Statistics for Genomics: PCA

May 08, 2025

Table of Contents

# About this Book

This book is intended to introduce principle component analysis (PCA) in genomics research.

## Skills Level

*Genetics*

*Programming skills*

## GDSCN Collection

This exercise is part of a collection of teaching resources developed through the *Genomic Data Science Community Network* (GDSCN). GDSCN works towards a vision where researchers, educators, and students from diverse backgrounds are able to fully participate in genomic data science research. Learn more about GDSCN by visiting <https://www.gdscn.org/home> or reading the [article in Genome Research](https://doi.org/10.1101/gr.276496.121).

Please check out our full collection of AnVIL and related resources: <https://hutchdatascience.org/AnVIL_Collection/>

# 1 Principal Components Analysis

## 1.1 Before You Start

Make sure you can start an RStudio session on AnVIL. Take a look at our guide [here](https://jhudatascience.org/GDSCN_Book_Statistics_for_Genomics_Differential_Expression/getting-set-up.html#introduction).

### 1.1.1 Libraries

We will be using the prcomp() command which comes loaded into baseR, as well as the [factoextra package](https://www.rdocumentation.org/packages/factoextra/versions/1.0.7) for visualization. You can install the factoextra package using the install.packages command.

# Install and load factoextra  
# AnVIL::install("factoextra")  
library(factoextra)

## 1.2 Introduction

Principal components analysis (PCA) is one of the oldest and most commonly used dimensional reduction techniques. PCA is an unsupervised machine learning algorithm that combines correlated dimensions into a single new variable. This new variable represents an axis or line in the dataset that describes the maximum amount of variation and becomes the first principal component (PC). After the first PC is created, the algorithm then identifies a new axis that is orthagonal (statistically independent) to the first PC while also describing the maximum amount of previously undescribed variation.

This process continues, with each PC capturing a decreasing amount of variation, until the maximum number of PCS is reached (max number of PCs = total number of variables in the original dataset).

The first PCs will capture most heterogeneity (and thus most of the signal) in our high-dimensional dataset by definition, while the variation in later PCs will mostly represent noise. Thus, we can use the first PCs to both examine structure in our dataset as well as for downstream analyses to reduce computational work and improve the fit of our statistical models.

GOING DEEPER: The math behind PCA

PCA is a method of matrix factorization. Essentially, the PCA approach allows a researcher to break down (or decompose) a large data matrix into smaller constituent parts (similar to the way we can break the number 30 into the factors 5 and 6). Each smaller matrix can be summarized by a vector, called the eigenvector. This vector can be resized; how much it is resized can be summarized by a number called the eigenvalue. You can also think of the eigenvalue as the variance of the eigenvector.

For a large matrix (like most biological datasets), there are many possible ways to decompose the matrix. In our example earlier with the number 30 we could choose to start with the number 5 or the number 6. With PCA, We always start with the eigenvector that has the largest possible eigenvalue. This eigenvector is what we call the first PC. What is left of the original data matrix is then decomposed further into smaller and smaller pieces, with each subsequent eigenvector chosen because it has the largest possible eigenvalue.

The PC variables created by this method are a linear combination of the original variables.The commands in this exercise use the Singular Value Decomposition (SVD) algorithm for matrix factorization. SVD breaks a large matrix down into three smaller matrices and works on both non-symmetric and symmetric matrices.

If you’d like to learn more, check out [Principal component analysis: a review and recent developments](www.ncbi.nlm.nih.gov/pmc/articles/PMC4792409/) or [Understanding Singular Value Decomposition](https://towardsdatascience.com/understanding-singular-value-decomposition-and-its-application-in-data-science-388a54be95d#:~:text=In%20linear%20algebra%2C%20the%20Singular%20Value%20Decomposition%20%28SVD%29,also%20has%20some%20important%20applications%20in%20data%20science).

## 1.3 Calculating PCs

### 1.3.1 Examining the data

There are many packages available for R and RStudio that can perform PCA. These packages will calculate PCA in similar ways, although some of their default parameters (such as how many PCs the algorithm will calculate and store) may be different. Be sure to read the manuals for any package you choose to use.

For this example, we will use the iris dataset that is preloaded into R. The statistician RA Fisher popularized this dataset for statistical work, although the original data was collected by Edgar Anderson. The dataset contains 4 measurements (petal length, petal width, sepal length, and sepal width) for 150 individuals from 3 different iris species.

Let’s start by looking at the first few rows of the dataset.

head(iris)

## Sepal.Length Sepal.Width Petal.Length Petal.Width Species  
## 1 5.1 3.5 1.4 0.2 setosa  
## 2 4.9 3.0 1.4 0.2 setosa  
## 3 4.7 3.2 1.3 0.2 setosa  
## 4 4.6 3.1 1.5 0.2 setosa  
## 5 5.0 3.6 1.4 0.2 setosa  
## 6 5.4 3.9 1.7 0.4 setosa

Each row represents a single individual flower. The first four columns are *active variables*, or the variables we will use for the principal components analysis. The fifth column is the species of each individual ( *Iris setosa*, *Iris virginica*, or *Iris versicolor*). This is a *supplementary variable*, or additional information about each sample that we will not use in the PCA.

### 1.3.2 Calculating the PCs

The actual PCA calculation is quite simple. We specify the calculations are only done on the first four columns of the iris dataset, and we are also scaling each variable so the mean is 0 and the standard deviation is 1.

pca.obj <- prcomp(iris[1:4], scale=T)  
str(pca.obj)

## List of 5  
## $ sdev : num [1:4] 1.708 0.956 0.383 0.144  
## $ rotation: num [1:4, 1:4] 0.521 -0.269 0.58 0.565 -0.377 ...  
## ..- attr(\*, "dimnames")=List of 2  
## .. ..$ : chr [1:4] "Sepal.Length" "Sepal.Width" "Petal.Length" "Petal.Width"  
## .. ..$ : chr [1:4] "PC1" "PC2" "PC3" "PC4"  
## $ center : Named num [1:4] 5.84 3.06 3.76 1.2  
## ..- attr(\*, "names")= chr [1:4] "Sepal.Length" "Sepal.Width" "Petal.Length" "Petal.Width"  
## $ scale : Named num [1:4] 0.828 0.436 1.765 0.762  
## ..- attr(\*, "names")= chr [1:4] "Sepal.Length" "Sepal.Width" "Petal.Length" "Petal.Width"  
## $ x : num [1:150, 1:4] -2.26 -2.07 -2.36 -2.29 -2.38 ...  
## ..- attr(\*, "dimnames")=List of 2  
## .. ..$ : NULL  
## .. ..$ : chr [1:4] "PC1" "PC2" "PC3" "PC4"  
## - attr(\*, "class")= chr "prcomp"

The prcomp() command will create a list containing multiple dataframes and vectors:

* *sdev*, a vector of the standard deviations for each PC
* *rotation*, a matrix of the eigenvectors for each variable
* *center*, a vector of the center value for each variable
* *scale*, a vector of the scaling factor for each variable
* *x*, a matrix of the individual PC values for each flower

## 1.4 Choosing the number of PCs

We have now calculated our PCs. In this dataset, we have a limited number of PCs and could choose to use all of them in downstream analysis, but this becomes impractical with high-dimensional data. The majority of the variation in the dataset will be captured by the first few PCs, so common practice is to identify a cutoff for how much variation a PC must explain in order to be included. If the data have been scaled and centered, it is also common to include only those PCs with an eigenvalue greater than or equal to 1.

### 1.4.1 Examining PCs in table format

Let’s take a look at a table of the eigenvalues and variation captured by each PC first. We’ll use a command from the factoextra package.

get\_eigenvalue(pca.obj)

## eigenvalue variance.percent cumulative.variance.percent  
## Dim.1 2.91849782 72.9624454 72.96245  
## Dim.2 0.91403047 22.8507618 95.81321  
## Dim.3 0.14675688 3.6689219 99.48213  
## Dim.4 0.02071484 0.5178709 100.00000

This table tells us the vast majority of the variation is described by just the first two PCs.

This information can also be calculated in the sdev vector of the pca.obj list. You don’t have to use an additional library like factoextra. Instead, you can convert the standard deviations to variances.

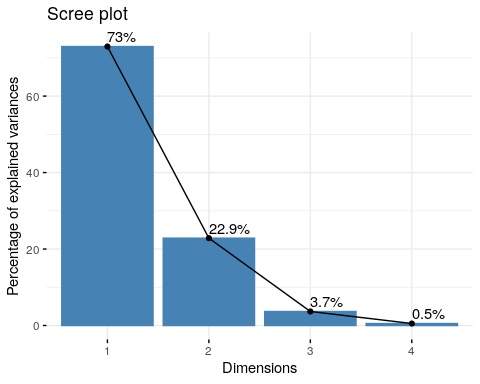
pca.obj$sdev^2 / sum(pca.obj$sdev^2)

## [1] 0.729624454 0.228507618 0.036689219 0.005178709

### 1.4.2 Examining PCs in a scree plot

It may be easier to examine this data visually, which we can do using a scree plot (Jollife 2002, Peres-Neto, Jackson, and Somers 2005).

fviz\_eig(pca.obj, addlabels = TRUE)



The PCs are on the x-axis, and the y-axis is the percentage of variance explained by each PC. As you can see, the percentage of explained variance drops off dramatically after the first two PCs.

Going forward, we might choose to use these two variables in our analyses.

## 1.5 Correlation between variables and PCs

At some point, we might be interested in knowing which variables are contributing to each PC.

### 1.5.1 Examining the contribution of each variable to PCs in table format

Let’s first look at this in table form. We first extract the results from our PCA output.

var <- get\_pca\_var(pca.obj)  
var

## Principal Component Analysis Results for variables  
## ===================================================  
## Name Description   
## 1 "$coord" "Coordinates for the variables"   
## 2 "$cor" "Correlations between variables and dimensions"  
## 3 "$cos2" "Cos2 for the variables"   
## 4 "$contrib" "contributions of the variables"

The get\_pca\_var creates a list of matrices with useful information about the PCA. For now, we’re interested in the contribution matrix.

var$contrib

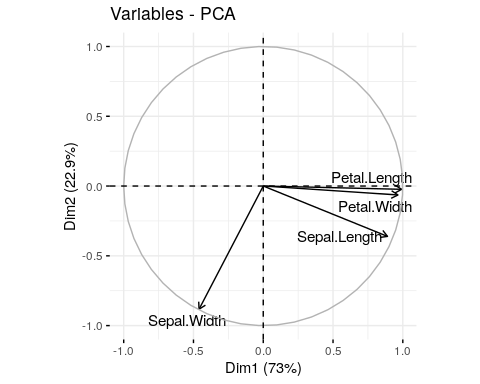
## Dim.1 Dim.2 Dim.3 Dim.4  
## Sepal.Length 27.150969 14.24440565 51.777574 6.827052  
## Sepal.Width 7.254804 85.24748749 5.972245 1.525463  
## Petal.Length 33.687936 0.05998389 2.019990 64.232089  
## Petal.Width 31.906291 0.44812296 40.230191 27.415396

This table tells us what percentage of each PC is contributed by each variable. A higher number (or percentage) means a greater contribution by that variable.

### 1.5.2 Examining the contribution of each variable to PCs with a correlation circle

We can also visualize this information using a correlation circle.

fviz\_pca\_var(pca.obj, col.var = "black", repel=T)



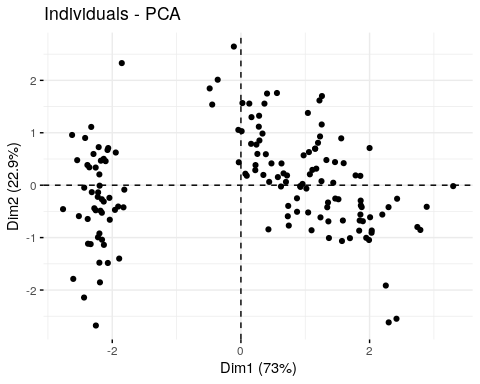
The size and position of the vectors (arrows) in the correlation circle can tell us quite a lot about the variables.

* Positively correlated variables are grouped together.
* Negatively correlated variables are positioned on opposite sides of the plot origin (opposed quadrants) .
* Vectors that end close to the circle outline greatly contribute to a given PC.
* Shorter vectors contribute very little.
* Vectors that are primarily horizontal mostly contribute to the first PC (the x-axis).
* Vectors that are more vertical contribute primarily to the second PC (the y-axis).

## 1.6 Visualizing structure using PCs

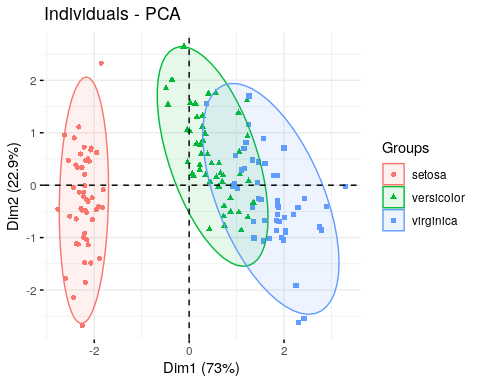
Sometimes it is helpful to plot the PC coordinates of the individual samples in order to identify any population structure in the dataset.

fviz\_pca\_ind(pca.obj, geom.ind = "point", mean.point=F)



Here we’ve created a scatter plot with the first PC (Dim1) on the x-axis and the second PC (Dim2) on the y-axis. We can clearly see the samples split into 2 clusters. It might be interesting see whether these clusters correspond with iris species (remember, we have that information in the fifth column of the iris data table).

fviz\_pca\_ind(pca.obj, geom.ind = "point", mean.point=F,  
 col.ind = iris$Species,  
 addEllipses = TRUE,  
 legend.title = "Groups"  
 )



We used the col.ind option to color the dots by species and the addEllipses option to create an ellipse around the center of each species cluster.

## 1.7 Recap

sessionInfo()

## R version 4.1.3 (2022-03-10)  
## Platform: x86\_64-pc-linux-gnu (64-bit)  
## Running under: Ubuntu 20.04.5 LTS  
##   
## Matrix products: default  
## BLAS: /usr/lib/x86\_64-linux-gnu/openblas-pthread/libblas.so.3  
## LAPACK: /usr/lib/x86\_64-linux-gnu/openblas-pthread/liblapack.so.3  
##   
## locale:  
## [1] LC\_CTYPE=en\_US.UTF-8 LC\_NUMERIC=C   
## [3] LC\_TIME=en\_US.UTF-8 LC\_COLLATE=en\_US.UTF-8   
## [5] LC\_MONETARY=en\_US.UTF-8 LC\_MESSAGES=en\_US.UTF-8   
## [7] LC\_PAPER=en\_US.UTF-8 LC\_NAME=C   
## [9] LC\_ADDRESS=C LC\_TELEPHONE=C   
## [11] LC\_MEASUREMENT=en\_US.UTF-8 LC\_IDENTIFICATION=C   
##   
## attached base packages:  
## [1] stats graphics grDevices utils datasets methods base   
##   
## other attached packages:  
## [1] factoextra\_1.0.7 ggplot2\_3.3.5   
##   
## loaded via a namespace (and not attached):  
## [1] tidyselect\_1.1.2 xfun\_0.26 purrr\_0.3.4 carData\_3.0-5   
## [5] colorspace\_2.0-3 vctrs\_0.4.1 generics\_0.1.2 htmltools\_0.5.2   
## [9] yaml\_2.3.5 utf8\_1.2.2 rlang\_1.0.2 pillar\_1.7.0   
## [13] ggpubr\_0.4.0 glue\_1.6.2 withr\_2.5.0 DBI\_1.1.2   
## [17] lifecycle\_1.0.1 stringr\_1.4.0 munsell\_0.5.0 ggsignif\_0.6.3   
## [21] gtable\_0.3.0 evaluate\_0.15 labeling\_0.4.2 knitr\_1.33   
## [25] tzdb\_0.3.0 fastmap\_1.1.0 fansi\_1.0.3 highr\_0.9   
## [29] broom\_0.8.0 Rcpp\_1.0.8.3 readr\_2.1.2 scales\_1.2.1   
## [33] backports\_1.4.1 abind\_1.4-5 farver\_2.1.0 fs\_1.5.2   
## [37] ottrpal\_1.0.1 hms\_1.1.1 digest\_0.6.29 stringi\_1.7.6   
## [41] bookdown\_0.24 rstatix\_0.7.0 dplyr\_1.0.8 ggrepel\_0.9.1   
## [45] grid\_4.1.3 cli\_3.2.0 tools\_4.1.3 magrittr\_2.0.3   
## [49] tibble\_3.1.6 crayon\_1.5.1 tidyr\_1.2.0 car\_3.0-12   
## [53] pkgconfig\_2.0.3 ellipsis\_0.3.2 assertthat\_0.2.1 rmarkdown\_2.10   
## [57] R6\_2.5.1 compiler\_4.1.3

# About the Authors

These credits are based on our [course contributors table guidelines](https://github.com/jhudsl/OTTR_Template/wiki/How-to-give-credits).

| Credits | Names |
| --- | --- |
| **Pedagogy** |  |
| Lead Content Instructor | Elizabeth Humphries |
| Content Idea Contributor | [Ava Hoffman](https://www.avahoffman.com/) |
| Content Directors | Kasper Hansen, [Jeff Leek](https://jtleek.com/), [Frederick Tan](https://carnegiescience.edu/frederick-tan) |
| **Production** |  |
| Content Publisher | [Ira Gooding](https://publichealth.jhu.edu/faculty/4130/ira-gooding) |
| **Technical** |  |
| Template Publishing Engineers | [Candace Savonen](https://www.cansavvy.com/), [Carrie Wright](https://carriewright11.github.io/), [Ava Hoffman](https://www.avahoffman.com/) |
| Publishing Maintenance Engineer | [Candace Savonen](https://www.cansavvy.com/) |
| Technical Publishing Stylists | [Carrie Wright](https://carriewright11.github.io/), [Candace Savonen](https://www.cansavvy.com/) |
| Package Developers ([ottrpal](https://github.com/jhudsl/ottrpal)) | [John Muschelli](https://johnmuschelli.com/), [Candace Savonen](https://www.cansavvy.com/), [Carrie Wright](https://carriewright11.github.io/) |
| **Funding** |  |
| Funder | [National Human Genome Research Institute (NHGRI)](https://www.genome.gov/) |
| Funding Staff | Fallon Bachman, Jennifer Vessio, Emily Voeglein |

## ─ Session info ───────────────────────────────────────────────────────────────  
## setting value  
## version R version 4.1.3 (2022-03-10)  
## os Ubuntu 20.04.5 LTS  
## system x86\_64, linux-gnu  
## ui X11  
## language (EN)  
## collate en\_US.UTF-8  
## ctype en\_US.UTF-8  
## tz Etc/UTC  
## date 2025-05-08  
## pandoc 2.17.1.1 @ /usr/local/bin/ (via rmarkdown)  
##   
## ─ Packages ───────────────────────────────────────────────────────────────────  
## package \* version date (UTC) lib source  
## bookdown 0.24 2023-03-22 [1] Github (rstudio/bookdown@88bc4ea)  
## cachem 1.0.6 2021-08-19 [1] RSPM (R 4.1.0)  
## callr 3.7.0 2021-04-20 [1] RSPM (R 4.1.0)  
## cli 3.2.0 2022-02-14 [1] RSPM (R 4.1.0)  
## crayon 1.5.1 2022-03-26 [1] RSPM (R 4.1.0)  
## desc 1.4.1 2022-03-06 [1] RSPM (R 4.1.0)  
## devtools 2.4.3 2021-11-30 [1] RSPM (R 4.1.0)  
## digest 0.6.29 2021-12-01 [1] RSPM (R 4.1.0)  
## ellipsis 0.3.2 2021-04-29 [1] RSPM (R 4.1.0)  
## evaluate 0.15 2022-02-18 [1] RSPM (R 4.1.0)  
## fastmap 1.1.0 2021-01-25 [1] RSPM (R 4.1.0)  
## fs 1.5.2 2021-12-08 [1] RSPM (R 4.1.0)  
## glue 1.6.2 2022-02-24 [1] RSPM (R 4.1.0)  
## htmltools 0.5.2 2021-08-25 [1] RSPM (R 4.1.0)  
## knitr 1.33 2023-03-22 [1] Github (yihui/knitr@a1052d1)  
## lifecycle 1.0.1 2021-09-24 [1] RSPM (R 4.1.0)  
## magrittr 2.0.3 2022-03-30 [1] RSPM (R 4.1.0)  
## memoise 2.0.1 2021-11-26 [1] RSPM (R 4.1.0)  
## pkgbuild 1.3.1 2021-12-20 [1] RSPM (R 4.1.0)  
## pkgload 1.2.4 2021-11-30 [1] RSPM (R 4.1.0)  
## prettyunits 1.1.1 2020-01-24 [1] RSPM (R 4.1.0)  
## processx 3.5.3 2022-03-25 [1] RSPM (R 4.1.0)  
## ps 1.6.0 2021-02-28 [1] RSPM (R 4.1.0)  
## purrr 0.3.4 2020-04-17 [1] RSPM (R 4.1.0)  
## R6 2.5.1 2021-08-19 [1] RSPM (R 4.1.0)  
## remotes 2.4.2 2021-11-30 [1] RSPM (R 4.1.0)  
## rlang 1.0.2 2023-03-22 [1] Github (r-lib/rlang@34b04a8)  
## rmarkdown 2.10 2023-03-22 [1] Github (rstudio/rmarkdown@02d3c25)  
## rprojroot 2.0.3 2022-04-02 [1] CRAN (R 4.1.3)  
## sessioninfo 1.2.2 2021-12-06 [1] RSPM (R 4.1.0)  
## stringi 1.7.6 2021-11-29 [1] RSPM (R 4.1.0)  
## stringr 1.4.0 2019-02-10 [1] RSPM (R 4.1.0)  
## testthat 3.0.1 2023-03-22 [1] Github (R-lib/testthat@e99155a)  
## usethis 2.1.6.9000 2023-03-22 [1] Github (r-lib/usethis@b4b8623)  
## withr 2.5.0 2022-03-03 [1] RSPM (R 4.1.0)  
## xfun 0.26 2023-03-22 [1] Github (yihui/xfun@74c2a66)  
## yaml 2.3.5 2022-02-21 [1] RSPM (R 4.1.0)  
##   
## [1] /usr/local/lib/R/site-library  
## [2] /usr/local/lib/R/library  
##   
## ──────────────────────────────────────────────────────────────────────────────

# 2 References