Revised Notes on Kmeans vs. kNN:

For the purposes of illustration, we will use the auto-mpg.data set from the UCI ML Repository:

auto-mpg.names is the description of the data set

1. Title: Auto-Mpg Data

2. Sources:

(a) Origin: This dataset was taken from the StatLib library which is

maintained at Carnegie Mellon University. The dataset was

used in the 1983 American Statistical Association Exposition.

(c) Date: July 7, 1993

3. Past Usage:

- See 2b (above)

- Quinlan,R. (1993). Combining Instance-Based and Model-Based Learning.

In Proceedings on the Tenth International Conference of Machine

Learning, 236-243, University of Massachusetts, Amherst. Morgan

Kaufmann.

4. Relevant Information:

This dataset is a slightly modified version of the dataset provided in

the StatLib library. In line with the use by Ross Quinlan (1993) in

predicting the attribute "mpg", 8 of the original instances were removed

because they had unknown values for the "mpg" attribute. The original

dataset is available in the file "auto-mpg.data-original".

"The data concerns city-cycle fuel consumption in miles per gallon,

to be predicted in terms of 3 multivalued discrete and 5 continuous

attributes." (Quinlan, 1993)

5. Number of Instances: 398

6. Number of Attributes: 9 including the class attribute

7. Attribute Information:

1. mpg: continuous

2. cylinders: multi-valued discrete

3. displacement: continuous

4. horsepower: continuous

5. weight: continuous

6. acceleration: continuous

7. model year: multi-valued discrete

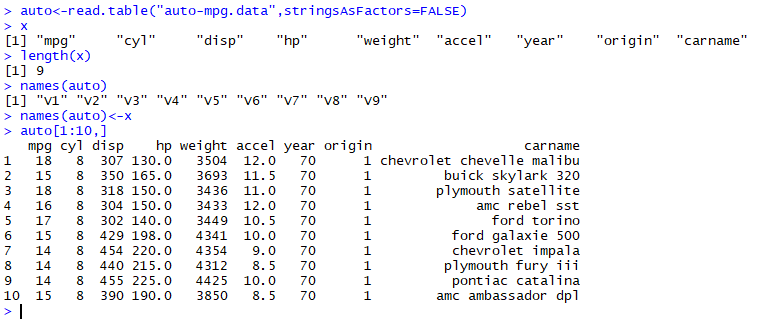
8. origin: multi-valued discrete

9. car name: string (unique for each instance)

8. Missing Attribute Values: horsepower has 6 missing valueslength(x)

…………….

Processing auto-mpg.data:

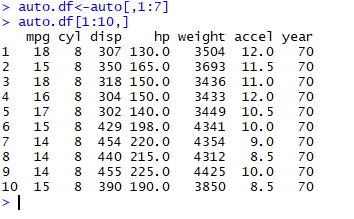


By using stringsAsFactors = FALSE, we ensure that no factors get inserted into the table. Factors cause problems with other functions.

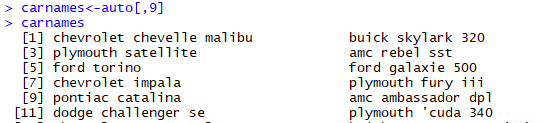
Note that read.table automatically assigns arbitrary variable names V1 through number of variables.

To assign variables names, we assign the names to variable x, check the length, then assign to names(auto).

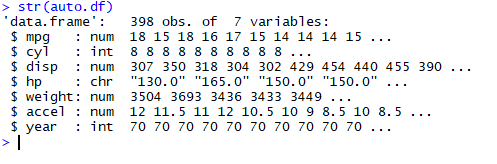
Strip off the last two columns because they do not contribute to analysis:



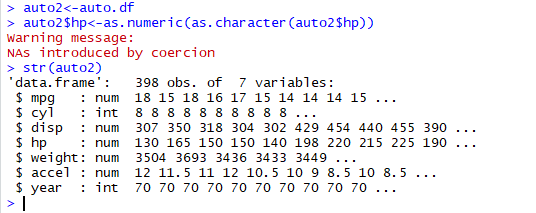
And, grab the car names for good measure:



Get the structure of auto.df:

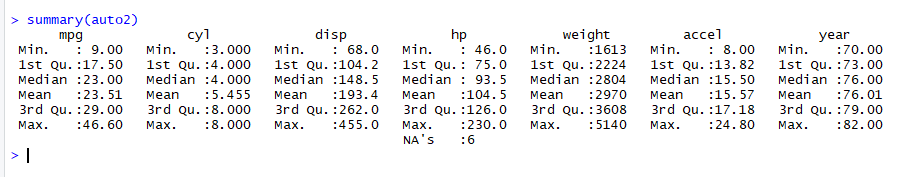


But, there are character values we need to convert to numeric. So copy auto.df to auto2. Use as.numeric to convert characters



Now, all values are numbers.

Get a summary of the 7 variables for auto:



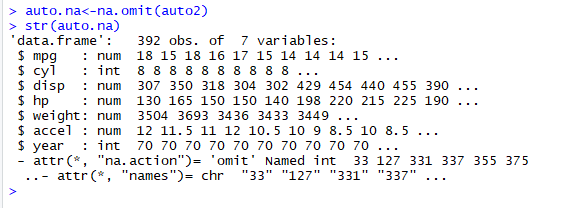
Note that six HP values are NAs. The NA rows do not participate in the computation.

We don’t have enough information replace those values so:

a. we can delete those rows. This might not be a good idea given the number of rows that have NAs, because the other columns may have useful information.

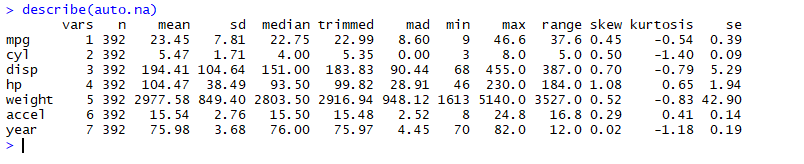
b. we can replace the NA values by the mean of the column, which probably skews the results for that column a bit. Again, if there are lots of rows with NAs, then this approach pushes the analysis towards the mean of the columns in which the NAs occur.

Use na.omit(<dataframe>) to remove rows that have NAs in them.



And, it tells us which rows were deleted. If we have other information, we could go back and fill in the values for the HP variable in those rows.

And, describe the statistics of the 7 variables:

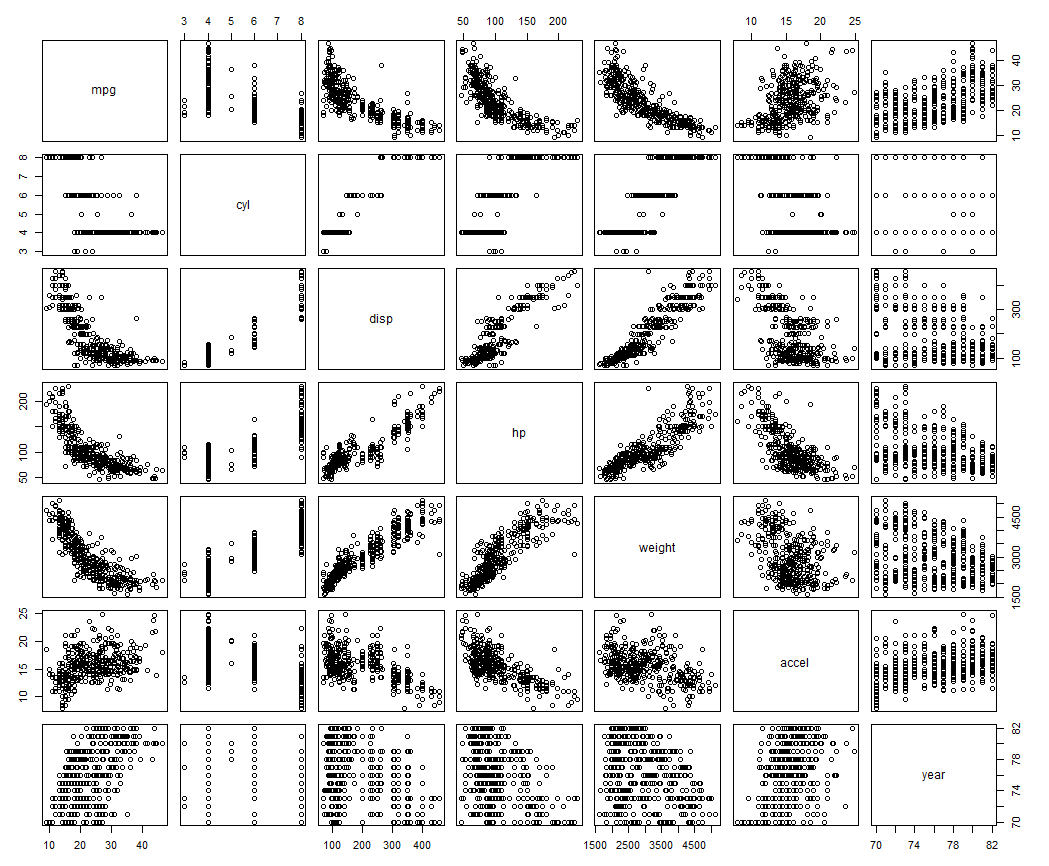


Look up skew and kurtosis on Wikipedia.

Note that the “skew” computer by describe is for each variable, whereas the “skew and “kurtosos” functions in package “e1071” are multivariate.

Removed six rows that had “?” for the value of $hp.

Plotting the 7 variables in a pairwise manner:



Look for linear or near linear relationships.

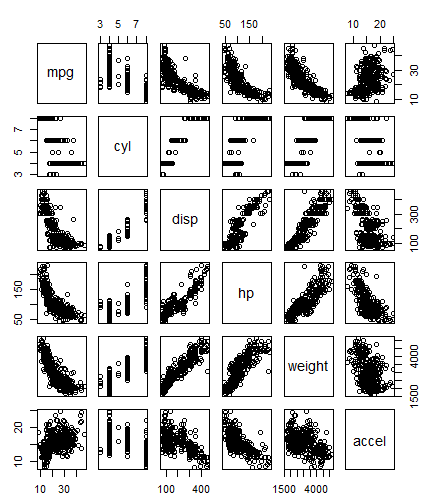
So, mpg vs. disp, mpg vs. hp, and mpg vs. weight all have distinct relationships, e.g., as one variable changes, so does the other in a distinct way.

Mpg vs. accel seems to be fuzzy. The relationship may be dependent on the efficiency of the car for which we have no data

Cyl vs. <other variables> doesn’t help a lot. As the number of cylinders goes up, the displacement goes up to accommodate them. Year vs. anything does not seem to have any relationship.

So, considering removing year from further consideration.





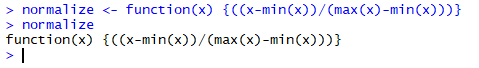
Keep these plots in mind because when we get to linear modeling we are going to look at these patterns.

Scaling: we normalize so that larger values for certain variables don’t overwhelm the smaller values of other variables.

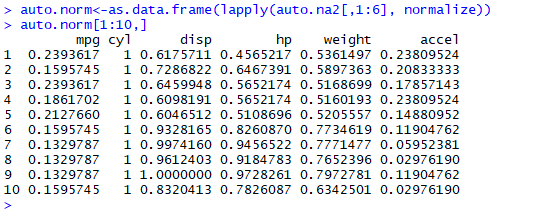
Consider two way to normalize:

Min-Max Normalization:

* A normalization strategy which linearly transforms x to y= (x-min)/(max-min)
* Where min and max are the minimum and maximum values in X, where X is the set of observed values of x.
* It can be easily seen that when x=min, then y=0, and. When x=max, then y=1.

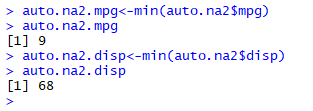


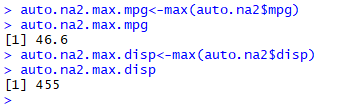
Scaling with normalize: The lapply() method applies to the entire data frame and all selected columns.

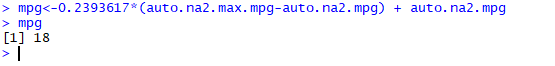


To get back to the original values, later:

x = norm(x) \* (max(x)-min(x) ) + min(x).



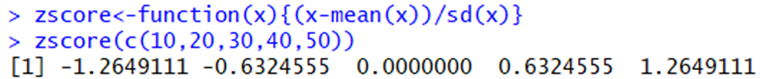


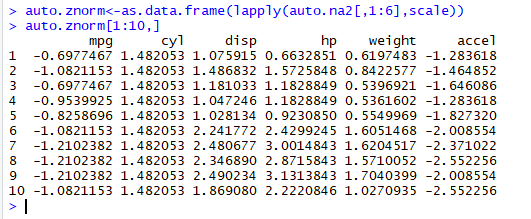


Voila!

Z-score normalization:

* All elements of the input vector are transformed into the output vector whose mean is approximately 0 while the standard deviation is in a range close to 1
* Scale() handles this for you





To get back to x, we use x = sd(x) + mean(x).

Try this with your data set.

Kmeans Clustering: We discussed this in class.

Here is the method invocation:

object = kmeans(x, centers, iter.max = 10, nstart = 1,

algorithm = c("Hartigan-Wong", "Lloyd", "Forgy", "MacQueen"), trace=FALSE)

fitted(object, method = c("centers", "classes"), ...)

where:

"centers" causes fitted to return cluster centers (one for each input point) and "classes" causes fitted to return a vector of class assignments.

Some things to think about:

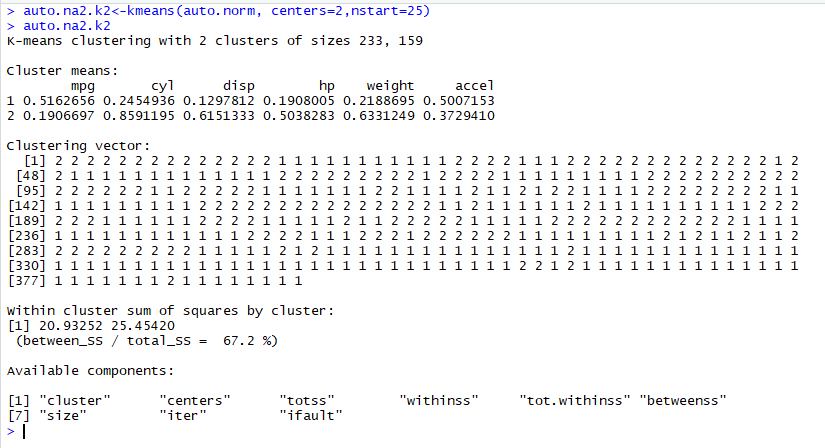
* Where does the distance d come from? For kmeans below, it is defined within the program.
* What if we used a different distance measure?
* How do we choose the best distance
* How do we choose k?
* Will it converge?

The default iter.max is 10, but can be overridden by the user.

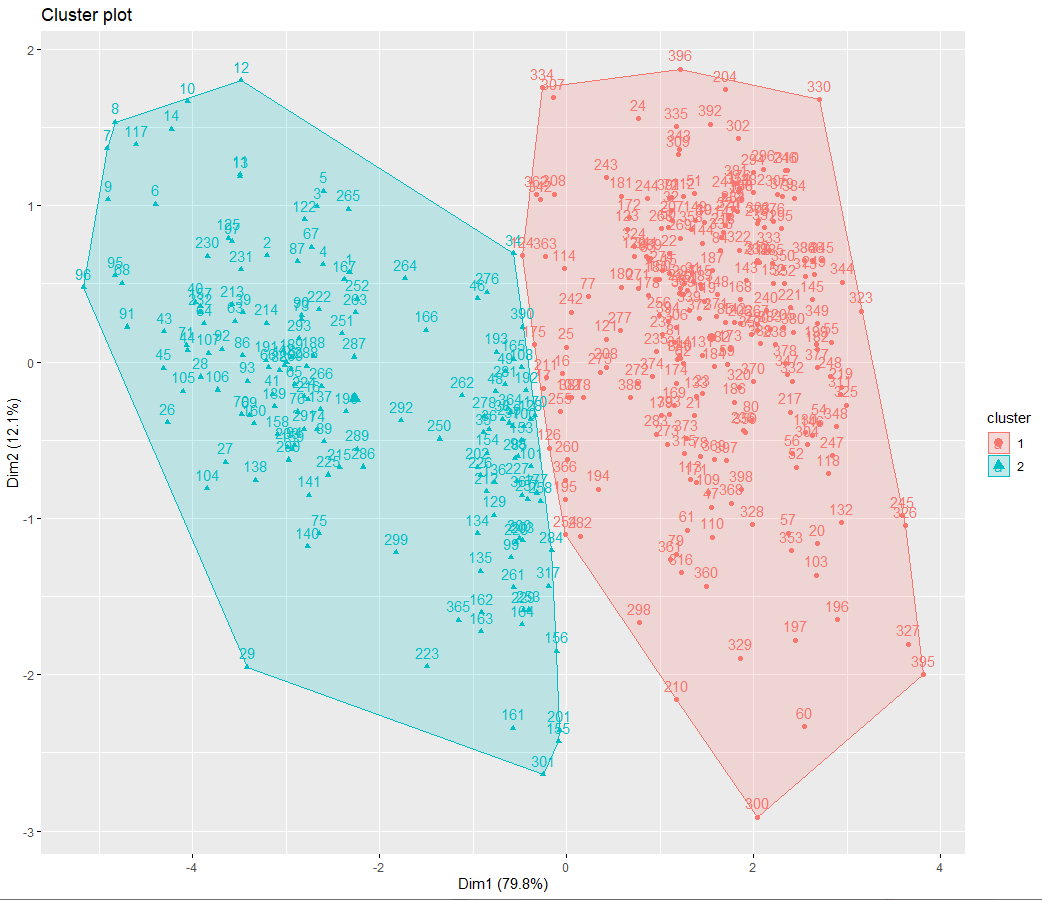
nstart specifies the number of random centers to be initially selected.

Usually, center is a number indicating the number of clusters we expect and nstart is equal to centers.

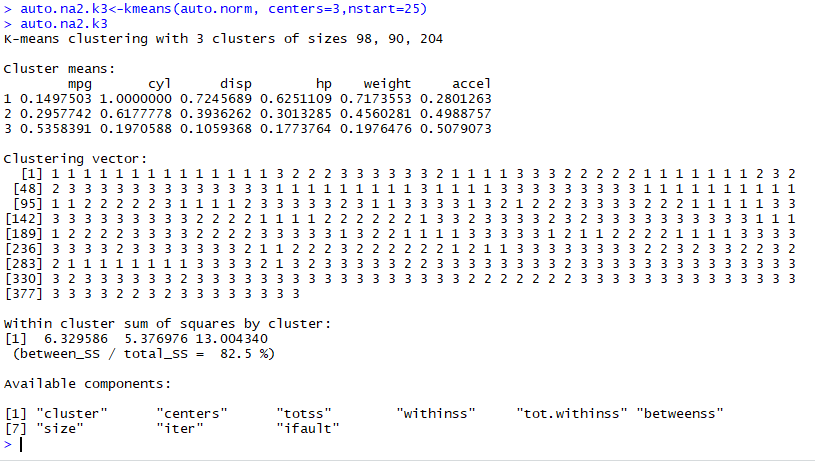
With k =2:



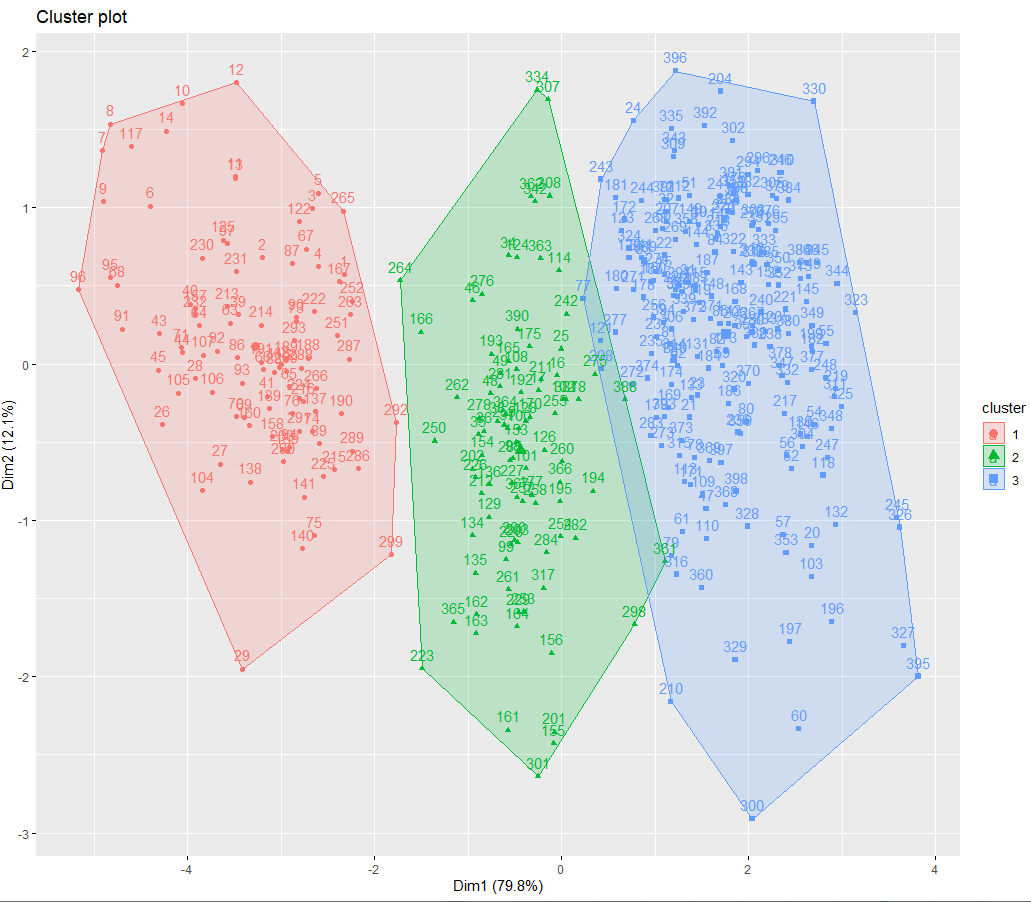
Using factoextra: factoextra::fviz\_cluster(auto.na2.k2,auto.na2



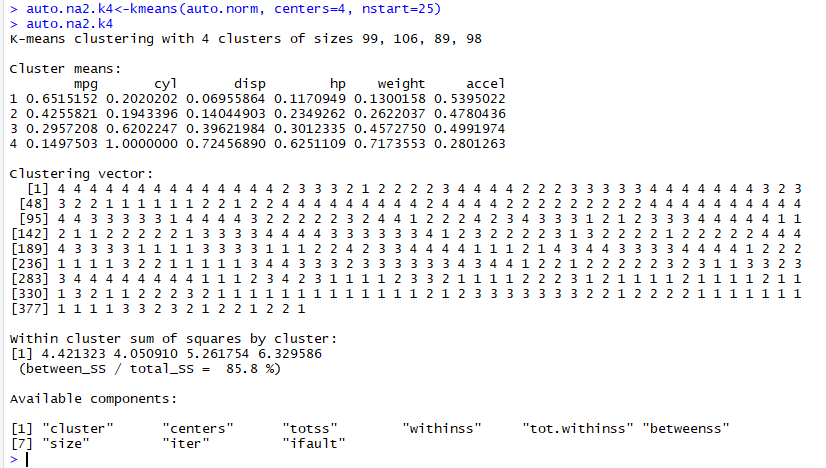
With k = 3:



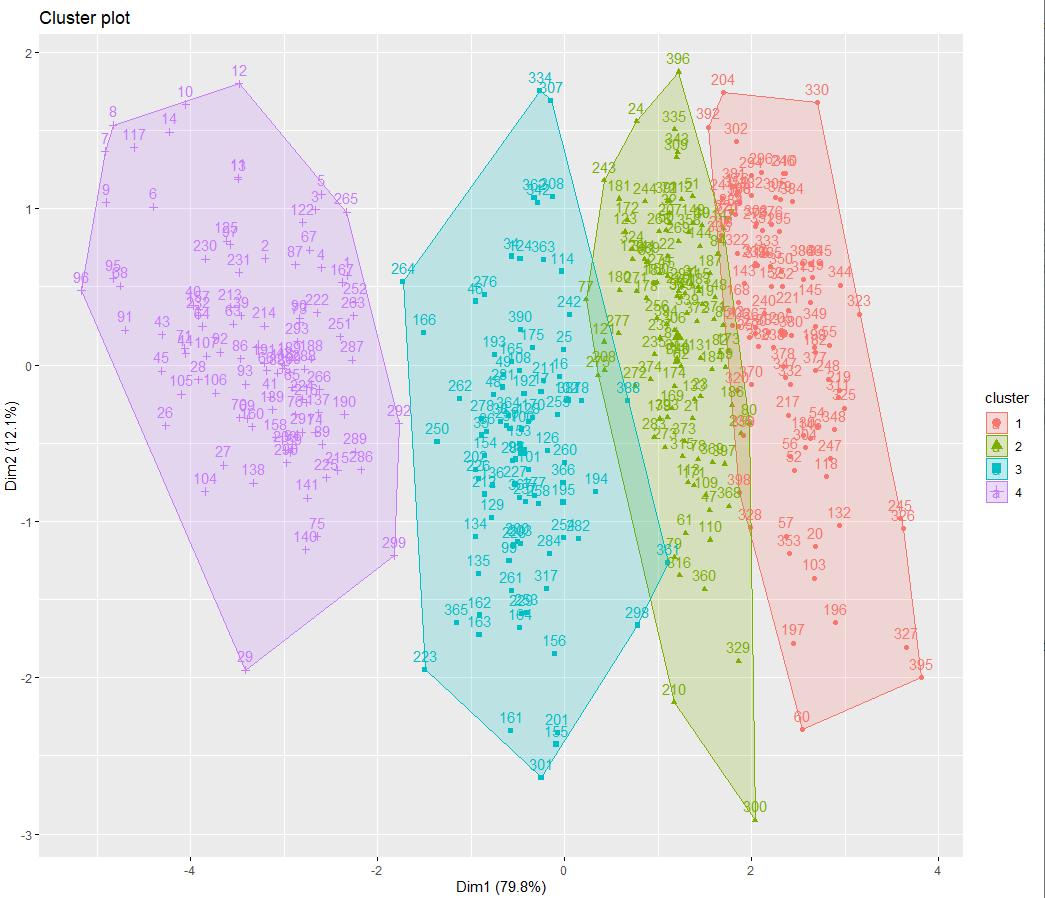
Using factoextra::fviz\_cluster(auto.na2.k3,auto.na2)



With k = 4:



Using factoextra::fviz\_cluster(auto.na2.k4,auto.na2)



In kmeans, the algorithm stops when:

Maximum number of iterations reached

The change to “within cluster sum-of-squares” in two successive iterations is less than the threshold value.

If the assignments of data points to clusters does not change during an iteration, then the algotrithm has converged to a at least a local minimum.

The within-cluster sum of squares is a measure of the variability of the observations within each cluster.

In general, a cluster that has a small sum of squares is more compact than a cluster that has a large sum of squares.

Clusters that have higher values exhibit greater variability of the observations within the cluster.

But, as the number of observations increases, the sum of squares becomes larger.

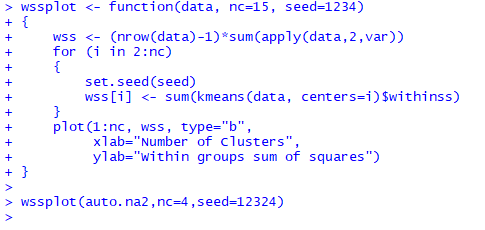
Whenever an assignment is changed, e.g., a data point moves from one cluster to another, the sum squared distances of the data points in the cluster is reduced.

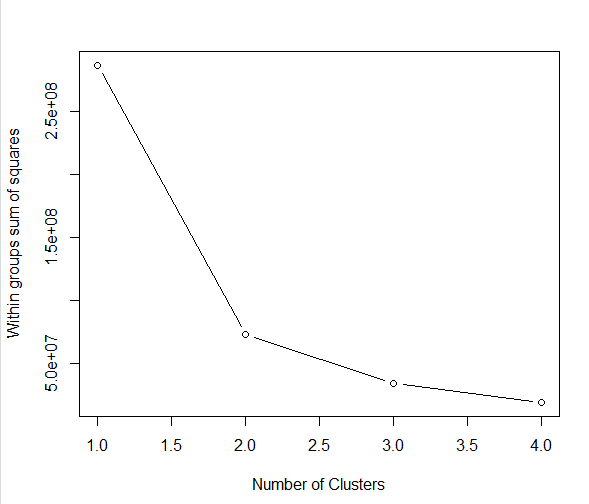
The ratio (between\_ss/total\_ss) is a measure of the variance in the data set.

k-means minimizes the within group dispersion and maximizes the between-group dispersion.

We want to keep vetween\_ss/total\_ss low which means the resulting cluster is compact.

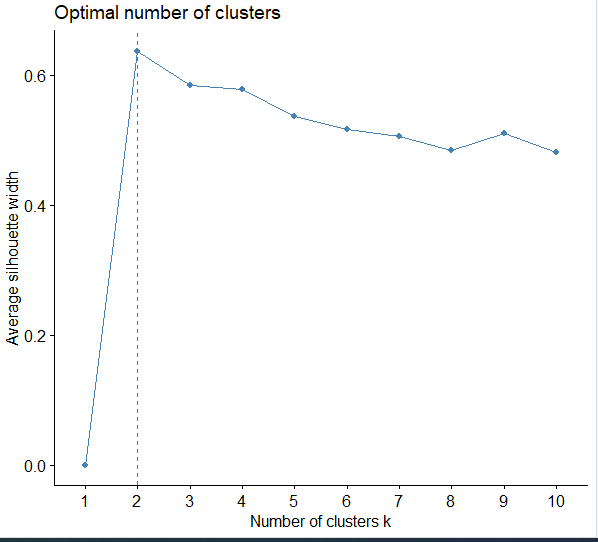
Finding a Good Number of Clusters:





Use the fviz\_nbclust() method in factoextra to get an estimate of the optimal number of clusters.

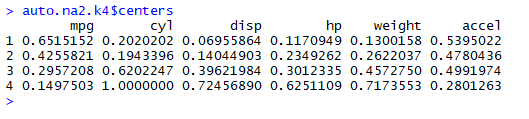
factoextra::fviz\_nbclust(auto.na2, FUNcluster=kmeans,print.summary=TRUE)



Examine the size of the clusters. Clusters that are too large or too small, they are not likely to be very useful because they lack descriptive power. So, for k =4, we see the cluster sizes are 99, 106, 89, and 88. Compared to k=2 or k=3, this is a homogenous distribution.

Now, examine the cluster centroids.

Look for the largest number in each column. Indicates that is most important cluster for that attribute.



So, mpg-1, cyl-4, disp-4, hp-4, weigh-4, accel-1.

If you are a car afiocinado, then cyl, disp, hp, and weight are all related.

You then need to go back and investigate the cars in each cluster to draw some conclusions. To do this, find the cars in each cluster and examine their characteristics.

Try this w/ scale (aka zscore). Then, you should get both positive and negative numbers. Positive numbers are above the mean while negative numbers are below the mean. Can you see any patterns when you highlight the maximum values in each column?

K-NN:

Compute k-nearest neighbour classifications for test set data points based on a model constructed from the data points of the training set.

For each row of the test set, the k nearest (in Euclidean distance) training set vectors are found, and the classification is decided by majority vote, with ties broken at random. If there are ties for the kth nearest vector, all candidates are included in the vote.

There are several knn algorithms in the CRAN Repository. This description is from class::knn.

knn(train, test, k = 1, prob = FALSE, use.all = TRUE)

where:

train matrix or data frame of training set cases.

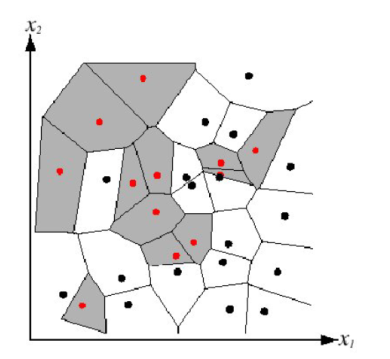
test matrix or data frame of test set cases. A vector will be interpreted as a row vector for a single case.

k number of neighbours considered.

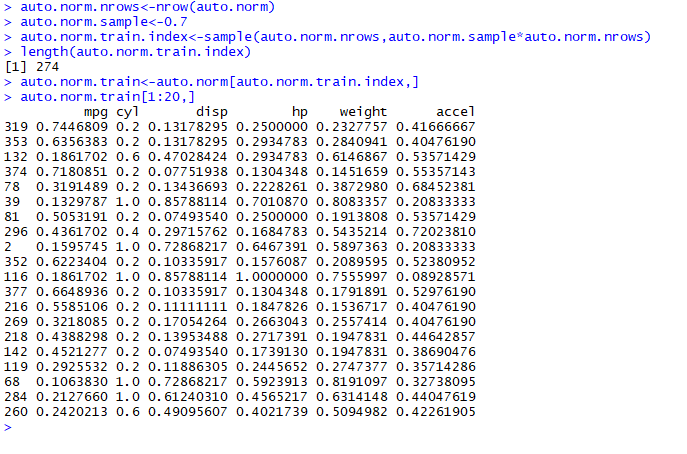
prob If this is true, the proportion of the votes for the winning class are returned as attribute prob.

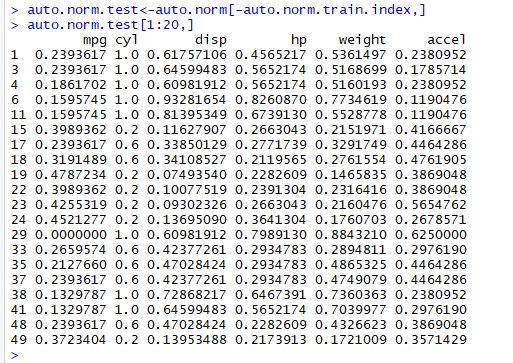
use.all controls handling of ties. If true, all distances equal to the kth largest are included. If false, a random selection of distances equal to the kth is chosen to use exactly k neighbours.

Knn does not compute decision boundaries, e.g., how the data points are separated in the data space, but these can be inferred. One way is to use Voronoi diagrams, where the line segments represent the boundaries between classes and are equidistant between two points of opposite classes. (YOU DO NOT HAVE TO DO THIS!)



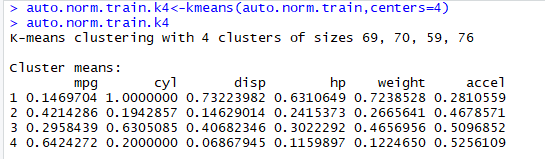
Creating training and test sets:





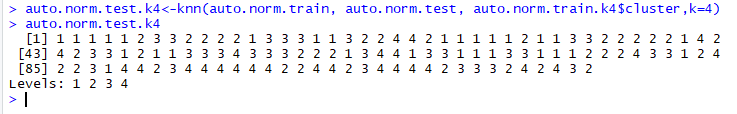
The only labels we have for the data points are the cluster ids after performing kmeans clustering.

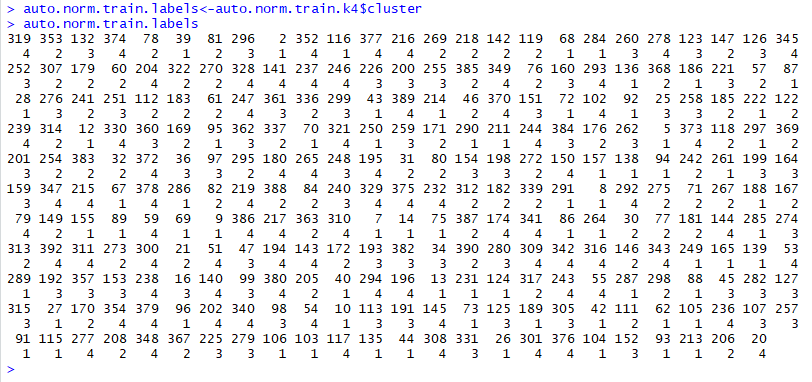
Create training labels for kNN :



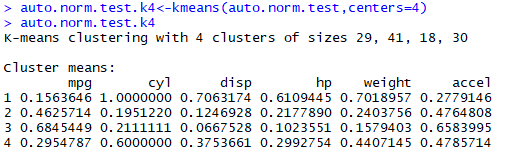
kNN returns feature vector with the predicted values for each of the test values.

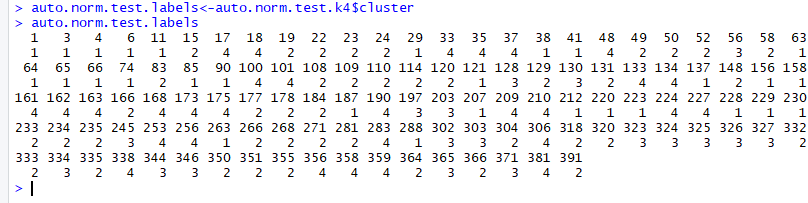
Run knn with k=4:





Create test labels via kmeans:

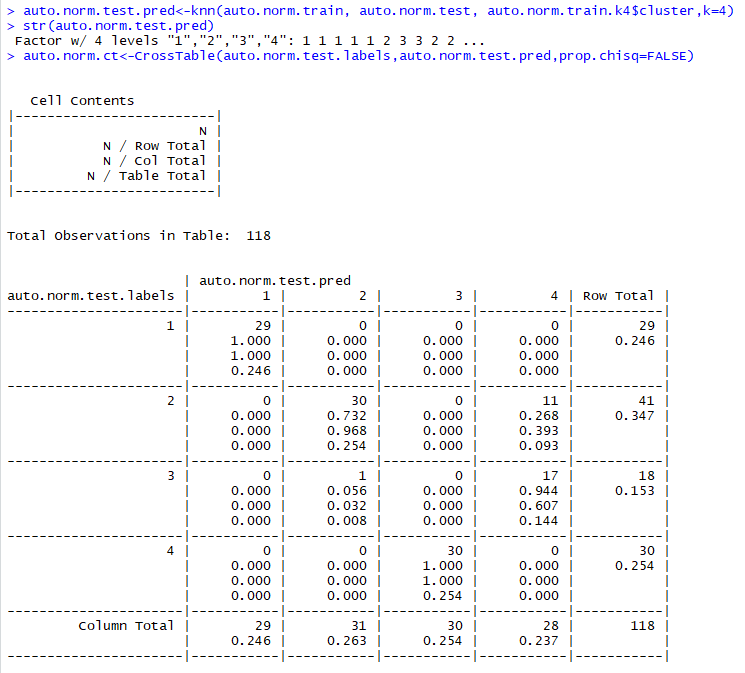




Compare the values of auto.norm.test.labels from knn to auto.norm.test.k4 from kmeans.

Evaluating kNN:

Use CrossTable in gmodels package:



The kmeans clusters were:

Cluster Count

1 29

2 41

3 18

4 30

Well, this seems OK, but the actual numbers tell a different story.

Compute the performance measures:

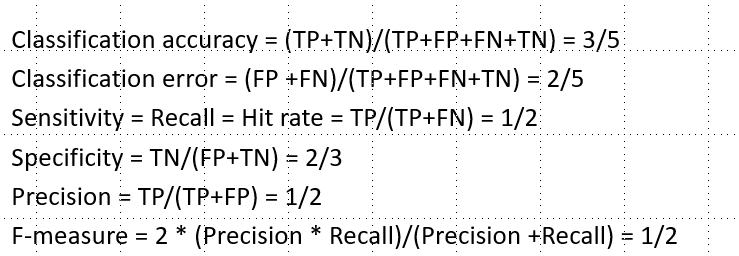
Performance Measures for Classification Models:

TP = True Positives

FP = False Positives

TN = True Negatives

FN = False Negatives



59 examples were classified correctly – clusters 1 and 2, but 59 were also classified incorrectly.

So, precision was 50%.

With k=2 and k=3, there was significant disparity in distributions in the clusters.

Together, these results suggest that:

(a) there are more than 4 clusters – good possibility

(b) there are dependencies between some of the attributes.

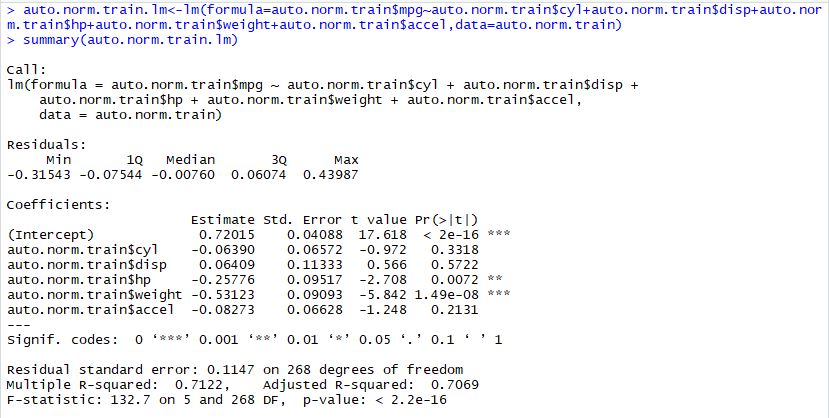
So, let’s use regression to see if we can detect where the dependencies might lie.

Clearly, mpg is dependent on cyl, disp, hp, weight, and accel for most cars then (when the data were collected).

We will use the lm function in the stats package which is automatically loaded by R when you initiate RStudio.

Models for lm are specified symbolically. A typical model has the form response ~ terms where response is the (numeric) response vector and terms is a series of terms which specifies a linear predictor for response.

Note: we have no weights for the different terms, e.g., we are treating them each equally. This is probably not realistic, but we have no data to justify any weights, except perhaps our intuition.



This is the linear model for the training set.

Note the squiggle (~) after auto.norm$mpg.

The Estimate gives an estimate of the regression coefficients.

The Std. Error gives the standard error for those estimates.

The t value = coefficient/Std. Error

The Pr(>|t|) converts the t value into a probability of the coefficient being 0.

It is customary to consider a coefficient to be significant at a 95 % level of significance (that is,  
probability being less than 0.05), which is represented by a "\*"

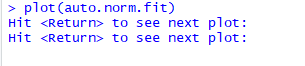
The Residual Standard Error is the RMS adjusted for the degrees of freedom and is an  
excellent indicator of the average deviation of the predicted value from the actual value.

The Adjusted R-squared value tells us what percentage of the variation in the outcome variable  
the regression model explains.

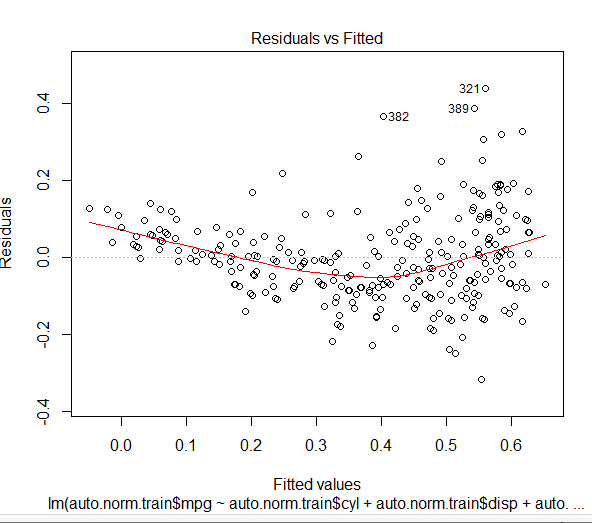
By default, lm uses the lowest level of Cyl, the first argument, as the reference level.

We use the whole data set rather than the train and test sets.(auto.norm.fit$fitted





Need to press “Enter” several times to see different plots.



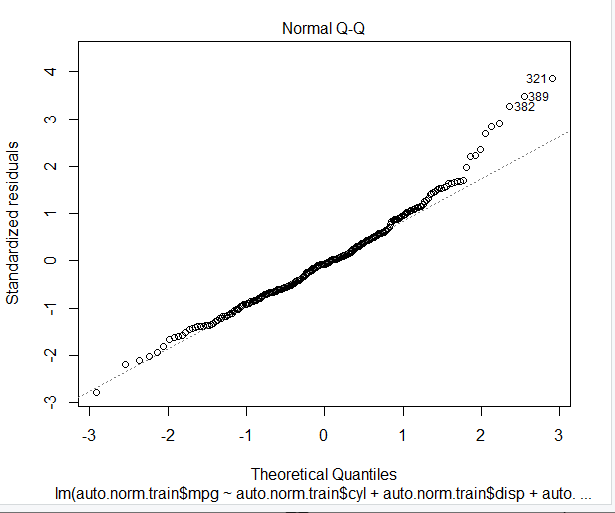
If residuals have a non-linear pattern, there could be a non-linear relationship between predictor variables and an outcome variable. The residuals are not equally distributed around the plot.

What do we see?

a. There are some outliers that we might want to remove that seem to be dragging the curve upward.

b. Since a curve is fitted, tells us there are some dependencies.

Source: <https://data.library.virginia.edu/diagnostic-plots/>

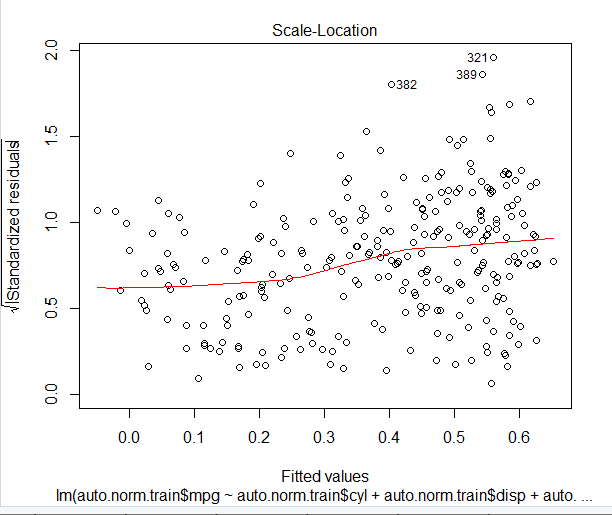


In statistics, a **Q–Q (quantile-quantile) plot** is a probability plot, which is a graphical method for comparing two probability distributions by plotting their quantiles against each other

If the distributions are linearly related, the points in the Q–Q plot will approximately lie on a line, but not necessarily on the line *y* = *x*.

This plot indicates the data are almost normally distributed, except for the high end. Points 321,382,389, are problematics. We should go back and review their values and maybe remove them from the data set.

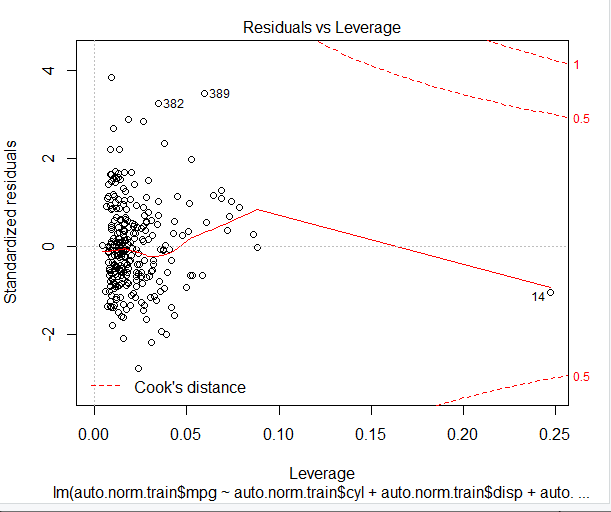
The deviation from the line at the high end is due to the outliers.



Also called a **Spread-Location plot**.

This plot shows if residuals are spread equally along the ranges of predictors. This is how you can check the assumption of equal variance (homoscedasticity). It’s good if you see a horizontal line with equally (randomly) spread points.

Difficult to visualize, but it seems more points are plotted above the line as we move to the right.



This plot helps us to find influential cases (i.e., subjects) if any. Not all outliers are influential in linear regression analysis (whatever outliers mean). Even though data have extreme values, they might not be influential to determine a regression line. That means, the results wouldn’t be much different if we either include or exclude them from analysis. They follow the trend in the majority of cases and they don’t really matter; they are not influential. On the other hand, some cases could be very influential even if they look to be within a reasonable range of the values. They could be extreme cases against a regression line and can alter the results if we exclude them from analysis. Another way to put it is that they don’t get along with the trend in the majority of the cases.

We look in the upper right or lower right for outlying values.

When points are outside of Cook’s distance, they are influential in affecting the regression.

We don’t have that case here. But, point 14 is problematic as it is an outlier.

What does having patterns in residuals mean to your investigation?

It tells you about your model and data. Your current model might not be the best way to understand your data if there’s so much good stuff left in the data.

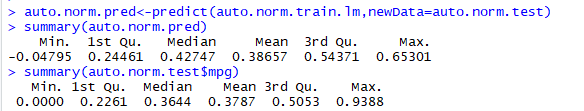
You may want to include a quadratic term in your model or you may have left our an important variable or you may want to remove outliers or problematic data.

How do you recover the original values since this analysis was conducted with the scaled values?

See the formulas above for recovering the original values from the normalized values.

Exercise for the reader: Repeat this analysis using glm which is based on lm.

Predict returns a vector of predicted values for the dependent variable (e.g., mpg) based on the training set.



So, maybe 4 clusters is not the best choice.

Note:

KMeans: dist(x,y) =

kNN: dist(x,y)=

Same equation, but different algorithms. Still results in clustering/classification.

So, conclude clustering and classification are two aspects of the same analytical process since they both result in labeling the ojects in a data set.

So, things to do:

1. Determine the dependent variable, e.g., mpg, and express it in the formula as the result of the other variables.

2. If you get questionable results (as above), then consider eliminating one or more predictor variables

3. Remember: Plotting is your friend!!

4. Rerun the kmeans after removing those predictor variable columns from the data set. Also, try 5, 6,… clusters.

5. Rerun kNN and lm to see if the numbers improve with k > 5.

Appendix : Different Formula Expressions

