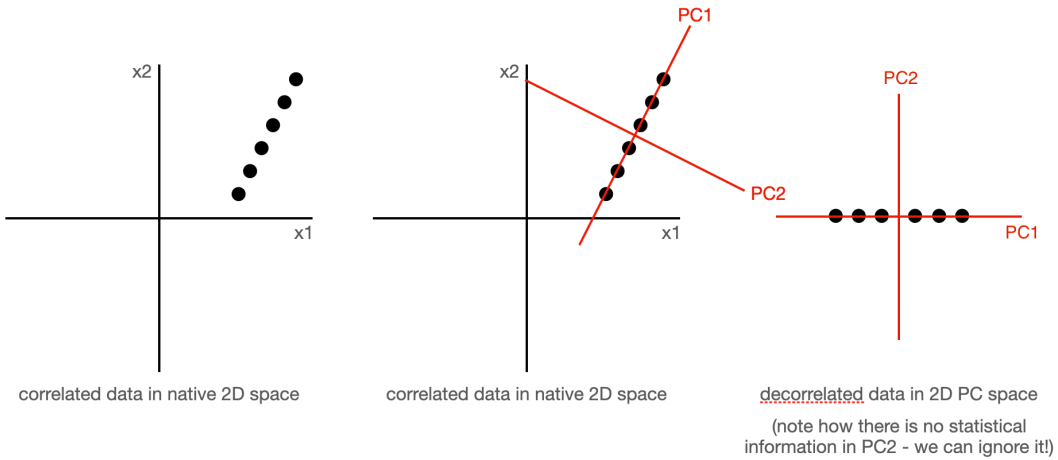
Principal Components Analysis

36-600

Principal Components Analysis

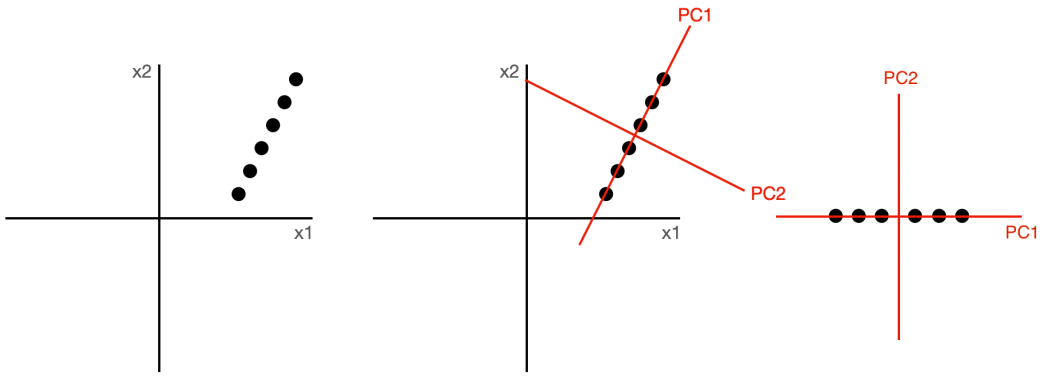
- Let's build up intuition for PCA by starting off with a picture:



- At left, we visualize the contents of a data frame with six rows (six data points) and two columns, x_1 and x_2
- What does the PCA algorithm do? It utilizes algorithms of linear algebra to move the coordinate system origin to the centroid of the point "cloud," then rotates the axes such that "PC 1" lays along the direction of greatest variation (i.e., along the points), and "PC 2" lays orthogonal to to "PC 1"
- In the end, PCA is the linear transformation of a coordinate system...

Principal Components Analysis

- ...but, there is actually more to PCA then just the transformation.



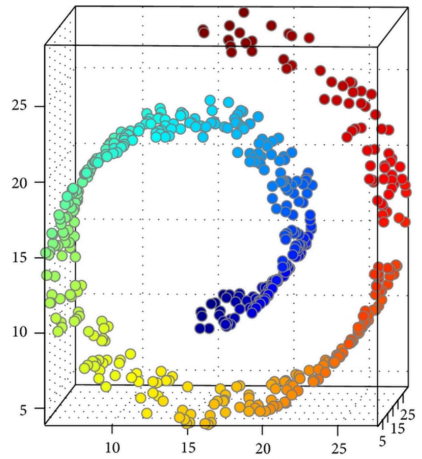
- At right, we move into "PC space," where PC1 has become the x axis, etc.
- Note how the data all lie along the PC1 axis
  - in PC space, we can see that the true space of the data is one-dimensional: a second dimension adds no statistical information at all
  - we can visualize and interpret the data using, e.g., a histogram of PC1 values and ignore PC2 entirely
- So in the end, PCA is also a dimension-reduction tool

Why Would We Use PCA?

- There are two principal (pun not intended) reasons to use PCA:
  - it can assist data visualization: if the predictor data lay within a subspace of the native space, we can try to visualize the data in that space
  - it can help statistical inference: PCA "decorrelates" the data — there is no linear association between the data along PC1 and PC2, etc. — so multicollinearity is mitigated!
- **NOTE:** *PCA is not appropriate for use with categorical data!*

PCA: It's a Linear Algorithm

- The main limitation of PCA is that it is a *linear* algorithm: it projects data to hyperplanes

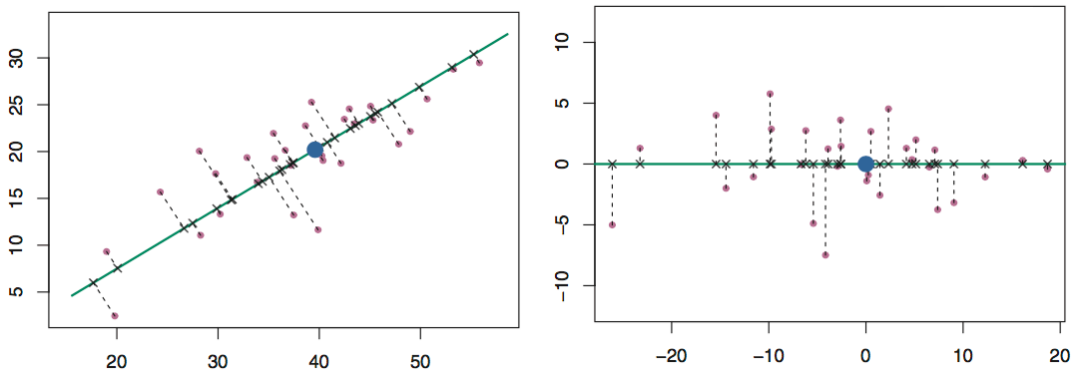


- These data lay on a curving two-dimensional strip embedded in a three-dimensional space
  - projecting these data to a two-dimensional plane or a one-dimensional line would effectively randomize the statistical information they contain
  - for visualization and/or learning, we could explore the use of *nonlinear* techniques like diffusion map or local linear embedding, or self-organizing maps or tSNE, etc.

PCA: It's Not Factor Analysis

- PCA is one of a family of related algorithms commonly dubbed *factor analyses*, and, for instance, some use the phrases "PCA" and "factor analysis" interchangeably
- The PCA algorithm as described here is a **deterministic algorithm**: it just ingests the data values as they are and produces a result, full stop
- *Exploratory factor analysis* is a variant of PCA which *does* attempt to take the random variability of the data into account
  - it maps the p observed variables to $k < p$ factors; "exploratory" means we do not know *a priori* how that mapping may occur
- *Confirmatory factor analysis* basically adds on a hypothesis-testing layer to EFA for confirming that links actually exist between the k factors uncovered by EFA and the p original variables

PCA: Algorithm



- The *score*, or the coordinate of the i^{th} observation along PC j is

$$Z_{ij} = \sum_{k=1}^{p} X_{ik}\phi_{kj}$$

  - k represents the k^{th} variable or feature (i.e., the k^{th} column of your [predictor] data frame)
  - ϕ is the rotation (or *loading*) matrix, which is normalized such that $\sum_{k=1}^{p} \phi_{kj}^2 = 1$

- Note that since we are "mixing axes" when doing PCA, it is generally best to standardize (or scale) the data frame before applying the algorithm

PCA: Choosing the Number of Dimensions to Retain

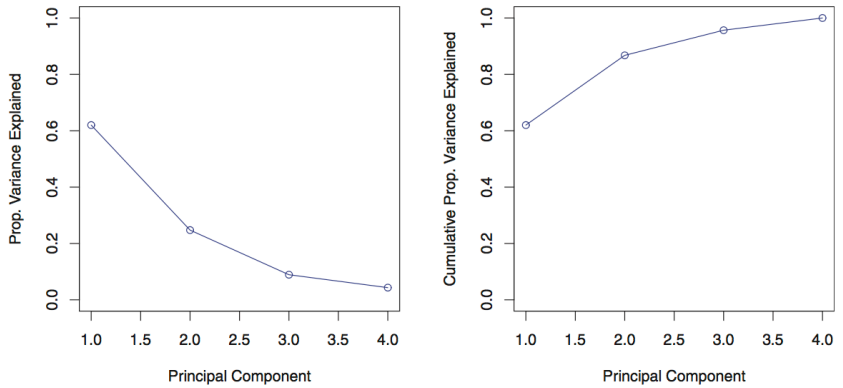
- Ideally, we choose $M < p$ such that

$$x_{ij} = \sum_{m=1}^{p} z_{im}\phi_{jm} \approx \sum_{m=1}^{M} z_{im}\phi_{jm}$$

  - in other words, we don't lose much ability to reconstruct the input data X by dropping the last $p - M$ principal components, which we assume represent random variation in the data (i.e., noise)

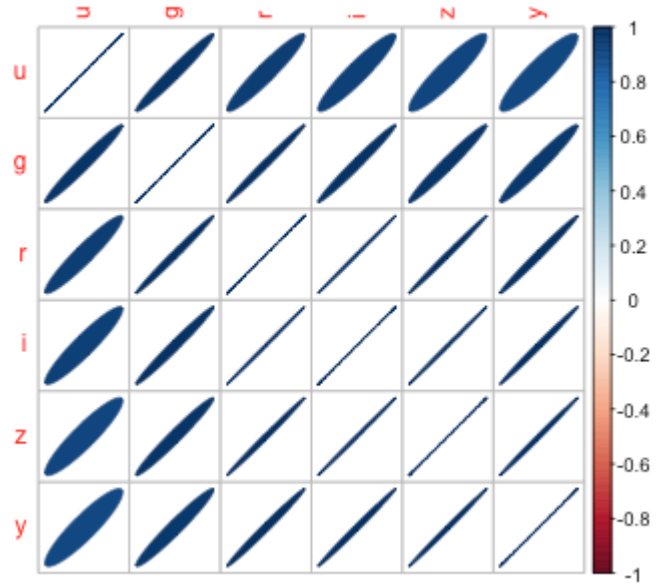
- One convention is to sum up the amount of variance explained in the first M PCs, and adopt the smallest value of M such that 90% or 95% or 99%, etc., of the overall variance is "explained"

- Another convention is to look for an "elbow" in the left plot below (the "scree plot")



PCA: Example

- Let's suppose we have a catalog of galaxies that contains measures of brightness in different wavelength bands, from u (for ultraviolet) to z and y in the near-infrared



- The six brightness metrics are obviously highly correlated!

PCA: Example

```
pca.out <- prcomp(df[,-7],scale=TRUE) # do not include response variable!
v       <- pca.out$sdev^2
round(cumsum(v/sum(v)),3)
```

```
## [1] 0.977 0.998 0.999 1.000 1.000 1.000
```

- The first principal component explains 97.7% of the variance in the dataset: the data can safely be transformed from a six-dimensional space to a one-dimensional one with minimal loss of statistical information
- Let's print the column(s) for the PCs that we retain:

```
round(pca.out$rotation[,1],3)
```

```
##     u     g     r     i     z     y
## 0.396 0.411 0.413 0.412 0.410 0.408
```

- Here, we see that all the original variables contribute almost equally to the first PC (sign doesn't matter)
  - we conclude that the predictor variables (for all intents and purposes) lie along a one-dimensional line in the native six-dimensional space, and that the orientation of the line is such that it spans all six of the original predictor variable axes

PCA: Example

- What information is in the second PC?

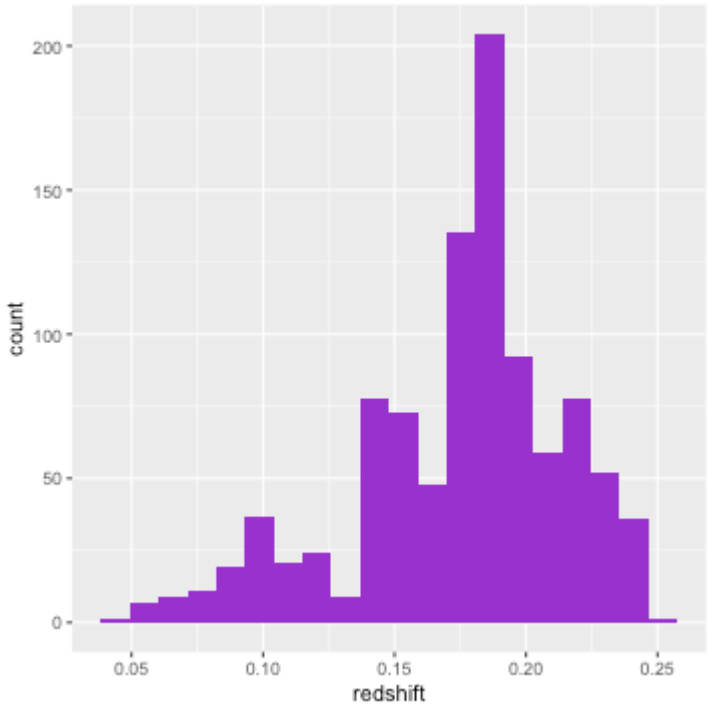
```
round(pca.out$rotation[,1:2],3)
```

```
##     PC1    PC2
## u 0.396  0.793
## g 0.411  0.247
## r 0.413 -0.089
## i 0.412 -0.205
## z 0.410 -0.320
## y 0.408 -0.398
```

- The second PC primarily maps to u-band magnitude...basically, the PC2 vector, orthogonal to the PC1 vector, points largely along the u-band axis, meaning there is some data variability along that axis that PC1 does not "pick up"

PC Regression

- Let's demonstrate how we would learn a principal components regression model using the distance of a galaxy from the Sun as a response variable



PC Regression

- Reminder: why are we doing this?
  - to mitigate multicollinearity
  - so we can "trust" the standard error estimates output in the summary

PC Regression

- Let's split the data:

```
set.seed(101)
s <- sample(nrow(df),round(0.7*nrow(df)))
```

- The test-set mean-squared error for a linear-regression model learned in the data's native space:

```
lm.out  <- lm(redshift~.,data=df,subset=s)
lm.pred <- predict(lm.out,newdata=df[-s,])
mean((lm.pred-df$redshift[-s])^2)
```

```
## [1] 0.0008416416
```

- The test-set mean-squared error for a PC regression model learned using all PCs:

```
df.pc <- data.frame(pca.out$x,df$redshift)
names(df.pc) <- c("PC1","PC2","PC3","PC4","PC5","PC6","y")
lm.out <- lm(y~.,data=df.pc,subset=s)
lm.pred <- predict(lm.out,newdata=df.pc[-s,])
mean((lm.pred-df.pc$y[-s])^2)
```

```
## [1] 0.0008416416
```

- The test-set MSEs are the same!

PC Regression

- What happens if we learn a model with just the first two PCs, the ones that account for 99.8% of the data variability?

```
lm.out <- lm(y~PC1+PC2,data=df.pc,subset=s)
lm.pred <- predict(lm.out,newdata=df.pc[-s,])
mean((lm.pred-df.pc$y[-s])^2)
```

```
## [1] 0.001437671
```

- The test-set MSE *increases*: there can be/will be a tradeoff between enhanced model interpretability and decreased model predictive ability if we summarily remove PCs
    - PC regression learns models with "shifted" predictor variable values: with sufficient sample size, this can markedly impact prediction results, even if the retained PCs represent an overwhelming proportion of the cumulative variance

PC Regression

- What happens if we utilize `bestglm`?

```
suppressMessages(library(bestglm))
bg.out <- bestglm(df.pc,IC="BIC")
bg.out$BestModel
```

```
##
## Call:
## lm(formula = y ~ ., data = data.frame(Xy[, c(bestset[-1], FALSE),
##     drop = FALSE], y = y))
##
## Coefficients:
## (Intercept)          PC1          PC2          PC3          PC4          PC5
##     0.17455      0.00569      0.02354     -0.09395      0.25060     -0.45007
##         PC6
##    -0.74064
```

- Upshot: the best set of PCs for visualization/intuition-building is not necessarily the best set of PCs for modeling!

PC Regression

- Let's summarize the "best" model

```
summary(bg.out$BestModel)$coefficients
```

```
##                Estimate   Std. Error    t value     Pr(>|t|)
## (Intercept)  0.174550826 0.0009369600 186.294847 0.000000e+00
## PC1          0.005690051 0.0003872504  14.693466 2.389390e-44
## PC2          0.023536423 0.0025977839   9.060193 6.830750e-19
## PC3         -0.093945739 0.0118547286  -7.924748 6.147270e-15
## PC4          0.250602201 0.0179780474  13.939345 1.901929e-40
## PC5         -0.450074996 0.0369510332 -12.180309 6.773436e-32
## PC6         -0.740638090 0.0671668890 -11.026833 9.593036e-27
```

- We see if we change, e.g., PC2 by 1 unit, the distance will change by 0.024...but how does that change map back to the original variables?
    - so while it can be worthwhile to shift to PC coordinates to mitigate multicollinearity, *crafting an inferential data story that refers back to the original variables can be difficult* (and this would be the reason why we wouldn't "just do" PC regression all the time!)