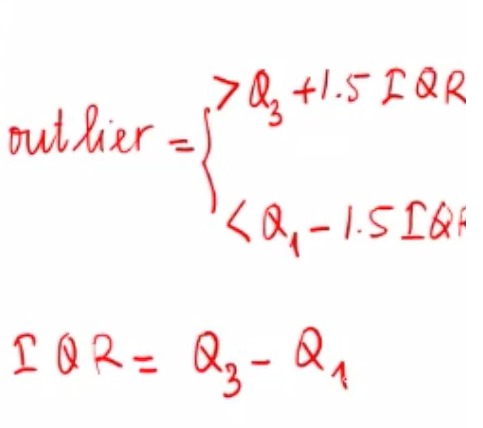


Histogram:

#for left skewed histograms, the left tail is longer than the right tail

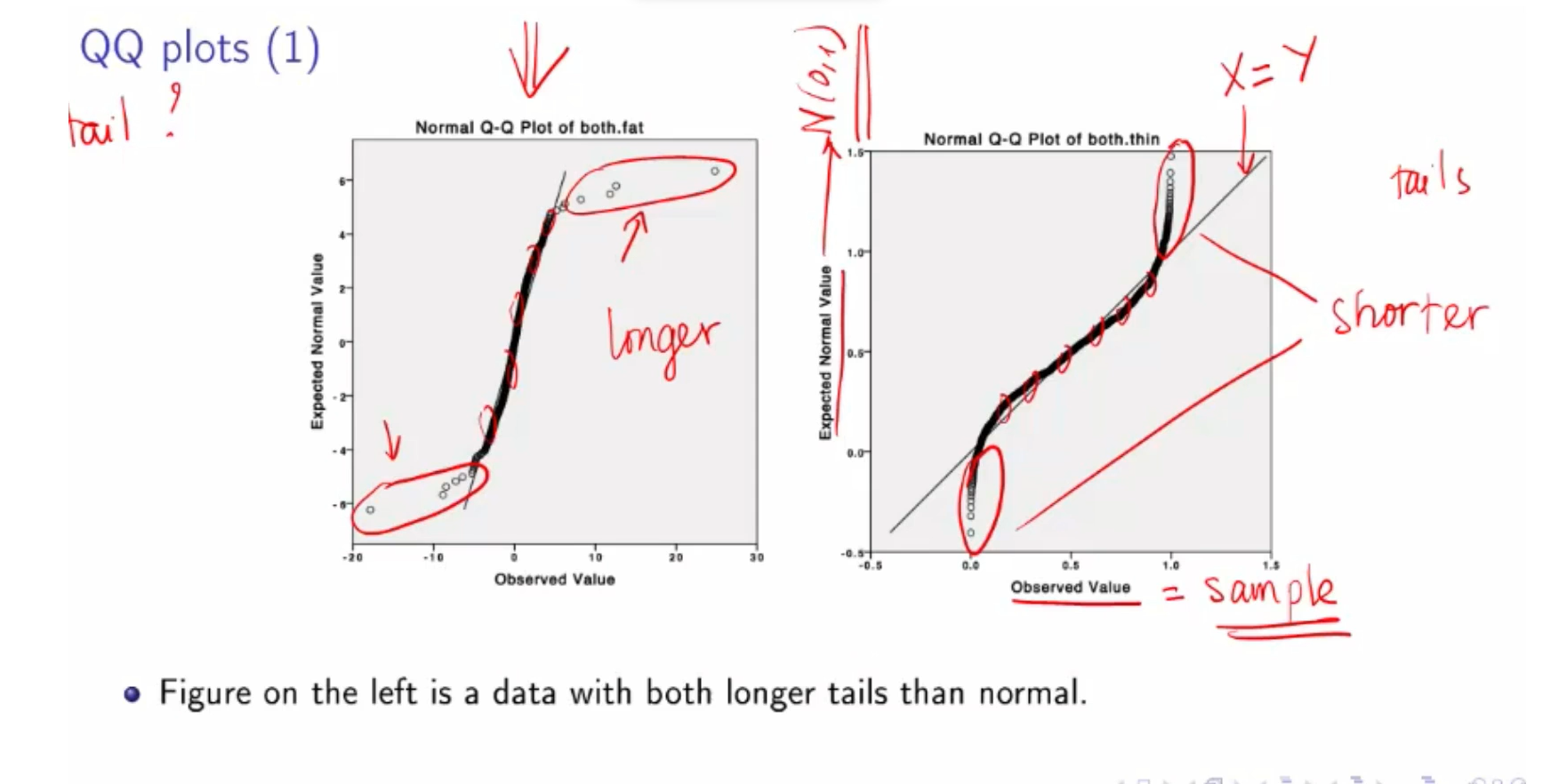
#ie theres more values on the left edge than the right edge, some vid said the left side looks more squished than the right side

#vice versa for right skew

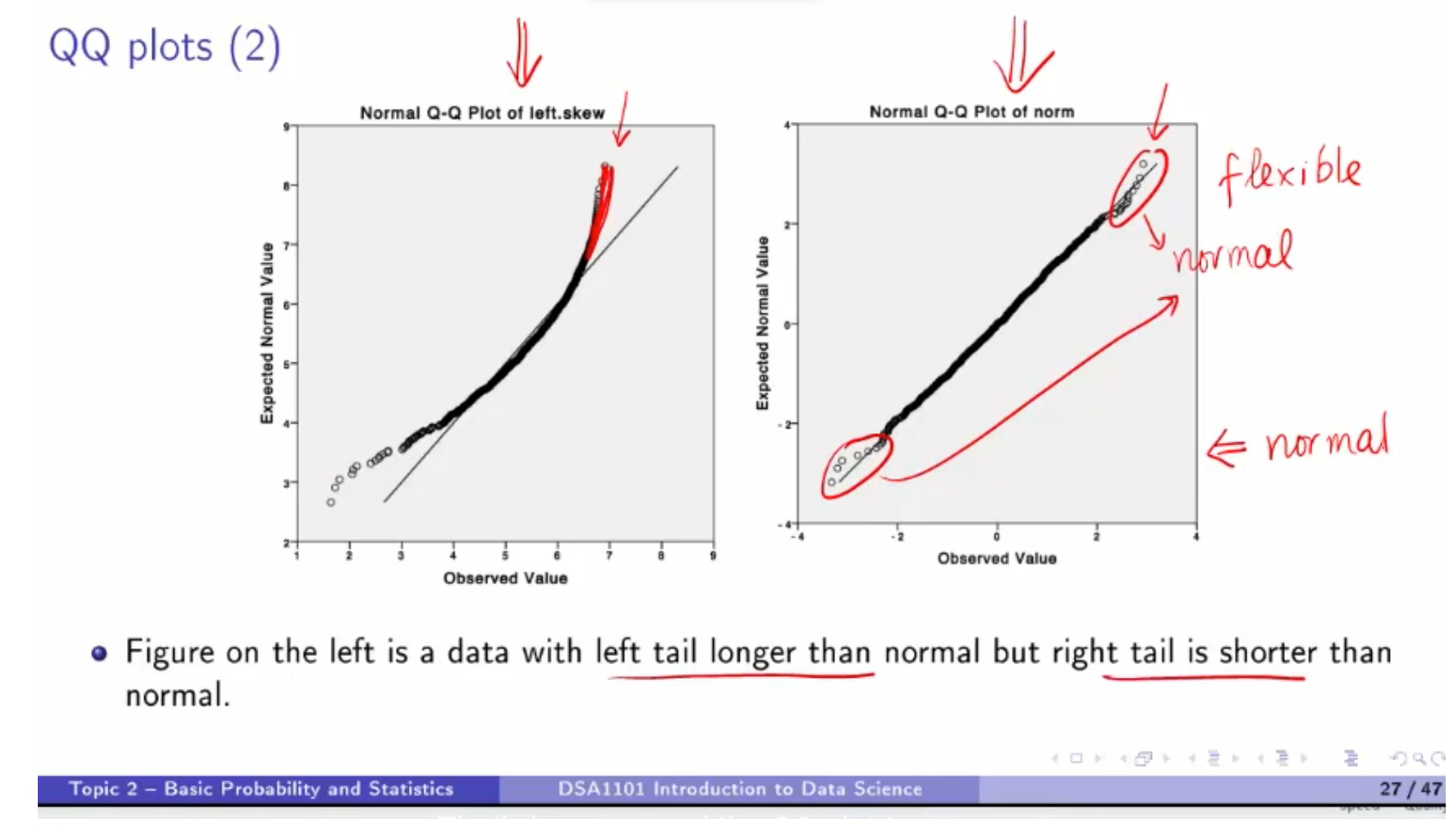


QQ Plots:

* Stands for “Quantile-Quantile” Plot
* Used to see if the sample follows (approx) a normal distribution or not
  + So the plot matches the sample sample quantiles against the theoretical quantiles of a N(0,1) distribution



* Figure on the right is for data with both tails shorter than normal



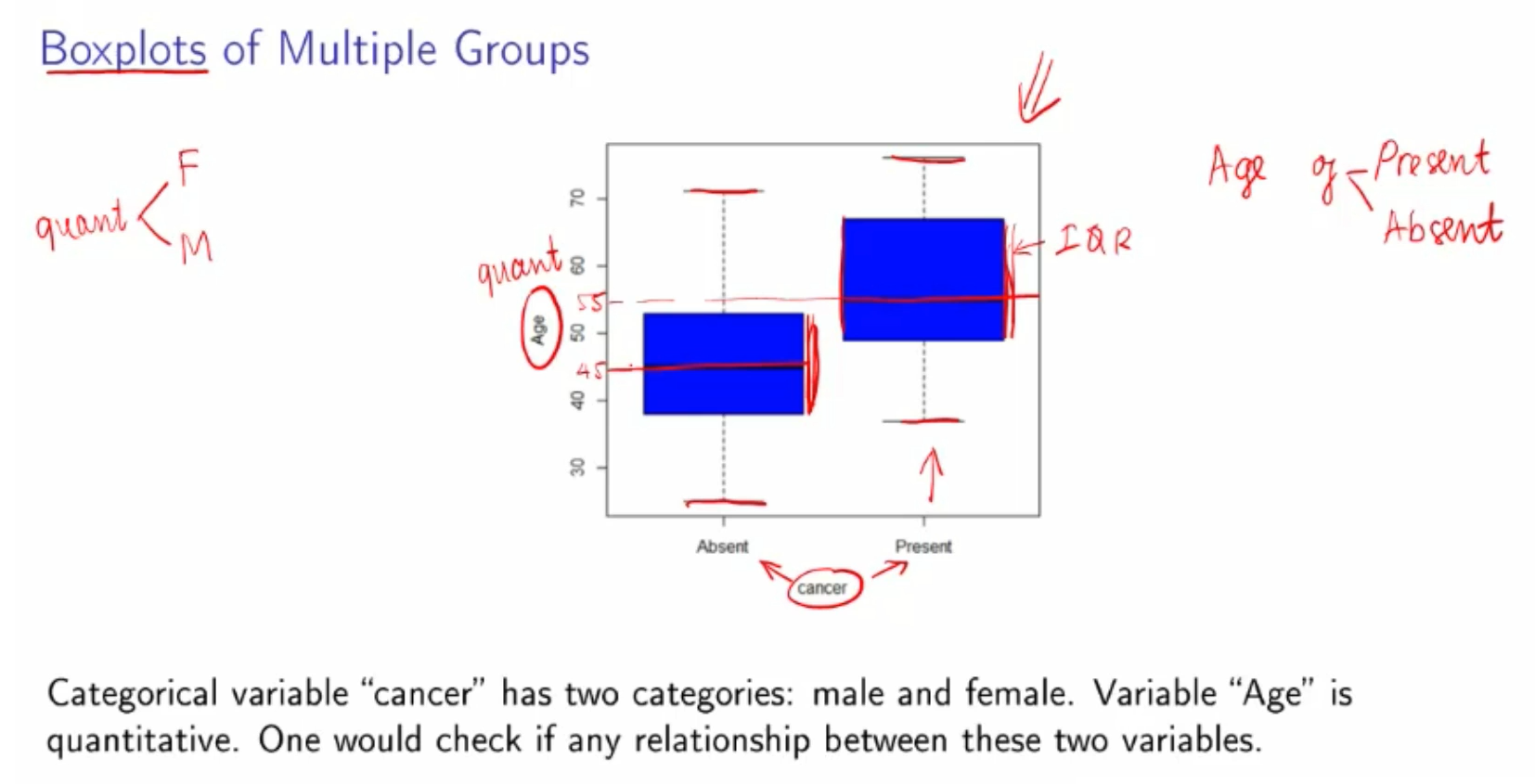
* For figure on the right, notice there are some plots not \*on\* the line, but does not show a strong trend that deviates from the straight line (ie not suddenly curving up or smth), so be flexible on it lol

SCATTERPLOT

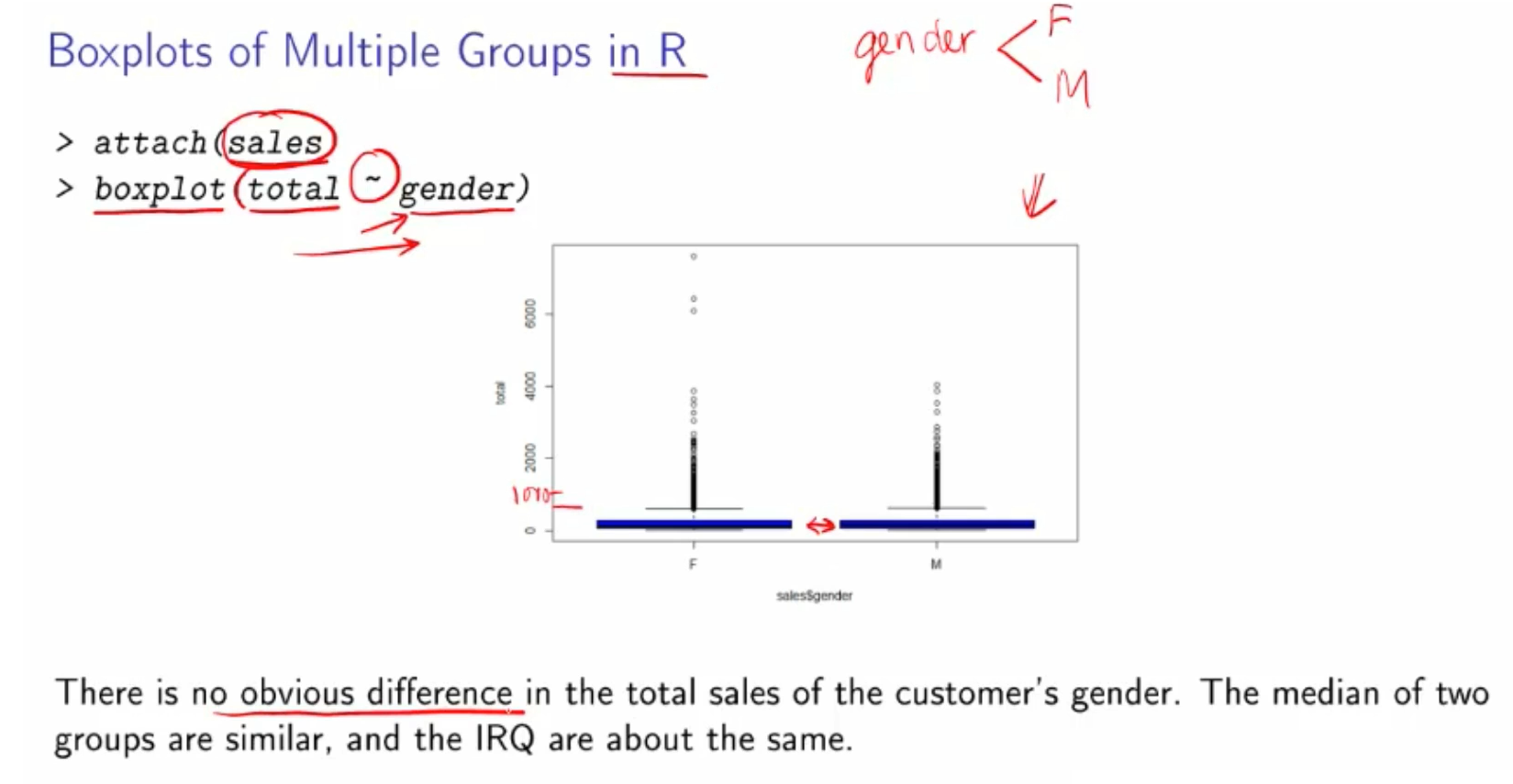


* X is the number of orders and Y is the number of sales
* Points are not random; seems like forming a trend => 2 variables are associated
  + Association is positive:  
    when order is small => total sales is small,  
    when order is large => total sales is large
  + Points do not form a straight line nicely, but approx it is quite straight => linear
  + Does not form any pattern of non-linear (eg quadratic/sin curve)
* Outliers: Some obvs that deviate from the rest ie the circled region, extremely large number of orders & large number of sales
* **TLDR the association is approximately linear, with a few extremely large values**
* **"plot of gender by align" => y by x => gender is y and align is x**

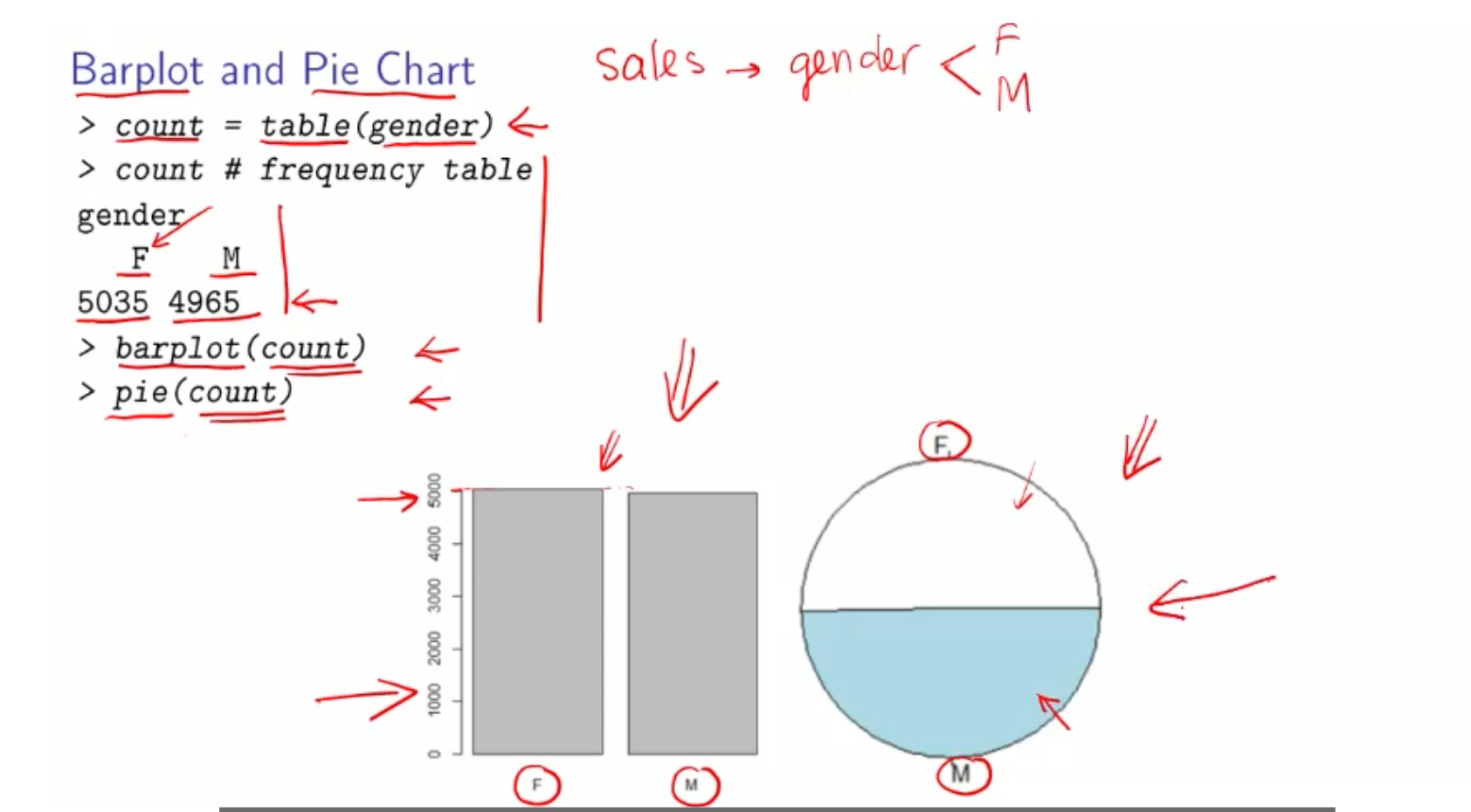
**BOXPLOT**

****

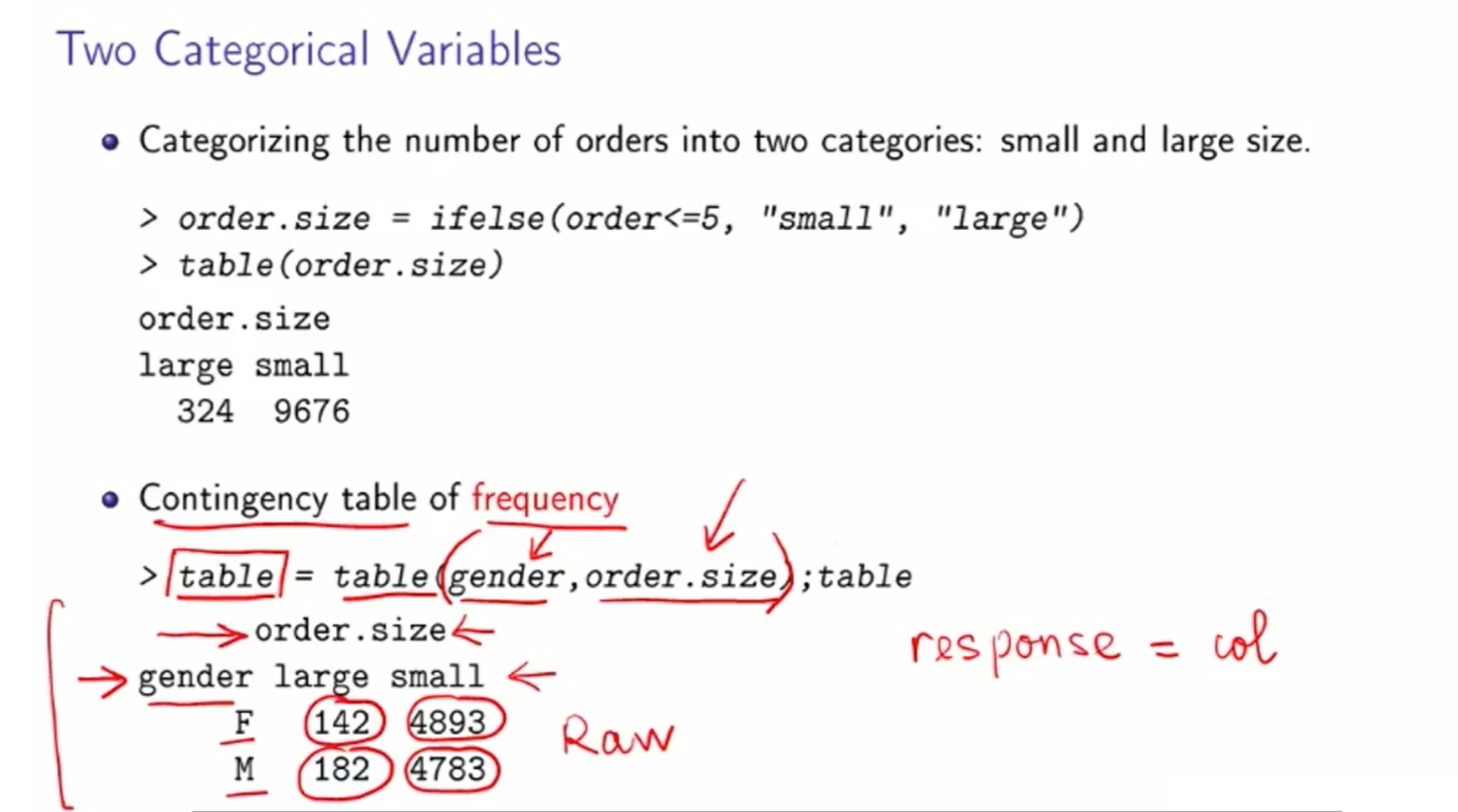
* **Compare: are the medians similar, and are the boxes overlapping or not (ie is the IQR within similar ranges)**
* **Both groups no outliers**
* **If the distribution of each group is unimodal, they both could be quite normal**
* **Range is quite similar**
* **Boxes are overlapping but not much**
  + **IQR or the group Present is larger than the group Absent  
    I.e spread in age in the center part of Present is larger**
* **Median age of Present > median age of Absent**
  + **Indicates that age could affect cancer status**

****

* **Gender is categorical, sales is quantitative**
* **Both have majority of datapoints below 1,000**
* **Both have many outliers**
* **Both boxes are overlapping and similar in size**
* **Median values are similar**
* **HENCE no obvious difference**

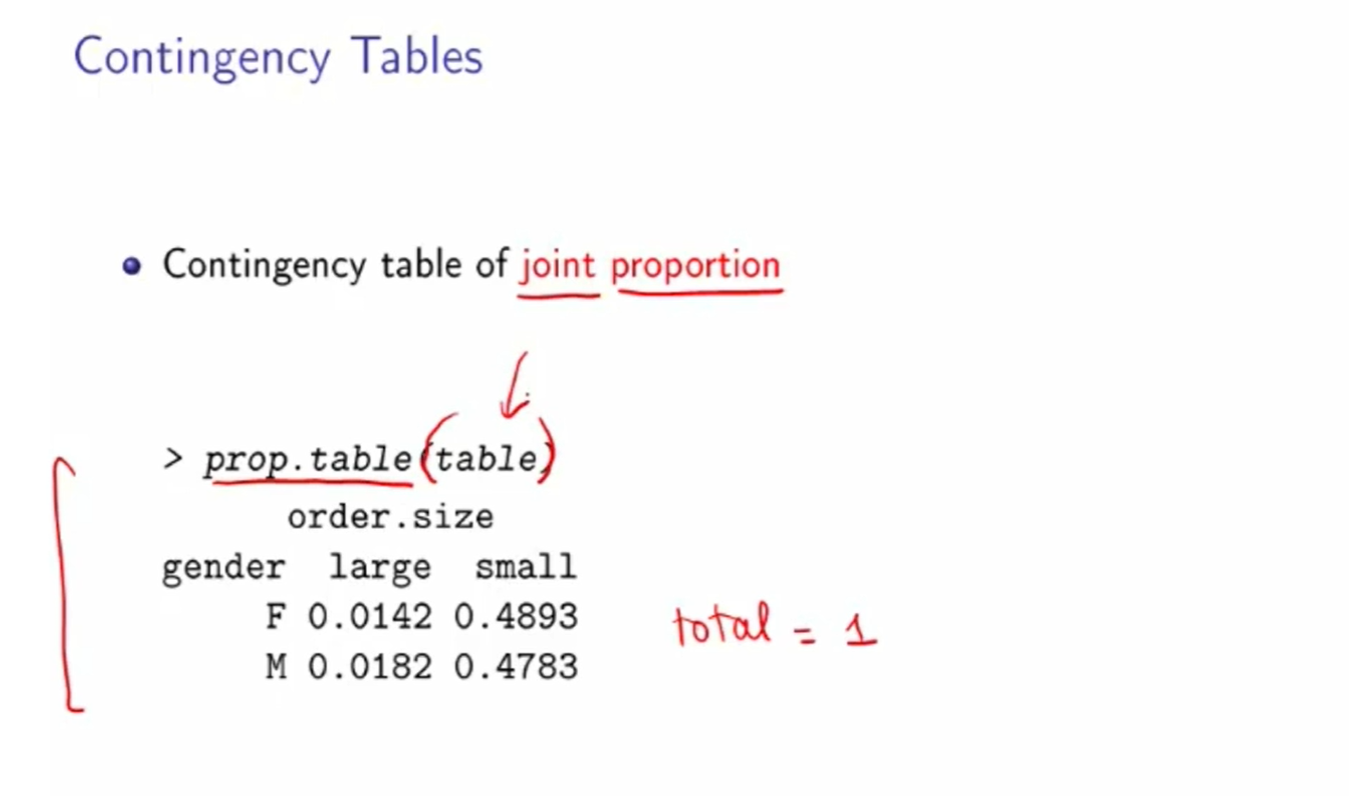
****

* **Females are the MODAL VALUE because they’re the most frequent categorical value under gender**

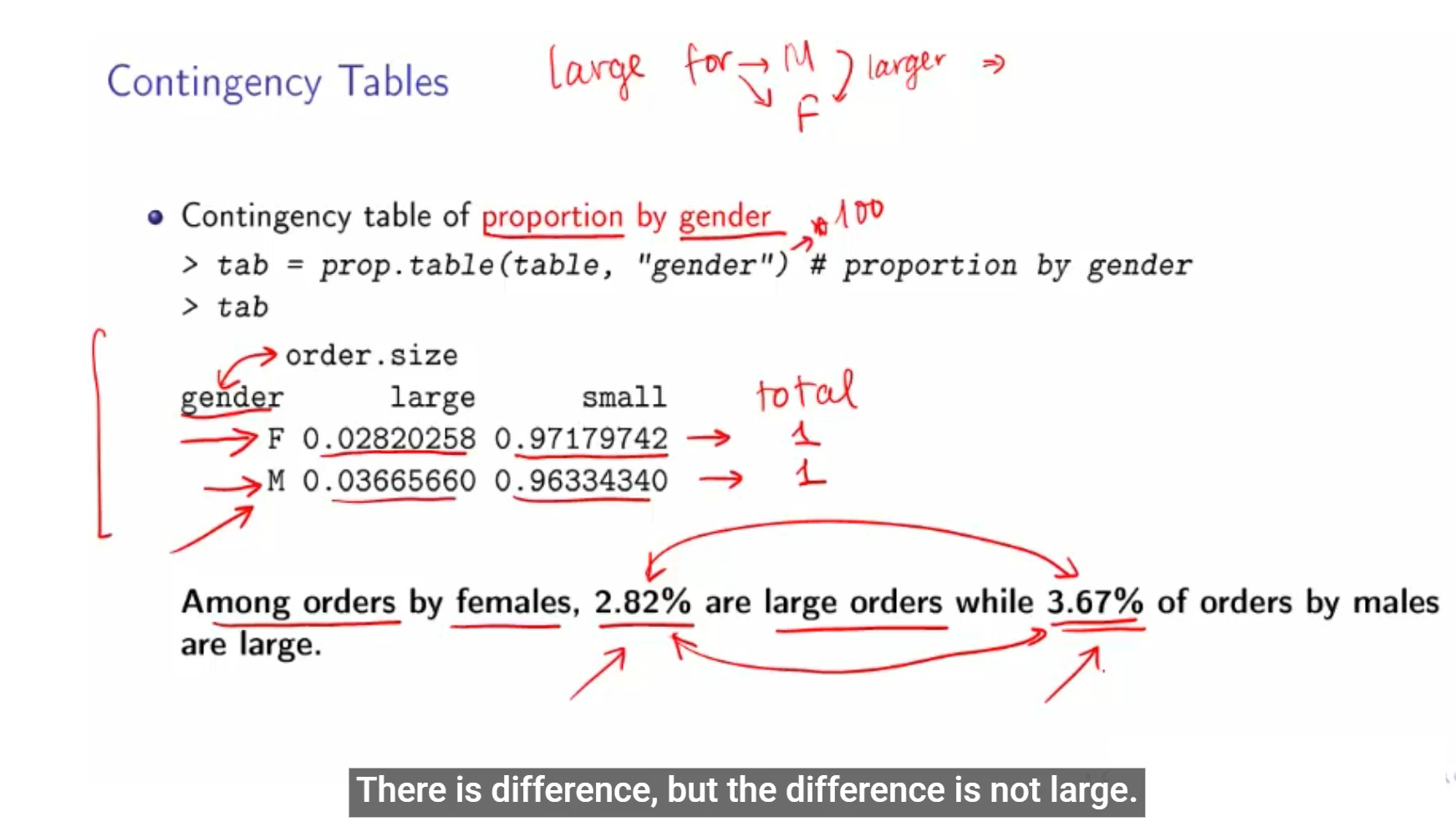
****

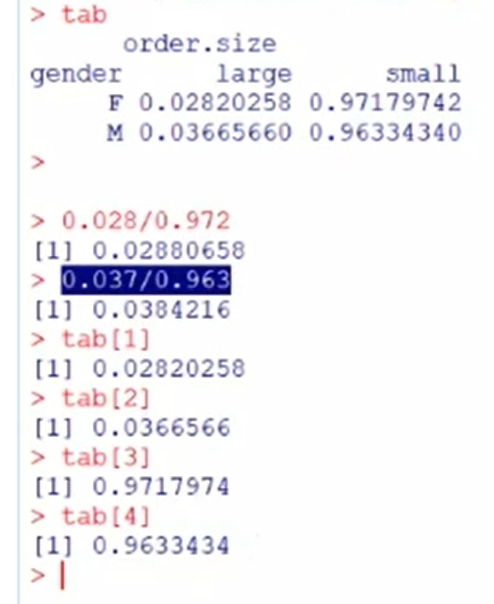
* **Response variable is the dependent variable lol, cus it changes or ‘responds’ to changes in the independent variable**
* **Response variable is usually put as the 2nd parameter in table() so it becomes the column. Its like table(row, column)**

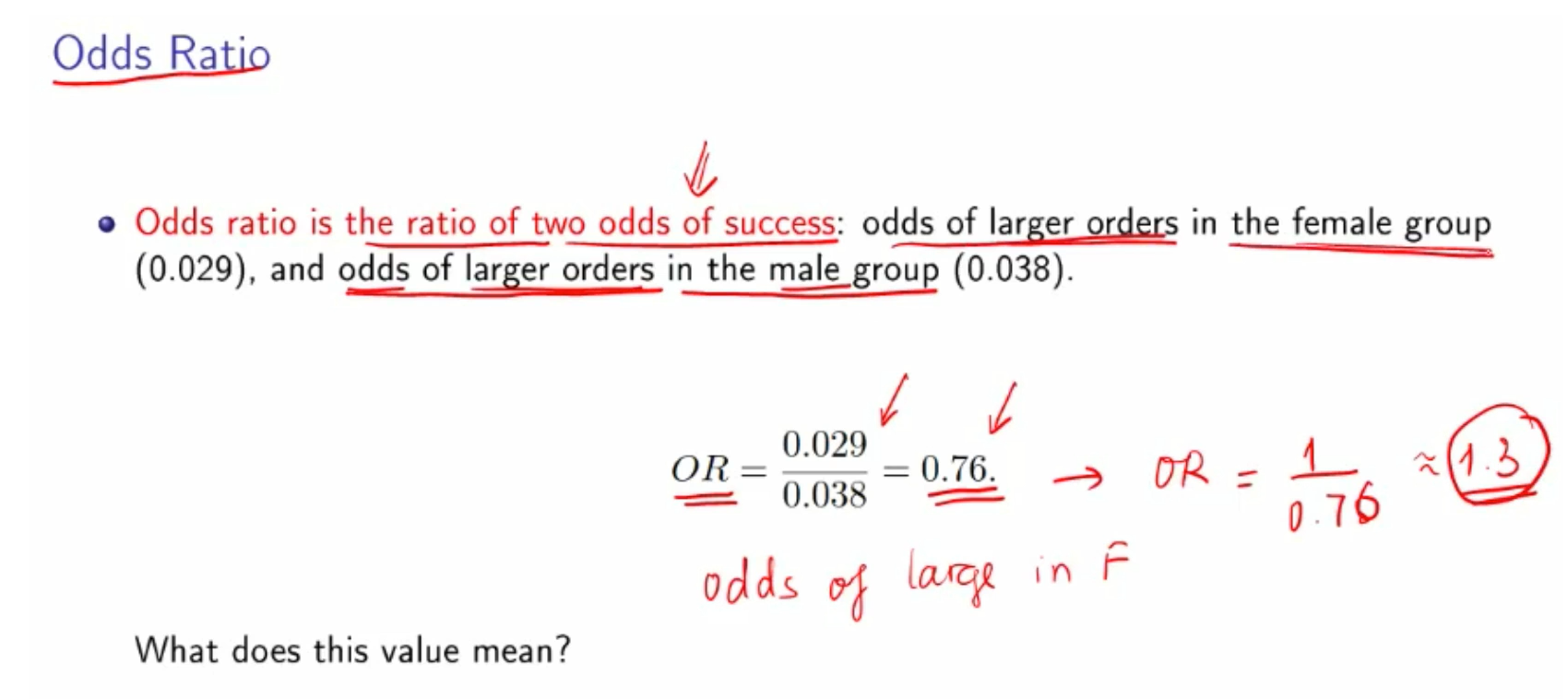
**CONTINGENCY TABLE**

****

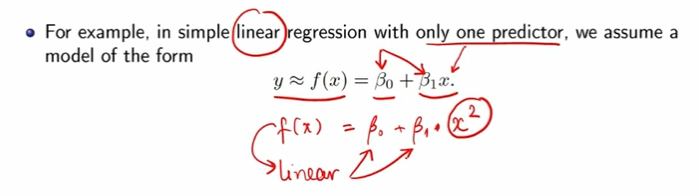
* **Total of all the rows and columns is 1**
* **prop.table( frequency table created using table() ) so 1 parameter only**

****

****

****

* **0.76: the odds of having large order amongst female is 0.76 times the odds of having large orders among males**
* **If you flip the numerator and denominator, the odds of having large order amongst males is 1.3 times the odds of having large orders amongst females (reciprocal of 0.76) ⇒ meaning is still the same**
* **Note that odds of success = p / (1-p)**

****

******

This is considered unimodal symmetric

***Add on pls. OG is from NM2204***

| THING U WANT TO DO | COMMAND(S) | REMARKS |
| --- | --- | --- |
| ***Import*** the ***data-set***, `playlist\_data.csv`  Import .txt file: use read.csv() but include parameters sep = "", header = TRUE | read\_csv("playlist\_data.csv")  read.csv("~/Github/DSA1101 Slayers/datasets/Colleges.txt", sep = "", header = TRUE)  Put header = TRUE when the first row is the names of the variables | REQUIRES TIDYVERSE LIBRARY |
| ***Assign*** the ***data-set*** to a variable, `playlist\_data` | playlist\_data <- read\_csv("playlist\_data.csv") | REQUIRES TIDYVERSE LIBRARY |
| ***Install packages*** | install.packages()  PACKAGESSSSSS with an s | No need re-install if used library before. |
| Get more information about `read\_csv()` command | ?read\_csv() |  |
| Display all the columns of the variable stacked one below another | glimpse(playlist\_data) |  |
| Any type of filtering  library(dplyr) | dplyr SPELLING PLEASE |  |
| distinct(): find the unique senders |  |  |
| Print all lines/specified number of lines | %>% print(***n = length(***playlist\_data$DJ\_Name)) | Filter (%>%) is required |
| Conjugate strings within a print statement  paste0() | paste0(“hello”, 4)  Prints hello4  Note that usage of “,”does not automatically give a space  paste0("the equation is: price(hat) = ", M2$coeff[1], " + ", M2$coeff[2], "price + ",M2$coeff[3], "I(NW = 0)") |  |
| Append vector | col = c(og\_vector, new\_values)  Eg col = c(col, "light")  OR  append(og\_vector, new\_values)  OR for an INDEXED LIST (typically with for loops)  skibbidi = c()  for (i in 1:15) {  kout = kmeans(standardised.X, centers = i)  **skibbidi[i] = kout$withinss**  } |  |
| ggplot garbage | ggplot(data = playlist\_data,  mapping = aes(x=Age,y=Rating)) +  geom\_point(shape=23, fill = "purple", size=3) +  labs(x = "Age", y = "Rating", title = "Scatter graph of Age against Ratings", caption = "My insight is that since there are both high and low ratings for both younger and older DJs, ratings are not dependent on age") | Refer to documentation: <https://ggplot2.tidyverse.org/reference/#plot-basics>  Refer to Challenge 2 |
| Initialize list | student\_info <- ***list***(student\_names=c("Alice","Bob","Catherine"), scores=c(85L, 92L, 78L), passed=c(TRUE, TRUE, FALSE)) | Refer to Challenge 3, Q10 |
| Explicit coercion  (conversion) | prices <- ***as.numeric***(prices) | Refer to Challenge 3, Q12 |
| Implicit coercion  (conversion) | nums <- c(5, 10, 15)  noms <- c("apple", "banana", "cherry")  ***numnoms <- c(nums, noms)***  Combining the numeric and character vectors will implicitly coerce the numeric vector to character. The resulting vector will be character type. R coerces data to a common type when vectors are combined. ***R coerces data to a common type when vectors are combined*** | Refer to Challenge 3, Q13 |
| What does unlist() do? | It converts a list to a vector. |  |
| Filter: tl;dr | comm\_data %>% ***filter(date == "2/8/2023" & channel == "Twitter")*** %>% select(date, channel, message)  common\_cyl <- cars %>%  filter(ncyl %in% c(4, 6, 8)) | The AND(&), OR(|) all apply here  <https://www.tutorialspoint.com/r/r_operators.htm>  Datacamp |
| arrange() | Will arrange rows in ASCENDING ORDER. To reverse order, put desc() |  |
| create a new data frame with summarised data  summarise()  Summarising based on other variable: eg summarise ave sentiment FOR EACH sender (summarise variable A categorised by variable B) | %>% summarise(total\_char = sum(nchar(message)))  %>% summarise(avg\_sentiment=me an(sentiment))  %>% group\_by(sender) %>% summarise(avg\_score=mean(sentiment)) | <https://dplyr.tidyverse.org/reference/summarise.html>   Challenge 4, Q8 & 12  Challenge 4, Q7 |
| Create dataframe / data frame | df <- data.frame(first\_column, second\_column) |  |
| Access ONE specific column only OR Access/limit to certain columns | data\_frame$column\_name  For MULTIPLE columns: use %>% select() |  |
| Access vector within list, given vector name | **mixed\_data** <- list( **numeric\_vector = c(10, 20, 30)**, character\_vector = c("red", "green", "blue"), logical\_vector = c(TRUE, FALSE, TRUE) )  mean\_numeric <- mean(***mixed\_data$numeric\_vector***) | Challenge 3, Q16 |
| Access a vector inside of a list, given the name of the vector  Eg uk bob = “tire”, you want to find “tire” and uk bob | **score\_bob <- student\_info$scores[student\_info$names == "Bob"]** | Challenge 3, Q10 |
| Find and ***print the indices of all occurrences of the word*** “apple.” | words <- c("apple", "banana", "cherry", "apple") **indices <- *which(words == "apple")*** print(indices) | Challenge 3, Q19 |
| Create frequency table | **table(row, column)**  **For all categorical variables data** | Topic 2 Video 8 |
| Create contingency table  Create a table where all the rows OR columns add to 1 | **prop.table(name\_of\_freq\_table)**  **prop.table(name\_of\_freq\_table, “name\_of\_row/column”)**  **Eg tab = prop.table(table, “gender”)#proportion by gender**  Remember if using a single variable, don’t use the variable directly  ie must do: prop.table(table(Churned))  Instead of prop.table(Churned)  **For all categorical variables data**  **NOTE: prop.table(name\_of\_freq\_table)with NO ADDITIONAL PARAMETERS will make the WHOLE TABLE TO ADD TO 1.**  **i.e.**  **[,1] [,2] [,3]**  **[1,] 1 3 5**  **[2,] 2 4 6**  **will become**  **[,1] [,2] [,3]**  **[1,] 0.04761905 0.1428571 0.2380952**  **[2,] 0.09523810 0.1904762 0.2857143**  **Where total = 1+2+3+4+5+6 = 21**  **[1,1] = 1/21 = 0.04761905**  **[2,1] = 2/21 = 0.09523810**  **To get proportions by row, try**  q = table(Churned)  q = q/rowSums(q)  Same goes for colSums() if columns need to be added.  [OR do **q = prop.table(q, “gender”)**#proportion by gender **but i dont trust you to do this**] | Topic 2 Video 8 |
| Create bar plot  No library needed | **barplot()**  barplot(max.temp,  main = "Maximum Temperatures in a Week",  xlab = "Degree Celsius",  ylab = "Day",  names.arg = c("Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"),  col = "darkred",  horiz = TRUE) |  |
| Create a Boxplot  Find outlier values of boxplot  (without ggplot) | **boxplot(total, xlab = "variable", col = "blue")**  **Outlier = boxplot(column\_variable)$out**  **Eg outtie = boxplot(fev\_var, xlab = "FEV", col = "blue")$out ; outtie** |  |
| Comment on boxplot outliers |  |  |
| Find the position of the outliers, ie the index of outliers | **index = which(FEV %in% c(out))**  **Note FEV is the name of the variable/column, not the entire db**  **FEV %in% c(out) gives you a logical list of TRUEs and FALSEs ⇒ which() will isolate the TRUEs and return the indexes of the outliers** |  |
| Get number of rows/columns  dim() | **Returns no. rows (obs) and columns (variables)**  **Stands for dimensions**  **dim(db)** |  |
| Directly copy dataset into Rstudio, by implicitly defining the variables based on column names  attach()  Note spelling! | **attach(db\_name)**  **Now the name of the dataset is fev**  **Can reference the variables/column directly eg.**  **Forced\_Expiratory\_Volume <- read.csv("~/GitHub/DSA1101 Slayers/datasets/FEV.csv")**  **attach(Forced\_Expiratory\_Volume)**  **hist(FEV, col = 10, freq = FALSE)**  **#where FEV is the name of the variable from the Forced\_Expiratory\_Volume db** | Tutorial 2 |
| Is the sample of (variable) normally distributed? | **USE QQPLOT. IF QN DOESNT ASK, INITIATE IT YOURSELF**  **qqnorm(fev\_var, pch = 10)**  **qqline(fev\_var, col = "red")** | Tutorial 2 |
| predict() | **predict(model\_fit, newdata = , type = “prob” )**  **type = “class” will just give u the most likely category.**  **Note that if categorical value, keep the factor = c(“3”) or factor = c(“1”) yada yada**  **Note if new data has multiple predictors/catagories, put the, in a data frame eg new\_data = data.frame(width = 5, length = “yes”)**  **NEW DATA IS LIKE ALL THE PREDICTORS, DO NOT PUT UR RESPONSE VARIABLE AS UR NEW DATA**  **Someone tell me why the FUCK new.data and newdata are 2 DIFFERENT BUT COMPLETELY VALID INPUT VARIABLES**  **USE newdata** |  |
| lm(price ~ area)$coeff | **Linear regression**  **M1 = lm()**  **summary(M1)** |  |
| geom\_bar usage  How to get bar chart to go of proportions (like percentage, in a piechart, but in bar from) | email %>%  mutate(has\_image = image>0) %>%  ggplot(aes(x = has\_image, fill = spam)) +  geom\_bar(position = "fill") | Datacamp |
| correlation  cor(heighty, fev\_var) | cor(1st\_var, 2st\_var)  #does NOT matter which variable goes first  #idt it works for categorical variables |  |
| Create a decision tree  Libraries:   * rpart * rpart.plot   Relevant functions:   * rpart() * rpart.control() | fit <- rpart(subscribed ~ job + marital + education + default + housing + loan + contact + poutcome,  method = “class”,  data = bankdata,  control = rpart.control(minsplit = 1),  parms = list(split = "information"))  method = "class", #tells R to return the category of response (yes or no)[DSA1101 will only use method = “class”, no “anova” or “poision”]  METHOD IS A STRING VARIABLE HOR ie  method = **"**class**"**  control = rpart.control(minsplit = 1), #tells R how to split the node; minsplit = 1 means a branch is created when there is at least 1 observation in that branch => ***tells R how big we want the tree to be*** (how many branches we want)  Can also split by complexity parameter:  control = rpart.control(cp = 0.001)  parms = list(split = "information")) # what criteria to use to choose the root node & internal nodes; either info gained or gini index; default val is gini  **TO PRINT THE TREE:**  rpart.plot(fit, type = 4, extra = 2, clip.right.labs = FALSE) | Topic 5 Pre-lect vid 3 |
| standardise all the predictors (the x es) | Lag1 = scale(Lag1) |  |
| Get the inverse of a matrix  Transpose the matrix  Multiply matrix by matrix | solve(matrix\_name)  t(matrix\_name)  matrix\_name %\*% matrix\_name |  |
| Create a matrix | cbind(1st\_desired\_col, 2nd\_desired\_col)  [bind the columns together]  Eg cbind(1, SAT) #SAT is variable name  returns  SAT  [1,] 1 1315  [2,] 1 1310  [3,] 1 1336  [4,] 1 1300  [5,] 1 1250  [6,] 1 1320  [7,] 1 1290  Or can do cbind(c(1,2), c(3,4))  [,1] [,2]  [1,] 1 3  [2,] 2 4  ALTERNATIVELY there is rbind()  rbind(1st\_desired\_row, 2nd\_desired\_row)  [bind the rows together] |  |
| Create scatterplot (Without ggplot) | plot(x, y, pch = 20)  #notice its x then y please  #pch is just the shape of the points, can ignore  For this type (type = “b”),  plot(1:K, wss, col = "blue", type="b", xlab="Number of Clusters", ylab="Within Sum of Squares") |  |
| Get R square value / R^2 / R2  Get adjusted R square | M1 = lm(price ~ size)  summary(M1) # take from here  OR  summary(hdb.model)$r.squared  adjusted R square:  summary(M1)**$adj.r.squared** |  |
| Form equation of Linear Model  M2$coeff | M2$coeff  ## (Intercept) size NW1  ## -15257.51385 77.98513 30569.08729  M2$coeff[1]: β0  M2$coeff[2]: β1  M2$coeff[2]: β2 |  |
| Cheatcode for equation of general linear model | paste0("log[phat/(1-phat)] = ", M2$coeff[1]," + ", M2$coeff[2], "Life\_expectancy + ", M2$coeff[3], "Adult\_mortality + ", M2$coeff[4], "infant\_deaths + ", M2$coeff[5], "Alcohol")  paste0("log[phat/(1-phat)] = ", M3$coeff[1]," + ", M3$coeff[2], "\*I(Age = Child) + ", M3$coef f[3], "\*I(Sex = Male) + ", M3$coeff[4], "\*I(Class = 2nd) + ", M3$coeff[5], "\*I(Class = 3rd) + ", M3$coeff[6], "\*I(Class = Crew)")  log[phat/(1-phat)] = 2.04383742228878 + 1.06154237626018\*I(Age = Child) + -2.42006034 580599\*I(Sex = Male) + -1.01809495158843\*I(Class = 2nd) + -1.77776221774166\*I(Class = 3rd) + -0.857676155374125\*I(Class = Crew) |  |
| knn() | Requires library(class)  knn.pred = knn(train.x, test.x, train.y, k = 1) | Documented in Topic 4 my own code |
| Create histogram  (without ggplot) | hist(Temperature, main = "Maximum daily temperature at La Guardia Airport", xlab = "Temperature in degrees Fahrenheit", xlim = c(50,100), col = "darkmagenta", freq = FALSE) |  |
| Overlay density plot line over histogram | lines(density(data$x), col = "red") |  |
| Sample data randomly  sample(values, size\_of\_subsample) | sample(my\_vec, size = 3)  sample(1:10, size = 3)  Will randomly give 3 numbers from 1 to 10  folds\_j <- sample(rep(1:n\_folds, length.out = n))  Eg  > sample(rep(1:4, length.out = 15))  [1] 1 3 4 4 3 2 2 2 3 1 3 1 2 1 4  > sample(rep(1:4, length.out = 15))  [1] 3 4 1 4 3 1 2 1 3 1 2 3 4 2 2 | **Rep basically duplicates the vector however many times**  #define vector  x <- c(1, 10, 50)  #replicate the vector three times  rep(x, times=3)  [1] 1 10 50 1 10 50 1 10 50 |
| standardised features  scale(all\_relavent\_predictors) | standardized.X = scale(caravan[,-86]) # scaling all the data set, except the last column |  |
| Forming Naive Bayes Classifier  Requires library(e1071) | model <- naiveBayes(Enrolls ~ Age+Income+JobSatisfaction+Desire, traindata)#, laplace=0)  model <- naiveBayes(response\_var ~ predictor1 + predictor2, traindata)  Then use:  results <- predict(model,testdata,"raw"); results  results[2]/results[1] |  |
| Create ROC / Find AUC  Requires library(ROCR) | Drop unnecessary columns first.  Idea: use testing data, see how well the model does. So split the test set into predictors and response.  #prelude (still using predict)  nb\_prediction = predict(nb\_model,  newdata = banktest[,-ncol(banktest)],  type = ‘raw’)  NOTE its banktest[,-ncol(banktest)] WITH THE NEGATIVE SIGN SO ITS EVERYTHING **BUT** THE LAST COLUMN => ***REMOVES THE RESPONSE VARIABLE***  type = ‘raw’ will give raw probabilities  If type = “response” doesnt work than idk just change to “class” or “raw” instead  library(ROCR)  score <- nb\_prediction[,c(“yes”)]  #this will return all the probabilities that each outcome will be positive (or ‘yes’).  #so if the prediction of ‘yes’ is 0.9, it’s very likely to be yes,, if the prediction of ‘yes’ is 0.001 then its very likely to be no  actual\_class <-   banktest$subscribed == ‘yes’ #this converts all to TRUE and FALSE which will be treated as 1 or 1 (due to how the method works)  pred <- prediction(score , actual\_class)  #this formats the input so ‘score’ and ‘actual\_class’ match with each other  perf <- performance(pred , "tpr", "fpr")  #now we can plot the figure  plot(perf, lwd = 2) #lwd specifies how thick the curve is  abline(a = 0,  b = 1,  col = ”blue”,  lty = -3) #create diagonal curve that slices square into half by its diagonal | Tutorial 8 and 9 |
| Predict for ROC / AUC  Ie using ROCR to predict  This is the main bits, repeated from previous row.  prediction() followed by performance(), then plot() | #TO FORMAT THE INPUT  pred <- prediction(score , actual\_class)  perf <- performance(pred , "tpr", "fpr")  plot(perf, lwd =2)  abline(a=0, b=1, col ="blue", lty =3) |  |
| Determine AUC value | AUC value determines what is the best threshold for a negative to become a positive (usually delta)  threshold <- round(as.numeric(unlist(perf@alpha.values)))  fpr <- round(as.numeric(unlist([perf@x.values](mailto:perf@x.values))), 4)  tpr <- round(as.numeric(unlist([perf@y.values](mailto:perf@x.values))), 4)  #adjust margins and plot TPR and FPR  par(mar = c(5,5,2,5))  #the plotting stuff: just copy paste LMAO  plot(alpha ,tpr , xlab ="Threshold", xlim =c(0 ,1) ,  ylab = "True positive rate ", type ="l", col = "blue")  par( new ="True")  plot(alpha ,fpr , xlab ="", ylab ="", axes =F, xlim =c(0 ,1) , type ="l", col = "red" )  axis( side =4) # to create an axis at the 4th side  mtext(side =4, line =3, "False positive rate")  text(0.18 ,0.18 , "FPR")  text(0.58 ,0.58 , "TPR") | <https://sets.netlify.app/module/65d9a044a1d9248bb0c34963/topic/65d9a076a1d9248bb0c34b55/6619fd77da66393bf2be69e8> not relavent |
| AUC Simplified | aucLM = performance(pred, measure = "auc")  aucLM@y.values[[1]] #AUC value = 0.8895238 |  |
| Final code for ROC / AUC for **KNN** | pred.knn = knn(train.X, test.X, cl = train.Y, k = 30) # "prob = TRUE" is NOT added --> to get the class labels  pred.knn.prob= knn(train.X, test.X, cl = train.Y, k = 30, prob = TRUE) # --> to get the probabilities of winning labels  winning.prob = attr(pred.knn.prob, "prob") # to extract the winning probabilities  n = length(test.Y)  prob = numeric(length = 0) # n is the length of "test.y" used in knn() above  for (i in 1:n) {  prob[i] = ifelse(pred.knn[i] == "yes", winning.prob[i], 1 - winning.prob[i])  }  helppain = prediction(prob, test.Y)  rocObjKNN = performance(helppain, measure = "tpr", x.measure = "fpr")  plot(rocObjKNN, lwd =2)  abline(a=0, b=1, col ="blue", lty =3) | Refer to statistical assignment (and below this document) |
| Attain accuracy (trust me bro) | M3 <- naiveBayes(Status ~ Life\_expectancy + Adult\_mortality + infant\_deaths + Alcohol,  data)  results <- predict(M3, data[,2:5], "class"); results  confusion.matrix = table(Status, results)  confusion.matrix  diag(confusion.matrix)  **accuracy = sum(diag(confusion.matrix))/sum(confusion.matrix)**  accuracy | Refer to Statistical Report |
| Logistic regression  Generalised Linear Regression (GLM) | M1 <- glm(Churned ~ . , # . is all columns except the response col  data = churn,  family = binomial(link = "logit")) #logit is default tho  Note: Churned must be equal to 0 and 1, it can be as.factor(), but the category names MUST be 0 or 1. Ironically it doesn't matter if you converted to factor, for glm.  REMEMBER ur LHS will be log(p/(1-p)), the log odds of having disease or wtvr. So the next step is finding p itself. |  |
| Interpret coefficient of logistic regression / generalised linear model | The coefficient of Alcohol is 0.148746770871717.  # When the alcohol consumption (per capita (15+) consumption, in liters of pure alcohol) increases by 1 unit, then the log odds of Status changing from developing to developed increases by 0.149, keeping all other variables the same.  # When alcohol consumption increases by 1 unit, then the odds of being a developed country changes by e^0.148 times, keeping all other variables the same  When the number of contacts increases by 1 unit, then the log odds of churning increases by [coefficient of contacts], keeping all other variables the same. => When the number of contacts increases by 1 unit, then the odds of success changes by e^0.38 times (or e^(coefficient of contacts) times), keeping all other variables the same  How to interpret married: Comparing a married person (Married = 1) vs a non-married person, when other variables are the same, then the log odds of churning will be larger by 0.07. => Comparing a married person (Married = 1) vs a non-married person, then the odds of churning will change by e^0.07 times. | Refer to past year paper working OR Topic-7-MY OWN code Logistic Regression |
| k-means clusters  No external library needed | kout = kmeans(hdb[,c("floor\_area\_sqm", "amenities")],  centers = 2) #centers = nummber of clusters  #note kmeans data only accepts dataframes  **Plot the clusters**  plot(floor\_area\_sqm,  hdb$amenities,  col = kout$cluster) #col is colours, so colours differentiate the 2 clusters  **Centroids of the clusters**  kout$centers #a matrix of cluster centers  # floor\_area\_sqm amenities  # 1 69.46017 7.448739  # 2 65.40782 37.279330  # so these are the coordinates of the 2 centroids (of the 2 clusters)  **Size of each cluster**  kout$size #the number of pts in each cluster  # [1] 595 179  **Within Sum of Squares**  kout$withinss  #sum of squares of all the different clusters added together  kout$tot.withinss | STANDARDISE THE INPUTS before starting K-means   * Use scale()   If doing manually, ie given all the indiv plot points, USE plot() TO DETERMINE WHERE ALL THE PLOTS ARE  REMEMBER K-MEANS IS VERY SCATTERPLOT BASED |
| Association Rules | library(arules)  library(arulesViz)  Groceries@itemInfo[1:10,] #all columns  Groceries@data[,100:110] #all rows  #note the 2nd one will give u the sparse matrix (unreadable dot and slash) |  |
| Return the actual itemsets (instead of the general data / summary) | inspect()  inspect(sort(itemset.1, by = "support")) |  |
| Attain itemsets that fit a certain support level | itemset.1 = apriori(Groceries, parameter = list(minlen = 1, maxlen = 1, support = 0.02, target = "frequent itemsets"))  #so here itemsets that hit the minimum support of 0.02 (2%) are considered frequent itemsets  #bc minlen = 1 and maxlen = 1, all returned itemsets only have 1 item (each itemset has a length of 1)  summary(itemset.1) | As support increases, number of returned itemsets decreases |
| Attain and plot all the rules | rules = apriori(Groceries, parameter = list(support = 0.001, confidence = 0.6, target = "rules"))  plot(rules)  #limit to only 100 dots / rules with the highest lift  plot(rules, measure = c("support", "confidence"), shading = "lift", col = "black", **limit = 100**)  **Attain the top few rules**  inspect(head(sort(rules, by = "lift"), 3))  #note that 3 is a parameter for head() ie head() controls how many outputs come out  **Plot the 5 rules with the highest lift**  highLiftRules = head(sort(rules, by = "lift"), 5)  plot(highLiftRules, method = "graph"  (refer below for a plot that is more easily readable) |  |
| *Interpreting the weirdass association rules plot* | highLiftRules = head(sort(rules, by = "lift"), 5)  plot(highLiftRules, method = "graph", engine = "igraph", edgeCol = "blue", alpha = 1)    As size increases, support increases  As colour darkens, lift increases    Those who buy instant food products AND soda, also tend to buy hamburger meat |  |
| Inspect / List the first 6 transactions | inspect(head(data)) #gives first 6 itemsets  inspect(head(data, 10)) | requires arules library |
| Add text to a plot  Manually add a point to a scatter plot | text(xcord, ycord, “text to insert”)  points(xcord, ycord, pch = 20, col = “green”) | points() WITH AN S ITS PLURAL!!! |
| Within Sum of Squares (WSS) | K = 15  wss <- numeric(K)#this creates an empty vector of 15 0s  for (k in 1:K) {  wss[k] <- sum(kmeans(scale(data[,c("floor\_area\_sqm","resale\_price")]), centers=k)$withinss)  }  plot(1:K, wss, col = "blue", type="b", xlab="Number of Clusters", ylab="Within Sum of Squares")  My own attempt is this:  kdata = scale(sendhelp[, c("floor\_area\_sqm","resale\_price")])  wss = numeric(15)  for (i in 1:15) {  wss[i] = sum(kmeans(kdata, centers = i)$withinss)  }  plot(1:15, wss, type = "b") | By default just plot everything first to get a sense of the data  Remember to scale the data before doing kmeans()! |
| Convert sparse matrix into readable itemset | apply(Groceries@data[,100:105], 2,  function(r) paste(Groceries@itemInfo[r, "labels"], collapse = ", ")) |  |

**NOTES FROM MIDTERM:**

1. **Remember to as.factor() EVERYTHING. Not sure? CHECK IF ANY CATEGORICAL VAR NEEDS TO BE CONVERTED FROM INT TO CAT. AS.FACTOR() EVERYTHING**
   1. **If attaching dataframe,**
      1. **breastdata$survival.status = as.factor(breastdata$survival.status) use the dollar signs first!!!**
      2. **THEN attach, after converting to factor**
      3. **CHECK AGAIN using glimpse or head**
      4. **breastdata$survival.status = as.factor(breastdata$survival.status)  
         attach(breastdata)  
         glimpse(breastdata)**
2. **Removing specific columns: DO NOT  
   breastdata = breastdata[,1] + breastdata[,3] + breastdata[,4]**
   1. **INSTEAD: breastdata = breastdata[,c(1,3,4)]**
   2. **OR to remove ID column / first column:  
      breastdata = breastdata[,-1]**
3. **Type 1 error: False Positive Rate (FPR)  
   Type 2 Error: False Negative Rate (FNR)**
4. **3 sf please!!**

**INCLUDING PICTURES IN R MARKDOWN:**

**```{r,eval=TRUE, echo=FALSE}**

**knitr::include\_graphics("~/GitHub/DSA1101 Slayers/Pictures/Confusion Matrix Pic.jpg")**

**knitr::include\_graphics("~/GitHub/DSA1101 Slayers/Pictures/CM spam emails.jpg")**

**```**

Hi Bryan, this is Dawn from DSA1101 T3, I wanted to ask about a mistake from assignment 2 (statistical report). I don’t understand this comment: “Wrong KNN ROC code (should use 1-attr)”

Hi,

Sorry to reply your question late. I'm stuffed with full schedule these days. :(((

Let's try to understand how ROC curve is plotted, then we can know the difference:

(1) the ROC curve uses the probability of response = yes, Pr (Y = 1) from model.

(2) for each value of threshold delta, like delta = 0.1, compare Pr(Y = 1) vs delta, if a data point has Pr(Y= 1) > delta then that point is predicted as "Yes" or "1"; otherwise that point will be classified as "No" or "0".

(3) With the classification as in (2) be applied to all the data points for each value of delta, the value of TPR and FPR are calculated for each delta value.

(4) With a series of delta from 0 to 1 (where R will generate this series on its own), the values of TPR and FPR will be calculates for each delta in this series.

(5) ROC curve is plotted.

That's how the ROC curve is generated.

Now, back to KNN: when we use function knn() and do not specify "prob = TRUE", the output we get are the labels ("0" and "1" ) for each point.

When we use as.numeric() to transform the labels "0" and "1" into numeric, we get 0 and 1.

After that, we use those 0 and 1 to plot the ROC curve. Then, at the step 1 above, R will treat 0 and 1 as the probabilities Pr(Y = 1) for data points. That means, for all the data points given, each of them has probability Pr(Y = 1) is either 0 or 1, no other values.

That's why the curve has only 1 bend for this case.

When we run knn() with "prob = TRUE", we will be able to get the probability of response variable equal to the WINNING category.

For example, Y has two category: Yes and No, then the output of

pred.knn = knn(... , prob = TRUE)

winning.prob = attr(pred.knn, "prob")

is a series of winning probabilities which are: the probability that Y = "Yes" if the point is classified as Yes and probability that Y = "No" is the point is classified as No.

To get the probability of Yes for every data point in this case, you might want to run

(here, I'm assuming the response has "yes" and "no"):

pred.knn = knn(train.X, test.X, cl = train.Y, k = 30) # "prob = TRUE" is NOT added --> to get the class labels

pred.knn.prob= knn(train.X, test.X, cl = train.Y, k = 30, prob = TRUE) # --> to get the probabilities of winning labels

winning.prob = attr(pred.knn.prob, "prob") # to extract the winning probabilities

n = length(test.Y)

prob = numeric(length = 0) # n is the length of "test.y" used in knn() above

for (i in 1:n) {

prob[i] = ifelse(pred.knn[i] == "yes", winning.prob[i], 1 - winning.prob[i])

}

Then, "prob" will be the probabilities of response be "Yes" for every point in the test set.

With this vector "prob", you can use to plot the ROC curve and calculate its AUC values.

helppain = prediction(prob, test.Y)

rocObjKNN = performance(helppain, measure = "tpr", x.measure = "fpr")

plot(rocObjKNN, lwd =2)

abline(a=0, b=1, col ="blue", lty =3)

How do I plot ROC for Naive Bayes and Logistic regression with class label, instead of probability?

With class label, like "0" and "1", you will need to change them to be numeric, as 0 and 1. For example:

x = as.factor(c(0, 1, 1, 0)) # x is a vector of labels "0" and "1"

y = as.numeric(paste(x)) # this will change vector x into y = a numeric vector

different types of models requires different ways to input for “type” in predict() function:

For Naïve Bayes, you have type = “raw” to get the probabilities of “1” and “0”; if you specify type = “class” then you get the predicted class label, either “1” or “0” only.

For DT, you can specify type = “class”, or type = “prob”.

For logistic regression, you should specify type = “response” to get the probability of success.

For KNN, when running knn(). If you specify prob = TRUE, you can get the probability of the winning class. You might check my 2 answers for Q83 above.

That’s how R defines for function predict() for different type of models.

We should choose a lower threshold (than 0.5) when the success criteria is very rare ie a very low percentage of the population will be positive

#how tpr and fpr changes when threshold changes

```{r getting the precise best alpha value, eval = TRUE, echo=TRUE}

cbind(alpha, tpr, fpr)

```

for churn data, delta is around 0.24

this lines up with the prop.table(table(Churned)) value where 0.217875 ppl churned (proportion of 'yes' in the data), so delta should be AROUND there (need not be exact)

The exact delta used is up to you, can depend on the users needs (if they want to prioritise TPR over FPR, etc)

Each value of delta will correspond to a value of TPR and FPR.

With 0.5, you can find TPR and FPR to see if both of them are good. If both of them are good then suitable; if any of them is not good then not suitable. :)