**MACHINE LEARNING CONCEPTS**

There are two types of machine learnings techniques.

1. Supervised Learning

This technique can be used when we know the output variable Y

1. Un-Supervised Learning

This technique can be used when we do not know the output variable Y.

**Un-Supervised Learning:**

There are three types of Un-Supervised learning Algorithms.

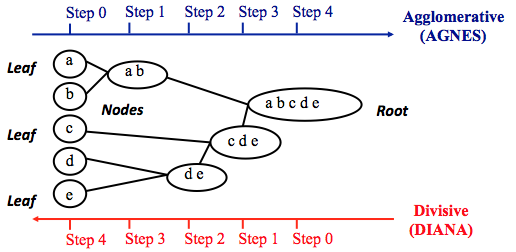
1. Hierarchical Clustering (H- Clustering)
2. K-Means Clustering
3. Principal Component Analysis (PCA)

**H- Clustering:**

Hierarchical clustering can be divided into two main types: *agglomerative* and *divisive*.

1. **Agglomerative clustering:** It’s also known as AGNES (Agglomerative Nesting). It works in a bottom-up manner. That is, each object is initially considered as a single-element cluster (leaf). At each step of the algorithm, the two clusters that are the most similar are combined into a new bigger cluster (nodes). This procedure is iterated until all points are member of just one single big cluster (root) (see figure below). The result is a tree which can be plotted as a dendrogram.
2. **Divisive hierarchical clustering:** It’s also known as DIANA (Divise Analysis) and it works in a top-down manner. The algorithm is an inverse order of AGNES. It begins with the root, in which all objects are included in a single cluster. At each step of iteration, the most heterogeneous cluster is divided into two. The process is iterated until all objects are in their own cluster (see figure below).

Note that agglomerative clustering is good at identifying small clusters. Divisive hierarchical clustering is good at identifying large clusters.

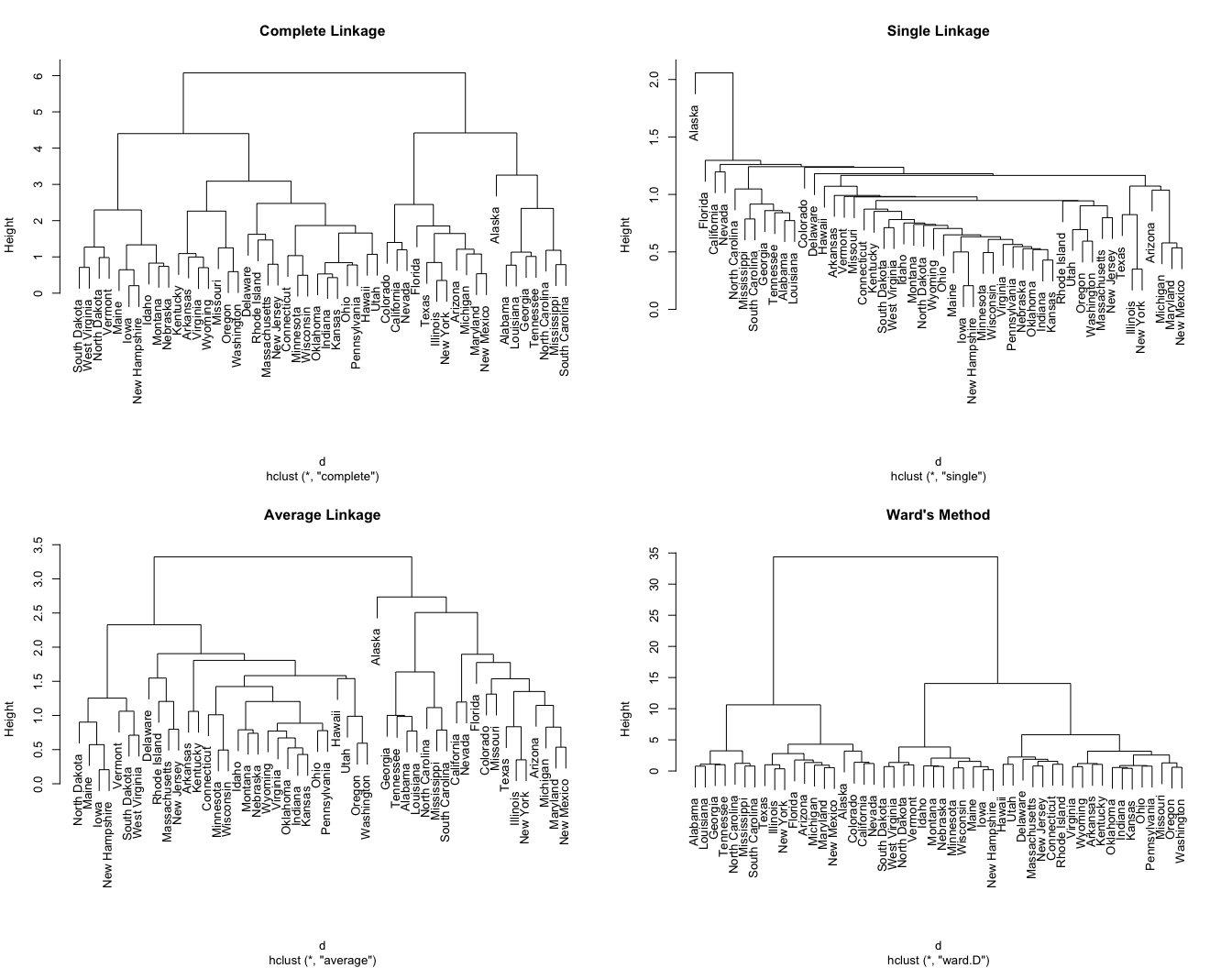


In R, the Euclidean distance is used by default to measure the dissimilarity between each pair of observations. As we already know, it’s easy to [compute the dissimilarity measure](https://uc-r.github.io/kmeans_clustering#distance) between two pairs of observations with the “get\_dist” function.

How do we measure the dissimilarity between two clusters of observations? A number of different cluster agglomeration methods (i.e, linkage methods) have been developed to answer to this question. The most common type’s methods are:

* **Maximum or complete linkage clustering:** It computes all pairwise dissimilarities between the elements in cluster 1 and the elements in cluster 2, and considers the largest value (i.e., maximum value) of these dissimilarities as the distance between the two clusters. It tends to produce more compact clusters.
* **Minimum or single linkage clustering:** It computes all pairwise dissimilarities between the elements in cluster 1 and the elements in cluster 2, and considers the smallest of these dissimilarities as a linkage criterion. It tends to produce long, “loose” clusters.
* **Mean or average linkage clustering:** It computes all pairwise dissimilarities between the elements in cluster 1 and the elements in cluster 2, and considers the average of these dissimilarities as the distance between the two clusters.
* **Centroid linkage clustering:** It computes the dissimilarity between the centroid for cluster 1 (a mean vector of length p variables) and the centroid for cluster 2.
* **Ward’s minimum variance method:** It minimizes the total within-cluster variance. At each step the pair of clusters with minimum between-cluster distance are merged.

We can see the differences these approaches in the following dendrogram:



**Step -1 : Data Preparation**

To perform a cluster analysis in R, generally, the data should be prepared as follows:

1. Rows are observations (individuals) and columns are variables
2. Any missing value in the data must be removed or estimated.
3. The data must be standardized (i.e., scaled) to make variables comparable. Recall that, standardization consists of transforming the variables such that they have mean zero and standard deviation one.[^scale]

**data()** # it is showing sample data from R

**data("USArrests")** # choose the dataset "USArrests from data"

**View(USArrests)** # View the data set

**# (OR)**

**df <- USArrests** # choose the dataset "USArrests from data"

**View(df)** # View the data

**df <- na.omit(df)** # To remove any missing value that might be present in the data

**View(df**) # View the data after removing any missing values

**df1 <-scale(df)** # we start by scaling/standardizing the data using the R function scale

**head(df1)**

head(df1)

Murder Assault UrbanPop Rape

Alabama 1.24256408 0.7828393 -0.5209066 -0.003416473

Alaska 0.50786248 1.1068225 -1.2117642 2.484202941

Arizona 0.07163341 1.4788032 0.9989801 1.042878388

Arkansas 0.23234938 0.2308680 -1.0735927 -0.184916602

California 0.27826823 1.2628144 1.7589234 2.067820292

Colorado 0.02571456 0.3988593 0.8608085 1.864967207

**Hierarchical Clustering with R**

There are different functions available in R for computing hierarchical clustering. The commonly used functions are:

* hclust [in stats package] and agnes [in cluster package] for agglomerative hierarchical clustering (HC)
* diana [in cluster package] for divisive HC

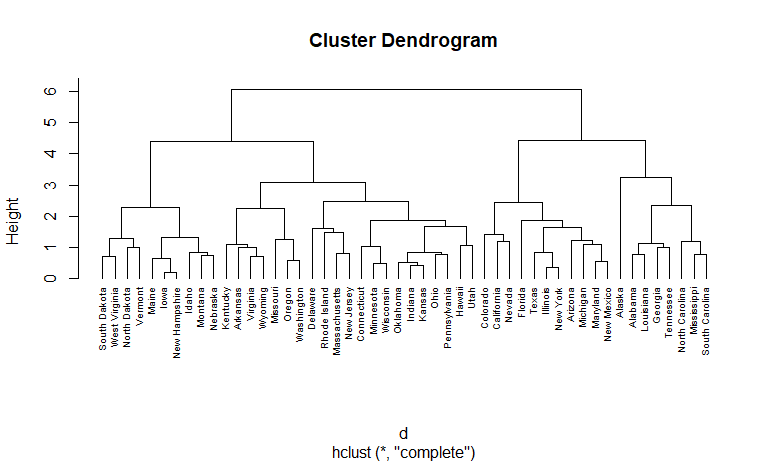
### **Step: 2 Agglomerative Hierarchical Clustering**

We can perform agglomerative HC with hclust. First we compute the dissimilarity values with dist and then feed these values into hclust and specify the agglomeration method to be used (i.e. “complete”, “average”, “single”, “ward.D”). We can then plot the dendrogram.

**d <- dist(df, method = "euclidean")** # Dissimilarity matrix

**hc1 <- hclust(d, method = "complete" )** # Hierarchical clustering using Complete Linkage

**plot(hc1, cex = 0.6, hang = -1)** # Plot the obtained dendrogram



Alternatively, we can use the agnes function. These functions behave very similarly; however, with the agnes function you can also get the agglomerative coefficient, which measures the amount of clustering structure found (values closer to 1 suggest strong clustering structure).

**hc2 <- agnes(df, method = "complete")** # Compute with agnes

**hc2$ac** # Agglomerative coefficient

hc2$ac

[1] 0.8531583

# which methods to assess for best

**m <- c( "average", "single", "complete", "ward")**

**names(m) <- c( "average", "single", "complete", "ward")**

# function to compute coefficient

**ac <- function(x) {**

**agnes(df, method = x)$ac**

**}**

**map\_dbl(m, ac)**

map\_dbl(m, ac)

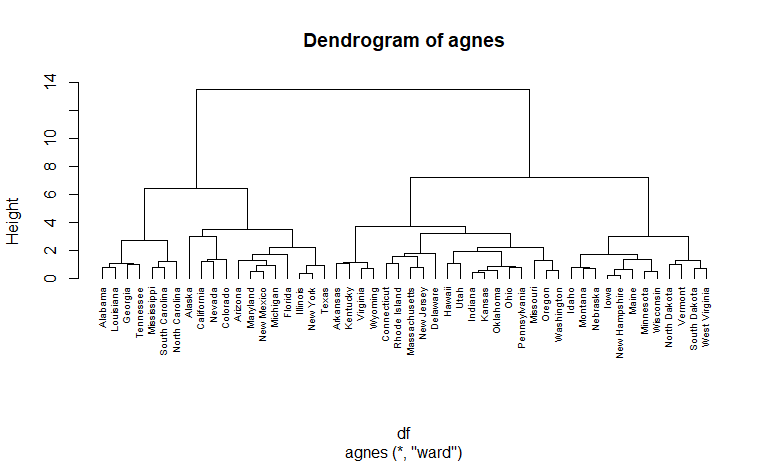
average single complete ward

0.7379371 0.6276128 0.8531583 0.9346210

# Similar to before we can visualize the dendrogram:

**hc3 <- agnes(df, method = "ward")**

**pltree(hc3, cex = 0.6, hang = -1, main = "Dendrogram of agnes")**



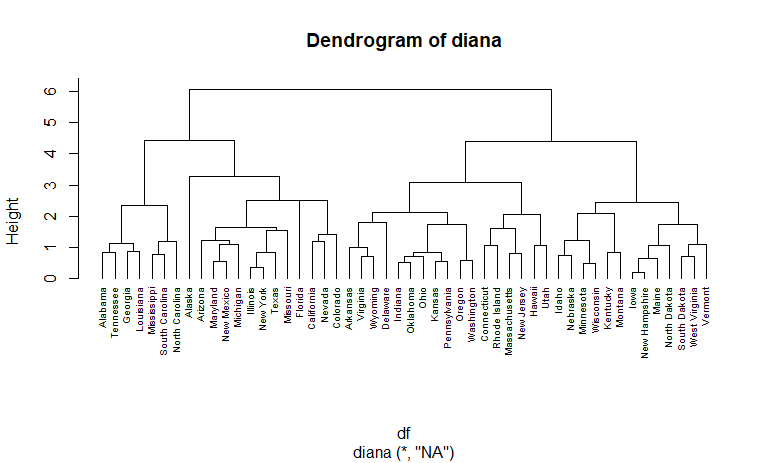
### **Step: 3 Divisive Hierarchical Clustering**

The R function diana provided by the cluster package allows us to perform divisive hierarchical clustering. diana works similar to agnes; however, there is no method to provide.

**hc4 <- diana(df)** # compute divisive hierarchical clustering

**hc4$dc** # Divise coefficient; amount of clustering structure found

**pltree(hc4, cex = 0.6, hang = -1, main = "Dendrogram of diana")** # plot dendrogram



**Step: 4 Working with Dendrogram**

**hc5 <- hclust(d, method = "ward.D2" )** # Ward's method

**sub\_grp <- cutree(hc5, k = 4)** # Cut tree into 4 groups

**table(sub\_grp)** # Number of members in each cluster

> table(sub\_grp) # Number of members in each cluster

sub\_grp

1 2 3 4

7 12 19 12

We can also use the cutree output to add the cluster each observation belongs to our original data.

**USArrests %>%**

**mutate(cluster = sub\_grp) %>%**

**head**

Murder Assault UrbanPop Rape cluster

1 13.2 236 58 21.2 1

2 10.0 263 48 44.5 2

3 8.1 294 80 31.0 2

4 8.8 190 50 19.5 3

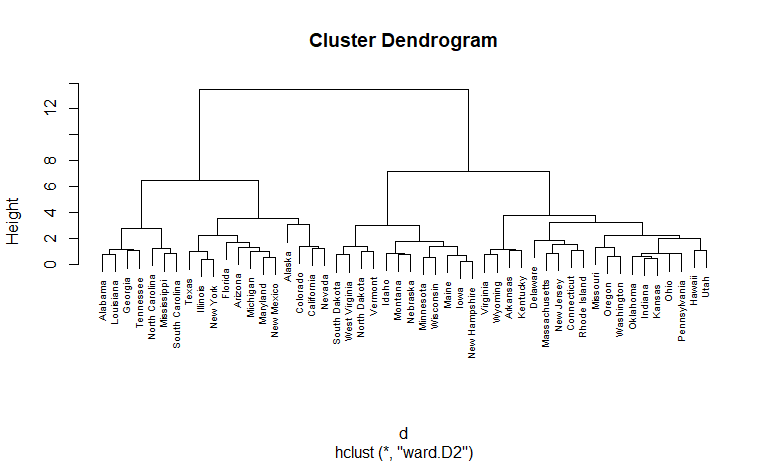
5 9.0 276 91 40.6 2

6 7.9 204 78 38.7 2

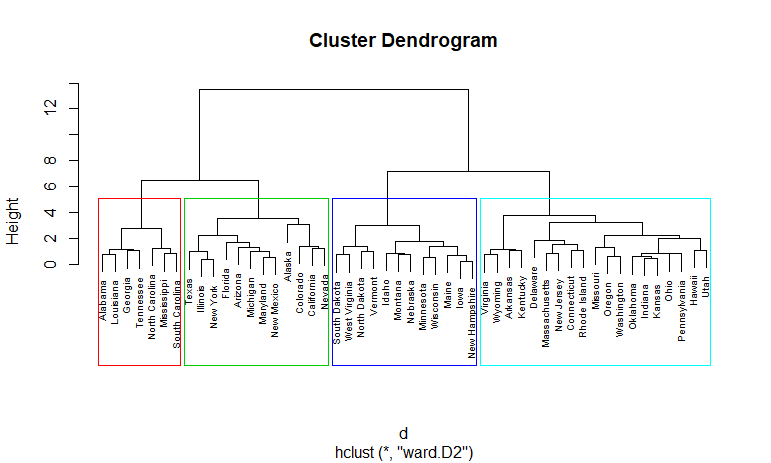
It’s also possible to draw the dendrogram with a border around the 4 clusters. The argument border is

used to specify the border colors for the rectangles:

**plot(hc5, cex = 0.6)** # Dispaly the Dendrogram



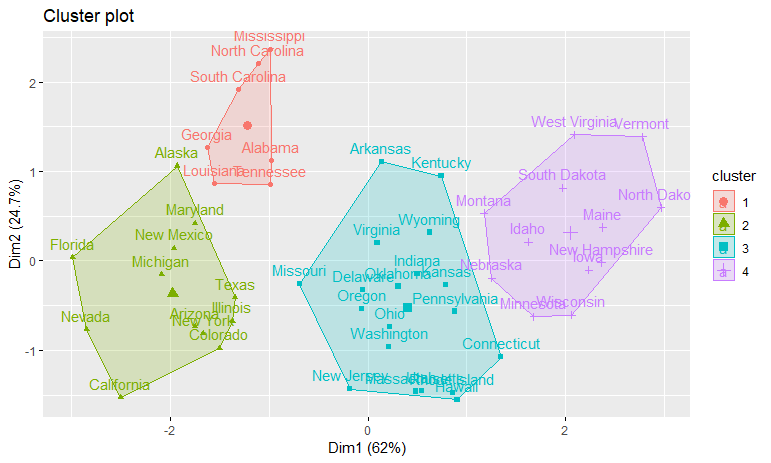
**rect.hclust(hc5, k = 4, border = 2:5)** # display the clusters with boarders



We can also use the fviz\_cluster function from the factoextra package to visualize the result in a scatter

plot.

**fviz\_cluster(list(data = df, cluster = sub\_grp))**



To use cutree with agnes and diana you can perform the following:

# Cut agnes() tree into 4 groups

**hc\_a <- agnes(df, method = "ward")**

**cutree(as.hclust(hc\_a), k = 4)**

# Cut diana() tree into 4 groups

**hc\_d <- diana(df)**

**cutree(as.hclust(hc\_d), k = 4)**

Lastly, we can also compare two dendrograms. Here we compare hierarchical clustering with complete linkage versus Ward’s method. The function tanglegram plots two dendrograms, side by side, with their

labels connected by lines.

# Compute distance matrix

**res.dist <- dist(df, method = "euclidean")**

# Compute 2 hierarchical clusterings

**hc1 <- hclust(res.dist, method = "complete")**

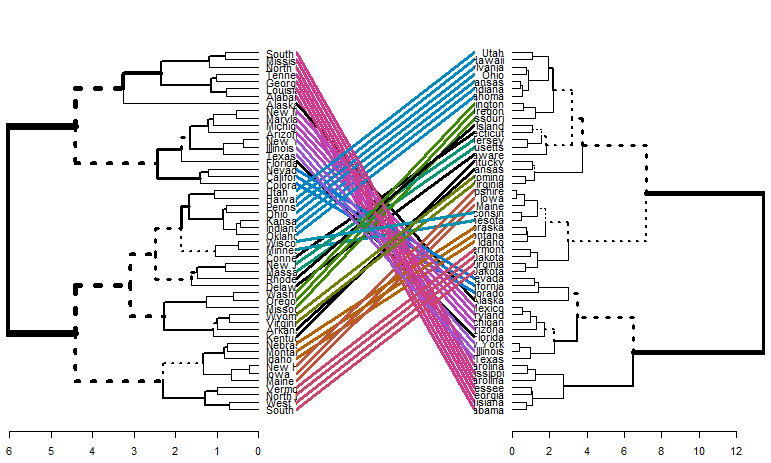
**hc2 <- hclust(res.dist, method = "ward.D2")**

# Create two dendrograms

**dend1 <- as.dendrogram (hc1)**

**dend2 <- as.dendrogram (hc2)**

**tanglegram(dend1, dend2)**



The output displays “unique” nodes, with a combination of labels/items not present in the other tree, highlighted with dashed lines. The quality of the alignment of the two trees can be measured using the function entanglement. Entanglement is a measure between 1 (full entanglement) and 0 (no entanglement). A lower entanglement coefficient corresponds to a good alignment. The output of tanglegram can be

customized using many other options as follow:

**dend\_list <- dendlist(dend1, dend2)**

**tanglegram(dend1, dend2,**

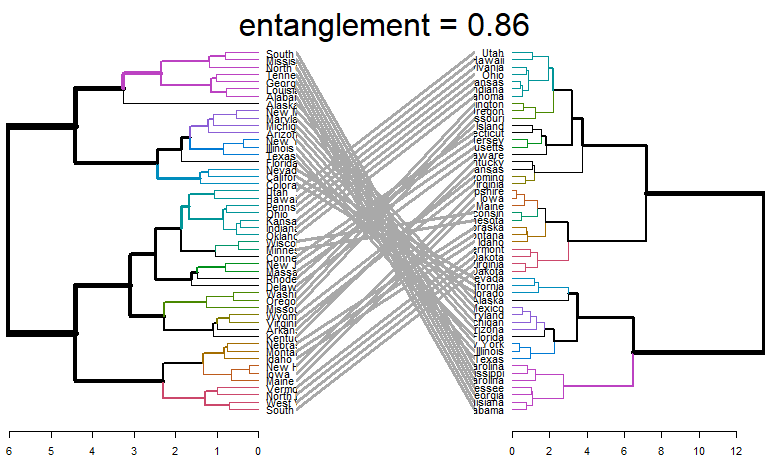
**highlight\_distinct\_edges = FALSE, # Turn-off dashed lines**

**common\_subtrees\_color\_lines = FALSE, # Turn-off line colors**

**common\_subtrees\_color\_branches = TRUE, # Color common branches**

**main = paste("entanglement =", round(entanglement(dend\_list), 2))**

**)**



## Step: 5 Determining Optimal Clusters

### **Elbow Method**

## To perform the [elbow method](https://uc-r.github.io/kmeans_clustering#elbow) we just need to change the second argument in fviz\_nbclust to FUN = hcut.

## fviz\_nbclust(df, FUN = hcut, method = "wss") # Elbow Method

## 

### **Average Silhouette Method**

## fviz\_nbclust(df, FUN = hcut, method = "silhouette") # Average Silhouette Method

## 

### **Gap Statistic Method**

## ap\_stat <- clusGap(df, FUN = hcut, nstart = 25, K.max = 10, B = 50) # Gap Statistic Method

## fviz\_gap\_stat(gap\_stat)

## 

**PRINCIPAL COMPONENT ANALYSIS (PCA)**

PCAs primary purpose is **NOT** as a ways of feature removal! PCA can reduce dimensionality but **it won’t reduce the number of features / variables in your data.** What this means is that you might discover that you can explain 99% of variance in your 1000 feature dataset by just using 3 principal components but you still need those 1000 features to construct those 3 principal components, this also means that in the case of predicting on future data you still need those same 1000 features on your new observations to construct the corresponding principal components.

## General methods for principal component analysis

There are two general methods to perform PCA in R:

* Spectral decomposition which examines the covariances / correlations between variables
* Singular value decomposition which examines the covariances / correlations between individuals

The function **princomp**() uses the spectral decomposition approach. The functions **prcomp**() and **PCA**()[FactoMineR] use the singular value decomposition (SVD).

## prcomp() and princomp() functions

The simplified format of these 2 functions are:

**prcomp(x, scale = FALSE)**

**princomp(x, cor = FALSE, scores = TRUE)**

**View(universities)** # view of the dataset

**mydata<- universities[-1]** # remove the first column which is not required for the analysis

**View(mydata)**

**pcaobj <-princomp(mydata , cor = TRUE , scores = TRUE , covmat = NULL )** # Calculate Principal component

**summary(pcaobj)**

summary(pcaobj)

Importance of components:

Comp.1 Comp.2 Comp.3 Comp.4 Comp.5 Comp.6 Comp.7 Comp.8

Standard deviation 2.3689840 1.3367184 1.1342671 1.0787864 0.35257084 0.162636495 2.410256e-08 0

Proportion of Variance 0.5612085 0.1786816 0.1286562 0.1163780 0.01243062 0.002645063 5.809336e-17 0

Cumulative Proportion 0.5612085 0.7398901 0.8685463 0.9849243 0.99735494 1.000000000 1.000000e+00 1

Comp.9 Comp.10

Standard deviation 0 0

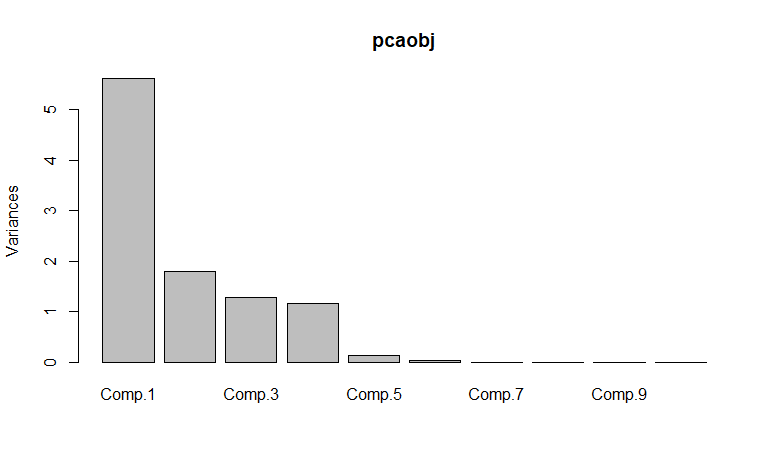
Proportion of Variance 0 0

Cumulative Proportion 1 1

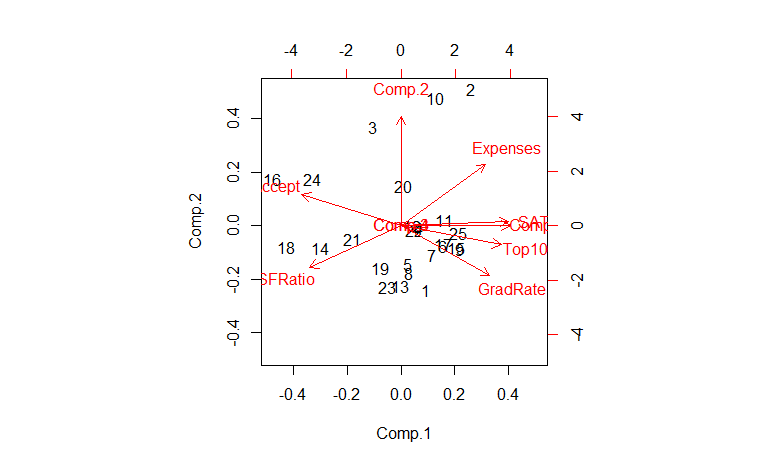
Recall that a property of PCA is that our components are sorted from largest to smallest with regard to their standard deviation (**Eigenvalues**). So let’s make sense of these:

* ***Standard deviation:*** This is simply the **eigenvalues**in our case since the data has been centered and scaled (**standardized**)
* ***Proportion of Variance***: This is the amount of variance the component accounts for in the data, ie. **PC1**accounts for **>44% of total variance** in the data alone
* ***Cumulative Proportion***: This is simply the accumulated amount of explained variance, ie. if we used **the first 10 components** we would be able to account for **>95% of total variance** in the data.

**plot(pcaobj)** # plot of the data



**biplot(pcaobj)**



**pcaobj$scores** # it is will display all the columns

**pcaobj$scores[, 1:4]** # it will display only 4 columns which are required for analysis and it will give 97% of accuracy of data

pcaobj$scores[, 1:4] # it will display only 4 columns whica re required for analysis and it will give 97% of accuracy of data

Comp.1 Comp.2 Comp.3 Comp.4

[1,] 1.11398887 -1.60387785 0.17176969 0.15181108

[2,] 3.11320038 3.40430218 1.77313898 0.38344103

[3,] -1.22715699 2.45817547 -0.56528901 2.86578536

[4,] 0.81821207 -0.06357506 0.12819401 -0.41906352

[5,] 0.34406908 -0.95729013 0.02169842 0.45679743

[6,] 1.87161362 -0.51894358 -0.53693983 0.03349224

[7,] 1.37536368 -0.73989543 -0.06800292 -0.54817657

[8,] 0.37367367 -1.18322195 -1.04584650 0.10624340

[9,] 2.61891646 -0.58250075 0.24599844 -1.20851896

[10,] 1.54795024 3.19403596 -0.93829710 -1.68614582

[11,] 1.90405836 0.13297029 0.36106473 0.69547601

[12,] 0.49733926 -0.01677697 -0.37238459 0.62953746

[13,] -0.04438655 -1.52083616 -1.05205977 0.61123271

[14,] -3.56711630 -0.56447986 -1.04963503 -1.38944294

[15,] 2.46681590 -0.56027966 -0.84541925 1.08483460

[16,] -5.70630900 1.17530550 -0.81770151 -0.61899766

[17,] 1.87487093 -0.46052481 0.67487958 -0.79293462

[18,] -5.05013639 -0.52380818 3.17755933 -1.21088100

[19,] -0.90741054 -1.05322743 3.02535962 2.02797824

[20,] 0.10784106 0.98020035 0.21296522 -1.33308947

[21,] -2.16557967 -0.33871695 -0.54218178 -0.12921552

[22,] 0.59819698 -0.11993763 -0.64709180 0.35106422

[23,] -0.58709098 -1.53283585 -0.89780368 0.45192469

[24,] -3.91455450 1.17311485 -0.95214647 0.86281741

[25,] 2.54363035 -0.17737634 0.53817122 -1.37596979

**# Considering top 4 principal component scores and binding them with mydata**

**mydata<-cbind(mydata,pcaobj$scores[,1:4])** # cbind used to bind the data in column wise

**View(mydata)**

**# preparing data for clustering (considering only pca scores as they represent the entire data)**

**clus\_data<-mydata[,7:10]**

**# Normalizing the data**

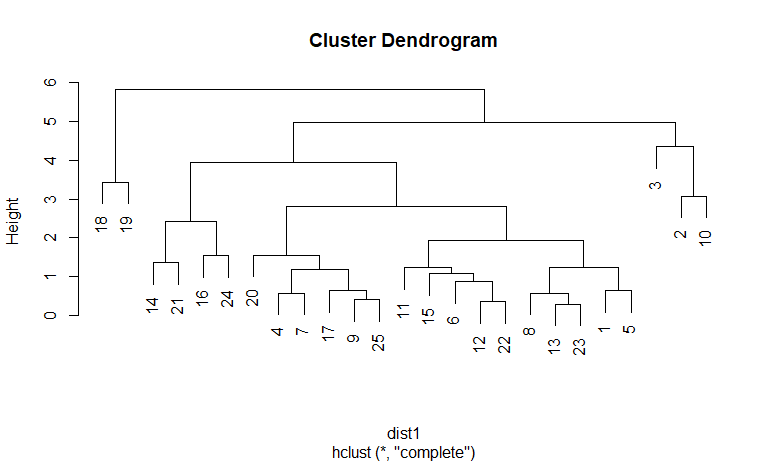
**norm\_clus<-scale(clus\_data)** # Scale function is used to normalize data

**dist1<-dist(norm\_clus,method = "euclidean")** # method for finding the distance

**# Clustering the data using hclust function --> Hierarchical**

**fit1<-hclust(dist1,method="complete")** # method here is complete linkage

**plot(fit1)** # Displaying Dendrogram



**groups<-cutree(fit1,3)** # Cutting the dendrogram for 3 clusters

**membership\_1<-as.matrix(groups)** # cluster numbering

**View(membership\_1)**

**final1<-cbind(membership\_1,mydata)** # binding column wise with original data

**View(final1)**

**View(aggregate(final1[,-c(2,9:11)],by=list(membership\_1),FUN=mean))** # drawn from the aggregate of the universities data on membership\_1

**K-Means Clustering**

## Data Preparation:

Prior to clustering data, you may want to remove or estimate missing data and rescale variables for comparability.

**# Prepare Data**

**mydata <- na.omit(university)** # listwise deletion of missing

**View(university)**

**mydata5 <- scale(university[,3:8])** # standardize variables after removing first & second columns

**View(mydata5)**

## Partitioning:

## ****K-means**** clustering is the most popular partitioning method. It requires the analyst to specify the number of clusters to extract. A plot of the within groups sum of squares by number of clusters extracted can help determine the appropriate number of clusters. The analyst looks for a bend in the plot similar to a screen test in factor analysis.

## # Determine number of clusters

## wss <- (nrow(mydata5)-1)\*sum(apply(mydata5,2,var))

## for (i in 2:15) wss[i] <- sum(kmeans(mydata5, centers=i)$withinss)

## plot(1:15, wss, type="b", xlab="Number of Clusters",ylab="Within groups sum of squares")

## 

## # K-Means Cluster Analysis

## fit <- kmeans(mydata5, 5) # 5 cluster solution

## fit

## # get cluster means

## aggregate(mydata5,by=list(fit$cluster),FUN=mean)

## # append cluster assignment

## mydata6 <- data.frame(mydata5, fit$cluster)

## View(mydata6)

## Hierarchical Agglomerative

There are a wide range of hierarchical clustering approaches. I have had good luck with Ward's method described below.

**# Ward Hierarchical Clustering**

**d <- dist(mydata6, method = "euclidean")** # distance matrix

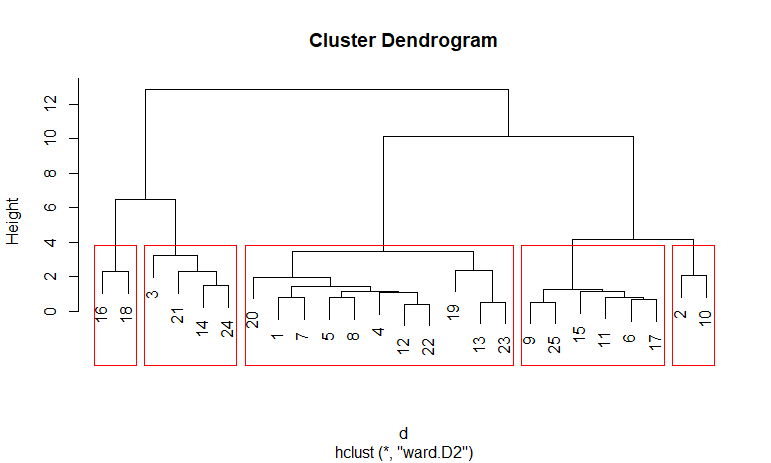
**fit <- hclust(d, method="ward.D2")**

**plot(fit)** # display dendogram

**groups <- cutree(fit, k=5)** # cut tree into 5 clusters

**# draw dendogram with red borders around the 5 clusters**

**rect.hclust(fit, k=5, border="red")**



**# Ward Hierarchical Clustering with Bootstrapped p values**

**install.packages("pvclust")**

**library(pvclust)**

**fit <- pvclust(mydata, method.hclust="ward.D2",method.dist="euclidean")**

**plot(fit)** # dendogram with p values

**# add rectangles around groups highly supported by the data**

**pvrect(fit, alpha=.95)**

****Model Based

Model based approaches assume a variety of data models and apply maximum likelihood estimation and Bayes criteria to identify the most likely model and number of clusters. Specifically, the **Mclust( )**function in the [mclust](http://cran.r-project.org/web/packages/mclust/index.html) package selects the optimal model according to BIC for EM initialized by hierarchical clustering for parameterized Gaussian mixture models. (phew!). One chooses the model and number of clusters with the largest BIC

**# Model Based Clustering**

**install.packages("mclust")**

**library(mclust)**

**fit <- Mclust(mydata6)**

**plot(fit)** # plot results

**summary(fit)**

**# display the best model**

Selection: 1

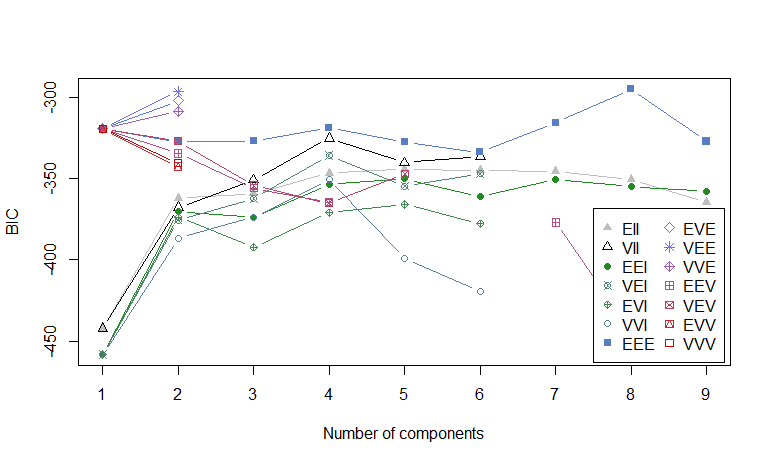
Model-based clustering plots:

1: BIC

2: classification

3: uncertainty

4: density



Selection: 2

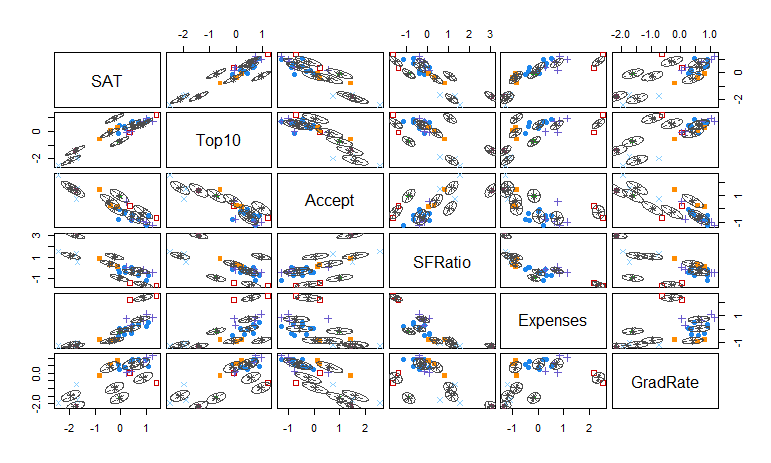
Model-based clustering plots:

1: BIC

2: classification

3: uncertainty

4: density



Selection: 3

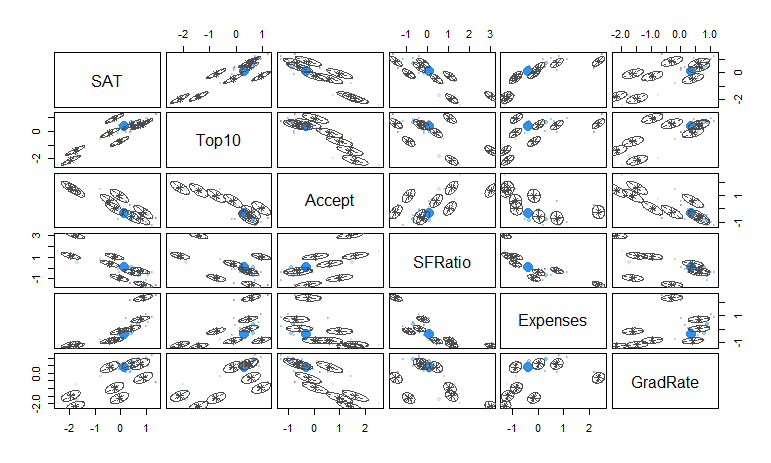
Model-based clustering plots:

1: BIC

2: classification

3: uncertainty

4: density



Selection: 4

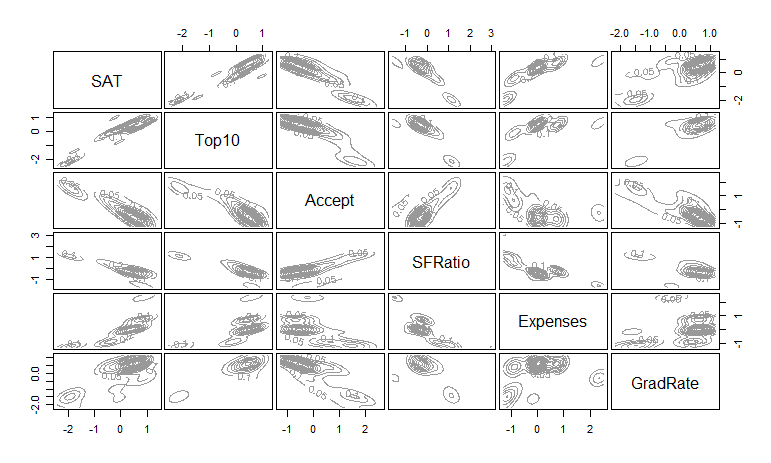
Model-based clustering plots:

1: BIC

2: classification

3: uncertainty

4: density



## Plotting Cluster Solutions

It is always a good idea to look at the cluster results.

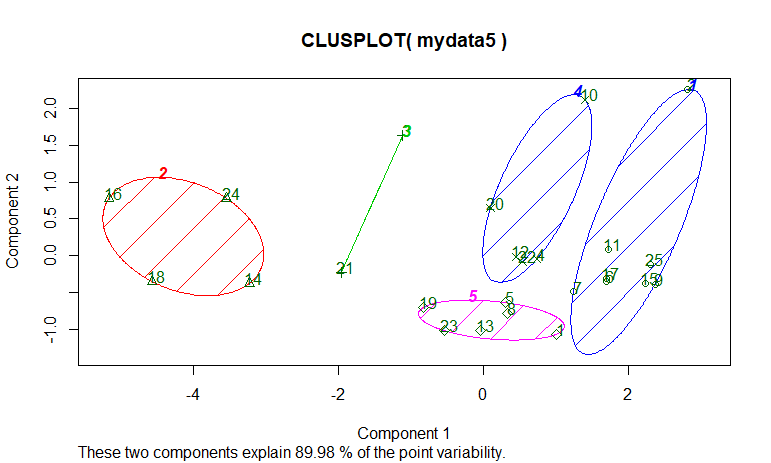
**# K-Means Clustering with 5 clusters**

**fit <- kmeans(mydata5, 5)** # Cluster Plot against 1st 2 principal components

**# vary parameters for most readable graph**

**library(cluster)**

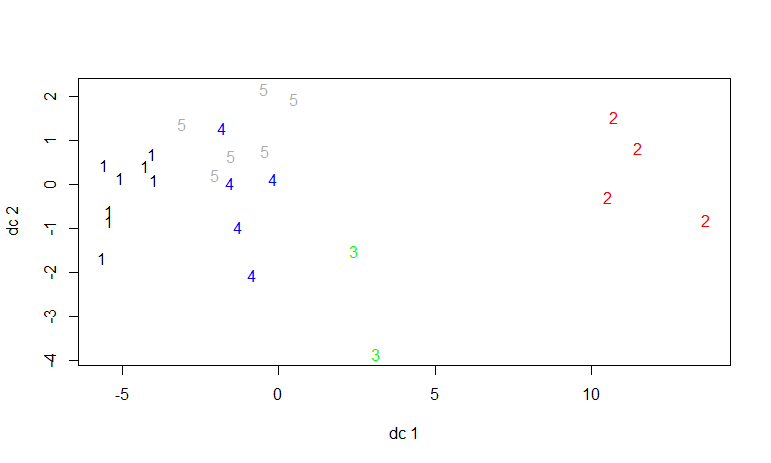
**clusplot(mydata5, fit$cluster, color=TRUE, shade=TRUE,labels=2, lines=0)**



**# Centroid Plot against 1st 2 discriminant functions**

**library(fpc)**

**plotcluster(mydata5, fit$cluster)**



## Validating cluster solutions

The function **cluster.stats()**in the [fpc](http://cran.r-project.org/web/packages/fpc/index.html) package provides a mechanism for comparing the similarity of two cluster solutions using a variety of validation criteria (Hubert's gamma coefficient, the Dunn index and the corrected rand index)

**# comparing 2 cluster solutions**

**library(fpc)**

**cluster.stats(d, fit1$cluster,fit2$cluster)**

**SUPERVISED LEARNING ALGORITHMS**

There are five types of supervised learning algorithms

1. K-Nearest Neighbor (KNN)
2. Support Vector Machine (SVM)
3. Random Forest
4. Decision Tree
5. Naïve Bayes Theorem

**K-Nearest Neighbor (KNN)**

**# drop the id feature**

**View(wbcd)**

**wbcd1<-wbcd[,-1]**

**View(wbcd1)**

**# table of diagnosis -> calculating number of entries with B and M**

**summary(wbcd1$diagnosis)**

**# recode diagnosis as a factor -> giving the full name to labels**

**wbcd1$diagnosis <- factor(wbcd1$diagnosis, levels = c("B", "M"),labels = c("Benign", "Malignant"))**

**View(wbcd1)**

**# table or proportions with more informative labels -> calculating the probability**

**round(prop.table(table(wbcd1$diagnosis)) \* 100, digits = 1)**

**# summary of dataset**

**summary(wbcd1)**

**# summarize three numeric features**

**summary(wbcd1[c("radius\_mean", "area\_mean", "smoothness\_mean")])**

**str(wbcd1)**

**# create normalization function**

**normalize <- function(x) {**

**return ((x - min(x)) / (max(x) - min(x)))**

**}**

**# test normalization function - result should be identical**

**#normalize(c(1, 2, 3, 4, 5))**

**#normalize(c(10, 20, 30, 40, 50))**

**# normalize the wbcd data -> applying the normalize function which is created in line 32**

**wbcd1\_n <- as.data.frame(lapply(wbcd1[2:31], normalize))**

**# confirm that normalization worked**

**View(wbcd1\_n)**

**summary(wbcd1\_n)**

**# create training and test data**

**wbcd1\_train <- wbcd1\_n[1:469, ]**

**View(wbcd1\_train)**

**wbcd1\_test <- wbcd1\_n[470:569, ]**

**View(wbcd1\_test)**

**# create labels for training and test data**

**wbcd\_train\_labels <- wbcd1[1:469, 1]**

**View(wbcd\_train\_labels)**

**wbcd\_test\_labels <- wbcd1[470:569, 1]**

**View(wbcd\_test\_labels)**

**#---- Training a model on the data ----**

**# load the "class" library**

**library(class)**

**wbcd\_train\_labels <- wbcd\_train\_labels[["diagnosis"]]**

**View(wbcd\_train\_labels)**

**wbcd\_test\_labels <- wbcd\_test\_labels[["diagnosis"]]**

**wbcd\_test\_pred <- knn(train = wbcd1\_train, test = wbcd1\_test,cl = wbcd\_train\_labels, k=23)**

**View(wbcd\_test\_pred)**

**##--------Evaluating model performance ----**

**# load the "gmodels" library**

**install.packages("gmodels")**

**library(gmodels)**

**# Create the cross tabulation of predicted vs. actual**

**CrossTable(x = wbcd\_test\_labels, y = wbcd\_test\_pred,prop.chisq=FALSE)**

**## Improving model performance ----**

**# re-classify test cases**

**wbcd\_test\_pred <- knn(train = wbcd1\_train, test = wbcd1\_test,cl = wbcd\_train\_labels, k=2)**

**# Create the cross tabulation of predicted vs. actual**

**CrossTable(x = wbcd\_test\_labels, y = wbcd\_test\_pred,prop.chisq=FALSE)**

**wbcd\_test\_pred <- knn(train = wbcd1\_train, test = wbcd1\_test, cl = wbcd\_train\_labels, k=1)**

**CrossTable(x = wbcd\_test\_labels, y = wbcd\_test\_pred, prop.chisq=FALSE)**

**wbcd\_test\_pred <- knn(train = wbcd1\_train, test = wbcd1\_test, cl = wbcd\_train\_labels, k=5)**

**CrossTable(x = wbcd\_test\_labels, y = wbcd\_test\_pred, prop.chisq=FALSE)**

**wbcd\_test\_pred <- knn(train = wbcd1\_train, test = wbcd1\_test, cl = wbcd\_train\_labels, k=11)**

**CrossTable(x = wbcd\_test\_labels, y = wbcd\_test\_pred, prop.chisq=FALSE)**

**wbcd\_test\_pred <- knn(train = wbcd1\_train, test = wbcd1\_test, cl = wbcd\_train\_labels, k=15)**

**CrossTable(x = wbcd\_test\_labels, y = wbcd\_test\_pred, prop.chisq=FALSE)**

**wbcd\_test\_pred <- knn(train = wbcd1\_train, test = wbcd1\_test, cl = wbcd\_train\_labels, k=21)**

**CrossTable(x = wbcd\_test\_labels, y = wbcd\_test\_pred, prop.chisq=FALSE)**

**wbcd\_test\_pred <- knn(train = wbcd1\_train, test = wbcd1\_test, cl = wbcd\_train\_labels, k=27)**

**CrossTable(x = wbcd\_test\_labels, y = wbcd\_test\_pred, prop.chisq=FALSE)**

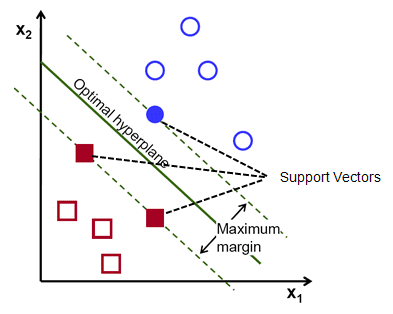
**SUPPORT VECTOR MACHINE (SVM)**

The main idea of support vector machine is to find the **optimal hyperplane** (line in 2D, plane in 3D and hyperplane in more than 3 dimensions) which **maximizes the margin between two classes**. In this case, two classes are red and blue balls. In layman's term, it is finding the optimal separating boundary to separate two classes (events and non-events).

**Support Vectors** are observations that supports hyperplane on either sides. In the image above, **filled** red and blue boxes and circles are **support vectors**.

**Hyperplane** is just a line in 2D and plane in 3D. In higher dimensions (more than 3D), it's called hyperplane. SVM help us to find a hyperplane (or separating boundary) that can separate two classes (red and blue dots).

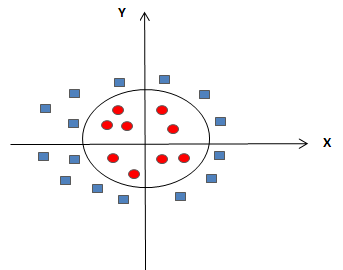
It is the distance between the hyperplane and the closest data point. If we double it, it would be equal to the margin.



**How to find the optimal hyperplane?**  
  
In your dataset, select two hyperplanes which separate the data with no points between them and maximize the distance between these two hyperplanes. The distance here is **'margin'**.

**How to treat Non-linear Separable Data?**

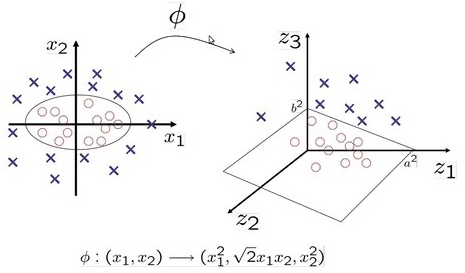
Imagine a case - if there is no straight line (or hyperplane) which can separate two classes? In the image shown below, there is a circle in 2D with red and blue data points all over it such that adjacent data points are of different colors.



SVM handles the above case by using a **kernel function** to handle non-linear separable data. It is explained in the next section.

**What is Kernel?**

In simple words, it is a method to make SVM run in case of non-linear separable data points. The kernel function transforms the data into a higher dimensional feature space to make it possible to perform the linear separation. See the image below –



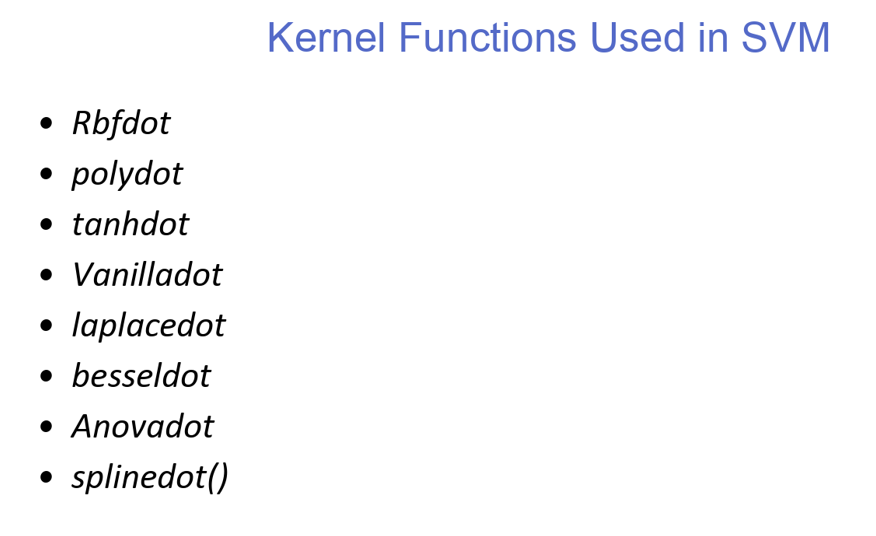
**Different Kernels**

1. **linear**: u'\*v

2. **polynomial**: (gamma\*u'\*v + coef0)^degree

3. **radial basis** (RBF) : exp(-gamma\*|u-v|^2)

4. **sigmoid**: tanh(gamma\*u'\*v + coef0)



**Load R Packages**  
  
Make sure you have the following packages already installed. If not, install them by using install.packages() function.

library(caret)  
library(kernlab)  
library(ROCR)

**# take the default data from R**

**data()**

**data("iris")**

**View(iris)**

**# Syntax for SVM model and applying SVM data on IRIS Dataset**

**install.packages('e1071', dependencies=TRUE)**

**library(e1071)**

**#build the SVM default method and as default it will excute Kernel of Radial Method**

**final\_svm <- svm(Species ~ ., data=iris)**

**summary(final\_svm)**

**# prediction of Default Radial method**

**pred <- predict(final\_svm,iris)**

**tab <-table(predicted = pred,Actual = iris$Species)**

**sum(diag(tab))/sum(tab)**

**# Build the model using linear methods with cost and Gamma use linear and radial as Linear methods**

**svm\_linear <- svm(Species ~ ., data=iris,kernel = "linear")**

**summary(svm\_linear)**

**# prediction of Linear Method**

**pred <- predict(svm\_linear,iris)**

**tab <-table(predicted = pred,Actual = iris$Species)**

**sum(diag(tab))/sum(tab)**

**## SVM Methods Using NON LINEAR METHODS**

**# For Non Linear Methods we user vaniladot , polydot,rbfdot without cost and gamma values**

**library(kernlab)**

**svm\_nonlinear <- ksvm(Species ~ ., data = iris, kernel = "rbfdot")**

**prediction <- predict(svm\_nonlinear,iris)**

**table(prediction)**

**prop.table(table(prediction))**

**tab <-table(prediction = prediction,Actual = iris$Species)**

**tab**

**sum(diag(tab))/sum(tab)**

**# Using polydot**

**svm\_nonlinear <- ksvm(Species ~ ., data = iris, kernel = "polydot")**

**prediction <- predict(svm\_nonlinear,iris)**

**table(prediction)**

**prop.table(table(prediction))**

**tab <-table(prediction = prediction,Actual = iris$Species)**

**tab**

**sum(diag(tab))/sum(tab)**

**# Using vanilladot**

**svm\_nonlinear <- ksvm(Species ~ ., data = iris, kernel = "vanilladot")**

**prediction <- predict(svm\_nonlinear,iris)**

**table(prediction)**

**prop.table(table(prediction))**

**tab <-table(prediction = prediction,Actual = iris$Species)**

**tab**

**sum(diag(tab))/sum(tab)**

**NAÏVE BAYES THEOREM**

Naive Bayes is a machine learning algorithm for classification problems. It is based on Bayes’ probability theorem. It is primarily used for text classification which involves high dimensional training data sets. A few examples are spam filtration, sentimental analysis, and classifying news articles.

It is not only known for its simplicity, but also for its effectiveness. It is fast to build models and make predictions with Naive Bayes algorithm. Naive Bayes is the first algorithm that should be considered for solving text classification problem. Hence, you should learn this algorithm thoroughly.

P(A|B)=P(B|A)P(A)/P(B)

Let us understand the statement first and then we will look at the proof of the statement. The components of the above statement are:

* P(A|B): Probability (conditional probability) of occurrence of event A  given the event Bis true
* P(A) and P(B): Probabilities of the occurrence of event A and B respectively
* P(B|A): Probability of the occurrence of event B  given the event A is true

The terminology in the Bayesian method of probability (more commonly used) is as follows:

* A is called the **proposition** and B is called the **evidence.**
* P(A) is called the **prior**probability of proposition and P(B) is called the **prior**probability of evidence.
* P(A|B) is called the **posterior.**
* P(B|A) is the **likelihood**.

**Applications**

The Naive Bayes algorithm is used in multiple real-life scenarios such as

1. **Text classification**: It is used as a probabilistic learning method for text classification. The Naive Bayes classifier is one of the most successful known algorithms when it comes to the classification of text documents, i.e., whether a text document belongs to one or more categories (classes).
2. **Spam filtration**: It is an example of text classification. This has become a popular mechanism to distinguish spam email from legitimate email. Several modern email services implement Bayesian spam filtering.  
   Many server-side email filters, such as DSPAM, SpamBayes, SpamAssassin, Bogofilter, and ASSP, use this technique.
3. **Sentiment Analysis**: It can be used to analyze the tone of tweets, comments, and reviews—whether they are negative, positive or neutral.
4. **Recommendation System**: The Naive Bayes algorithm in combination with collaborative filtering is used to build hybrid recommendation systems which help in predicting if a user would like a given resource or not.

**# we load mlbench, fetch the dataset**

library(mlbench)

**# uploading housevotes dataset**

**data()**

**data("HouseVotes84")**

**View(HouseVotes84)**

**mydata<-na.omit(HouseVotes84) # remove missing data vlaues**

**View(mydata)**

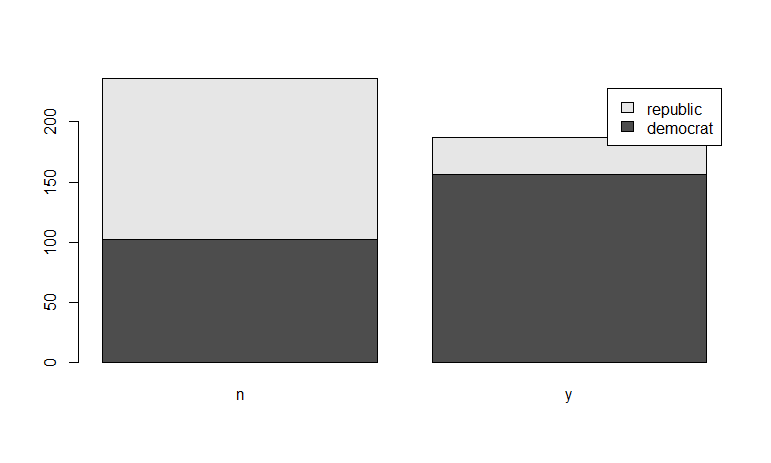
**help("HouseVotes84")**

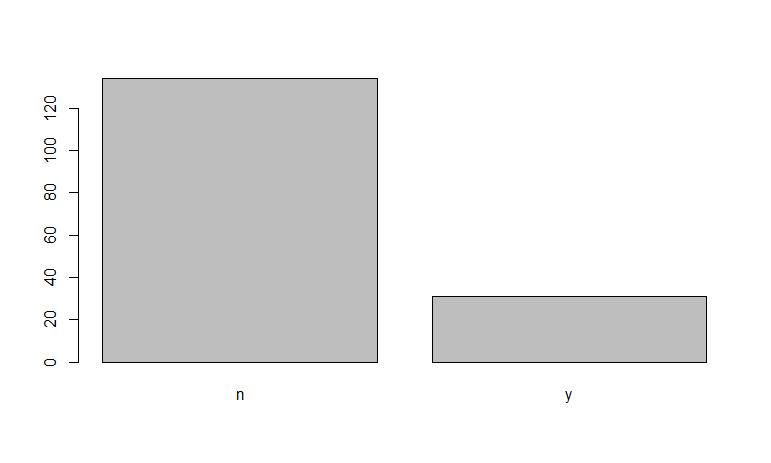
**# Exploratory Data Analysis**

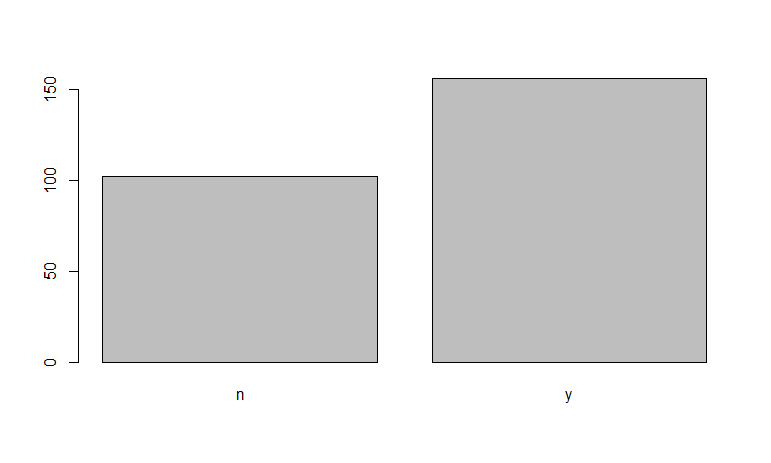
**summary(HouseVotes84)**

**summary(mydata)**

**barplot(table(as.factor(HouseVotes84[,1]),as.factor(HouseVotes84[,2])),legend=c("democrat","republic"))**

**plot(as.factor(HouseVotes84[HouseVotes84$Class=="republican",2]))**

**plot(as.factor(HouseVotes84[HouseVotes84$Class=="democrat",2]))**



**str(HouseVotes84)**

**str(mydata)**

#train\_x<-HouseVotes84[,c()]

# imputing missing values

# We do this by randomly assigning values ( y or n) to NAs, based on the proportion of members

#of a party who have voted y or n

#na.omit(train)

#newdata <- na.omit(is.null(NA))

#View(newdata)

**set.seed(3)**

**train<-order(runif(290))**

**test<--train**

**View(train)**

**View(test)**

**# Divide the data into Training and Testing Data**

**training<-HouseVotes84[train,]**

**View(training)**

**testing<-HouseVotes84[test,]**

**View(testing)**

**# Building the naive Baye model**

**library(e1071)**

**model<-naiveBayes(training$Class~.,data=training)**

**model**

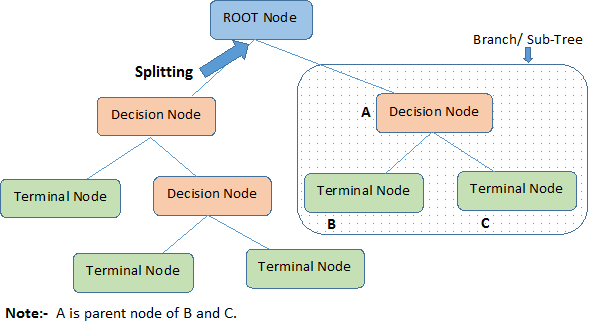
**# Predicting the model on Test data**

**pred<-predict(model,newdata = testing[,-1])**

**mean(pred==testing[,1])**

**DECISION TREE**

Decision tree is a type of supervised learning algorithm that can be used in both regression and classification problems. It works for both categorical and continuous input and output variables.



Let's identify important terminologies on Decision Tree, looking at the image above:

* **Root Node** represents the entire population or sample. It further gets divided into two or more homogeneous sets.
* **Splitting** is a process of dividing a node into two or more sub-nodes.
* When a sub-node splits into further sub-nodes, it is called a **Decision Node**.
* Nodes that do not split is called a **Terminal Node** or a **Leaf**.
* When you remove sub-nodes of a decision node, this process is called **Pruning**. The opposite of pruning is **Splitting**.
* A sub-section of an entire tree is called **Branch**.
* A node, which is divided into sub-nodes is called a **parent node** of the sub-nodes; whereas the sub-nodes are called the **child** of the parent node.

The algorithm of the decision tree models works by repeatedly partitioning the data into multiple sub-spaces, so that the outcomes in each final sub-space is as homogeneous as possible. This approach is technically called *recursive partitioning*. The produced result consists of a set of rules used for predicting the outcome variable, which can be either:

* a continuous variable, for regression trees
* a categorical variable, for classification trees

The decision rules generated by the CART (Classification & Regression Trees) predictive model are generally visualized as a binary tree.

**# we neeed to install C50 package to use**

**install.packages("C50")**

**library(C50)**

**install.packages("tree")**

**library(tree)**

**data()**

**data("iris")**

**View(iris)**

**?iris** # Help from IRIS

**# Splitting data into training and testing. As the species are in order**

**# splitting the data based on species**

**iris\_setosa<-iris[iris$Species=="setosa",] # devide the data based on setosa**

**iris\_versicolor <- iris[iris$Species=="versicolor",] # devide the data based on versicolor**

**iris\_virginica <- iris[iris$Species=="virginica",] # devide the data based on virginica**

**iris\_train <- rbind(iris\_setosa[1:25,],iris\_versicolor[1:25,],iris\_virginica[1:25,])**

**iris\_test <- rbind(iris\_setosa[26:50,],iris\_versicolor[26:50,],iris\_virginica[26:50,])**

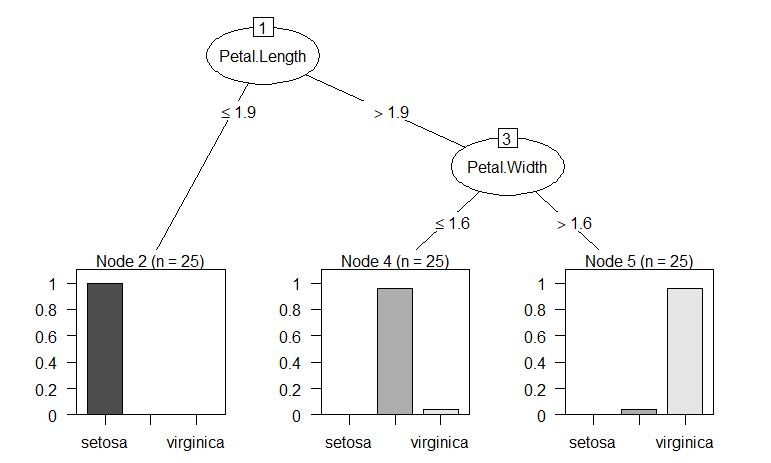
**View(iris\_train)**

**View(iris\_test)**

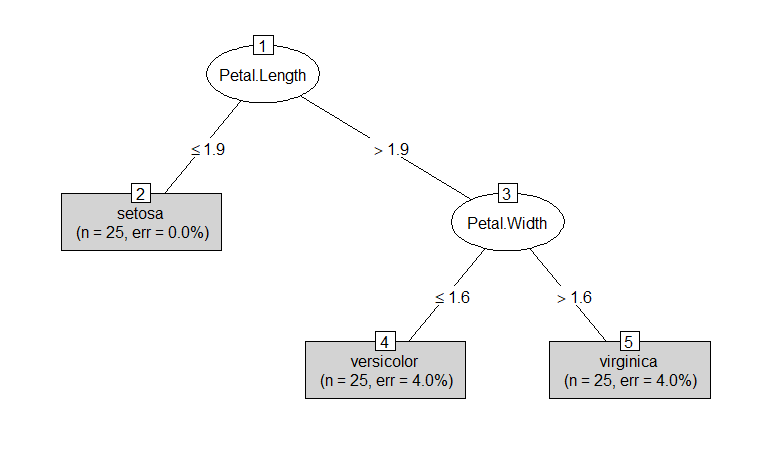
**# Building model on training data**

**irisc5.0\_train <- C5.0(iris\_train[,-5],iris\_train$Species)**

**plot(irisc5.0\_train) # Tree graph**



**plot(irisc5.0\_train,type="simple")**



**# Training accuracy**

**mean(iris\_train$Species==predict(irisc5.0\_train,iris\_train)) # 97.33% Accuracy**

**predc5.0\_test <- predict(irisc5.0\_train,newdata=iris\_test) # predicting on test data**

**mean(predc5.0\_test==iris\_test$Species) # 94.66% accuracy**

**library(gmodels)**

**# Cross tablez**

**CrossTable(iris\_test$Species,predc5.0\_test)**

**RANDOM FOREST**

Random Forest is one of the most versatile machine learning algorithms available today. With its built-in ensembling capacity, the task of building a decent generalized model (on any dataset) gets much easier. However, I've seen people using random forest as a black box model; i.e., they don't understand what's happening beneath the code. They just code.

In fact, the easiest part of machine learning is *coding*. If you are new to machine learning, the random forest algorithm should be on your tips. Its ability to solve—both regression and classification problems along with robustness to correlated features and variable importance plot gives us enough head start to solve various problems.

Random forest is a tree-based algorithm which involves building several trees (decision trees), then combining their output to improve generalization ability of the model. The method of combining trees is known as an ensemble method. Ensembling is nothing but a combination of weak learners (individual trees) to produce a strong learner.

**# Using Random Forest**

**install.packages("randomForest")**

**library(randomForest)**

**data()**

**data(iris)**

**View(iris)**

**# Splitting data into training and testing. As the species are in order**

**# splitting the data based on species**

**iris\_setosa<-iris[iris$Species=="setosa",]** # divide the data based on setosa

**View(iris\_setosa)**

**iris\_versicolor <- iris[iris$Species=="versicolor",]** # divide the data based on versicolor

**View(iris\_versicolor)**

**iris\_virginica <- iris[iris$Species=="virginica",]** # divide the data based on virginica

**View(iris\_virginica)**

**iris\_train <- rbind(iris\_setosa[1:25,],iris\_versicolor[1:25,],iris\_virginica[1:25,])**

**View(iris\_train)**

**iris\_test <- rbind(iris\_setosa[26:50,],iris\_versicolor[26:50,],iris\_virginica[26:50,])**

**View(iris\_test)**

**# Building a random forest model on training data**

**model1 <- randomForest(Species~.,data=iris\_train)**

**model1**

# Output show

# Number of Trees is 500 , Number of variables tried at each spilit is 2

# Error Rate is 4 %

**# Fine tuning parameters of Random Forest model**

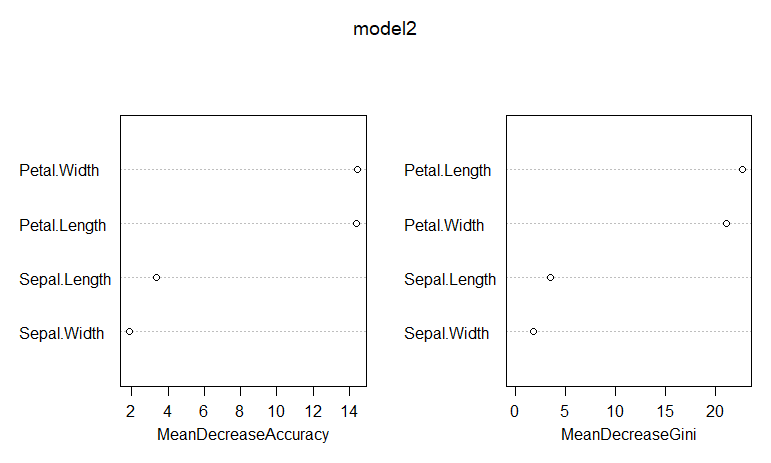
**model2 <- randomForest(Species~.,data=iris\_train, ntree = 100, importance = TRUE)**

**model2**

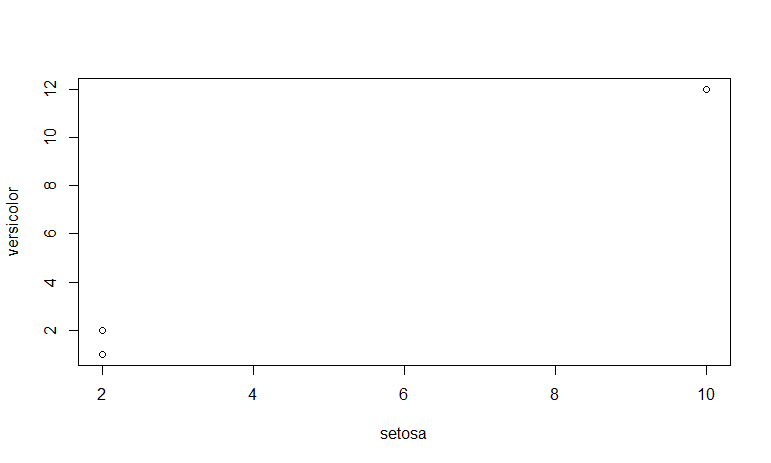
**#To check Importance variables**

**importance(model2)**

**varImpPlot(model2)**



**plot(round(importance(model2)))**



**?randomeforest** # Help for random froest

**table(predict(model1),iris\_train$Species)**

**# Predicting test data**

**pred\_test <- predict(model1,newdata=iris\_test)**

**table(pred\_test,iris\_test$Species)**

**#Checking the accuracy by creating confusion matrix**

**CM <- table(pred\_test,iris\_test$Species)**

**CM**

**accuracy <- (sum(diag(CM)))/sum(CM)**

**accuracy**

**# predicting for Model2**

**pred\_test <- predict(model2,newdata=iris\_test)**

**table(pred\_test,iris\_test$Species)**

**#Checking the accuracy by creating confusion matrix**

**CM <- table(pred\_test,iris\_test$Species)**

**CM**

**accuracy <- (sum(diag(CM)))/sum(CM)**

**accuracy**

**PACKAGES**

## To load data

[DBI](https://db.rstudio.com/dbi/) - The standard for for communication between R and relational database management systems. Packages that connect R to databases depend on the DBI package.

[odbc](https://db.rstudio.com/odbc/) - Use any ODBC driver with the odbc package to connect R to your database. Note: RStudio professional products come with [professional drivers](https://www.rstudio.com/products/drivers/) for some of the most popular databases.

[RMySQL](http://www.rdocumentation.org/packages/RMySQL/functions/RMySQL-package), [RPostgresSQL](http://www.rdocumentation.org/packages/RPostgresSQL), [RSQLite](http://www.rdocumentation.org/packages/RSQLite) - If you'd like to read in data from a database, these packages are a good place to start. Choose the package that fits your type of database.

[XLConnect](https://cran.rstudio.com/web/packages/XLConnect), [xlsx](https://cran.rstudio.com/web/packages/xlsx) - These packages help you read and write Micorsoft Excel files from R. You can also just export your spreadsheets from Excel as .csv's.

[foreign](http://www.rdocumentation.org/packages/foreign) - Want to read a SAS data set into R? Or an SPSS data set? Foreign provides functions that help you load data files from other programs into R.

[haven](https://haven.tidyverse.org/) - Enables R to read and write data from SAS, SPSS, and Stata.

R can handle plain text files – no package required. Just use the functions read.csv, read.table, and read.fwf. If you have even more exotic data, consult the CRAN [guide](https://cran.rstudio.com/doc/manuals/R-data.pdf) to data import and export.

For more information about using R with databases see [*db.rstudio.com*](https://db.rstudio.com/).

## To manipulate data

[dplyr](http://blog.rstudio.org/2014/01/17/introducing-dplyr/) - Essential shortcuts for subsetting, summarizing, rearranging, and joining together data sets. dplyr is our go to package for fast data manipulation.

[tidyr](http://blog.rstudio.org/2014/07/22/introducing-tidyr/) - Tools for changing the layout of your data sets. Use the gather and spread functions to convert your data into the [tidy format](http://www.jstatsoft.org/v59/i10/paper), the layout R likes best.

[stringr](http://journal.r-project.org/archive/2010-2/RJournal_2010-2_Wickham.pdf) - Easy to learn tools for regular expressions and character strings.

[lubridate](http://www.r-statistics.com/2012/03/do-more-with-dates-and-times-in-r-with-lubridate-1-1-0/) - Tools that make working with dates and times easier.

## To visualize data

[ggplot2](http://docs.ggplot2.org/current/) - R's famous package for making beautiful graphics. ggplot2 lets you use the [grammar of graphics](http://vita.had.co.nz/papers/layered-grammar.pdf) to build layered, customizable plots.

[ggvis](http://ggvis.rstudio.com/) - Interactive, web based graphics built with the grammar of graphics.

[rgl](http://rgl.neoscientists.org/about.shtml) - Interactive 3D visualizations with R

[htmlwidgets](http://www.htmlwidgets.org/) - A fast way to build interactive (javascript based) visualizations with R. Packages that implement htmlwidgets include:

* [leaflet](http://rstudio.github.io/leaflet/) (maps)
* [dygraphs](http://rstudio.github.io/dygraphs) (time series)
* [DT](http://rstudio.github.io/DT/) (tables)
* [diagrammeR](http://rich-iannone.github.io/DiagrammeR/) (diagrams)
* [network3D](http://christophergandrud.github.io/networkD3/) (network graphs)
* [threeJS](https://github.com/bwlewis/rthreejs) (3D scatterplots and globes).

[googleVis](https://cran.rstudio.com/web/packages/googleVis) - Let's you use Google Chart tools to visualize data in R. Google Chart tools used to be called Gapminder, the graphing software Hans Rosling made famous in hie TED talk.

## To model data

[car](http://www.rdocumentation.org/packages/car) - car's [Anova](http://www.rdocumentation.org/packages/car/functions/Anova) function is popular for making type II and type III Anova tables.

[mgcv](http://www.rdocumentation.org/packages/mgcv/functions/mgcv-package) - Generalized Additive Models

[lme4](http://www.rdocumentation.org/packages/lme4/functions/lme4-package)/[nlme](http://www.rdocumentation.org/packages/nlme/functions/nlme) - Linear and Non-linear mixed effects models

[randomForest](http://www.rdocumentation.org/packages/randomForest/functions/randomForest) - Random forest methods from machine learning

[multcomp](http://www.rdocumentation.org/packages/multcomp) - Tools for multiple comparison testing

[vcd](http://www.rdocumentation.org/packages/vcd) - Visualization tools and tests for categorical data

[glmnet](http://www.rdocumentation.org/packages/glmnet/functions/glmnet) - Lasso and elastic-net regression methods with cross validation

[survival](http://www.rdocumentation.org/packages/survival) - Tools for survival analysis

[caret](https://cran.rstudio.com/web/packages/caret) - Tools for training regression and classification models

## To report results

[shiny](http://shiny.rstudio.com/) - Easily make interactive, web apps with R. A perfect way to explore data and share findings with non-programmers.

[R Markdown](https://rmarkdown.rstudio.com/) - The perfect workflow for reproducible reporting. Write R code in your [markdown](http://daringfireball.net/projects/markdown/) reports. When you run render, R Markdown will replace the code with its results and then export your report as an HTML, pdf, or MS Word document, or a HTML or pdf slideshow. The result? Automated reporting. R Markdown is integrated straight into RStudio.

[xtable](http://www.rdocumentation.org/packages/xtable/functions/xtable) - The [xtable](http://www.rdocumentation.org/packages/xtable/functions/xtable) function takes an R object (like a data frame) and returns the latex or HTML code you need to paste a pretty version of the object into your documents. Copy and paste, or pair up with R Markdown.

## For Spatial data

[sp](https://cran.rstudio.com/web/packages/sp), [maptools](http://www.rdocumentation.org/packages/maptools) - Tools for loading and using spatial data including shapefiles.

[maps](http://www.rdocumentation.org/packages/maps) - Easy to use map polygons for plots.

[ggmap](http://journal.r-project.org/archive/2013-1/kahle-wickham.pdf) - Download street maps straight from Google maps and use them as a background in your ggplots.

## For Time Series and Financial data

[zoo](https://cran.rstudio.com/web/packages/zoo) - Provides the most popular format for saving time series objects in R.

[xts](https://cran.rstudio.com/web/packages/xts) - Very flexible tools for manipulating time series data sets.

[quantmod](http://www.quantmod.com/) - Tools for downloading financial data, plotting common charts, and doing technical analysis.

## To write high performance R code

[Rcpp](http://dirk.eddelbuettel.com/code/rcpp.html) - Write R functions that call C++ code for lightning fast speed.

[data.table](http://www.rdocumentation.org/packages/data.table) - An alternative way to organize data sets for very, very fast operations. Useful for big data.

[parallel](http://www.rdocumentation.org/packages/parallel/functions/parallel-package) - Use parallel processing in R to speed up your code or to crunch large data sets.

## To work with the web

[XML](http://www.omegahat.org/RSXML/shortIntro.pdf) - Read and create XML documents with R

[jsonlite](https://cran.rstudio.com/web/packages/jsonlite) - Read and create JSON data tables with R

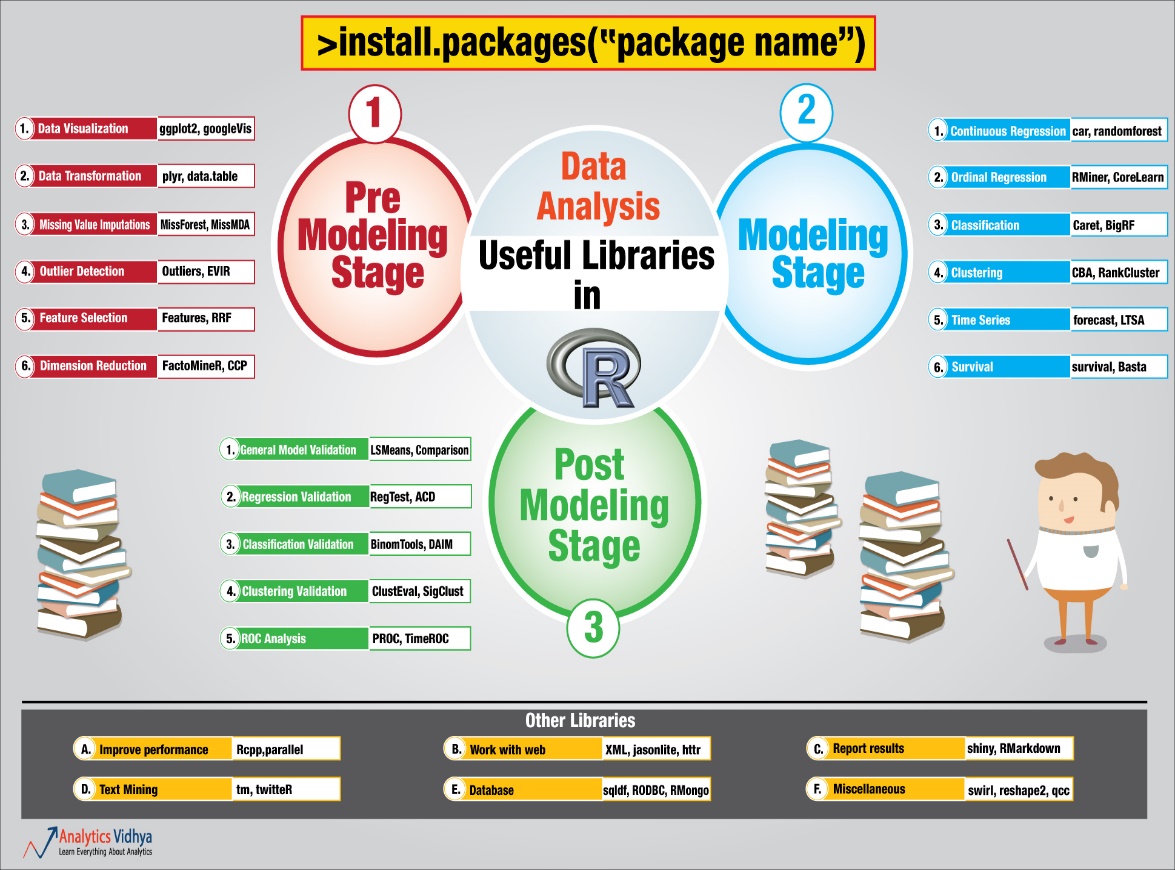
[httr](http://www.rdocumentation.org/packages/httr/functions/httr) - A set of useful tools for working with http connections

## To write your own R packages

[devtools](https://www.rstudio.com/projects/devtools/) - An essential suite of tools for turning your code into an R package.

[testthat](http://journal.r-project.org/archive/2011-1/RJournal_2011-1_Wickham.pdf) - testthat provides an easy way to write unit tests for your code projects.

[roxygen2](http://www.rdocumentation.org/packages/roxygen2) - A quick way to document your R packages. roxygen2 turns inline code comments into documentation pages and builds a package namespace.



#### **My favorite R packages for data visualization and munging**

| **PACKAGPdkageE** | **CATEGORY** | **DESCRIPTIONsummary** | **SAMPLE USE** | **AUTHOR** |
| --- | --- | --- | --- | --- |
| [dplyr](http://cran.rstudio.com/web/packages/dplyr/vignettes/introduction.html) | data wrangling, data analysis | The essential data-munging R package when working with data frames. Especially useful for operating on data by categories. CRAN. | [See the intro vignette](http://cran.rstudio.com/web/packages/dplyr/vignettes/introduction.html) | Hadley Wickham |
| [purrr](http://purrr.tidyverse.org/) | data wrangling | purrr makes it easy to apply a function to each item in a list and return results in the format of your choice. It's more complex to learn than the older [plyr](http://plyr.had.co.nz/) package, but also more robust. And, its functions are more standardized than base R's apply family -- plus it's got functions for tasks like error-checking. CRAN. | map\_df(mylist, myfunction) More: [Charlotte Wickham's purr tutorial video](https://www.rstudio.com/resources/videos/happy-r-users-purrr-tutorial/), the [purrr cheat sheet](https://github.com/rstudio/cheatsheets/raw/master/purrr.pdf) PDF download. | Hadley Wickham |
| [readxl](https://github.com/hadley/readxl/) | data import | Fast way to read Excel files in R, without dependencies such as Java. CRAN. | read\_excel("my-spreadsheet.xls", sheet = 1) | Hadley Wickham |
| [googlesheets](https://github.com/jennybc/googlesheets) | data import, data export | Easily read data from and post data to Google Sheets. While no longer under active development (it will be replaced by the googledrive and googlesheets4 packages), I find the package still works well. CRAN. | mysheet <- gs\_title("Google Spreadsheet Title") mydata <- mydata <- gs\_read(mysheet, ws = “WorksheetTitle”) | Jennifer Bryan |
| [readr](https://github.com/hadley/readr) and [vroom](https://www.tidyverse.org/articles/2019/05/vroom-1-0-0/) | data import | Base R handles most of these functions; but if you have huge files, these packages offer faster and standardized way to read CSVs and similar files into R. readr has been around for awhile; vroom is a speedier alternative, useful for larger data sets. Eventually the packages will likely merge. CRAN. | read\_csv(myfile.csv) or vroom(myfile.csv) | Hadley Wickham (readr), Jim Hester (vroom) |
| [rio](http://cran.r-project.org/web/packages/rio/vignettes/rio.html) | data import, data export | rio has a good idea: Pull a lot of separate data-reading packages into one, so you just need to remember 2 functions: import and export. CRAN. | import("myfile") | Thomas J. Leeper & others |
| [tidyxl](https://nacnudus.github.io/tidyxl/) | data import, data wrangling | If you've ever wanted to tear your hair out over an Excel file with merged cells, data in column headers, headers mixed in data, and key information in color coding, this is the package for you. Each cell is imported in its own row, with information about data type, position, and color, not just value, allowing you to reshape the data from there. Super time saver for messy data. CRAN. | xlsx\_cells("my\_nightmare\_file.xlsx") | Duncan Garmonsway |
| [Hmisc](http://biostat.mc.vanderbilt.edu/wiki/Main/HmiscNew) | data analysis | There are a number of useful functions in here. Two of my favorites: describe, a more robust summary function, and Cs, which creates a vector of quoted character strings from unquoted comma-separated text. Cs(so, it, goes) creates c("so", "it", "goes"). CRAN. | describe(mydf) Cs(so, it, goes) | Frank E Harrell Jr & others |
| [datapasta](https://github.com/MilesMcBain/datapasta) | data import | Data copy and paste: Meet reproducible research. If you've copied data from the Web, a spreadsheet, or other source into your clipboard, datapasta lets you paste it into R as an R object, with the code to reproduce it. It includes RStudio add-ins as well as command-line functions for transposing data, turning it into markdown format, and more. CRAN. | df\_paste() to create a data frame, vector\_paste() to create a vector. | Miles McBain |
| [sqldf](https://github.com/ggrothendieck/sqldf) | data wrangling, data analysis | Do you know a great SQL query you'd use if your R data frame were in a SQL database? Run SQL queries on your data frame with sqldf. CRAN. | sqldf("select \* from mydf where mycol > 4") | G. Grothendieck |
| [jsonlite](https://www.opencpu.org/posts/jsonlite-a-smarter-json-encoder/) | data import, data wrangling | Parse json within R or turn R data frames into json. CRAN. | myjson <- toJSON(mydf, pretty=TRUE) mydf2 <- fromJSON(myjson) | Jeroen Ooms & others |
| [XML](http://cran.r-project.org/web/packages/XML/) | data import, data wrangling | Many functions for elegantly dealing with XML and HTML, such as readHTMLTable. CRAN. | mytables <- readHTMLTable(myurl) | Duncan Temple Lang |
| [httr](https://github.com/hadley/httr) | data import, data wrangling | An R interface to http protocols; useful for pulling data from APIs. See the [httr quickstart guide](https://cran.r-project.org/web/packages/httr/vignettes/quickstart.html). CRAN. | r <- GET("http://httpbin.org/get") content(r, "text") | Hadley Wickham |
| [quantmod](http://www.quantmod.com/examples/intro/) | data import, data visualization, data analysis | Even if you're not interested in analyzing and graphing financial investment data, quantmod has easy-to-use functions for importing economic as well as financial data from sources like the Federal Reserve. CRAN. | getSymbols("AITINO", src="FRED") | Jeffrey A. Ryan |
| [tidyquant](https://business-science.github.io/tidyquant/) | data import, data visualization, data analysis | Another financial package that's useful for importing, analyzing and visualizing data, integrating aspects of other popular finance packages as well as tidyverse tools. With thorough documentation. CRAN. | aapl\_key\_ratios <- tq\_get("AAPL", get = "key.ratios") | Matt Dancho |
| [rvest](http://cran.r-project.org/web/packages/rvest/vignettes/selectorgadget.html) | data import, web scraping | Web scraping: Extract data from HTML pages. Inspired by Python's Beautiful Soup. Works well with Selectorgadget. CRAN. | [See the package vignette](http://cran.r-project.org/web/packages/rvest/vignettes/selectorgadget.html) | Hadley Wickham |
| [tidyr](https://github.com/hadley/tidyr) | data wrangling | tidyr initially won me over with specialized functions like fill (fill in missing columns from data above) and replace\_na. But now I also use it for its main purpose too: helping you change data row and column formats from "wide" to "long". CRAN. | See my step-by-step instructions and video in [R tip: Reshape data with tidyr](https://www.infoworld.com/article/3310078/r-language/reshape-data-in-r-with-the-tidyr-package.html). | Hadley Wickham |
| [splitstackshape](https://github.com/mrdwab/splitstackshape) | data wrangling | It's rare that I'd recommend a package that hasn't been updated in years, but the cSplit() function solves a rather complex shaping problem in an astonishingly easy way. If you have a data frame column with one or more comma-separated values (think a survey question with "select all that apply"), this is worth an install if you want to separate each item into its own new data frame row.. CRAN. | cSplit(mydata, "multi\_val\_column", sep = ",", direction = "long"). | Ananda Mahto |
| [magrittr](https://github.com/tidyverse/magrittr) | data wrangling | This package gave us the %>% symbol for chaining R operations, but it's got other useful operators such as %<>% for mutating a data frame in place and and . as a placeholder for the original object being operated upon. CRAN. | mydf %<>% mutate(newcol = myfun(colname)) | Stefan Milton Bache & Hadley Wickham |
| [validate](https://github.com/data-cleaning/validate) | data wrangling | Intuitive data validation based on rules you can define, save and re-use. CRAN. | See the [introductory vignette](https://cran.r-project.org/web/packages/validate/vignettes/intro.html). | Mark van der Loo & Edwin de Jonge |
| [testthat](https://github.com/hadley/testthat) | programming | Package that makes it easy to write unit tests for your R code. CRAN. | See the [testing chapter](http://r-pkgs.had.co.nz/tests.html) of Hadley Wickham's book on R packages. | Hadley Wickham |
| [data.table](https://github.com/Rdatatable/data.table/wiki/Getting-started) | data wrangling, data analysis | Popular package for heavy-duty data wrangling. While I typically prefer dplyr, data.table has many fans for its speed with large data sets. CRAN. | [Useful tutorial](http://blog.yhathq.com/posts/fast-summary-statistics-with-data-dot-table.html) | Matt Dowle & others |
| [stringr](https://github.com/hadley/stringr) | data wrangling | Numerous functions for text manipulation. Some are similar to existing base R functions but in a more standard format, including working with regular expressions. Some of my favorites: str\_pad and str\_trim. CRAN. | str\_pad(myzipcodevector, 5, "left", "0") | Hadley Wickham |
| [lubridate](https://github.com/hadley/lubridate) | data wrangling | Everything you ever wanted to do with date arithmetic, although understanding & using available functionality can be somewhat complex. CRAN. | mdy("05/06/2015") + months(1) [More examples in the package vignette](http://cran.r-project.org/web/packages/lubridate/vignettes/lubridate.html) | Garrett Grolemund, Hadley Wickham & others |
| [DataExplorer](https://boxuancui.github.io/DataExplorer/) | data analysis | Not sure where to get started looking at a data set? Want to get a basic handle on that data without running multiple commands like str() and plot()? DataExplorer attempts to offer one-click report generation to show and visualize basics about a data set, such as distributions and missing data. CRAN. | create\_report(mydataframe) | Boxuan Cui |
| [zoo](http://cran.r-project.org/web/packages/zoo/) | data wrangling, data analysis | Robust package with a slew of functions for dealing with time series data; I like the handy rollmean function with its align=right and fill=NA options for calculating moving averages. CRAN. | rollmean(mydf, 7) | Achim Zeileis & others |
| [knitr](http://yihui.name/knitr/) | data display | Add R to a markdown document and easily generate reports in HTML, Word and other formats. A must-have if you're interested in reproducible research and automating the journey from data analysis to report creation -- as is the [rmarkdown package](https://rmarkdown.rstudio.com/). CRAN. | See the [Minimal Examples](http://yihui.name/knitr/demo/minimal/) knitr page and [RStudio's R Markdown page](https://rmarkdown.rstudio.com/). | Yihui Xie & others |
| [remedy](https://thinkr-open.github.io/remedy/) | data display | RStudio add-in offers a menu for R Markdown formatting commands, so you no longer need to remember and/or type code for things like making an HTML list or embedding a YouTube video. And, since add-in commands can be assigned custom keyboard shortcuts, you can create your own shortcuts for tasks like bolding text. GitHub. | See [the package website](https://thinkr-open.github.io/remedy/). | Colin Fay & others |
| [officeR](https://github.com/davidgohel/officer) | data display | Import and edit Microsoft Word and PowerPoint documents, making it easy to add R-generated analysis and visualizations to existing as well as new reports and presentations. CRAN. | my\_doc <- read\_docx() %>%  body\_add\_img(src = myplot) [The package website](https://davidgohel.github.io/officer/index.html) has many more examples. | David Gohel |
| [listviewer](http://www.buildingwidgets.com/blog/2015/4/14/week-15-listviewer) | data display, data wrangling | While RStudio has since added a list-viewing option, this HTML widget still offers an elegant way to view complex nested lists within R. GitHub timelyportfolio/listviewer. | jsonedit(mylist) | Kent Russell |
| [DT](https://rstudio.github.io/DT/) | data display | Create a sortable, searchable table in one line of code with this R interface to the jQuery DataTables plug-in. GitHub rstudio/DT. | datatable(mydf) | RStudio |
| [ggplot2](http://ggplot2.org/) | data visualization | Powerful, flexible and well-thought-out dataviz package following 'grammar of graphics' syntax to create static graphics, but be prepared for a steep learning curve. CRAN. | qplot(factor(myfactor), data=mydf, geom="bar", fill=factor(myfactor)) See my [searchable ggplot2 cheat sheet](http://www.computerworld.com/article/2935394/business-intelligence/my-ggplot2-cheat-sheet-search-by-task.html) and [time-saving code snippets](http://www.computerworld.com/article/2936729/data-analytics/free-download-save-r-data-visualization-time-with-these-ggplot2-code-snippets.html). | Hadley Wickham |
| [patchwork](https://github.com/thomasp85/patchwork) | data visualization | Easily combine ggplot2 plots and keep the new, merged plot a ggplot2 object. plot\_layout() adds ability to set columns, rows, and relative sizes of each component graphic. GitHub. | plot1 + plot2 + plot\_layout(ncol=1) | Thomas Lin Pedersen |
| [ggiraph](http://davidgohel.github.io/ggiraph/) | data visualization | Make ggplot2 plots interactive with this extension's new geom functions such geom\_bar\_interactive and arguments for tooltips and JavaScript onclicks. CRAN. | g <- ggplot(mpg, aes( x = displ, y = cty, color = drv) ) my\_gg <- g + geom\_point\_interactive(aes(tooltip = model), size = 2) ggiraph(code = print(my\_gg), width = .7). | David Gohel |
| [esquisse](https://github.com/dreamRs/esquisse) | data visualization | This RStudio add-in offers a drag-and-drop interface for ggplot2. And, it generates codes for the graph you create with the GUI. It's a useful tool for exploring different color palettes and themes, even if you're comfortable creating your visualizations directly in R. CRAN. | See examples on the [project's website](https://github.com/dreamRs/esquisse) . | Victor Perrier and Fanny Meyer, dreamRs |
| [dygraphs](https://rstudio.github.io/dygraphs/index.html) | data visualization | Create HTML/JavaScript graphs of time series - one-line command if your data is an xts object. CRAN. | dygraph(myxtsobject) | JJ Allaire & RStudio |
| [googleVis](https://github.com/mages/googleVis) | data visualization | Tap into the Google Charts API using R. CRAN. | mychart <- gvisColumnChart(mydata) plot(Column) [Numerous examples here](http://cran.r-project.org/web/packages/googleVis/vignettes/googleVis_examples.html) | Markus Gesmann & others |
| [metricsgraphics](http://hrbrmstr.github.io/metricsgraphics/) | data visualization | R interface to the metricsgraphics JavaScript library for bare-bones line, scatterplot and bar charts. GitHub hrbrmstr/metricsgraphics. | [See package intro](http://hrbrmstr.github.io/metricsgraphics/) | Bob Rudis |
| [taucharts](https://github.com/hrbrmstr/taucharts) | data visualization | This html widget library is especially useful for scatterplots where you want to view multiple regression options. However, it does much more than that, including line and bar charts with legends and tooltips. GitHub hrbrmstr/taucharts. | [See the author's post on RPubs](http://rpubs.com/hrbrmstr/taucharts) | Bob Rudis |
| [RColorBrewer](http://cran.r-project.org/web/packages/RColorBrewer/index.html) | data visualization | Not a designer? RColorBrewer helps you select color palettes for your visualizations. CRAN.   Note: For even more palettes, check out packages [viridis](https://cran.r-project.org/web/packages/viridis/vignettes/intro-to-viridis.html) for colors that print well in greyscale and are easier to read if you're color blind, [pals](https://github.com/kwstat/pals), [rcartcolor](https://github.com/Nowosad/rcartocolor) for map colors, [colorr](https://github.com/cdcrabtree/colorr) for sports-team colors, [nord](https://github.com/jkaupp/nord) for "Northern-themed Color palettes," and [wesanderson](https://github.com/karthik/wesanderson) for color schemes used by director Wes Anderson. | [See Jennifer Bryan's tutorial](http://www.stat.ubc.ca/~jenny/STAT545A/block14_colors.html#rcolorbrewer) | Erich Neuwirth |
| [sf](https://github.com/r-spatial/sf/) | mapping, data wrangling | This package makes it much easier to do GIS work in R. Simple features protocols make geospatial data look a lot like regular data frames, while various functions allow for analysis such as determining whether points are in a polygons. A GIS game-changer for R. CRAN. | See the package vignettes, starting with the introduction, [Simple Features for R](https://r-spatial.github.io/sf/articles/sf1.html). | Edzer Pebesma & others |
| [leaflet](http://rstudio.github.io/leaflet/) | mapping | Map data using the Leaflet JavaScript library within R. GitHub rstudio/leaflet. | [See my tutorial](http://www.computerworld.com/article/2893271/5-data-visualizations-in-5-minutes-each-in-5-lines-or-less-of-r.html) | RStudio |
| [ggmap](https://github.com/dkahle/ggmap) | mapping | Although I don't use this package often for its main purpose of pulling down background map tiles, it's my go-to for geocoding up to 2,500 addresses with the Google Maps API with its geocode and mutate\_geocode functions. CRAN. | geocode("492 Old Connecticut Path, Framingham, MA") | David Kahle &Hadley Wickham |
| [rgeocodio](https://github.com/hrbrmstr/rgeocodio) | mapping | This is a useful geocoding alternative, especially when ggmap generates messages that you're over your Google Maps API quota when you're not. It uses the [geocod.io service](https://www.geocod.io/). An API key is needed, but you can get one free that includes 2,500 lookups a day. GitHub hrbrmstr/rgeocodio. | gio\_geocode("492 Old Connecticut Path, Framingham, MA") | Bob Rudis |
| [tmap & tmaptools](http://cran.r-project.org/web/packages/tmap/vignettes/tmap-nutshell.html) | mapping | This package offer an easy way to read in shape files and join data files with geographic info, as well as do some exploratory mapping. Recent functionality adds support for simple features, interactive maps and creating leaflet objects. Plus, tmaptools::palette\_explorer() is a great tool for picking ColorBrewer palettes. CRAN. | [See the package vignette](http://cran.r-project.org/web/packages/tmap/vignettes/tmap-nutshell.html) or my [mapping in R tutorial](http://www.computerworld.com/article/3038270/data-analytics/create-maps-in-r-in-10-fairly-easy-steps.html) | Martijn Tennekes |
| [colourpicker](https://github.com/daattali/colourpicker) | data visualization | The package's RStudio add-in makes it easy to browse through and select R's built-in colors, or get hex codes for custom colors not available by name. The plotHelper() function lets you select colors and see how they'd look on a scatter plot. CRAN. | [See the GitHub repo](https://github.com/daattali/colourpicker). | Dean Attali |
| [mapsapi](https://cran.r-project.org/web/packages/mapsapi/vignettes/intro.html) | mapping, data wrangling | This interface to the Google Maps Direction and Distance Matrix APIs let you analyze and map distances and driving routes. CRAN. | google\_directions( origin = c(my\_longitude, my\_latitude), destination = c(my\_address), alternatives = TRUE Also [see the vignette](https://cran.r-project.org/web/packages/mapsapi/vignettes/intro.html) | Michael Dorman |
| [tidycensus](https://github.com/walkerke/tidycensus) | mapping, data wrangling | Want to analyze and map U.S. Census Bureau data from 5-year American Community Surveys or 10-year censuses? This makes it easy to download numerical and geospatial info in R-ready format. CRAN. | See [Basic usage of tidycensus](https://walkerke.github.io/tidycensus/articles/basic-usage.html). | Kyle E. Walker |
| [glue](https://github.com/tidyverse/glue) | data wrangling | Main function, also glue, evaluates variables and R expressions within a quoted string, as long as they're enclosed by {} braces. This makes for an elegant paste() replacement. CRAN. | glue("Today is {Sys.Date()}") | Jim Hester |
| [rga](https://github.com/skardhamar/rga) | Web analytics | Use Google Analytics with R. GitHub skardhamar/rga. | [See package README file](https://github.com/skardhamar/rga) and [my tutorial](http://www.computerworld.com/article/2486018/business-intelligence-how-to-extract-custom-data-from-the-google-analytics-api.html) | Bror Skardhamar |
| [googleanalyticsR](http://code.markedmondson.me/googleAnalyticsR/) | Web analytics | Another option for using Google Analytics with R, including adding features from GA's version 4 API. Also has anti-sampling options. CRAN. | [See package website](http://code.markedmondson.me/googleAnalyticsR/index.html). | Mark Edmonson |
| [RSiteCatalyst](http://randyzwitch.com/rsitecatalyst/) | Web analytics | Use Adobe Analytics with R. GitHub randyzwitch/RSiteCatalyst. | [See intro video](http://randyzwitch.com/rsitecatalyst/) | Randy Zwitch |
| [roxygen2](https://cran.r-project.org/web/packages/roxygen2/vignettes/roxygen2.html) | package development | Useful tools for documenting functions within R packages. CRAN. | [See this short, easy-to-read blog post on writing R packages](http://hilaryparker.com/2014/04/29/writing-an-r-package-from-scratch/), as well as the [roxygen2 introductory vignette](https://cran.r-project.org/web/packages/roxygen2/vignettes/roxygen2.html). | Hadley Wickham & others |
| [shiny](http://shiny.rstudio.com/) | data visualization | Turn R data into interactive Web applications. I've seen some nice (if sometimes sluggish) apps and it's got many enthusiasts. CRAN. | [See the tutorial](http://shiny.rstudio.com/tutorial/) | RStudio |
| [flexdashboard](http://rmarkdown.rstudio.com/flexdashboard/) | data visualization | If Shiny is too complex and involved for your needs, this package offers a simpler (if somewhat less robust) solution based on R Markdown. CRAN. | More info in [Using flexdashboard](http://rmarkdown.rstudio.com/flexdashboard/using.html) | JJ Allaire, RStudio & others |
| [openxlsx](https://github.com/awalker89/openxlsx) | misc | If you need to write to an Excel file as well as read, this package is easy to use and offers a lot of options for formatting your spreadsheet. CRAN. | write.xlsx(mydf, "myfile.xlsx") | Alexander Walker |
| [gmodels](http://cran.r-project.org/web/packages/gmodels/gmodels.pdf) | data wrangling, data analysis | There are several functions for modeling data here, but the one I use, CrossTable, simply creates cross-tabs with loads of options -- totals, proprotions and several statistical tests. CRAN. | CrossTable(myxvector, myyvector, prop.t=FALSE, prop.chisq = FALSE) | Gregory R. Warnes |
| [janitor](https://github.com/sfirke/janitor) | data wrangling, data analysis | Basic data cleaning made easy, such as finding duplicates by multiple columns, making R-friendly column names and removing empty columns. It also has some nice tabulating tools, like adding a total row, as well as generating tables with percentages and easy crosstabs. And, its get\_dupes() function is an elegant way of finding duplicate rows in data frames, either based on one column, several columns, or entire rows. CRAN. | tabyl(mydf, sort = TRUE) %>% adorn\_totals("row") | Samuel Firke |
| [car](https://mran.microsoft.com/package/car/) | data wrangling | car's recode function makes it easy to bin continuous numerical data into categories or factors. While base R's cut accomplishes the same task, I find recode's syntax to be more intuitive - just remember to put the entire recoding formula within double quotation marks. dplyr's [case\_when() function](http://dplyr.tidyverse.org/reference/case_when.html) is another option worth considering. CRAN. | recode(x, "1:3='Low'; 4:7='Mid'; 8:hi='High'") | John Fox & others |
| [rcdimple](https://github.com/timelyportfolio/rcdimple) | data visualization | R interface to the dimple JavaScript library with numerous customization options. Good choice for JavaScript bar charts, among others. GitHub timelyportfolio/rcdimple. | dimple(mtcars, mpg ~ cyl, type = "bar") | Kent Russell |
| [scales](https://cran.r-project.org/web/packages/scales/) | data wrangling | While this package has many more sophisticated ways to help you format data for graphing, it's worth a download just for the comma(), percent() and dollar() functions. CRAN. | comma(mynumvec) | Hadley Wickham |
| [plotly](https://plot.ly/r/) | data visualization | R interface to the Plotly JavaScript library that was open-sourced in late 2015. Basic graphs have a distinctive look which may not be for everyone, but it's full-featured, relatively easy to learn (especially if you know ggplot2) and includes a ggplotly() function to turn graphs created with ggplot2 interactive. CRAN. | d <- diamonds[sample(nrow(diamonds), 1000), ]  plot\_ly(d, x = carat, y = price, text = paste("Clarity: ", clarity), mode = "markers", color = carat, size = carat) | Carson Sievert & others |
| [highcharter](http://jkunst.com/highcharter/) | data visualization | R wrapper for the robust and well documented Highcharts JavaScript library, one of my favorite choices for presentation-quality interactive graphics. The package uses ggplot2-like syntax, including options for handling both long and wide data, and comes with plenty of examples. Note that a [paid Highcharts license](https://shop.highsoft.com/highcharts) is needed to use this for commercial or government work (it's free for personal and non-profit projects). CRAN. . CRAN. | hchart(mydf, "charttype", hcaes(x = xcol, y = ycol, group = groupbycol)) | Joshua Kunst & others |
| [profvis](https://rstudio.github.io/profvis/) | programming | Is your R code sluggish? This package gives you a visual representative of your code line by line so you can find the speed bottlenecks. CRAN. | profvis({ *your code here* }) | Winston Chang & others |
| [tidytext](https://github.com/juliasilge/tidytext) | text mining | Elegant implementation of text mining functions using Hadley Wickham's "tidy data" principles. CRAN. | See [tidytextmining.com](http://tidytextmining.com/) for numerous examples. | Julia Silge & David Robinson |
| [diffobj](https://cran.r-project.org/web/packages/diffobj/vignettes/diffobj.html) | data analysis | Base R's identical() function tells you whether or not two objects are the same; but if they're not, it won't tell you why. diffobj gives you a visual representation of how two R objects differ. CRAN. | diffObj(x,y) | Brodie Gaslam & Michael B. Allen |
| [Prophet](https://facebookincubator.github.io/prophet/) | forecasting | I don't do much forecasting analysis; but if I did, I'd start with this package. CRAN. | See the [Quick start guide](https://facebookincubator.github.io/prophet/docs/quick_start.html#r-api). | Sean Taylor & Ben Letham at Facebook |
| [feather](https://blog.rstudio.org/2016/03/29/feather/) | data import, data export | This binary data-file format can be read by both Python and R, making data interchange easier between the two languages. It's also built for I/O speed. CRAN. | write\_feather(mydf, "myfile") | Wes McKinney & Hadley Wickham |
| [fst](http://www.fstpackage.org/) | data import, data export | Another alternative for binary file storage (R-only), fst was built for fast storage and retrieval, with access speeds above 1 GB/sec. It also offers compression that doesn't slow data access too much, as well as the ability to import a specific range of rows (by row number). CRAN. | write.fst(mydf, "myfile.fst", 100) | Mark Klik |
| [googleAuthR](http://code.markedmondson.me/googleAuthR/) | data import | If you want to use data from a Google API in an R project and there's not yet a specific package for that API, this is the place to turn for authenticating CRAN. | See examples on [the package website](http://code.markedmondson.me/googleAuthR/) and [this gist](https://gist.github.com/MarkEdmondson1234/0198e283cd228565f9313cf36f35c7ab) for use with Google Calendars. CRAN. | Mark Edmondson |
| [devtools](https://github.com/hadley/devtools) | package development, package installation | devtools has a slew of functions aimed at helping you create your own R packages, such as automatically running all example code in your help files to make sure everything works. Requires [Rtools](http://cran.r-project.org/bin/windows/Rtools/) on Windows and [XCode](https://developer.apple.com/xcode/downloads/) on a Mac. On CRAN. | run\_examples() | Hadley Wickham & others |
| [remotes](https://github.com/r-pkgs/remotes) | package installation | If you want to install R packages from GitHub, devtools was long the go-to. However, it has a ton of other functions and some hefty dependences. remotes is a lighter-weight alternative if all you want is to install packages from GitHub as well as Bitbucket and some other sources. CRAN. ([ghit](https://cran.r-project.org/web/packages/ghit/) is another option, but is GitHub-only.) | remotes::install\_github("mangothecat/franc") | Gabor Csardi & others |
| [githubinstall](http://hoxo-m.github.io/githubinstall/) | package installation | Do you want to install a package from GitHub without typing out the GitHub user name along with the repo name? Whether because you can't remember a package's GitHub owner's name, that name is long/complex to type out, or you just want to save yourself a little typing, this package is a handy option. Simply run githubinstall("packagename") and the package will suggest an account; then you respond Y to install or n if it's the wrong one. It even includes fuzzy matching if you misspell a package name! | githubinstall::githubinstall::("AnomalyDetection") | Koji Makiyama |
| [installr](https://github.com/talgalili/installr/) | misc | Windows only: Update your installed version of R from within R. On CRAN. | updateR() | Tal Galili & others |
| [reinstallr](https://github.com/calligross/reinstallr) | misc | Seeks to find packages that had previously been installed on your system and need to be re-installed after upgrading R. CRAN. | reinstallr() | Calli Gross |
| [usethis](https://github.com/r-lib/usethis) | package development, programming | Initially aimed at package development, usethis now includes useful functions for any coding project. Among its handy features are an edit family that lets you easily update your .Renvironment and .Rprofile files. On CRAN, but install GitHub version from "r-lib/usethis" for latest updates. | edit\_r\_environ() | Hadley Wickham, Jennifer Bryan & RStudio |
| [here](https://github.com/krlmlr/here) | misc | This package has one function with a single, useful purpose: find your project's working directory. Surprisingly helpful if you want your code to run on more than one system. CRAN. | my\_project\_directory <- here() | Kirill Müller |
| [pacman](https://cran.r-project.org/web/packages/pacman/vignettes/Introduction_to_pacman.html) | misc, package installation | This package is another that aims to solve one problem, and solve it well: package installation. The main functions will loadi a package that's already installed or installing it first if it's not available. While this is certainly possible to do with base R's require() and an if statement, p\_load() is so much more elegant for CRAN packages, or p\_load\_gh() for GitHub. Other useful options include p\_temp(), which allows for a temporary, this-session-only package installation. CRAN. | p\_load(dplyr, here, tidycensus) | Tyler Rinker |
| [plumber](https://www.rplumber.io/) | data export, programming | Turn any R function into a host-able API with a line or two of code. This well-thought-out package makes it easy to use R for data handling in other, non-R coding projects. CRAN. | See the [documentation](https://www.rplumber.io/docs/) or my article [Create your own Slack bots -- and Web APIs -- with R](https://www.computerworld.com/article/3171766/application-development/how-to-create-your-own-slack-bots-and-web-apis-in-r.html) | Jeff Allen, Trestle Technology & others |
| [echarts4r](http://echarts4r.john-coene.com/index.html) | data visualization | R wrapper for the powerful and flexible ECharts JavaScript library. It features dozens of chart and graph types, from bar and line charts to sunbursts, heat maps, and geographical maps. Hundreds of customizations not explicitly mentioned in the package docs are nevertheless available; you just need to peruse the [original ECharts documentation](https://ecomfe.github.io/echarts-doc/public/en/option.html). (ECharts is an Apache Software Foundation incubator project.) CRAN. | mtcars %>% e\_charts(wt) %>% e\_line(mpg) | John Coene |
| [dataCompareR](https://github.com/capitalone/dataCompareR) | data wrangling | A quick and elegant way to compare two data frames, either row by row or by a specified key. CRAN. | rCompare(mydf1, mydf2) | Rob Noble-Eddy at CapitalOne & others |
| [cloudyR project](https://cloudyr.github.io/) | data import, data export | This is a collection of packages aimed at making it easier for R to work with cloud platforms such as Amazon Web Services, Google and Travis-CI. Some are already on CRAN, some can be found on GitHub. | See the [list of packages](http://cloudyr.github.io/packages/index.html). | Various |
| [flyio](https://github.com/socialcopsdev/flyio/) | data import, data export | This is a bit like rio, but for the cloud: It offers a common set of functions whether you're using Amazon's S3 or Google Cloud. Set your data source, authenticate with your credentials (which can be stored in an R environmental variable), set a bucket name, and off you go. GitHub. | See the [GitHub repo](https://github.com/socialcopsdev/flyio/) or [YouTube video](https://youtu.be/vrRFjv25Bxc) of a demo at the Delhi useR meetup. | SocialCops |
| [geofacet](https://github.com/hafen/geofacet) | data visualization, mapping | To be honest, I rarely need the ability create "geofacets" -- maps with same-sized blocks in geospatially appropriate locations. However, this package is so cool that I had to include it. Geofaceting is best understood by [looking at an example](https://cloud.githubusercontent.com/assets/1275592/26282368/6118d06a-3dc5-11e7-96b4-6a511800b6d3.png). The package lets you create your own geofacet visualizations using ggplot2 and built-in grids such as US states, EU countries and San Francisco Bay Area counties. Even more impressive, it comes with design-your-own geofacet grid capabilities. CRAN. | grid\_design() | Ryan Hafen |
| [reticulate](https://rstudio.github.io/reticulate/) | programming | If you know Python as well as R, this package offers a suite of tools for calling Python from within R, as well as "translating" between R and Python objects such as Pandas data frames and R data frames. CRAN. | See the [reticulate package website](https://rstudio.github.io/reticulate/). | JJ Allaire |
| [slackr](https://github.com/hrbrmstr/slackr) | collaboration | Do you use Slack? If so, you can send messages and files into a Slack channel, as long as you've got a token from that Slack. Useful to run analysis and then quickly share results with a team. GitHub hrbrmstr/slackr | See the [GitHub repo](https://github.com/hrbrmstr/slackr). | Bob Rudis |
| [beepr](https://cran.r-project.org/package=beepr) | misc | This is pretty much pure fun. Yes, getting an audible notification when code finishes running or encounters an error could be useful; but here, the available sounds include options like a fanfare flourish, a Mario Brothers tune, and even a scream. CRAN. | beep("wilhelm") | Rasmus Bååth |

A few important points for newbies:

To install a package from CRAN, use the command install.packages("packagename") -- of course substituting the actual package name for packagename and putting it in quotation marks. Package names, like pretty much everything else in R, are case sensitive.

To install from GitHub, it's easiest to use the install-github function from the devtools package, using the format devtools::install\_github("githubaccountname/packagename"). That means you first want to install the devtools package on your system with install.packages("devtools"). Note that devtools sometimes needs some extra non-R software on your system -- more specifically, an [Rtools download for Windows](http://cran.r-project.org/bin/windows/Rtools/) or [Xcode for OS X](https://developer.apple.com/xcode/downloads/). There's [more information about devtools here](https://github.com/hadley/devtools).

In order to use a package's function during your R session, you need to do one of two things. One option is to load it into your R session with the library("packagename") or require("packagename"). The other is to call the function including the package name, like this: packagename::functioname(). Package names, like pretty much everything else in R, are case sensitive.