Unsupervised Learning

Byteflow Dynamics

10/8/2017

Contents

1 What is machine learning?

Machine learning refers to different methods of learning from data using a computer. There are two main different types of machine learning.

Supervised Learning

Supervised learning is basically pattern recognition. You train your algorithm with a set of *labeled* data, then the trained algorithm will be used to predict labels for new data points.

Majority of machine learning problems are supervised learning, and they can be:

- Classification
- Regression

Unsupervised Learning

Unsupervised learning is used when data is not labeled. It's used for data exploration to find patterns or structure in the data. The most common types are:

- Principal Component Analysis
- Clustering

2 Clustering

Clustering is a set of techniques used to divide an unlabeled data set into distinct groups (clusters). Observations within the same cluster are similar to each other (homogeneous), and observations in different groups are different from each other.

There are several clustering techniques that use different methods to find clusters in a data set.

- K-means clustering
- Hierarchical clustering

3 K-Means Clustering

K-means clustering is a method to partition observations in a data set into K distinct clusters. The algorithm works as follows:

1. Assign each observation to a cluster *randomly*
2. Calculate the centroid of each cluster
3. Reassign each observation to the cluster whose centroid is closest
4. Repeat steps 2 & 3 until the cluster assignments stop changing

4 Example: K-means clustering

Let's use a built-in data set to try the K-means algorithm.

```
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
```

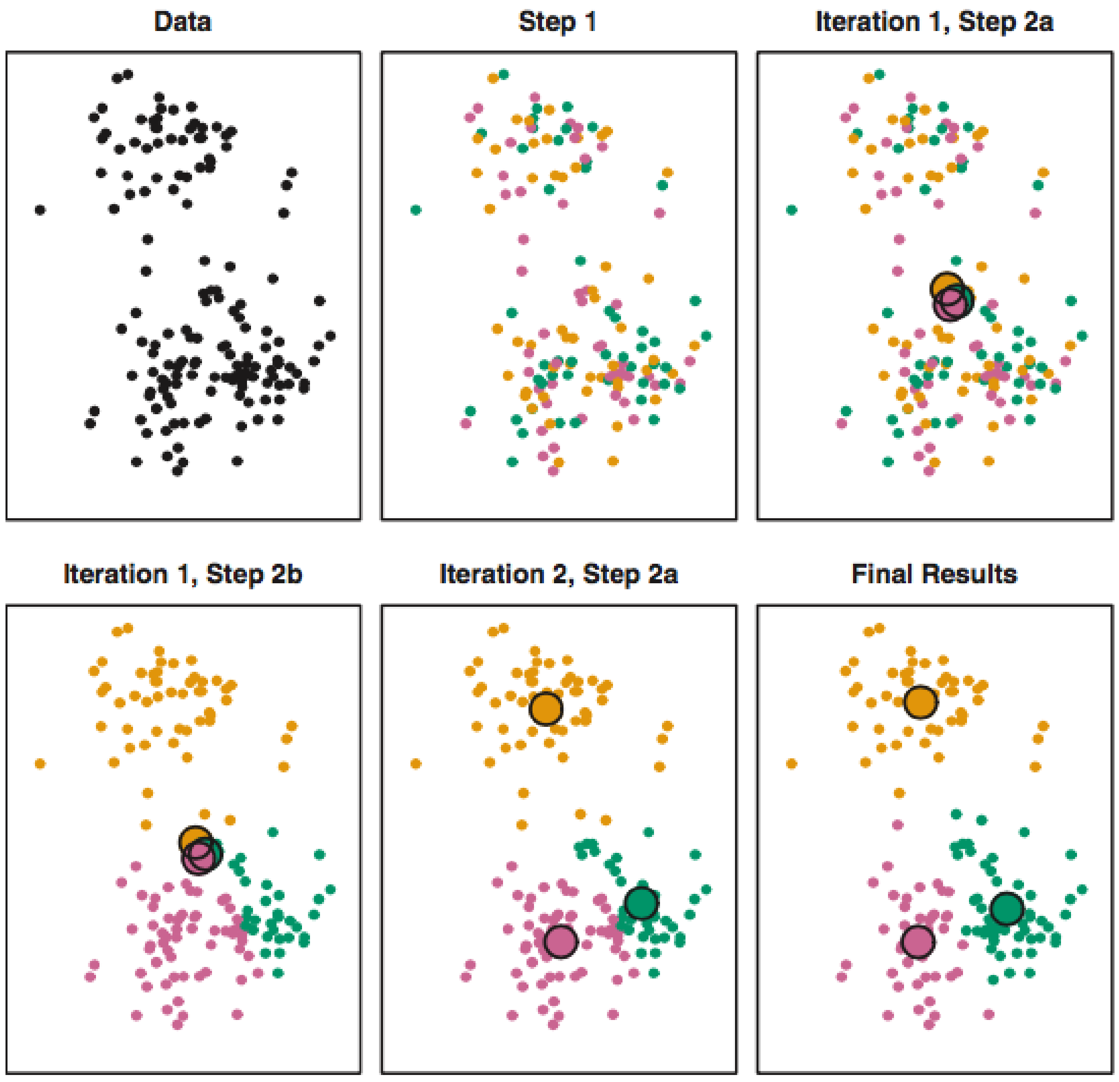
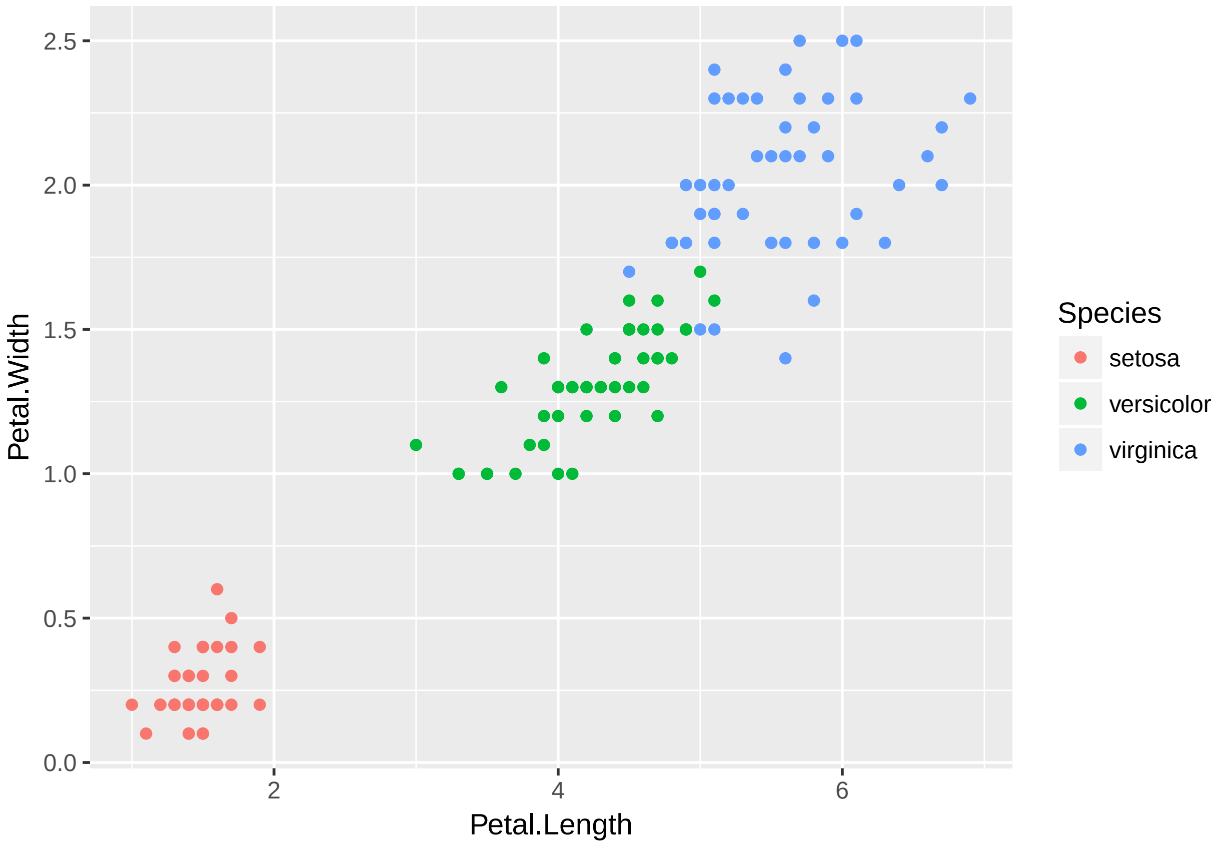


Figure 1: The progress of the K-means algorithm

4.0.1 Do some data exploration. Open the data to check, read about the data by typing ?iris, plot the data, etc. (3 min)

You may have found that Petal.Length and Petal.Width are similar among the same species but varied considerably between different species. To demonstrate it, make a plot.

```
ggplot(data = iris) +
  geom_point(mapping = aes(Petal.Length, Petal.Width, color = Species))
```



Now let's try if we can cluster the iris data based on Petal.Length and Petal.Width to get these clusters!

5 Find optimal number of clusters

K-means clustering method requires you to specify the number of clusters K. To find the optimal number of clusters for the data, we will calculate Within-Sum-of-Squares (WSS). WSS is a measure to explain the homogeneity within a cluster. A good clustering will have small WSS.

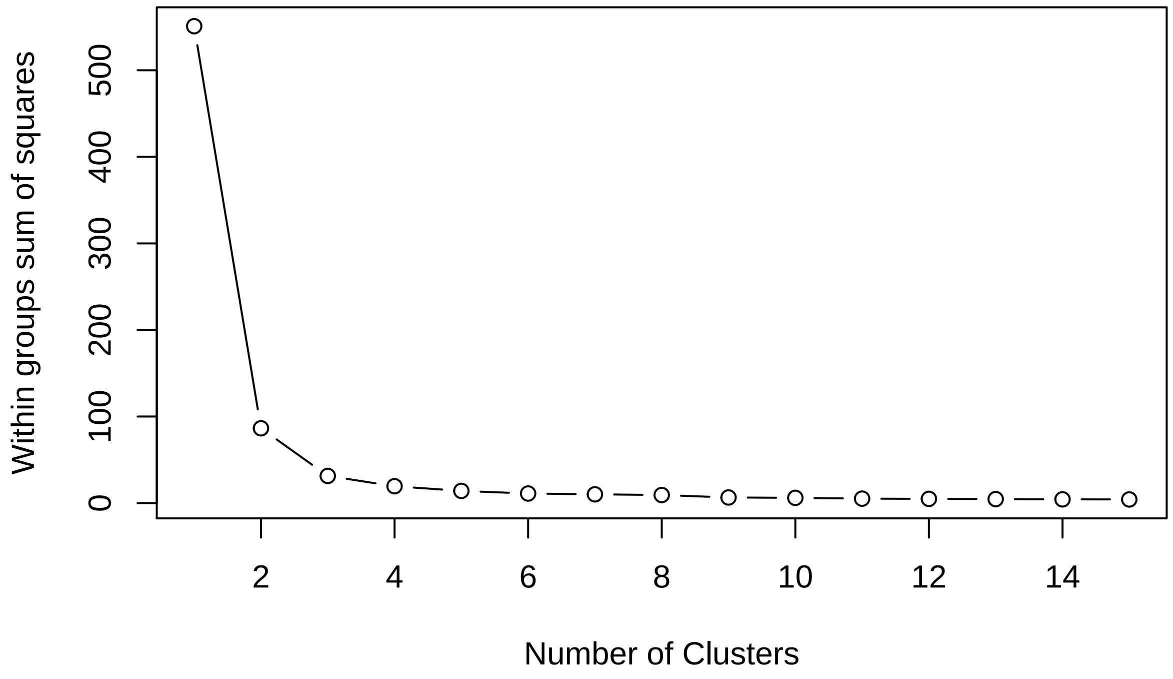
Unfortunately, there is no simple way to do this. Let's define a wssplot function that calculates wss and plots it.

```
wssplot <- function(data, nc=15, seed=1234){
  wss <- (nrow(data)-1)*sum(apply(data,2,var))
  for (i in 2:nc){
     set.seed(seed)
     wss[i] <- sum(kmeans(data, centers=i)$withinss)}
  plot(1:nc, wss, type="b", xlab="Number of Clusters",
  ylab="Within groups sum of squares")}
```

nc is the maximum number of clusters we consider, and seed is random initialization of clusters. As long as you set the same seed, you will get the same result every time you run it (reproducible).

wssplot is a function, so if you run it once, you can use it easily.

```
wssplot(iris[, 3:4])
```



We want wss to be small (homogeneous clusters). We can see that after 3 clusters, increasing the number of clusters doesn't affect wss much. So in this case, the optimal number of clusters is 3.

6 Perform K-means clustering

We use a funtion *kmeans( )* to perform K-means clustering. The template is as follows:

```
cluster <- kmeans(<data>, <K>, nstart = <n>)
```

Replace with your data, with the number of clusters, and with the number of initial configurations.

For reproducibility, we will set the seed. set.seed( ) generates random numbers, which will be useful when running simulations to ensure all results are reproducible. *nstart* option attempts multiple initial configurations and reports best one (more efficient!).

```
set.seed(20)
irisCluster <- kmeans(iris[, 3:4], 3, nstart = 20)
irisCluster
```

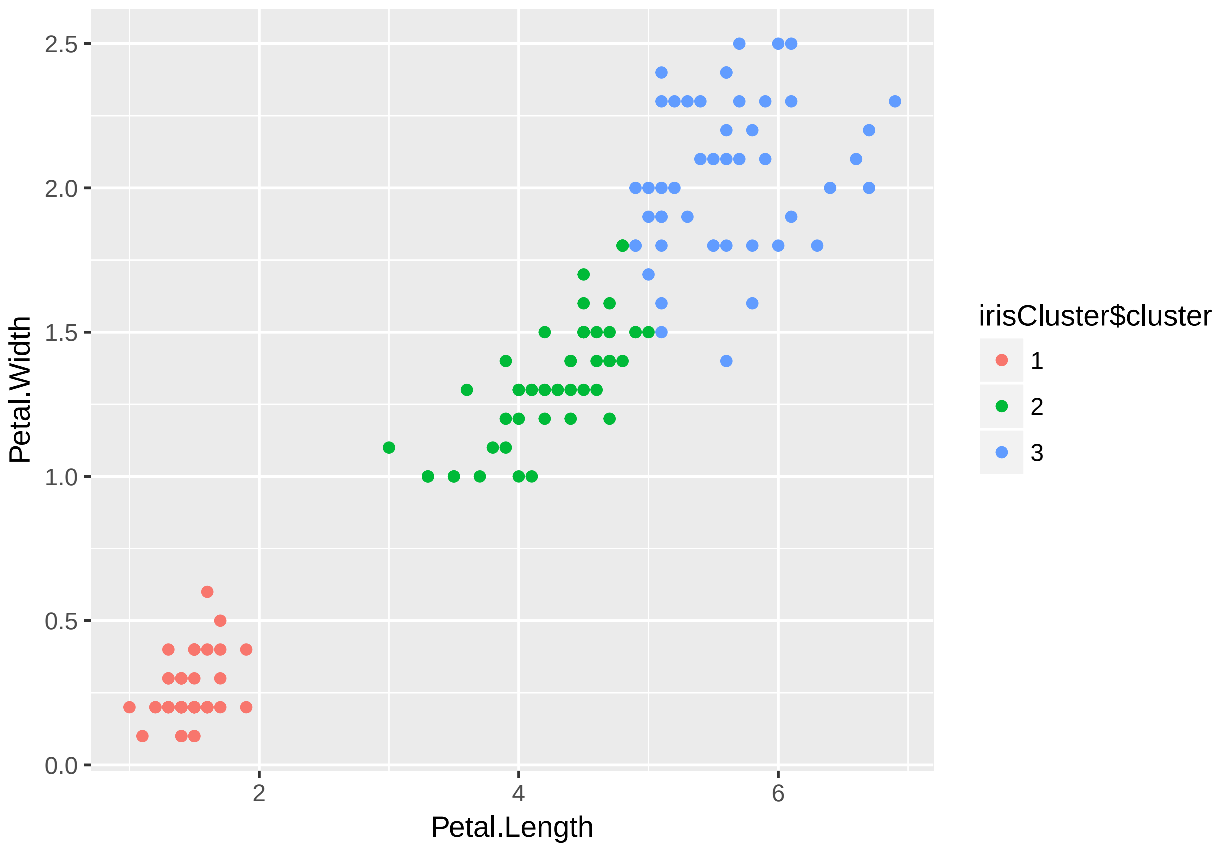
```
## K-means clustering with 3 clusters of sizes 50, 52, 48
##
## Cluster means:
##   Petal.Length Petal.Width
## 1     1.462000    0.246000
## 2     4.269231    1.342308
## 3     5.595833    2.037500
##
## Clustering vector:
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [36] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
##  [71] 2 2 2 2 2 2 2 3 2 2 2 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3
## [106] 3 2 3 3 3 3 3 3 3 3 3 3 3 3 2 3 3 3 3 3 3 2 3 3 3 3 3 3 3 3 3 3 3 2 3
## [141] 3 3 3 3 3 3 3 3 3 3
##
## Within cluster sum of squares by cluster:
## [1]  2.02200 13.05769 16.29167
##  (between_SS / total_SS =  94.3 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"
## [5] "tot.withinss" "betweenss"    "size"         "iter"
## [9] "ifault"
```

```
table(irisCluster$cluster, iris$Species)
```

```
##
##     setosa versicolor virginica
##   1     50          0         0
##   2      0         48         4
##   3      0          2        46
```

You can see that all setosa species are in cluster 1, most of versicolor is in cluster 2, most of virginica in cluster 3. That means we were able to cluster the iris data based on Petal lentgh and Petal width into 3 clusters which correspond to species.

```
irisCluster$cluster <- as.factor(irisCluster$cluster)
ggplot(iris, aes(Petal.Length, Petal.Width, color = irisCluster$cluster)) + geom_point()
```



The plot looks very similar to the one with species as color.

7 Excercise: K-means Clustering

Cluster the same iris dataset based on Petal.Length and Sepal.Length and check if they correspond to species. Try different numbers of clusters to see how the result changes.

```
set.seed(20)
irisCluster <- kmeans(iris[, c(1,3)], 3, nstart = 20)
irisCluster
```

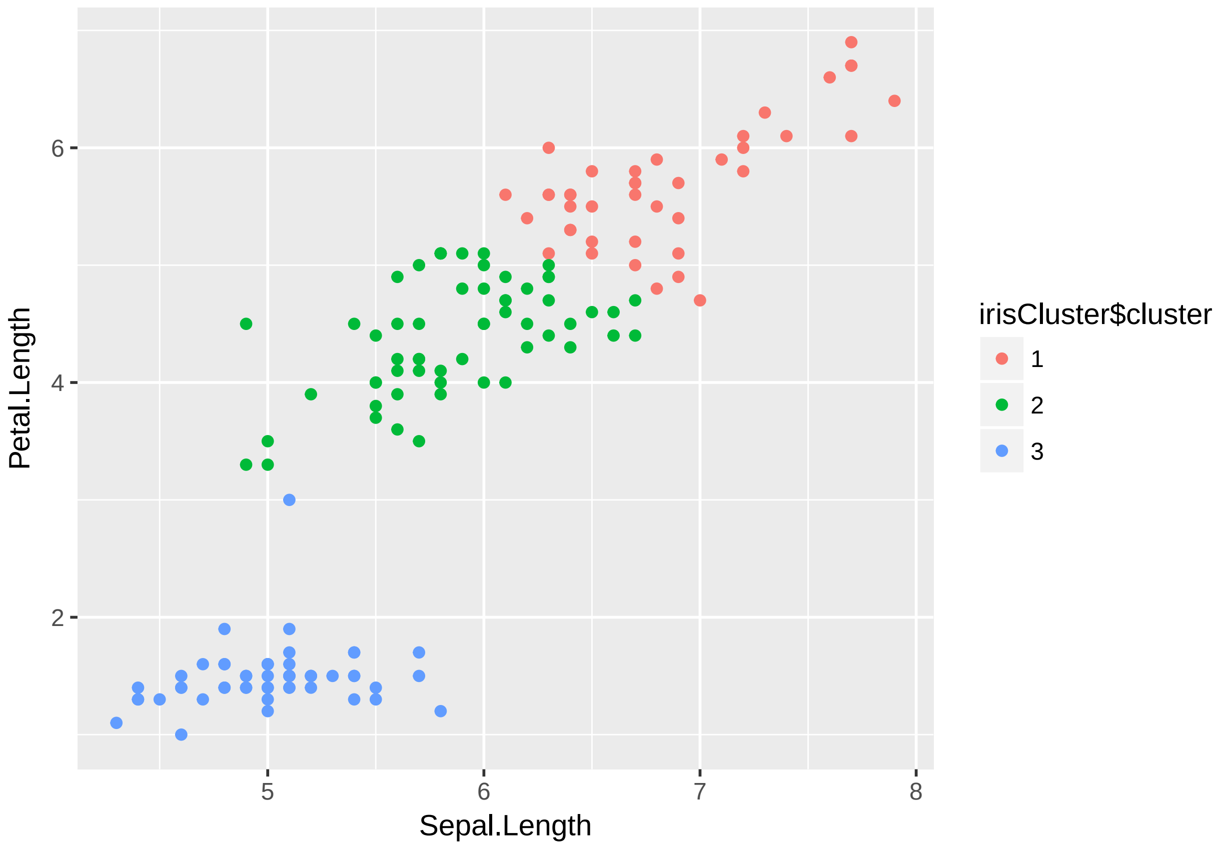
```
## K-means clustering with 3 clusters of sizes 41, 58, 51
##
## Cluster means:
##   Sepal.Length Petal.Length
## 1     6.839024     5.678049
## 2     5.874138     4.393103
## 3     5.007843     1.492157
##
## Clustering vector:
##   [1] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
##  [36] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 1 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
##  [71] 2 2 2 2 2 2 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 2 1 2 1 1 1
## [106] 1 2 1 1 1 1 1 1 2 2 1 1 1 1 2 1 2 1 2 1 1 2 2 1 1 1 1 1 1 1 1 1 1 2 1
## [141] 1 1 2 1 1 1 2 1 1 2
```

```
##
## Within cluster sum of squares by cluster:
## [1] 20.407805 23.508448  9.893725
##  (between_SS / total_SS =  90.5 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"
## [5] "tot.withinss" "betweenss"    "size"         "iter"
## [9] "ifault"
```

```
table(irisCluster$cluster, iris$Species)
```

```
##
##     setosa versicolor virginica
##   1      0          4        37
##   2      0         45        13
##   3     50          1         0
```

```
irisCluster$cluster <- as.factor(irisCluster$cluster)
ggplot(iris, aes(Sepal.Length, Petal.Length, color = irisCluster$cluster)) + geom_point()
```



Sepal.Length and Petal.Length do a decent job at clustering the iris data, but not as well as Petal Length & Width.

8 Principal Component Analysis (PCA)

PCA is a popular approach used to reduce dimensionality. When we have a large set of *correlated* variables, PCA can summarize it with a smaller number of *uncorrelated* variables (principal components) that can collectively explain most of the variability found in the original set.

PCA in unsupervised learning problems to discover and visualize patterns in high dimensional unlabeled datasets.

8.0.1 How is PCA useful?

- Supervised Learning
  - Avoids overfitting
  - Avoids multicollinearity
- Unsupervised Learning
  - Makes it easier to visualize data
  - Chooses variables for clustering

9 Sample data

Let's look at a sample data set.

```r
library(datasets)
cars <- mtcars[,-1]
head(cars)
```

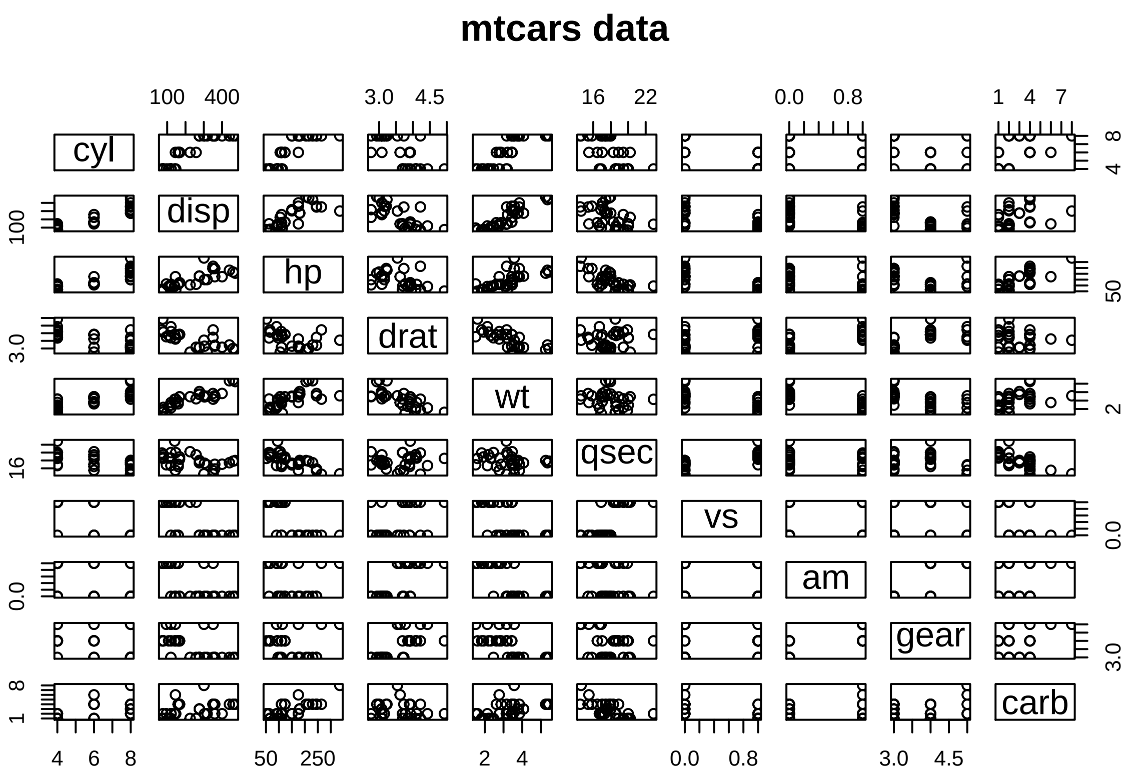
```
##                   cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4           6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag       6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710          4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive      6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant             6  225 105 2.76 3.460 20.22  1  0    3    1
```

9.0.1 Do some data exploration. Open the data to check, read about the data by typing ?mtcars, plot the data, etc. (3 min)

---

mtcars contains fuel consumption and automobile design and performance data for 32 cars extracted from the 1974 Motor Trend US magazine. It contains 32 observations (rows) and 10 variables (columns). It is not a big dataset but even in a dataset this size, it is challenging to visualize patterns among variables.

```r
require(graphics)
pairs(cars, main = "mtcars data")
```



Above plot is a pairwise scatter plot of all variables in mtcars. These are a lot of scatter plots to go through…imagine you had more variables in your dataset!

PCA comes in handy to solve this problem!

10 How does PCA work?

PCA reduces the number of variables in your data. It will be easier to find patterns in the lower-dimensional data. Below is how to find principal components.

10.0.1 First Principal Component

The first principal component Z_1 calculated by PCA is a normalized, linear combination of the variables $X_1, X_2, ..., X_p$ that has the largest variance. The variables are centered to have mean zero (column average of all variables are zero).

Check column means for mtcars data.

```
summary(cars)
```

```
##       cyl             disp             hp             drat      
##  Min.   :4.000   Min.   : 71.1   Min.   : 52.0   Min.   :2.760  
##  1st Qu.:4.000   1st Qu.:120.8   1st Qu.: 96.5   1st Qu.:3.080  
##  Median :6.000   Median :196.3   Median :123.0   Median :3.695  
##  Mean   :6.188   Mean   :230.7   Mean   :146.7   Mean   :3.597  
##  3rd Qu.:8.000   3rd Qu.:326.0   3rd Qu.:180.0   3rd Qu.:3.920  
##  Max.   :8.000   Max.   :472.0   Max.   :335.0   Max.   :4.930  
##        wt             qsec             vs               am        
##  Min.   :1.513   Min.   :14.50   Min.   :0.0000   Min.   :0.0000  
##  1st Qu.:2.581   1st Qu.:16.89   1st Qu.:0.0000   1st Qu.:0.0000
##  Median :3.325   Median :17.71   Median :0.0000   Median :0.0000
##  Mean   :3.217   Mean   :17.85   Mean   :0.4375   Mean   :0.4062
##  3rd Qu.:3.610   3rd Qu.:18.90   3rd Qu.:1.0000   3rd Qu.:1.0000
##  Max.   :5.424   Max.   :22.90   Max.   :1.0000   Max.   :1.0000
##       gear             carb
##  Min.   :3.000   Min.   :1.000
##  1st Qu.:3.000   1st Qu.:2.000
##  Median :4.000   Median :2.000
##  Mean   :3.688   Mean   :2.812
##  3rd Qu.:4.000   3rd Qu.:4.000
##  Max.   :5.000   Max.   :8.000
```

The mean for **disp** is much larger than most other variables. If we do not center these variables, the principal components will be greatly influenced by the **disp** variable.

The first principal component Z_1:

$Z_1 = \phi_{11}X_1 + \phi_{21}X_2 + ... + \phi_{p1}X_p$

$\phi_1, \phi_2, ..., \phi_p$ are called the *loadings* of the first principal component. The loadings are constrained so that

$\Sigma_{j=1}^{p}\phi_{j1}^2 = 1$

Since Z_1 and $X_1, X_2, ...$ are vectors, we can write each element of Z_1 as:

$z_{i1} = \phi_{11}x_{i1} + \phi_{21}x_{i2} + ... + \phi_{p1}x_{ip}$

where z_{i1} is called the first principal component scores, ϕ's are the first principal component loading vectors, and x's are the variable values for each observation centered to have mean zero.

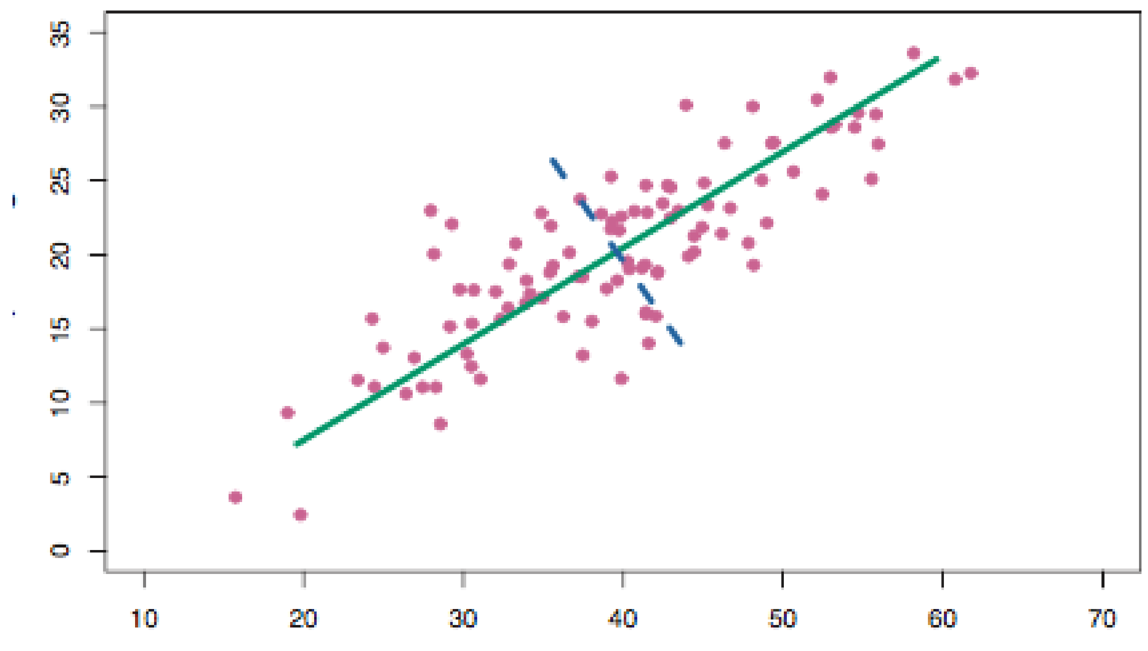


Figure 2: The green line represents the first principal component loading vector for a sample dataset

10.0.2 Second Principal Component

The second principal component Z_2 is also a linear combination of the variables that has the largest variance, but **not correlated to** Z_1. This means Z_2 is orthogonal to Z_1 (blue dashed line in above figure).

10.0.3 Higher dimensional data

If your data set has more than 2 variables, there are multiple principal components which can be found in a similar way. In general, the number of Principal Components created is the minimum of either n-1 where n is the number of observations (rows) in the dataset or p which is the number of variables (columns) in your data. So in our mtcars dataset, the minimum value between 31 (number of observations - 1) and 10 (number of variables in the cars dataset) is the latter (10). So the expected number of Principal Components is 10.

However, in many cases, the first couple of principal components can explain most of the variance in the data, therefore those are most important.

11 PCA on mtcars data

To run a PCA, we use a built-in function prcomp( ).

```
pr.out=prcomp(cars, scale=TRUE)
summary(pr.out)
```

```
## Importance of components%s:
##                          PC1   PC2     PC3     PC4     PC5     PC6     PC7
## Standard deviation     2.400 1.628 0.77280 0.51914 0.47143 0.45839 0.36458
## Proportion of Variance 0.576 0.265 0.05972 0.02695 0.02223 0.02101 0.01329
## Cumulative Proportion  0.576 0.841 0.90071 0.92766 0.94988 0.97089 0.98419
##                            PC8     PC9    PC10
## Standard deviation     0.28405 0.23163 0.15426
## Proportion of Variance 0.00807 0.00537 0.00238
## Cumulative Proportion  0.99226 0.99762 1.00000
```

As expected, we have 10 principal components. The first two principal components account for 84% of all the variance in the data.

The output contains a few useful components. **center** and **scale** correspond to the means and standard deviations of the variables used to scale them. **rotation** contains principal component loading vectors. **x** contains the principal component scores.

```
pr.out$center
```

```
##        cyl       disp         hp       drat         wt       qsec
##   6.187500 230.721875 146.687500   3.596563   3.217250  17.848750
##         vs         am       gear       carb
##   0.437500   0.406250   3.687500   2.812500
```

```
pr.out$scale
```

```
##         cyl        disp          hp        drat          wt        qsec
##   1.7859216 123.9386938  68.5628685   0.5346787   0.9784574   1.7869432
##          vs          am        gear        carb
##   0.5040161   0.4989909   0.7378041   1.6152000
```

```
pr.out$rotation
```

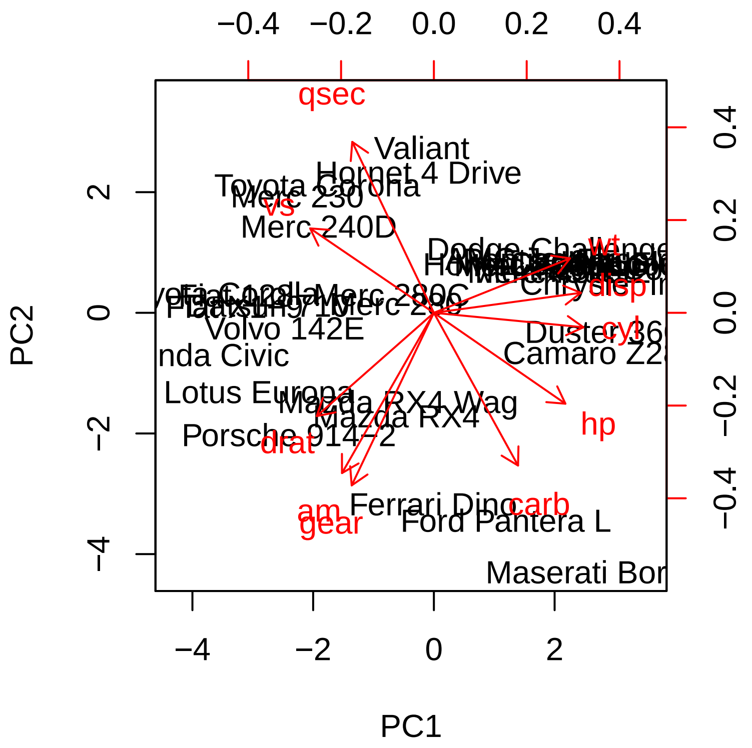
```
##             PC1         PC2         PC3           PC4         PC5
## cyl   0.4029711 -0.03901479  0.13874360 -8.040022e-05  0.06148048
## disp  0.3959243  0.05393117  0.01633491 -2.646304e-01  0.33851109
## hp    0.3543255 -0.24496137 -0.18225874  6.000387e-02  0.52828704
## drat -0.3155948 -0.27847781 -0.13057734 -8.528509e-01  0.10299748
## wt    0.3668004  0.14675805 -0.38579961 -2.527210e-01 -0.14410292
## qsec -0.2198982  0.46066271 -0.40307004 -7.174202e-02 -0.21341845
## vs   -0.3333571  0.22751987 -0.41252247  2.119502e-01  0.62369179
## am   -0.2474991 -0.43201042  0.23493804  3.190779e-02  0.04930286
## gear -0.2214375 -0.46516217 -0.27929375  2.623809e-01  0.02039816
## carb  0.2267080 -0.41169300 -0.56172255  1.233534e-01 -0.36576403
##              PC6         PC7         PC8          PC9          PC10
## cyl  -0.18206407 -0.04257067  0.07041306 -0.863268748  0.1670687388
## disp  0.35738419  0.19767431 -0.14361684 -0.020039738 -0.6838300858
## hp   -0.03269674 -0.08503414  0.58708325  0.291428365  0.2462606844
## drat -0.23386885  0.03226657  0.04010725 -0.086765162  0.0544414772
## wt    0.43201764 -0.03368560 -0.36605124  0.075971836  0.5318885631
## qsec  0.29265169 -0.03797611  0.59621869 -0.244573292 -0.1545795278
## vs   -0.11710663 -0.23387904 -0.36246041 -0.182200371 -0.0055443849
## am    0.60874338 -0.54631997  0.02588771 -0.154149509 -0.0003995261
## gear  0.24560902  0.69429321 -0.01069942 -0.198369367  0.0741152014
## carb -0.25782743 -0.33623769 -0.08067483  0.003086198 -0.3585136181
```

```
pr.out$x
```

```
##                             PC1        PC2         PC3         PC4
## Mazda RX4           -0.62217900 -1.7124758  0.65463530 -0.09927344
## Mazda RX4 Wag       -0.59549821 -1.5298641  0.42777458 -0.18761896
## Datsun 710          -2.75742785  0.1207011  0.40798148  0.25930035
## Hornet 4 Drive      -0.25512188  2.3270299  0.09359211  0.49796096
## Hornet Sportabout    2.00049846  0.7626191  1.00037428 -0.07979434
## Valiant             -0.20164079  2.7350268 -0.09185849  0.97987352
## Duster 360           2.80149673 -0.3131981  0.31900407  0.05230118
## Merc 240D           -1.90811489  1.4313363 -0.94060538  0.13666266
## Merc 230            -2.26404399  1.9326774 -1.72363955 -0.29482335
## Merc 280            -0.62695425  0.1484818 -1.41146053 -0.06511292
## Merc 280C           -0.70078923  0.3031580 -1.54679893 -0.08920166
## Merc 450SE           2.13434865  0.6873507  0.31363072  0.13036323
## Merc 450SL           1.98227907  0.6879131  0.40257778  0.21015059
## Merc 450SLC          1.95179957  0.7985300  0.29263750  0.18117717
## Cadillac Fleetwood   3.48428461  0.8279433 -0.63664339 -0.29506008
## Lincoln Continental  3.54122953  0.7353865 -0.71441970 -0.41085933
## Chrysler Imperial    3.43870788  0.4383336 -0.69172473 -0.68543121
## Fiat 128            -3.27713336  0.2883295  0.27305357 -0.07216806
## Honda Civic         -3.82281909 -0.6931984  0.19946485 -1.16821813
## Toyota Corolla      -3.57895964  0.2717839  0.28744456 -0.20311549
## Toyota Corona       -1.93135981  2.0686961 -0.02966552 -0.03700092
## Dodge Challenger     2.01577827  1.0101176  1.15883107  0.59544106
## AMC Javelin          1.65607750  0.8990033  0.99826393  0.00794614
## Camaro Z28           2.61675255 -0.6587364  0.18122676 -0.80825776
## Pontiac Firebird     2.31772986  0.8849626  0.85628550 -0.15935609
## Fiat X1-9           -3.30537397  0.1017705  0.50615241  0.01852124
## Porsche 914-2       -2.40162292 -2.0250366  0.86722440 -0.55920857
## Lotus Europa        -2.89992596 -1.3619253  0.35024252  1.14103656
## Ford Pantera L       1.19346975 -3.4431821  0.19435700 -0.59057152
## Ferrari Dino        -0.01395827 -3.1706273 -0.36844420  0.94442403
## Maserati Bora        2.50111614 -4.2982141 -1.40626795  0.86112527
## Volvo 142E          -2.47264544 -0.2546928 -0.22322599 -0.21121215
##                             PC5         PC6         PC7         PC8
## Mazda RX4           -0.91301160 -0.05071642 -0.39462481 -0.17516204
## Mazda RX4 Wag       -1.01744888  0.15358601 -0.41530488 -0.08371463
## Datsun 710           0.43969395  0.49938548 -0.28670292 -0.08279167
## Hornet 4 Drive       0.54350235  0.03498121 -0.05731631 -0.18522291
## Hornet Sportabout    0.19725533 -0.16066326  0.28084619  0.11484446
## Valiant              0.22396184  0.31809354 -0.14807049 -0.04520855
## Duster 360           0.41705536 -0.67539322 -0.19808954  0.17649488
## Merc 240D           -0.34368346 -0.07205390  0.63099942 -0.36860790
## Merc 230            -0.40168440  0.25187200  0.53428775  0.92060834
## Merc 280             0.00988357 -0.83264980  0.16606523 -0.53502739
## Merc 280C           -0.06177571 -0.73438646  0.15331403 -0.33483569
## Merc 450SE          -0.37422241 -0.19007986 -0.10241229  0.09037688
## Merc 450SL          -0.34803513 -0.30744537 -0.09495743  0.28430503
## Merc 450SLC         -0.40317177 -0.21986001 -0.10517959  0.39906064
## Cadillac Fleetwood  -0.14219278  0.88135458 -0.09006162 -0.23128765
## Lincoln Continental -0.09094906  0.86198753 -0.11996891 -0.24498423
## Chrysler Imperial    0.07371611  0.59617123 -0.14537068 -0.18002195
## Fiat 128             0.11089583  0.41503168 -0.30021369  0.06905493
## Honda Civic          0.13173464 -0.53223713 -0.40418134 -0.13164928
## Toyota Corolla       0.11180156  0.24162130 -0.29921871  0.35982163
## Toyota Corona        0.15965566 -0.66155881 -0.16233903  0.30167573
## Dodge Challenger    -0.17908168 -0.08850767  0.22176283 -0.15978596
## AMC Javelin         -0.18102958 -0.26657165  0.21675724  0.06096161
## Camaro Z28           0.50150383 -0.88288629 -0.18281516 -0.01739179
## Pontiac Firebird     0.22979248  0.16902958  0.32583871 -0.07826248
## Fiat X1-9            0.21881957  0.20554130 -0.27849834 -0.02233553
## Porsche 914-2       -0.61201693  0.29544757  1.01407155  0.01409535
## Lotus Europa         0.66742893  0.02454347  0.46006919 -0.23567860
## Ford Pantera L       1.10648101  0.33748539  0.65439473  0.15080322
## Ferrari Dino        -0.83977562 -0.07984664  0.02386098 -0.11302784
## Maserati Bora        0.40933284  0.01136722 -0.40302504  0.44961748
## Volvo 142E           0.35556417  0.45735740 -0.49391711 -0.16672409
##                             PC9        PC10
## Mazda RX4           -0.06643215 -0.16059118
## Mazda RX4 Wag       -0.12327824 -0.07041622
## Datsun 710           0.15979295  0.17889329
## Hornet 4 Drive      -0.10031905 -0.16390788
## Hornet Sportabout   -0.10653960 -0.17831510
## Valiant             -0.15204103  0.03333561
## Duster 360           0.35667892 -0.19196430
## Merc 240D            0.23587221 -0.03074150
## Merc 230            -0.06024938 -0.12885023
## Merc 280            -0.25639079  0.12258738
## Merc 280C           -0.33851090  0.07068438
## Merc 450SE          -0.05987323  0.38370350
## Merc 450SL          -0.11364573  0.18157881
## Merc 450SLC         -0.16451024  0.17415676
## Cadillac Fleetwood   0.05153324 -0.25397692
## Lincoln Continental  0.12002837 -0.03629507
## Chrysler Imperial    0.19830960  0.14300739
## Fiat 128            -0.11457991  0.12737102
## Honda Civic         -0.22502404 -0.27760133
## Toyota Corolla      -0.22751312 -0.05564372
## Toyota Corona        0.59661181 -0.03071268
## Dodge Challenger    -0.11598282 -0.01962089
## AMC Javelin         -0.24245905  0.01393101
## Camaro Z28           0.35372942  0.10012634
## Pontiac Firebird    -0.07430795 -0.18857955
## Fiat X1-9           -0.05719010  0.03096987
## Porsche 914-2        0.39716636  0.11976937
## Lotus Europa         0.16430071 -0.09851131
## Ford Pantera L      -0.41931173  0.12735940
## Ferrari Dino         0.13571032 -0.05176130
## Maserati Bora        0.02592148 -0.09005060
## Volvo 142E           0.22250369  0.22006566
```

11.0.1 Plot the first two pricipal components

```
biplot(pr.out, scale=0)
```



This graph shows the first two principal components which explain ~ 84% of the variance in the mtcars dataset. The red arrows represent the first two principal component loading vectors (top and right axes). The black car names represent the scores for the first two principal components (left and bottom axes).

12 How many principal components should we keep?

PCA creates more principal components than we need. We need a way to quantitatively determine how many principal components to use in later analysis.

One way to decide on the number of principal components to keep is plot the Proportion of Variance Explained (PVE) by each principal component.

The PVE is obtained by squaring the standard deviation (variance) and dividing it by the total variance.

```
pr.out$sdev
```

```
##  [1] 2.4000453 1.6277725 0.7727968 0.5191403 0.4714341 0.4583857 0.3645821
##  [8] 0.2840450 0.2316298 0.1542606
```

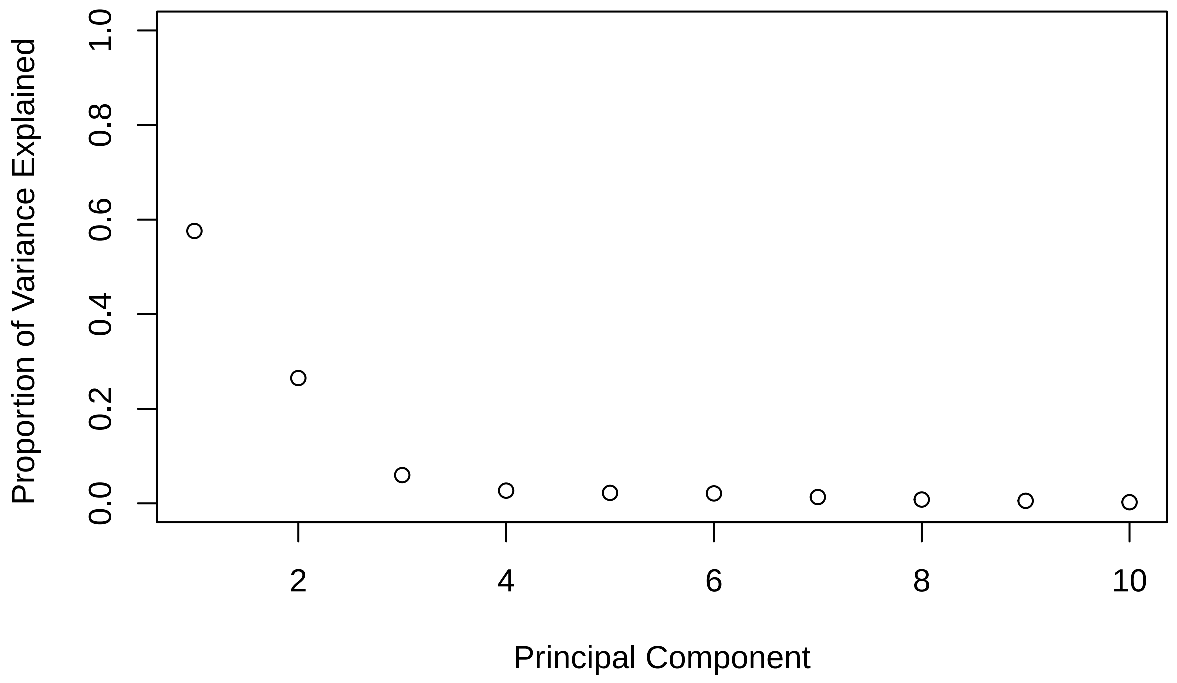
```
pr.var <- pr.out$sdev^2
pve <- pr.var / sum(pr.var)

pve
```

```
##  [1] 0.576021744 0.264964319 0.059721486 0.026950667 0.022225006
##  [6] 0.021011744 0.013292009 0.008068158 0.005365235 0.002379633
```

Check these numbers are the same as the second row of summary(pr.out).

12.0.1 Plot PVE

```
plot(pve, xlab="Principal Component", ylab="Proportion of Variance Explained ", ylim=c(0,1))
```



We can see that after the 3rd principal component, the PVE is small. So in this case, we can use the first 3 principal components in another machine learning problem like clustering and regression.

13 Applying PCA result to linear regression

Put the first 3 principal components into a data frame.

```
pc <- as.data.frame(pr.out$x[,1:3])
```

Multiple linear regression to find a correlation between mpg and first 3 PCs.

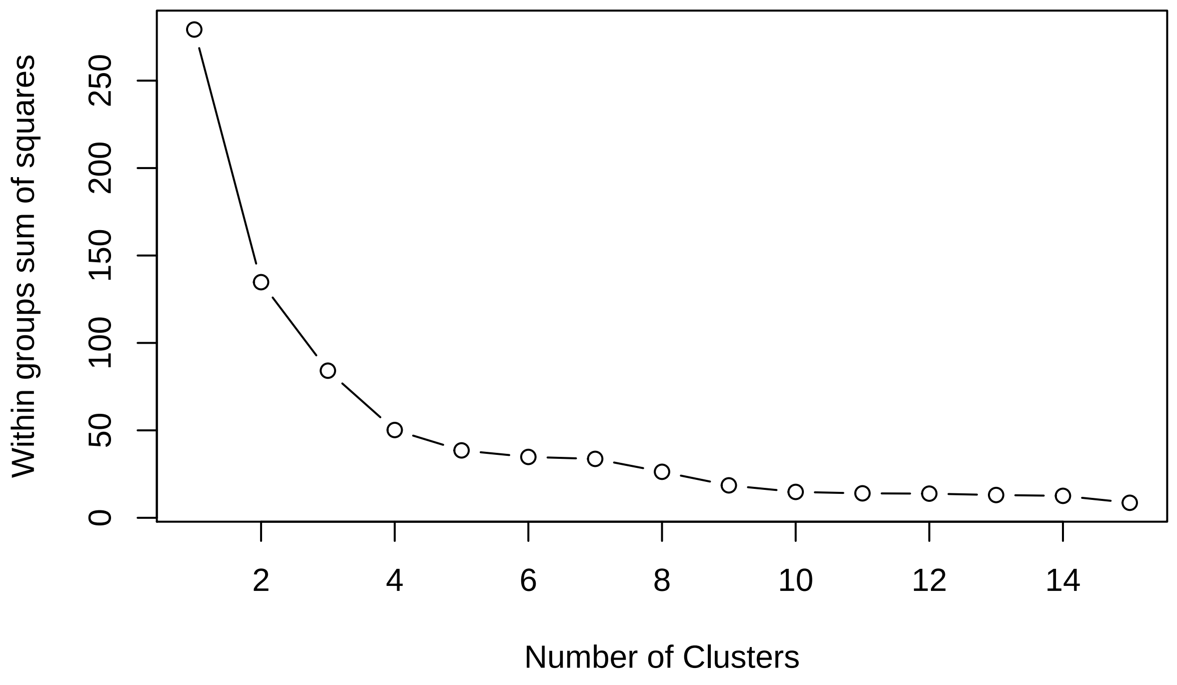
```
lm.fit=lm(mtcars$mpg ~ pc$PC1 + pc$PC2 + pc$PC3)
summary(lm.fit)
```

```
##
## Call:
## lm(formula = mtcars$mpg ~ pc$PC1 + pc$PC2 + pc$PC3)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -4.097 -1.425 -0.534  1.303  5.303
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  20.0906     0.4283  46.909  < 2e-16 ***
## pc$PC1       -2.2813     0.1813 -12.583 4.83e-13 ***
## pc$PC2       -0.1163     0.2673  -0.435   0.6668
## pc$PC3        1.2993     0.5631   2.307   0.0286 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.423 on 28 degrees of freedom
## Multiple R-squared:  0.854,  Adjusted R-squared:  0.8384
## F-statistic: 54.61 on 3 and 28 DF,  p-value: 8.019e-12
```

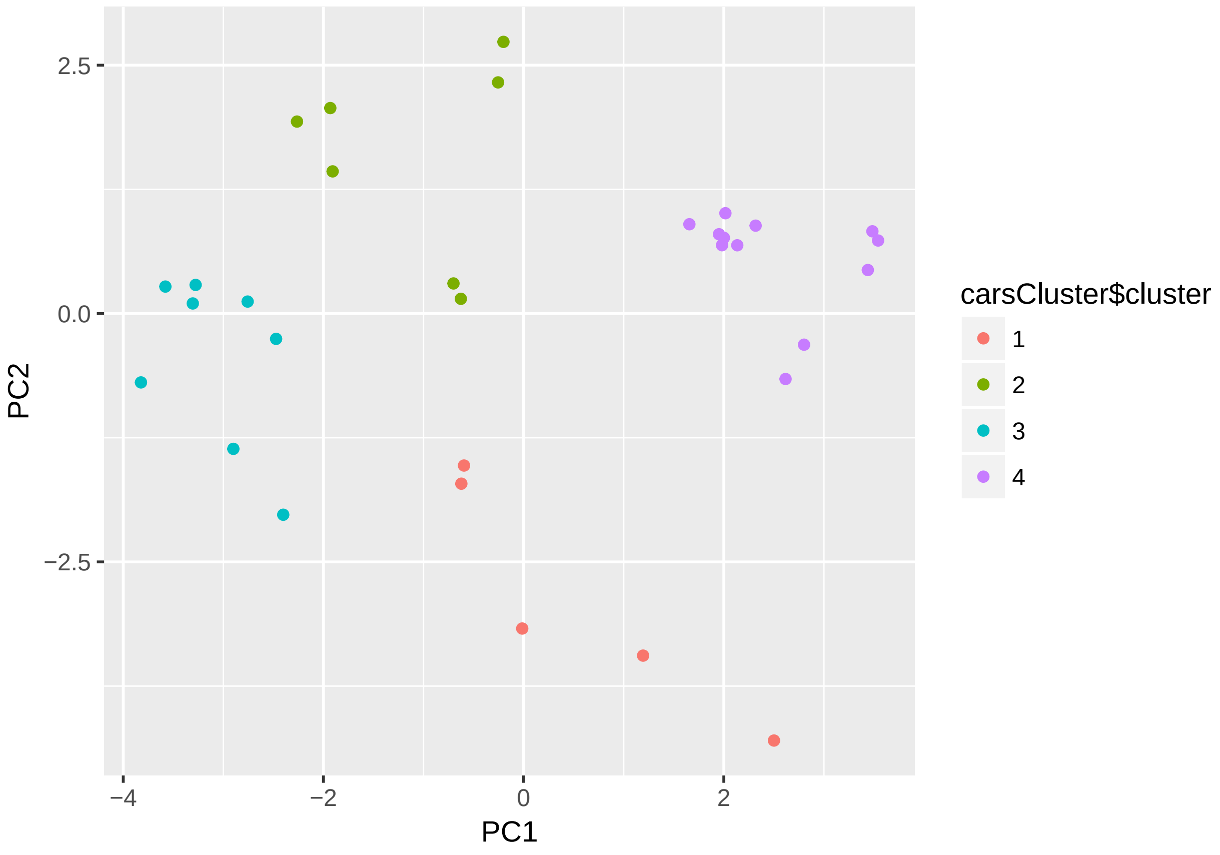
14 Applying PCA results to K-means clustering

Find the optimal number of clusters and cluster data using the first and second principal components.

```
wssplot(pc)
```

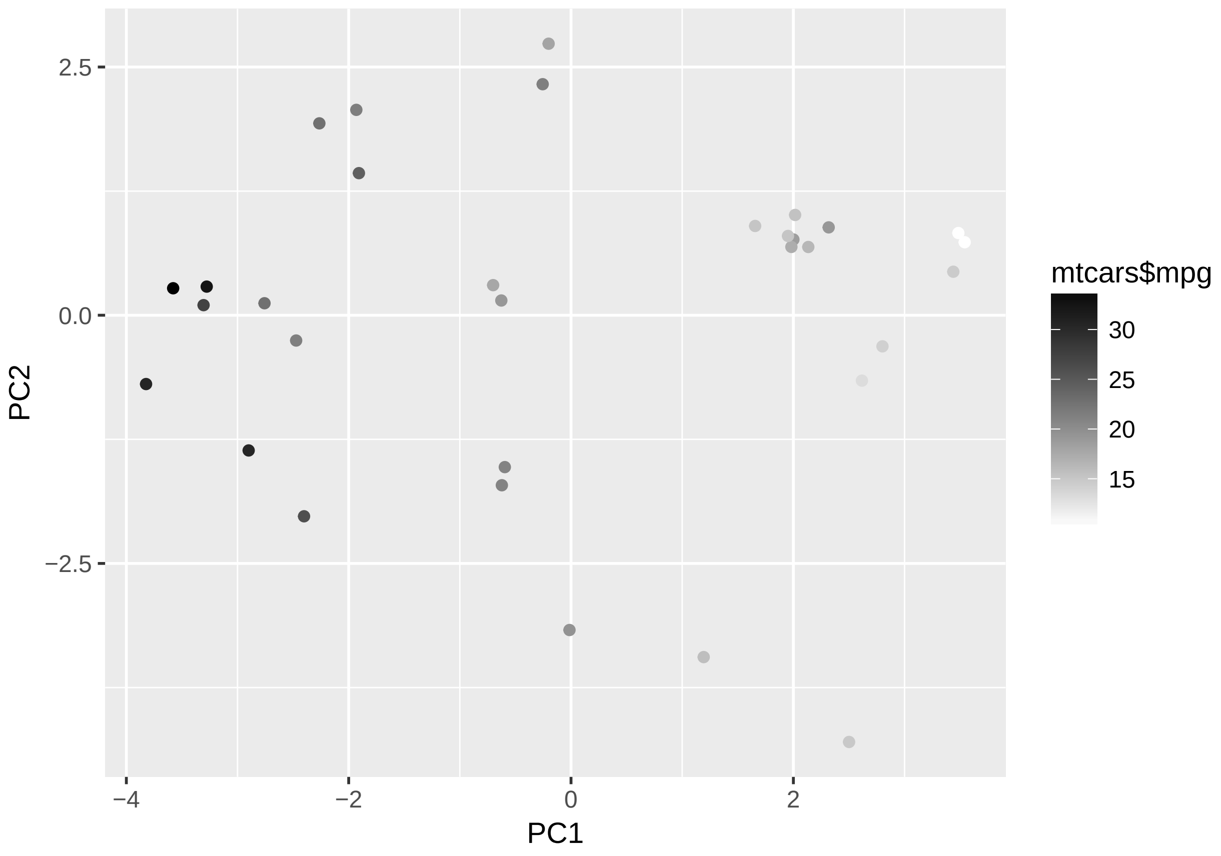


```r
# Looks like K = 4 wil be good

set.seed(20)
carsCluster <- kmeans(pc, 4, nstart = 20)

carsCluster$cluster <- as.factor(carsCluster$cluster)
ggplot(pc) +
  geom_point(mapping = aes(PC1, PC2, color = carsCluster$cluster))
```



Compare the result with the mpg variable in the mtcars data.

```r
ggplot(pc) +
  geom_point(mapping = aes(PC1, PC2, color = mtcars$mpg)) +
  scale_colour_gradient(low = "white", high = "black")
```



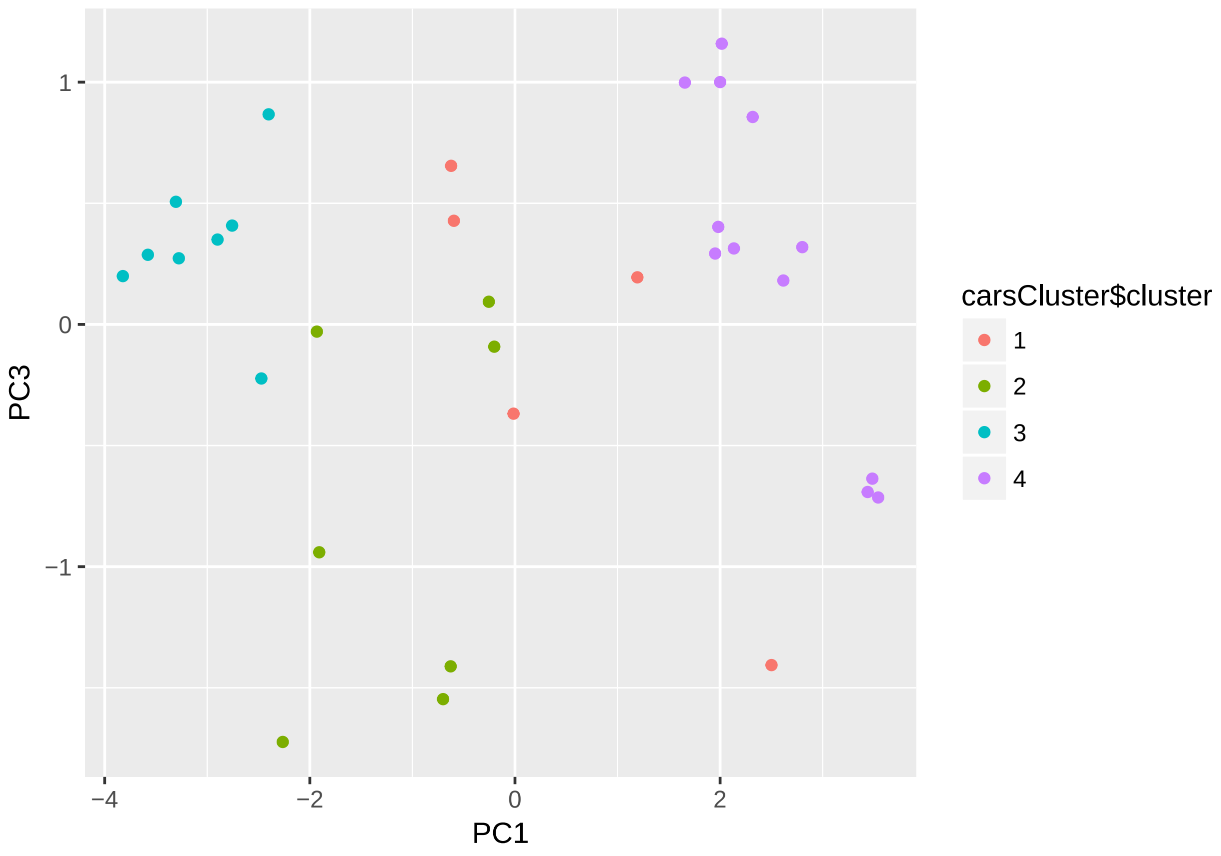
Clustering picks up the pattern in the distribution of fuel efficiency.

14.0.1 Exercise

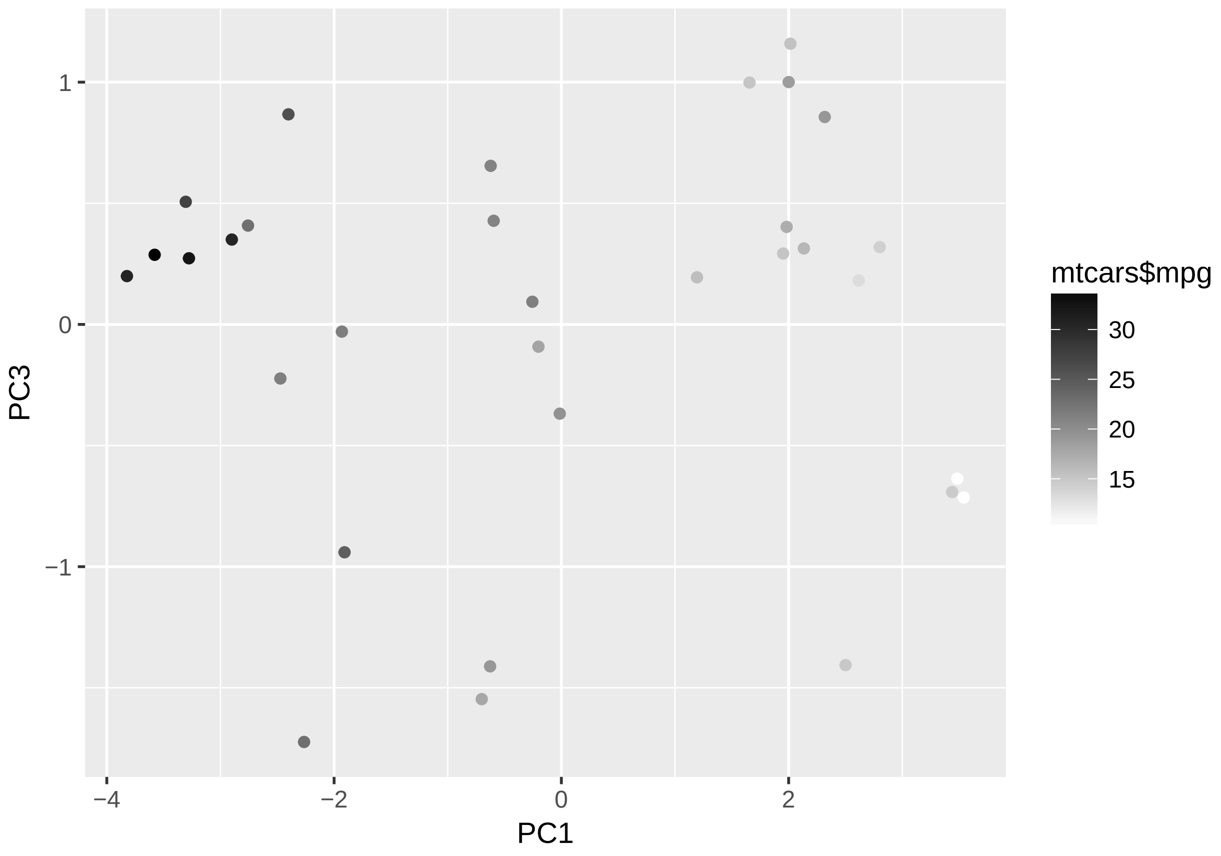
Plot the clustering result using the other combinations of principal components (PC1 & PC3, PC2 & PC3). Compare your plots with above plots and see if there is any similarity or difference, and why.

PC1 & PC3

```r
ggplot(pc) +
  geom_point(mapping = aes(PC1, PC3, color = carsCluster$cluster))
```



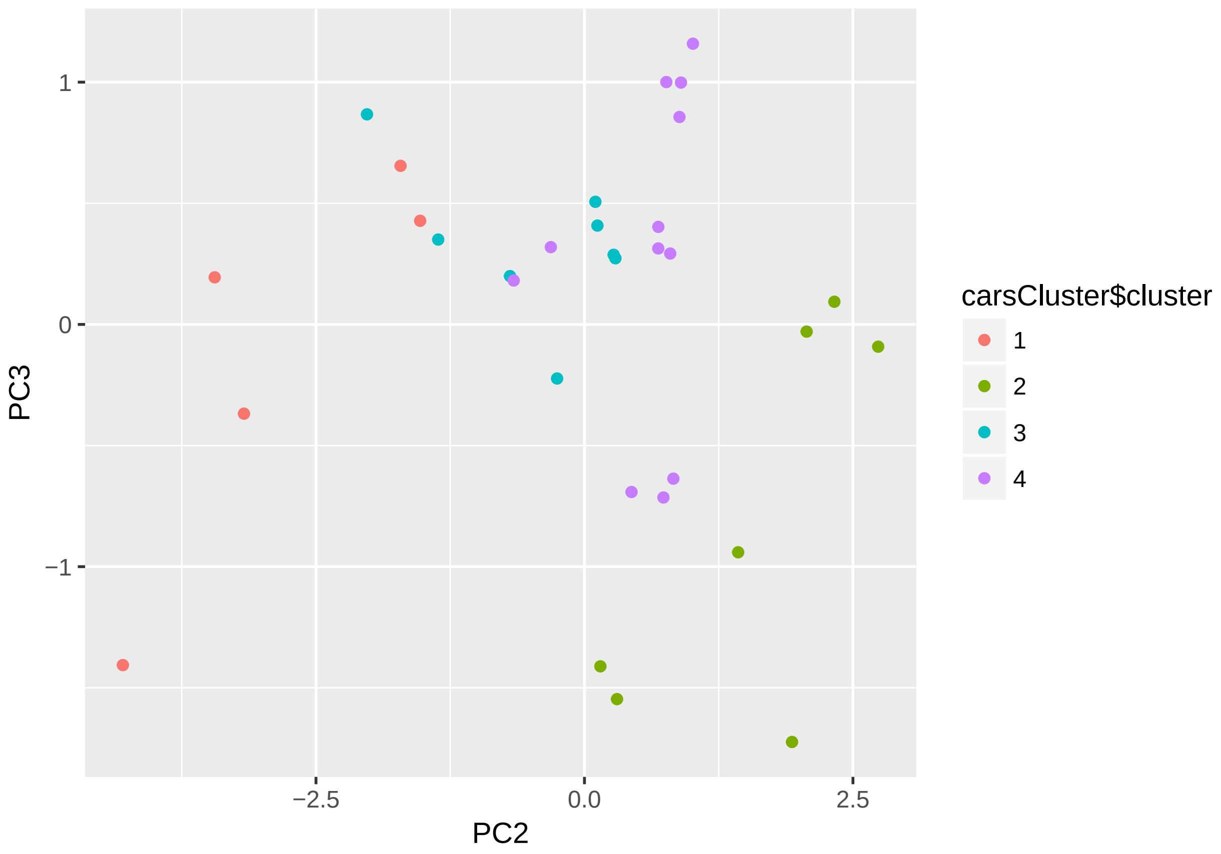
```
ggplot(pc) +
  geom_point(mapping = aes(PC1, PC3, color = mtcars$mpg)) +
  scale_colour_gradient(low = "white", high = "black")
```



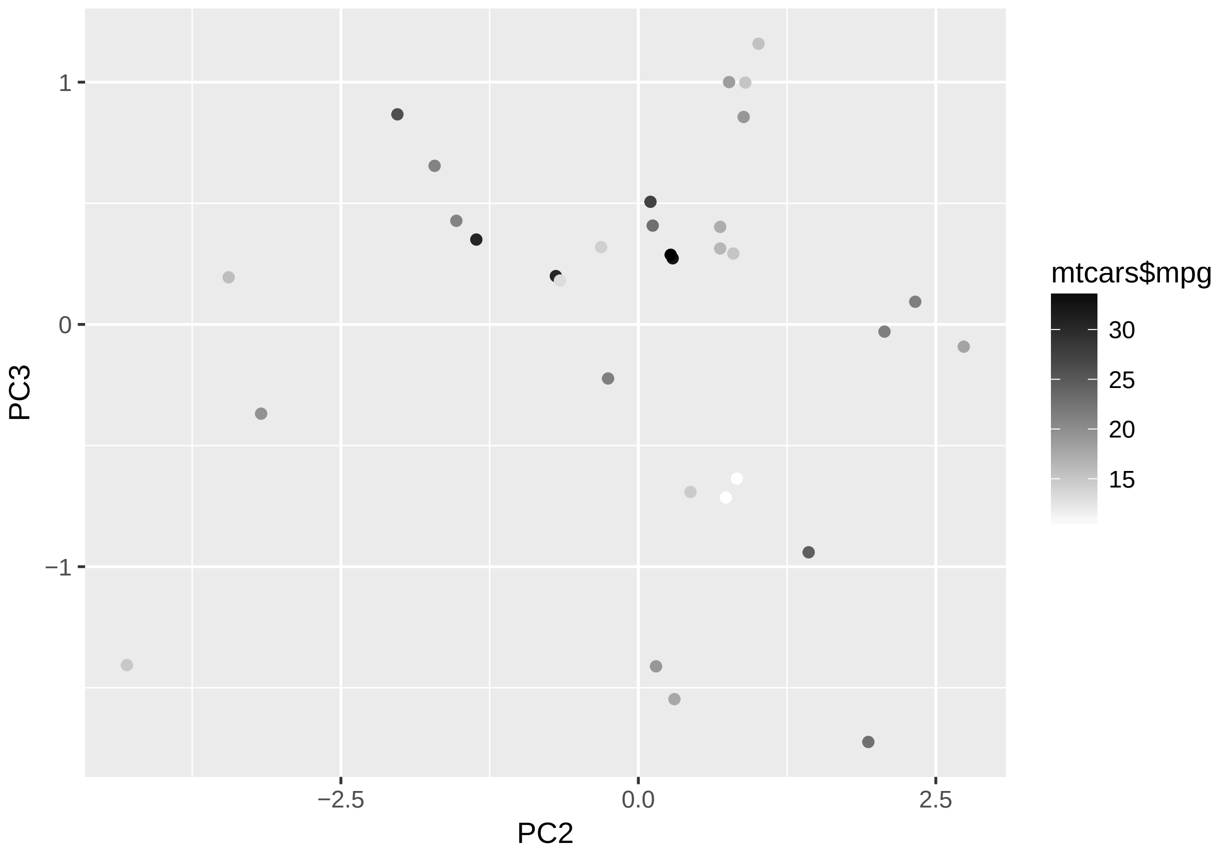
This is similar to what we had with PC1 and PC2.

PC2 & PC3

```
ggplot(pc) +
  geom_point(mapping = aes(PC2, PC3, color = carsCluster$cluster))
```



```r
ggplot(pc) +
  geom_point(mapping = aes(PC2, PC3, color = mtcars$mpg)) +
  scale_colour_gradient(low = "white", high = "black")
```



In this case, the clusters are less clearly separated because PC1 is not included.

End.