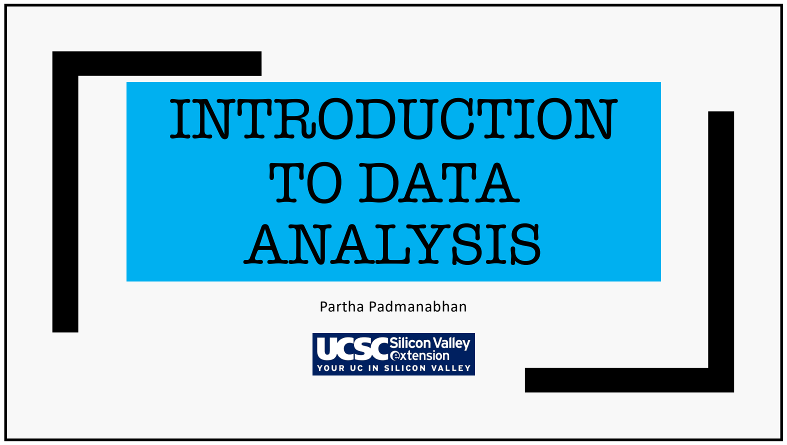
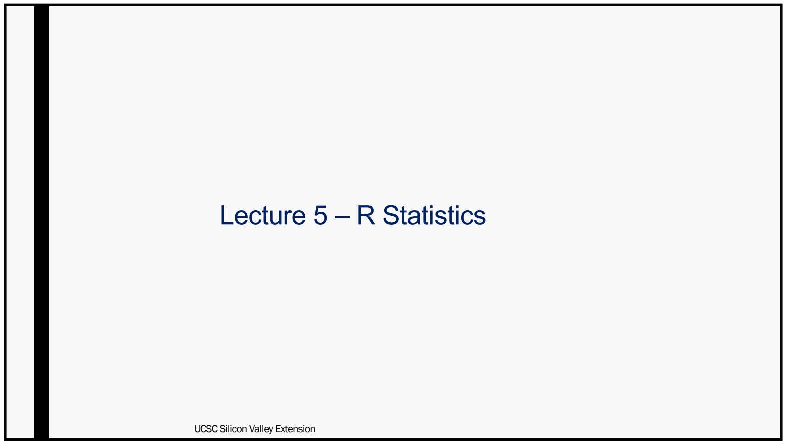


INTRODUCTION TO DATA ANALYSIS

Partha Padmanabhan

UCSC Silicon Valley extension
YOUR UC IN SILICON VALLEY

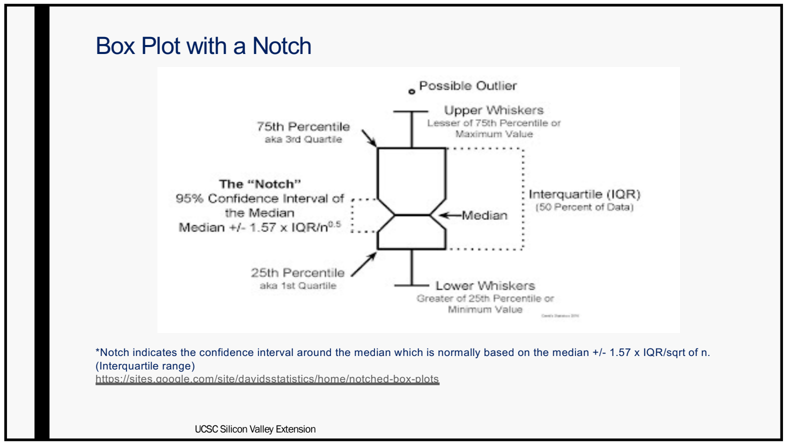
1



Lecture 5 – R Statistics

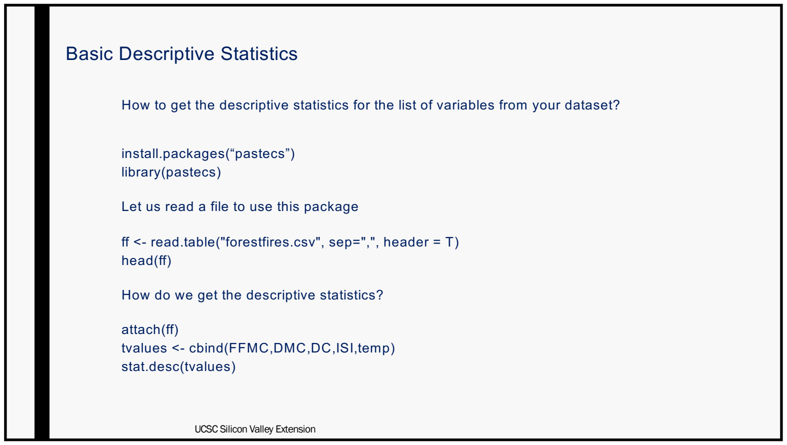
UCSC Silicon Valley Extension

2



Box Plot with a Notch

*Notch indicates the confidence interval around the median which is normally based on the median +/- 1.57 x IQR/sqrt of n. (Interquartile range)
https://sites.google.com/site/davidsstatistics/home/notched-box-plots



Basic Descriptive Statistics

How to get the descriptive statistics for the list of variables from your dataset?

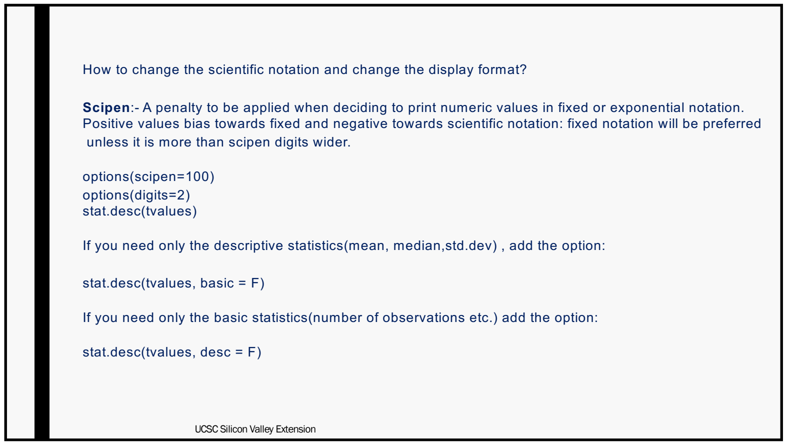
```
install.packages(“pastecs”)
library(pastecs)
```

Let us read a file to use this package

```
ff <- read.table("forestfires.csv", sep=",", header = T)
head(ff)
```

How do we get the descriptive statistics?

```
attach(ff)
tvalues <- cbind(FFMC,DMC,DC,ISI,temp)
stat.desc(tvalues)
```



How to change the scientific notation and change the display format?

Scipen:- A penalty to be applied when deciding to print numeric values in fixed or exponential notation. Positive values bias towards fixed and negative towards scientific notation: fixed notation will be preferred unless it is more than scipen digits wider.

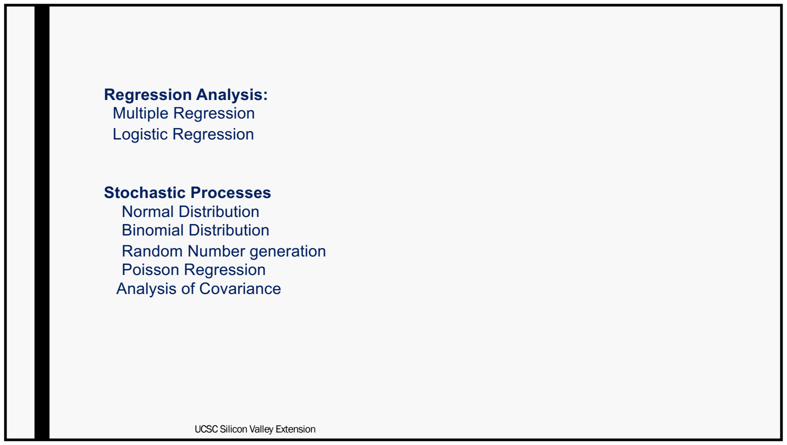
```
options(scipen=100)
options(digits=2)
stat.desc(tvalues)
```

If you need only the descriptive statistics(mean, median,std.dev) , add the option:

```
stat.desc(tvalues, basic = F)
```

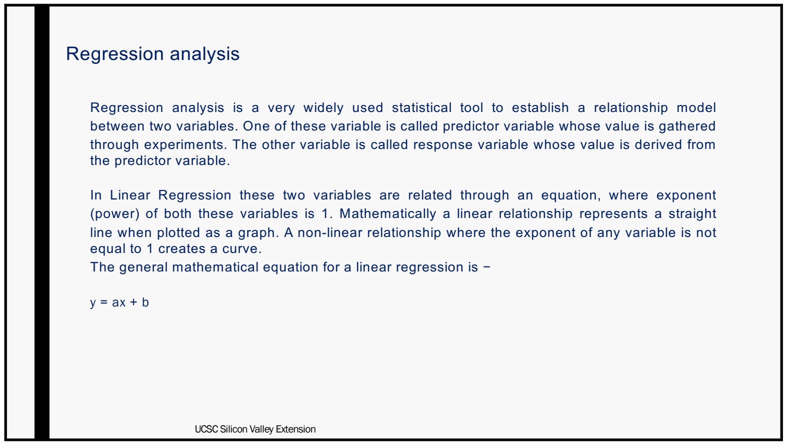
If you need only the basic statistics(number of observations etc.) add the option:

```
stat.desc(tvalues, desc = F)
```



Regression Analysis:
Multiple Regression
Logistic Regression

Stochastic Processes
Normal Distribution
Binomial Distribution
Random Number generation
Poisson Regression
Analysis of Covariance

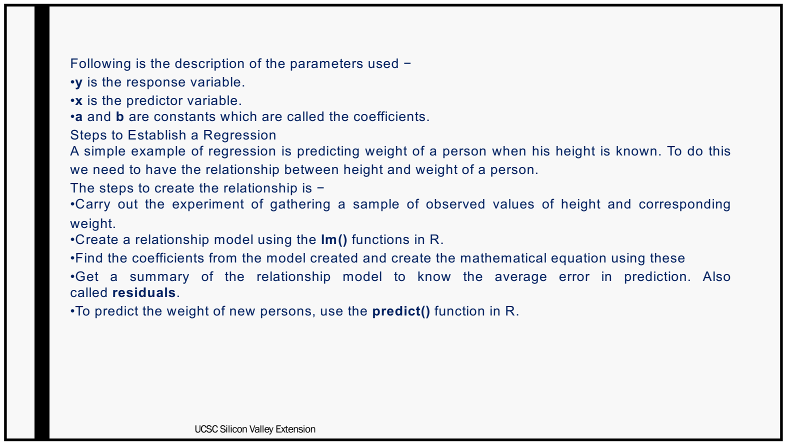


Regression analysis

Regression analysis is a very widely used statistical tool to establish a relationship model between two variables. One of these variable is called predictor variable whose value is gathered through experiments. The other variable is called response variable whose value is derived from the predictor variable.

In Linear Regression these two variables are related through an equation, where exponent (power) of both these variables is 1. Mathematically a linear relationship represents a straight line when plotted as a graph. A non-linear relationship where the exponent of any variable is not equal to 1 creates a curve.
The general mathematical equation for a linear regression is −

$y = ax + b$

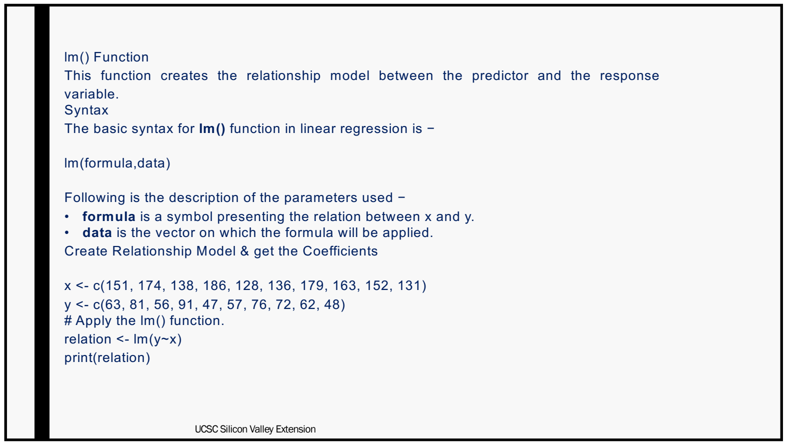


Following is the description of the parameters used −

- **y** is the response variable.
- **x** is the predictor variable.
- **a** and **b** are constants which are called the coefficients.

Steps to Establish a Regression
A simple example of regression is predicting weight of a person when his height is known. To do this we need to have the relationship between height and weight of a person.
The steps to create the relationship is −

- Carry out the experiment of gathering a sample of observed values of height and corresponding weight.
- Create a relationship model using the **lm()** functions in R.
- Find the coefficients from the model created and create the mathematical equation using these
- Get a summary of the relationship model to know the average error in prediction. Also called **residuals**.
- To predict the weight of new persons, use the **predict()** function in R.



lm() Function

This function creates the relationship model between the predictor and the response variable.

Syntax

The basic syntax for **lm()** function in linear regression is −

```
lm(formula,data)
```

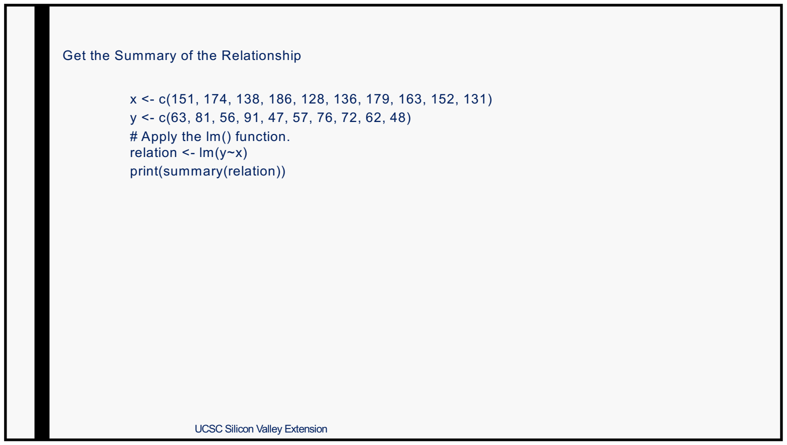
Following is the description of the parameters used −

- **formula** is a symbol presenting the relation between x and y.
- **data** is the vector on which the formula will be applied.

Create Relationship Model & get the Coefficients

```
x <- c(151, 174, 138, 186, 128, 136, 179, 163, 152, 131)
y <- c(63, 81, 56, 91, 47, 57, 76, 72, 62, 48)
# Apply the lm() function.
relation <- lm(y~x)
print(relation)
```

9



Get the Summary of the Relationship

```
x <- c(151, 174, 138, 186, 128, 136, 179, 163, 152, 131)
y <- c(63, 81, 56, 91, 47, 57, 76, 72, 62, 48)
# Apply the lm() function.
relation <- lm(y~x)
print(summary(relation))
```

10

predict() Function

Syntax

The basic syntax for predict() in linear regression is −

predict(object, newdata) Following is the description of the parameters used −

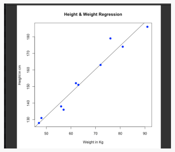
- **object** is the formula which is already created using the lm() function.
- **newdata** is the vector containing the new value for predictor variable.

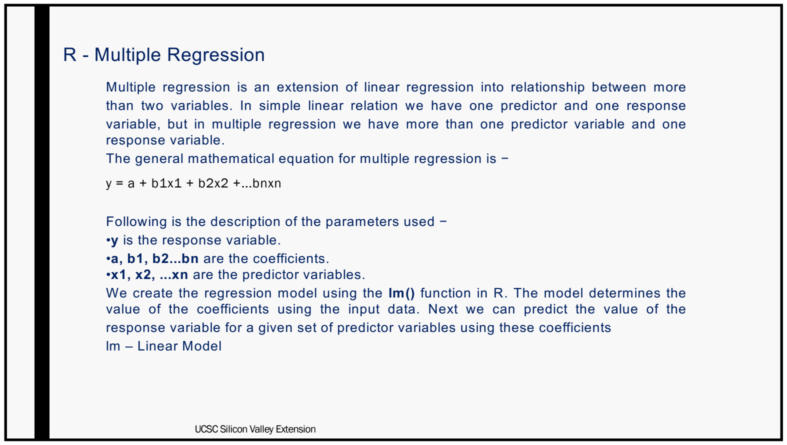
Predict the weight of new persons

```
# The predictor vector.
x <- c(151, 174, 138, 186, 128, 136, 179, 163, 152, 131)
# The resposne vector.
y <- c(63, 81, 56, 91, 47, 57, 76, 72, 62, 48)
# Apply the lm() function.
relation <- lm(y~x)
# Find weight of a person with height 170.
a <- data.frame(x = 170)
 result <- predict(relation,a)
print(result)
```

Visualize the Regression Graphically

```
# Create the predictor and response variable.
x <- c(151, 174, 138, 186, 128, 136, 179, 163, 152, 131)
y <- c(63, 81, 56, 91, 47, 57, 76, 72, 62, 48)
relation <- lm(y~x)
# Give the chart file a name.
png(file = "linearregression.png")
# Plot the chart.
plot(y,x,col = "blue",main = "Height & Weight Regression", abline(lm(x~y)),cex = 1.3,pch = 16,xlab
= "Weight in Kg",ylab = "Height in cm")
# Save the file.
dev.off()
```





R - Multiple Regression

Multiple regression is an extension of linear regression into relationship between more than two variables. In simple linear relation we have one predictor and one response variable, but in multiple regression we have more than one predictor variable and one response variable.

The general mathematical equation for multiple regression is −

y = a + b1x1 + b2x2 +...bnxn

Following is the description of the parameters used −

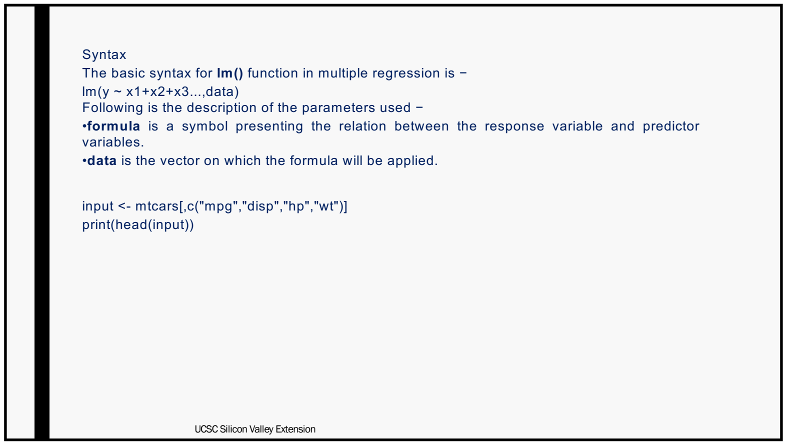
•**y** is the response variable.

•**a, b1, b2...bn** are the coefficients.

•**x1, x2, ...xn** are the predictor variables.

We create the regression model using the **lm()** function in R. The model determines the value of the coefficients using the input data. Next we can predict the value of the response variable for a given set of predictor variables using these coefficients

lm – Linear Model



Syntax

The basic syntax for **lm()** function in multiple regression is −

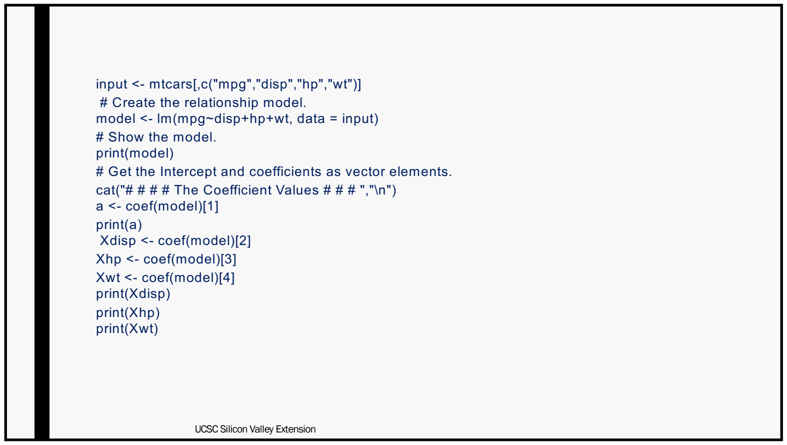
lm(y ~ x1+x2+x3...,data)

Following is the description of the parameters used −

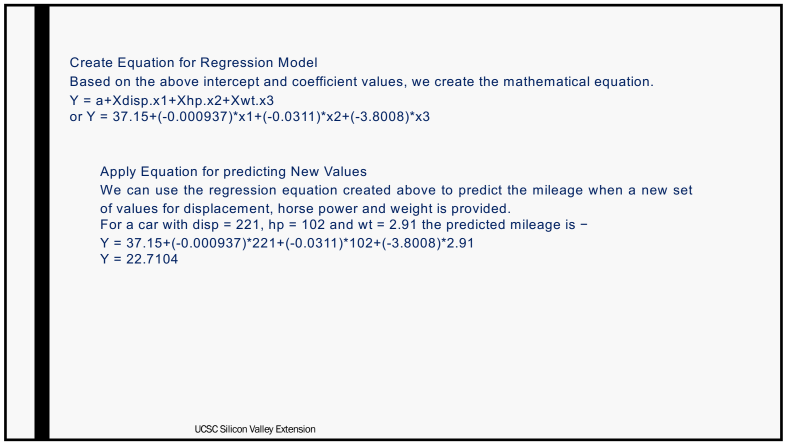
•**formula** is a symbol presenting the relation between the response variable and predictor variables.

•**data** is the vector on which the formula will be applied.

```
input <- mtcars[,c("mpg","disp","hp","wt")]
print(head(input))
```



```
input <- mtcars[,c("mpg","disp","hp","wt")]
 # Create the relationship model.
model <- lm(mpg~disp+hp+wt, data = input)
# Show the model.
print(model)
# Get the Intercept and coefficients as vector elements.
cat("# # # # The Coefficient Values # # # ","\n")
a <- coef(model)[1]
print(a)
 Xdisp <- coef(model)[2]
Xhp <- coef(model)[3]
Xwt <- coef(model)[4]
print(Xdisp)
print(Xhp)
print(Xwt)
```



Create Equation for Regression Model

Based on the above intercept and coefficient values, we create the mathematical equation.

Y = a+Xdisp.x1+Xhp.x2+Xwt.x3
or Y = 37.15+(-0.000937)*x1+(-0.0311)*x2+(-3.8008)*x3

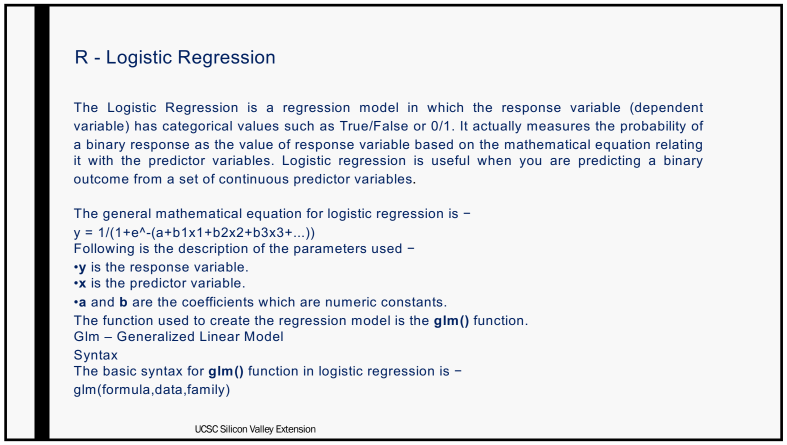
Apply Equation for predicting New Values

We can use the regression equation created above to predict the mileage when a new set of values for displacement, horse power and weight is provided.

For a car with disp = 221, hp = 102 and wt = 2.91 the predicted mileage is −

Y = 37.15+(-0.000937)*221+(-0.0311)*102+(-3.8008)*2.91

Y = 22.7104



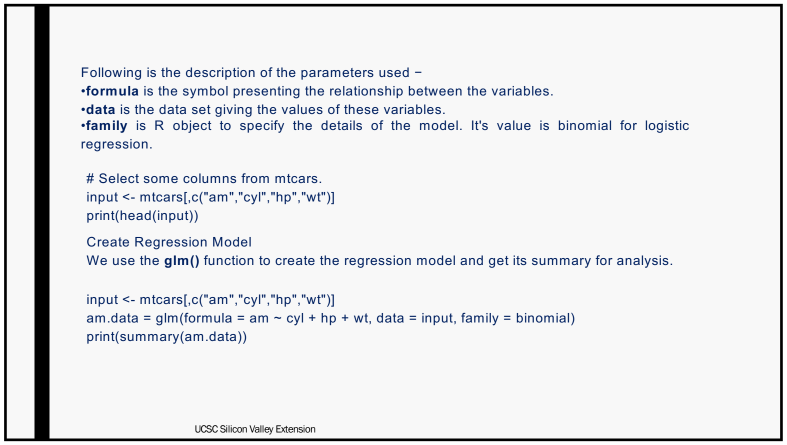
R - Logistic Regression

The Logistic Regression is a regression model in which the response variable (dependent variable) has categorical values such as True/False or 0/1. It actually measures the probability of a binary response as the value of response variable based on the mathematical equation relating it with the predictor variables. Logistic regression is useful when you are predicting a binary outcome from a set of continuous predictor variables.

The general mathematical equation for logistic regression is −
y = 1/(1+e^-(a+b1x1+b2x2+b3x3+...))
Following is the description of the parameters used −

- **y** is the response variable.
- **x** is the predictor variable.
- **a** and **b** are the coefficients which are numeric constants.

The function used to create the regression model is the **glm()** function.
Glm – Generalized Linear Model
Syntax
The basic syntax for **glm()** function in logistic regression is −
glm(formula,data,family)



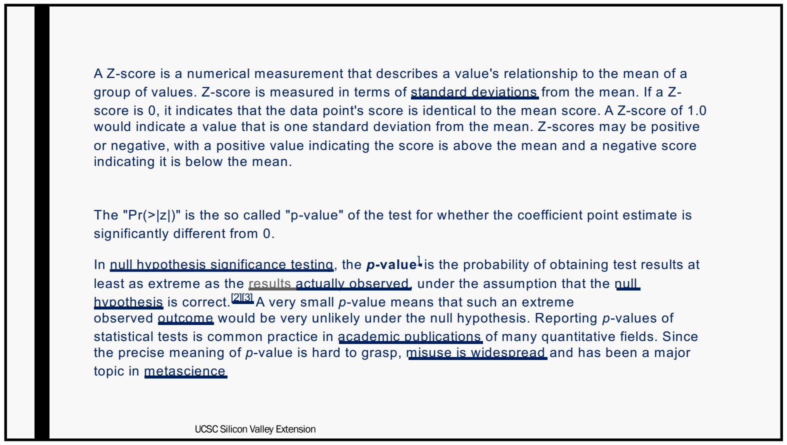
Following is the description of the parameters used −

- **formula** is the symbol presenting the relationship between the variables.
- **data** is the data set giving the values of these variables.
- **family** is R object to specify the details of the model. It's value is binomial for logistic regression.

```
# Select some columns from mtcars.
input <- mtcars[,c("am","cyl","hp","wt")]
print(head(input))
```

Create Regression Model
We use the **glm()** function to create the regression model and get its summary for analysis.

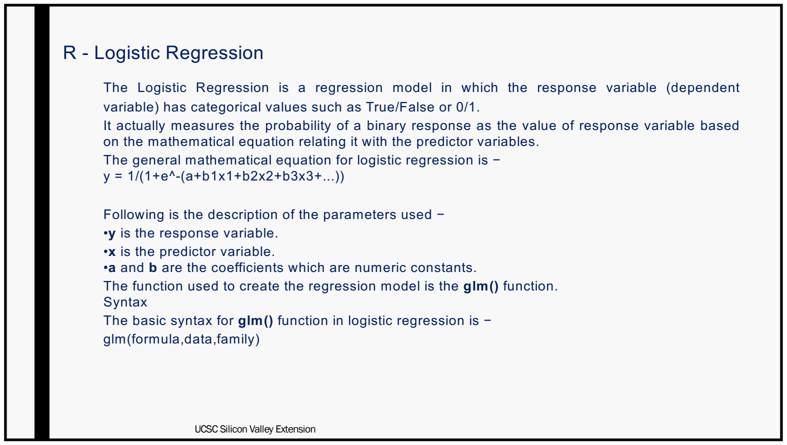
```
input <- mtcars[,c("am","cyl","hp","wt")]
am.data = glm(formula = am ~ cyl + hp + wt, data = input, family = binomial)
print(summary(am.data))
```



A Z-score is a numerical measurement that describes a value's relationship to the mean of a group of values. Z-score is measured in terms of standard deviations from the mean. If a Z-score is 0, it indicates that the data point's score is identical to the mean score. A Z-score of 1.0 would indicate a value that is one standard deviation from the mean. Z-scores may be positive or negative, with a positive value indicating the score is above the mean and a negative score indicating it is below the mean.

The "Pr(>|z|)" is the so called "p-value" of the test for whether the coefficient point estimate is significantly different from 0.

In null hypothesis significance testing, the ***p*-value**[1] is the probability of obtaining test results at least as extreme as the results actually observed, under the assumption that the null hypothesis is correct.[2][3] A very small *p*-value means that such an extreme observed outcome would be very unlikely under the null hypothesis. Reporting *p*-values of statistical tests is common practice in academic publications of many quantitative fields. Since the precise meaning of *p*-value is hard to grasp, misuse is widespread and has been a major topic in metascience.



R - Logistic Regression

The Logistic Regression is a regression model in which the response variable (dependent variable) has categorical values such as True/False or 0/1.

It actually measures the probability of a binary response as the value of response variable based on the mathematical equation relating it with the predictor variables.

The general mathematical equation for logistic regression is −

y = 1/(1+e^-(a+b1x1+b2x2+b3x3+...))

Following is the description of the parameters used −

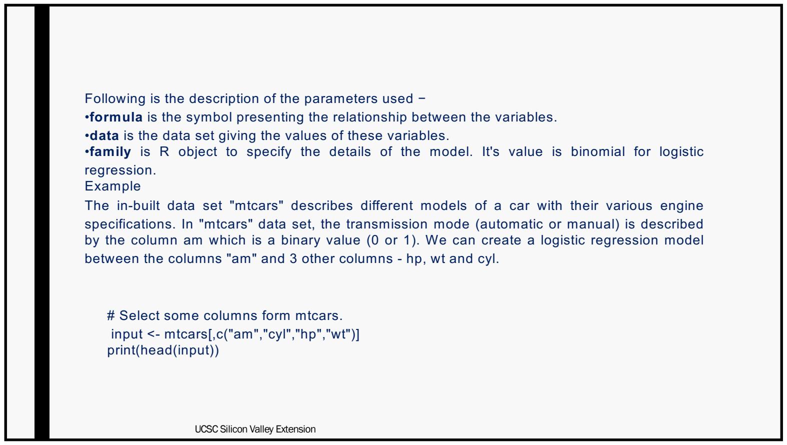
- **y** is the response variable.
- **x** is the predictor variable.
- **a** and **b** are the coefficients which are numeric constants.

The function used to create the regression model is the **glm()** function.

Syntax

The basic syntax for **glm()** function in logistic regression is −

glm(formula,data,family)



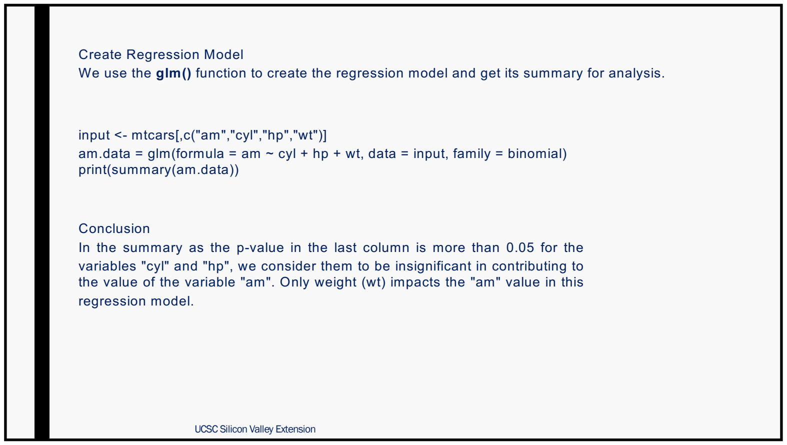
Following is the description of the parameters used −

- **formula** is the symbol presenting the relationship between the variables.
- **data** is the data set giving the values of these variables.
- **family** is R object to specify the details of the model. It's value is binomial for logistic regression.

Example

The in-built data set "mtcars" describes different models of a car with their various engine specifications. In "mtcars" data set, the transmission mode (automatic or manual) is described by the column am which is a binary value (0 or 1). We can create a logistic regression model between the columns "am" and 3 other columns - hp, wt and cyl.

```
# Select some columns form mtcars.
 input <- mtcars[,c("am","cyl","hp","wt")]
print(head(input))
```



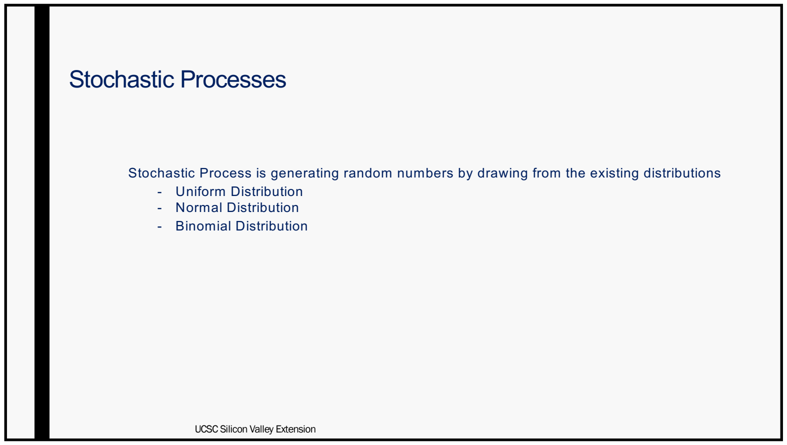
Create Regression Model

We use the **glm()** function to create the regression model and get its summary for analysis.

```
input <- mtcars[,c("am","cyl","hp","wt")]
am.data = glm(formula = am ~ cyl + hp + wt, data = input, family = binomial)
print(summary(am.data))
```

Conclusion

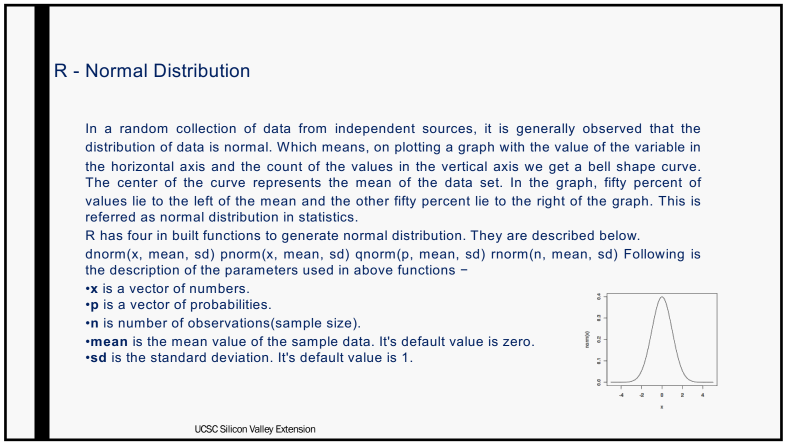
In the summary as the p-value in the last column is more than 0.05 for the variables "cyl" and "hp", we consider them to be insignificant in contributing to the value of the variable "am". Only weight (wt) impacts the "am" value in this regression model.



Stochastic Processes

Stochastic Process is generating random numbers by drawing from the existing distributions

- Uniform Distribution
- Normal Distribution
- Binomial Distribution



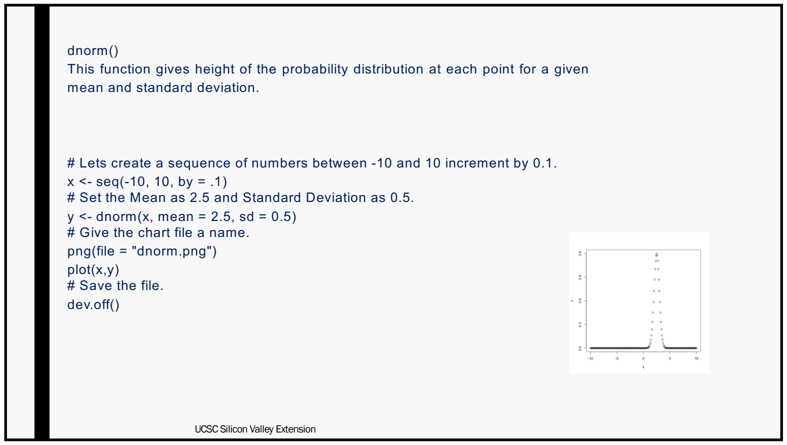
R - Normal Distribution

In a random collection of data from independent sources, it is generally observed that the distribution of data is normal. Which means, on plotting a graph with the value of the variable in the horizontal axis and the count of the values in the vertical axis we get a bell shape curve. The center of the curve represents the mean of the data set. In the graph, fifty percent of values lie to the left of the mean and the other fifty percent lie to the right of the graph. This is referred as normal distribution in statistics.

R has four in built functions to generate normal distribution. They are described below.

dnorm(x, mean, sd) pnorm(x, mean, sd) qnorm(p, mean, sd) rnorm(n, mean, sd) Following is the description of the parameters used in above functions −

- **x** is a vector of numbers.
- **p** is a vector of probabilities.
- **n** is number of observations(sample size).
- **mean** is the mean value of the sample data. It's default value is zero.
- **sd** is the standard deviation. It's default value is 1.

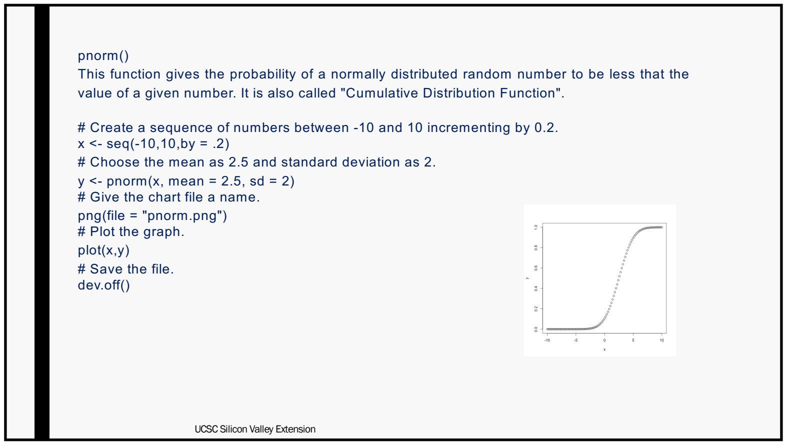


dnorm()

This function gives height of the probability distribution at each point for a given mean and standard deviation.

```
# Lets create a sequence of numbers between -10 and 10 increment by 0.1.
x <- seq(-10, 10, by = .1)
# Set the Mean as 2.5 and Standard Deviation as 0.5.
y <- dnorm(x, mean = 2.5, sd = 0.5)
# Give the chart file a name.
png(file = "dnorm.png")
plot(x,y)
# Save the file.
dev.off()
```

25



pnorm()

This function gives the probability of a normally distributed random number to be less that the value of a given number. It is also called "Cumulative Distribution Function".

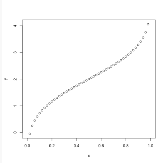
```
# Create a sequence of numbers between -10 and 10 incrementing by 0.2.
x <- seq(-10,10,by = .2)
# Choose the mean as 2.5 and standard deviation as 2.
y <- pnorm(x, mean = 2.5, sd = 2)
# Give the chart file a name.
png(file = "pnorm.png")
# Plot the graph.
plot(x,y)
# Save the file.
dev.off()
```

26

qnorm()

This function takes the probability value and gives a number whose cumulative value matches the probability value.

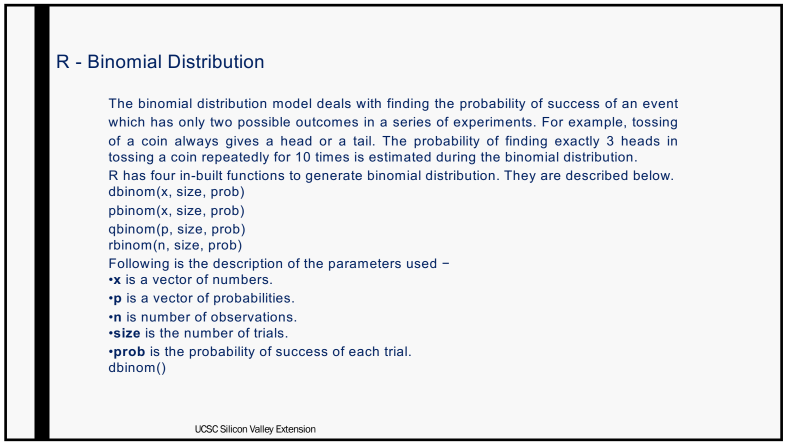
```
# Create a sequence of probability values incrementing by 0.02.
x <- seq(0, 1, by = 0.02)
# Choose the mean as 2 and standard deviation as 3.
y <- qnorm(x, mean = 2, sd = 1)
# Give the chart file a name.
png(file = "qnorm.png")
# Plot the graph.
plot(x,y)
# Save the file.
dev.off()
```



rnorm()

This function is used to generate random numbers whose distribution is normal. It takes the sample size as input and generates that many random numbers. We draw a histogram to show the distribution of the generated numbers.

```
# Create a sample of 50 numbers which are normally distributed.
y <- rnorm(50)
# Give the chart file a name.
png(file = "rnorm.png")
# Plot the histogram for this sample.
hist(y, main = "Normal Distribution")
# Save the file.
dev.off()
```



R - Binomial Distribution

The binomial distribution model deals with finding the probability of success of an event which has only two possible outcomes in a series of experiments. For example, tossing of a coin always gives a head or a tail. The probability of finding exactly 3 heads in tossing a coin repeatedly for 10 times is estimated during the binomial distribution.

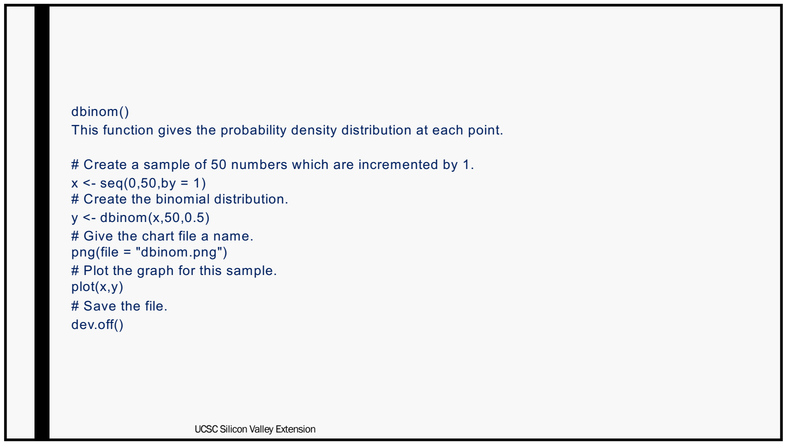
R has four in-built functions to generate binomial distribution. They are described below.

dbinom(x, size, prob)
pbinom(x, size, prob)
qbinom(p, size, prob)
rbinom(n, size, prob)

Following is the description of the parameters used −

- **x** is a vector of numbers.
- **p** is a vector of probabilities.
- **n** is number of observations.
- **size** is the number of trials.
- **prob** is the probability of success of each trial.

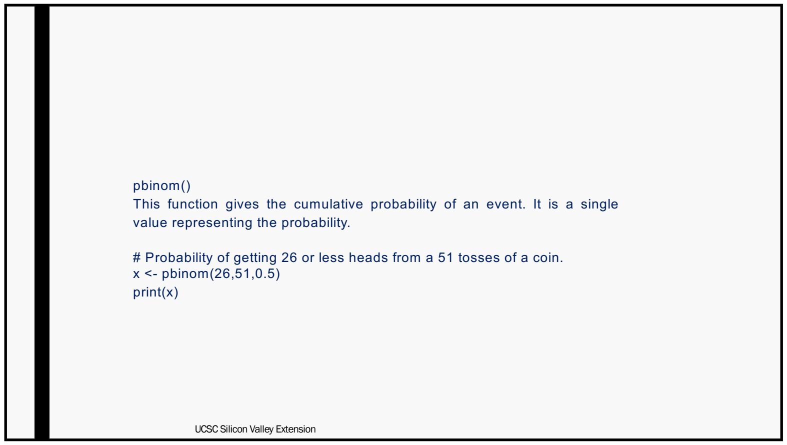
dbinom()



dbinom()

This function gives the probability density distribution at each point.

```
# Create a sample of 50 numbers which are incremented by 1.
x <- seq(0,50,by = 1)
# Create the binomial distribution.
y <- dbinom(x,50,0.5)
# Give the chart file a name.
png(file = "dbinom.png")
# Plot the graph for this sample.
plot(x,y)
# Save the file.
dev.off()
```

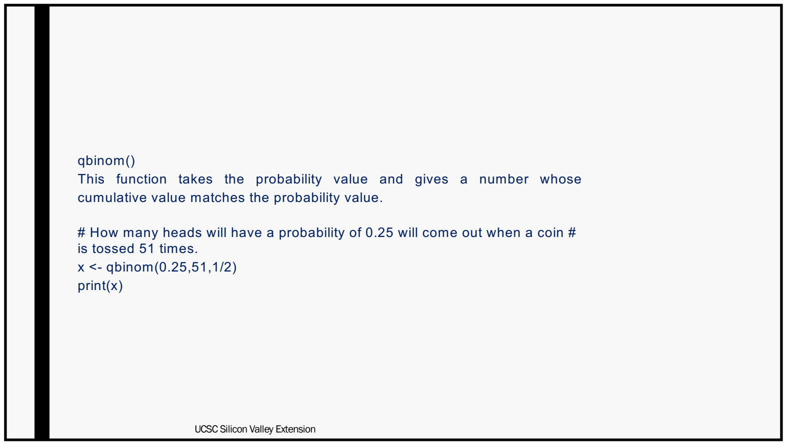


pbinom()

This function gives the cumulative probability of an event. It is a single value representing the probability.

```
# Probability of getting 26 or less heads from a 51 tosses of a coin.
x <- pbinom(26,51,0.5)
print(x)
```

31



qbinom()

This function takes the probability value and gives a number whose cumulative value matches the probability value.

```
# How many heads will have a probability of 0.25 will come out when a coin #
is tossed 51 times.
x <- qbinom(0.25,51,1/2)
print(x)
```

32

rbinom()
This function generates required number of random values of given probability from a given sample.

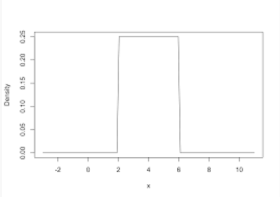
```
# Find 8 random values from a sample of 150 with probability of 0.4.
x <- rbinom(8,150,.4)
print(x)
```

Random Number generation

Draw from existing distributions:

a) Uniform Distribution

The **continuous uniform distribution** is the probability distribution of random number selection from the continuous interval between *a* and *b*

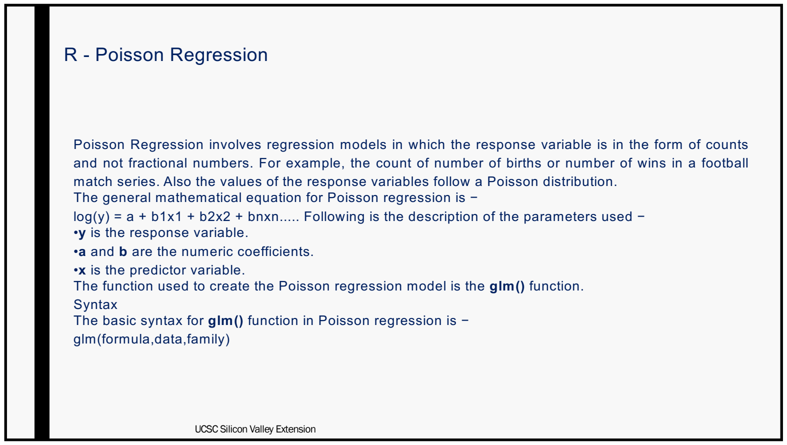


R command:

```
runif(n,min,max)
```

n= Number of times that we want to draw from the distribution

Min & Max (Range of values)



R - Poisson Regression

Poisson Regression involves regression models in which the response variable is in the form of counts and not fractional numbers. For example, the count of number of births or number of wins in a football match series. Also the values of the response variables follow a Poisson distribution.

The general mathematical equation for Poisson regression is −

log(y) = a + b1x1 + b2x2 + bnxn..... Following is the description of the parameters used −

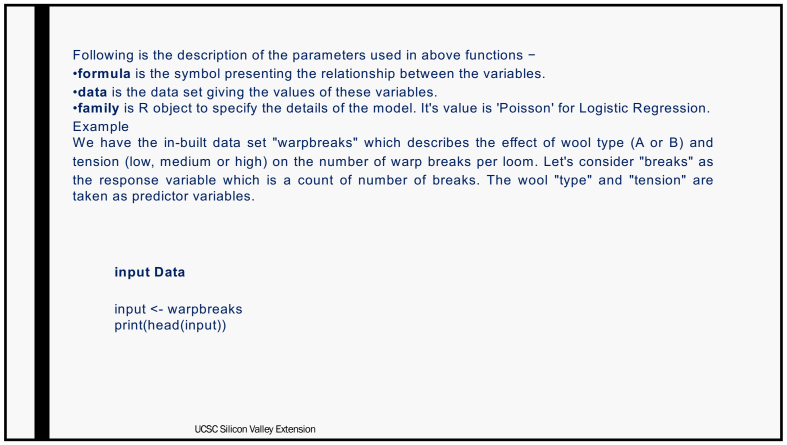
- **y** is the response variable.
- **a** and **b** are the numeric coefficients.
- **x** is the predictor variable.

The function used to create the Poisson regression model is the **glm()** function.

Syntax

The basic syntax for **glm()** function in Poisson regression is −

glm(formula,data,family)



Following is the description of the parameters used in above functions −

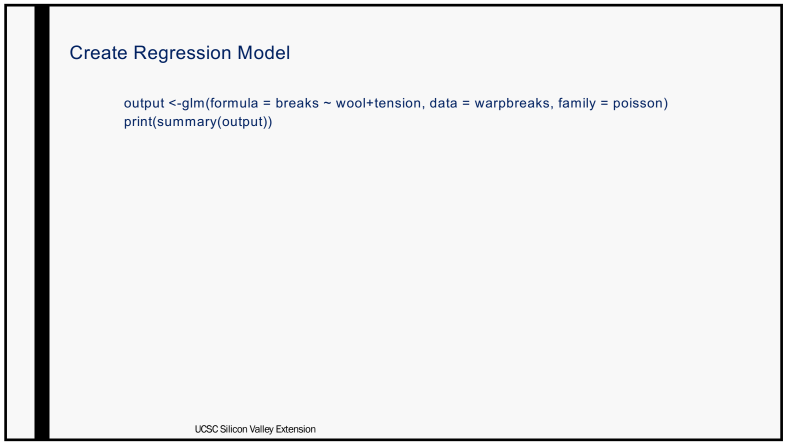
- **formula** is the symbol presenting the relationship between the variables.
- **data** is the data set giving the values of these variables.
- **family** is R object to specify the details of the model. It's value is 'Poisson' for Logistic Regression.

Example

We have the in-built data set "warpbreaks" which describes the effect of wool type (A or B) and tension (low, medium or high) on the number of warp breaks per loom. Let's consider "breaks" as the response variable which is a count of number of breaks. The wool "type" and "tension" are taken as predictor variables.

input Data

```
input <- warpbreaks
print(head(input))
```

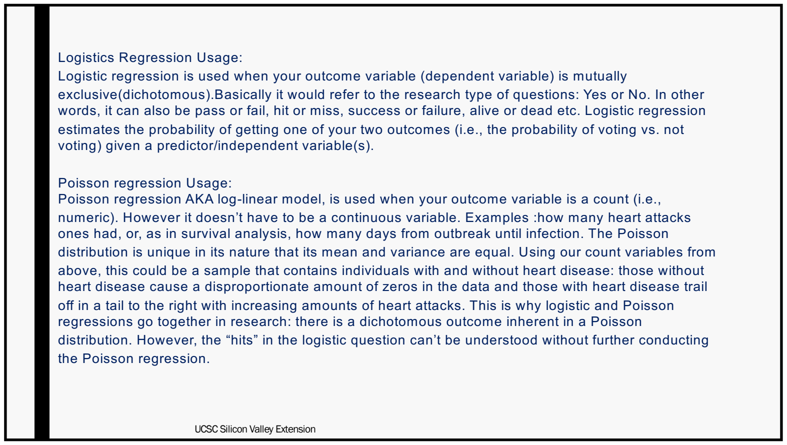


Create Regression Model

```
output <-glm(formula = breaks ~ wool+tension, data = warpbreaks, family = poisson)
print(summary(output))
```

UCSC Silicon Valley Extension

37

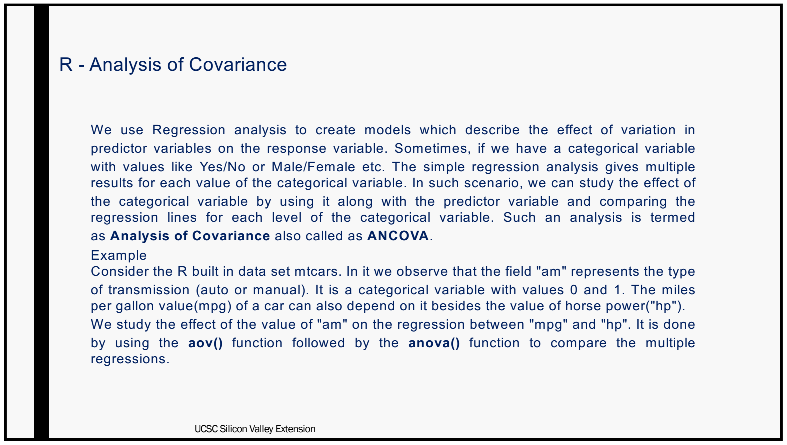


Logistics Regression Usage:
Logistic regression is used when your outcome variable (dependent variable) is mutually exclusive(dichotomous).Basically it would refer to the research type of questions: Yes or No. In other words, it can also be pass or fail, hit or miss, success or failure, alive or dead etc. Logistic regression estimates the probability of getting one of your two outcomes (i.e., the probability of voting vs. not voting) given a predictor/independent variable(s).

Poisson regression Usage:
Poisson regression AKA log-linear model, is used when your outcome variable is a count (i.e., numeric). However it doesn't have to be a continuous variable. Examples :how many heart attacks ones had, or, as in survival analysis, how many days from outbreak until infection. The Poisson distribution is unique in its nature that its mean and variance are equal. Using our count variables from above, this could be a sample that contains individuals with and without heart disease: those without heart disease cause a disproportionate amount of zeros in the data and those with heart disease trail off in a tail to the right with increasing amounts of heart attacks. This is why logistic and Poisson regressions go together in research: there is a dichotomous outcome inherent in a Poisson distribution. However, the “hits” in the logistic question can't be understood without further conducting the Poisson regression.

UCSC Silicon Valley Extension

38

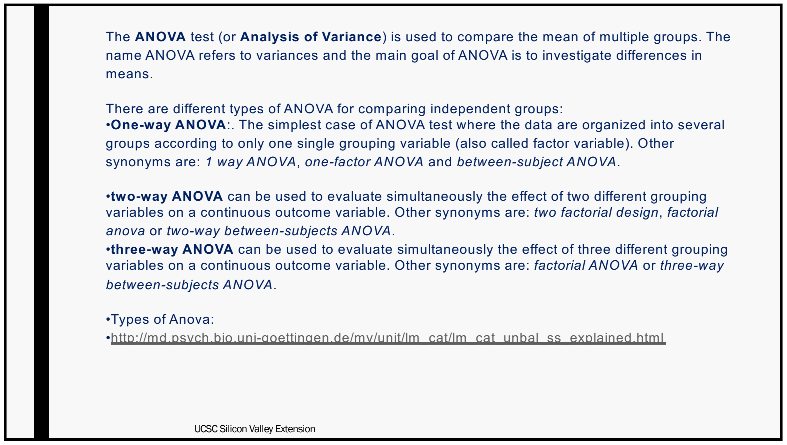


R - Analysis of Covariance

We use Regression analysis to create models which describe the effect of variation in predictor variables on the response variable. Sometimes, if we have a categorical variable with values like Yes/No or Male/Female etc. The simple regression analysis gives multiple results for each value of the categorical variable. In such scenario, we can study the effect of the categorical variable by using it along with the predictor variable and comparing the regression lines for each level of the categorical variable. Such an analysis is termed as **Analysis of Covariance** also called as **ANCOVA**.

Example

Consider the R built in data set mtcars. In it we observe that the field "am" represents the type of transmission (auto or manual). It is a categorical variable with values 0 and 1. The miles per gallon value(mpg) of a car can also depend on it besides the value of horse power("hp"). We study the effect of the value of "am" on the regression between "mpg" and "hp". It is done by using the **aov()** function followed by the **anova()** function to compare the multiple regressions.



The **ANOVA** test (or **Analysis of Variance**) is used to compare the mean of multiple groups. The name ANOVA refers to variances and the main goal of ANOVA is to investigate differences in means.

There are different types of ANOVA for comparing independent groups:

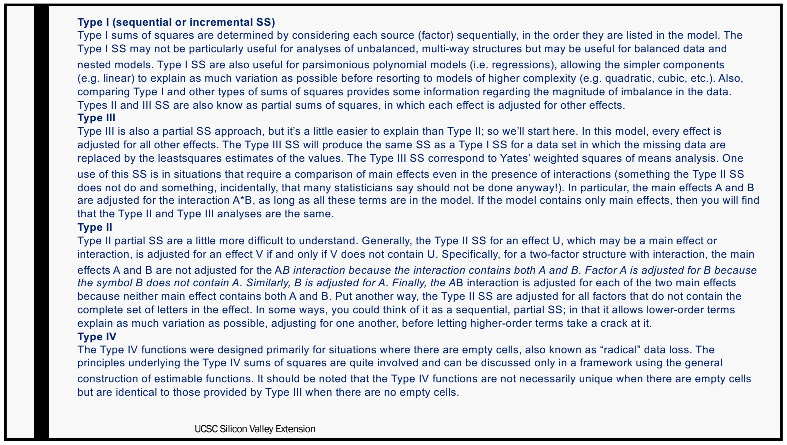
•**One-way ANOVA**:. The simplest case of ANOVA test where the data are organized into several groups according to only one single grouping variable (also called factor variable). Other synonyms are: *1 way ANOVA*, *one-factor ANOVA* and *between-subject ANOVA*.

•**two-way ANOVA** can be used to evaluate simultaneously the effect of two different grouping variables on a continuous outcome variable. Other synonyms are: *two factorial design*, *factorial anova* or *two-way between-subjects ANOVA*.

•**three-way ANOVA** can be used to evaluate simultaneously the effect of three different grouping variables on a continuous outcome variable. Other synonyms are: *factorial ANOVA* or *three-way between-subjects ANOVA*.

•Types of Anova:

•http://md.psych.bio.uni-goettingen.de/mv/unit/lm_cat/lm_cat_unbal_ss_explained.html



Type I (sequential or incremental SS)

Type I sums of squares are determined by considering each source (factor) sequentially, in the order they are listed in the model. The Type I SS may not be particularly useful for analyses of unbalanced, multi-way structures but may be useful for balanced data and nested models. Type I SS are also useful for parsimonious polynomial models (i.e. regressions), allowing the simpler components (e.g. linear) to explain as much variation as possible before resorting to models of higher complexity (e.g. quadratic, cubic, etc.). Also, comparing Type I and other types of sums of squares provides some information regarding the magnitude of imbalance in the data. Types II and III SS are also know as partial sums of squares, in which each effect is adjusted for other effects.

Type III

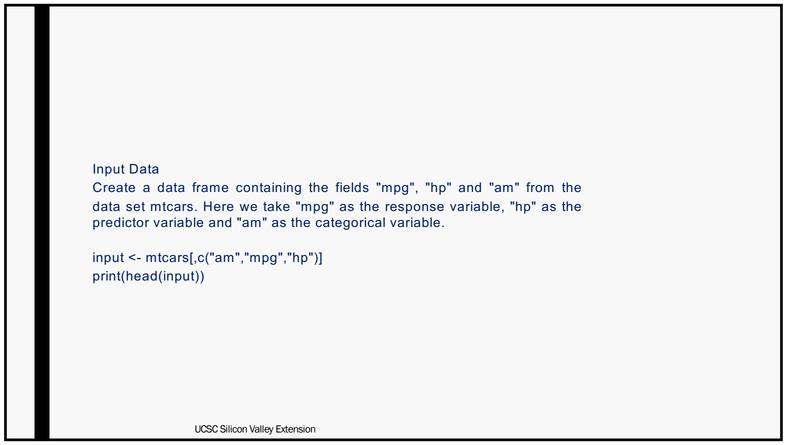
Type III is also a partial SS approach, but it’s a little easier to explain than Type II; so we’ll start here. In this model, every effect is adjusted for all other effects. The Type III SS will produce the same SS as a Type I SS for a data set in which the missing data are replaced by the leastsquares estimates of the values. The Type III SS correspond to Yates’ weighted squares of means analysis. One use of this SS is in situations that require a comparison of main effects even in the presence of interactions (something the Type II SS does not do and something, incidentally, that many statisticians say should not be done anyway!). In particular, the main effects A and B are adjusted for the interaction A*B, as long as all these terms are in the model. If the model contains only main effects, then you will find that the Type II and Type III analyses are the same.

Type II

Type II partial SS are a little more difficult to understand. Generally, the Type II SS for an effect U, which may be a main effect or interaction, is adjusted for an effect V if and only if V does not contain U. Specifically, for a two-factor structure with interaction, the main effects A and B are not adjusted for the A*B interaction because the interaction contains both A and B. Factor A is adjusted for B because the symbol B does not contain A. Similarly, B is adjusted for A. Finally, the A*B interaction is adjusted for each of the two main effects because neither main effect contains both A and B. Put another way, the Type II SS are adjusted for all factors that do not contain the complete set of letters in the effect. In some ways, you could think of it as a sequential, partial SS; in that it allows lower-order terms explain as much variation as possible, adjusting for one another, before letting higher-order terms take a crack at it.

Type IV

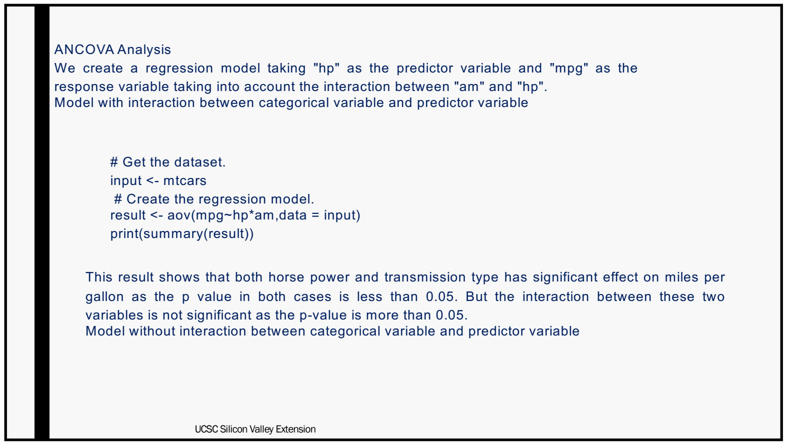
The Type IV functions were designed primarily for situations where there are empty cells, also known as “radical” data loss. The principles underlying the Type IV sums of squares are quite involved and can be discussed only in a framework using the general construction of estimable functions. It should be noted that the Type IV functions are not necessarily unique when there are empty cells but are identical to those provided by Type III when there are no empty cells.



Input Data

Create a data frame containing the fields "mpg", "hp" and "am" from the data set mtcars. Here we take "mpg" as the response variable, "hp" as the predictor variable and "am" as the categorical variable.

```
input <- mtcars[,c("am","mpg","hp")]
print(head(input))
```

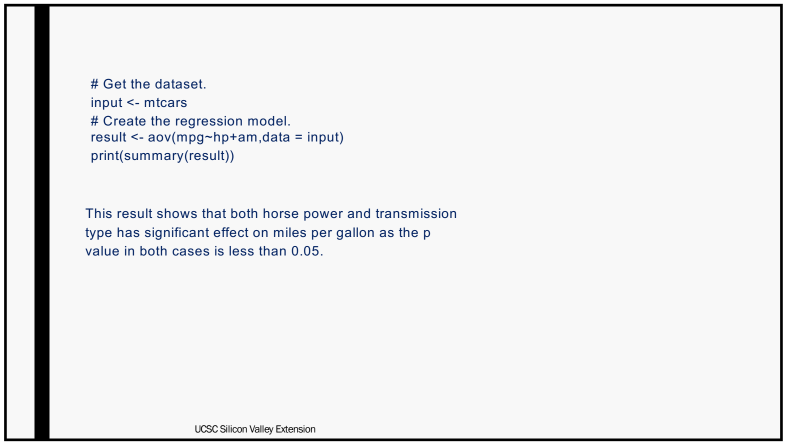


ANCOVA Analysis
We create a regression model taking "hp" as the predictor variable and "mpg" as the response variable taking into account the interaction between "am" and "hp".
Model with interaction between categorical variable and predictor variable

```
# Get the dataset.
input <- mtcars
 # Create the regression model.
result <- aov(mpg~hp*am,data = input)
print(summary(result))
```

This result shows that both horse power and transmission type has significant effect on miles per gallon as the p value in both cases is less than 0.05. But the interaction between these two variables is not significant as the p-value is more than 0.05.
Model without interaction between categorical variable and predictor variable

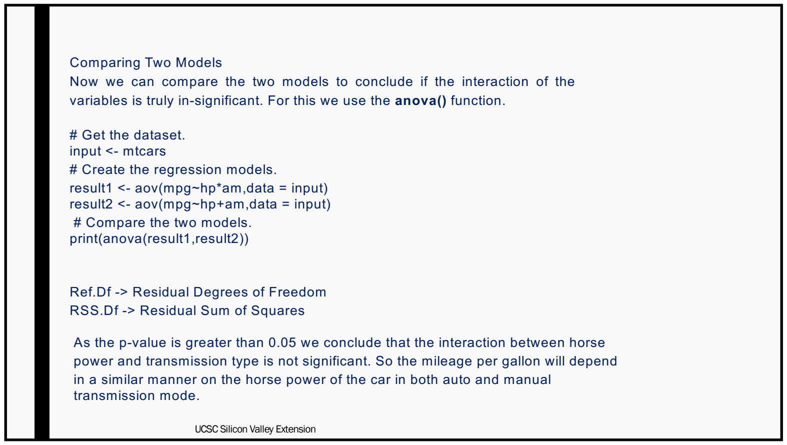
43



```
# Get the dataset.
input <- mtcars
# Create the regression model.
result <- aov(mpg~hp+am,data = input)
print(summary(result))
```

This result shows that both horse power and transmission type has significant effect on miles per gallon as the p value in both cases is less than 0.05.

44



Comparing Two Models

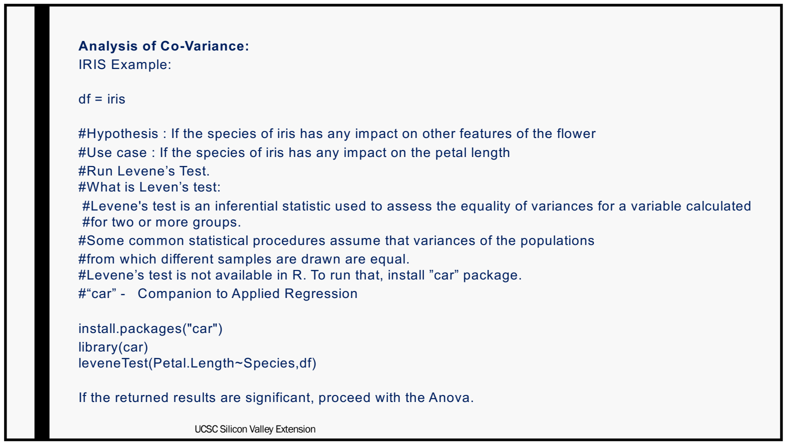
Now we can compare the two models to conclude if the interaction of the variables is truly in-significant. For this we use the **anova()** function.

```
# Get the dataset.
input <- mtcars
# Create the regression models.
result1 <- aov(mpg~hp*am,data = input)
result2 <- aov(mpg~hp+am,data = input)
 # Compare the two models.
print(anova(result1,result2))
```

Ref.Df -> Residual Degrees of Freedom

RSS.Df -> Residual Sum of Squares

As the p-value is greater than 0.05 we conclude that the interaction between horse power and transmission type is not significant. So the mileage per gallon will depend in a similar manner on the horse power of the car in both auto and manual transmission mode.

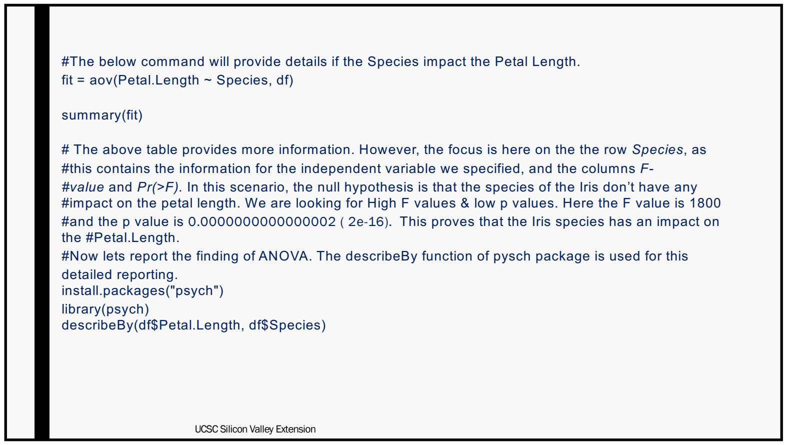


Analysis of Co-Variance:

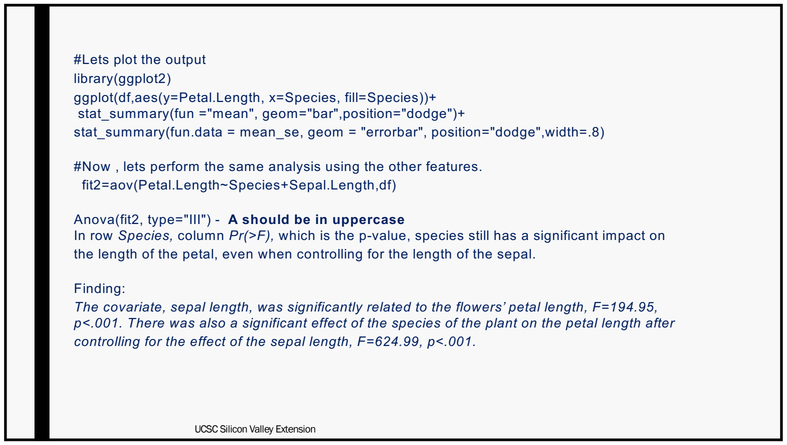
IRIS Example:

```
df = iris

#Hypothesis : If the species of iris has any impact on other features of the flower
#Use case : If the species of iris has any impact on the petal length
#Run Levene's Test.
#What is Leven's test:
 #Levene's test is an inferential statistic used to assess the equality of variances for a variable calculated
 #for two or more groups.
#Some common statistical procedures assume that variances of the populations
#from which different samples are drawn are equal.
#Levene's test is not available in R. To run that, install "car" package.
#"car" -   Companion to Applied Regression

install.packages("car")
library(car)
leveneTest(Petal.Length~Species,df)
```

If the returned results are significant, proceed with the Anova.



```
#The below command will provide details if the Species impact the Petal Length.
fit = aov(Petal.Length ~ Species, df)

summary(fit)

# The above table provides more information. However, the focus is here on the the row Species, as
#this contains the information for the independent variable we specified, and the columns F-
#value and Pr(>F). In this scenario, the null hypothesis is that the species of the Iris don't have any
#impact on the petal length. We are looking for High F values & low p values. Here the F value is 1800
#and the p value is 0.0000000000000002 ( 2e-16).  This proves that the Iris species has an impact on
the #Petal.Length.
#Now lets report the finding of ANOVA. The describeBy function of pysch package is used for this
detailed reporting.
install.packages("psych")
library(psych)
describeBy(df$Petal.Length, df$Species)
```

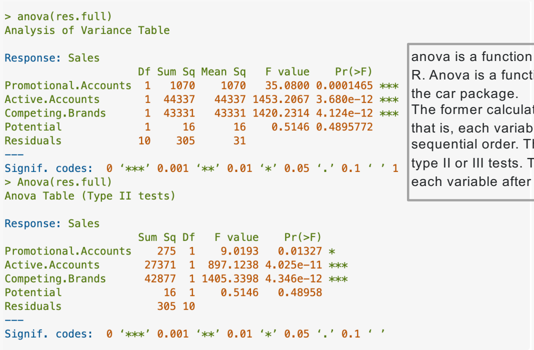


```
#Lets plot the output
library(ggplot2)
ggplot(df,aes(y=Petal.Length, x=Species, fill=Species))+
 stat_summary(fun ="mean", geom="bar",position="dodge")+
stat_summary(fun.data = mean_se, geom = "errorbar", position="dodge",width=.8)

#Now , lets perform the same analysis using the other features.
  fit2=aov(Petal.Length~Species+Sepal.Length,df)
```

Anova(fit2, type="III") - **A should be in uppercase**
In row *Species*, column *Pr(>F)*, which is the p-value, species still has a significant impact on the length of the petal, even when controlling for the length of the sepal.

Finding:
The covariate, sepal length, was significantly related to the flowers' petal length, F=194.95, p<.001. There was also a significant effect of the species of the plant on the petal length after controlling for the effect of the sepal length, F=624.99, p<.001.

Difference between anova & Anova function



```
> anova(res.full)
Analysis of Variance Table

Response: Sales
                     Df Sum Sq Mean Sq   F value    Pr(>F)
Promotional.Accounts  1   1070    1070   35.0800 0.0001465 ***
Active.Accounts       1  44337   44337 1453.2067 3.680e-12 ***
Competing.Brands      1  43331   43331 1420.2314 4.124e-12 ***
Potential             1     16      16    0.5146 0.4895772
Residuals            10    305      31
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
> Anova(res.full)
Anova Table (Type II tests)

Response: Sales
                     Sum Sq Df   F value    Pr(>F)
Promotional.Accounts    275  1    9.0193   0.01327 *
Active.Accounts       27371  1  897.1238 4.025e-11 ***
Competing.Brands      42877  1 1405.3398 4.346e-12 ***
Potential                16  1    0.5146   0.48958
Residuals               305 10
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’
```

anova is a function in base R. Anova is a function in the car package.
The former calculates type I tests, that is, each variable is added in sequential order. The latter calculates type II or III tests. Type II tests test each variable after all the others.

R - Time Series Analysis

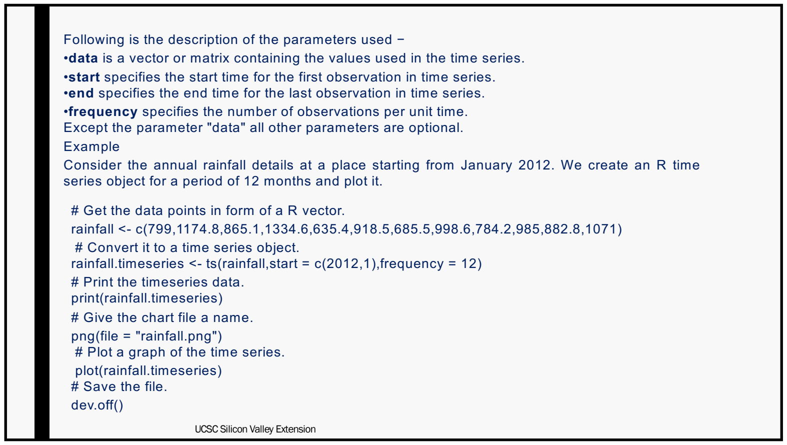
Time series is a series of data points in which each data point is associated with a timestamp. A simple example is the price of a stock in the stock market at different points of time on a given day. Another example is the amount of rainfall in a region at different months of the year. R language uses many functions to create, manipulate and plot the time series data. The data for the time series is stored in an R object called **time-series object**. It is also a R data object like a vector or data frame.

The time series object is created by using the **ts()** function.

Syntax

The basic syntax for **ts()** function in time series analysis is −

timeseries.object.name <- ts(data, start, end, frequency)



Following is the description of the parameters used −

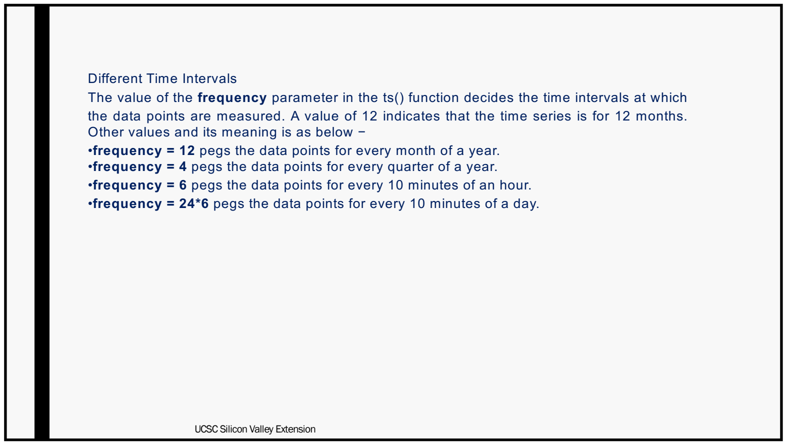
- **data** is a vector or matrix containing the values used in the time series.
- **start** specifies the start time for the first observation in time series.
- **end** specifies the end time for the last observation in time series.
- **frequency** specifies the number of observations per unit time.

Except the parameter "data" all other parameters are optional.

Example

Consider the annual rainfall details at a place starting from January 2012. We create an R time series object for a period of 12 months and plot it.

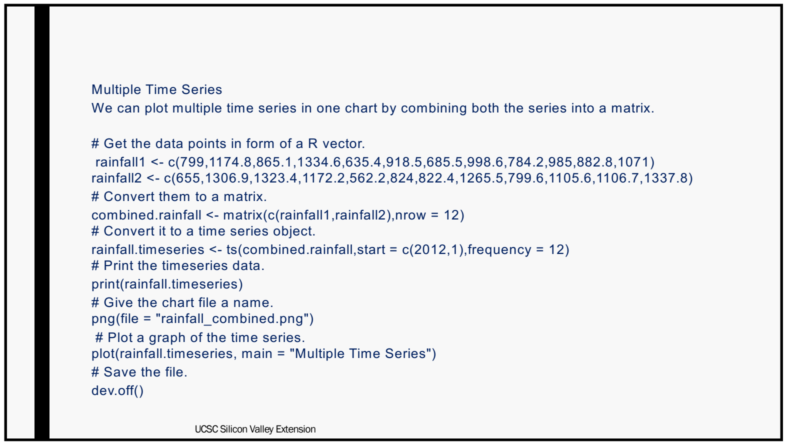
```
# Get the data points in form of a R vector.
rainfall <- c(799,1174.8,865.1,1334.6,635.4,918.5,685.5,998.6,784.2,985,882.8,1071)
 # Convert it to a time series object.
rainfall.timeseries <- ts(rainfall,start = c(2012,1),frequency = 12)
# Print the timeseries data.
print(rainfall.timeseries)
# Give the chart file a name.
png(file = "rainfall.png")
 # Plot a graph of the time series.
 plot(rainfall.timeseries)
# Save the file.
dev.off()
```



Different Time Intervals

The value of the **frequency** parameter in the ts() function decides the time intervals at which the data points are measured. A value of 12 indicates that the time series is for 12 months. Other values and its meaning is as below −

- **frequency = 12** pegs the data points for every month of a year.
- **frequency = 4** pegs the data points for every quarter of a year.
- **frequency = 6** pegs the data points for every 10 minutes of an hour.
- **frequency = 24*6** pegs the data points for every 10 minutes of a day.

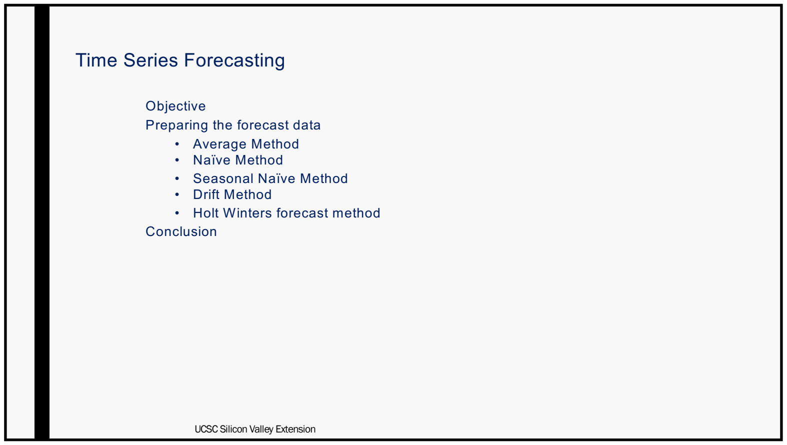


Multiple Time Series

We can plot multiple time series in one chart by combining both the series into a matrix.

```
# Get the data points in form of a R vector.
 rainfall1 <- c(799,1174.8,865.1,1334.6,635.4,918.5,685.5,998.6,784.2,985,882.8,1071)
rainfall2 <- c(655,1306.9,1323.4,1172.2,562.2,824,822.4,1265.5,799.6,1105.6,1106.7,1337.8)
# Convert them to a matrix.
combined.rainfall <- matrix(c(rainfall1,rainfall2),nrow = 12)
# Convert it to a time series object.
rainfall.timeseries <- ts(combined.rainfall,start = c(2012,1),frequency = 12)
# Print the timeseries data.
print(rainfall.timeseries)
# Give the chart file a name.
png(file = "rainfall_combined.png")
 # Plot a graph of the time series.
plot(rainfall.timeseries, main = "Multiple Time Series")
# Save the file.
dev.off()
```

53



Time Series Forecasting

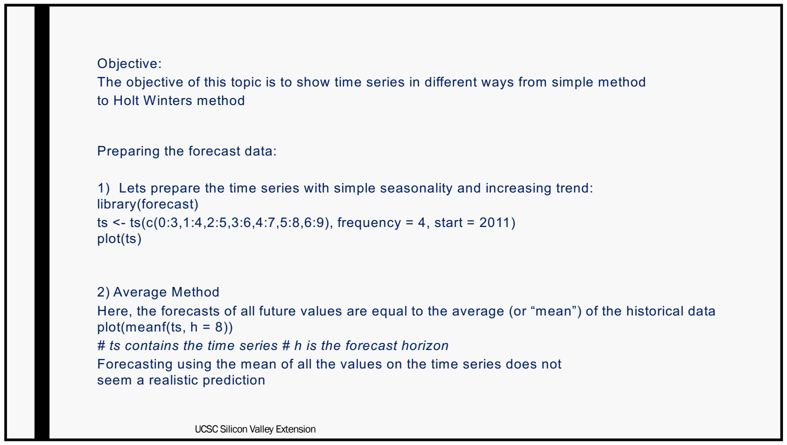
Objective

Preparing the forecast data

- Average Method
- Naïve Method
- Seasonal Naïve Method
- Drift Method
- Holt Winters forecast method

Conclusion

54



Objective:
The objective of this topic is to show time series in different ways from simple method to Holt Winters method

Preparing the forecast data:

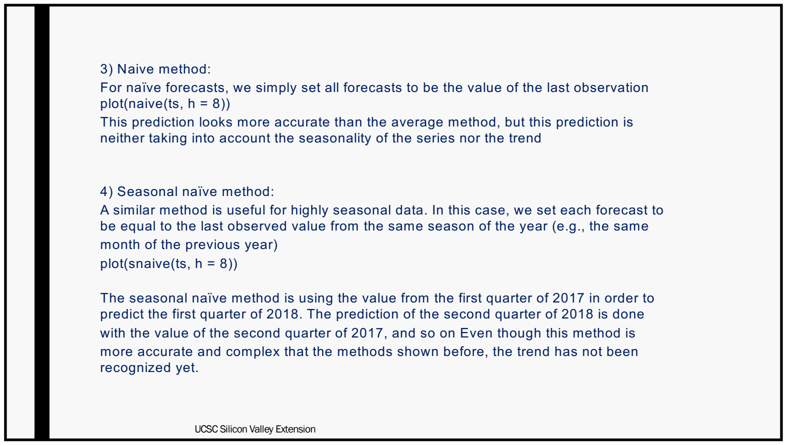
1) Lets prepare the time series with simple seasonality and increasing trend:

```
library(forecast)
ts <- ts(c(0:3,1:4,2:5,3:6,4:7,5:8,6:9), frequency = 4, start = 2011)
plot(ts)
```

2) Average Method
Here, the forecasts of all future values are equal to the average (or “mean”) of the historical data

```
plot(meanf(ts, h = 8))
# ts contains the time series # h is the forecast horizon
```

Forecasting using the mean of all the values on the time series does not seem a realistic prediction



3) Naive method:
For naïve forecasts, we simply set all forecasts to be the value of the last observation

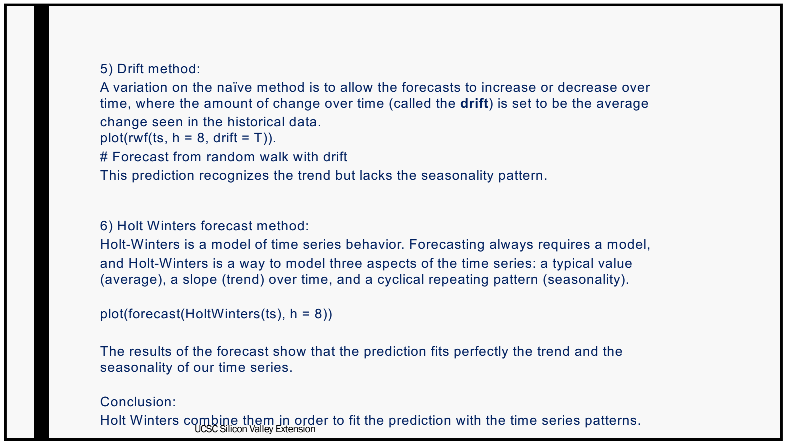
```
plot(naive(ts, h = 8))
```

This prediction looks more accurate than the average method, but this prediction is neither taking into account the seasonality of the series nor the trend

4) Seasonal naïve method:
A similar method is useful for highly seasonal data. In this case, we set each forecast to be equal to the last observed value from the same season of the year (e.g., the same month of the previous year)

```
plot(snaive(ts, h = 8))
```

The seasonal naïve method is using the value from the first quarter of 2017 in order to predict the first quarter of 2018. The prediction of the second quarter of 2018 is done with the value of the second quarter of 2017, and so on Even though this method is more accurate and complex that the methods shown before, the trend has not been recognized yet.



5) Drift method:

A variation on the naïve method is to allow the forecasts to increase or decrease over time, where the amount of change over time (called the **drift**) is set to be the average change seen in the historical data.

```
plot(rwf(ts, h = 8, drift = T)).
# Forecast from random walk with drift
```

This prediction recognizes the trend but lacks the seasonality pattern.

6) Holt Winters forecast method:

Holt-Winters is a model of time series behavior. Forecasting always requires a model, and Holt-Winters is a way to model three aspects of the time series: a typical value (average), a slope (trend) over time, and a cyclical repeating pattern (seasonality).

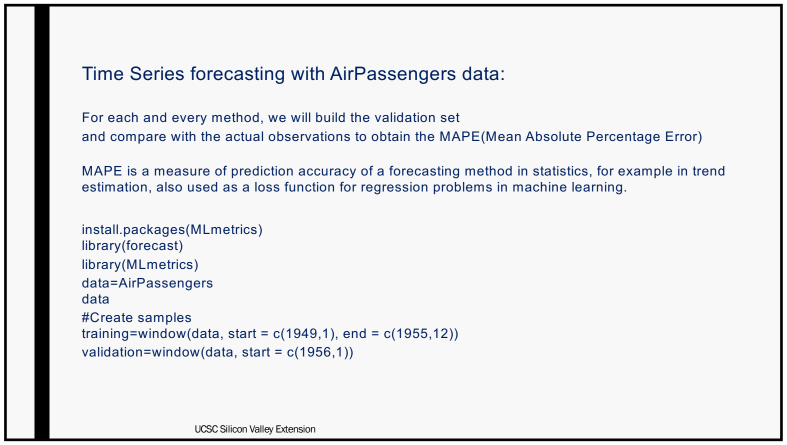
```
plot(forecast(HoltWinters(ts), h = 8))
```

The results of the forecast show that the prediction fits perfectly the trend and the seasonality of our time series.

Conclusion:

Holt Winters combine them in order to fit the prediction with the time series patterns.

57



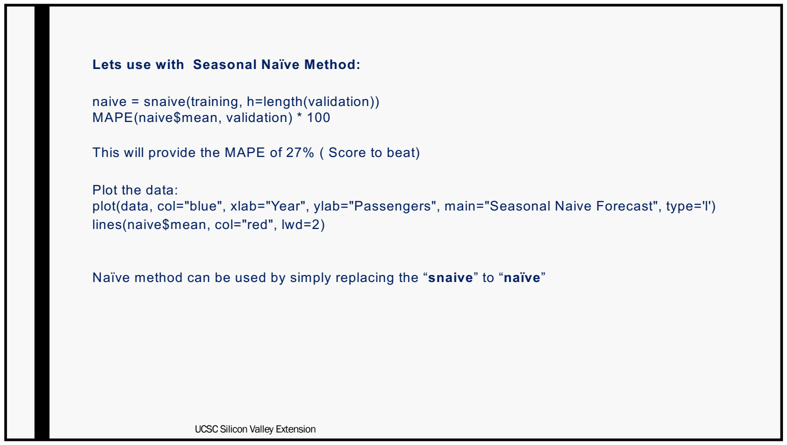
Time Series forecasting with AirPassengers data:

For each and every method, we will build the validation set
and compare with the actual observations to obtain the MAPE(Mean Absolute Percentage Error)

MAPE is a measure of prediction accuracy of a forecasting method in statistics, for example in trend estimation, also used as a loss function for regression problems in machine learning.

```
install.packages(MLmetrics)
library(forecast)
library(MLmetrics)
data=AirPassengers
data
#Create samples
training=window(data, start = c(1949,1), end = c(1955,12))
validation=window(data, start = c(1956,1))
```

58



Lets use with Seasonal Naïve Method:

```
naive = snaive(training, h=length(validation))
MAPE(naive$mean, validation) * 100
```

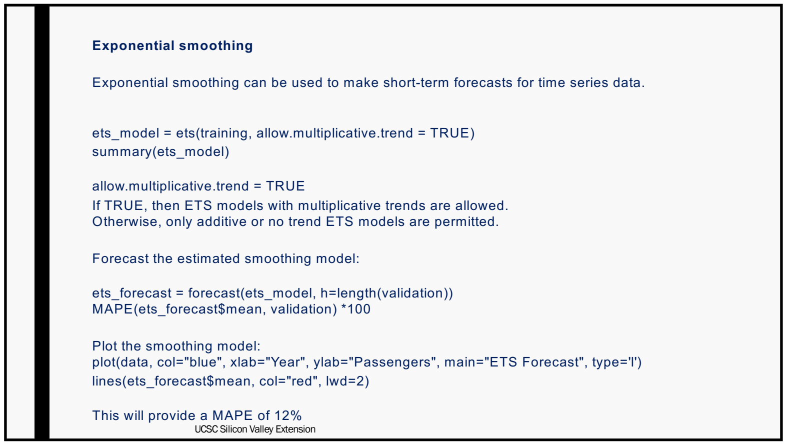
This will provide the MAPE of 27% (Score to beat)

Plot the data:

```
plot(data, col="blue", xlab="Year", ylab="Passengers", main="Seasonal Naive Forecast", type='l')
lines(naive$mean, col="red", lwd=2)
```

Naïve method can be used by simply replacing the “**snaive**” to “**naïve**”

59



Exponential smoothing

Exponential smoothing can be used to make short-term forecasts for time series data.

```
ets_model = ets(training, allow.multiplicative.trend = TRUE)
summary(ets_model)
```

allow.multiplicative.trend = TRUE
If TRUE, then ETS models with multiplicative trends are allowed.
Otherwise, only additive or no trend ETS models are permitted.

Forecast the estimated smoothing model:

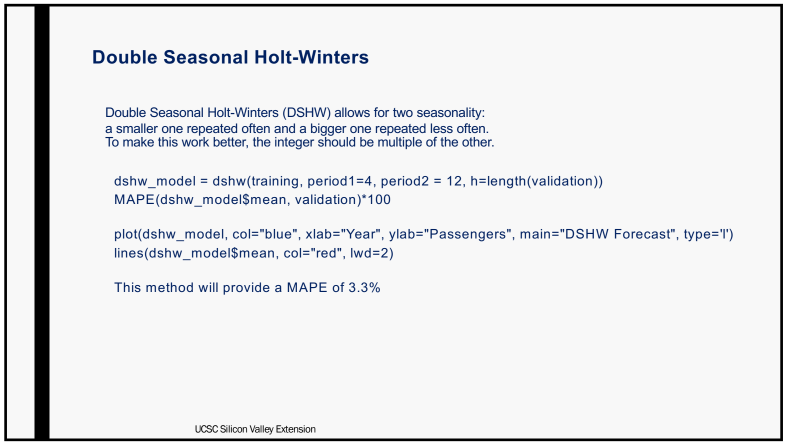
```
ets_forecast = forecast(ets_model, h=length(validation))
MAPE(ets_forecast$mean, validation) *100
```

Plot the smoothing model:

```
plot(data, col="blue", xlab="Year", ylab="Passengers", main="ETS Forecast", type='l')
lines(ets_forecast$mean, col="red", lwd=2)
```

This will provide a MAPE of 12%

60



Double Seasonal Holt-Winters

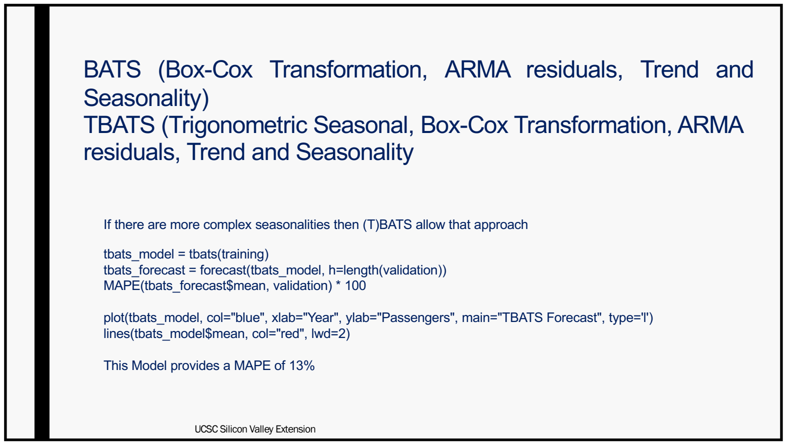
Double Seasonal Holt-Winters (DSHW) allows for two seasonality:
a smaller one repeated often and a bigger one repeated less often.
To make this work better, the integer should be multiple of the other.

```
dshw_model = dshw(training, period1=4, period2 = 12, h=length(validation))
MAPE(dshw_model$mean, validation)*100

plot(dshw_model, col="blue", xlab="Year", ylab="Passengers", main="DSHW Forecast", type='l')
lines(dshw_model$mean, col="red", lwd=2)
```

This method will provide a MAPE of 3.3%

UCSC Silicon Valley Extension

61



BATS (Box-Cox Transformation, ARMA residuals, Trend and Seasonality) TBATS (Trigonometric Seasonal, Box-Cox Transformation, ARMA residuals, Trend and Seasonality

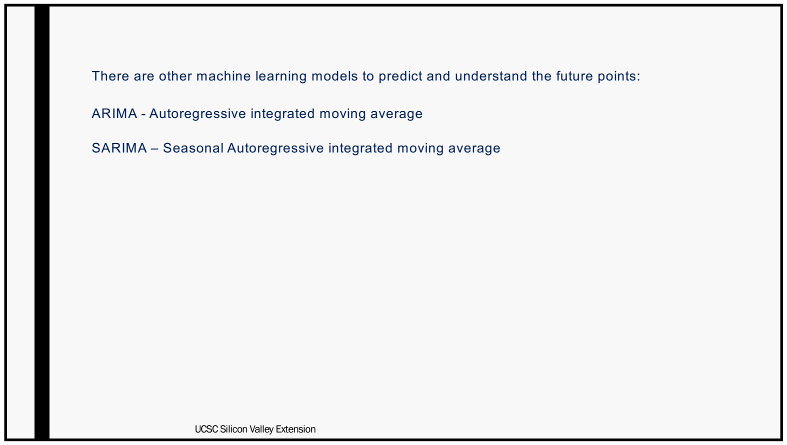
If there are more complex seasonalities then (T)BATS allow that approach

```
tbats_model = tbats(training)
tbats_forecast = forecast(tbats_model, h=length(validation))
MAPE(tbats_forecast$mean, validation) * 100

plot(tbats_model, col="blue", xlab="Year", ylab="Passengers", main="TBATS Forecast", type='l')
lines(tbats_model$mean, col="red", lwd=2)
```

This Model provides a MAPE of 13%

UCSC Silicon Valley Extension

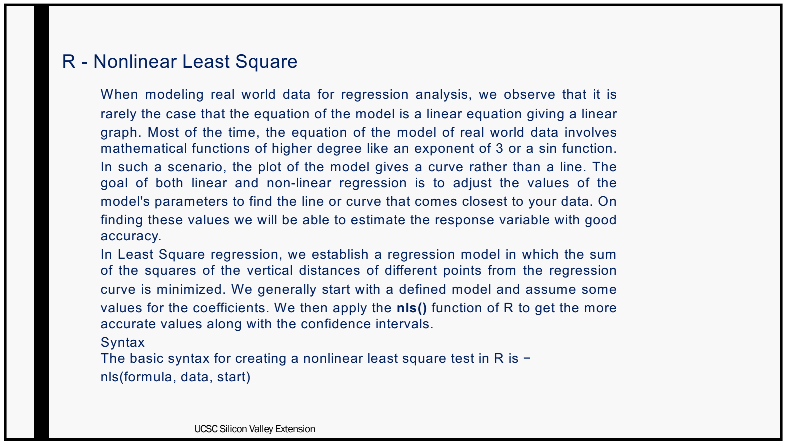
62



There are other machine learning models to predict and understand the future points:

ARIMA - Autoregressive integrated moving average

SARIMA – Seasonal Autoregressive integrated moving average



R - Nonlinear Least Square

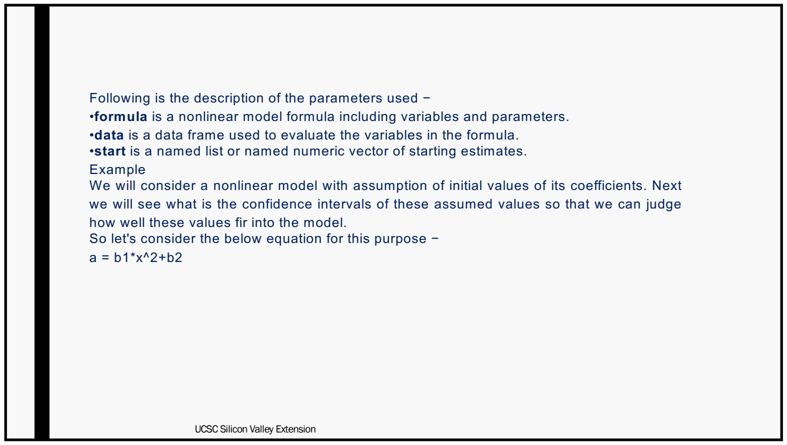
When modeling real world data for regression analysis, we observe that it is rarely the case that the equation of the model is a linear equation giving a linear graph. Most of the time, the equation of the model of real world data involves mathematical functions of higher degree like an exponent of 3 or a sin function. In such a scenario, the plot of the model gives a curve rather than a line. The goal of both linear and non-linear regression is to adjust the values of the model's parameters to find the line or curve that comes closest to your data. On finding these values we will be able to estimate the response variable with good accuracy.

In Least Square regression, we establish a regression model in which the sum of the squares of the vertical distances of different points from the regression curve is minimized. We generally start with a defined model and assume some values for the coefficients. We then apply the **nls()** function of R to get the more accurate values along with the confidence intervals.

Syntax

The basic syntax for creating a nonlinear least square test in R is −

nls(formula, data, start)



Following is the description of the parameters used −

- **formula** is a nonlinear model formula including variables and parameters.
- **data** is a data frame used to evaluate the variables in the formula.
- **start** is a named list or named numeric vector of starting estimates.

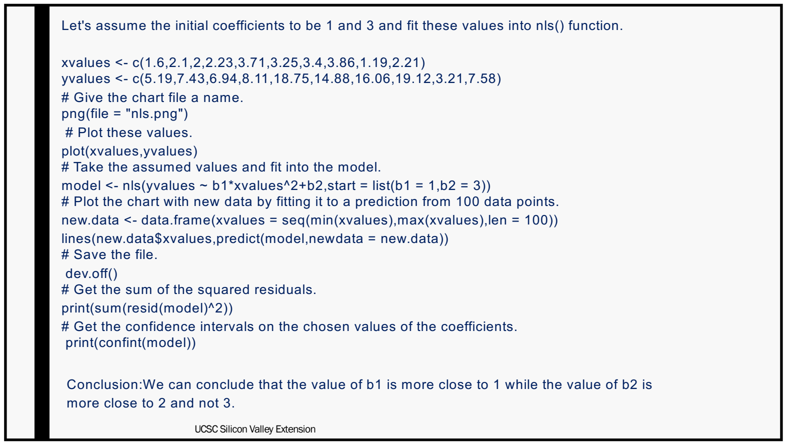
Example

We will consider a nonlinear model with assumption of initial values of its coefficients. Next we will see what is the confidence intervals of these assumed values so that we can judge how well these values fir into the model.

So let's consider the below equation for this purpose −

a = b1*x^2+b2

65

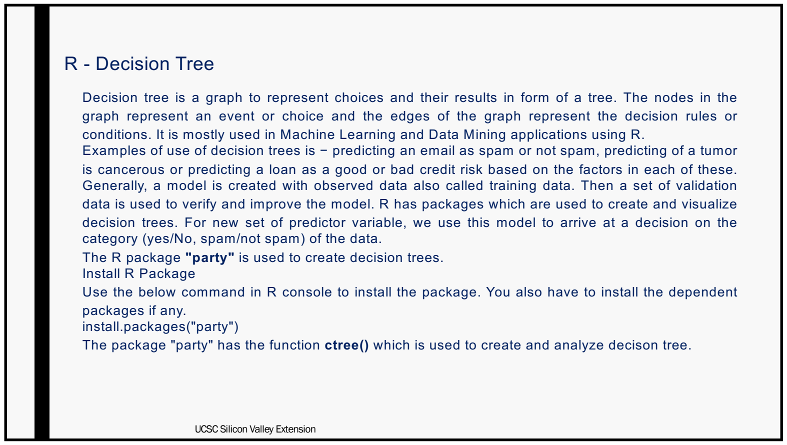


Let's assume the initial coefficients to be 1 and 3 and fit these values into nls() function.

```
xvalues <- c(1.6,2.1,2,2.23,3.71,3.25,3.4,3.86,1.19,2.21)
yvalues <- c(5.19,7.43,6.94,8.11,18.75,14.88,16.06,19.12,3.21,7.58)
# Give the chart file a name.
png(file = "nls.png")
 # Plot these values.
plot(xvalues,yvalues)
# Take the assumed values and fit into the model.
model <- nls(yvalues ~ b1*xvalues^2+b2,start = list(b1 = 1,b2 = 3))
# Plot the chart with new data by fitting it to a prediction from 100 data points.
new.data <- data.frame(xvalues = seq(min(xvalues),max(xvalues),len = 100))
lines(new.data$xvalues,predict(model,newdata = new.data))
# Save the file.
 dev.off()
# Get the sum of the squared residuals.
print(sum(resid(model)^2))
# Get the confidence intervals on the chosen values of the coefficients.
 print(confint(model))
```

Conclusion:We can conclude that the value of b1 is more close to 1 while the value of b2 is more close to 2 and not 3.

66



R - Decision Tree

Decision tree is a graph to represent choices and their results in form of a tree. The nodes in the graph represent an event or choice and the edges of the graph represent the decision rules or conditions. It is mostly used in Machine Learning and Data Mining applications using R.

Examples of use of decision trees is − predicting an email as spam or not spam, predicting of a tumor is cancerous or predicting a loan as a good or bad credit risk based on the factors in each of these. Generally, a model is created with observed data also called training data. Then a set of validation data is used to verify and improve the model. R has packages which are used to create and visualize decision trees. For new set of predictor variable, we use this model to arrive at a decision on the category (yes/No, spam/not spam) of the data.

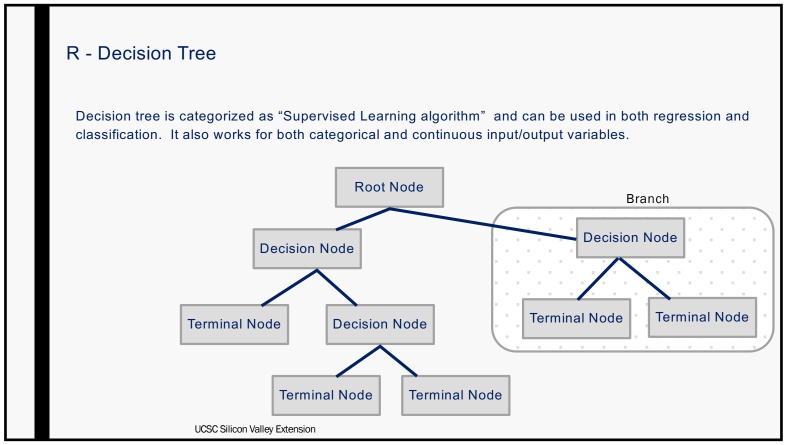
The R package **"party"** is used to create decision trees.

Install R Package

Use the below command in R console to install the package. You also have to install the dependent packages if any.

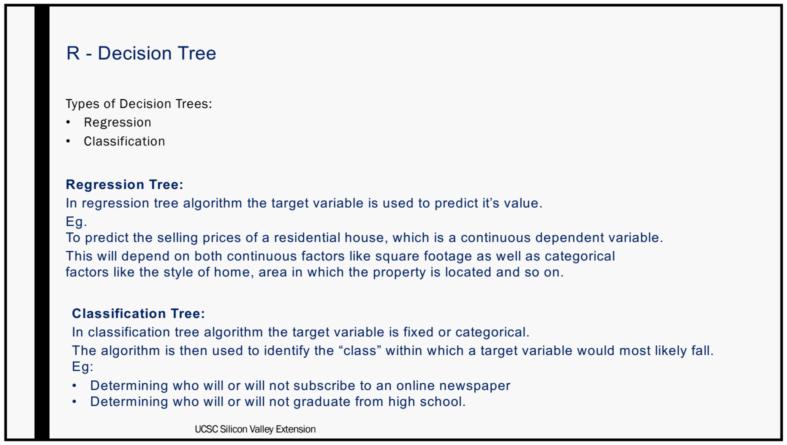
```
install.packages("party")
```

The package "party" has the function **ctree()** which is used to create and analyze decison tree.



R - Decision Tree

Decision tree is categorized as “Supervised Learning algorithm” and can be used in both regression and classification. It also works for both categorical and continuous input/output variables.



R - Decision Tree

Types of Decision Trees:

- Regression
- Classification

Regression Tree:

In regression tree algorithm the target variable is used to predict it's value.

Eg.

To predict the selling prices of a residential house, which is a continuous dependent variable.

This will depend on both continuous factors like square footage as well as categorical factors like the style of home, area in which the property is located and so on.

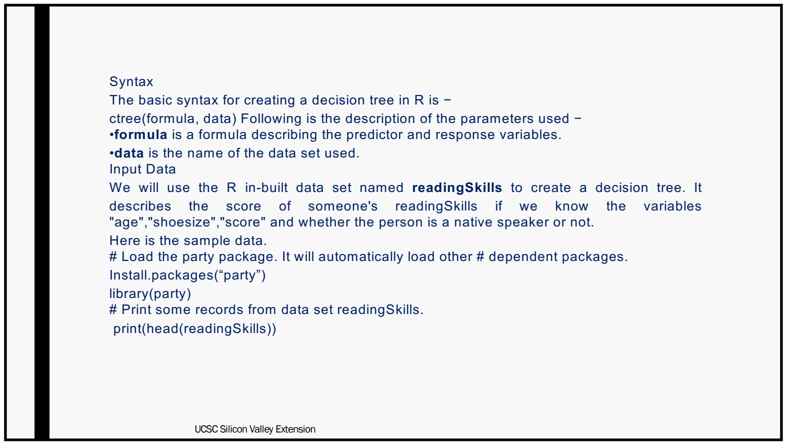
Classification Tree:

In classification tree algorithm the target variable is fixed or categorical.

The algorithm is then used to identify the "class" within which a target variable would most likely fall.

Eg:

- Determining who will or will not subscribe to an online newspaper
- Determining who will or will not graduate from high school.



Syntax

The basic syntax for creating a decision tree in R is −

ctree(formula, data) Following is the description of the parameters used −

•**formula** is a formula describing the predictor and response variables.

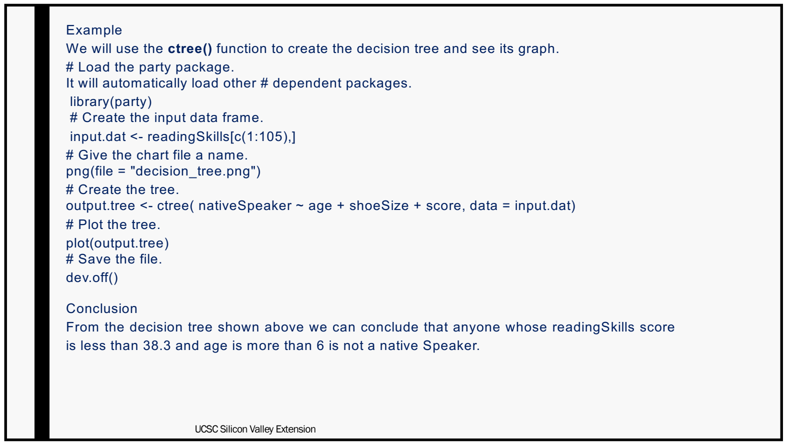
•**data** is the name of the data set used.

Input Data

We will use the R in-built data set named **readingSkills** to create a decision tree. It describes the score of someone's readingSkills if we know the variables "age","shoesize","score" and whether the person is a native speaker or not.

Here is the sample data.

```
# Load the party package. It will automatically load other # dependent packages.
Install.packages(“party”)
library(party)
# Print some records from data set readingSkills.
 print(head(readingSkills))
```



Example

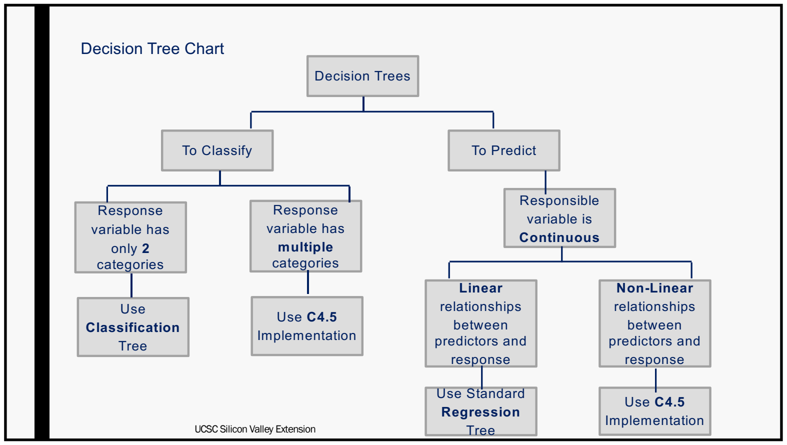
We will use the **ctree()** function to create the decision tree and see its graph.

```
# Load the party package.
It will automatically load other # dependent packages.
 library(party)
 # Create the input data frame.
 input.dat <- readingSkills[c(1:105),]
# Give the chart file a name.
png(file = "decision_tree.png")
# Create the tree.
output.tree <- ctree( nativeSpeaker ~ age + shoeSize + score, data = input.dat)
# Plot the tree.
plot(output.tree)
# Save the file.
dev.off()
```

Conclusion

From the decision tree shown above we can conclude that anyone whose readingSkills score is less than 38.3 and age is more than 6 is not a native Speaker.

71



Decision Tree Chart

Decision Trees

To Classify

To Predict

Response variable has only **2** categories

Response variable has **multiple** categories

Responsible variable is **Continuous**

Use **Classification** Tree

Use **C4.5** Implementation

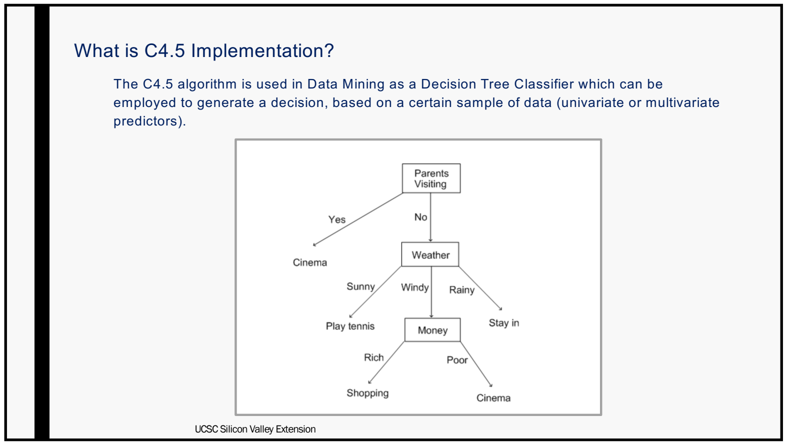
Linear relationships between predictors and response

Non-Linear relationships between predictors and response

Use Standard **Regression** Tree

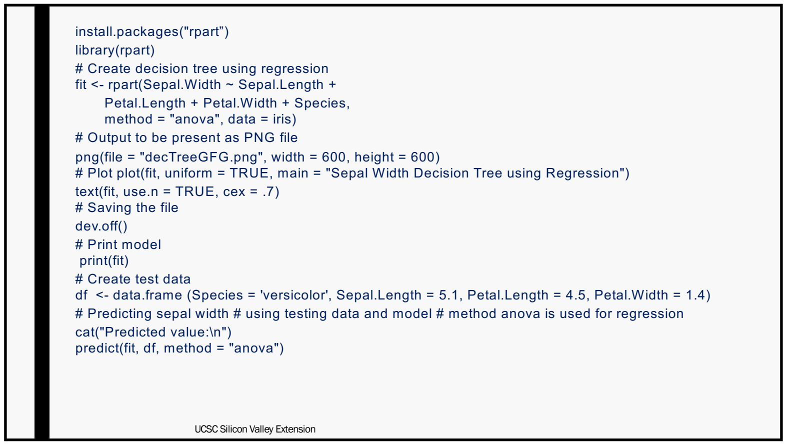
Use **C4.5** Implementation

72

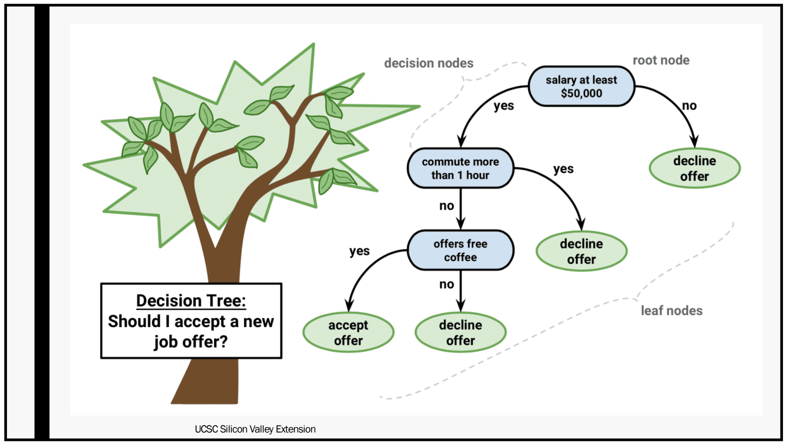


What is C4.5 Implementation?

The C4.5 algorithm is used in Data Mining as a Decision Tree Classifier which can be employed to generate a decision, based on a certain sample of data (univariate or multivariate predictors).



```
install.packages("rpart")
library(rpart)
# Create decision tree using regression
fit <- rpart(Sepal.Width ~ Sepal.Length +
      Petal.Length + Petal.Width + Species,
      method = "anova", data = iris)
# Output to be present as PNG file
png(file = "decTreeGFG.png", width = 600, height = 600)
# Plot plot(fit, uniform = TRUE, main = "Sepal Width Decision Tree using Regression")
text(fit, use.n = TRUE, cex = .7)
# Saving the file
dev.off()
# Print model
 print(fit)
# Create test data
df  <- data.frame (Species = 'versicolor', Sepal.Length = 5.1, Petal.Length = 4.5, Petal.Width = 1.4)
# Predicting sepal width # using testing data and model # method anova is used for regression
cat("Predicted value:\n")
predict(fit, df, method = "anova")
```



Decision Tree:
Should I accept a new job offer?

decision nodes

root node

leaf nodes

salary at least $50,000

yes

no

commute more than 1 hour

decline offer

no

yes

offers free coffee

decline offer

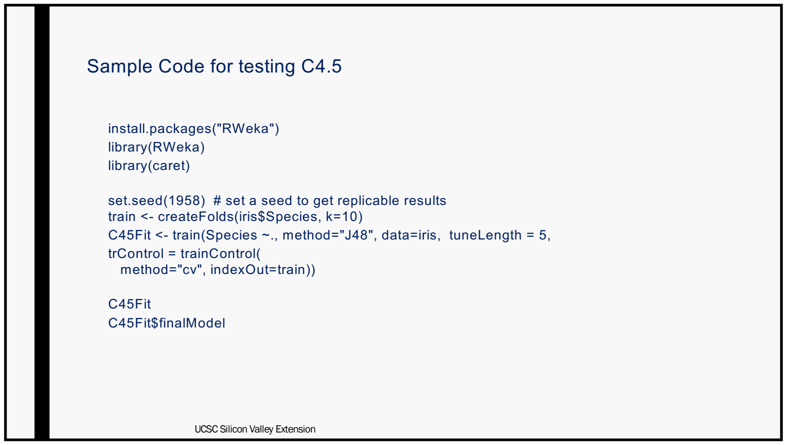
yes

no

accept offer

decline offer

75



Sample Code for testing C4.5

```
install.packages("RWeka")
library(RWeka)
library(caret)

set.seed(1958)  # set a seed to get replicable results
train <- createFolds(iris$Species, k=10)
C45Fit <- train(Species ~., method="J48", data=iris,  tuneLength = 5,
trControl = trainControl(
   method="cv", indexOut=train))

C45Fit
C45Fit$finalModel
```

76

R - Random Forest

In the random forest approach, a large number of decision trees are created. Every observation is fed into every decision tree. The most common outcome for each observation is used as the final output. A new observation is fed into all the trees and taking a majority vote for each classification model.

An error estimate is made for the cases which were not used while building the tree. That is called an **OOB (Out-of-bag)** error estimate which is mentioned as a percentage.

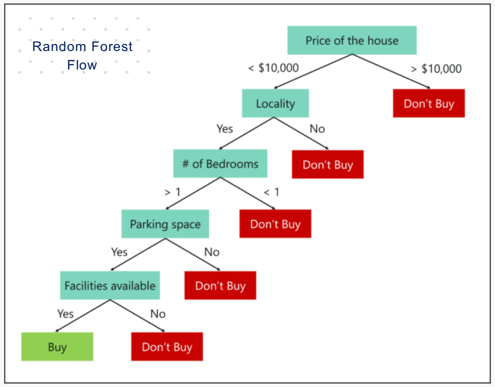
The R package **"randomForest"** is used to create random forests.

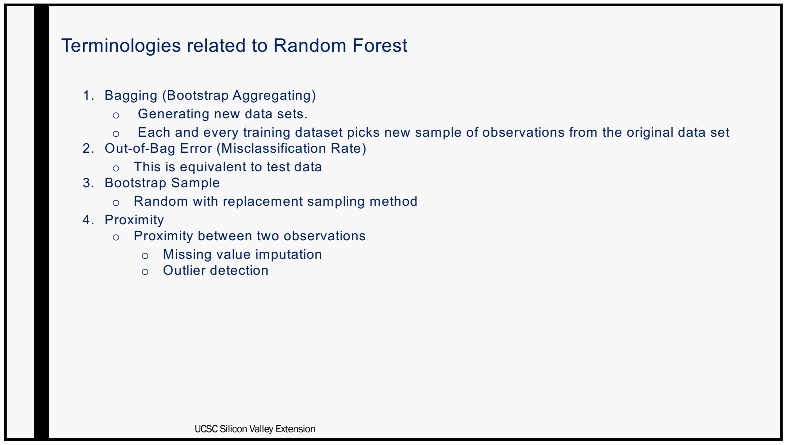
Install R Package

Use the below command in R console to install the package. You also have to install the dependent packages if any.

install.packages("randomForest")

The package "randomForest" has the function **randomForest()** which is used to create and analyze random forests.

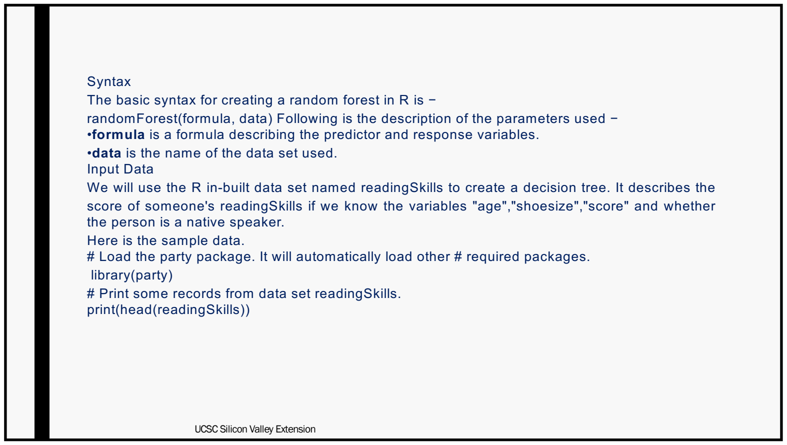




Terminologies related to Random Forest

1. Bagging (Bootstrap Aggregating)
   - Generating new data sets.
   - Each and every training dataset picks new sample of observations from the original data set
2. Out-of-Bag Error (Misclassification Rate)
   - This is equivalent to test data
3. Bootstrap Sample
   - Random with replacement sampling method
4. Proximity
   - Proximity between two observations
     - Missing value imputation
     - Outlier detection

79



Syntax

The basic syntax for creating a random forest in R is −

randomForest(formula, data) Following is the description of the parameters used −

•**formula** is a formula describing the predictor and response variables.

•**data** is the name of the data set used.

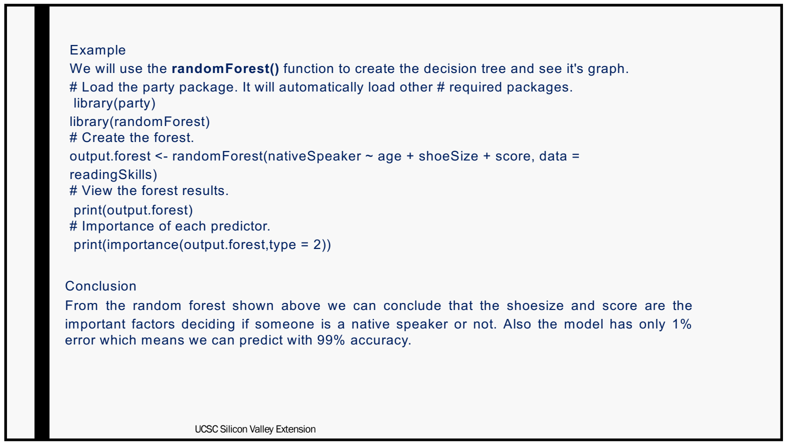
Input Data

We will use the R in-built data set named readingSkills to create a decision tree. It describes the score of someone's readingSkills if we know the variables "age","shoesize","score" and whether the person is a native speaker.

Here is the sample data.

```
# Load the party package. It will automatically load other # required packages.
 library(party)
# Print some records from data set readingSkills.
print(head(readingSkills))
```

80



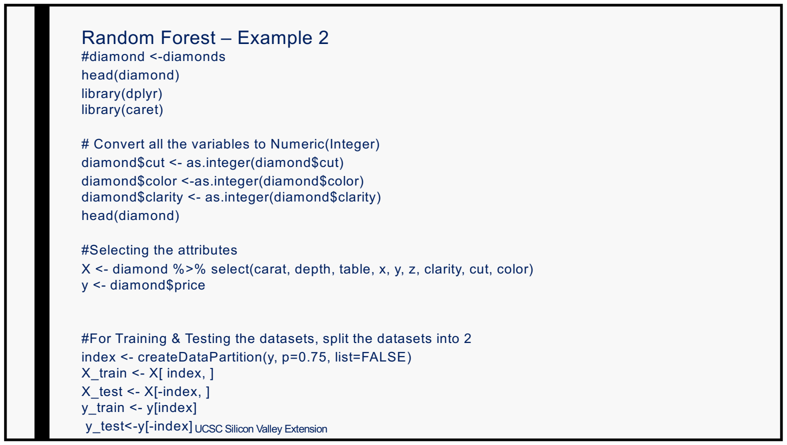
Example

We will use the **randomForest()** function to create the decision tree and see it's graph.

```
# Load the party package. It will automatically load other # required packages.
 library(party)
library(randomForest)
# Create the forest.
output.forest <- randomForest(nativeSpeaker ~ age + shoeSize + score, data =
readingSkills)
# View the forest results.
 print(output.forest)
# Importance of each predictor.
 print(importance(output.forest,type = 2))
```

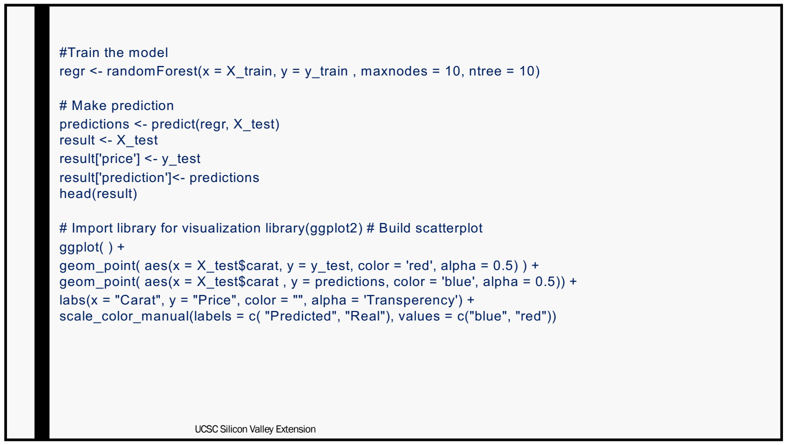
Conclusion

From the random forest shown above we can conclude that the shoesize and score are the important factors deciding if someone is a native speaker or not. Also the model has only 1% error which means we can predict with 99% accuracy.



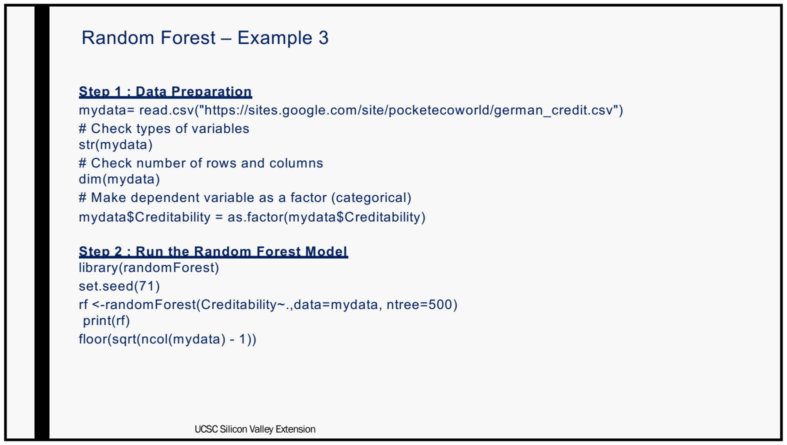
Random Forest – Example 2

```
#diamond <-diamonds
head(diamond)
library(dplyr)
library(caret)

# Convert all the variables to Numeric(Integer)
diamond$cut <- as.integer(diamond$cut)
diamond$color <-as.integer(diamond$color)
diamond$clarity <- as.integer(diamond$clarity)
head(diamond)

#Selecting the attributes
X <- diamond %>% select(carat, depth, table, x, y, z, clarity, cut, color)
y <- diamond$price


#For Training & Testing the datasets, split the datasets into 2
index <- createDataPartition(y, p=0.75, list=FALSE)
X_train <- X[ index, ]
X_test <- X[-index, ]
y_train <- y[index]
 y_test<-y[-index]
```



```
#Train the model
regr <- randomForest(x = X_train, y = y_train , maxnodes = 10, ntree = 10)

# Make prediction
predictions <- predict(regr, X_test)
result <- X_test
result['price'] <- y_test
result['prediction']<- predictions
head(result)

# Import library for visualization library(ggplot2) # Build scatterplot
ggplot( ) +
geom_point( aes(x = X_test$carat, y = y_test, color = 'red', alpha = 0.5) ) +
geom_point( aes(x = X_test$carat , y = predictions, color = 'blue', alpha = 0.5)) +
labs(x = "Carat", y = "Price", color = "", alpha = 'Transperency') +
scale_color_manual(labels = c( "Predicted", "Real"), values = c("blue", "red"))
```

83



Random Forest – Example 3

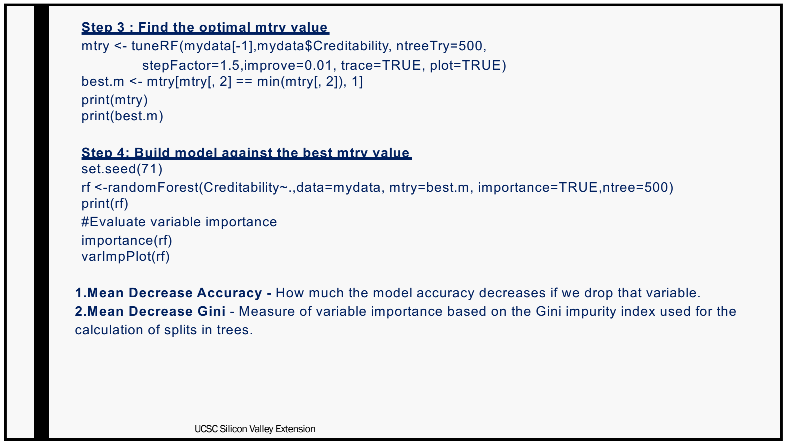
Step 1 : Data Preparation

```
mydata= read.csv("https://sites.google.com/site/pocketecoworld/german_credit.csv")
# Check types of variables
str(mydata)
# Check number of rows and columns
dim(mydata)
# Make dependent variable as a factor (categorical)
mydata$Creditability = as.factor(mydata$Creditability)
```

Step 2 : Run the Random Forest Model

```
library(randomForest)
set.seed(71)
rf <-randomForest(Creditability~.,data=mydata, ntree=500)
 print(rf)
floor(sqrt(ncol(mydata) - 1))
```

84



Step 3 : Find the optimal mtry value

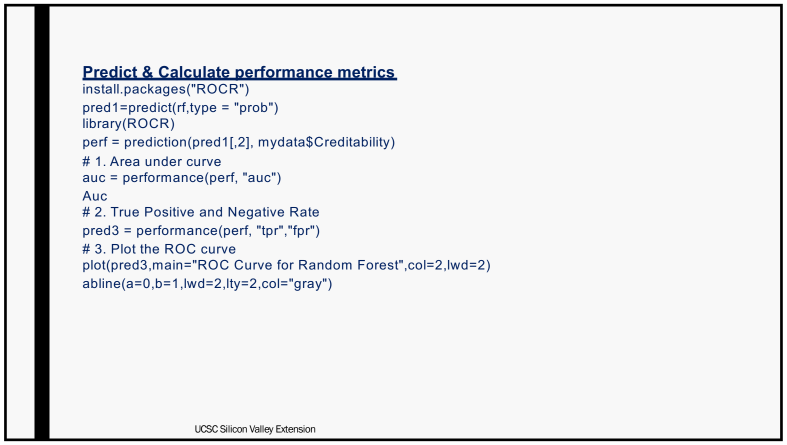
```
mtry <- tuneRF(mydata[-1],mydata$Creditability, ntreeTry=500,
          stepFactor=1.5,improve=0.01, trace=TRUE, plot=TRUE)
best.m <- mtry[mtry[, 2] == min(mtry[, 2]), 1]
print(mtry)
print(best.m)
```

Step 4: Build model against the best mtry value

```
set.seed(71)
rf <-randomForest(Creditability~.,data=mydata, mtry=best.m, importance=TRUE,ntree=500)
print(rf)
#Evaluate variable importance
importance(rf)
varImpPlot(rf)
```

1. **Mean Decrease Accuracy -** How much the model accuracy decreases if we drop that variable.
2. **Mean Decrease Gini** - Measure of variable importance based on the Gini impurity index used for the calculation of splits in trees.

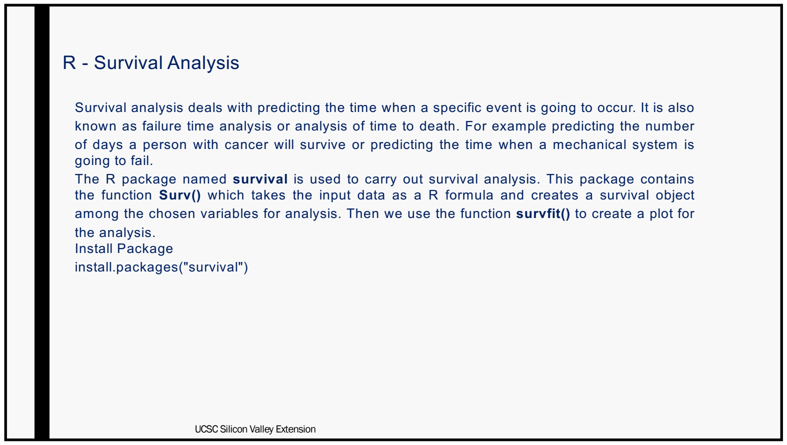
85



Predict & Calculate performance metrics

```
install.packages("ROCR")
pred1=predict(rf,type = "prob")
library(ROCR)
perf = prediction(pred1[,2], mydata$Creditability)
# 1. Area under curve
auc = performance(perf, "auc")
Auc
# 2. True Positive and Negative Rate
pred3 = performance(perf, "tpr","fpr")
# 3. Plot the ROC curve
plot(pred3,main="ROC Curve for Random Forest",col=2,lwd=2)
abline(a=0,b=1,lwd=2,lty=2,col="gray")
```

86



R - Survival Analysis

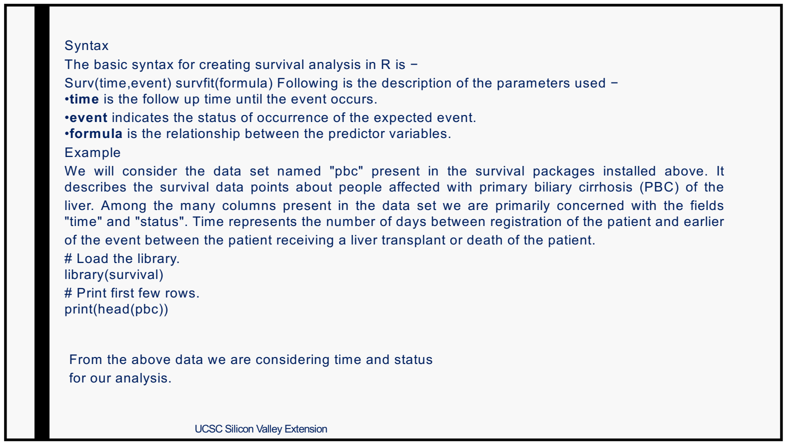
Survival analysis deals with predicting the time when a specific event is going to occur. It is also known as failure time analysis or analysis of time to death. For example predicting the number of days a person with cancer will survive or predicting the time when a mechanical system is going to fail.

The R package named **survival** is used to carry out survival analysis. This package contains the function **Surv()** which takes the input data as a R formula and creates a survival object among the chosen variables for analysis. Then we use the function **survfit()** to create a plot for the analysis.

Install Package

```
install.packages("survival")
```

87



Syntax

The basic syntax for creating survival analysis in R is −

Surv(time,event) survfit(formula) Following is the description of the parameters used −

- **time** is the follow up time until the event occurs.
- **event** indicates the status of occurrence of the expected event.
- **formula** is the relationship between the predictor variables.

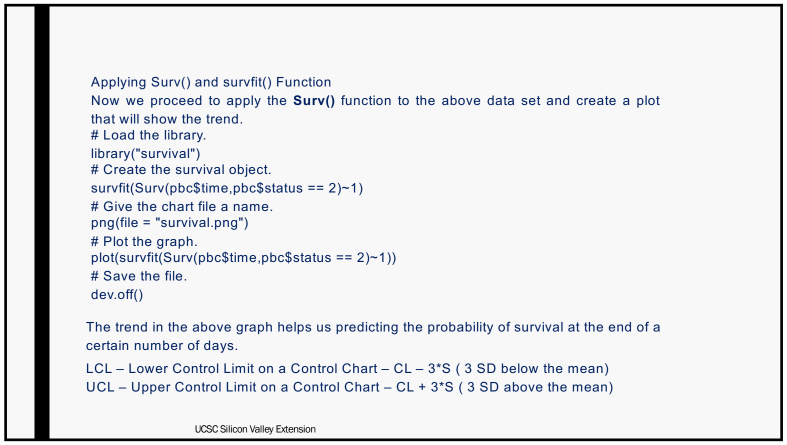
Example

We will consider the data set named "pbc" present in the survival packages installed above. It describes the survival data points about people affected with primary biliary cirrhosis (PBC) of the liver. Among the many columns present in the data set we are primarily concerned with the fields "time" and "status". Time represents the number of days between registration of the patient and earlier of the event between the patient receiving a liver transplant or death of the patient.

```
# Load the library.
library(survival)
# Print first few rows.
print(head(pbc))
```

From the above data we are considering time and status for our analysis.

88



Applying Surv() and survfit() Function

Now we proceed to apply the **Surv()** function to the above data set and create a plot that will show the trend.

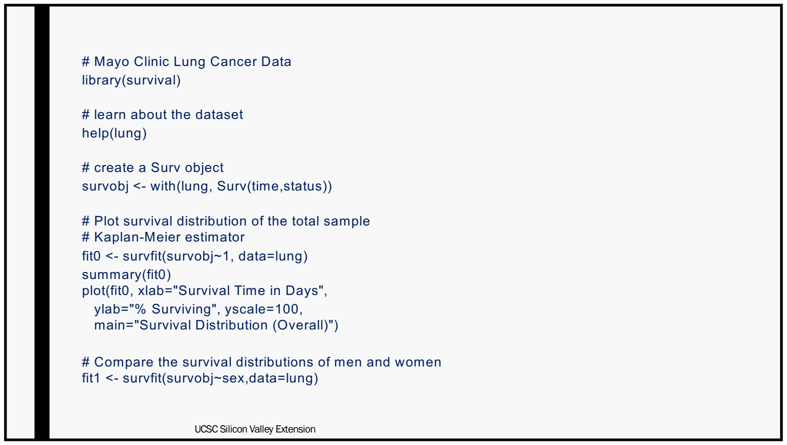
```
# Load the library.
library("survival")
# Create the survival object.
survfit(Surv(pbc$time,pbc$status == 2)~1)
# Give the chart file a name.
png(file = "survival.png")
# Plot the graph.
plot(survfit(Surv(pbc$time,pbc$status == 2)~1))
# Save the file.
dev.off()
```

The trend in the above graph helps us predicting the probability of survival at the end of a certain number of days.

LCL – Lower Control Limit on a Control Chart – CL – 3*S (3 SD below the mean)

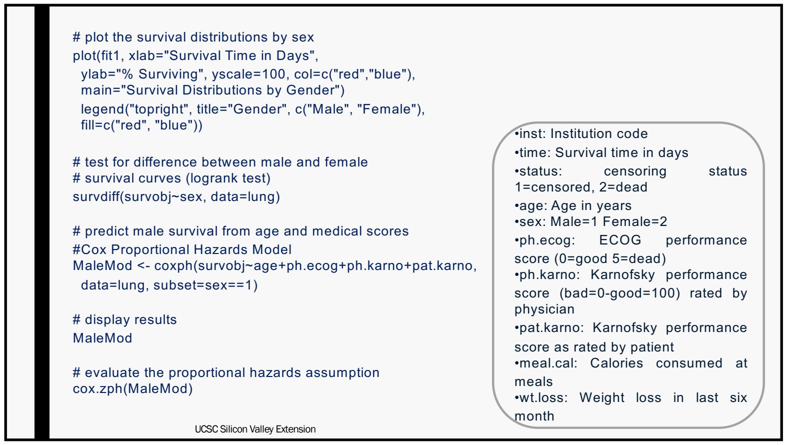
UCL – Upper Control Limit on a Control Chart – CL + 3*S (3 SD above the mean)

89



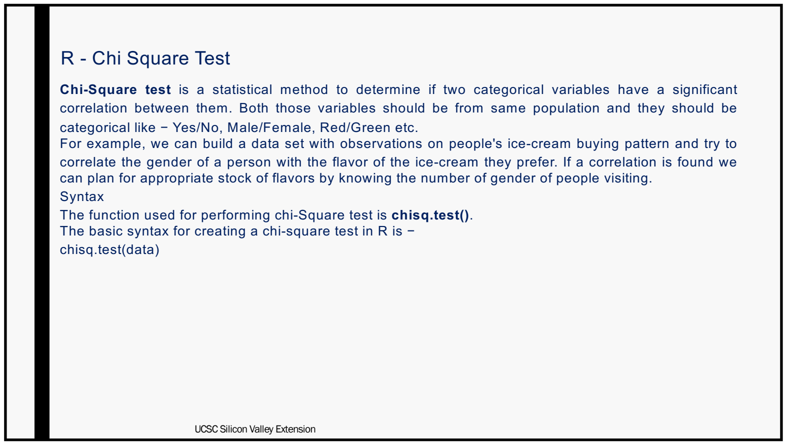
```
# Mayo Clinic Lung Cancer Data
library(survival)

# learn about the dataset
help(lung)

# create a Surv object
survobj <- with(lung, Surv(time,status))

# Plot survival distribution of the total sample
# Kaplan-Meier estimator
fit0 <- survfit(survobj~1, data=lung)
summary(fit0)
plot(fit0, xlab="Survival Time in Days",
   ylab="% Surviving", yscale=100,
   main="Survival Distribution (Overall)")

# Compare the survival distributions of men and women
fit1 <- survfit(survobj~sex,data=lung)
```

90



```
# plot the survival distributions by sex
plot(fit1, xlab="Survival Time in Days",
  ylab="% Surviving", yscale=100, col=c("red","blue"),
  main="Survival Distributions by Gender")
  legend("topright", title="Gender", c("Male", "Female"),
  fill=c("red", "blue"))

# test for difference between male and female
# survival curves (logrank test)
survdiff(survobj~sex, data=lung)

# predict male survival from age and medical scores
#Cox Proportional Hazards Model
MaleMod <- coxph(survobj~age+ph.ecog+ph.karno+pat.karno,
  data=lung, subset=sex==1)

# display results
MaleMod

# evaluate the proportional hazards assumption
cox.zph(MaleMod)
```

- inst: Institution code
- time: Survival time in days
- status: censoring status 1=censored, 2=dead
- age: Age in years
- sex: Male=1 Female=2
- ph.ecog: ECOG performance score (0=good 5=dead)
- ph.karno: Karnofsky performance score (bad=0-good=100) rated by physician
- pat.karno: Karnofsky performance score as rated by patient
- meal.cal: Calories consumed at meals
- wt.loss: Weight loss in last six month



R - Chi Square Test

Chi-Square test is a statistical method to determine if two categorical variables have a significant correlation between them. Both those variables should be from same population and they should be categorical like − Yes/No, Male/Female, Red/Green etc.

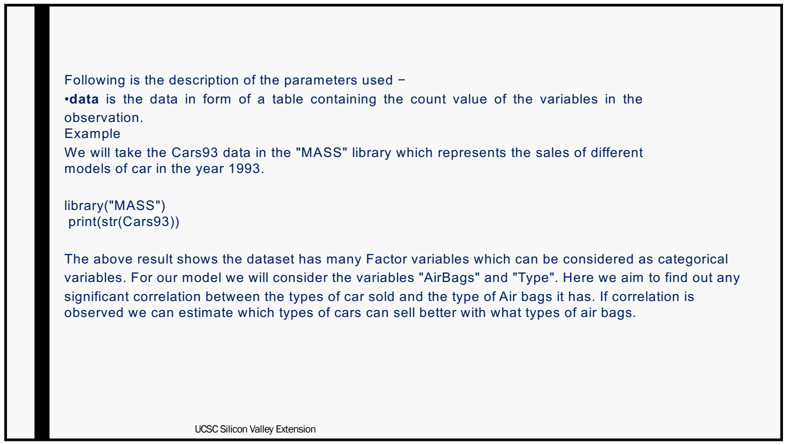
For example, we can build a data set with observations on people's ice-cream buying pattern and try to correlate the gender of a person with the flavor of the ice-cream they prefer. If a correlation is found we can plan for appropriate stock of flavors by knowing the number of gender of people visiting.

Syntax

The function used for performing chi-Square test is **chisq.test()**.

The basic syntax for creating a chi-square test in R is −

```
chisq.test(data)
```



Following is the description of the parameters used −

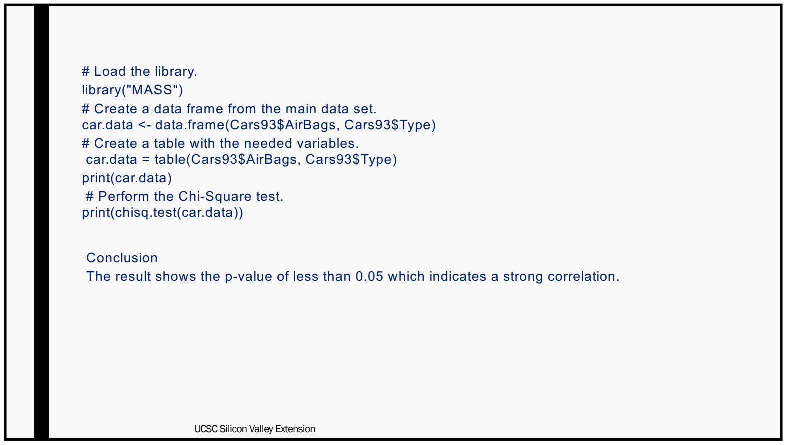
•**data** is the data in form of a table containing the count value of the variables in the observation.

Example

We will take the Cars93 data in the "MASS" library which represents the sales of different models of car in the year 1993.

```
library("MASS")
 print(str(Cars93))
```

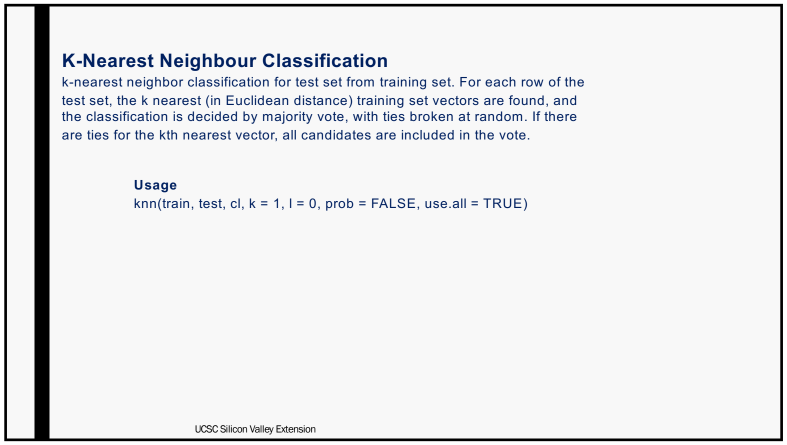
The above result shows the dataset has many Factor variables which can be considered as categorical variables. For our model we will consider the variables "AirBags" and "Type". Here we aim to find out any significant correlation between the types of car sold and the type of Air bags it has. If correlation is observed we can estimate which types of cars can sell better with what types of air bags.



```
# Load the library.
library("MASS")
# Create a data frame from the main data set.
car.data <- data.frame(Cars93$AirBags, Cars93$Type)
# Create a table with the needed variables.
 car.data = table(Cars93$AirBags, Cars93$Type)
print(car.data)
 # Perform the Chi-Square test.
print(chisq.test(car.data))
```

Conclusion

The result shows the p-value of less than 0.05 which indicates a strong correlation.

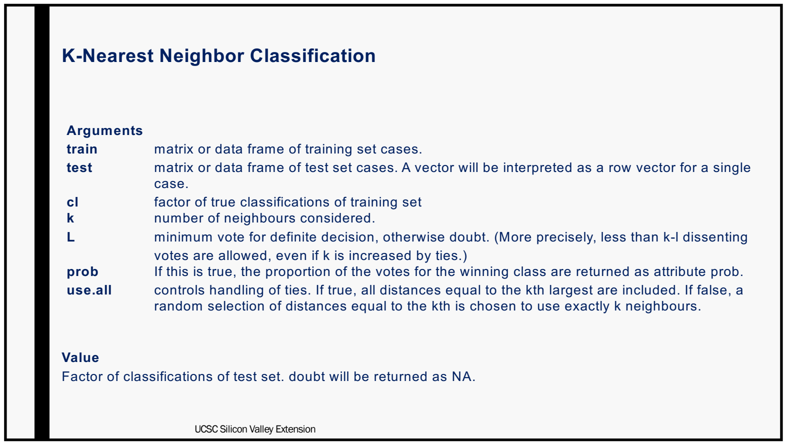


K-Nearest Neighbour Classification

k-nearest neighbor classification for test set from 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.

Usage

```
knn(train, test, cl, k = 1, l = 0, prob = FALSE, use.all = TRUE)
```



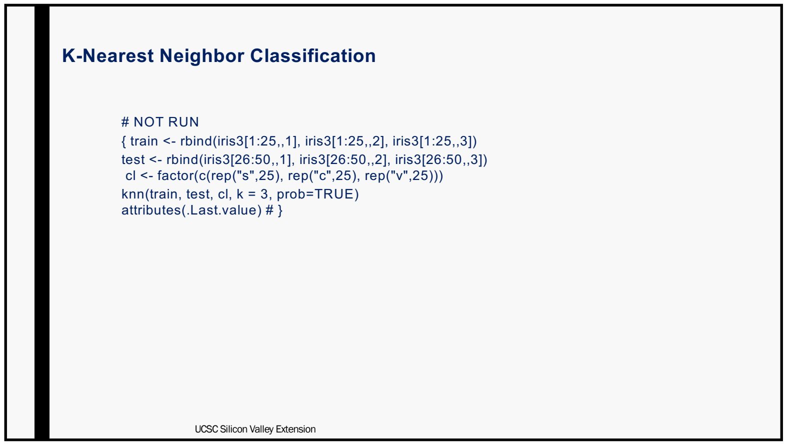
K-Nearest Neighbor Classification

Arguments

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.
cl	factor of true classifications of training set
k	number of neighbours considered.
L	minimum vote for definite decision, otherwise doubt. (More precisely, less than k-l dissenting votes are allowed, even if k is increased by ties.)
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.

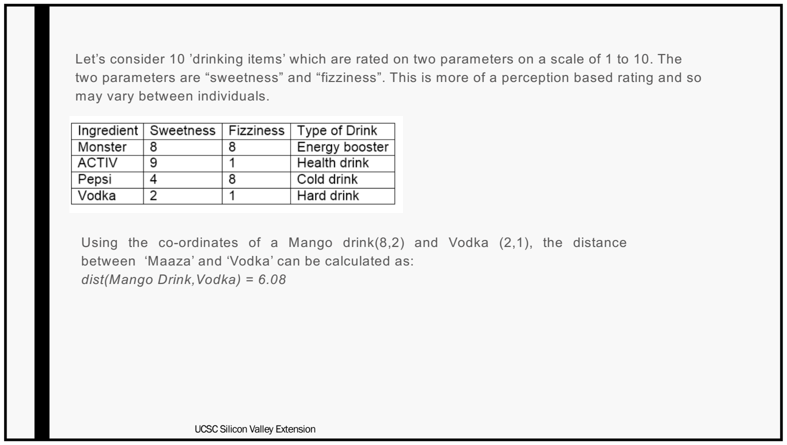
Value

Factor of classifications of test set. doubt will be returned as NA.



K-Nearest Neighbor Classification

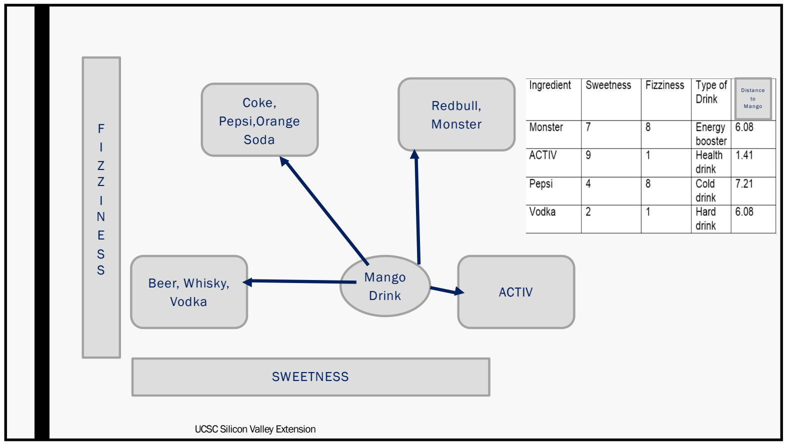
```
# NOT RUN
{ train <- rbind(iris3[1:25,,1], iris3[1:25,,2], iris3[1:25,,3])
test <- rbind(iris3[26:50,,1], iris3[26:50,,2], iris3[26:50,,3])
 cl <- factor(c(rep("s",25), rep("c",25), rep("v",25)))
knn(train, test, cl, k = 3, prob=TRUE)
attributes(.Last.value) # }
```



Let's consider 10 'drinking items' which are rated on two parameters on a scale of 1 to 10. The two parameters are "sweetness" and "fizziness". This is more of a perception based rating and so may vary between individuals.

Ingredient	Sweetness	Fizziness	Type of Drink
Monster	8	8	Energy booster
ACTIV	9	1	Health drink
Pepsi	4	8	Cold drink
Vodka	2	1	Hard drink

Using the co-ordinates of a Mango drink(8,2) and Vodka (2,1), the distance between 'Maaza' and 'Vodka' can be calculated as:
dist(Mango Drink,Vodka) = 6.08



FIZZINESS

Coke, Pepsi,Orange Soda

Redbull, Monster

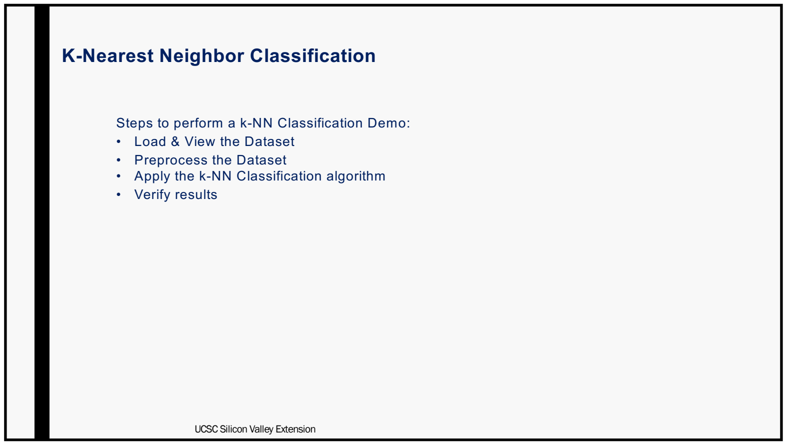
Beer, Whisky, Vodka

Mango Drink

ACTIV

SWEETNESS

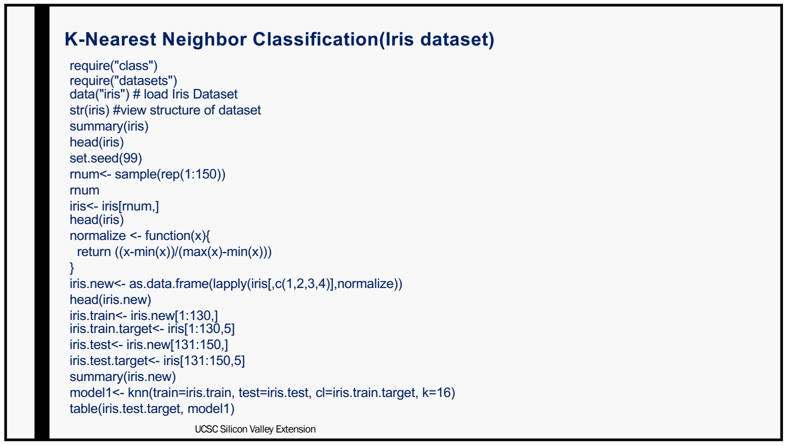
Ingredient	Sweetness	Fizziness	Type of Drink	Distance to Mango
Monster	7	8	Energy booster	6.08
ACTIV	9	1	Health drink	1.41
Pepsi	4	8	Cold drink	7.21
Vodka	2	1	Hard drink	6.08



K-Nearest Neighbor Classification

Steps to perform a k-NN Classification Demo:

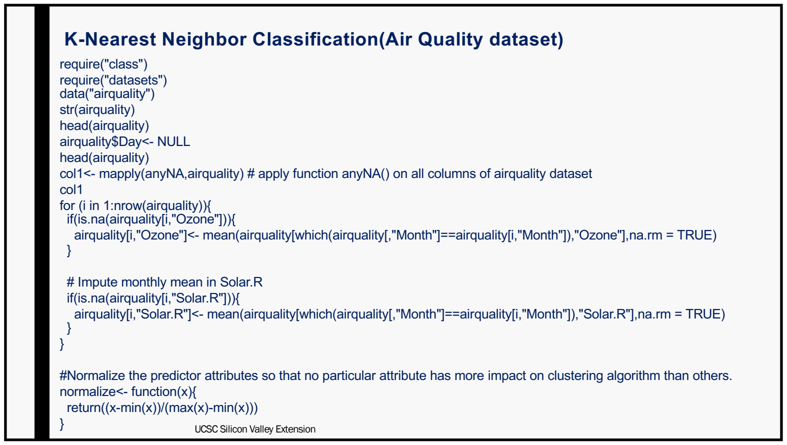
- Load & View the Dataset
- Preprocess the Dataset
- Apply the k-NN Classification algorithm
- Verify results



K-Nearest Neighbor Classification(Iris dataset)

```
require("class")
require("datasets")
data("iris") # load Iris Dataset
str(iris) #view structure of dataset
summary(iris)
head(iris)
set.seed(99)
rnum<- sample(rep(1:150))
rnum
iris<- iris[rnum,]
head(iris)
normalize <- function(x){
  return ((x-min(x))/(max(x)-min(x)))
}
iris.new<- as.data.frame(lapply(iris[,c(1,2,3,4)],normalize))
head(iris.new)
iris.train<- iris.new[1:130,]
iris.train.target<- iris[1:130,5]
iris.test<- iris.new[131:150,]
iris.test.target<- iris[131:150,5]
summary(iris.new)
model1<- knn(train=iris.train, test=iris.test, cl=iris.train.target, k=16)
table(iris.test.target, model1)
```

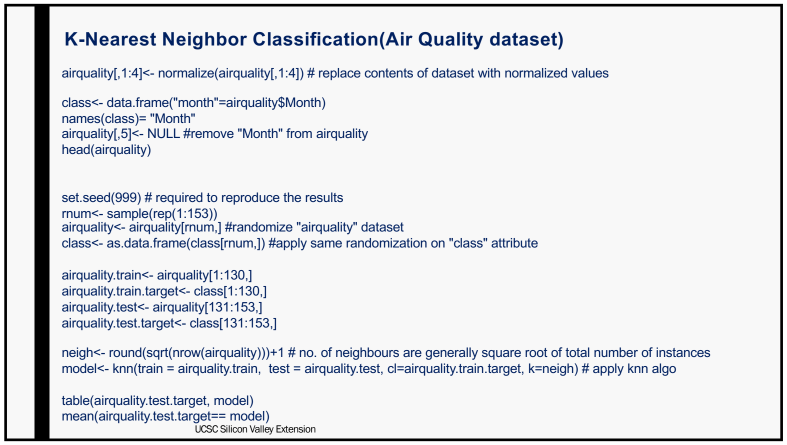
101



K-Nearest Neighbor Classification(Air Quality dataset)

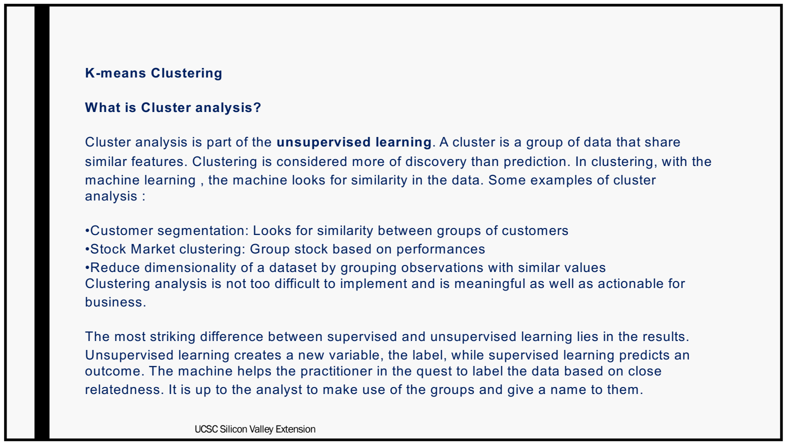
```
require("class")
require("datasets")
data("airquality")
str(airquality)
head(airquality)
airquality$Day<- NULL
head(airquality)
col1<- mapply(anyNA,airquality) # apply function anyNA() on all columns of airquality dataset
col1
for (i in 1:nrow(airquality)){
  if(is.na(airquality[i,"Ozone"])){
    airquality[i,"Ozone"]<- mean(airquality[which(airquality[,"Month"]==airquality[i,"Month"]),"Ozone"],na.rm = TRUE)
  }

  # Impute monthly mean in Solar.R
  if(is.na(airquality[i,"Solar.R"])){
    airquality[i,"Solar.R"]<- mean(airquality[which(airquality[,"Month"]==airquality[i,"Month"]),"Solar.R"],na.rm = TRUE)
  }
}

#Normalize the predictor attributes so that no particular attribute has more impact on clustering algorithm than others.
normalize<- function(x){
  return((x-min(x))/(max(x)-min(x)))
}
```

102



K-Nearest Neighbor Classification(Air Quality dataset)

```
airquality[,1:4]<- normalize(airquality[,1:4]) # replace contents of dataset with normalized values

class<- data.frame("month"=airquality$Month)
names(class)= "Month"
airquality[,5]<- NULL #remove "Month" from airquality
head(airquality)


set.seed(999) # required to reproduce the results
rnum<- sample(rep(1:153))
airquality<- airquality[rnum,] #randomize "airquality" dataset
class<- as.data.frame(class[rnum,]) #apply same randomization on "class" attribute

airquality.train<- airquality[1:130,]
airquality.train.target<- class[1:130,]
airquality.test<- airquality[131:153,]
airquality.test.target<- class[131:153,]

neigh<- round(sqrt(nrow(airquality)))+1 # no. of neighbours are generally square root of total number of instances
model<- knn(train = airquality.train,  test = airquality.test, cl=airquality.train.target, k=neigh) # apply knn algo

table(airquality.test.target, model)
mean(airquality.test.target== model)
```



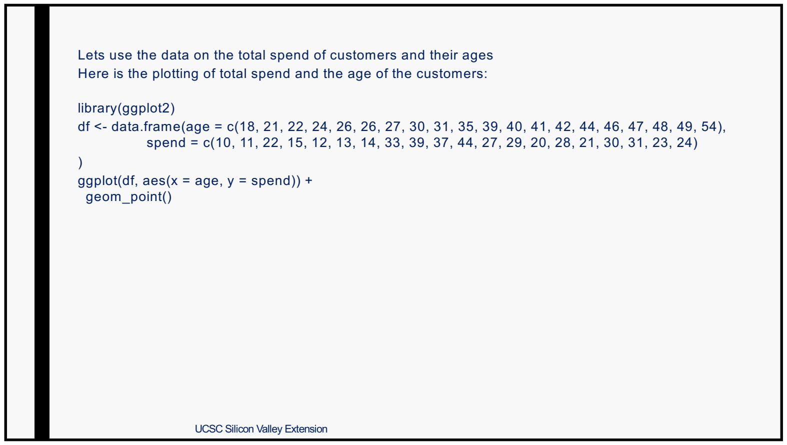
K-means Clustering

What is Cluster analysis?

Cluster analysis is part of the **unsupervised learning**. A cluster is a group of data that share similar features. Clustering is considered more of discovery than prediction. In clustering, with the machine learning , the machine looks for similarity in the data. Some examples of cluster analysis :

•Customer segmentation: Looks for similarity between groups of customers
•Stock Market clustering: Group stock based on performances
•Reduce dimensionality of a dataset by grouping observations with similar values
Clustering analysis is not too difficult to implement and is meaningful as well as actionable for business.

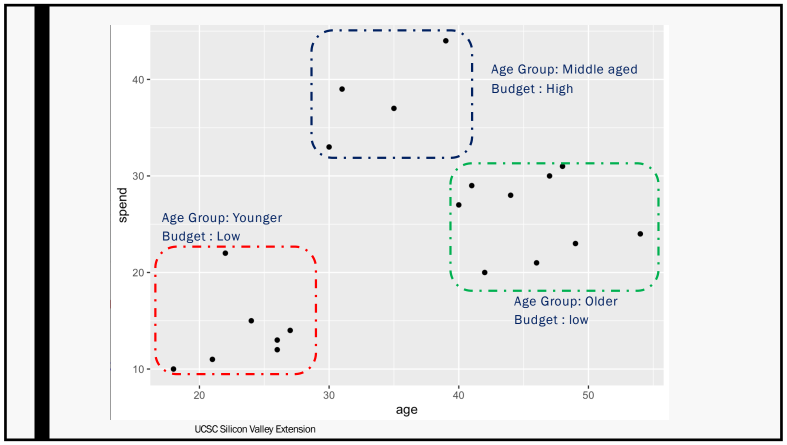
The most striking difference between supervised and unsupervised learning lies in the results. Unsupervised learning creates a new variable, the label, while supervised learning predicts an outcome. The machine helps the practitioner in the quest to label the data based on close relatedness. It is up to the analyst to make use of the groups and give a name to them.



Lets use the data on the total spend of customers and their ages
Here is the plotting of total spend and the age of the customers:

```
library(ggplot2)
df <- data.frame(age = c(18, 21, 22, 24, 26, 26, 27, 30, 31, 35, 39, 40, 41, 42, 44, 46, 47, 48, 49, 54),
                spend = c(10, 11, 22, 15, 12, 13, 14, 33, 39, 37, 44, 27, 29, 20, 28, 21, 30, 31, 23, 24)
)
ggplot(df, aes(x = age, y = spend)) +
  geom_point()
```

105



Age Group: Middle aged
Budget : High

Age Group: Younger
Budget : Low

Age Group: Older
Budget : low

106