  2.49
b[afternoon cafe_id:18]                   0.10  0.32 -0.31  0.10  0.51
b[(Intercept) cafe_id:19]                -0.39  0.29 -0.76 -0.39 -0.03
b[afternoon cafe_id:19]                   0.87  0.30  0.49  0.87  1.27
b[(Intercept) cafe_id:20]                 0.10  0.29 -0.26  0.10  0.47
b[afternoon cafe_id:20]                   0.08  0.29 -0.28  0.08  0.44
sigma                                     0.47  0.03  0.44  0.47  0.51
Sigma[cafe_id:(Intercept),(Intercept)]    0.90  0.32  0.57  0.84  1.31
Sigma[cafe_id:afternoon,(Intercept)]     -0.32  0.17 -0.54 -0.29 -0.13
Sigma[cafe_id:afternoon,afternoon]        0.37  0.16  0.21  0.34  0.58
```

Shrinkage in 2 dimensions (parameter scale)

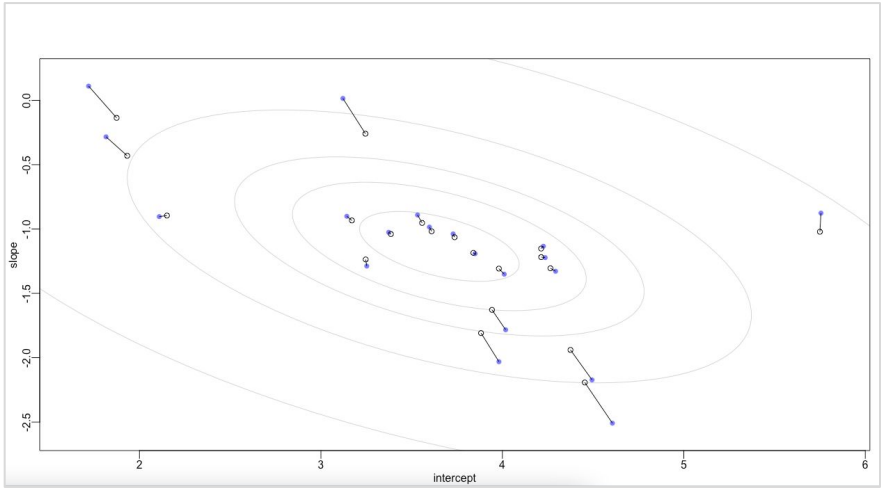
The multilevel model estimates posterior distributions for intercepts and slopes of each cafe

The inferred correlation between these varying effects was used to pool information across them

Together, the variances and correlation define an inferred multivariate Gaussian prior for the varying effects

And this prior, learned from the data, adaptively regularizes both the intercepts and slopes

To see the consequence of this adaptive regularization, shrinkage, let's plot the posterior mean varying effects and compare them to the raw, unpooled estimates



Shrinkage in 2 dimensions (outcome scale)

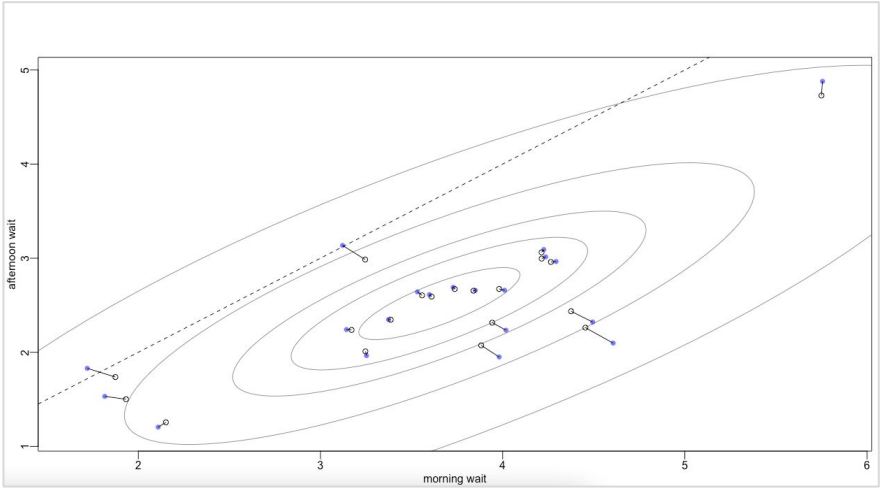
The horizontal axis shows the expected morning wait, in minutes, for each cafe. The vertical axis shows the expected afternoon wait

The open points are posterior predictions, using the pooled estimates. The filled blue points are the raw unpooled intercepts and slopes

The diagonal dashed line shows where morning wait is equal to afternoon wait

Notice that shrinkage on the parameter scale naturally produces shrinkage where we actually care about it: on the outcome scale

It also implies a population of wait times. The population is now positively correlated – cafes with longer morning waits also tend to have longer afternoon waits. But the population lies mostly below the dashed line, so you'll wait less in the afternoon, on average



Principal Component Analysis

USArrests dataset

Dataset

This example uses the `USArrests` dataset that is built into R

This dataset contains four variables that represent the number of arrests per 100,000 residents for Assault, Murder, and Rape in each of the fifty US states in 1973, as well as the percentage of the population living in urban areas, UrbanPop

```
> # compute the variance of each variable
> round( apply(USArrests , 2 , var) , 2 )
  Murder  Assault UrbanPop     Rape
   18.97  6945.17   209.52    87.73
```

```
> USArrests
               Murder Assault UrbanPop Rape
Alabama          13.2     236       58 21.2
Alaska           10.0     263       48 44.5
Arizona           8.1     294       80 31.0
Arkansas          8.8     190       50 19.5
California        9.0     276       91 40.6
Colorado          7.9     204       78 38.7
Connecticut       3.3     110       77 11.1
Delaware          5.9     238       72 15.8
Florida          15.4     335       80 31.9
Georgia          17.4     211       60 25.8
Hawaii            5.3      46       83 20.2
Idaho             2.6     120       54 14.2
Illinois         10.4     249       83 24.0
Indiana           7.2     113       65 21.0
Iowa              2.2      56       57 11.3
Kansas            6.0     115       66 18.0
Kentucky          9.7     109       52 16.3
Louisiana        15.4     249       66 22.2
Maine             2.1      83       51  7.8
Maryland         11.3     300       67 27.8
Massachusetts     4.4     149       85 16.3
Michigan         12.1     255       74 35.1
Minnesota         2.7      72       66 14.9
Mississippi      16.1     259       44 17.1
Missouri          9.0     178       70 28.2
Montana           6.0     109       53 16.4
Nebraska          4.3     102       62 16.5
Nevada           12.2     252       81 46.0
New Hampshire     2.1      57       56  9.5
New Jersey        7.4     159       89 18.8
New Mexico       11.4     285       70 32.1
New York         11.1     254       86 26.1
North Carolina   13.0     337       45 16.1
North Dakota      0.8      45       44  7.3
Ohio              7.3     120       75 21.4
Oklahoma          6.6     151       68 20.0
Oregon            4.9     159       67 29.3
Pennsylvania      6.3     106       72 14.9
Rhode Island      3.4     174       87  8.3
South Carolina   14.4     279       48 22.5
South Dakota      3.8      86       45 12.8
Tennessee        13.2     188       59 26.9
Texas            12.7     201       80 25.5
Utah              3.2     120       80 22.9
Vermont           2.2      48       32 11.2
Virginia          8.5     156       63 20.7
Washington        4.0     145       73 26.2
West Virginia     5.7      81       39  9.3
Wisconsin         2.6      53       66 10.8
Wyoming           6.8     161       60 15.6
```

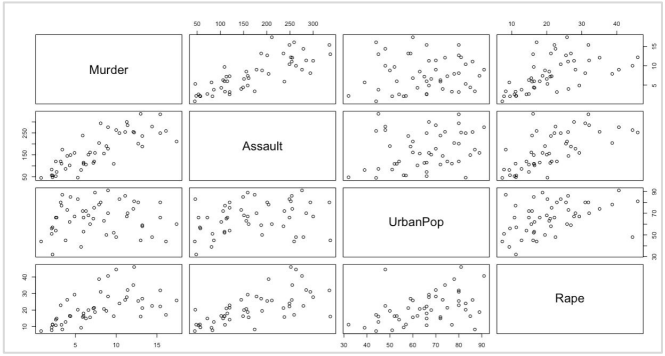
Principal Component Analysis (PCA)

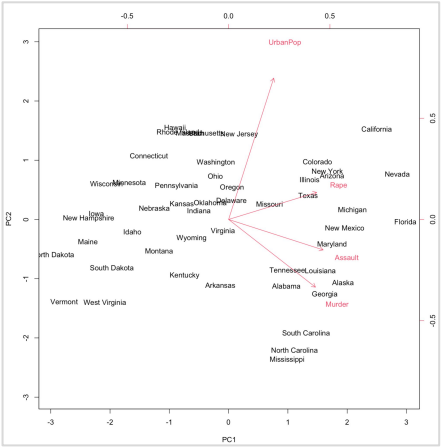
The goal of PCA is to explain most of the variability in the data with a smaller number of variables than the original data set

For a large data set with p variables, we could examine *pairwise plots of each variable against every other variable*, but even for moderate p, the number of these plots becomes excessive and not useful. For example, when p = 10 there are p(p – 1)/2 = 45 scatterplots that could be analyzed!

In particular, we would like to find a low-dimensional representation of the data that captures as much of the information as possible. For instance, if we can obtain a *two-dimensional representation* of the data that captures most of the information, then we can plot the observations in this low-dimensional space

PCA provides a tool to do just this. It finds a low-dimensional representation of a data set that contains as much of the variation as possible





Principal Component Analysis (PCA)

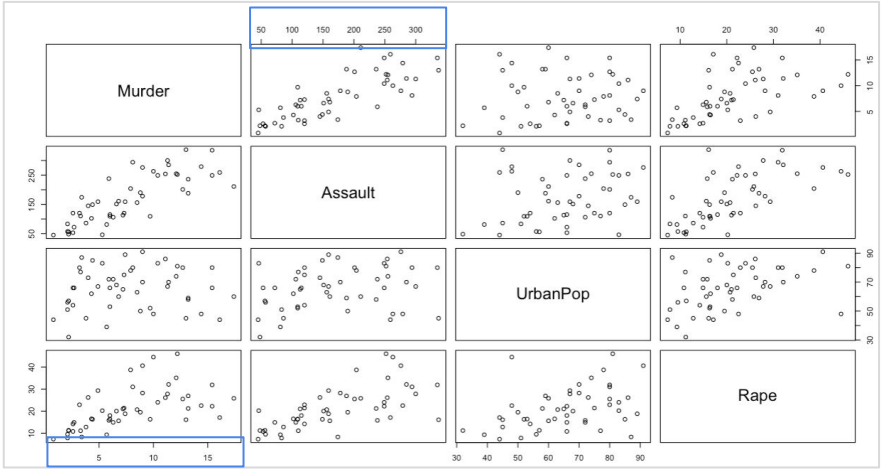
The idea is that each of the n observations lives in p-dimensional space, but not all of these dimensions are equally interesting

PCA seeks a small number of dimensions that are as interesting as possible, where *the concept of interesting is measured by the amount that the observations vary* along each dimension

An important thing to remember is ***PCA is influenced by the magnitude of each variable***; therefore, the results obtained when we perform PCA will also depend on whether the variables have been individually scaled

Each of the dimensions found by PCA is a linear combination of the p features

We can take these linear combinations of the measurements and reduce the number of plots necessary for visual analysis while retaining most of the information present in the data




```
> # compute the variance of each variable
> round( apply(USArrests , 2 , var) , 2 )
  Murder  Assault UrbanPop     Rape
   18.97 6945.17   209.52    87.73
```

It is usually beneficial to center and scale each variable to eliminate potential problems with the measurement scale of each variable

For example, the variance of Assault is 6945, while the variance of Murder is only 18.9

The Assault data isn't necessarily more variable, it's simply on a different scale relative to Murder

Computing principal components

The first principal component of the observations, y_1, is that linear combination of the original variables whose sample variance is greatest amongst all possible such linear combinations:

$y_1 = a_{11}x_1 + a_{12}x_2 + \ldots + a_{1q}x_q$, subject to $\mathbf{a}_1^T\mathbf{a}_1 = 1$

The second principal component, y_2, is that linear combination of the original variables that accounts for a maximal proportion of the remaining variance subject to being uncorrelated with the first principal component:

$y_2 = a_{21}x_1 + a_{22}x_2 + \ldots + a_{2q}x_q$, subject to $\mathbf{a}_2^T\mathbf{a}_2 = 1$ and $\mathbf{a}_2^T\mathbf{a}_1 = 0$

Subsequent components, y_j, are defined similarly:

$y_j = a_{j1}x_1 + a_{j2}x_2 + \ldots + a_{jq}x_q$, subject to $\mathbf{a}_j^T\mathbf{a}_j = 1$ and $\mathbf{a}_j^T\mathbf{a}_i = 0$ for $(i < j)$

Computing principal components

To find the coefficients defining the first principal component, i.e. the *loadings*, we need to choose the elements of the vector $\mathbf{a}_1$ so as to maximize the variance of y_1 subject to the constraint $\mathbf{a}_1^T\mathbf{a}_1 = 1$

To maximize a function of several variables subject to one or more constraints, one can use the method of *Lagrange multipliers*

This leads to the solution that $\mathbf{a}_1$ is the eigenvector of the sample covariance matrix, **S**, corresponding to its largest eigenvalue. That is, the eigenvector corresponding to the largest eigenvalue of the covariance matrix is the set of *loadings* that explains the greatest proportion of the variability

The other components are derived in similar fashion, with $\mathbf{a}_j$ being the eigenvector of **S** associated with its *j*th largest eigenvalue

```
> # create a new data frame with centered and scaled variables
> USArrests_scaled <- apply(USArrests , 2 , scale)
>
> # calculate the covariance matrix
> arrests_cov <- cov(USArrests_scaled)
> arrests_cov
            Murder   Assault   UrbanPop      Rape
Murder   1.00000000 0.8018733 0.06957262 0.5635788
Assault  0.80187331 1.0000000 0.25887170 0.6652412
UrbanPop 0.06957262 0.2588717 1.00000000 0.4113412
Rape     0.56357883 0.6652412 0.41134124 1.0000000
>
> # calculate eigenvalues & eigenvectors
> arrests_eigen <- eigen(arrests_cov)
> arrests_eigen
eigen() decomposition
$values
[1] 2.4802416 0.9897652 0.3565632 0.1734301

$vectors
           [,1]       [,2]       [,3]        [,4]
[1,] -0.5358995  0.4181809 -0.3412327  0.64922780
[2,] -0.5831836  0.1879856 -0.2681484 -0.74340748
[3,] -0.2781909 -0.8728062 -0.3780158  0.13387773
[4,] -0.5434321 -0.1673186  0.8177779  0.08902432

>
> str(arrests_eigen)
List of 2
 $ values : num [1:4] 2.48 0.99 0.357 0.173
 $ vectors: num [1:4, 1:4] -0.536 -0.583 -0.278 -0.543 0.418 ...
 - attr(*, "class")= chr "eigen"
```

Computing loadings

Let us explore the set of *loadings* for the first principal component (PC1) and second principal component (PC2)

Eigenvectors that are calculated in any software package are unique up to a sign flip. By default, eigenvectors in R point in the negative direction. For this example, we'd prefer the eigenvectors point in the positive direction, so we multiply the default loadings by –1

Each principal component vector defines a direction in feature space. Because eigenvectors are orthogonal to every other eigenvector, loadings and principal components are uncorrelated with one another

By examining the principal component vectors, we can infer:

- the first principal component (PC1) roughly corresponds to an overall rate of serious crimes since Murder, Assault, and Rape have the largest values
- the second component (PC2) is affected by UrbanPop more than the other three variables, so it roughly corresponds to the level of urbanization of the state, with some opposite, smaller influence by murder rate

```
> # eigenvectors that are calculated in any software
> # package are unique up to a sign flip
> # by default, eigenvectors in R point in the negative direction
>
> # for this example, we would prefer the eigenvectors
> # point in the positive direction because it leads to
> # more logical interpretation of graphical results
>
> # to use the positive-pointing vector,
> # multiply the default loadings by -1
> phi <- -phi
> row.names(phi) <- c("Murder", "Assault", "UrbanPop", "Rape")
> colnames(phi) <- c("PC1", "PC2")
> phi
              PC1        PC2
Murder   0.5358995 -0.4181809
Assault  0.5831836 -0.1879856
UrbanPop 0.2781909  0.8728062
Rape     0.5434321  0.1673186
```

Plotting principal components scores

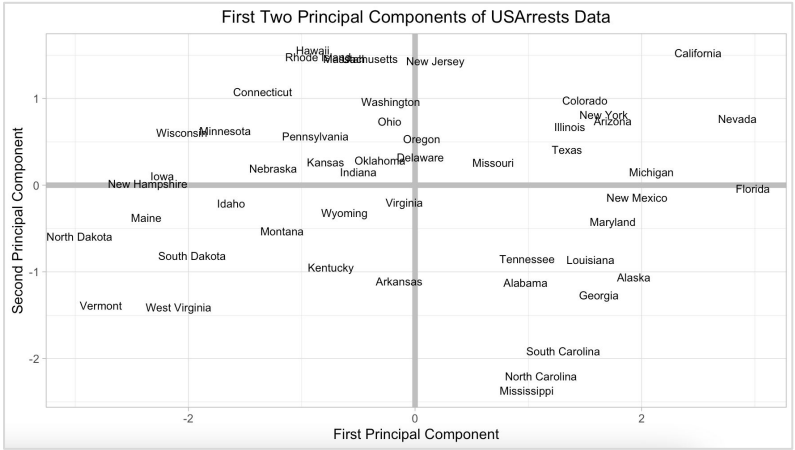
If we project the n data points $x_1, \ldots, x_n$ onto the first eigenvector, the projected values are called the principal component *scores* for each observation

Once we calculate the first and second principal components for each US state, we can plot them against each other and produce a two-dimensional value of the data:

- The first principal component (x-axis) roughly corresponds to the rate of serious crimes. States such as California, Florida, and Nevada have high rates of serious crimes, while states such as North Dakota and Vermont have far lower rates
- The second component (y-axis) is roughly explained as urbanization, which implies that states such as Hawaii and California are highly urbanized, while Mississippi and the Carolinas are far less so

A state close to the origin, such as Indiana or Virginia, is close to average in both categories

Because PCA is unsupervised, it is not making predictions about crime rates, but simply making connections between observations using fewer measurements




```
> # calculate principal components scores
> PC1 <- as.matrix(USArrests_scaled) %*% phi[,1]
> PC2 <- as.matrix(USArrests_scaled) %*% phi[,2]
>
> # create a data frame with principal components scores
> PC <- data.frame(State = row.names(USArrests), PC1, PC2)
> head(PC)
       State        PC1        PC2
1    Alabama  0.9756604 -1.1220012
2     Alaska  1.9305379 -1.0624269
3    Arizona  1.7454429  0.7384595
4   Arkansas -0.1399989 -1.1085423
5 California  2.4986128  1.5274267
6   Colorado  1.4993407  0.9776297
```

Selecting the number of principal components

In the previous analysis, we only looked at two of the four principal components. How did we know to use two principal components? And how well is the data explained by these two principal components compared to using the full data set?

Proportion of Variance Explained (PVE)
The *j*th principal component accounts for a proportion P_j of the total variation of the original data, where

$P_j = \lambda_j / \text{trace}(\mathbf{S})$, where λ_j are the eigenvalues of **S**

In other words, one can calculate the PVE of the *j*th principal component by taking the *j*th eigenvalue and dividing by the number of principal components

```
> # calculate the proportion of variance explained (PVE)
> PVE <- arrests_eigen$values / sum(arrests_eigen$values)
> round(PVE, 2)
[1] 0.62 0.25 0.09 0.04
```

The first *m* principal components, where $m < q$, account for a proportion

$P^{(m)} = \sum_m \lambda_j / \text{trace}(\mathbf{S})$

Selecting the number of principal components

The vector of PVE for each principal component is

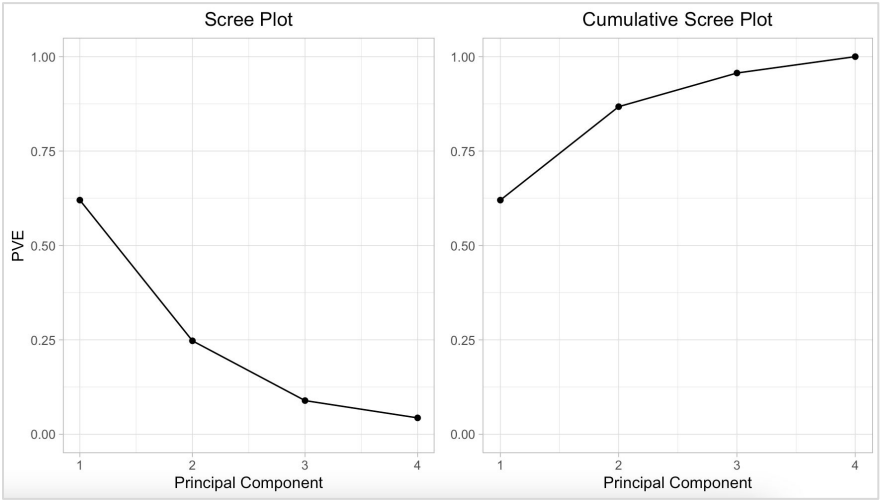
```
> # calculate the proportion of variance explained (PVE)
> PVE <- arrests_eigen$values / sum(arrests_eigen$values)
> round(PVE, 2)
[1] 0.62 0.25 0.09 0.04
```

The first principal component explains 62% of the variability

The second principal component explains 25% of the variability

Together, the first two principal components explain 87% of the variability

It is often advantageous to plot the PVE and cumulative PVE



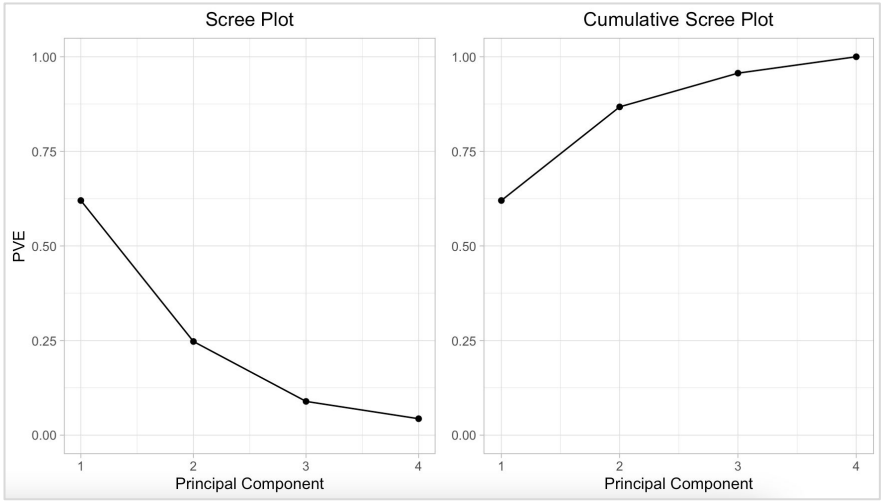
Deciding how many principal components to use

For a general n x p data matrix X, there are up to *min*(n–1,p) principal components. However, because the point of PCA is to significantly reduce the number of variables, we want to use the smallest number of principal components possible to explain most of the variability

The frank answer is that there is no robust method for determining how many components to use. As the number of observations, the number of variables, and the application vary, a different level of accuracy and variable reduction are desirable

The most common technique for determining how many principal components to keep is eyeballing the scree plot, which is the left-hand plot. To determine the number of components, we look for the “elbow point”, where the PVE significantly drops off

In our example, because we only have 4 variables to begin with, reduction to 2 variables while still explaining 87% of the variability is a good improvement



PCA in R: prcomp

R has several built-in functions (along with numerous add-on packages) that simplifies performing PCA

One of these built-in functions is `prcomp`. With `prcomp` we can perform many of the previous calculations quickly

By default, the `prcomp` function centers the variables to have mean zero. By using the option `scale = TRUE`, we scale the variables to have standard deviation one

The output from prcomp contains a number of useful quantities

```
> # conduct a principal component analysis using the prcomp function
> USArrests_pca <- prcomp(USArrests, scale = TRUE)
> names(USArrests_pca)
[1] "sdev"     "rotation" "center"   "scale"    "x"
```

The `center` and `scale` components correspond to the **means** and **standard deviations** of the variables that were used for scaling prior to implementing PCA

```
> # means
> USArrests_pca$center
  Murder  Assault UrbanPop     Rape
   7.788  170.760   65.540   21.232
>
> # standard deviations
> USArrests_pca$scale
   Murder   Assault  UrbanPop      Rape
 4.355510 83.337661 14.474763  9.366385
```

The rotation matrix provides the principal component **loadings**; each column of `pca_result$rotation`contains the corresponding principal component loading vector

```
> # principal component loadings
> USArrests_pca$rotation
                PC1        PC2        PC3         PC4
Murder   -0.5358995  0.4181809 -0.3412327  0.64922780
Assault  -0.5831836  0.1879856 -0.2681484 -0.74340748
UrbanPop -0.2781909 -0.8728062 -0.3780158  0.13387773
Rape     -0.5434321 -0.1673186  0.8177779  0.08902432
```

PCA in R: prcomp

We can also obtain the principal components **scores** from our results, stored in the `x` list item of our results

`prcomp` also outputs `sdev`, the *standard deviation* of each principal component

```
> # standard deviation of each principal component
> USArrests_pca$sdev
[1] 1.5748783 0.9948694 0.5971291 0.4164494
```

the *variance explained* by each principal component is the square of these

```
> # variance explained by each principal component
> ( VE <- USArrests_pca$sdev^2 )
[1] 2.4802416 0.9897652 0.3565632 0.1734301
```

To compute the *proportion of variance explained* by each principal component, we divide the variance explained by each principal component by the total

```
> # proportion of variance explained by each principal component
> PVE <- VE / sum(VE)
> round(PVE, 2)
[1] 0.62 0.25 0.09 0.04
```

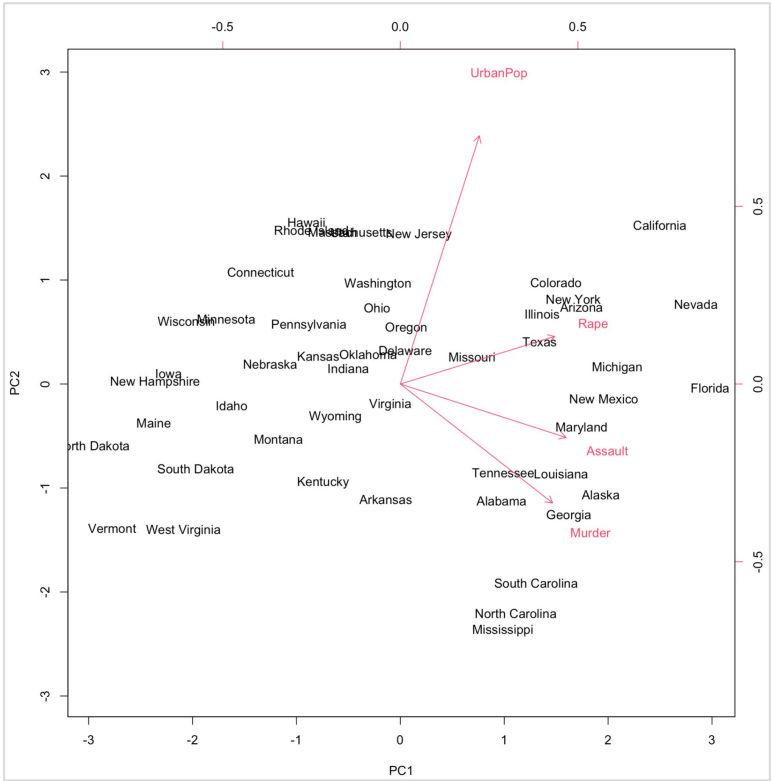
```
> # principal component scores
> USArrests_pca$x
                PC1         PC2         PC3          PC4
Alabama     -0.97566045  1.12200121 -0.43980366  0.154696581
Alaska      -1.93053788  1.06242692  2.01950027 -0.434175454
Arizona     -1.74544285 -0.73845954  0.05423025 -0.826264240
Arkansas     0.13999894  1.10854226  0.11342217 -0.180973554
California  -2.49861285 -1.52742672  0.59254100 -0.338559240
Colorado    -1.49934074 -0.97762966  1.08400162  0.001450164
Connecticut  1.34499236 -1.07798362 -0.63679250 -0.117278736
Delaware    -0.04722981 -0.32208890 -0.71141032 -0.873113315
Florida     -2.98275967  0.03883425 -0.57103206 -0.095317042
Georgia     -1.62280742  1.26608838 -0.33901818  1.065974459
Hawaii       0.90348448 -1.55467609  0.05027151  0.893733198
Idaho        1.62331903  0.20885253  0.25719021 -0.494087852
Illinois    -1.36505197 -0.67498834 -0.67068647 -0.120794916
Indiana      0.50038122 -0.15003926  0.22576277  0.420397595
Iowa         2.23099579 -0.10300828  0.16291036  0.017379470
Kansas       0.78887206 -0.26744941  0.02529648  0.204421034
Kentucky     0.74331256  0.94880748 -0.02808429  0.663817237
Louisiana   -1.54909076  0.86230011 -0.77560598  0.450157791
Maine        2.37274014  0.37260865 -0.06502225 -0.327138529
Maryland    -1.74564663  0.42335704 -0.15566968 -0.553450589
Massachusetts  0.48128007 -1.45967706 -0.60337172 -0.177793902
Michigan    -2.08725025 -0.15383500  0.38100046  0.101343128
Minnesota    1.67566951 -0.62590670  0.15153200  0.066640316
Mississippi -0.98647919  2.36973712 -0.73336290  0.213342049
Missouri    -0.68978426 -0.26070794  0.37365033  0.223554811
Montana      1.17353751  0.53147851  0.24440796  0.122498555
Nebraska     1.25291625 -0.19200440  0.17380930  0.015733156
Nevada      -2.84550542 -0.76780502  1.15168793  0.311354436
New Hampshire  2.35995585 -0.01790055  0.03648498 -0.032804291
New Jersey  -0.17974128 -1.43493745 -0.75677041  0.240936580
New Mexico  -1.96012351  0.14141308  0.18184598 -0.336121113
New York    -1.66566662 -0.81491072 -0.63661186 -0.013348844
North Carolina -1.11208808  2.20561081 -0.85489245 -0.944789648
North Dakota  2.96215223  0.59309738  0.29824930 -0.251434626
Ohio         0.22369436 -0.73477837 -0.03082616  0.469152817
Oklahoma     0.30864928 -0.28496113 -0.01515592  0.010228476
Oregon      -0.05852787 -0.53596999  0.93038718 -0.235390872
Pennsylvania  0.87948680 -0.56536050 -0.39660218  0.355452378
Rhode Island  0.85509072 -1.47698328 -1.35617705 -0.607402746
South Carolina -1.30744986  1.91397297 -0.29751723 -0.130145378
South Dakota  1.96779669  0.81506822  0.38538073 -0.108470512
Tennessee   -0.98969377  0.85160534  0.18619262  0.646302674
Texas       -1.34151838 -0.40833518 -0.48712332  0.636731051
Utah         0.54503180 -1.45671524  0.29077592 -0.081486749
Vermont      2.77325613  1.38819435  0.83280797 -0.143433697
Virginia     0.09536670  0.19772785  0.01159482  0.209246429
Washington   0.21472339 -0.96037394  0.61859067 -0.218628161
West Virginia  2.08739306  1.41052627  0.10372163  0.130583080
Wisconsin    2.05881199 -0.60512507 -0.13746933  0.182253407
Wyoming      0.62310061  0.31778662 -0.23824049 -0.164976866
```

PCA in R: Biplot of two principal components

One can plot the first two principal components (or any two principal components) using `biplot`

The points are labeled by the row names from the original data frame, which in this example are the names of the US States

In addition, the first two loadings for the four variables in the dataset are given in a second coordinate system, illustrating the difference between *UrbanPop*, the percentage of the population living in urban areas, and the number of arrests per 100,000 residents for *Assault*, *Murder*, and *Rape* in each of the fifty US states in 1973



Principal Component Analysis

Olympic heptathlon dataset

Dataset

This example uses the `heptathlon` dataset that is built into the R package `HSAUR`

This dataset contains the results for all 25 competitors in all seven disciplines of the heptathlon, including 100m hurdles, high jump, shot put, 200m, long jump, javelin, and 800m, from the 1988 Olympics held in Seoul

We can use PCA to explore the structure of the data and assess how the derived principal component scores related to the scores assigned by the official scoring system

```
> # compute the variance of each variable
> round( apply(heptathlon[, -score] , 2 , var) , 2 )
 hurdles highjump     shot  run200m longjump  javelin  run800m
    0.54     0.01     2.23     0.94     0.22    12.57    68.74
```

```
> heptathlon[, -score]
                     hurdles highjump  shot run200m longjump javelin run800m
Joyner-Kersee (USA)     3.73     1.86 15.80    4.05     7.27   45.66   34.92
John (GDR)              3.57     1.80 16.23    2.96     6.71   42.56   37.31
Behmer (GDR)            3.22     1.83 14.20    3.51     6.68   44.54   39.23
Sablovskaite (URS)      2.81     1.80 15.23    2.69     6.25   42.78   31.19
Choubenkova (URS)       2.91     1.74 14.76    2.68     6.32   47.46   35.53
Schulz (GDR)            2.67     1.83 13.50    1.96     6.33   42.82   37.64
Fleming (AUS)           3.04     1.80 12.88    3.02     6.37   40.28   30.89
Greiner (USA)           2.87     1.80 14.13    2.13     6.47   38.00   29.78
Lajbnerova (CZE)        2.79     1.83 14.28    1.75     6.11   42.20   27.38
Bouraga (URS)           3.17     1.77 12.62    3.02     6.28   39.06   28.69
Wijnsma (HOL)           2.67     1.86 13.01    1.58     6.34   37.86   31.94
Dimitrova (BUL)         3.18     1.80 12.88    3.02     6.37   40.28   30.89
Scheider (SWI)          2.57     1.86 11.58    1.74     6.05   47.50   28.50
Braun (FRG)             2.71     1.83 13.16    1.83     6.12   44.58   20.61
Ruotsalainen (FIN)      2.63     1.80 12.32    2.00     6.08   45.44   26.37
Yuping (CHN)            2.49     1.86 14.21    1.61     6.40   38.60   16.76
Hagger (GB)             2.95     1.80 12.75    1.14     6.34   35.76   24.95
Brown (USA)             2.35     1.83 12.69    1.78     6.13   44.34   17.00
Mulliner (GB)           2.03     1.71 12.68    1.69     6.10   37.76   25.41
Hautenauve (BEL)        2.38     1.77 11.81    1.00     5.99   35.68   29.53
Kytola (FIN)            2.11     1.77 11.66    0.92     5.75   39.48   30.08
Geremias (BRA)          2.19     1.71 12.95    1.11     5.50   39.64   19.41
Hui-Ing (TAI)           1.57     1.68 10.00    1.38     5.47   39.14   26.13
Jeong-Mi (KOR)          1.89     1.71 10.83    0.00     5.50   39.26   24.26
Launa (PNG)             0.00     1.50 11.78    0.45     4.88   46.38    0.00
```

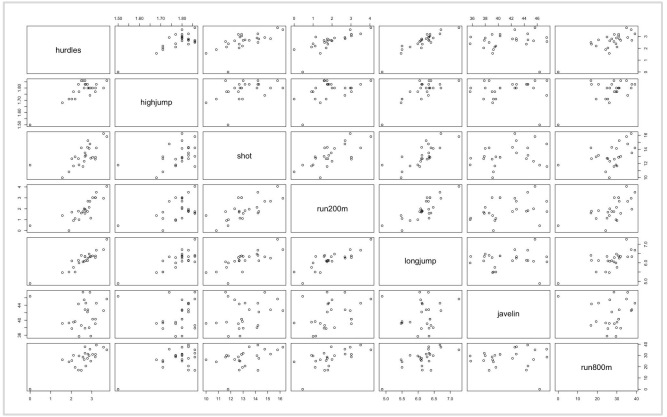
Principal Component Analysis (PCA)

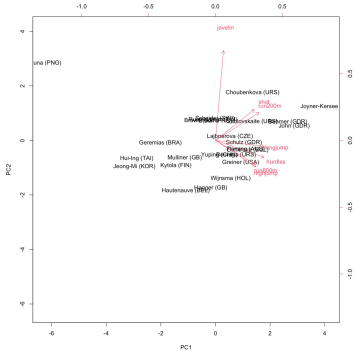
The goal of PCA is to explain most of the variability in the data with a smaller number of variables than the original data set

For a large data set with p variables, we could examine *pairwise plots of each variable against every other variable*, but even for moderate p, the number of these plots becomes excessive and not useful. For example, when p = 10 there are p(p – 1)/2 = 45 scatterplots that could be analyzed!

In particular, we would like to find a low-dimensional representation of the data that captures as much of the information as possible. For instance, if we can obtain a *two-dimensional representation* of the data that captures most of the information, then we can plot the observations in this low-dimensional space

PCA provides a tool to do just this. It finds a low-dimensional representation of a data set that contains as much of the variation as possible





Principal Component Analysis (PCA)

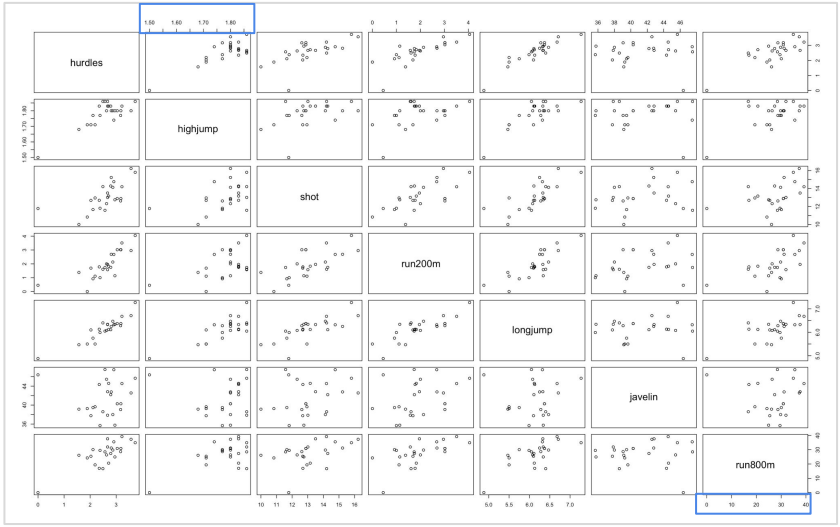
The idea is that each of the n observations lives in p-dimensional space, but not all of these dimensions are equally interesting

PCA seeks a small number of dimensions that are as interesting as possible, where *the concept of interesting is measured by the amount that the observations vary* along each dimension

An important thing to remember is ***PCA is influenced by the magnitude of each variable***; therefore, the results obtained when we perform PCA will also depend on whether the variables have been individually scaled

Each of the dimensions found by PCA is a linear combination of the p features

We can take these linear combinations of the measurements and reduce the number of plots necessary for visual analysis while retaining most of the information present in the data




```
> # compute the variance of each variable
> round( apply(heptathlon[, -score] , 2 , var) , 2 )
 hurdles highjump     shot  run200m longjump  javelin  run800m
    0.54     0.01     2.23     0.94     0.22    12.57    68.74
```

It is usually beneficial to center and scale each variable to eliminate potential problems with the measurement scale of each variable

For example, the variance of run800m is 69, while the variance of high jump is only 0.01

The run800m data isn't necessarily more variable, it's simply on a different scale relative to high jump

Computing principal components

The first principal component of the observations, y_1, is that linear combination of the original variables whose sample variance is greatest amongst all possible such linear combinations:

$y_1 = a_{11}x_1 + a_{12}x_2 + \ldots + a_{1q}x_q$, subject to $\mathbf{a}_1^T\mathbf{a}_1 = 1$

The second principal component, y_2, is that linear combination of the original variables that accounts for a maximal proportion of the remaining variance subject to being uncorrelated with the first principal component:

$y_2 = a_{21}x_1 + a_{22}x_2 + \ldots + a_{2q}x_q$, subject to $\mathbf{a}_2^T\mathbf{a}_2 = 1$ and $\mathbf{a}_2^T\mathbf{a}_1 = 0$

Subsequent components, y_j, are defined similarly:

$y_j = a_{j1}x_1 + a_{j2}x_2 + \ldots + a_{jq}x_q$, subject to $\mathbf{a}_j^T\mathbf{a}_j = 1$ and $\mathbf{a}_j^T\mathbf{a}_i = 0$ for $(i < j)$

Computing principal components

To find the coefficients defining the first principal component, i.e. the *loadings*, we need to choose the elements of the vector $\mathbf{a}_1$ so as to maximize the variance of y_1 subject to the constraint $\mathbf{a}_1^T\mathbf{a}_1 = 1$

To maximize a function of several variables subject to one or more constraints, one can use the method of *Lagrange multipliers*

This leads to the solution that $\mathbf{a}_1$ is the eigenvector of the sample covariance matrix, **S**, corresponding to its largest eigenvalue. That is, the eigenvector corresponding to the largest eigenvalue of the covariance matrix is the set of *loadings* that explains the greatest proportion of the variability

The other components are derived in similar fashion, with $\mathbf{a}_j$ being the eigenvector of **S** associated with its *j*th largest eigenvalue

```
> # create new data frame with centered variables
> heptathlon_scaled <- apply(heptathlon[, -score] , 2 , scale)
>
> # calculate the covariance matrix
> heptathlon_cov <- cov(heptathlon_scaled)
> heptathlon_cov
            hurdles    highjump      shot    run200m   longjump      javelin     run800m
hurdles  1.000000000 0.811402536 0.6513347 0.7737205 0.91213362  0.007762549  0.77925711
highjump 0.811402536 1.000000000 0.4407861 0.4876637 0.78244227  0.002153016  0.59116282
shot     0.651334688 0.440786140 1.0000000 0.6826704 0.74307300  0.268988837  0.41961957
run200m  0.773720543 0.487663685 0.6826704 1.0000000 0.81720530  0.333042722  0.61681006
longjump 0.912133617 0.782442273 0.7430730 0.8172053 1.00000000  0.067108409  0.69951116
javelin  0.007762549 0.002153016 0.2689888 0.3330427 0.06710841  1.000000000 -0.02004909
run800m  0.779257110 0.591162823 0.4196196 0.6168101 0.69951116 -0.020049088  1.00000000
>
> # calculate eigenvalues & eigenvectors
> heptathlon_eigen <- eigen(heptathlon_cov)
> heptathlon_eigen
eigen() decomposition
$values
[1] 4.46027516 1.19432056 0.52101413 0.45716683 0.24526674 0.07295558 0.04900101

$vectors
           [,1]        [,2]        [,3]        [,4]        [,5]        [,6]        [,7]
[1,] -0.4528710  0.15792058 -0.04514996 -0.02653873  0.09494792  0.78334101 -0.38024707
[2,] -0.3771992  0.24807386 -0.36777902 -0.67999172 -0.01879888 -0.09939981  0.43393114
[3,] -0.3630725 -0.28940743  0.67618919 -0.12431725 -0.51165201  0.05085983  0.21762491
[4,] -0.4078950 -0.26038545  0.08359211  0.36106580  0.64983404 -0.02495639  0.45338483
[5,] -0.4562318  0.05587394  0.13931653 -0.11129249  0.18429810 -0.59020972 -0.61206388
[6,] -0.0754090 -0.84169212 -0.47156016 -0.12079924 -0.13510669  0.02724076 -0.17294667
[7,] -0.3749594  0.22448984 -0.39585671  0.60341130 -0.50432116 -0.15555520  0.09830963

>
> str(heptathlon_eigen)
List of 2
 $ values : num [1:7] 4.46 1.194 0.521 0.457 0.245 ...
 $ vectors: num [1:7, 1:7] -0.453 -0.377 -0.363 -0.408 -0.456 ...
 - attr(*, "class")= chr "eigen"
```

Computing loadings

Let us explore the set of *loadings* for the first principal component (PC1) and second principal component (PC2)

Eigenvectors that are calculated in any software package are unique up to a sign flip. By default, eigenvectors in R point in the negative direction. For this example, we'd prefer the eigenvectors point in the positive direction, so we multiply the default loadings by –1

Each principal component vector defines a direction in feature space. Because eigenvectors are orthogonal to every other eigenvector, loadings and principal components are uncorrelated with one another

By examining the principal component vectors, we can infer:

- the first principal component (PC1) roughly corresponds to the non-javelin events since hurdles, high jump, shot, run20m, long jump, and run800m have the largest values
- the second component (PC2) is affected by javelin more than the other variables, so it roughly corresponds to that event

```
> # eigenvectors that are calculated in any software
> # package are unique up to a sign flip
> # by default, eigenvectors in R point in the negative direction
>
> # for this example, we would prefer the eigenvectors
> # point in the positive direction because it leads to
> # more logical interpretation of graphical results
>
> # to use the positive-pointing vector,
> # multiply the default loadings by -1
> phi <- -phi
> row.names(phi) <- c("hurdles", "highjump", "shot", "run200m",
+                     "longjump", "javelin", "run800m")
> colnames(phi) <- c("PC1", "PC2")
> phi
              PC1         PC2
hurdles  0.4528710 -0.15792058
highjump 0.3771992 -0.24807386
shot     0.3630725  0.28940743
run200m  0.4078950  0.26038545
longjump 0.4562318 -0.05587394
javelin  0.0754090  0.84169212
run800m  0.3749594 -0.22448984
```

Plotting principal components scores

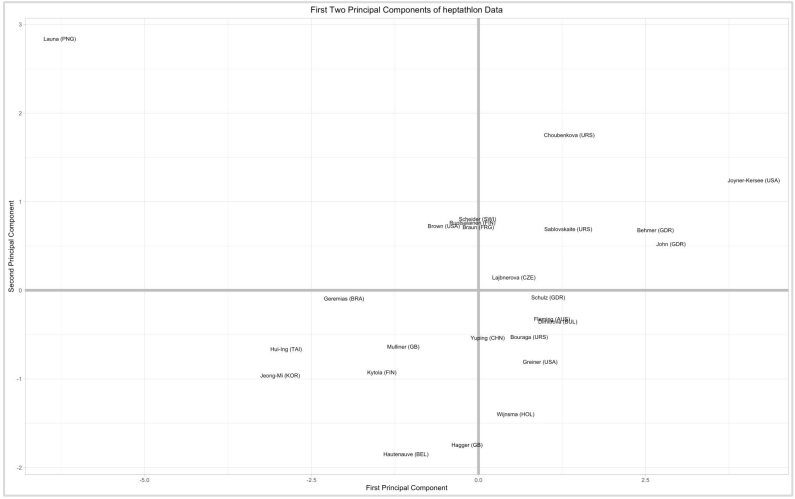
If we project the n data points $x_1, \ldots, x_n$ onto the first eigenvector, the projected values are called the principal component *scores* for each observation

Once we calculate the first and second principal components for each competitor, we can plot them against each other and produce a two-dimensional value of the data:

- The first principal component (x-axis) roughly corresponds to the non-javelin events. Competitors such as Joyner-Kersee, John, and Behmer have scores, while competitors such as Hui-Ing, Jeong-Mi, and Launa have lower scores
- The second component (y-axis) is roughly explained as the javelin event, which implies that competitors such as Launa, Choubenkova, and Joyner-Kersee performed well, while Hautenauve, Hagger, and Wijnsma performed less well

A competitor close to the origin, such as Lajbnerova, is close to average in both categories

Because PCA is unsupervised, it is not making predictions about heptathlon scores, but simply making connections between observations using fewer measurements




```
> # calculate principal components scores
> PC1 <- as.matrix(heptathlon_scaled) %*% phi[,1]
> PC2 <- as.matrix(heptathlon_scaled) %*% phi[,2]
>
> # create a data frame with principal components scores
> PC <- data.frame(Competitor = row.names(heptathlon), PC1, PC2)
> head(PC)
          Competitor      PC1         PC2
1 Joyner-Kersee (USA) 4.121448  1.24240435
2          John (GDR) 2.882186  0.52372600
3        Behmer (GDR) 2.649634  0.67876243
4  Sablovskaite (URS) 1.343351  0.69228324
5   Choubenkova (URS) 1.359026  1.75316563
6        Schulz (GDR) 1.043847 -0.07940725
```

Selecting the number of principal components

In the previous analysis, we only looked at two of the four principal components. How did we know to use two principal components? And how well is the data explained by these two principal components compared to using the full data set?

Proportion of Variance Explained (PVE)
The *j*th principal component accounts for a proportion P_j of the total variation of the original data, where

$P_j = \lambda_j / \text{trace}(\mathbf{S})$, where λ_j are the eigenvalues of **S**

In other words, one can calculate the PVE of the *j*th principal component by taking the *j*th eigenvalue and dividing by the number of principal components

```
> # calculate the proportion of variance explained (PVE)
> PVE <- heptathlon_eigen$values / sum(heptathlon_eigen$values)
> round(PVE, 2)
[1] 0.64 0.17 0.07 0.07 0.04 0.01 0.01
```

The first *m* principal components, where $m < q$, account for a proportion

$P^{(m)} = \sum_m \lambda_j / \text{trace}(\mathbf{S})$

Selecting the number of principal components

The vector of PVE for each principal component is

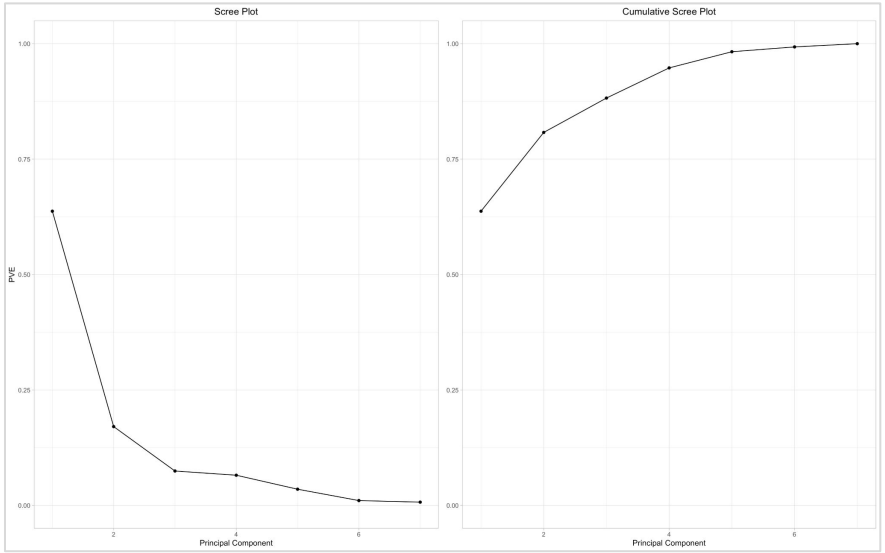
```
> # calculate the proportion of variance explained (PVE)
> PVE <- heptathlon_eigen$values / sum(heptathlon_eigen$values)
> round(PVE, 2)
[1] 0.64 0.17 0.07 0.07 0.04 0.01 0.01
```

The first principal component explains 64% of the variability

The second principal component explains 17% of the variability

Together, the first two principal components explain 81% of the variability

It is often advantageous to plot the PVE and cumulative PVE



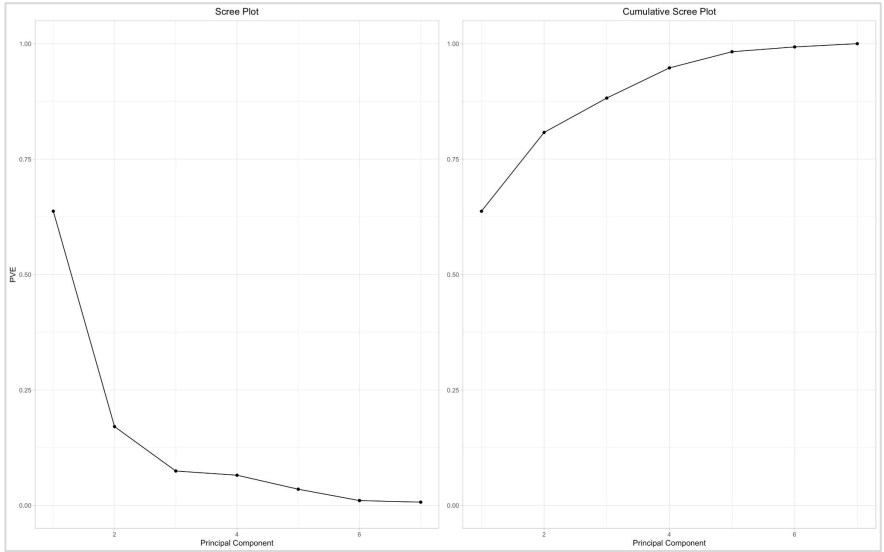
Deciding how many principal components to use

For a general n x p data matrix X, there are up to *min*(n−1,p) principal components. However, because the point of PCA is to significantly reduce the number of variables, we want to use the smallest number of principal components possible to explain most of the variability

The frank answer is that there is no robust method for determining how many components to use. As the number of observations, the number of variables, and the application vary, a different level of accuracy and variable reduction are desirable

The most common technique for determining how many principal components to keep is eyeballing the scree plot, which is the left-hand plot. To determine the number of components, we look for the “elbow point”, where the PVE significantly drops off

In our example, because we only have 7 variables to begin with, reduction to 2 variables while still explaining 81% of the variability is a good improvement



PCA in R: prcomp

R has several built-in functions (along with numerous add-on packages) that simplifies performing PCA

One of these built-in functions is `prcomp`. With `prcomp` we can perform many of the previous calculations quickly

By default, the `prcomp` function centers the variables to have mean zero. By using the option `scale = TRUE`, we scale the variables to have standard deviation one

The output from prcomp contains a number of useful quantities

```
> # conduct a principal component analysis using the prcomp function
> heptathlon_pca <- prcomp(heptathlon[, -score], scale = TRUE)
> names(heptathlon_pca)
[1] "sdev"     "rotation" "center"   "scale"    "x"
```

The `center` and `scale` components correspond to the **means** and **standard deviations** of the variables that were used for scaling prior to implementing PCA

```
> # means
> heptathlon_pca$center
 hurdles highjump     shot  run200m longjump  javelin  run800m
  2.5800   1.7820  13.1176   1.9608   6.1524  41.4824  27.3760
>
> # standard deviations
> heptathlon_pca$scale
   hurdles   highjump       shot    run200m   longjump    javelin    run800m
0.73664781 0.07794229 1.49188438 0.96955712 0.47421233 3.54565612 8.29108809
```

The rotation matrix provides the principal component **loadings**; each column of `pca_result$rotation` contains the corresponding principal component loading vector

```
> heptathlon_pca$rotation <- -heptathlon_pca$rotation
> heptathlon_pca$rotation
             PC1         PC2         PC3         PC4         PC5         PC6         PC7
hurdles  0.4528710 -0.15792058  0.04514996 -0.02653873  0.09494792  0.78334101 -0.38024707
highjump 0.3771992 -0.24807386  0.36777902 -0.67999172 -0.01879888 -0.09939981  0.43393114
shot     0.3630725  0.28940743 -0.67618919 -0.12431725 -0.51165201  0.05085983  0.21762491
run200m  0.4078950  0.26038545 -0.08359211  0.36106580  0.64983404 -0.02495639  0.45338483
longjump 0.4562318 -0.05587394 -0.13931653 -0.11129249  0.18429810 -0.59020972 -0.61206388
javelin  0.0754090  0.84169212  0.47156016 -0.12079924 -0.13510669  0.02724076 -0.17294667
run800m  0.3749594 -0.22448984  0.39585671  0.60341130 -0.50432116 -0.15555520  0.09830963
```

PCA in R: prcomp

We can also obtain the principal components **scores** from our results, stored in the `x` list item of our results

`prcomp` also outputs `sdev`, the *standard deviation* of each principal component

```
> # standard deviation of each principal component
> heptathlon_pca$sdev
[1] 2.1119364 1.0928497 0.7218131 0.6761411 0.4952441 0.2701029 0.2213617
```

the *variance explained* by each principal component is the square of these

```
> # variance explained by each principal component
> ( VE <- heptathlon_pca$sdev^2 )
[1] 4.46027516 1.19432056 0.52101413 0.45716683 0.24526674 0.07295558 0.04900101
```

To compute the *proportion of variance explained* by each principal component, we divide the variance explained by each principal component by the total

```
> # proportion of variance explained by each principal component
> PVE <- VE / sum(VE)
> round(PVE, 2)
[1] 0.64 0.17 0.07 0.07 0.04 0.01 0.01
```

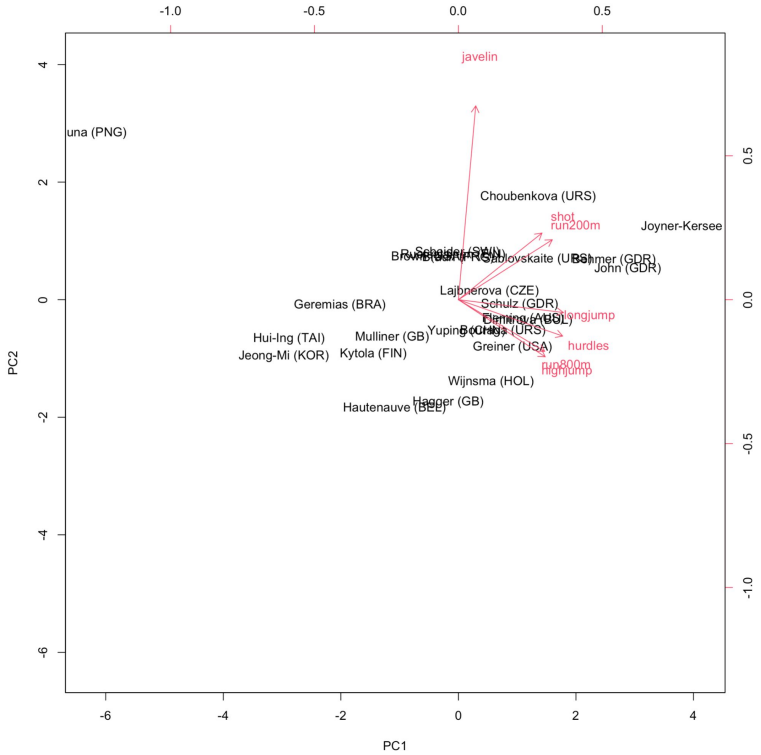
```
> # principal component scores
> round( heptathlon_pca$x , 2 )
                      PC1   PC2   PC3   PC4   PC5   PC6   PC7
Joyner-Kersee (USA)  4.12  1.24 -0.37 -0.02  0.43 -0.34 -0.35
John (GDR)           2.88  0.52 -0.90  0.48 -0.70  0.24 -0.14
Behmer (GDR)         2.65  0.68  0.46  0.68  0.11 -0.24  0.13
Sablovskaite (URS)   1.34  0.69 -0.60  0.14 -0.45  0.09  0.49
Choubenkova (URS)    1.36  1.75  0.15  0.84 -0.69  0.13 -0.24
Schulz (GDR)         1.04 -0.08  0.67  0.21 -0.74 -0.36  0.10
Fleming (AUS)        1.10 -0.32  0.07  0.49  0.76  0.08  0.14
Greiner (USA)        0.92 -0.81 -0.81  0.03 -0.09 -0.15 -0.03
Lajbnerova (CZE)     0.53  0.15 -0.16 -0.62 -0.57  0.27  0.25
Bouraga (URS)        0.76 -0.53 -0.18  0.67  1.02  0.40  0.02
Wijnsma (HOL)        0.56 -1.40  0.14 -0.41 -0.29 -0.34  0.18
Dimitrova (BUL)      1.19 -0.35  0.08  0.48  0.78  0.23  0.07
Scheider (SWI)      -0.02  0.81  1.97 -0.73  0.02  0.00 -0.04
Braun (FRG)          0.00  0.71  0.32 -1.07  0.18  0.27 -0.04
Ruotsalainen (FIN)  -0.09  0.76  0.95 -0.27  0.18  0.14 -0.14
Yuping (CHN)         0.14 -0.54 -1.07 -1.63  0.21 -0.28  0.17
Hagger (GB)         -0.17 -1.74 -0.59 -0.47  0.06  0.15 -0.52
Brown (USA)         -0.52  0.73  0.31 -1.29  0.50 -0.07  0.01
Mulliner (GB)       -1.13 -0.63 -0.73  0.58  0.16 -0.43 -0.08
Hautenauve (BEL)    -1.09 -1.85 -0.01  0.26 -0.19 -0.10 -0.09
Kytola (FIN)        -1.45 -0.92  0.65  0.21 -0.50 -0.07  0.13
Geremias (BRA)      -2.01 -0.09 -0.65 -0.02 -0.24  0.64  0.22
Hui-Ing (TAI)       -2.88 -0.66  0.75  1.12  0.47 -0.18  0.21
Jeong-Mi (KOR)      -2.97 -0.96  0.57  0.12 -0.58  0.18 -0.38
Launa (PNG)         -6.27  2.84 -1.03  0.24  0.17 -0.26 -0.06
```

PCA in R: Biplot of two principal components

One can plot the first two principal components (or any two principal components) using `biplot`

The points are labeled by the row names from the original data frame, which in this example are the names of the competitors

In addition, the first two loadings for the seven events scores in the dataset are given in a second coordinate system, illustrating the difference between *javelin* and the other six events in the 1988 Olympic heptathlon



Appendix

Resources

Regression and Other Stories

Statistical Rethinking

Statistical rethinking with brms, ggplot2, and the tidyverse: Second edition

Bayes Rules!

Tidy Modeling with R

Doing Bayesian Data Analysis, Second edition

Doing Bayesian Data Analysis in brms and the tidyverse

rstanarm vignettes

bayesplot vignettes

R for Data Science

R Graphics Cookbook