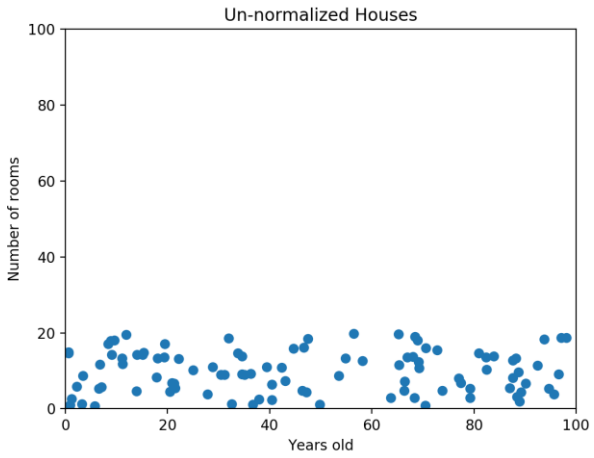
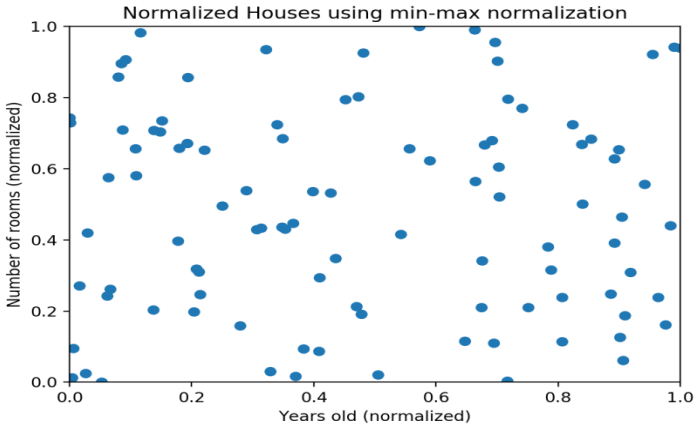
Tutorial – Week 2: Pre-processing (normalisation, z-score, PCA)

Why Normalize?

Many machine learning algorithms attempt to find trends in the data by comparing features of data points. However, there is an issue when the features are on drastically different scales. For example, consider a dataset of houses. Two potential features might be the number of rooms in the house, and the total age of the house in years. A machine learning algorithm could try to predict which house would be best for you. However, when the algorithm compares data points, the feature with the larger scale will completely dominate the other. Take a look at the image below:



When the data looks squished like that, we know we have a problem. The machine learning algorithm should realize that there is a huge difference between a house with 2 rooms and a house with 20 rooms. But right now, because two houses can be 100 years apart, the difference in the number of rooms contributes less to the overall difference. As a more extreme example, imagine what the graph would look like if the x-axis was the cost of the house. The data would look even more squeezed; the difference in the number of rooms would be even less relevant because the cost of two houses could have a difference of thousands of GBP. The goal of normalization is to make every datapoint have the same scale so each feature is equally important. The image below shows the same house data normalized using min-max normalization.



Min-Max Normalization

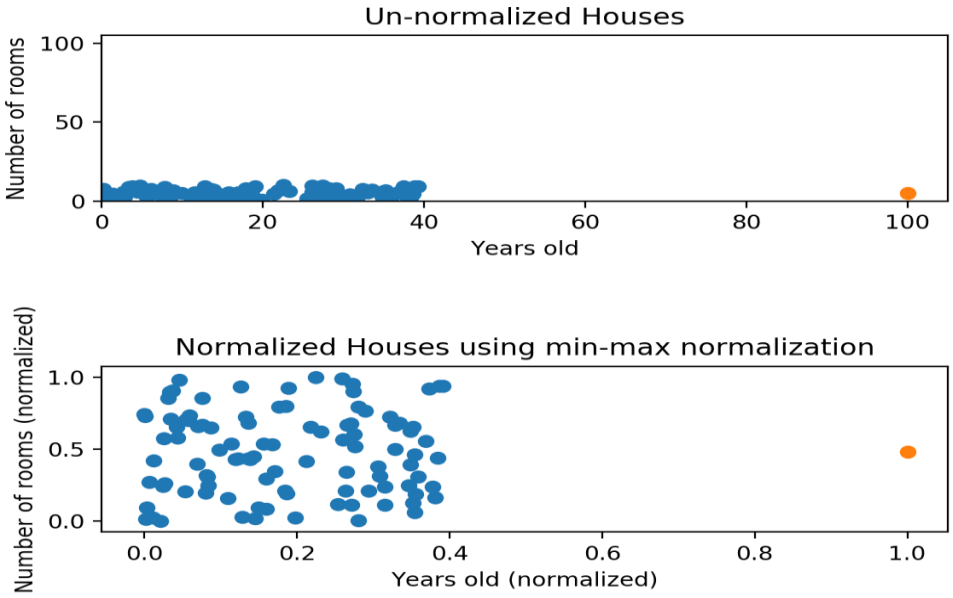
Min-max normalization is one of the most common ways to normalize data. For every feature, the minimum value of that feature gets transformed into a 0, the maximum value gets transformed into a 1, and every other value gets transformed into a decimal between 0 and 1. For example, if the

minimum value of a feature was 20, and the maximum value was 40, then 30 would be transformed to about 0.5 since it is halfway between 20 and 40. The formula is as follows:

Norm_value = (real_value – min) / (max-min)

Check yourselves how this formula has been derived (we can discuss it at Tutorial session as well)

Min-max normalization has one fairly significant downside: it does not handle outliers very well. For example, if you have 99 values between 0 and 40, and one value is 100, then the 99 values will all be transformed to a value between 0 and 0.4. That data is just as squished as before! Take a look at the image below to see an example of this.



Normalizing fixed the squeezing problem on the y-axis, but the x-axis is still problematic. Now if we were to compare these points, the y-axis would dominate; the y-axis can differ by 1, but the x-axis can only differ by 0.4.

The above normalisation formula, works only if we have 1 and 0 as the max and min normalised values. What is the case for general normalisation?

For more generic normalisation, here is the generic formula:

$$v' = \frac{v - \min_A}{\max_A - \min_A}(new\_\max_A - new\_\min_A) + new\_\min_A$$

Suppose, we have 4 numbers: 8, 10, 15, 20, and we wish to normalise them

For Marks as 8:

$$MinMax = \frac{(V - Min\ marks)}{Max\ omarks - Min\ marks}(newMax - newMin)^{\square} + newMin$$

$$MinMax = \frac{(8 - 8)}{20 - 8} * (1 - 0)^{\square} + 0$$

$$MinMax = \frac{(0)}{12} * 1$$

$$MinMax = 0$$

For Marks as 10:

$$MinMax = \frac{(10 - 8)}{20 - 8} * (1 - 0)^{\square} + 0$$

$$MinMax = \frac{(2)}{12} * 1$$

$$MinMax = 0.16$$

For Marks as 15:

$$MinMax = \frac{(15 - 8)}{20 - 8} * (1 - 0)^{\square} + 0$$

$$MinMax = \frac{(7)}{12} * 1$$

$$MinMax = \mathbf{0.58}$$

For Marks as 20:

$$MinMax = \frac{(20 - 8)}{20 - 8} * (1 - 0)^{\square} + 0$$

$$MinMax = \frac{(12)}{12} * 1$$

$$MinMax = 1$$

What are z-score?

A z-score measures exactly how many standard deviations above or below the mean a data point is. Here's the formula for calculating a z-score:

$$z = \frac{\text{data point} - \text{mean}}{\text{standard deviation}}$$

Here's the same formula written with symbols:

$$z = \frac{x - \mu}{\sigma}$$

Here are some important facts about z-scores:

- A positive z-score says the data point is above average.
- A negative z-score says the data point is below average.
- A z-score close to 0 says the data point is close to average.

Suppose, we have the same 4 numbers: 8, 10, 15, 20, and we wish to find their z- score

$$s = \sqrt{\frac{\sum(x - \bar{x})^2}{n}}$$

$$\text{Standard deviation} = \sqrt{\frac{\sum(\text{every individual value of marks} - \text{mean of marks})^2}{n}}$$

Mean of marks = 8 + 10 + 15 + 20 /4 = 13.25

$$= \sqrt{\frac{(8 - 13.25)^2 + (10 - 13.25)^2 + (15 - 13.25)^2 + (20 - 13.25)^2}{4}}$$

$$= \sqrt{\frac{(-5.25)^2 + (-3.25)^2 + (1.75)^2 + (6.75)^2}{4}}$$

$$= \sqrt{\frac{27.56 + 10.56 + 3.06 + 45.56}{4}} = \sqrt{\frac{86.74}{4}} = \sqrt{21.6} = 4.6$$

$$ZScore = \frac{x - \mu}{\sigma} = \frac{8 - 13.25}{4.6} = -1.14$$

$$ZScore = \frac{x - \mu}{\sigma} = \frac{10 - 13.25}{4.6} = -0.7$$

$$ZScore = \frac{x - \mu}{\sigma} = \frac{15 - 13.25}{4.6} = 0.3\dot{}$$

$$ZScore = \frac{x - \mu}{\sigma} = \frac{20 - 13.25}{4.6} = 1.4$$

Advantages of the z-score

The z-score is a very useful statistic of the data due to the following facts;

- It allows a data administrator to understand the probability of a score occurring within the normal distribution of the data.
- The z-score enables a data administrator to compare two different scores that are from different normal distributions of the data.

Linear Algebra - Basics

Find all the eigenvalues for the given matrices

$$A = \begin{bmatrix} 6 & 16 \\ -1 & -4 \end{bmatrix}$$

First we'll need to define the matrix λI-A

$$\lambda I - A = \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix} - \begin{bmatrix} 6 & 16 \\ -1 & -4 \end{bmatrix} = \begin{bmatrix} \lambda - 6 & -16 \\ 1 & \lambda + 4 \end{bmatrix}$$

Next we need the determinant of this matrix, which gives us the characteristic polynomial.

$$\det(\lambda I - A) = (\lambda - 6)(\lambda + 4) - (-16) = \lambda^2 - 2\lambda - 8$$

Now, set this equal to zero and solve for the eigenvalues.

$$\lambda^2 - 2\lambda - 8 = (\lambda - 4)(\lambda + 2) = 0 \qquad \Rightarrow \qquad \lambda_1 = -2,\ \lambda_2 = 4$$

So, we have two eigenvalues and since they occur only once in the list they are both simple eigenvalues.

For each eigenvalue we will need to solve the system,

$$(\lambda I - A)\mathbf{x} = \mathbf{0}$$

to determine the general form of the eigenvector. Once we have that we can use the general form of the eigenvector to find a basis for the eigenspace.

We know that the eigenvalues for this matrix are $\lambda_1 = -2$ and $\lambda_2 = 4$.

Let's first find the eigenvector(s) and eigenspace for $\lambda_1 = -2$. Referring to Example 2 for the formula for $\lambda I - A$ and plugging $\lambda_1 = -2$ into this we can see that the system we need to solve is,

$$\begin{bmatrix} -8 & -16 \\ 1 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

We'll leave it to you to verify that the solution to this system is,

$$x_1 = -2t \qquad\qquad x_2 = t$$

Therefore, the general eigenvector corresponding to $\lambda_1 = -2$ is of the form,

$$\mathbf{x} = \begin{bmatrix} -2t \\ t \end{bmatrix} = t \begin{bmatrix} -2 \\ 1 \end{bmatrix}$$

The eigenspaces is all vectors of this form and so we can see that a basis for the eigenspace corresponding to $\lambda_1 = -2$ is,

$$\mathbf{v}_1 = \begin{bmatrix} -2 \\ 1 \end{bmatrix}$$

Now, let's find the eigenvector(s) and eigenspace for $\lambda_2 = 4$. Plugging $\lambda_2 = 4$ into the formula for $\lambda I - A$ from Example 2 gives the following system we need to solve,

$$\begin{bmatrix} -2 & -16 \\ 1 & 8 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

The solution to this system is (you should verify this),

$$x_1 = -8t \qquad x_2 = t$$

The general eigenvector and a basis for the eigenspace corresponding to $\lambda_2 = 4$ is then,

$$\mathbf{x} = \begin{bmatrix} -8t \\ t \end{bmatrix} = t \begin{bmatrix} -8 \\ 1 \end{bmatrix} \qquad \& \qquad \mathbf{v}_2 = \begin{bmatrix} -8 \\ 1 \end{bmatrix}$$

Note that if we wanted our hands on specific eigenvalues for each eigenvector the basis vector for each eigenspace would work. So, if we do that we could use the following eigenvectors (and their corresponding eigenvalues) if we'd like.

$$\lambda_1 = -2 \quad \mathbf{v}_1 = \begin{bmatrix} -2 \\ 1 \end{bmatrix}, \quad \lambda_2 = 4 \quad \mathbf{v}_2 = \begin{bmatrix} -8 \\ 1 \end{bmatrix}$$

Note as well that these eigenvectors are linearly independent vectors.

PCA – simple numerical example:

Let us analyse the following 3-variate dataset with 10 observations. Each observation consists of 3 measurements on a wafer: thickness, horizontal displacement, and vertical displacement. Normally, we use computational methods to calculate PCA, as it is rather computational expensive.

$$\mathbf{x} = \begin{bmatrix} 7 & 4 & 3 \\ 4 & 1 & 8 \\ 6 & 3 & 5 \\ 8 & 6 & 1 \\ 8 & 5 & 7 \\ 7 & 2 & 9 \\ 5 & 3 & 3 \\ 9 & 5 & 8 \\ 7 & 4 & 5 \\ 8 & 2 & 2 \end{bmatrix}$$

First compute the correlation matrix.

$$\mathbf{R} = \begin{bmatrix} 1.00 & 0.67 & -0.10 \\ 0.67 & 1.00 & -0.29 \\ -0.10 & -0.29 & 1.00 \end{bmatrix}$$

Next solve for the roots of $\mathbf{R}$ using software.

	λ value	proportion
1	1.769	0.590
2	0.927	0.899
3	0.304	1.000

- Each eigenvalue satisfies $|\mathbf{R} - \lambda\mathbf{I}| = 0$.
- The sum of the eigenvalues $= 3 = p$, which is equal to the trace of $\mathbf{R}$ (i.e., the sum of the main diagonal elements).
- The determinant of $\mathbf{R}$ is the product of the eigenvalues.
- The product is $\lambda_1 \times \lambda_2 \times \lambda_3 = 0.499$.

Substituting the first eigenvalue of 1.769 and $\mathbf{R}$ in the appropriate equation we obtain

$$\begin{bmatrix} -0.769 & 0.670 & -0.100 \\ 0.670 & -0.769 & -0.290 \\ -0.100 & -0.290 & -0.769 \end{bmatrix} \begin{bmatrix} v_{11} \\ v_{21} \\ v_{31} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}.$$

This is the matrix expression for three homogeneous equations with three unknowns and yields the first column of $\mathbf{V}$: 0.64 0.69 -0.34 (again, a computerized solution is indispensable).

Repeating this procedure for the other two eigenvalues yields the matrix $\mathbf{V}$.

$$\mathbf{V} = \begin{bmatrix} 0.64 & 0.38 & -0.66 \\ 0.69 & 0.10 & 0.72 \\ -0.34 & 0.91 & 0.20 \end{bmatrix}$$

Notice that if you multiply $\mathbf{V}$ by its transpose, the result is an identity matrix, $\mathbf{V}'\mathbf{V} = \mathbf{I}$.

Now form the matrix $\mathbf{L}^{1/2}$, which is a diagonal matrix whose elements are the square roots of the eigenvalues of $\mathbf{R}$. Then obtain $\mathbf{S}$, the factor structure, using $\mathbf{S} = \mathbf{V}\mathbf{L}^{1/2}$.

$$\begin{bmatrix} 0.64 & 0.38 & -0.66 \\ 0.69 & 0.10 & 0.72 \\ -0.34 & 0.91 & 0.20 \end{bmatrix} \begin{bmatrix} 1.33 & 0 & 0 \\ 0 & 0.96 & 0 \\ 0 & 0 & 0.55 \end{bmatrix} = \begin{bmatrix} 0.85 & 0.37 & -0.37 \\ 0.91 & 0.10 & 0.40 \\ -0.45 & 0.88 & 0.11 \end{bmatrix}$$

So, for example, 0.91 is the correlation between the second variable and the first principal component.

Next compute the communality, using the first two eigenvalues only.

$$\mathbf{SS}' = \begin{bmatrix} 0.85 & 0.37 \\ 0.91 & 0.09 \\ -0.45 & 0.88 \end{bmatrix} \begin{bmatrix} 0.85 & 0.91 & -0.45 \\ 0.37 & 0.09 & 0.88 \end{bmatrix} = \begin{bmatrix} 0.8662 & 0.8140 & -0.0606 \\ 0.8140 & 0.8420 & -0.3321 \\ -0.0606 & -0.3321 & 0.9876 \end{bmatrix}$$

Communality consists of the diagonal elements.

	var
1	0.8662
2	0.8420
3	0.9876

This means that the first two principal components "explain" 86.62 % of the first variable, 84.20 % of the second variable, and 98.76 % of the third.

The coefficient matrix, $\mathbf{B}$, is formed using the reciprocals of the diagonals of $\mathbf{L}^{1/2}$.

$$\mathbf{B} = \mathbf{V}\mathbf{L}^{-1/2} = \begin{bmatrix} 0.48 & 0.40 & -1.20 \\ 0.52 & 0.10 & 1.31 \\ -0.26 & 0.95 & 0.37 \end{bmatrix}$$

Finally, we can compute the factor scores from $\mathbf{ZB}$, where $\mathbf{Z}$ is $\mathbf{X}$ converted to standard score form. These columns are the *principal factors*.

$$\mathbf{F} = \mathbf{ZB} = \begin{bmatrix} 0.41 & -0.69 & 0.06 \\ -2.11 & 0.07 & 0.63 \\ -0.46 & -0.32 & 0.30 \\ 1.62 & -1.00 & 0.70 \\ 0.70 & 1.09 & 0.65 \\ -0.86 & 1.32 & -0.85 \\ -0.60 & -1.31 & 0.86 \\ 0.94 & 1.72 & -0.04 \\ 0.22 & 0.03 & 0.34 \\ 0.15 & -0.91 & -2.65 \end{bmatrix}$$

(i.e. standard score form means to scale it via z-score)

R- Implementations

Why normalize or scale the data?

There can be instances found in data frame where values for one feature could be in the range 1-100, and values for other feature in the range 1-10000000. In scenarios like these, owing to mere greater numeric range, the impact on response variables by the feature having greater numeric range could be more than the one having less numeric range, and this could, in turn, impact prediction accuracy. The objective is to improve predictive accuracy and not allow a particular feature impact the prediction due to large numeric value range. Thus, we may need to normalize or scale values under different features such that they fall under common range. Take a look at following example:

In these (and next) tutorials, **codes are provided in blue colour**. **Make sure that you have installed the appropriate packages before calling them via the library command.**

```
# Age vector
age <- c(25, 35, 50)
# Salary vector
salary <- c(200000, 1200000, 2000000)
# Data frame created using age and salary
df <- data.frame( "Age" = age, "Salary" = salary, stringsAsFactors = FALSE)
df
 #Age  Salary
 # 25  200000
 # 35 1200000
 # 50 2000000
```

Pay attention on how values for age and salary vary in different ranges.

Min-Max Normalization

Data frame could be normalized using Min-Max normalization technique which is specified by the following formula to be applied on each value of features to be normalized.

$(X - \min(X))/(\max(X) - \min(X))$

This is equivalent to the following R function

```
normalize <- function(x) {
   return ((x - min(x)) / (max(x) - min(x)))
  }
```

In order to apply the normalize function on each of the features of the above data frame (df), the following code could be used. Pay attention to usage of lapply function (check R documentation for further info).

```
dfNorm <- as.data.frame(lapply(df, normalize))
# One could also use sequence such as df[1:2]
dfNorm <- as.data.frame(lapply(df[1:2], normalize))
dfNorm
```

In case, one wish to specify a set of features such as salary, following formula could be used:

```
# Note df[2]
dfNorm <- as.data.frame(lapply(df[2], normalize))
#  or  df["Salary"]
```

```
dfNorm <- as.data.frame(lapply(df["Salary"], normalize))
```
If we wish to use the full normalization formula, then we need to create an additional function

```
new_normalize <- function(x, new_max=1,new_min=0)  # see how we define the max min values
{
  a= (((x-min(x))* (new_max-new_min))/(max(x)-min(x)))+new_min
  return(a)
}
fNorm1 <- as.data.frame(lapply(df[1:2], new_normalize))
dfNorm1
```

Z-Score Standardization

The disadvantage with min-max normalization technique is that it tends to bring data towards the mean. If there is a need for outliers to get weighted more than the other values, z-score standardization technique suits better. In order to achieve z-score standardization, one could use **R's built-in scale()** function. Take a look at following example where scale function is applied on "df" data frame mentioned above.

```
dfNormZ <- as.data.frame( scale(df[1:2] ))
dfNormZ
```
The output is given as:

```
        Age      Salary
1 -0.9271726 -1.03490978
2 -0.1324532  0.07392213
3  1.0596259  0.96098765
```

The scale function performs the z_score. If we wish to create our own z-score function, we can do it easily.

```
z_score = function(x) {
  return((x - mean(x)) / sd(x))
}
dfNorm4 <- as.data.frame(lapply(df, z_score))
dfNorm4
```
We get exactly the same as in the scale function.

Tutorial for PCA

Principal Component Analysis (PCA) involves the process by which principal components are computed, and their role in understanding the data. PCA is an unsupervised approach, which means that it is performed on a set of variables $X_1, X_2, \ldots, X_p$ with no associated response Y. PCA reduces the dimensionality of the data set, allowing most of the variability to be explained using fewer variables. PCA is commonly used as one step in a series of analyses. You can use PCA to reduce the number of variables when you have too many predictors relative to the number of observations.

Our first attempt in PCA

This first exercise primarily leverages the USArrests data set that is built into R. This is a set that 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. The data set also contains the percentage of the population living in urban areas, *UrbanPop*.

```
library(tidyverse)   # data manipulation and visualization  (first load tidyverse and then rlang!)
library(gridExtra)  # plot arrangement
library(ggplot2)     # for ggplot and qplot
library(rlang)       # is needed in tidyverse package
```

```
data("USArrests")
head(USArrests, 10)
```

```
##            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
```

Preparing the Data

It is usually beneficial for each variable to be centred at zero for PCA, due to the fact that it makes comparing each principal component to the mean straightforward. This also eliminates potential problems with the scale of each variable. For example, the variance of Assault is 6945, while the variance of Murder is only 18.97. The Assault data isn't necessarily more variable, it's simply on a different scale relative to Murder.

```
# compute variance of each variable
apply(USArrests, 2, var)
```

The second input here denotes whether we wish to compute the mean of the rows, 1, or the columns, 2. Here we are interested for the columns (attributes)

```
##     Murder    Assault   UrbanPop       Rape
##   18.97047 6945.16571  209.51878   87.72916
```

Standardizing each variable will fix this issue.

```
# create new data frame with centered variables
scaled_df <- apply(USArrests, 2, scale)
head(scaled_df)
```

```
##         Murder   Assault   UrbanPop        Rape
## [1,] 1.24256408 0.7828393 -0.5209066 -0.003416473
## [2,] 0.50786248 1.1068225 -1.2117642  2.484202941
## [3,] 0.07163341 1.4788032  0.9989801  1.042878388
## [4,] 0.23234938 0.2308680 -1.0735927 -0.184916602
## [5,] 0.27826823 1.2628144  1.7589234  2.067820292
## [6,] 0.02571456 0.3988593  0.8608085  1.864967207
```

However, keep in mind that there may be instances where scaling is not desirable. An example would be if every variable in the data set had the same units and the analyst wished to capture this difference in variance for his/her results. Since *Murder*, *Assault*, and *Rape* are all measured on occurrences per 100,000 people this may be reasonable depending on how you want to interpret the results. But since *UrbanPop* is measured as a percentage of total population it wouldn't make sense to compare the variability of *UrbanPop* to *Murder*, *Assault*, and *Rape*. The 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.*

Principal Components

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 pair-wise 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 $p(p-1)/2=45$ scatter plots that could be analyzed! Clearly, a better method is required to visualize the *n* observations when *p* is large. 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. 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. Each of the dimensions found by PCA is a linear combination of the *p* features and 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.

Therefore, in order to calculate principal components, we start by using the cov() function to calculate the covariance matrix, followed by the eigen command to calculate the eigenvalues of the matrix. Eigen produces an object that contains both the ordered eigenvalues ($values) and the corresponding eigenvector matrix ($vectors).

```
# Calculate eigenvalues & eigenvectors
arrests.cov <- cov(scaled_df)
arrests.eigen <- eigen(arrests.cov)
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 ...
```

For our example, we'll take the first two sets of loadings and store them in the matrix phi

```
# Extract the loadings
(phi <- arrests.eigen$vectors[,1:2])
```

```
##            [,1]       [,2]
## [1,] -0.5358995  0.4181809
## [2,] -0.5831836  0.1879856
## [3,] -0.2781909 -0.8728062
```

```
## [4,] -0.5434321 -0.1673186
```

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 because it leads to more logical interpretation of graphical results as we'll see shortly. To use the positive-pointing vector, we multiply the default loadings by -1. The set of loadings for the first principal component (*PC1*) and second principal component (*PC2*) are shown below:

```
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
```

Each principal component vector defines a direction in feature space. Because eigenvectors are orthogonal to every other eigenvector, loadings and therefore principal components are **uncorrelated** with one another, and form a basis of the new space. This holds true no matter how many dimensions are being used.

By examining the principal component vectors above, 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. If we project the n data points $x_1,...,x_n$ onto the first eigenvector, the projected values are called the principal component *scores* for each observation

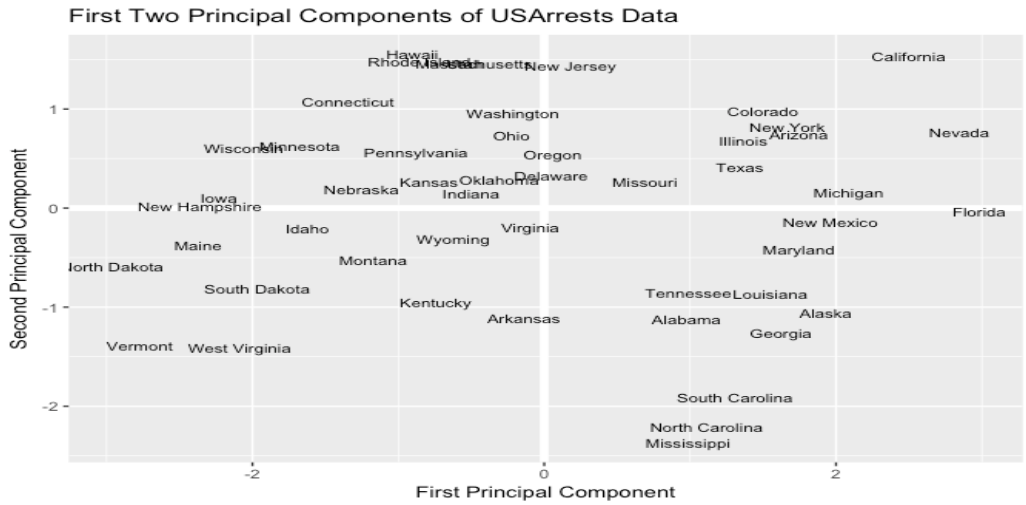
```
# Calculate Principal Components scores
PC1 <- as.matrix(scaled_df) %*% phi[,1]      # %*% in R does matrix multiplication
PC2 <- as.matrix(scaled_df) %*% phi[,2]

# Create 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
```

Now that we've calculated the first and second principal components for each US state, we can plot them against each other and produce a two-dimensional view 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.

```
# Plot Principal Components for each State
ggplot(PC, aes(PC1, PC2)) +
  modelr::geom_ref_line(h = 0) +
  modelr::geom_ref_line(v = 0) +
  geom_text(aes(label = State), size = 3) +
  xlab("First Principal Component") +
  ylab("Second Principal Component") +
  ggtitle("First Two Principal Components of USArrests Data")
```



Because PCA is unsupervised, this analysis on its own is not making predictions about crime rates, but simply making connections between observations using fewer measurements.

Selecting the Number of Principal Components

Note that in this 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?

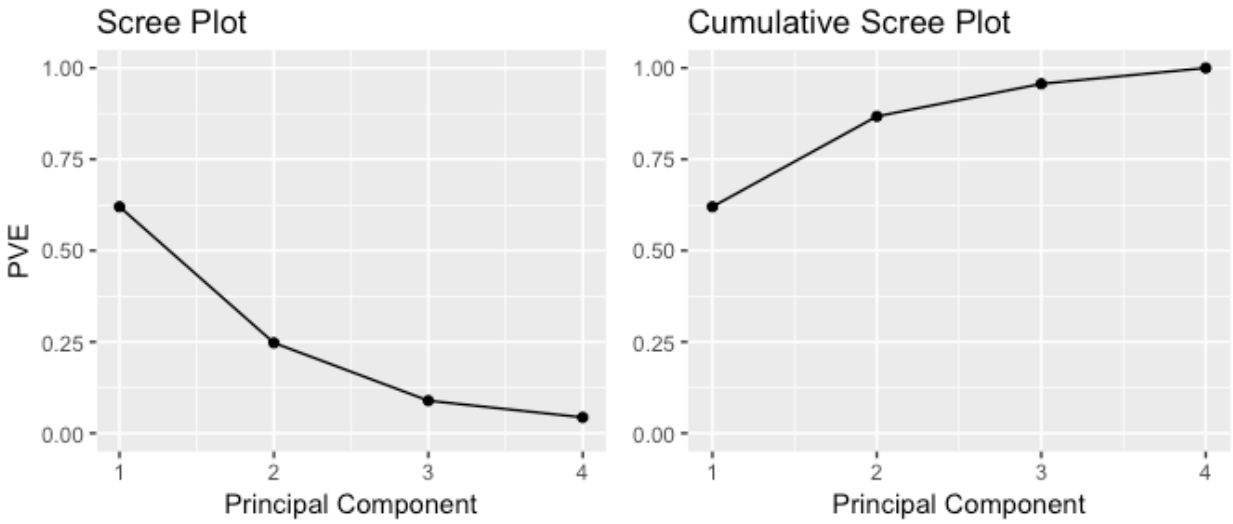
The Proportion of Variance Explained

We have mentioned that PCA reduces the dimensionality while explaining most of the variability, but there is a more technical method for measuring exactly what percentage of the variance was retained in these principal components. By performing some algebra, the proportion of variance explained (PVE) by the *m*th principal component can be calculated. It can be shown that the PVE of the *m*th principal component can be more simply calculated by taking the *m*th eigenvalue and dividing it by the number of principal components, *p*. A vector of PVE for each principal component is calculated:

```
PVE <- arrests.eigen$values / sum(arrests.eigen$values)
round(PVE, 2)
## [1] 0.62 0.25 0.09 0.04
```

The first principal component in this example explains 62% of the variability, and the second principal component explains 25%. Together, the first two principal components explain 87% of the variability. It is often advantageous to plot the PVE and cumulative PVE, for reasons explained in the following section of this tutorial. The plot of each is shown below.

```
# PVE (aka scree) plot
PVEplot <- qplot(c(1:4), PVE) +
  geom_line() +
  xlab("Principal Component") +
  ylab("PVE") +
  ggtitle("Scree Plot") +
  ylim(0, 1)

# Cumulative PVE plot
cumPVE <- qplot(c(1:4), cumsum(PVE)) +
  geom_line() +
  xlab("Principal Component") +
  ylab(NULL) +
  ggtitle("Cumulative Scree Plot") +
  ylim(0,1)

grid.arrange(PVEplot, cumPVE, ncol = 2)
```



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 that can be calculated. 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. In reality, 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 shown above and stored in the ggplot object PVEplot.

(A **Scree Plot** is a simple line segment **plot** that shows the fraction of total variance in the data as explained or represented by each PC. The PCs are ordered, and by definition are therefore assigned a number label, by decreasing order of contribution to total variance).

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.

Built-in PCA Functions

In the above example we manually computed many of the attributes of PCA (i.e. eigenvalues, eigenvectors, principal components scores). This was meant to help you learn about PCA by understanding what manipulations are occurring with the data; however, using this process to repeatedly perform PCA may be a bit tedious. Fortunately R has several built-in functions (along with numerous add-on packages) that simplify 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.

```
pca_result <- prcomp(USArrests, scale = TRUE)
names(pca_result)
## [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
pca_result$center
##   Murder  Assault UrbanPop  Rape
##    7.788  170.760   65.540   21.232

# standard deviations
pca_result$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.

```
pca_result$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
```

We can see that there are four distinct principal components. This is to be expected because there are in general min(n−1,p) informative principal components in a data set with *n* observations and *p* variables. Also, notice that *PC1* and *PC2* are opposite signs from what we computed earlier. Recall that by default, eigenvectors in R point in the negative direction. We can adjust this with a simple change.

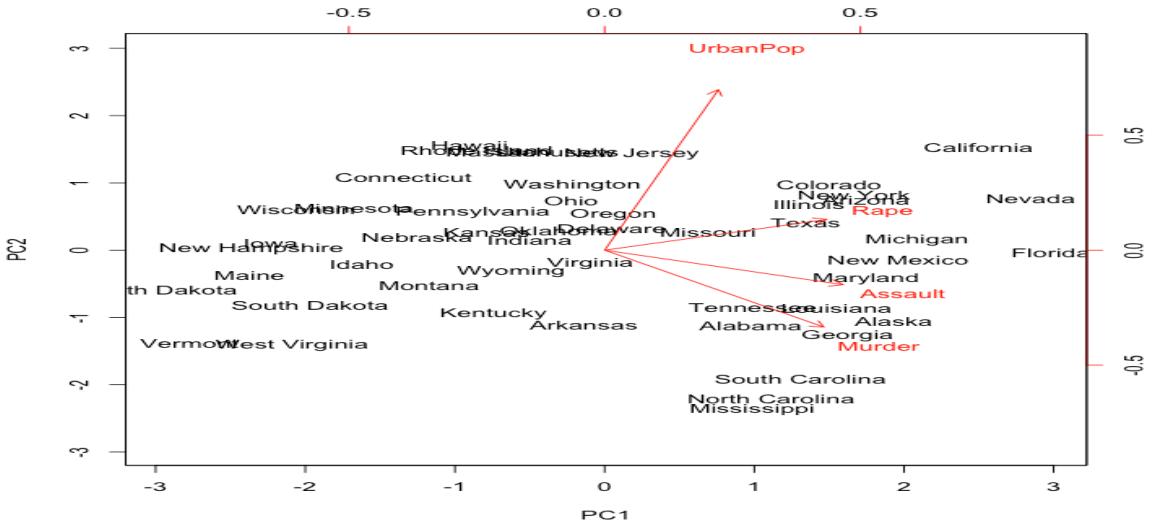
```
pca_result$rotation <- -pca_result$rotation
pca_result$rotation
##              PC1               PC2       PC3        PC4
## Murder   0.5358995 -0.4181809  0.3412327 -0.64922780
## Assault  0.5831836 -0.1879856  0.2681484  0.74340748
## UrbanPop 0.2781909  0.8728062  0.3780158 -0.13387773
## Rape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

Now our *PC1* and *PC2* match what we computed earlier. We can also obtain the principal components scores from our results as these are stored in the *x* list item of our results. However, we also want to make a slight sign adjustment to our scores to point them in the positive direction.

```
pca_result$x <- - pca_result$x
head(pca_result$x)
##                PC1       PC2        PC3         PC4
## Alabama    0.9756604 -1.1220012  0.43980366 -0.154696581
## Alaska     1.9305379 -1.0624269 -2.01950027  0.434175454
## Arizona    1.7454429  0.7384595 -0.05423025  0.826264240
## Arkansas  -0.1399989 -1.1085423 -0.11342217  0.180973554
## California 2.4986128  1.5274267 -0.59254100  0.338559240
## Colorado   1.4993407  0.9776297 -1.08400162 -0.001450164
```

Now we can plot the first two principal components using biplot. Alternatively, if you wanted to plot principal components 3 vs. 4 you can include choices = 3:4 within biplot (the default is choices = 1:2). The output is very similar to what we produced earlier; however, you'll notice the labeled errors which indicate the directional influence each variable has on the principal components. The scale = 0 argument to biplot ensures that the arrows are scaled to represent the loadings; other values for scale give slightly different biplots with different interpretations.

```
biplot(pca_result, scale = 0)
```



The prcomp function also outputs the standard deviation of each principal component.

```
pca_result$sdev
## [1] 1.5748783 0.9948694 0.5971291 0.4164494
```

The variance explained by each principal component is obtained by squaring these values:

```
(VE <- pca_result$sdev^2)
## [1] 2.4802416 0.9897652 0.3565632 0.1734301
```

To compute the proportion of variance explained by each principal component, we simply divide the variance explained by each principal component by the total variance explained by all four principal components:

```
PVE <- VE / sum(VE)
```

```
round(PVE, 2)
## [1] 0.62 0.25 0.09 0.04
```

As before, we see that that the first principal component explains 62% of the variance in the data, the next principal component explains 25% of the variance, and so forth. And we could proceed to plot the PVE and cumulative PVE to provide us our scree plots as we did earlier.

Extra Homework

Computing the principal components in R is straightforward with the functions prcomp() and princomp(). The difference between the two is simply the method employed to calculate PCA.
For princomp: The calculation is done using eigen on the correlation or covariance matrix, as determined by cor.
For prcomp: The calculation is done by singular value decomposition (SVD) of the (centered and possibly scaled) data matrix, not by using eigen on the covariance matrix. This is generally the preferred method for numerical accuracy.

Try this:

```
new_pca =princomp(~Murder + Assault + UrbanPop + Rape, data= USArrests, cor=TRUE)
summary(new_pca, loadings=TRUE)
predict(new_pca)
screeplot(new_pca,type="lines")
biplot(new_pca)
```