# IST 707 - Applied Machine Learning - Final Project

## Name: Sean Deery, Mark Stiles, Chris Drabb

## Date: 2023-06-08

# General

The following analysis will focus on the Laptop League dataset. The dataset is a product inventory from the [FlipKart website] (<https://www.flipkart.com/laptops-store>). Laptops are an important tool for many people. There are lots of options and factors to consider when buying one. It’s easy to get overwhelmed by the number of variations but there’s a lot of standard configurations that you can choose from to meet your specific need. This analysis will aim to answer what those common configurations are, what price points they break down into and to ultimately predict the price given a specific set of options .

## Setup and Libraries

## ── Attaching packages ─────────────────────────────────────── tidyverse 1.3.2 ──  
## ✔ ggplot2 3.3.6 ✔ purrr 0.3.4  
## ✔ tibble 3.1.8 ✔ dplyr 1.0.9  
## ✔ tidyr 1.2.1 ✔ stringr 1.4.1  
## ✔ readr 2.1.3 ✔ forcats 0.5.2  
## ── Conflicts ────────────────────────────────────────── tidyverse\_conflicts() ──  
## ✖ dplyr::filter() masks stats::filter()  
## ✖ dplyr::lag() masks stats::lag()

## Warning: package 'arules' was built under R version 4.2.3

## Loading required package: Matrix  
##   
## Attaching package: 'Matrix'  
##   
## The following objects are masked from 'package:tidyr':  
##   
## expand, pack, unpack  
##   
##   
## Attaching package: 'arules'  
##   
## The following object is masked from 'package:dplyr':  
##   
## recode  
##   
## The following objects are masked from 'package:base':  
##   
## abbreviate, write

## Warning: package 'arulesViz' was built under R version 4.2.3

## Warning: package 'GGally' was built under R version 4.2.3

## Registered S3 method overwritten by 'GGally':  
## method from   
## +.gg ggplot2

## Warning: package 'naivebayes' was built under R version 4.2.3

## naivebayes 0.9.7 loaded  
## Loading required package: lattice  
##   
## Attaching package: 'caret'  
##   
## The following object is masked from 'package:purrr':  
##   
## lift  
##   
## Loading required package: gsubfn  
## Loading required package: proto  
## Loading required package: RSQLite

## Warning: package 'randomForest' was built under R version 4.2.3

## randomForest 4.7-1.1  
## Type rfNews() to see new features/changes/bug fixes.  
##   
## Attaching package: 'randomForest'  
##   
## The following object is masked from 'package:dplyr':  
##   
## combine  
##   
## The following object is masked from 'package:ggplot2':  
##   
## margin

## Warning: package 'mclust' was built under R version 4.2.3

## Package 'mclust' version 6.0.0  
## Type 'citation("mclust")' for citing this R package in publications.  
##   
## Attaching package: 'mclust'  
##   
## The following object is masked from 'package:purrr':  
##   
## map  
##   
## Welcome! Want to learn more? See two factoextra-related books at https://goo.gl/ve3WBa

# setup colors  
syracuseOrange <- "#D44500"  
syracuseBlue <- "#0C233f"

### Functions

Many of the repeatable processes have been converted to functions and loaded first to be used as-needed.

# This function splits a dataset by a percentage  
get\_split = function(data\_set, percent\_amount)  
{  
 split\_set = sample(nrow(data\_set),nrow(data\_set)\*percent\_amount)   
 return(split\_set)  
}  
  
# This function measures the accuracy of a prediction result and returns a percentage  
get\_accuracy\_rate = function(results\_table, total\_cases)   
{  
 diagonal\_sum = sum(diag(results\_table))  
 acc = (diagonal\_sum / total\_cases)\*100  
 return(acc)  
}  
  
# The next set of functions are wrappers for the train\_model function with presets  
train\_bayes = function(data\_set)  
{  
 results = train\_model(data\_set, "bayes")  
 return(results)  
}  
  
train\_tree = function(data\_set, control\_obj\_value)  
{  
 results = train\_model(data\_set, "tree", control\_obj=control\_obj\_value)  
 return(results)  
}  
  
train\_knn = function(data\_set, kguess\_value)  
{  
 results = train\_model(data\_set, "knn", kguess\_num=kguess\_value)  
 return(results)  
}  
  
train\_svm = function(data\_set, kernel\_type\_value, cost\_num\_value)  
{  
 results = train\_model(data\_set, "svm", kernel\_type=kernel\_type\_value, cost\_num=cost\_num\_value)  
 return(results)  
}  
  
train\_forest = function(data\_set, replace\_type\_value)  
{  
 results = train\_model(data\_set, "forest", replace\_type=replace\_type\_value)  
 return(results)  
}  
  
# This function handles the CV folding in the same way for each model. The specified model is trained and then runs a prediction on holdout values  
train\_model = function(data\_set, model\_type, kguess\_num=7, kernel\_type="radial", replace\_type=FALSE, cost\_num=1, control\_obj=NULL, distance\_method="euclidean", cluster\_method="ward.D")  
{  
 kfolds = 5  
 holdout\_set <- split(sample(1:nrow(data\_set)), 1:kfolds)  
   
 all\_results <- data.frame(orig=c(), pred=c())  
 for (k in 1:kfolds)   
 {  
 new\_test <- data\_set[holdout\_set[[k]], ]  
 new\_train <- data\_set[-holdout\_set[[k]], ]  
 new\_test\_no\_label <- new\_test[-c(1)]  
 new\_test\_just\_label <- new\_test[c(1)]  
   
 pred\_values = NULL  
 if (model\_type == "knn")  
 {  
 pred\_values = knn(train=new\_train, test=new\_test, cl=new\_train$CostCategory, k=kguess\_num, prob=FALSE)   
 }  
 else if (model\_type == "svm")  
 {  
 svm\_model = svm(CostCategory ~ ., new\_train, kernel=kernel\_type, na.action=na.pass, cost=cost\_num)  
 pred\_values = predict(svm\_model, new\_test\_no\_label, type=c("class"))  
 }  
 else if (model\_type == "forest")  
 {  
 forest\_model = randomForest(CostCategory ~ ., new\_train, replace=replace\_type, na.action=na.pass)  
 pred\_values = predict(forest\_model, new\_test\_no\_label, type=c("class"))  
 }  
 else if (model\_type == "bayes")  
 {  
 bayes\_model = naiveBayes(CostCategory ~ ., new\_train, na.action=na.pass)  
 pred\_values = predict(bayes\_model, new\_test\_no\_label)  
 }  
 else if (model\_type == "tree")  
 {  
 tree\_model <- rpart(CostCategory ~ ., new\_train, method="class", control=control\_obj)  
 pruned\_tree\_model <- prune(tree\_model, cp=tree\_model$cptable[which.min(tree\_model$cptable[,"xerror"]),"CP"])  
 pred\_values = predict(pruned\_tree\_model, new\_test\_no\_label, type="class")  
 }  
   
 all\_results = rbind(all\_results, data.frame(orig=new\_test\_just\_label$CostCategory, pred=pred\_values))  
 }  
 table\_results = table(all\_results$orig, all\_results$pred)  
 accuracy\_results = get\_accuracy\_rate(table\_results, length(all\_results$pred))  
   
 return(accuracy\_results)  
}  
  
# This function will scale down data to between zero and one  
normalize <- function(x, column) {  
 return ((x[,column] - min(x[,column])) / (max(x[,column]) - min(x[,column])))  
}

# Analysis and Models

## Laptop Data

Laptop League: A Comprehensive Dataset for Laptops - <https://www.kaggle.com/datasets/shrutiambekar/laptop-league-a-comprehensive-dataset-for-laptops>

# Variables  
  
# Company: This column represents the Laptops Company name.  
# Rating: The average rating (out of 5) of the laptop based on user reviews.  
# No\_of\_ratings: The number of ratings provided by users for the laptop.  
# Review: A brief summary of the user reviews for the laptop.  
# Size: The size of the laptop's screen in Cm.  
# Processor: The brand and model of the laptop's processor.  
# RAM: The amount of Random Access Memory (RAM) in the laptop in gigabytes (GB).  
# Memory: The storage capacity of the laptop's hard disk drive (HDD) or solid-state drive (SSD) in gigabytes (GB).  
# OpSys: The operating system installed on the laptop, such as Windows, macOS, or Linux or Others  
# Price: The current price of the laptop in INDIAN INR().  
# MRP: The manufacturer's suggested retail price (MRP)INDIAN INR()  
# ImgURL: The URL of the image of the laptop.

### loading the data

laptops <- read.csv("Laptop\_Information.csv")  
str(laptops)

## 'data.frame': 984 obs. of 13 variables:  
## $ X : int 0 1 2 3 4 5 6 7 8 9 ...  
## $ Company : chr "HP" "Lenovo" "ASUS" "DELL" ...  
## $ Rating : num 4.2 4.2 4.4 4.1 4.2 4.1 4.6 4.3 4.2 4.1 ...  
## $ No\_of\_ratings: int 253 974 5389 332 487 4541 9 2841 491 1022 ...  
## $ Review : int 20 75 535 26 53 523 1 241 50 123 ...  
## $ Size : num 36 36 40 40 40 40 40 40 40 36 ...  
## $ Processor : chr "AMD Ryzen 5 Hexa Core Processor" "Intel Core i3 Processor (11th Gen)" "Intel Core i5 Processor (10th Gen)" "Processor: Intel i3-1115G4 (Base- 1.7 GHz & Turbo up to 4.10 GHz) 2 Cores" ...  
## $ RAM : int 16 8 8 8 8 8 8 8 8 4 ...  
## $ Memory : chr "512 GB SSD" "256 GB SSD" "512 GB SSD" "Display: 15.6\" FHD WVA AG Narrow Border" ...  
## $ OpSys : chr "64 bit Windows 11 " "64 bit Windows 11 " "Windows 11 " "Graphics & Keyboard: Integrated & Standard Keyboard" ...  
## $ Price : int 49990 33990 49990 37990 35990 38990 34990 55990 40450 23990 ...  
## $ MRP : int 59240 60890 70990 58489 50990 59999 44253 83890 50585 33990 ...  
## $ ImgURL : chr "https://rukminim1.flixcart.com/image/312/312/l5fnhjk0/computer/c/p/c/14s-fy1005au-thin-and-light-laptop-hp-orig"| \_\_truncated\_\_ "https://rukminim1.flixcart.com/image/312/312/keaaavk0/computer/c/q/t/lenovo-na-thin-and-light-laptop-original-i"| \_\_truncated\_\_ "https://rukminim1.flixcart.com/image/312/312/l3rmzrk0/computer/s/z/r/-original-imagetgzg4pgszmt.jpeg?q=70" "https://rukminim1.flixcart.com/image/312/312/xif0q/computer/x/o/u/-original-imaghkk994ybh4fh.jpeg?q=70" ...

### Null Checking

sum(is.na(laptops))

## [1] 0

### Cleanup and Feature Creation

# fix outlier sizes  
laptops[laptops$Size == 103,]$Size = 41  
laptops[laptops$Size == 101,]$Size = 39  
laptops[laptops$Size == 97,]$Size = 38  
laptops[laptops$Size == 89,]$Size = 35  
  
# adding missing or incorrect values based on manual lookups to the source site of the dataset  
  
# setting conditions  
aw\_cond = laptops$Company == "ALIENWARE"  
price\_cond\_1 = laptops$Price == 188490  
price\_cond\_2 = laptops$Price == 202490  
price\_cond\_3 = laptops$Price == 250000  
price\_cond\_4 = laptops$Price == 325990  
# alienware uses windows  
laptops[aw\_cond,]$OpSys = "64 bit Windows 10 Operating System"  
# low end alienware  
laptops[aw\_cond & price\_cond\_1,]$Memory = "512 GB SSD"  
laptops[aw\_cond & price\_cond\_1,]$Processor = "Intel Core i7 Processor (10th Gen)"  
laptops[aw\_cond & price\_cond\_1,]$RAM = "16"  
# mid range alienware  
laptops[aw\_cond & (price\_cond\_2 | price\_cond\_3),]$Memory = "1 TB SSD"  
laptops[aw\_cond & (price\_cond\_2 | price\_cond\_3),]$Processor = "Intel Core i7 Processor (10th Gen)"  
laptops[aw\_cond & (price\_cond\_2 | price\_cond\_3),]$RAM = "16"  
# high end alienware  
laptops[aw\_cond & price\_cond\_4,]$Memory = "1 TB SSD"  
laptops[aw\_cond & price\_cond\_4,]$Processor = "Intel Core i9 Processor (10th Gen)"  
laptops[aw\_cond & price\_cond\_4,]$RAM = "32"  
# chrome has one memory size  
laptops[grep("Chrome", laptops$OpSys),]$Memory = "64 GB EMMC"  
# update keyed on nvidia specs that matched with a brand in site search results  
laptops[grep("1050", laptops$Processor),]$Memory = "1 TB HDD"  
laptops[grep("1050", laptops$Processor),]$OpSys = "64 bit Windows 10 Operating System"  
laptops[grep("1050", laptops$Processor),]$Processor = "Intel Core i5 Processor (7th Gen)"  
# same nvidia specs new field  
laptops[grep("1050", laptops$OpSys),]$Processor = "Intel Core i7 Processor (7th Gen)"  
laptops[grep("1050", laptops$OpSys),]$Memory = "1 TB HDD"  
laptops[grep("1050", laptops$OpSys),]$OpSys = "Linux"  
# new nvidia specs  
laptops[grep("1650", laptops$Processor),]$Memory = "512 GB SSD"  
laptops[grep("1650", laptops$Processor),]$OpSys = "64 bit Windows 10 Operating System"  
laptops[grep("1650", laptops$Processor),]$Processor = "Intel Core i5 Processor (10th Gen)"  
# MSI nvidia specs  
laptops[intersect(grep("2060", laptops$Processor),which(laptops$Company == "MSI",TRUE)) ,]$Memory = "512 GB SSD"  
laptops[intersect(grep("2060", laptops$Processor),which(laptops$Company == "MSI",TRUE)) ,]$OpSys = "64 bit Windows 10 Operating System"  
laptops[intersect(grep("2060", laptops$Processor),which(laptops$Company == "MSI",TRUE)) ,]$Processor = "Intel Core i7 Processor (8th Gen)"  
# DELL nvidia specs  
laptops[intersect(grep("2070", laptops$Processor),which(laptops$Company == "DELL",TRUE)) ,]$Memory = "1 TB SSD"  
laptops[intersect(grep("2070", laptops$Processor),which(laptops$Company == "DELL",TRUE)) ,]$OpSys = "64 bit Windows 10 Operating System"  
laptops[intersect(grep("2070", laptops$Processor),which(laptops$Company == "DELL",TRUE)) ,]$Processor = "Intel Core i9 Processor (10th Gen)"  
# DELL new nvidia specs and unique review value  
laptops[intersect(grep("3050", laptops$Processor),which(laptops$Company == "DELL",TRUE)) ,]$Memory = "512 GB SSD"  
laptops[intersect(grep("3050", laptops$Processor),which(laptops$Company == "DELL",TRUE)) ,]$OpSys = "64 bit Windows 10 Operating System"  
laptops[intersect(grep("3050", laptops$Processor),which(laptops$Company == "DELL" & laptops$Review == 9,TRUE)) ,]$Processor = "AMD Ryzen 5 Hexa Core Processor"  
laptops[intersect(grep("3050", laptops$Processor),which(laptops$Company == "DELL" & laptops$Review == 1,TRUE)) ,]$Processor = "AMD Ryzen 7 Octa Core Processor"  
# update where 16 gb was reported in operating system field  
laptops[grep("16 GB", laptops$OpSys),]$RAM = 16  
  
# Create a new column ProcessorManufacturer from the Processor Column  
laptops <- laptops %>% mutate(ProcessorManufacturer = case\_when(  
 grepl('Intel', Processor, ignore.case = TRUE) ~ 'Intel',  
 grepl('AMD', Processor, ignore.case = TRUE) ~ 'AMD',  
 grepl('R3-5425U', Processor, ignore.case = TRUE) ~ 'AMD',  
 grepl('R5-5625U', Processor, ignore.case = TRUE) ~ 'AMD',  
 grepl('NVIDIA', Processor, ignore.case = TRUE) ~ 'NVIDIA',  
 grepl('MediaTek', Processor, ignore.case = TRUE) ~ 'MediaTek',  
 TRUE ~ 'Unknown'  
 ))  
  
# Create a new column DiskGB from the Memory and Processor column  
laptops <- laptops %>% mutate(DiskGB = case\_when(  
 grepl('64 GB EMMC', Memory, ignore.case = TRUE) ~ '64',  
 grepl('1 TB SSD', Memory, ignore.case = TRUE) ~ '1000',  
 grepl('2 TB SSD', Memory, ignore.case = TRUE) ~ '2000',  
 grepl('512 GB SSD', Memory, ignore.case = TRUE) ~ '512',  
 grepl('512 GB SSD', Processor, ignore.case = TRUE) ~ '512',   
 grepl('256 GB SSD', Memory, ignore.case = TRUE) ~ '256',  
 grepl('128 GB SSD', Memory, ignore.case = TRUE) ~ '128',  
 grepl('1 TB HDD', Memory, ignore.case = TRUE) ~ '1000',  
 grepl('1 TB HDD|1 TB SSD', Memory, ignore.case = TRUE) ~ '2000',  
 grepl('1 TB HDD|512 TB SSD', Memory, ignore.case = TRUE) ~ '1512',  
 grepl('1 TB HDD|256 TB SSD', Memory, ignore.case = TRUE) ~ '1256',  
 grepl('512 TB HDD|512 TB SSD', Memory, ignore.case = TRUE) ~ '1024',  
 grepl('256 TB HDD|256 TB SSD', Memory, ignore.case = TRUE) ~ '512',  
 TRUE ~ 'Unknown'  
 ))  
  
# Create a new column OS from the OpSys and Memory column  
laptops <- laptops %>% mutate(OS = case\_when(  
 grepl('Windows', OpSys, ignore.case = TRUE) ~ 'Windows',  
 grepl('Windows', Processor, ignore.case = TRUE) ~ 'Windows',  
 grepl('Windows', Memory, ignore.case = TRUE) ~ 'Windows',  
 grepl('Win', OpSys, ignore.case = TRUE) ~ 'Windows',  
 grepl('Win', Memory, ignore.case = TRUE) ~ 'Windows',  
 grepl('Mac OS', OpSys, ignore.case = TRUE) ~ 'Mac OS',  
 grepl('Chrome', OpSys, ignore.case = TRUE) ~ 'Chrome OS',  
 grepl('Prime OS', OpSys, ignore.case = TRUE) ~ 'Prime OS',  
 grepl('DOS', OpSys, ignore.case = TRUE) ~ 'DOS',  
 grepl('Linux', OpSys, ignore.case = TRUE) ~ 'Linux',  
 TRUE ~ 'Unknown'  
 ))  
  
# Create a new column Cores from the Processor Column originating from the product website detail page  
laptops <- laptops %>% mutate(Cores = case\_when(  
 grepl('Dual', Processor, ignore.case = TRUE) ~ '2',  
 grepl('Quad', Processor, ignore.case = TRUE) ~ '4',  
 grepl('Hexa', Processor, ignore.case = TRUE) ~ '6',  
 grepl('Octa', Processor, ignore.case = TRUE) ~ '8',  
 grepl('2 Cores', Processor, ignore.case = TRUE) ~ '2',  
 grepl('6 Cores', Processor, ignore.case = TRUE) ~ '6',  
 grepl('10 Cores', Processor, ignore.case = TRUE) ~ '10',  
 grepl('14 Cores', Processor, ignore.case = TRUE) ~ '14',  
 grepl('Apple M1 Pro Processor', Processor, ignore.case = TRUE) ~ '8',  
 grepl('Apple M1 Max Processor', Processor, ignore.case = TRUE) ~ '10',  
 grepl('Apple M1 Processor', Processor, ignore.case = TRUE) ~ '8',  
 grepl('Apple M2 Max Processor', Processor, ignore.case = TRUE) ~ '10',  
 grepl('Apple M2 Pro Processor', Processor, ignore.case = TRUE) ~ '12',  
 grepl('Apple M2 Processor', Processor, ignore.case = TRUE) ~ '8',  
 grepl('Intel Core i3 Processor \\(10th Gen\\)', Processor, ignore.case = TRUE) ~ '4',  
 grepl('Intel Core i3 Processor \\(11th Gen\\)', Processor, ignore.case = TRUE) ~ '4',  
 grepl('Intel Core i3 Processor \\(12th Gen\\)', Processor, ignore.case = TRUE) ~ '4',  
 grepl('Intel Core i3 Processor \\(13th Gen\\)', Processor, ignore.case = TRUE) ~ '4',  
 grepl('Intel Core i3 Processor \\(7th Gen\\)', Processor, ignore.case = TRUE) ~ '2',  
 grepl('Evo Core i5 Processor', Processor, ignore.case = TRUE) ~ '6',  
 grepl('Intel Core i5 Processor \\(10th Gen\\)', Processor, ignore.case = TRUE) ~ '6',  
 grepl('Intel Core i5 Processor \\(11th Gen\\)', Processor, ignore.case = TRUE) ~ '6',  
 grepl('Intel Core i5 Processor \\(12th Gen\\)', Processor, ignore.case = TRUE) ~ '6',  
 grepl('Intel Core i5 Processor \\(13th Gen\\)', Processor, ignore.case = TRUE) ~ '6',  
 grepl('Intel Core i5 Processor \\(4th Gen\\)', Processor, ignore.case = TRUE) ~ '4',  
 grepl('Intel Core i5 Processor \\(7th Gen\\)', Processor, ignore.case = TRUE) ~ '4',  
 grepl('Intel Core i5 Processor \\(8th Gen\\)', Processor, ignore.case = TRUE) ~ '6',  
 grepl('Intel Core i5 Processor \\(9th Gen\\)', Processor, ignore.case = TRUE) ~ '6',  
 grepl('Intel Core i7 Processor \\(10th Gen\\)', Processor, ignore.case = TRUE) ~ '8',  
 grepl('Intel Core i7 Processor \\(11th Gen\\)', Processor, ignore.case = TRUE) ~ '8',  
 grepl('Intel Core i7 Processor \\(12th Gen\\)', Processor, ignore.case = TRUE) ~ '8',  
 grepl('Intel Core i7 Processor \\(13th Gen\\)', Processor, ignore.case = TRUE) ~ '8',  
 grepl('Intel Core i7 Processor \\(7th Gen\\)', Processor, ignore.case = TRUE) ~ '4',  
 grepl('Intel Core i7 Processor \\(8th Gen\\)', Processor, ignore.case = TRUE) ~ '6',  
 grepl('Intel Core i9 Processor \\(10th Gen\\)', Processor, ignore.case = TRUE) ~ '10',  
 grepl('Intel Core i9 Processor \\(11th Gen\\)', Processor, ignore.case = TRUE) ~ '8',  
 grepl('Intel Core i9 Processor \\(12th Gen\\)', Processor, ignore.case = TRUE) ~ '8',  
 grepl('Intel Core i9 Processor \\(13th Gen\\)', Processor, ignore.case = TRUE) ~ '8',  
 grepl('Intel Pentium Silver Processor', Processor, ignore.case = TRUE) ~ '4',  
 grepl('MediaTek MediaTek Kompanio 500 Processor', Processor, ignore.case = TRUE) ~ '8',  
 grepl('MediaTek MediaTek MT8788 Processor', Processor, ignore.case = TRUE) ~ '8',  
 grepl('Powered by 11th Gen Intel Evo Core i5 Processor ', Processor, ignore.case = TRUE) ~ '6',  
 grepl('Processor: 10th Generation Intel Core i3-1005G1 Processor', Processor, ignore.case = TRUE) ~ '4',  
 grepl('Processor: AMD Ryzen 7-5825U', Processor, ignore.case = TRUE) ~ '8',  
 grepl('Processor: AMD Ryzen R5-5600H', Processor, ignore.case = TRUE) ~ '6',  
 grepl('Processor: Intel i7-11800H', Processor, ignore.case = TRUE) ~ '8',  
 grepl('Processor: Intel PQC-N5030', Processor, ignore.case = TRUE) ~ '4',  
 grepl('Processor: R3-5425U', Processor, ignore.case = TRUE) ~ '4',  
 grepl('Processor: R5-5625U', Processor, ignore.case = TRUE) ~ '6',  
 TRUE ~ 'Unknown'  
 ))  
  
# Remove the character columns  
laptops <- laptops[,!(names(laptops) %in% c("X", "ImgURL", "Memory", "OpSys", "Processor"))]  
  
# convert company to a factor  
laptops$Company <- as.factor(laptops$Company)  
  
# convert processor manufacturer to a factor  
laptops$ProcessorManufacturer <- as.factor(laptops$ProcessorManufacturer)  
  
# convert ram to a factor  
laptops$RAM <- ordered(laptops$RAM, levels=c("4", "8", "16","32"))  
  
# convert memory to an ordered factor  
laptops$DiskGB <- ordered(laptops$DiskGB, levels=c("64", "128", "256", "512", "1000", "2000", "Unknown"))  
  
# convert memory to an ordered factor  
laptops$Cores <- ordered(laptops$Cores, levels=c("2", "4", "6", "8", "10", "12", "14", "Unknown"))  
  
# convert OS to a factor  
laptops$OS <- as.factor(laptops$OS)  
  
# convert Rupees to USD  
laptops$Price = unlist(lapply(laptops$Price \* 0.012, function(x) if(is.numeric(x)) round(x, 2) else x))  
laptops$MRP = unlist(lapply(laptops$MRP \* 0.012, function(x) if(is.numeric(x)) round(x, 2) else x))  
  
# Cost Category: Discretization of Price (an arbitrary list)  
laptops$CostCategory = cut(laptops$Price, breaks=c(0,500,1000,2000,3000,4000,Inf),labels=c("Lightweight","Consumer","Commercial", "Gaming", "Mining","Scientific"))  
laptops = laptops %>% relocate(CostCategory)  
   
# Difference between suggested retail price and retail price (negative values are when retail price is higher than suggested)  
laptops$CostSavings = laptops$MRP - laptops$Price  
  
# Deal Quality: compare average component ratio to price ratio (capped at 2000 because anything higher than that is a fair deal or worse)  
ram\_ratio = (as.numeric(laptops$RAM) / 4)  
disk\_ratio = (as.numeric(laptops$DiskGB) / 6)  
cores\_ratio = (as.numeric(laptops$RAM) / 7)  
component\_ratio = (ram\_ratio + disk\_ratio + cores\_ratio) / 3  
price\_ratio = laptops$Price / 2000  
laptops$DealQuality = mapply(function(x,y) if(x > y) return(1) else return(0), component\_ratio, price\_ratio)

### Numerical Laptop Dataset

# laptop subset with factors converted to numeric  
laptops\_num = laptops[,c("CostCategory", "Company", "Rating", "Size", "RAM", "Price", "ProcessorManufacturer", "DiskGB", "OS", "Cores", "CostSavings")]  
  
laptops\_num$RAM = as.numeric(laptops\_num$RAM)  
laptops\_num$DiskGB = as.numeric(laptops\_num$DiskGB)  
laptops\_num$Cores = as.numeric(laptops\_num$Cores)  
laptops\_num$Company = as.numeric(laptops\_num$Company)  
laptops\_num$ProcessorManufacturer = as.numeric(laptops\_num$ProcessorManufacturer)  
laptops\_num$OS = as.numeric(laptops\_num$OS)  
  
# remove unknown values  
laptops\_num = laptops\_num[laptops\_num$Cores != 8 & laptops\_num$DiskGB != 7,]

### Normalized Laptop Dataset

# normalize ranges to put the variables on equal footing  
laptops\_norm = data.frame(CostCategory = laptops\_num$CostCategory)  
laptops\_norm$Company = normalize(laptops\_num, "Company")  
laptops\_norm$Rating = normalize(laptops\_num, "Rating")  
laptops\_norm$Size = normalize(laptops\_num, "Size")  
laptops\_norm$RAM = normalize(laptops\_num, "RAM")  
laptops\_norm$Price = normalize(laptops\_num, "Price")  
laptops\_norm$ProcessorManufacturer = normalize(laptops\_num, "ProcessorManufacturer")  
laptops\_norm$DiskGB = normalize(laptops\_num, "DiskGB")  
laptops\_norm$OS = normalize(laptops\_num, "OS")  
laptops\_norm$Cores = normalize(laptops\_num, "Cores")  
laptops\_norm$CostSavings = normalize(laptops\_num, "CostSavings")  
laptops\_norm$CostCategory = laptops\_num$CostCategory

### Structure and Summary

print("Original")

## [1] "Original"

str(laptops)

## 'data.frame': 984 obs. of 15 variables:  
## $ CostCategory : Factor w/ 6 levels "Lightweight",..: 2 1 2 1 1 1 1 2 1 1 ...  
## $ Company : Factor w/ 19 levels "acer","ALIENWARE",..: 8 10 4 6 4 17 8 10 8 4 ...  
## $ Rating : num 4.2 4.2 4.4 4.1 4.2 4.1 4.6 4.3 4.2 4.1 ...  
## $ No\_of\_ratings : int 253 974 5389 332 487 4541 9 2841 491 1022 ...  
## $ Review : int 20 75 535 26 53 523 1 241 50 123 ...  
## $ Size : num 36 36 40 40 40 40 40 40 40 36 ...  
## $ RAM : Ord.factor w/ 4 levels "4"<"8"<"16"<"32": 3 2 2 2 2 2 2 2 2 1 ...  
## $ Price : num 600 408 600 456 432 ...  
## $ MRP : num 711 731 852 702 612 ...  
## $ ProcessorManufacturer: Factor w/ 5 levels "AMD","Intel",..: 1 2 2 2 2 2 2 2 2 2 ...  
## $ DiskGB : Ord.factor w/ 7 levels "64"<"128"<"256"<..: 4 3 4 7 4 4 4 4 4 3 ...  
## $ OS : Factor w/ 7 levels "Chrome OS","DOS",..: 7 7 7 6 7 7 7 7 7 7 ...  
## $ Cores : Ord.factor w/ 8 levels "2"<"4"<"6"<"8"<..: 3 2 3 1 2 3 2 3 2 1 ...  
## $ CostSavings : num 111 323 252 246 180 ...  
## $ DealQuality : num 1 1 1 1 1 1 1 1 1 1 ...

print("Numerical")

## [1] "Numerical"

str(laptops\_num)

## 'data.frame': 944 obs. of 11 variables:  
## $ CostCategory : Factor w/ 6 levels "Lightweight",..: 2 1 2 1 1 1 2 1 1 1 ...  
## $ Company : num 8 10 4 4 17 8 10 8 4 8 ...  
## $ Rating : num 4.2 4.2 4.4 4.2 4.1 4.6 4.3 4.2 4.1 4.2 ...  
## $ Size : num 36 36 40 40 40 40 40 40 36 36 ...  
## $ RAM : num 3 2 2 2 2 2 2 2 1 2 ...  
## $ Price : num 600 408 600 432 468 ...  
## $ ProcessorManufacturer: num 1 2 2 2 2 2 2 2 2 2 ...  
## $ DiskGB : num 4 3 4 4 4 4 4 4 3 3 ...  
## $ OS : num 7 7 7 7 7 7 7 7 7 7 ...  
## $ Cores : num 3 2 3 2 3 2 3 2 1 2 ...  
## $ CostSavings : num 111 323 252 180 252 ...

print("Normalized")

## [1] "Normalized"

str(laptops\_norm)

## 'data.frame': 944 obs. of 11 variables:  
## $ CostCategory : Factor w/ 6 levels "Lightweight",..: 2 1 2 1 1 1 2 1 1 1 ...  
## $ Company : num 0.389 0.5 0.167 0.167 0.889 ...  
## $ Rating : num 0.758 0.758 0.818 0.758 0.727 ...  
## $ Size : num 0.412 0.412 0.647 0.647 0.647 ...  
## $ RAM : num 0.667 0.333 0.333 0.333 0.333 ...  
## $ Price : num 0.085 0.0431 0.085 0.0484 0.0562 ...  
## $ ProcessorManufacturer: num 0 0.25 0.25 0.25 0.25 0.25 0.25 0.25 0.25 0.25 ...  
## $ DiskGB : num 0.6 0.4 0.6 0.6 0.6 0.6 0.6 0.6 0.4 0.4 ...  
## $ OS : num 1 1 1 1 1 1 1 1 1 1 ...  
## $ Cores : num 0.4 0.2 0.4 0.2 0.4 0.2 0.4 0.2 0 0.2 ...  
## $ CostSavings : num 0.735 0.781 0.766 0.75 0.766 ...

## Exploration and Visualization

### Correlation Pairs

To identify which variables might be good predictors, the correlation pairs was generated to quickly surface good candidates.

# filter out the unknown categories from core count and disk size  
pair\_columns = c("Rating", "Size", "RAM", "Price", "DiskGB", "Cores", "CostSavings")  
  
pair\_plot = ggpairs(laptops\_norm, columns=pair\_columns, aes(color=CostCategory, alpha = 0.5), upper=list(continuous = wrap("cor", size = 1.5)), labs(title="Figure 1"))  
ggsave(file="pair-plot.jpg")

## Saving 5 x 4 in image

print(pair\_plot)

## Warning in cor(x, y): the standard deviation is zero  
  
## Warning in cor(x, y): the standard deviation is zero  
  
## Warning in cor(x, y): the standard deviation is zero  
  
## Warning in cor(x, y): the standard deviation is zero  
  
## Warning in cor(x, y): the standard deviation is zero  
  
## Warning in cor(x, y): the standard deviation is zero  
  
## Warning in cor(x, y): the standard deviation is zero  
  
## Warning in cor(x, y): the standard deviation is zero  
  
## Warning in cor(x, y): the standard deviation is zero  
  
## Warning in cor(x, y): the standard deviation is zero  
  
## Warning in cor(x, y): the standard deviation is zero

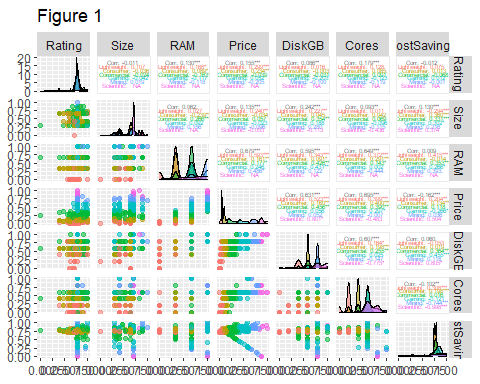


Figure 1 shows some interesting matches for price. The RAM, DiskGB, Cores as well as Size and Rating to a lesser degree and a small subset of products within cost savings which are anomalous prices compared to Manufacturer Recommended Prices (MRP).

### Company

The Company values span 18 product brands.

# display the count of laptops per company  
sort(summary(laptops$Company), decreasing = TRUE)

## Lenovo ASUS HP DELL acer MSI APPLE Infinix   
## 284 206 161 129 57 54 35 28   
## SAMSUNG ALIENWARE realme GIGABYTE Vaio Avita LG Nokia   
## 6 4 4 3 3 2 2 2   
## RedmiBook Mi Primebook   
## 2 1 1

The values converted to proportions for each brand

# Get percentages for each category  
sort(summary(laptops$Company)/sum(summary(laptops$Company)), decreasing = TRUE)

## Lenovo ASUS HP DELL acer MSI   
## 0.288617886 0.209349593 0.163617886 0.131097561 0.057926829 0.054878049   
## APPLE Infinix SAMSUNG ALIENWARE realme GIGABYTE   
## 0.035569106 0.028455285 0.006097561 0.004065041 0.004065041 0.003048780   
## Vaio Avita LG Nokia RedmiBook Mi   
## 0.003048780 0.002032520 0.002032520 0.002032520 0.002032520 0.001016260   
## Primebook   
## 0.001016260

# Create a barplot of Company  
laptops%>% ggplot(aes(x=fct\_infreq(Company))) + geom\_bar(fill=syracuseOrange) +   
 ggtitle("Figure 2 - Laptops per Company") + xlab("Company") + ylab("Count") + theme(plot.title = element\_text(hjust=0.5)) +  
 theme(axis.text.x = element\_text(angle = 45, vjust = 1, hjust=1))

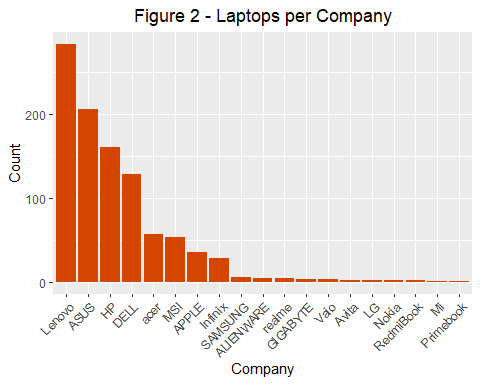


Figure 2 shows the distribution of Company brands within the dataset. Lenovo has the most along with ASUS, HP and DELL. A second tier including Acer, MSI, APPLE and Infinix make up lower but substantial amount. The rest are a long tail of small samples.

laptop\_means = laptops %>% group\_by(Company) %>% summarize(AvgPrice = mean(Price))  
  
ggplot(laptop\_means) +   
 aes(x=reorder(Company, -AvgPrice), y=AvgPrice) +   
 geom\_bar(position="dodge",stat="identity", fill=syracuseOrange) +   
 ggtitle("Figure 3 - Average Laptop Price per Company") +   
 labs(x="Company", y="Price") +   
 theme(plot.title = element\_text(hjust=0.5), axis.text.x = element\_text(angle = 45, vjust = 1, hjust=1))

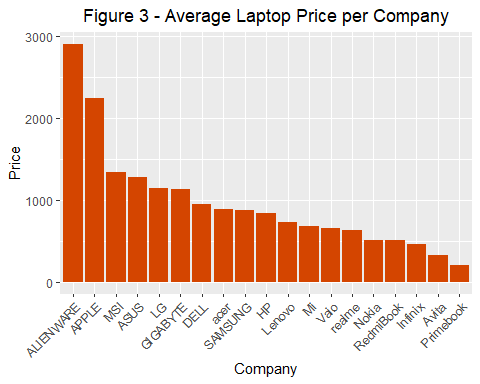


Figure 3 shows the average price per Company. The brand with the highest average price is ALIENWARE and APPLE with a quick drop followed by a long tail of linearly decreasing values for the remaining brands. Although Lenovo, ASUS, HP and DELL have the highest number of laptops they make substantially less per laptop than ALIENWARE and APPLE.

### Rating

# Get a summary of rating  
summary(laptops$Rating)

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 1.700 4.200 4.254 4.254 4.300 5.000

# Plot the distribution of ratings  
ggplot(laptops) +   
 aes(x=Rating) +   
 geom\_histogram(fill=syracuseOrange, color="#eeeeee", bins=50) +  
 ggtitle("Figure 4 - Ratings Distribution") +   
 labs(x="Rating", y="Count") +   
 theme(plot.title = element\_text(hjust=0.5), axis.text.x = element\_text(angle = 45, vjust = 1, hjust=1))

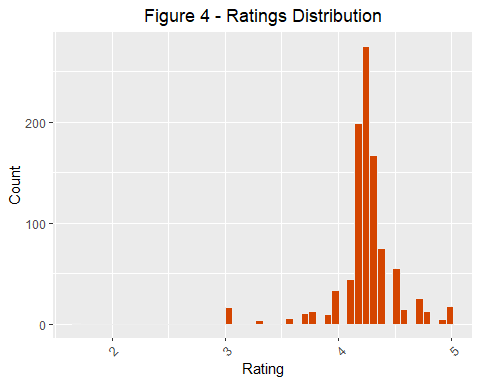


Figure 4 shows that most of the ratings are between 4 and 5. It might indicate the business is carrying products they are confident they can sell.

# Find outliers that are 3 standard deviations away from the mean  
# get mean and Standard deviation  
mean = mean(laptops$Rating)  
std = sd(laptops$Rating)  
  
# get threshold values for outliers  
Tmin = mean-(3\*std)  
Tmax = mean+(3\*std)  
  
# find the outliers  
low\_ratings <- laptops[which(laptops$Rating < Tmin | laptops$Rating > Tmax),]  
low\_ratings[which.min(low\_ratings$Rating),]

## CostCategory Company Rating No\_of\_ratings Review Size RAM Price MRP  
## 854 Commercial DELL 1.7 3 0 34 16 1379.88 1669.25  
## ProcessorManufacturer DiskGB OS Cores CostSavings DealQuality  
## 854 Intel 512 Windows 6 289.37 0

low\_ratings

## CostCategory Company Rating No\_of\_ratings Review Size RAM Price MRP  
## 68 Consumer Lenovo 3.0 4 0 40 8 642.00 972.00  
## 168 Consumer Lenovo 3.0 4 0 40 8 642.00 972.00  
## 193 Consumer DELL 3.3 3 0 40 8 513.48 730.04  
## 288 Consumer Lenovo 3.0 4 0 40 8 642.00 972.00  
## 360 Consumer Lenovo 3.0 4 0 40 8 642.00 972.00  
## 438 Lightweight Lenovo 3.0 4 0 40 8 466.68 704.16  
## 456 Consumer Lenovo 3.0 4 0 40 8 642.00 972.00  
## 480 Consumer Lenovo 3.0 4 0 40 8 642.00 972.00  
## 507 Consumer DELL 3.3 3 0 40 8 519.48 645.60  
## 528 Consumer Lenovo 3.0 4 0 40 8 642.00 972.00  
## 531 Commercial ASUS 3.0 2 1 40 8 1655.88 1757.88  
## 552 Consumer Lenovo 3.0 4 0 40 8 642.00 972.00  
## 648 Consumer Lenovo 3.0 4 0 40 8 642.00 972.00  
## 672 Consumer Lenovo 3.0 4 0 40 8 642.00 972.00  
## 696 Consumer Lenovo 3.0 4 0 40 8 642.00 972.00  
## 816 Consumer Lenovo 3.0 4 0 40 8 642.00 972.00  
## 854 Commercial DELL 1.7 3 0 34 16 1379.88 1669.25  
## 858 Commercial HP 3.3 3 1 34 16 1967.88 2315.72  
## 862 Commercial acer 3.0 2 1 40 16 1727.88 2159.99  
## 881 Consumer HP 3.3 4 0 36 8 587.88 779.36  
## 888 Consumer Lenovo 3.0 4 0 40 8 642.00 972.00  
## ProcessorManufacturer DiskGB OS Cores CostSavings DealQuality  
## 68 Intel 512 Windows 6 330.00 1  
## 168 Intel 512 Windows 6 330.00 1  
## 193 Intel 512 Windows 4 216.56 1  
## 288 Intel 512 Windows 6 330.00 1  
## 360 Intel 512 Windows 6 330.00 1  
## 438 Intel 256 Windows 4 237.48 1  
## 456 Intel 512 Windows 6 330.00 1  
## 480 Intel 512 Windows 6 330.00 1  
## 507 Intel 512 Windows 4 126.12 1  
## 528 Intel 512 Windows 6 330.00 1  
## 531 Unknown Unknown Windows Unknown 102.00 0  
## 552 Intel 512 Windows 6 330.00 1  
## 648 Intel 512 Windows 6 330.00 1  
## 672 Intel 512 Windows 6 330.00 1  
## 696 Intel 512 Windows 6 330.00 1  
## 816 Intel 512 Windows 6 330.00 1  
## 854 Intel 512 Windows 6 289.37 0  
## 858 Intel 512 Windows 8 347.84 0  
## 862 Intel 1000 Windows 8 432.11 0  
## 881 Intel 512 Windows 6 191.48 1  
## 888 Intel 512 Windows 6 330.00 1

# Plot the distribution of ratings  
ggplot(low\_ratings) +   
 aes(x=Rating) +   
 geom\_histogram(fill=syracuseOrange, color="#eeeeee", bins=10) +  
 ggtitle("Figure 5 - Low Ratings Distribution") +   
 labs(x="Rating", y="Count") +   
 theme(plot.title = element\_text(hjust=0.5), axis.text.x = element\_text(angle = 45, vjust = 1, hjust=1))

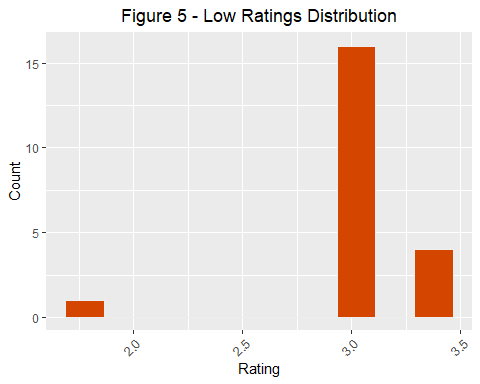


Figure 5, and the tabular output show the low rating distribution and composition. What’s interesting is that most have a price well below the MRP. They also have very few number of ratings but what’s most unexpected is that most are indicated as a good deal and commercial grade. Perhaps a poor build or other component dragged down a model with higher expected potential.

ggplot(laptops) +   
 aes(x=Rating,y=Price) +   
 geom\_point(color=syracuseOrange) +  
 labs(x="Rating", y="Price", title="Figure 6 - Price by Rating") +   
 theme(plot.title = element\_text(hjust=0.5), axis.text.x = element\_text(angle = 45, vjust = 1, hjust=1))

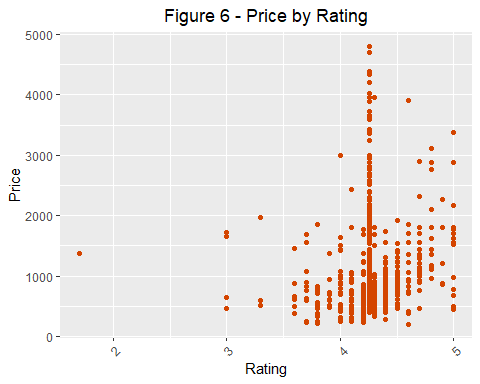


Figure 6, unsurprisingly shows an increase in Rating corresponds to an increase in Price.

### No\_of\_ratings

# Get a summary of number of ratings  
summary(laptops$No\_of\_ratings)

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 1.00 53.75 495.00 1372.65 2841.00 13376.00

# Plot the distribution of number of ratings  
ggplot(laptops) +   
 aes(x=No\_of\_ratings) +   
 geom\_histogram(fill=syracuseOrange, color="#eeeeee", bins=20) +  
 ggtitle("Figure 7 - Number of Ratings Distribution") +   
 labs(x="Rating Count per Product", y="Count") +   
 theme(plot.title = element\_text(hjust=0.5), axis.text.x = element\_text(angle = 45, vjust = 1, hjust=1))

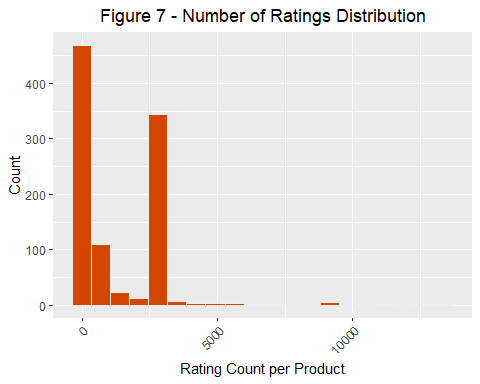


Figure 7 shows the distribution of Rating Count per Product. Most laptops have low rating counts but there’s an unusual spike around 2500. This could be worth further investigation. There’s also a long tail of review counts up almost through 1500.

# Find outliers that are 3 standard deviations away from the mean  
# get mean and Standard deviation  
mean = mean(laptops$No\_of\_ratings)  
std = sd(laptops$No\_of\_ratings)  
  
# get threshold values for outliers  
Tmin = mean-(3\*std)  
Tmax = mean+(3\*std)  
  
# find the outliers  
high\_no\_ratings <- laptops[which(laptops$No\_of\_ratings < Tmin | laptops$No\_of\_ratings > Tmax),]  
high\_no\_ratings[which.max(high\_no\_ratings$No\_of\_ratings),]

## CostCategory Company Rating No\_of\_ratings Review Size RAM Price MRP  
## 302 Consumer acer 4.4 13376 1431 40 8 635.88 1079.99  
## ProcessorManufacturer DiskGB OS Cores CostSavings DealQuality  
## 302 Unknown Unknown Windows Unknown 444.11 1

high\_no\_ratings

## CostCategory Company Rating No\_of\_ratings Review Size RAM Price MRP  
## 19 Lightweight ASUS 4.3 11170 1101 40 8 419.88 551.88  
## 47 Commercial APPLE 4.7 9278 829 34 8 1114.80 922.68  
## 111 Commercial APPLE 4.7 9278 829 34 8 1199.88 1330.80  
## 263 Consumer realme 4.4 7358 982 40 8 665.88 839.99  
## 277 Commercial APPLE 4.7 9278 829 34 8 1139.88 1198.80  
## 302 Consumer acer 4.4 13376 1431 40 8 635.88 1079.99  
## 358 Commercial APPLE 4.7 9278 829 34 8 1319.88 1330.80  
## 543 Consumer realme 4.4 7358 982 40 8 719.88 839.99  
## 595 Commercial APPLE 4.7 9278 829 34 8 1139.88 1198.80  
## 647 Consumer realme 4.4 12879 1904 36 8 563.88 659.99  
## 719 Consumer realme 4.4 12879 1904 40 8 563.88 659.99  
## ProcessorManufacturer DiskGB OS Cores CostSavings DealQuality  
## 19 Intel 512 Windows 4 132.00 1  
## 47 Unknown 256 Mac OS 8 -192.12 0  
## 111 Unknown 512 Mac OS 8 130.92 0  
## 263 Intel Unknown Unknown 6 174.11 1  
## 277 Unknown 256 Mac OS 8 58.92 0  
## 302 Unknown Unknown Windows Unknown 444.11 1  
## 358 Unknown 512 Mac OS 8 10.92 0  
## 543 Intel Unknown Unknown 6 120.11 1  
## 595 Unknown 256 Mac OS 8 58.92 0  
## 647 Intel 256 Windows 4 96.11 1  
## 719 Unknown Unknown Unknown Unknown 96.11 1

# Plot the distribution of ratings  
ggplot(high\_no\_ratings) +   
 aes(x=No\_of\_ratings) +   
 geom\_histogram(fill=syracuseOrange, color="#eeeeee", bins=15) +  
 ggtitle("Figure 8 - High Number of Ratings Distribution") +   
 labs(x="Rating Count per Product", y="Count") +   
 theme(plot.title = element\_text(hjust=0.5), axis.text.x = element\_text(angle = 45, vjust = 1, hjust=1))

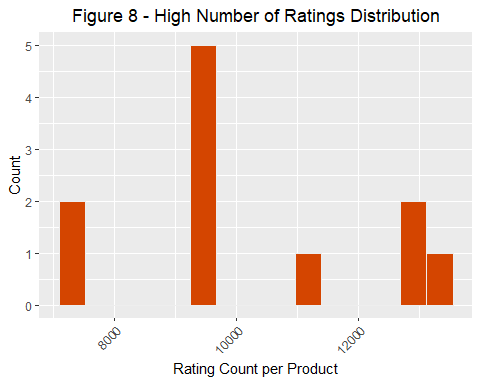


Figure 8 shows the distribution of high number of rating counts.

### Review

# Get a summary of number of reviews  
summary(laptops$Review)

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 0.0 6.0 66.5 127.7 241.0 1904.0

# Plot the distribution of number of reviews  
ggplot(laptops) +   
 aes(x=Review) +   
 geom\_histogram(fill=syracuseOrange, color="#eeeeee", bins=20) +  
 ggtitle("Figure 9 - Number of Reviews Distribution") +   
 labs(x="Review per Product", y="Count") +   
 theme(plot.title = element\_text(hjust=0.5), axis.text.x = element\_text(angle = 45, vjust = 1, hjust=1))

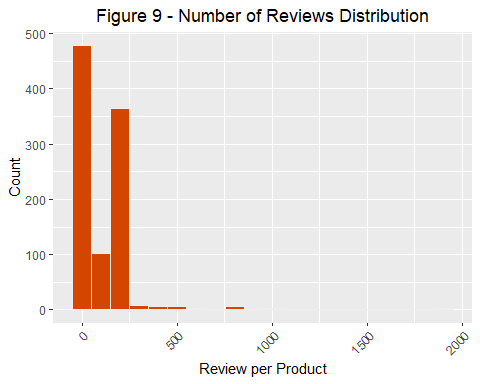


Figure 9 shows the distribution of Numbers of Review per Product. There’s mostly less than 250 Number of Reviews per product.

# Find outliers that are 3 standard deviations away from the mean  
# get mean and Standard deviation  
mean = mean(laptops$Review)  
std = sd(laptops$Review)  
  
# get threshold values for outliers  
Tmin = mean-(3\*std)  
Tmax = mean+(3\*std)  
  
# find the outliers  
high\_no\_reviews <- laptops[which(laptops$Review < Tmin | laptops$Review > Tmax),]  
high\_no\_reviews[which.max(high\_no\_reviews$Review),]

## CostCategory Company Rating No\_of\_ratings Review Size RAM Price MRP  
## 647 Consumer realme 4.4 12879 1904 36 8 563.88 659.99  
## ProcessorManufacturer DiskGB OS Cores CostSavings DealQuality  
## 647 Intel 256 Windows 4 96.11 1

high\_no\_reviews

## CostCategory Company Rating No\_of\_ratings Review Size RAM Price MRP  
## 19 Lightweight ASUS 4.3 11170 1101 40 8 419.88 551.88  
## 47 Commercial APPLE 4.7 9278 829 34 8 1114.80 922.68  
## 111 Commercial APPLE 4.7 9278 829 34 8 1199.88 1330.80  
## 263 Consumer realme 4.4 7358 982 40 8 665.88 839.99  
## 277 Commercial APPLE 4.7 9278 829 34 8 1139.88 1198.80  
## 302 Consumer acer 4.4 13376 1431 40 8 635.88 1079.99  
## 358 Commercial APPLE 4.7 9278 829 34 8 1319.88 1330.80  
## 487 Lightweight ASUS 4.3 5544 790 36 4 377.88 435.25  
## 543 Consumer realme 4.4 7358 982 40 8 719.88 839.99  
## 595 Commercial APPLE 4.7 9278 829 34 8 1139.88 1198.80  
## 647 Consumer realme 4.4 12879 1904 36 8 563.88 659.99  
## 719 Consumer realme 4.4 12879 1904 40 8 563.88 659.99  
## 791 Lightweight HP 3.8 6056 689 40 4 239.88 305.40  
## 911 Lightweight Infinix 4.2 5878 1022 36 8 455.88 599.99  
## ProcessorManufacturer DiskGB OS Cores CostSavings DealQuality  
## 19 Intel 512 Windows 4 132.00 1  
## 47 Unknown 256 Mac OS 8 -192.12 0  
## 111 Unknown 512 Mac OS 8 130.92 0  
## 263 Intel Unknown Unknown 6 174.11 1  
## 277 Unknown 256 Mac OS 8 58.92 0  
## 302 Unknown Unknown Windows Unknown 444.11 1  
## 358 Unknown 512 Mac OS 8 10.92 0  
## 487 Intel 256 Windows 2 57.37 1  
## 543 Intel Unknown Unknown 6 120.11 1  
## 595 Unknown 256 Mac OS 8 58.92 0  
## 647 Intel 256 Windows 4 96.11 1  
## 719 Unknown Unknown Unknown Unknown 96.11 1  
## 791 MediaTek 64 Chrome OS 8 65.52 1  
## 911 Intel 256 Windows 4 144.11 1

# Plot the distribution of reviews  
ggplot(high\_no\_reviews) +   
 aes(x=Review) +   
 geom\_histogram(fill=syracuseOrange, color="#eeeeee", bins=50) +  
 ggtitle("Figure 10 - High Number of Reviews Distribution") +   
 labs(x="Review per Product", y="Count") +   
 theme(plot.title = element\_text(hjust=0.5), axis.text.x = element\_text(angle = 45, vjust = 1, hjust=1))

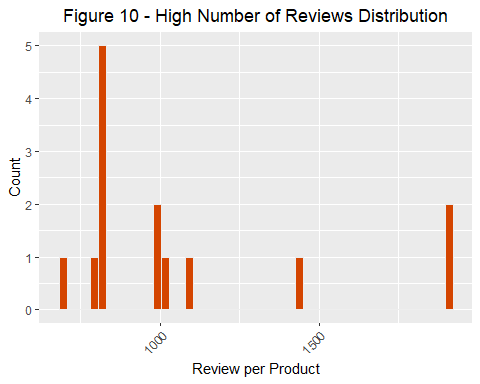


Figure 10 shows the distribution of High Numbers of Review per Product. There’s mostly less than 250 Number of Reviews per product.

### Size

# Get a summary of number of reviews  
summary(laptops$Size)

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 29.00 36.00 40.00 38.66 40.00 46.00

# Plot the distribution of number of reviews  
ggplot(laptops) +   
 aes(x=Size) +   
 geom\_histogram(fill=syracuseOrange, color="#eeeeee", bins=20) +  
 ggtitle("Figure 11 - Screen Size Distribution") +   
 labs(x="Size (in cm)", y="Count") +   
 theme(plot.title = element\_text(hjust=0.5), axis.text.x = element\_text(angle = 45, vjust = 1, hjust=1))

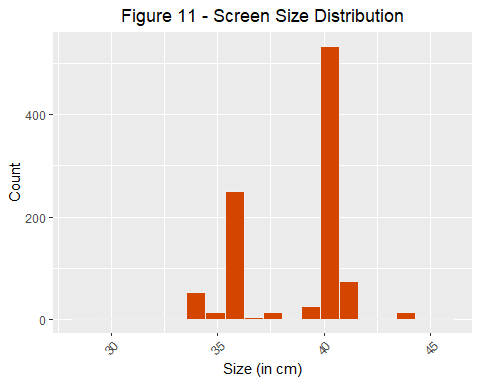


Figure 11 shows the distribution of screen size. There appears to be two ranges of popular sizes; 36 cm or 40 cm.

ggplot(laptops) +   
 aes(x=Size,y=Price) +   
 geom\_point(color=syracuseOrange) +  
 labs(x="Size (in cm)", y="Price", title="Figure 12 - Price by Size") +   
 theme(plot.title = element\_text(hjust=0.5), axis.text.x = element\_text(angle = 45, vjust = 1, hjust=1))

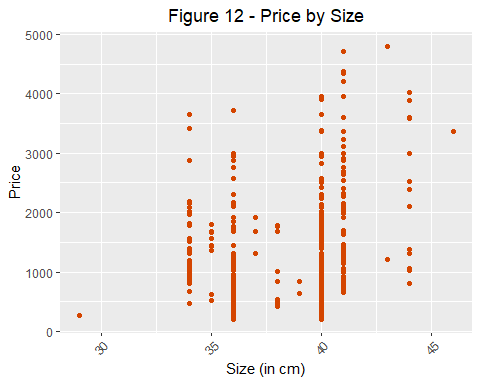


Figure 12 shows the Price by Size. There’s a strong correlation between larger screen sizes and higher prices.

# Find outliers that are 3 standard deviations away from the mean  
# get mean and Standard deviation  
mean = mean(laptops$Size)  
std = sd(laptops$Size)  
  
# get threshold values for outliers  
Tmin = mean-(3\*std)  
Tmax = mean+(3\*std)  
  
# find the outliers  
high\_size <- laptops[which(laptops$Size < Tmin | laptops$Size > Tmax),]  
high\_size[high\_size$Size==max(high\_size$Size),]

## CostCategory Company Rating No\_of\_ratings Review Size RAM Price MRP  
## 105 Mining ASUS 4.253597 2841 241 46 32 3359.88 4031.88  
## ProcessorManufacturer DiskGB OS Cores CostSavings DealQuality  
## 105 Intel 1000 Windows 8 672 0

high\_size

## CostCategory Company Rating No\_of\_ratings Review Size RAM Price MRP  
## 105 Mining ASUS 4.253597 2841 241 46 32 3359.88 4031.88  
## 754 Lightweight ASUS 4.100000 202 18 29 4 269.88 371.88  
## ProcessorManufacturer DiskGB OS Cores CostSavings DealQuality  
## 105 Intel 1000 Windows 8 672 0  
## 754 Intel 128 Windows 2 102 1

# Plot the distribution of high size  
ggplot(high\_size) +   
 aes(x=Size) +   
 geom\_histogram(fill=syracuseOrange, color="#eeeeee", bins=10) +  
 labs(x="Size (in cm)", y="Count", title="Figure 13 - High Monitor Size (Cm) Distribution") +   
 theme(plot.title = element\_text(hjust=0.5), axis.text.x = element\_text(angle = 45, vjust = 1, hjust=1))

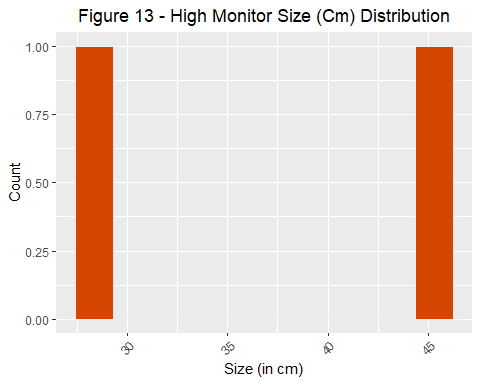


Figure 13 shows the distribution of larger screen sizes.

### Processor Manufacturer

# display the count of laptops per company  
sort(summary(laptops$ProcessorManufacturer), decreasing = TRUE)

## Intel AMD Unknown MediaTek NVIDIA   
## 662 271 47 2 2

# Get percentages for each category  
sort(summary(laptops$ProcessorManufacturer)/sum(summary(laptops$ProcessorManufacturer)), decreasing = TRUE)

## Intel AMD Unknown MediaTek NVIDIA   
## 0.67276423 0.27540650 0.04776423 0.00203252 0.00203252

# Create a barplot of Company  
ggplot(laptops[laptops$ProcessorManufacturer != "Unknown",]) +   
 aes(x=fct\_infreq(ProcessorManufacturer)) +   
 geom\_bar(fill=syracuseOrange, color="#eeeeee") +  
 ggtitle("Figure 14 - Processor Manufacturer Distribution") +   
 labs(x="", y="") +   
 theme(plot.title = element\_text(hjust=0.5), axis.text.x = element\_text(angle = 45, vjust = 1, hjust=1))

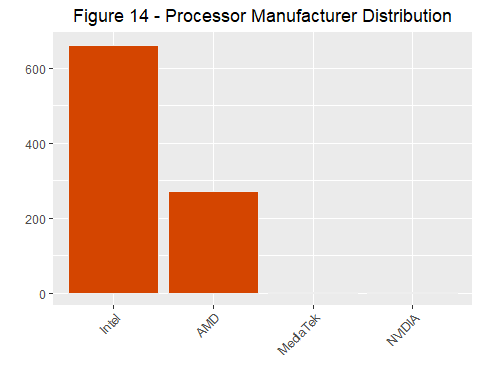


Figure 14 shows the distribution of Processor Manufacturers. Intel makes almost three times as many of the laptop processors than the next competitor AMD. MediaTek and NVIDIA are barely represented.

ggplot(laptops[laptops$ProcessorManufacturer != "Unknown",]) +   
 aes(x=factor(ProcessorManufacturer, levels=c("Intel", "AMD", "MediaTek", "NVIDIA")),y=Price) +   
 geom\_boxplot(outlier.colour=syracuseOrange, outlier.shape=16, outlier.size=2, show.legend=FALSE, color=syracuseOrange) +   
 labs(x="", y="USD", title="Figure 15 - Price by Manufacturer") +   
 theme(plot.title = element\_text(hjust=0.5), axis.text.x = element\_text(angle = 45, vjust = 1, hjust=1))

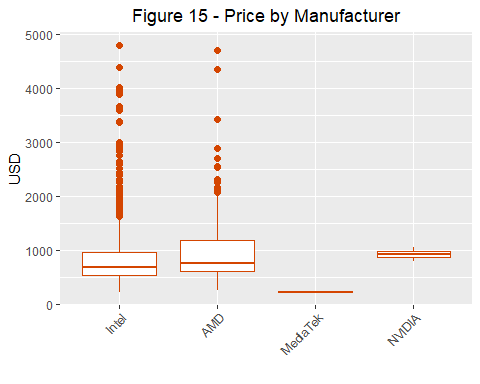


Figure 15 shows the Price distribution per Processor Manufacturer. MediaTek has the lowest Price point of the group. NVIDIA has a similar average price as Intel or AMD but the latter two have many more outliers at higher price bands.

### RAM

# display the count of each RAM type  
sort(summary(laptops$RAM), decreasing = TRUE)

## 8 16 32 4   
## 528 374 45 37

# Get percentages for each RAM type  
sort(summary(laptops$RAM)/sum(summary(laptops$RAM)), decreasing = TRUE)

## 8 16 32 4   
## 0.53658537 0.38008130 0.04573171 0.03760163

# Create a barplot of Laptops per RAM  
ggplot(laptops) +   
 aes(x=factor(RAM, level=c("4", "8", "16", "32"))) +   
 geom\_bar(fill=syracuseOrange, color="#eeeeee") +  
 ggtitle("Figure 16 - RAM Size Distribution") +   
 labs(x="", y="") +   
 theme(plot.title = element\_text(hjust=0.5), axis.text.x = element\_text(angle = 45, vjust = 1, hjust=1))

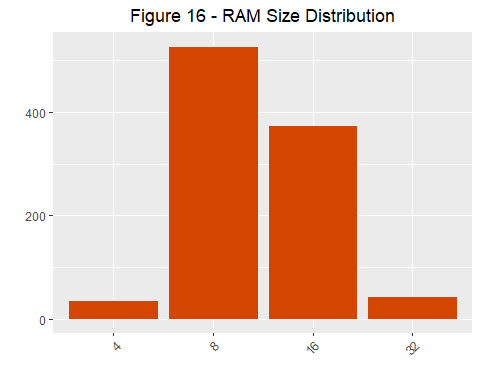


Figure 16 shows the distribution of RAM sizes in Gigabytes (GB). The 8 and 16 GB RAM make up the majority of the laptop RAM sizes. There’s slightly more 32 GB laptops than 4 GB laptops but the general distribution is normal.

ggplot(laptops) +   
 aes(x=RAM,y=Price) +   
 geom\_boxplot(outlier.colour=syracuseOrange, outlier.shape=16, outlier.size=2, show.legend=FALSE, color=syracuseOrange) +  
 labs(x="Gigabytes", y="USD", title="Figure 17 - Price by RAM Size") +   
 theme(plot.title = element\_text(hjust=0.5), axis.text.x = element\_text(angle = 45, vjust = 1, hjust=1))

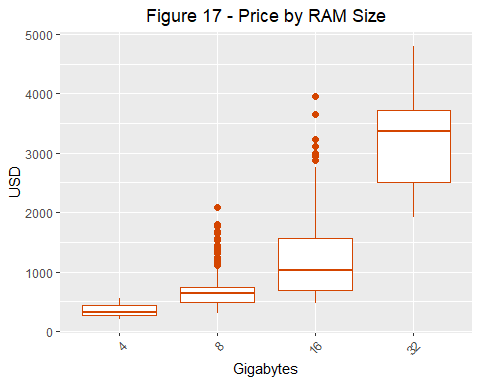


Figure 17 shows the Price distribution by RAM size. There is a strong correlation between higher RAM sizes and higher prices. The relationship appears to be more exponential than linear and could explain why there are much fewer laptops with the max amount of RAM. The cost at the high end is prohibitive.

### Disk Size

# display the count of each Memory type  
sort(summary(laptops$DiskGB), decreasing = TRUE)

## 512 1000 256 Unknown 2000 128 64   
## 607 163 156 36 11 6 5

# Get percentages for each Memory type  
sort(summary(laptops$DiskGB)/sum(summary(laptops$DiskGB)), decreasing = TRUE)

## 512 1000 256 Unknown 2000 128   
## 0.616869919 0.165650407 0.158536585 0.036585366 0.011178862 0.006097561   
## 64   
## 0.005081301

# Create a barplot of Memory  
ggplot(laptops[laptops$DiskGB != "Unknown",]) +   
 aes(x=factor(DiskGB, level=c("64", "128", "256", "512", "1000", "2000"))) +   
 geom\_bar(fill=syracuseOrange, color="#eeeeee") +  
 ggtitle("Figure 18 - Laptop Count by Disk Size") +   
 labs(x="", y="") +   
 theme(plot.title = element\_text(hjust=0.5), axis.text.x = element\_text(angle = 45, vjust = 1, hjust=1))

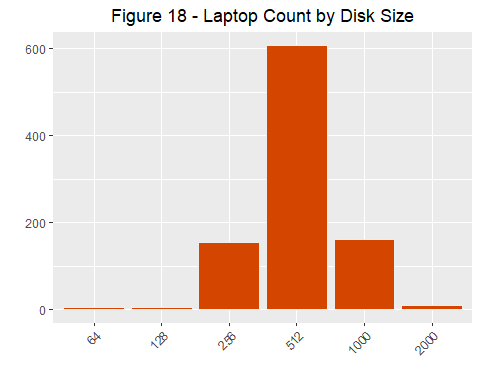


Figure 18 shows the distribution of Disk Size in Gigabytes (GB). The Disk Sizes are a left skewed normal distribution centering on 512 GB disk drives.

ggplot(laptops[laptops$DiskGB != "Unknown",]) +   
 aes(x=factor(DiskGB, level=c("64", "128", "256", "512", "1000", "2000")), y=Price) +   
 geom\_boxplot(outlier.colour=syracuseOrange, outlier.shape=16, outlier.size=2, show.legend=FALSE, color=syracuseOrange) +  
 labs(x="Gigabytes", y="USD", title="Figure 19 - Price by Disk Size") +   
 theme(plot.title = element\_text(hjust=0.5), axis.text.x = element\_text(angle = 45, vjust = 1, hjust=1))

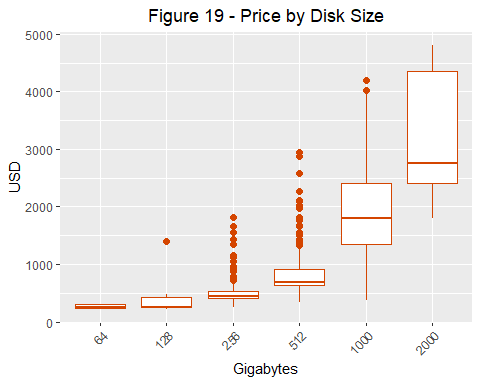


Figure 19 shows the Price distribution for the Disk Size. Similar to the RAM size, the Disk Size appears to have an exponential relationship to the price. This could make the higher disk sizes cost prohibitive and generally for purposes that can bear the cost.

### OS

# display the count of each OS type  
sort(summary(laptops$OS), decreasing = TRUE)

## Windows Mac OS Unknown Chrome OS DOS Linux Prime OS   
## 920 35 19 5 3 1 1

# Get percentages for each OS type  
sort(summary(laptops$OS)/sum(summary(laptops$OS)), decreasing = TRUE)

## Windows Mac OS Unknown Chrome OS DOS Linux   
## 0.934959350 0.035569106 0.019308943 0.005081301 0.003048780 0.001016260   
## Prime OS   
## 0.001016260

# Create a barplot of OS  
ggplot(laptops[laptops$OS != "Unknown",]) +   
 aes(x=fct\_infreq(OS)) +   
 geom\_bar(fill=syracuseOrange, color="#eeeeee") +  
 labs(x="", y="", title="Figure 20 - Laptop Count by OS") +   
 theme(plot.title = element\_text(hjust=0.5), axis.text.x = element\_text(angle = 45, vjust = 1, hjust=1))

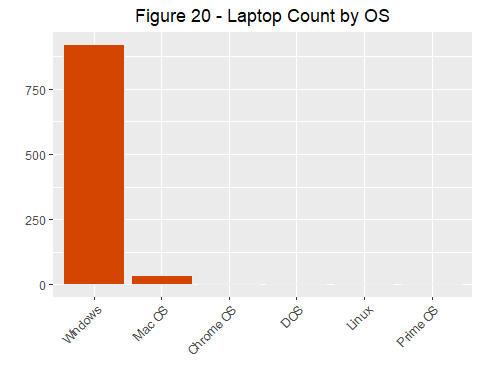


Figure 21 shows the distribution of Operating Systems. Nearly all are Windows with a small amount using Mac OS. There’s a very small distribution between the rest of the OS’s.

ggplot(laptops[laptops$OS != "Unknown",]) +   
 aes(x=factor(OS, levels=c("Windows", "Mac OS", "Chrome OS", "DOS", "Linux", "Prime OS")),y=Price) +   
 geom\_boxplot(outlier.colour=syracuseOrange, outlier.shape=16, outlier.size=2, show.legend=FALSE, color=syracuseOrange) +  
 labs(x="", y="USD", title="Figure 22 - Price by OS") +   
 theme(plot.title = element\_text(hjust=0.5), axis.text.x = element\_text(angle = 45, vjust = 1, hjust=1))

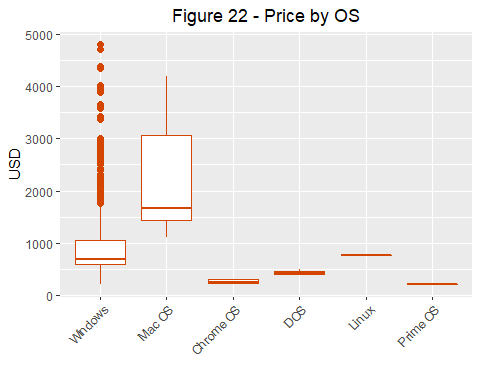


Figure 22 shows the Price distribution per Operating System. Apple commands the highest price premium for it’s brand. Windows has a similar price point unexpectedly with Linux but Windows also has a high number of high priced outliers. The remaining systems share the low end of the market.

### Price

# Get a summary of price  
summary(laptops$Price)

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 203.9 596.9 703.7 985.7 1109.3 4799.9

# Plot the distribution of price  
options(scipen=999)  
  
ggplot(laptops) +   
 aes(x=Price) +   
 geom\_histogram(fill=syracuseOrange, color="#eeeeee", bins=50) +  
 ggtitle("Figure 23 - Laptop Count by Price") +   
 labs(x="", y="") +   
 theme(plot.title = element\_text(hjust=0.5), axis.text.x = element\_text(angle = 45, vjust = 1, hjust=1))

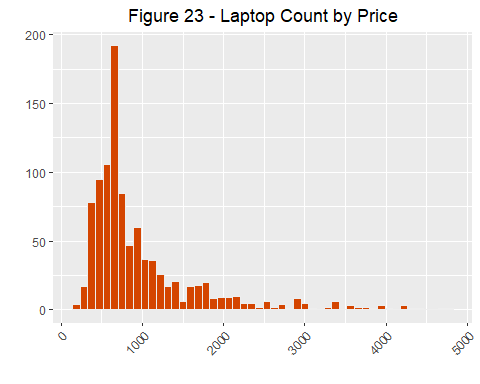


Figure 23 shows the Price distribution. Most prices are below $1,000. There’s a significant number of laptops between $1,000 and $2,000 but few beyond that. The high end does reach almost up to $5,000.

# Find outliers that are 3 standard deviations away from the mean  
# get mean and Standard deviation  
mean = mean(laptops$Price)  
std = sd(laptops$Price)  
  
# get threshold values for outliers  
Tmin = mean-(3\*std)  
Tmax = mean+(3\*std)  
  
# find the outliers  
high\_price <- laptops[which(laptops$Price < Tmin | laptops$Price > Tmax),]  
high\_price[high\_price$Price==max(high\_price$Price),]

## CostCategory Company Rating No\_of\_ratings Review Size RAM Price MRP  
## 185 Scientific MSI 4.253597 2841 241 43 32 4799.88 5375.88  
## ProcessorManufacturer DiskGB OS Cores CostSavings DealQuality  
## 185 Intel 2000 Windows 8 576 0

high\_price

## CostCategory Company Rating No\_of\_ratings Review Size RAM Price  
## 38 Mining APPLE 4.253597 2841 241 36 32 3718.80  
## 73 Mining ASUS 4.253597 2841 241 41 32 3359.88  
## 89 Scientific ASUS 4.253597 2841 241 41 32 4379.88  
## 93 Scientific APPLE 4.253597 2841 241 41 32 4198.80  
## 105 Mining ASUS 4.253597 2841 241 46 32 3359.88  
## 133 Scientific ASUS 4.253597 2841 241 41 32 4343.88  
## 134 Mining ASUS 5.000000 4 0 40 32 3383.88  
## 135 Mining HP 4.253597 2841 241 44 32 3587.88  
## 147 Scientific ASUS 4.253597 2841 241 41 32 4703.88  
## 153 Scientific APPLE 4.253597 2841 241 41 32 4198.80  
## 162 Mining DELL 4.253597 2841 241 44 32 3887.88  
## 163 Mining APPLE 4.253597 2841 241 41 16 3238.80  
## 165 Mining ASUS 4.253597 2841 241 40 32 3383.88  
## 169 Mining APPLE 4.300000 13 5 41 32 3958.80  
## 174 Mining APPLE 4.253597 2841 241 41 16 3238.80  
## 179 Scientific DELL 4.253597 2841 241 44 32 4019.88  
## 185 Scientific MSI 4.253597 2841 241 43 32 4799.88  
## 249 Mining ASUS 4.253597 2841 241 34 32 3419.88  
## 255 Mining DELL 4.253597 2841 241 40 32 3365.88  
## 287 Scientific APPLE 4.253597 2841 241 41 32 4198.80  
## 373 Mining ALIENWARE 4.600000 10 3 40 32 3911.88  
## 399 Mining ASUS 4.253597 2841 241 40 16 3959.88  
## 447 Mining ASUS 4.253597 2841 241 34 16 3659.76  
## 466 Mining APPLE 4.253597 2841 241 36 32 3718.80  
## 489 Mining DELL 4.253597 2841 241 44 32 3605.88  
## 657 Mining ASUS 4.253597 2841 241 41 32 3599.88  
## 713 Mining DELL 4.253597 2841 241 40 32 3659.88  
## MRP ProcessorManufacturer DiskGB OS Cores CostSavings DealQuality  
## 38 922.68 Unknown 1000 Mac OS 10 -2796.12 0  
## 73 3959.88 Intel 1000 Windows 8 600.00 0  
## 89 5183.88 Intel 2000 Windows 8 804.00 0  
## 93 922.68 Unknown 1000 Mac OS 10 -3276.12 0  
## 105 4031.88 Intel 1000 Windows 8 672.00 0  
## 133 5183.88 AMD 2000 Windows 8 840.00 0  
## 134 4319.88 Intel 1000 Windows 8 936.00 0  
## 135 4136.83 Intel 2000 Windows 8 548.95 0  
## 147 5615.88 AMD 2000 Windows 8 912.00 0  
## 153 922.68 Unknown 1000 Mac OS 10 -3276.12 0  
## 162 5217.96 Intel 1000 Windows 8 1330.08 0  
## 163 922.68 Unknown 1000 Mac OS 12 -2316.12 0  
## 165 4355.88 Intel 1000 Windows 8 972.00 0  
## 169 922.68 Unknown 1000 Mac OS 10 -3036.12 0  
## 174 922.68 Unknown 1000 Mac OS 12 -2316.12 0  
## 179 4488.04 Intel 1000 Windows 8 468.16 0  
## 185 5375.88 Intel 2000 Windows 8 576.00 0  
## 249 4103.88 AMD 1000 Windows 8 684.00 0  
## 255 4383.41 Intel 1000 Windows 8 1017.53 0  
## 287 922.68 Unknown 1000 Mac OS 10 -3276.12 0  
## 373 4533.58 Intel 1000 Windows 10 621.70 0  
## 399 4355.88 Intel Unknown Windows 10 396.00 0  
## 447 4607.88 Intel 1000 Windows 8 948.12 0  
## 466 922.68 Unknown 1000 Mac OS 10 -2796.12 0  
## 489 4696.54 Intel 1000 Windows 8 1090.66 0  
## 657 4319.88 Intel 1000 Windows 8 720.00 0  
## 713 4207.54 Intel 1000 Windows 8 547.66 0

# Plot the distribution of high size  
ggplot(high\_price) +   
 aes(x=Price) +   
 geom\_histogram(fill=syracuseOrange, color="#eeeeee", bins=20) +  
 labs(x="", y="", title="Figure 24 - High Price Distribution") +   
 theme(plot.title = element\_text(hjust=0.5), axis.text.x = element\_text(angle = 45, vjust = 1, hjust=1))

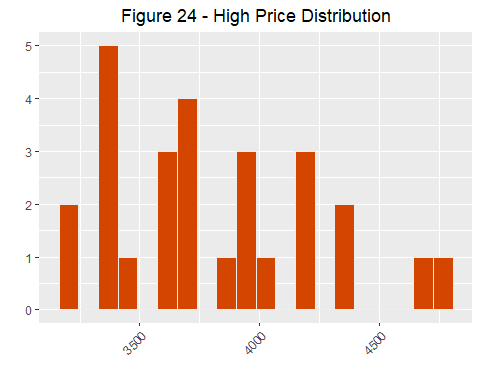


Figure 24 shows the high end of the price distribution.

ggplot(laptops) +  
 aes(x=Company, y=Price) +  
 geom\_jitter(shape=16, position=position\_jitter(0.4), color=syracuseOrange) +  
 labs(x="", y="", title="Figure 25 - Price Distribution by Company") +   
 theme(axis.text.x = element\_text(angle = 45, vjust = 1, hjust = 1))

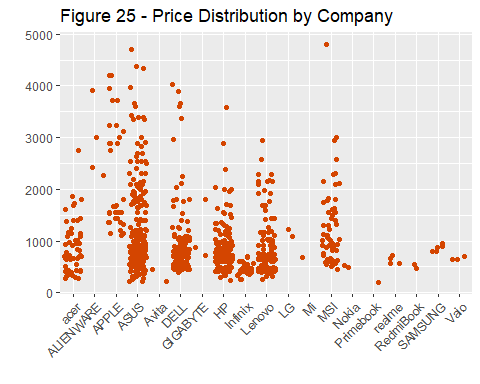


Figure 25 shows the price distribution by Company. Seven of the nineteen companies have products with prices that span a wide range of prices. Most do not have high price bands. Interesting but unsurprisingly, Apple has a much higher floor than other companies.

ggplot(laptops[laptops$OS != "Unknown",]) +  
 aes(x=OS, y=Price) +  
 geom\_jitter(shape=16, position=position\_jitter(0.4), color=syracuseOrange) +  
 labs(x="", y="", title="Figure 26 - Price Distribution by OS") +   
 theme(axis.text.x = element\_text(angle = 45, vjust = 1, hjust = 1))

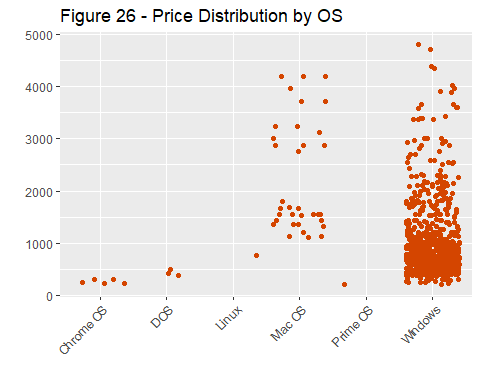


Figure 26 shows the Price distribution by Operating System. Windows and Mac OS hold the highest prices but similar to the previous plot, Mac OS has a higher floor price.

ggplot(laptops[laptops$ProcessorManufacturer != "Unknown",]) +  
 aes(x=ProcessorManufacturer, y=Price) +  
 geom\_jitter(shape=16, position=position\_jitter(0.4), color=syracuseOrange) +  
 labs(x="", y="", title="Figure 27 - Price Distribution by ProcessorManufacturer") +   
 theme(axis.text.x = element\_text(angle = 45, vjust = 1, hjust = 1))

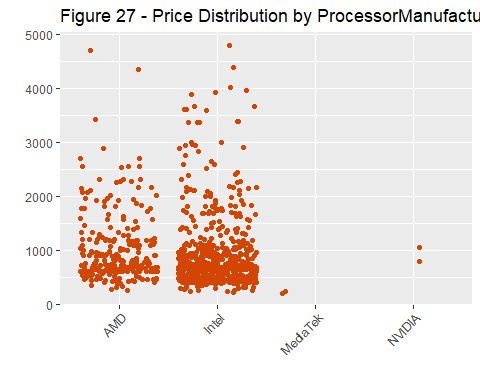


Figure 27 shows the price Distribution by Processor Manufacturer. AMD and Intel have nearly the entire market and similar price bands although AMD has a slightly higher price floor.

### MRP

# Get a summary of mrp  
summary(laptops$MRP)

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 275.9 730.7 974.7 1208.9 1298.1 5615.9

# Plot the distribution of mrp  
options(scipen=999)  
  
ggplot(laptops) +   
 aes(x=MRP) +   
 geom\_histogram(fill=syracuseOrange, color="#eeeeee", bins=50) +  
 ggtitle("Figure 28 - Manufacturer Recommended Price Distribution") +   
 labs(x="USD", y="") +   
 theme(plot.title = element\_text(hjust=0.5), axis.text.x = element\_text(angle = 45, vjust = 1, hjust=1))

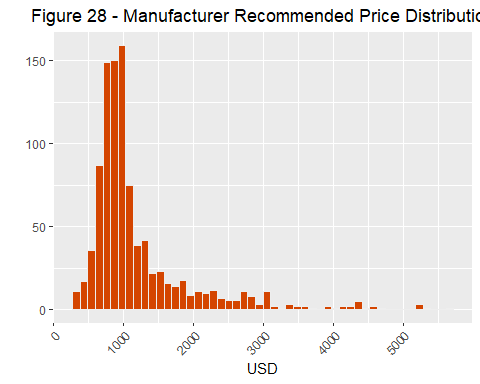


Figure 28 shows the distribution of MRP values. Similar to the Price distribution, most of the values are below $1,000. Unlike the Price distribution the center point is slightly higher. This would indicate many of the laptops have a lower price than recommended by the manufacturer.

ggplot(laptops) +   
 aes(x=MRP,y=Price) +   
 geom\_point(color=syracuseOrange) +  
 labs(x="MRP USD", y="Price USD", title="Figure 29.1 - Price by MRP") +   
 theme(plot.title = element\_text(hjust=0.5), axis.text.x = element\_text(angle = 45, vjust = 1, hjust=1))

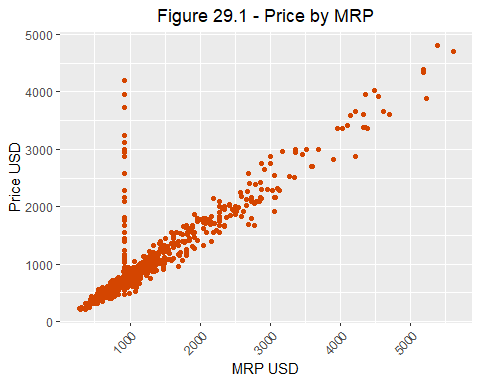


Figure 29.1 shows a strong correlation between MRP and Price. There’s an odd set of outliers whose price is significanly higher than the suggested price.

ggplot(laptops[laptops$MRP < 1000 & laptops$Price > 1000,]) +   
 aes(x=Company) +   
 geom\_bar(fill=syracuseOrange, color="#eeeeee", bins=50) +  
 ggtitle("Figure 29.2 - Manufacturer Recommended Price Distribution") +   
 labs(x="USD", y="") +   
 theme(plot.title = element\_text(hjust=0.5), axis.text.x = element\_text(angle = 45, vjust = 1, hjust=1))

## Warning: Ignoring unknown parameters: bins

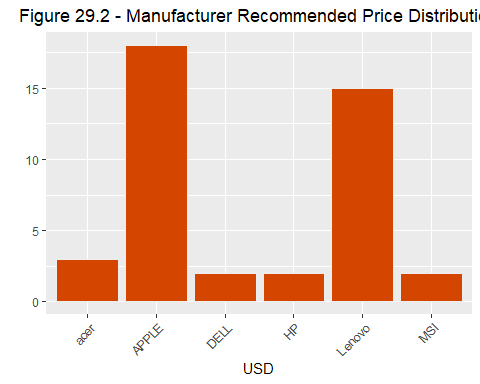


Figure 29.2 shows that the outliers from the previous plot were predominantly Apple or Lenovo laptops.

summary(laptops[laptops$MRP < 1000 & laptops$Price > 1000,c("Company", "RAM", "DiskGB", "Cores")])

## Company RAM DiskGB Cores   
## APPLE :18 4 : 0 64 : 0 8 :25   
## Lenovo :15 8 :11 128 : 1 6 : 6   
## acer : 3 16:24 256 : 6 10 : 6   
## DELL : 2 32: 7 512 :20 12 : 3   
## HP : 2 1000 :13 Unknown: 2   
## MSI : 2 2000 : 1 2 : 0   
## (Other): 0 Unknown: 1 (Other): 0

Further analysis shows that the high prices weren’t justified by large RAM sizes, or high numbers of Cores. Most did have a high Disk Size.

# Find outliers that are 3 standard deviations away from the mean  
# get mean and Standard deviation  
mean = mean(laptops$MRP)  
std = sd(laptops$MRP)  
# get threshold values for outliers  
  
Tmin = mean-(3\*std)  
Tmax = mean+(3\*std)  
# find the outliers  
  
high\_mrp <- laptops[which(laptops$MRP < Tmin | laptops$MRP > Tmax),]  
high\_mrp[high\_mrp$MRP==max(high\_mrp$MRP),]

## CostCategory Company Rating No\_of\_ratings Review Size RAM Price MRP  
## 147 Scientific ASUS 4.253597 2841 241 41 32 4703.88 5615.88  
## ProcessorManufacturer DiskGB OS Cores CostSavings DealQuality  
## 147 AMD 2000 Windows 8 912 0

high\_mrp

## CostCategory Company Rating No\_of\_ratings Review Size RAM Price  
## 73 Mining ASUS 4.253597 2841 241 41 32 3359.88  
## 89 Scientific ASUS 4.253597 2841 241 41 32 4379.88  
## 105 Mining ASUS 4.253597 2841 241 46 32 3359.88  
## 133 Scientific ASUS 4.253597 2841 241 41 32 4343.88  
## 134 Mining ASUS 5.000000 4 0 40 32 3383.88  
## 135 Mining HP 4.253597 2841 241 44 32 3587.88  
## 147 Scientific ASUS 4.253597 2841 241 41 32 4703.88  
## 162 Mining DELL 4.253597 2841 241 44 32 3887.88  
## 165 Mining ASUS 4.253597 2841 241 40 32 3383.88  
## 179 Scientific DELL 4.253597 2841 241 44 32 4019.88  
## 185 Scientific MSI 4.253597 2841 241 43 32 4799.88  
## 249 Mining ASUS 4.253597 2841 241 34 32 3419.88  
## 255 Mining DELL 4.253597 2841 241 40 32 3365.88  
## 373 Mining ALIENWARE 4.600000 10 3 40 32 3911.88  
## 399 Mining ASUS 4.253597 2841 241 40 16 3959.88  
## 405 Gaming ASUS 4.253597 2841 241 40 32 2819.88  
## 419 Gaming ASUS 4.253597 2841 241 41 32 2699.88  
## 441 Gaming ASUS 4.253597 2841 241 41 32 2999.88  
## 447 Mining ASUS 4.253597 2841 241 34 16 3659.76  
## 449 Gaming ASUS 4.253597 2841 241 41 32 2699.88  
## 489 Mining DELL 4.253597 2841 241 44 32 3605.88  
## 657 Mining ASUS 4.253597 2841 241 41 32 3599.88  
## 713 Mining DELL 4.253597 2841 241 40 32 3659.88  
## 897 Gaming ASUS 5.000000 4 0 34 32 2879.88  
## MRP ProcessorManufacturer DiskGB OS Cores CostSavings DealQuality  
## 73 3959.88 Intel 1000 Windows 8 600.00 0  
## 89 5183.88 Intel 2000 Windows 8 804.00 0  
## 105 4031.88 Intel 1000 Windows 8 672.00 0  
## 133 5183.88 AMD 2000 Windows 8 840.00 0  
## 134 4319.88 Intel 1000 Windows 8 936.00 0  
## 135 4136.83 Intel 2000 Windows 8 548.95 0  
## 147 5615.88 AMD 2000 Windows 8 912.00 0  
## 162 5217.96 Intel 1000 Windows 8 1330.08 0  
## 165 4355.88 Intel 1000 Windows 8 972.00 0  
## 179 4488.04 Intel 1000 Windows 8 468.16 0  
## 185 5375.88 Intel 2000 Windows 8 576.00 0  
## 249 4103.88 AMD 1000 Windows 8 684.00 0  
## 255 4383.41 Intel 1000 Windows 8 1017.53 0  
## 373 4533.58 Intel 1000 Windows 10 621.70 0  
## 399 4355.88 Intel Unknown Windows 10 396.00 0  
## 405 3899.88 Intel 1000 Windows 8 1080.00 0  
## 419 3587.88 AMD 2000 Windows 8 888.00 0  
## 441 3683.88 Intel 1000 Windows 8 684.00 0  
## 447 4607.88 Intel 1000 Windows 8 948.12 0  
## 449 3599.88 AMD 2000 Windows 8 900.00 0  
## 489 4696.54 Intel 1000 Windows 8 1090.66 0  
## 657 4319.88 Intel 1000 Windows 8 720.00 0  
## 713 4207.54 Intel 1000 Windows 8 547.66 0  
## 897 4211.88 AMD 1000 Windows 8 1332.00 0

# Plot the distribution of high size  
ggplot(high\_mrp) +   
 aes(x=MRP) +   
 geom\_histogram(fill=syracuseOrange, color="#eeeeee", bins=30) +  
 ggtitle("Figure 30 - High Manufacturer Recommended Price Distribution") +   
 labs(x="USD", y="") +   
 theme(plot.title = element\_text(hjust=0.5), axis.text.x = element\_text(angle = 45, vjust = 1, hjust=1))

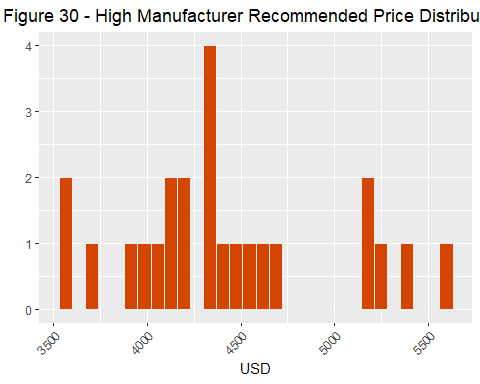


Figure 30 shows the high end of the MRP distribution.

### Cores

# Get a summary of cores  
summary(laptops$Cores)

## 2 4 6 8 10 12 14 Unknown   
## 35 209 442 263 13 5 1 16

# Plot the distribution of cores  
options(scipen=999)  
  
ggplot(laptops[laptops$Cores != "Unknown",]) +   
 aes(x=Cores) +   
 geom\_bar(fill=syracuseOrange, color="#eeeeee", bins=10) +  
 ggtitle("Figure 31 - CPU Cores Distribution") +   
 labs(x="", y="") +   
 theme(plot.title = element\_text(hjust=0.5), axis.text.x = element\_text(angle = 45, vjust = 1, hjust=1))

## Warning: Ignoring unknown parameters: bins

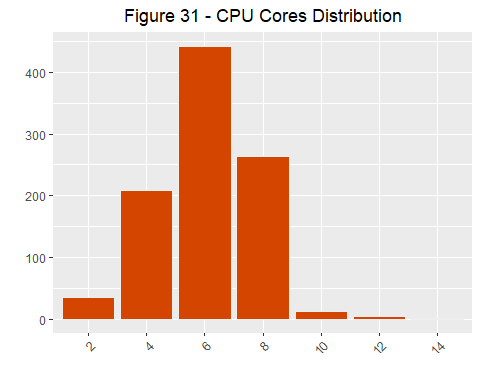
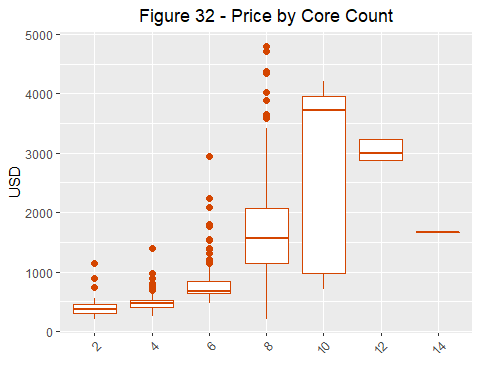


Figure 31 shows a normal distribution of CPU Cores. The center sits at 6 cores. Very few have the highest number of cores.

ggplot(laptops[laptops$Cores != "Unknown",]) +   
 aes(x=factor(Cores),y=Price) +   
 geom\_boxplot(outlier.colour=syracuseOrange, outlier.shape=16, outlier.size=2, show.legend=FALSE, color=syracuseOrange) +  
 labs(x="", y="USD", title="Figure 32 - Price by Core Count") +   
 theme(plot.title = element\_text(hjust=0.5), axis.text.x = element\_text(angle = 45, vjust = 1, hjust=1))



### Deal Quality

# Get a summary of deal quality  
summary(laptops$DealQulity)

## Length Class Mode   
## 0 NULL NULL

# Plot the distribution of deal quality  
options(scipen=999)  
  
laptops\_qual = laptops %>% mutate(QualityCategory = ifelse(laptops$DealQuality == 1, "Good Quality", "Bad Quality"))  
  
ggplot(laptops\_qual) +   
 aes(x=QualityCategory) +   
 geom\_bar(fill=syracuseOrange, color="#eeeeee", bins=10) +  
 ggtitle("Figure 33 - Deal Quality Distribution") +   
 labs(x="", y="") +   
 theme(plot.title = element\_text(hjust=0.5), axis.text.x = element\_text(angle = 45, vjust = 1, hjust=1))

## Warning: Ignoring unknown parameters: bins

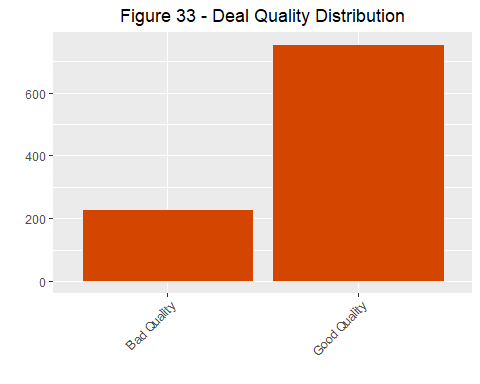


Figure 33 shows that there are almost four times as many good quality deals as bad quality deals.

### CostCategory

# Get a summary of cores  
summary(laptops$CostCategory)

## Lightweight Consumer Commercial Gaming Mining Scientific   
## 182 510 205 59 20 8

# Plot the distribution of cores  
options(scipen=999)  
  
ggplot(laptops) +   
 aes(x=CostCategory) +   
 geom\_bar(fill=syracuseOrange, color="#eeeeee", bins=10) +  
 ggtitle("Figure 34 - Cost Category Distribution") +   
 labs(x="", y="") +   
 theme(plot.title = element\_text(hjust=0.5), axis.text.x = element\_text(angle = 45, vjust = 1, hjust=1))

## Warning: Ignoring unknown parameters: bins

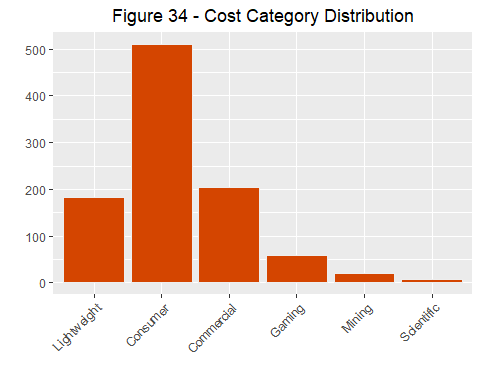


Figure 34 shows a right skewed distribution of Cost Categories. Most laptops are consumer grade with a steadily shrinking tail of higher cost laptops.

ggplot(laptops[laptops$Cores != "Unknown",]) +   
 aes(x=CostCategory, y=Cores) +   
 geom\_point(color=syracuseOrange) +   
 ggtitle("Figure 35 - Cost Category by Core") +   
 labs(x="", y="Cores") +   
 geom\_count(aes(size = after\_stat(prop), group = 1), color=syracuseOrange, show.legend=FALSE) +  
 scale\_size\_area(max\_size = 10) +   
 theme(plot.title = element\_text(hjust=0.5), axis.text.x = element\_text(angle = 45, vjust = 1, hjust=1))

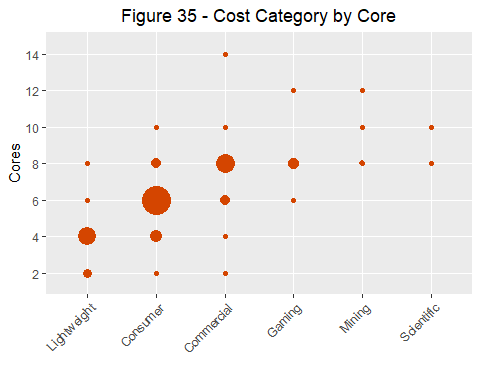


Figure 35 shows the Cost Category by CPU Core Count. There’s a strong correlation between higher cost laptop and more CPU Cores. There is a high density of laptops in the Lightweight category with 4 cores, the consumer category with 6 cores or the commercial category with 8 cores.

### CostSavings

# Get a summary of cores  
summary(laptops$CostSavings)

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## -3276.1 144.0 252.0 223.1 334.8 1332.0

# Plot the distribution of cores  
options(scipen=999)  
  
ggplot(laptops) +   
 aes(x=CostSavings) +   
 geom\_histogram(fill=syracuseOrange, color="#eeeeee", bins=50) +  
 ggtitle("Figure 36.1 - Cost Savings Distribution") +   
 labs(x="", y="") +   
 theme(plot.title = element\_text(hjust=0.5), axis.text.x = element\_text(angle = 45, vjust = 1, hjust=1))

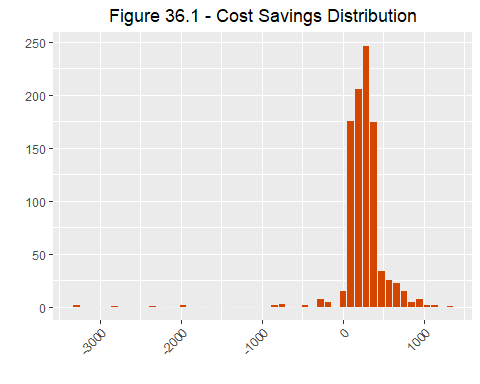
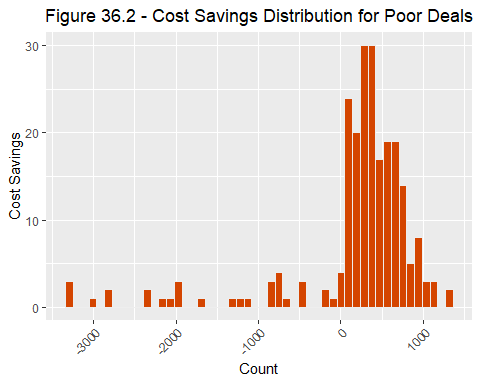
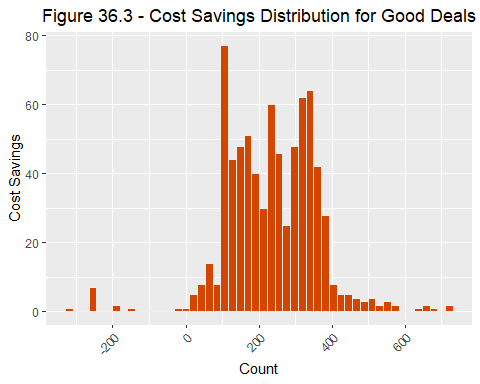


Figure 36.1 shows the distribution of the Cost Savings. The majority of cost savings sits between $0 and $1,000 dollars. This means that most laptops were sold lower than the MRP. There are a small but significant group of laptops that are sold at a cost much higher than than MRP.

# see how cost savings are distributed between good and bad deals  
ggplot(laptops[laptops$DealQuality == 0,]) +   
 aes(x=CostSavings) +   
 geom\_histogram(fill=syracuseOrange, color="#eeeeee", bins=50) +  
 ggtitle("Figure 36.2 - Cost Savings Distribution for Poor Deals") +   
 labs(x="Count", y="Cost Savings") +   
 theme(plot.title = element\_text(hjust=0.5), axis.text.x = element\_text(angle = 45, vjust = 1, hjust=1))



ggplot(laptops[laptops$DealQuality == 1,]) +   
 aes(x=CostSavings) +   
 geom\_histogram(fill=syracuseOrange, color="#eeeeee", bins=50) +  
 ggtitle("Figure 36.3 - Cost Savings Distribution for Good Deals") +   
 labs(x="Count", y="Cost Savings") +   
 theme(plot.title = element\_text(hjust=0.5), axis.text.x = element\_text(angle = 45, vjust = 1, hjust=1))



Figures 36.2 and 36.3 shows further examination on the cost savings differences between good and bad deals. The good deals tend to have a higher cost savings than bad deals which is expected.

ggplot(laptops) +   
 aes(x=Company, y=CostSavings) +   
 geom\_boxplot(outlier.colour=syracuseOrange, outlier.shape=16, outlier.size=2, show.legend=FALSE, color=syracuseOrange) +   
 ggtitle("Figure 37 - Cost Savings by Company") +   
 labs(x="", y="Cost Savings") +   
 theme(plot.title = element\_text(hjust=0.5), axis.text.x = element\_text(angle = 45, vjust = 1, hjust=1))

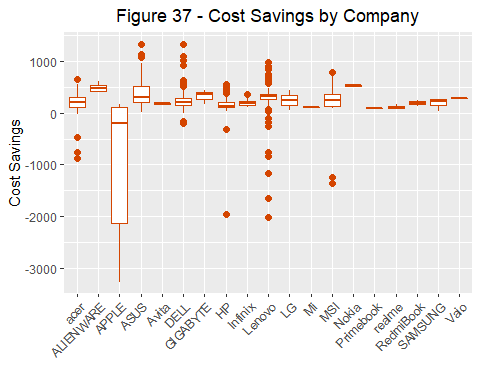


Figure 37 shows the Cost Savings distribution by Company. Confirming earlier results, APPLE and lenovo make up a considerable volume of the laptops costing more than their MRP.

ggplot(laptops) +   
 aes(x=Price, y=CostSavings) +   
 ggtitle("Figure 38 - Cost Savings by Price") +   
 labs(x="Price", y="Cost Savings") +   
 geom\_point(color=syracuseOrange) +  
 theme(plot.title = element\_text(hjust=0.5), axis.text.x = element\_text(angle = 45, vjust = 1, hjust=1))

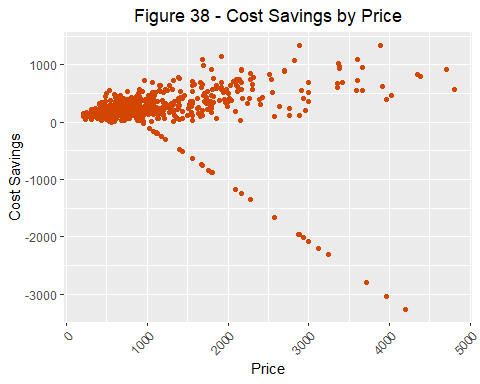


Figure 38 shows little correlation between Price point and Cost Savings with an odd set of outliers whose Cost Savings decreases with Price.

ggplot(laptops[laptops$CostSavings < 500 & laptops$Price > 1000,]) +   
 aes(x=Company) +   
 geom\_bar(fill=syracuseOrange, color="#eeeeee", bins=50) +  
 ggtitle("Figure 39 - Cost Savings Distribution by Company") +   
 labs(x="USD", y="") +   
 theme(plot.title = element\_text(hjust=0.5), axis.text.x = element\_text(angle = 45, vjust = 1, hjust=1))

## Warning: Ignoring unknown parameters: bins

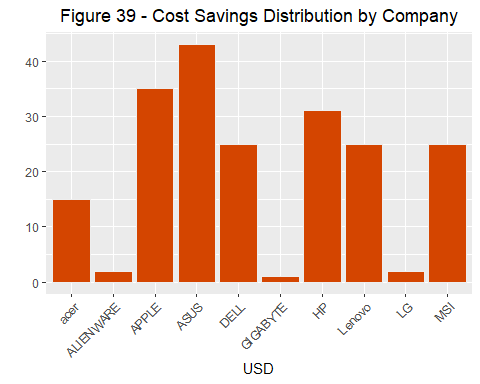


Figure 39 shows that the outliers from a varied group of companies. Although the most companies with wide divergence between Price and MRP were APPLE and Lenovo, the volume of the negative Cost Savings was more evenly spread out.

### Linear Model

linear\_model = lm(formula = Price ~ Company + Rating + Size + RAM + ProcessorManufacturer + DiskGB + OS + Cores + CostSavings, data=laptops\_norm)  
summary(linear\_model)

##   
## Call:  
## lm(formula = Price ~ Company + Rating + Size + RAM + ProcessorManufacturer +   
## DiskGB + OS + Cores + CostSavings, data = laptops\_norm)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -0.17806 -0.05358 -0.01391 0.04345 0.52200   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) -0.11478 0.04850 -2.367 0.018147 \*   
## Company -0.08527 0.01654 -5.154 0.00000031118 \*\*\*  
## Rating 0.02153 0.03575 0.602 0.547150   
## Size 0.07851 0.02282 3.441 0.000605 \*\*\*  
## RAM 0.27884 0.01938 14.385 < 0.0000000000000002 \*\*\*  
## ProcessorManufacturer 0.13313 0.02079 6.404 0.00000000024 \*\*\*  
## DiskGB 0.34960 0.03133 11.158 < 0.0000000000000002 \*\*\*  
## OS -0.13267 0.03123 -4.248 0.00002371565 \*\*\*  
## Cores 0.18347 0.02702 6.790 0.00000000002 \*\*\*  
## CostSavings -0.08471 0.03751 -2.259 0.024134 \*   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 0.08995 on 934 degrees of freedom  
## Multiple R-squared: 0.6812, Adjusted R-squared: 0.6782   
## F-statistic: 221.8 on 9 and 934 DF, p-value: < 0.00000000000000022

Using a linear model to explore and identify which relationships were significant shows that nearly all of the values except Rating contributed to the prediction of Price. The linear model is very different from the classification models used later to predict the PriceCategory but is a reasonable way to confirm which values were helpful in those predictions.

# Results

## Association Rule Mining

Association Rule Mining finds rules that will predict the occurrence of an item based on the occurrences of other items in the transaction. The Apriori algorithm is a method to efficiently generate frequent item sets and then prune rules with low confidence. To generate candidates efficiently the algorithm calculates the support for each item set. The support is the fraction of transactions that contain the itemset. The algorithm discards the itemset if the support is below the minimum support threshold. If the support is above the threshold, the algorithm builds larger itemsets with it and repeats the process until it runs out of itemsets. To prune the generated rules down, the algorithm calculates the confidence of the rule. The confidence shows how frequently items in Y appear in transactions that contain X. The algorithm starts with rules with the highest number of items, and if the confidence does not meet the confidence threshold then it discards that rule and any rule with the same right-hand side item. This results in a list of rules that includes the rule’s support, confidence, and lift. The lift is a measure of dependent or correlated events. It is analyzed because support and confidence are limited and could sometimes be misleading. A lift value greater than 1 indicates that the items in the rule and appear more often together than expected, and that the rule is meaningful.

# set the seed  
set.seed(1234)

### Pre-processing for ARM

# Remove the Price column which has been discretized into Cost Category  
arm\_laptops <- laptops[,!names(laptops) %in% c("Price")]  
  
# Discretize Rating  
arm\_laptops$Rating <- as.ordered(cut(arm\_laptops$Rating, breaks = c(0,0.5,1.5,2.5,3.5,4.5,5.1),   
 labels = c("zero", "one", "two", "three", "four", "five"),  
 right=FALSE))  
# Discretize Number of Ratings  
arm\_laptops$No\_of\_ratings <- as.ordered(cut(arm\_laptops$No\_of\_ratings, breaks = c(0,500,1000,1500,2000,2500,3000,6000,14001),   
 labels = c("0-499","500-999","1000-1499","1500-1999","2000-2499","2500-2999","3000-5999","6000-14000"),  
 right=FALSE))  
# Discretize Review  
arm\_laptops$Review <- as.ordered(cut(arm\_laptops$Review, breaks = c(0,250,500,750,2001),   
 labels = c("0-249", "250-499", "500-749", "750-2000"),  
 right=FALSE))  
# Convert Size to facor  
arm\_laptops$Size <- factor(arm\_laptops$Size)  
# Discretize MRP  
arm\_laptops$MRP <- as.ordered(cut(arm\_laptops$MRP, breaks = c(0,50000,100000,150000,200000,500001),   
 labels = c("0-49999", "50000-99999", "100000-149999","150000-199999","200000-500000"),  
 right=FALSE))  
# Discretize Cost Savings  
arm\_laptops$CostSavings <- as.ordered(cut(arm\_laptops$CostSavings, breaks = c(-Inf,0,200,400,Inf),   
 labels = c("<$0", "$0-199","$200-399",">=$400"),  
 right=FALSE))  
# Convert DealQuality to facotr  
arm\_laptops$DealQuality <- factor(arm\_laptops$DealQuality)  
# show the data types  
str(arm\_laptops)

## 'data.frame': 984 obs. of 14 variables:  
## $ CostCategory : Factor w/ 6 levels "Lightweight",..: 2 1 2 1 1 1 1 2 1 1 ...  
## $ Company : Factor w/ 19 levels "acer","ALIENWARE",..: 8 10 4 6 4 17 8 10 8 4 ...  
## $ Rating : Ord.factor w/ 4 levels "two"<"three"<..: 3 3 3 3 3 3 4 3 3 3 ...  
## $ No\_of\_ratings : Ord.factor w/ 8 levels "0-499"<"500-999"<..: 1 2 7 1 1 7 1 6 1 3 ...  
## $ Review : Ord.factor w/ 4 levels "0-249"<"250-499"<..: 1 1 3 1 1 3 1 1 1 1 ...  
## $ Size : Factor w/ 12 levels "29","34","35",..: 4 4 8 8 8 8 8 8 8 4 ...  
## $ RAM : Ord.factor w/ 4 levels "4"<"8"<"16"<"32": 3 2 2 2 2 2 2 2 2 1 ...  
## $ MRP : Ord.factor w/ 1 level "0-49999": 1 1 1 1 1 1 1 1 1 1 ...  
## $ ProcessorManufacturer: Factor w/ 5 levels "AMD","Intel",..: 1 2 2 2 2 2 2 2 2 2 ...  
## $ DiskGB : Ord.factor w/ 7 levels "64"<"128"<"256"<..: 4 3 4 7 4 4 4 4 4 3 ...  
## $ OS : Factor w/ 7 levels "Chrome OS","DOS",..: 7 7 7 6 7 7 7 7 7 7 ...  
## $ Cores : Ord.factor w/ 8 levels "2"<"4"<"6"<"8"<..: 3 2 3 1 2 3 2 3 2 1 ...  
## $ CostSavings : Ord.factor w/ 4 levels "<$0"<"$0-199"<..: 2 3 3 3 2 3 2 3 2 2 ...  
## $ DealQuality : Factor w/ 2 levels "0","1": 2 2 2 2 2 2 2 2 2 2 ...

The Apriori algorithm was run with support of 0.01 to 0.5 and confidence of 0.7 to 0.9. Running the algorithm with a support threshold of 0.3 and a confidence threshold of 0.8 resulted in the most interesting rules. The top 20 had a confidence of 0.82-0.94 and a lift of 1.58-1.86. Out of the top 20 rules generated, 2 of them had Cores=6 on the right hand side, 5 of the, had CostCategory=Consumer on the right hand side, and the rest had Disk=512 on the right hand side. The most interesting rule associated with CostCategory=Consumer was Rating=four, Review=0-249, DiskGB=512, Cores=6, and DealQuality=1. This indicates that laptops that have 512GB disk memory, 6 cores, a rating of 4, and up to 249 reviews are generally between $500 to $1000. The DealQuality value also indicates that these laptops are a relatively low price for components.

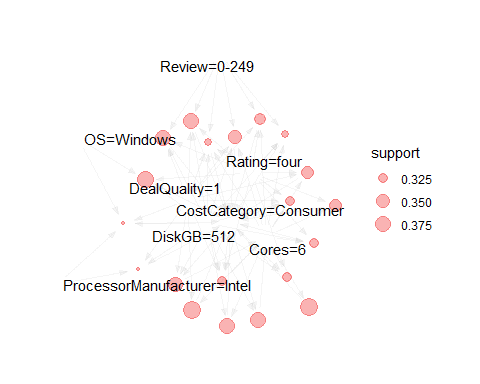
# Get the rules  
rules <- apriori(arm\_laptops, parameter=list(supp = 0.3, conf=0.8), control = list(verbose=F))  
## find non-redundant rules using improvement of confidence  
rules\_non\_redundant <- rules[!is.redundant(rules)]  
# Sort the rules  
rules\_non\_redundant<-sort(rules\_non\_redundant, by="lift", decreasing=TRUE)  
# Inspect the top 20 rules   
as(rules\_non\_redundant[1:20], "data.frame")

## rules  
## 1019 {CostCategory=Consumer,Rating=four,DiskGB=512,DealQuality=1} => {Cores=6}  
## 418 {CostCategory=Consumer,Rating=four,DiskGB=512} => {Cores=6}  
## 414 {CostCategory=Consumer,DiskGB=512,DealQuality=1} => {Cores=6}  
## 103 {CostCategory=Consumer,DiskGB=512} => {Cores=6}  
## 1598 {Rating=four,Review=0-249,DiskGB=512,Cores=6,DealQuality=1} => {CostCategory=Consumer}  
## 1018 {Rating=four,DiskGB=512,Cores=6,DealQuality=1} => {CostCategory=Consumer}  
## 1072 {Rating=four,Review=0-249,Cores=6,DealQuality=1} => {CostCategory=Consumer}  
## 433 {Rating=four,Cores=6,DealQuality=1} => {CostCategory=Consumer}  
## 1028 {Review=0-249,DiskGB=512,Cores=6,DealQuality=1} => {CostCategory=Consumer}  
## 105 {Cores=6,DealQuality=1} => {CostCategory=Consumer}  
## 1043 {Rating=four,Review=0-249,DiskGB=512,Cores=6} => {CostCategory=Consumer}  
## 417 {Rating=four,DiskGB=512,Cores=6} => {CostCategory=Consumer}  
## 107 {Rating=four,Cores=6} => {CostCategory=Consumer}  
## 102 {DiskGB=512,Cores=6} => {CostCategory=Consumer}  
## 16 {Cores=6} => {CostCategory=Consumer}  
## 1293 {ProcessorManufacturer=Intel,DiskGB=512,OS=Windows,DealQuality=1} => {CostCategory=Consumer}  
## 688 {ProcessorManufacturer=Intel,DiskGB=512,DealQuality=1} => {CostCategory=Consumer}  
## 1741 {Rating=four,Review=0-249,DiskGB=512,OS=Windows,DealQuality=1} => {CostCategory=Consumer}  
## 1316 {Rating=four,Review=0-249,DiskGB=512,DealQuality=1} => {CostCategory=Consumer}  
## 1311 {Rating=four,DiskGB=512,OS=Windows,DealQuality=1} => {CostCategory=Consumer}  
## support confidence coverage lift count  
## 1019 0.3221545 0.8342105 0.3861789 1.857156 317  
## 418 0.3231707 0.8324607 0.3882114 1.853261 318  
## 414 0.3638211 0.8211009 0.4430894 1.827971 358  
## 103 0.3678862 0.8171558 0.4502033 1.819188 362  
## 1598 0.3140244 0.9392097 0.3343496 1.812122 309  
## 1018 0.3221545 0.9378698 0.3434959 1.809537 317  
## 1072 0.3333333 0.9371429 0.3556911 1.808134 328  
## 433 0.3445122 0.9364641 0.3678862 1.806825 339  
## 1028 0.3526423 0.9278075 0.3800813 1.790123 347  
## 105 0.3902439 0.9275362 0.4207317 1.789599 384  
## 1043 0.3150407 0.9011628 0.3495935 1.738714 310  
## 417 0.3231707 0.9008499 0.3587398 1.738110 318  
## 107 0.3475610 0.8906250 0.3902439 1.718382 342  
## 102 0.3678862 0.8894349 0.4136179 1.716086 362  
## 16 0.3963415 0.8823529 0.4491870 1.702422 390  
## 1293 0.3058943 0.8361111 0.3658537 1.613203 301  
## 688 0.3058943 0.8337950 0.3668699 1.608734 301  
## 1741 0.3770325 0.8281250 0.4552846 1.597794 371  
## 1316 0.3770325 0.8262806 0.4563008 1.594236 371  
## 1311 0.3861789 0.8225108 0.4695122 1.586962 380

# Visualize the rules in a map  
plot(rules\_non\_redundant[1:20],method="graph", shading=NULL, main="Figure 39 - ARM Graph Plot")

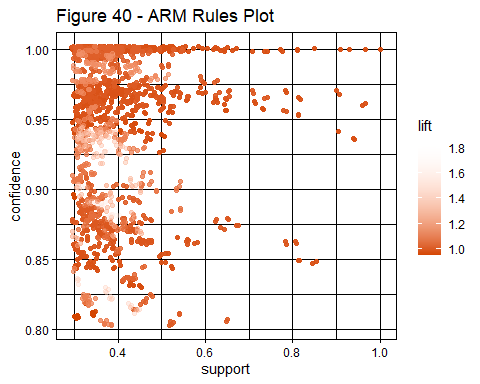
## Warning: Unknown control parameters: main

## Available control parameters (with default values):  
## layout = stress  
## circular = FALSE  
## ggraphdots = NULL  
## edges = <environment>  
## nodes = <environment>  
## nodetext = <environment>  
## colors = c("#EE0000FF", "#EEEEEEFF")  
## engine = ggplot2  
## max = 100  
## verbose = FALSE



plot(rules, main="Figure 40 - ARM Rules Plot", color=syracuseOrange)

## To reduce overplotting, jitter is added! Use jitter = 0 to prevent jitter.



### ARM with Cost Category set on the right hand side

Setting the CostCategory=Lightweight variable as the right-hand side ensures the generated rules show what variable values generally correlate with the laptops that cost between 0 and $500. The algorithm was run with a support of 0.001 to 0.9 and a confidence of 0.7 to 0.9. Running the algorithm with a support threshold of 0.01 and a confidence threshold of 0.8 resulted in the most interesting rules. The top 20 had a confidence of 1 and a lift of 5.4. The rules contained many of the smaller components like RAM=4, DiskGB=256, and Cores=2 which is to be expected. The interesting thing about these rules is that the Company Lenovo is grouped with CostSavings=$200-399 whereas ASUS is grouped with CostSavings=$0-199.

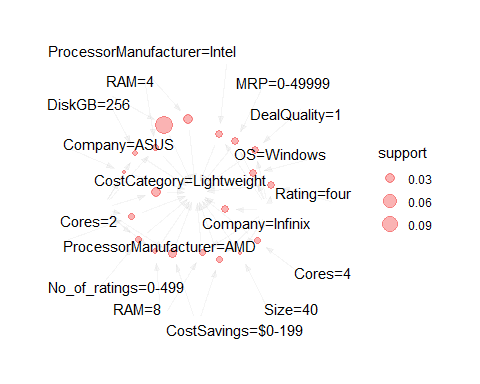
# Get the rules  
rules<-apriori(data=arm\_laptops, parameter=list(supp=0.01,conf = 0.7),   
 appearance = list(default="lhs",rhs="CostCategory=Lightweight"),  
 control = list(verbose=F))  
## find non-redundant rules using improvement of confidence  
rules\_non\_redundant <- rules[!is.redundant(rules)]  
# Sort the rules  
rules\_non\_redundant<-sort(rules\_non\_redundant, decreasing=TRUE,by="lift")  
# Inspect the top 20 rules   
as(rules\_non\_redundant[1:20], "data.frame")

## rules  
## 5 {Company=Infinix,Cores=4} => {CostCategory=Lightweight}  
## 14 {RAM=4,Cores=2} => {CostCategory=Lightweight}  
## 15 {DiskGB=256,Cores=2} => {CostCategory=Lightweight}  
## 29 {Company=ASUS,RAM=4} => {CostCategory=Lightweight}  
## 87 {Company=Infinix,No\_of\_ratings=0-499,RAM=8} => {CostCategory=Lightweight}  
## 89 {Company=Infinix,Size=40,RAM=8} => {CostCategory=Lightweight}  
## 94 {Company=Infinix,Review=0-249,RAM=8} => {CostCategory=Lightweight}  
## 150 {Rating=four,No\_of\_ratings=0-499,Cores=2} => {CostCategory=Lightweight}  
## 249 {No\_of\_ratings=500-999,Size=36,DiskGB=256} => {CostCategory=Lightweight}  
## 259 {No\_of\_ratings=500-999,Size=36,Cores=4} => {CostCategory=Lightweight}  
## 269 {Company=Lenovo,No\_of\_ratings=500-999,Size=36} => {CostCategory=Lightweight}  
## 270 {No\_of\_ratings=500-999,Size=36,CostSavings=$200-399} => {CostCategory=Lightweight}  
## 278 {Company=Lenovo,No\_of\_ratings=500-999,CostSavings=$200-399} => {CostCategory=Lightweight}  
## 325 {Company=Lenovo,Size=36,DiskGB=256} => {CostCategory=Lightweight}  
## 395 {Company=Lenovo,Size=36,Cores=4} => {CostCategory=Lightweight}  
## 404 {Company=Lenovo,Cores=4,CostSavings=$200-399} => {CostCategory=Lightweight}  
## 586 {Company=ASUS,Rating=four,Cores=2,CostSavings=$0-199} => {CostCategory=Lightweight}  
## 592 {Rating=four,ProcessorManufacturer=AMD,Cores=2,CostSavings=$0-199} => {CostCategory=Lightweight}  
## 639 {Rating=four,OS=Windows,Cores=2,CostSavings=$0-199} => {CostCategory=Lightweight}  
## 666 {Rating=four,RAM=8,OS=Windows,Cores=2} => {CostCategory=Lightweight}  
## support confidence coverage lift count  
## 5 0.01626016 1 0.01626016 5.406593 16  
## 14 0.01524390 1 0.01524390 5.406593 15  
## 15 0.01422764 1 0.01422764 5.406593 14  
## 29 0.01321138 1 0.01321138 5.406593 13  
## 87 0.01117886 1 0.01117886 5.406593 11  
## 89 0.01117886 1 0.01117886 5.406593 11  
## 94 0.01219512 1 0.01219512 5.406593 12  
## 150 0.01626016 1 0.01626016 5.406593 16  
## 249 0.04573171 1 0.04573171 5.406593 45  
## 259 0.04369919 1 0.04369919 5.406593 43  
## 269 0.04369919 1 0.04369919 5.406593 43  
## 270 0.04369919 1 0.04369919 5.406593 43  
## 278 0.04471545 1 0.04471545 5.406593 44  
## 325 0.04674797 1 0.04674797 5.406593 46  
## 395 0.04471545 1 0.04471545 5.406593 44  
## 404 0.07113821 1 0.07113821 5.406593 70  
## 586 0.01117886 1 0.01117886 5.406593 11  
## 592 0.01321138 1 0.01321138 5.406593 13  
## 639 0.02134146 1 0.02134146 5.406593 21  
## 666 0.01219512 1 0.01219512 5.406593 12

# Visualize the rules in a map  
plot(rules[1:20],method="graph", shading=NULL, main="Figure 41 - ARM Graph Plot")

## Warning: Unknown control parameters: main

## Available control parameters (with default values):  
## layout = stress  
## circular = FALSE  
## ggraphdots = NULL  
## edges = <environment>  
## nodes = <environment>  
## nodetext = <environment>  
## colors = c("#EE0000FF", "#EEEEEEFF")  
## engine = ggplot2  
## max = 100  
## verbose = FALSE



Setting the CostCategory=Consumer variable as the right-hand side ensures the generated rules show what variable values generally correlate with the laptops that cost between $500 and $1000. The algorithm was run with a support of 0.001 to 0.9 and a confidence of 0.7 to 0.9. Running the algorithm with a support threshold of 0.01 and a confidence threshold of 0.8 resulted in the most interesting rules. The top 20 had a confidence of 1 and a lift of 1.93. The rules contained components like RAM=8, Cores=6, Size=40, and DiskGB=512. The interesting thing about these rules is that the Companys acer and Dell are grouped with DealQuality=1. The rules also contain Lenovo, ASUS, and HP indicating the variety of options at the Consumer level.

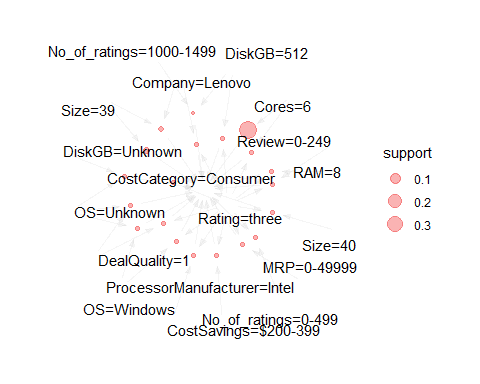
# Get the rules  
rules<-apriori(data=arm\_laptops, parameter=list(supp=.01,conf = 0.8),   
 appearance = list(default="lhs",rhs="CostCategory=Consumer"),  
 control = list(verbose=F))  
## find non-redundant rules using improvement of confidence  
rules\_non\_redundant <- rules[!is.redundant(rules)]  
# Sort the rules  
rules\_non\_redundant<-sort(rules\_non\_redundant, decreasing=TRUE,by="lift")  
# Inspect the top 20 rules   
as(rules\_non\_redundant[1:20], "data.frame")

## rules  
## 2 {Size=39} => {CostCategory=Consumer}  
## 9 {Rating=three,Cores=6} => {CostCategory=Consumer}  
## 85 {Company=Lenovo,Rating=three,DiskGB=512} => {CostCategory=Consumer}  
## 119 {Rating=three,RAM=8,DiskGB=512} => {CostCategory=Consumer}  
## 125 {Rating=three,Size=40,DiskGB=512} => {CostCategory=Consumer}  
## 132 {Rating=three,DiskGB=512,DealQuality=1} => {CostCategory=Consumer}  
## 222 {Company=acer,No\_of\_ratings=0-499,Cores=6} => {CostCategory=Consumer}  
## 232 {Company=ASUS,No\_of\_ratings=500-999,CostSavings=$200-399} => {CostCategory=Consumer}  
## 235 {Company=ASUS,No\_of\_ratings=500-999,OS=Windows} => {CostCategory=Consumer}  
## 238 {No\_of\_ratings=500-999,Size=40,Cores=6} => {CostCategory=Consumer}  
## 258 {Company=ASUS,Rating=five,RAM=8} => {CostCategory=Consumer}  
## 279 {Company=HP,Size=36,Cores=6} => {CostCategory=Consumer}  
## 398 {Company=Lenovo,No\_of\_ratings=0-499,Cores=6} => {CostCategory=Consumer}  
## 1058 {Company=acer,Cores=6,CostSavings=$200-399,DealQuality=1} => {CostCategory=Consumer}  
## 1070 {Company=acer,RAM=8,ProcessorManufacturer=Intel,Cores=6} => {CostCategory=Consumer}  
## 1077 {Company=acer,Size=40,ProcessorManufacturer=Intel,Cores=6} => {CostCategory=Consumer}  
## 1152 {Company=DELL,ProcessorManufacturer=AMD,Cores=6,DealQuality=1} => {CostCategory=Consumer}  
## 1176 {Company=DELL,No\_of\_ratings=2500-2999,Size=40,DealQuality=1} => {CostCategory=Consumer}  
## 1189 {Company=DELL,No\_of\_ratings=0-499,Size=40,Cores=6} => {CostCategory=Consumer}  
## 1200 {Company=DELL,RAM=8,Cores=6,DealQuality=1} => {CostCategory=Consumer}  
## support confidence coverage lift count  
## 2 0.02642276 1 0.02642276 1.929412 26  
## 9 0.01422764 1 0.01422764 1.929412 14  
## 85 0.01321138 1 0.01321138 1.929412 13  
## 119 0.01626016 1 0.01626016 1.929412 16  
## 125 0.01524390 1 0.01524390 1.929412 15  
## 132 0.01626016 1 0.01626016 1.929412 16  
## 222 0.01219512 1 0.01219512 1.929412 12  
## 232 0.01219512 1 0.01219512 1.929412 12  
## 235 0.01219512 1 0.01219512 1.929412 12  
## 238 0.01117886 1 0.01117886 1.929412 11  
## 258 0.01219512 1 0.01219512 1.929412 12  
## 279 0.05995935 1 0.05995935 1.929412 59  
## 398 0.09247967 1 0.09247967 1.929412 91  
## 1058 0.01219512 1 0.01219512 1.929412 12  
## 1070 0.01016260 1 0.01016260 1.929412 10  
## 1077 0.01219512 1 0.01219512 1.929412 12  
## 1152 0.01016260 1 0.01016260 1.929412 10  
## 1176 0.01422764 1 0.01422764 1.929412 14  
## 1189 0.01524390 1 0.01524390 1.929412 15  
## 1200 0.03252033 1 0.03252033 1.929412 32

# Visualize the rules in a map  
plot(rules[1:20],method="graph", shading=NULL, main="Figure 41 - ARM Graph Plot")

## Warning: Unknown control parameters: main

## Available control parameters (with default values):  
## layout = stress  
## circular = FALSE  
## ggraphdots = NULL  
## edges = <environment>  
## nodes = <environment>  
## nodetext = <environment>  
## colors = c("#EE0000FF", "#EEEEEEFF")  
## engine = ggplot2  
## max = 100  
## verbose = FALSE



Setting the CostCategory=Commercial variable as the right-hand side ensures the generated rules show what variable values generally correlate with the laptops that cost between $1000 and $2000. The algorithm was run with a support of 0.001 to 0.02 and a confidence of 0.7 to 0.9. Running the algorithm with a support threshold of 0.02 and a confidence threshold of 0.8 resulted in the most interesting rules. The top 20 had a confidence of 0.85-1 and a lift of 4.06-4.8. The rules contained components like RAM=16, Cores=8, and DiskGB=1000. The interesting thing about these rules is that the DealQuality=0 and CostSavings=$200-399 are in many of the rules. This indicates that the laptops over $1000 may not be worth it unless the components are necessary.

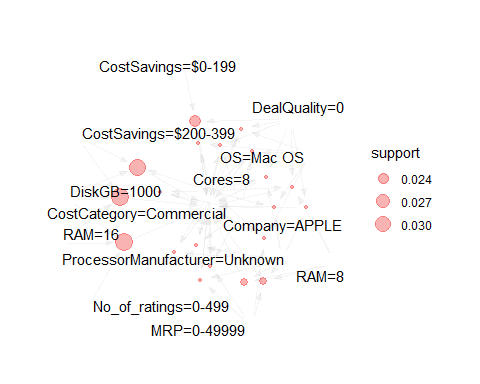
# Get the rules  
rules<-apriori(data=arm\_laptops, parameter=list(supp=.02,conf = 0.8),   
 appearance = list(default="lhs",rhs="CostCategory=Commercial"),  
 control = list(verbose=F))  
## find non-redundant rules using improvement of confidence  
rules\_non\_redundant <- rules[!is.redundant(rules)]  
# Sort the rules  
rules\_non\_redundant<-sort(rules\_non\_redundant, decreasing=TRUE,by="lift")  
# Inspect the top 20 rules   
as(rules\_non\_redundant[1:20], "data.frame")

## rules  
## 13 {RAM=8,ProcessorManufacturer=Unknown,DealQuality=0} => {CostCategory=Commercial}  
## 69 {Rating=four,No\_of\_ratings=0-499,Cores=8,DealQuality=0} => {CostCategory=Commercial}  
## 58 {No\_of\_ratings=0-499,Size=40,RAM=16,DiskGB=1000} => {CostCategory=Commercial}  
## 65 {RAM=16,Cores=8,CostSavings=$200-399,DealQuality=0} => {CostCategory=Commercial}  
## 44 {RAM=16,DiskGB=1000,Cores=8,CostSavings=$200-399} => {CostCategory=Commercial}  
## 19 {Cores=8,CostSavings=$0-199,DealQuality=0} => {CostCategory=Commercial}  
## 17 {RAM=16,DiskGB=1000,CostSavings=$200-399} => {CostCategory=Commercial}  
## 60 {No\_of\_ratings=0-499,RAM=16,DiskGB=1000,OS=Windows} => {CostCategory=Commercial}  
## 77 {No\_of\_ratings=0-499,Size=40,RAM=16,DealQuality=0} => {CostCategory=Commercial}  
## 79 {No\_of\_ratings=0-499,RAM=16,OS=Windows,DealQuality=0} => {CostCategory=Commercial}  
## 51 {No\_of\_ratings=0-499,Size=40,DiskGB=1000,Cores=8} => {CostCategory=Commercial}  
## 1 {OS=Mac OS,Cores=8} => {CostCategory=Commercial}  
## 2 {Company=APPLE,Cores=8} => {CostCategory=Commercial}  
## 3 {ProcessorManufacturer=Unknown,Cores=8} => {CostCategory=Commercial}  
## 20 {RAM=8,Cores=8,DealQuality=0} => {CostCategory=Commercial}  
## 21 {RAM=16,CostSavings=$200-399,DealQuality=0} => {CostCategory=Commercial}  
## 86 {Rating=four,No\_of\_ratings=0-499,OS=Windows,DealQuality=0} => {CostCategory=Commercial}  
## 16 {DiskGB=1000,Cores=8,CostSavings=$200-399} => {CostCategory=Commercial}  
## 25 {Rating=four,No\_of\_ratings=0-499,DealQuality=0} => {CostCategory=Commercial}  
## 4 {RAM=8,ProcessorManufacturer=Unknown} => {CostCategory=Commercial}  
## support confidence coverage lift count  
## 13 0.02134146 1.0000000 0.02134146 4.800000 21  
## 69 0.02337398 0.9200000 0.02540650 4.416000 23  
## 58 0.02235772 0.9166667 0.02439024 4.400000 22  
## 65 0.03252033 0.9142857 0.03556911 4.388571 32  
## 44 0.03150407 0.9117647 0.03455285 4.376471 31  
## 19 0.02540650 0.8928571 0.02845528 4.285714 25  
## 17 0.03252033 0.8888889 0.03658537 4.266667 32  
## 60 0.03252033 0.8888889 0.03658537 4.266667 32  
## 77 0.02439024 0.8888889 0.02743902 4.266667 24  
## 79 0.03963415 0.8863636 0.04471545 4.254545 39  
## 51 0.02235772 0.8800000 0.02540650 4.224000 22  
## 1 0.02134146 0.8750000 0.02439024 4.200000 21  
## 2 0.02134146 0.8750000 0.02439024 4.200000 21  
## 3 0.02134146 0.8750000 0.02439024 4.200000 21  
## 20 0.02134146 0.8750000 0.02439024 4.200000 21  
## 21 0.03455285 0.8717949 0.03963415 4.184615 34  
## 86 0.02540650 0.8620690 0.02947154 4.137931 25  
## 16 0.03150407 0.8611111 0.03658537 4.133333 31  
## 25 0.02845528 0.8484848 0.03353659 4.072727 28  
## 4 0.02235772 0.8461538 0.02642276 4.061538 22

# Visualize the rules in a map  
plot(rules[1:20],method="graph", shading=NULL, main="Figure 41 - ARM Graph Plot")

## Warning: Unknown control parameters: main

## Available control parameters (with default values):  
## layout = stress  
## circular = FALSE  
## ggraphdots = NULL  
## edges = <environment>  
## nodes = <environment>  
## nodetext = <environment>  
## colors = c("#EE0000FF", "#EEEEEEFF")  
## engine = ggplot2  
## max = 100  
## verbose = FALSE

 Setting the CostCategory=Gaming variable as the right-hand side ensures the generated rules show what variable values generally correlate with the laptops that cost between $2000 and $3000. The algorithm was run with a support of 0.001 and a confidence of 0.8. The top 20 had a confidence of 0.85-1 and a lift of 4.06-4.8. The rules contained components like RAM=32, Cores=12, and DiskGB=2000. The interesting thing about these rules is that the Company’s included Alienware, MSI, acer, Lenovo, and ASUS indicating there are a variety of options at the gaming level.

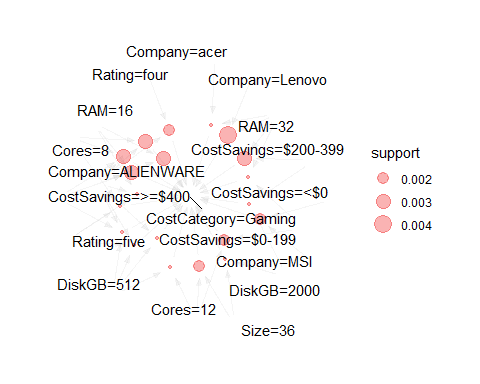
# Get the rules  
rules<-apriori(data=arm\_laptops, parameter=list(supp=.001,conf = 0.8),   
 appearance = list(default="lhs",rhs="CostCategory=Gaming"),  
 control = list(verbose=F))  
## find non-redundant rules using improvement of confidence  
rules\_non\_redundant <- rules[!is.redundant(rules)]  
# Sort the rules  
rules\_non\_redundant<-sort(rules\_non\_redundant, decreasing=TRUE,by="lift")  
# Inspect the top 20 rules   
as(rules\_non\_redundant[1:20], "data.frame")

## rules  
## 1 {Company=ALIENWARE,Cores=8} => {CostCategory=Gaming}  
## 2 {Company=ALIENWARE,RAM=16} => {CostCategory=Gaming}  
## 3 {Company=ALIENWARE,DiskGB=512} => {CostCategory=Gaming}  
## 4 {Company=ALIENWARE,Rating=four} => {CostCategory=Gaming}  
## 5 {Size=36,Cores=12} => {CostCategory=Gaming}  
## 6 {Cores=12,CostSavings=$0-199} => {CostCategory=Gaming}  
## 7 {DiskGB=512,Cores=12} => {CostCategory=Gaming}  
## 8 {Size=36,DiskGB=2000} => {CostCategory=Gaming}  
## 9 {DiskGB=2000,CostSavings=$200-399} => {CostCategory=Gaming}  
## 10 {Company=MSI,CostSavings=<$0} => {CostCategory=Gaming}  
## 11 {Company=acer,RAM=32} => {CostCategory=Gaming}  
## 12 {Company=Lenovo,RAM=32} => {CostCategory=Gaming}  
## 13 {RAM=32,CostSavings=$0-199} => {CostCategory=Gaming}  
## 14 {RAM=32,CostSavings=$200-399} => {CostCategory=Gaming}  
## 94 {Company=acer,Size=41,DiskGB=2000} => {CostCategory=Gaming}  
## 95 {Company=acer,No\_of\_ratings=2500-2999,DiskGB=2000} => {CostCategory=Gaming}  
## 99 {RAM=16,DiskGB=2000,CostSavings=>=$400} => {CostCategory=Gaming}  
## 101 {Company=ASUS,RAM=16,DiskGB=2000} => {CostCategory=Gaming}  
## 113 {No\_of\_ratings=2500-2999,RAM=16,DiskGB=2000} => {CostCategory=Gaming}  
## 120 {Company=MSI,Size=44,RAM=32} => {CostCategory=Gaming}  
## support confidence coverage lift count  
## 1 0.003048780 1 0.003048780 16.67797 3  
## 2 0.003048780 1 0.003048780 16.67797 3  
## 3 0.001016260 1 0.001016260 16.67797 1  
## 4 0.002032520 1 0.002032520 16.67797 2  
## 5 0.002032520 1 0.002032520 16.67797 2  
## 6 0.002032520 1 0.002032520 16.67797 2  
## 7 0.001016260 1 0.001016260 16.67797 1  
## 8 0.001016260 1 0.001016260 16.67797 1  
## 9 0.001016260 1 0.001016260 16.67797 1  
## 10 0.002032520 1 0.002032520 16.67797 2  
## 11 0.001016260 1 0.001016260 16.67797 1  
## 12 0.004065041 1 0.004065041 16.67797 4  
## 13 0.001016260 1 0.001016260 16.67797 1  
## 14 0.003048780 1 0.003048780 16.67797 3  
## 94 0.001016260 1 0.001016260 16.67797 1  
## 95 0.001016260 1 0.001016260 16.67797 1  
## 99 0.001016260 1 0.001016260 16.67797 1  
## 101 0.001016260 1 0.001016260 16.67797 1  
## 113 0.001016260 1 0.001016260 16.67797 1  
## 120 0.001016260 1 0.001016260 16.67797 1

# Visualize the rules in a map  
plot(rules[1:20],method="graph", shading=NULL, main="Figure 41 - ARM Graph Plot")

## Warning: Unknown control parameters: main

## Available control parameters (with default values):  
## layout = stress  
## circular = FALSE  
## ggraphdots = NULL  
## edges = <environment>  
## nodes = <environment>  
## nodetext = <environment>  
## colors = c("#EE0000FF", "#EEEEEEFF")  
## engine = ggplot2  
## max = 100  
## verbose = FALSE



Setting the CostCategory=Mining variable as the right-hand side ensures the generated rules show what variable values generally correlate with the laptops that cost between $3000 and $4000. The algorithm was run with a support of 0.001 and a confidence of 0.8. The top 20 had a confidence of 1 and a lift of 49.2. The rules contained components like RAM=32, Cores=10, and DiskGB=2000. One interesting thing about these rules is that Size is in the list multiple times and they are all pretty large laptops at sizes 41 to 46cm.

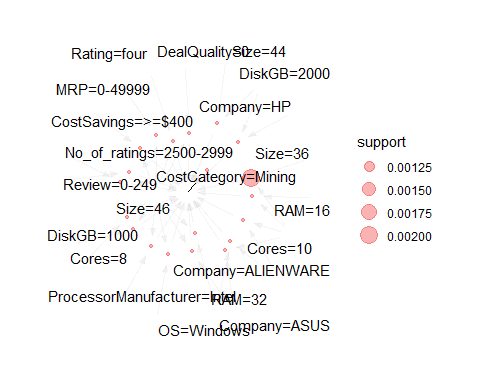
# Get the rules  
rules<-apriori(data=arm\_laptops, parameter=list(supp=.001,conf = 0.8),   
 appearance = list(default="lhs",rhs="CostCategory=Mining"),  
 control = list(verbose=F))  
## find non-redundant rules using improvement of confidence  
rules\_non\_redundant <- rules[!is.redundant(rules)]  
# Sort the rules  
rules\_non\_redundant<-sort(rules\_non\_redundant, decreasing=TRUE,by="lift")  
# Inspect the top 20 rules   
as(rules\_non\_redundant[1:20], "data.frame")

## rules  
## 1 {Size=46} => {CostCategory=Mining}  
## 14 {Company=ALIENWARE,Cores=10} => {CostCategory=Mining}  
## 15 {Company=ALIENWARE,RAM=32} => {CostCategory=Mining}  
## 16 {Size=44,DiskGB=2000} => {CostCategory=Mining}  
## 17 {Company=HP,DiskGB=2000} => {CostCategory=Mining}  
## 18 {Company=ASUS,Cores=10} => {CostCategory=Mining}  
## 19 {Size=36,Cores=10} => {CostCategory=Mining}  
## 20 {RAM=16,Cores=10} => {CostCategory=Mining}  
## 108 {Company=ALIENWARE,Rating=five,DiskGB=1000} => {CostCategory=Mining}  
## 109 {Size=41,Cores=12,CostSavings=<$0} => {CostCategory=Mining}  
## 110 {Size=41,DiskGB=1000,Cores=12} => {CostCategory=Mining}  
## 133 {No\_of\_ratings=0-499,OS=Mac OS,Cores=10} => {CostCategory=Mining}  
## 135 {Company=APPLE,No\_of\_ratings=0-499,Cores=10} => {CostCategory=Mining}  
## 137 {DiskGB=Unknown,Cores=10,DealQuality=0} => {CostCategory=Mining}  
## 138 {No\_of\_ratings=2500-2999,DiskGB=Unknown,Cores=10} => {CostCategory=Mining}  
## 140 {DiskGB=Unknown,OS=Windows,Cores=10} => {CostCategory=Mining}  
## 142 {No\_of\_ratings=0-499,Cores=10,CostSavings=<$0} => {CostCategory=Mining}  
## 143 {Rating=five,RAM=32,Cores=10} => {CostCategory=Mining}  
## 144 {RAM=32,Cores=10,CostSavings=>=$400} => {CostCategory=Mining}  
## 146 {No\_of\_ratings=0-499,RAM=32,Cores=10} => {CostCategory=Mining}  
## support confidence coverage lift count  
## 1 0.00101626 1 0.00101626 49.2 1  
## 14 0.00101626 1 0.00101626 49.2 1  
## 15 0.00101626 1 0.00101626 49.2 1  
## 16 0.00101626 1 0.00101626 49.2 1  
## 17 0.00101626 1 0.00101626 49.2 1  
## 18 0.00101626 1 0.00101626 49.2 1  
## 19 0.00203252 1 0.00203252 49.2 2  
## 20 0.00101626 1 0.00101626 49.2 1  
## 108 0.00101626 1 0.00101626 49.2 1  
## 109 0.00203252 1 0.00203252 49.2 2  
## 110 0.00203252 1 0.00203252 49.2 2  
## 133 0.00101626 1 0.00101626 49.2 1  
## 135 0.00101626 1 0.00101626 49.2 1  
## 137 0.00101626 1 0.00101626 49.2 1  
## 138 0.00101626 1 0.00101626 49.2 1  
## 140 0.00101626 1 0.00101626 49.2 1  
## 142 0.00101626 1 0.00101626 49.2 1  
## 143 0.00101626 1 0.00101626 49.2 1  
## 144 0.00101626 1 0.00101626 49.2 1  
## 146 0.00203252 1 0.00203252 49.2 2

# Visualize the rules in a map  
plot(rules[1:20],method="graph", shading=NULL, main="Figure 41 - ARM Graph Plot")

## Warning: Unknown control parameters: main

## Available control parameters (with default values):  
## layout = stress  
## circular = FALSE  
## ggraphdots = NULL  
## edges = <environment>  
## nodes = <environment>  
## nodetext = <environment>  
## colors = c("#EE0000FF", "#EEEEEEFF")  
## engine = ggplot2  
## max = 100  
## verbose = FALSE

 Setting the CostCategory=Scientific variable as the right-hand side ensures the generated rules show what variable values generally correlate with the laptops that cost over $4000. The algorithm was run with a support of 0.001 and a confidence of 0.8. The top 14 had a confidence of 1 and a lift of 123. The rules contained components like RAM=32, Cores=10, DiskGB=2000, and Size=43. One interesting thing about these rules is that the companies included MSI, ASUS and Apple. MSI was grouped with a CostSavings=>=$400 so that may be a good company to check out for a good deal.

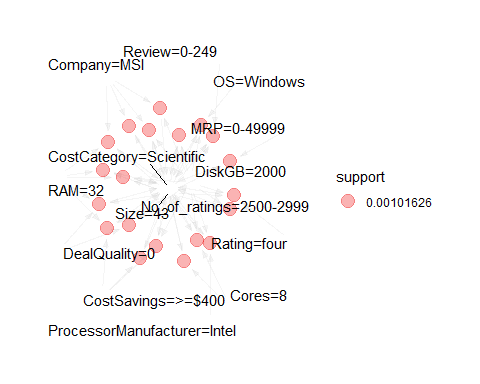
# Get the rules  
rules<-apriori(data=arm\_laptops, parameter=list(supp=.001,conf = 0.8),   
 appearance = list(default="lhs",rhs="CostCategory=Scientific"),  
 control = list(verbose=F))  
## find non-redundant rules using improvement of confidence  
rules\_non\_redundant <- rules[!is.redundant(rules)]  
# Sort the rules  
rules\_non\_redundant<-sort(rules\_non\_redundant, decreasing=TRUE,by="lift")  
# Inspect the top 20 rules   
as(rules\_non\_redundant, "data.frame")

## rules  
## 1 {Size=43,DiskGB=2000} => {CostCategory=Scientific}  
## 2 {Size=43,RAM=32} => {CostCategory=Scientific}  
## 3 {Company=MSI,Size=43} => {CostCategory=Scientific}  
## 4 {Size=43,CostSavings=>=$400} => {CostCategory=Scientific}  
## 5 {Size=43,Cores=8} => {CostCategory=Scientific}  
## 6 {Company=MSI,DiskGB=2000} => {CostCategory=Scientific}  
## 62 {No\_of\_ratings=2500-2999,Size=41,Cores=10} => {CostCategory=Scientific}  
## 63 {Company=MSI,RAM=32,CostSavings=>=$400} => {CostCategory=Scientific}  
## 258 {Company=ASUS,RAM=32,ProcessorManufacturer=Intel,DiskGB=2000} => {CostCategory=Scientific}  
## 295 {Size=41,ProcessorManufacturer=Intel,DiskGB=2000,CostSavings=>=$400} => {CostCategory=Scientific}  
## 296 {Company=ASUS,Size=41,ProcessorManufacturer=Intel,DiskGB=2000} => {CostCategory=Scientific}  
## 307 {No\_of\_ratings=2500-2999,Size=41,RAM=32,OS=Mac OS} => {CostCategory=Scientific}  
## 308 {Company=APPLE,No\_of\_ratings=2500-2999,Size=41,RAM=32} => {CostCategory=Scientific}  
## 309 {No\_of\_ratings=2500-2999,Size=41,RAM=32,ProcessorManufacturer=Unknown} => {CostCategory=Scientific}  
## support confidence coverage lift count  
## 1 0.00101626 1 0.00101626 123 1  
## 2 0.00101626 1 0.00101626 123 1  
## 3 0.00101626 1 0.00101626 123 1  
## 4 0.00101626 1 0.00101626 123 1  
## 5 0.00101626 1 0.00101626 123 1  
## 6 0.00101626 1 0.00101626 123 1  
## 62 0.00304878 1 0.00304878 123 3  
## 63 0.00101626 1 0.00101626 123 1  
## 258 0.00101626 1 0.00101626 123 1  
## 295 0.00101626 1 0.00101626 123 1  
## 296 0.00101626 1 0.00101626 123 1  
## 307 0.00304878 1 0.00304878 123 3  
## 308 0.00304878 1 0.00304878 123 3  
## 309 0.00304878 1 0.00304878 123 3

# Visualize the rules in a map  
plot(rules[1:20],method="graph", shading=NULL, main="Figure 41 - ARM Graph Plot")

## Warning: Unknown control parameters: main

## Available control parameters (with default values):  
## layout = stress  
## circular = FALSE  
## ggraphdots = NULL  
## edges = <environment>  
## nodes = <environment>  
## nodetext = <environment>  
## colors = c("#EE0000FF", "#EEEEEEFF")  
## engine = ggplot2  
## max = 100  
## verbose = FALSE



All of the following models will use a 2-fold CV training strategy due to the length of time required to test all the parameters and model variations. It’s also a consistent base that should allow each model to be compared on a relatively similar baseline. The results will be in the form of an accuracy value determined by the prediction table outputs.

## Expectation Maximization (EM)

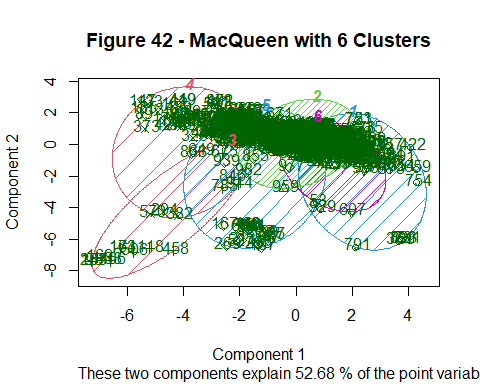
dataset\_list <- list("original" = laptops, "numerical" = laptops\_num, "normalized" = laptops\_norm)  
  
best\_em\_acc = 0  
best\_em\_ds = ""  
for (name in names(dataset\_list))   
{  
 ds = dataset\_list[[name]]  
 classes <- ds$CostCategory  
 mod\_ssc <- MclustSSC(ds, classes)  
 pred\_ssc <- predict(mod\_ssc)  
 t\_results = table(Predicted = pred\_ssc$classification, Actual = classes, useNA = "ifany")  
 accuracy\_results = get\_accuracy\_rate(t\_results, length(classes))  
   
 if(accuracy\_results > best\_em\_acc)  
 {  
 best\_em\_acc = accuracy\_results  
 best\_em\_ds = name  
 }  
}  
  
print(paste("The best Expectation Maximization accuracy was", round(best\_em\_acc, digits = 1), "using the", best\_em\_ds, "dataset", sep=" "))

## [1] "The best Expectation Maximization accuracy was 99.9 using the normalized dataset"

## K-Means

The K-Means algorithm will be testing 2 datasets, 4 different algorithms and 7 different cluster sizes to provide a wide range of possible outcomes.

set.seed(68)  
  
dataset\_list <- list("numerical" = laptops\_num, "normalized" = laptops\_norm)  
algorithms = c("Hartigan-Wong", "Lloyd", "Forgy", "MacQueen")  
cluster\_counts = c(2,3,4,5,6,7,8,9)  
  
best\_k\_acc = 0  
best\_k\_algo = ""  
best\_k\_clust = 0  
best\_k\_model = NULL  
best\_k\_ds = ""  
for (name in names(dataset\_list))   
{  
 ds = dataset\_list[[name]]  
 ds\_km = ds[,-c(1)]  
 classes <- ds$CostCategory  
   
 for (algo\_name in algorithms)  
 {  
 for(clusters in cluster\_counts)  
 {  
 cluster\_model <- kmeans(ds\_km, clusters, algorithm=algo\_name)  
   
 cluster\_model\_f <- as.factor(cluster\_model$cluster)  
 pred\_values = table(cluster\_model\_f, classes)  
   
 accuracy\_results = get\_accuracy\_rate(pred\_values, length(classes))  
 if(accuracy\_results > best\_k\_acc)  
 {  
 best\_k\_acc <- accuracy\_results  
 best\_k\_algo <- algo\_name  
 best\_k\_clust <- clusters  
 best\_k\_model <- cluster\_model\_f  
 best\_k\_ds <- name  
 }  
 }  
 }  
}  
  
laptops\_num\_km = dataset\_list[[best\_k\_ds]][,-c(1)]  
print(clusplot(laptops\_num\_km, best\_k\_model, color=TRUE, shade=TRUE, labels=2, lines=0, main=paste("Figure 42 -", best\_k\_algo, "with", best\_k\_clust, "Clusters", sep=" ")))

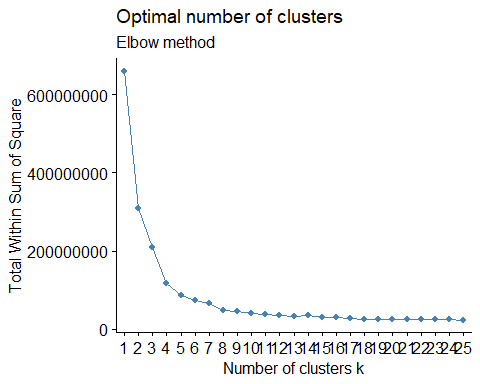


## $Distances  
## NULL  
##   
## $Shading  
## [1] 7.425403 21.613558 3.588364 4.641833 6.816200 10.914642

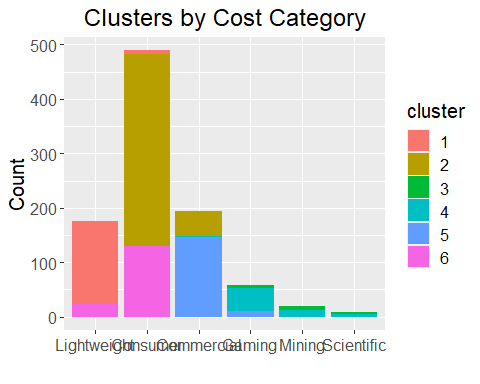
print(paste("The best K-Means Clustering accuracy was", round(best\_k\_acc, digits = 1), "with", best\_k\_clust, "clusters, using", best\_k\_algo, "clustering method and the", best\_k\_ds, "dataset", sep=" "))

## [1] "The best K-Means Clustering accuracy was 57.8 with 6 clusters, using MacQueen clustering method and the numerical dataset"

#Use "elbow method" to find optimal cluster amount  
fviz\_nbclust(laptops\_num\_km, kmeans, method = "wss", k.max = 25) +  
labs(subtitle = "Elbow method")



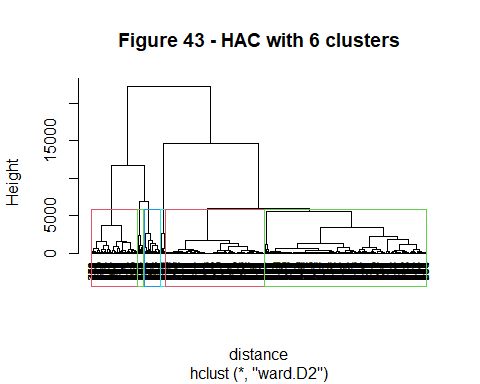
# Create a dataframe for the K Means analysis  
laptops\_km <- laptops\_num  
  
# Add the clusters to laptops\_km  
laptops\_km$cluster <- as.factor(best\_k\_model)  
  
# Create a bar plot of Cost Category and documents, showing the cluster assignment  
ggplot(data=laptops\_km, aes(x=CostCategory, fill=cluster)) +  
 geom\_bar(stat="count") +  
 labs(title = "Clusters by Cost Category", x="", y="Count") +  
 theme(plot.title = element\_text(hjust=0.5), text=element\_text(size=15))



## Hierachical Agglomerative Clustering (HAC)

The HAC will be testing 2 different datasets, 6 different distance functions, 8 different clustering methods and four cluster counts to test a large number of options and find a best fit model.

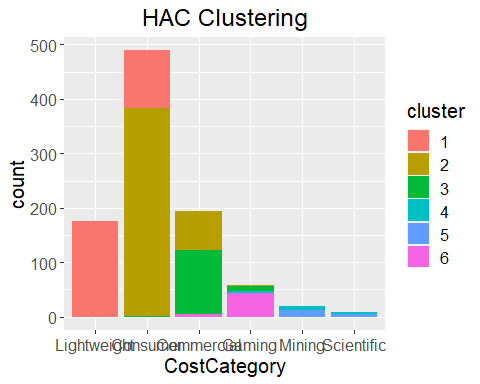
set.seed(23)  
  
dataset\_list <- list("numerical" = laptops\_num, "normalized" = laptops\_norm)  
distance\_methods = c("euclidean", "maximum", "manhattan", "canberra", "binary", "minkowski")  
cluster\_methods = c("ward.D", "ward.D2", "single", "complete", "average", "mcquitty", "median", "centroid")  
cluster\_counts = c(3, 6, 9, 12)  
  
best\_hac\_acc = 0  
best\_hac\_dist = ""  
best\_hac\_clust = ""  
best\_hac\_clust\_count = 0  
best\_hac\_ds = ""  
  
for (name in names(dataset\_list))   
{  
 ds = dataset\_list[[name]]  
 for (distance\_name in distance\_methods)  
 {  
 for (cluster\_name in cluster\_methods)  
 {  
 for(clusters in cluster\_counts)  
 {  
 new\_test\_just\_label = ds[,c(1)]  
 distance = dist(ds[,-c(1)], method = distance\_name)  
 hac = hclust(distance, method=cluster\_name)  
 cut = cutree(hac, clusters)  
 pred\_values = table(cut, new\_test\_just\_label)  
   
 hac\_acc = get\_accuracy\_rate(pred\_values, length(new\_test\_just\_label))  
   
 if(hac\_acc > best\_hac\_acc)  
 {  
 best\_hac\_acc = hac\_acc  
 best\_hac\_dist = distance\_name  
 best\_hac\_clust = cluster\_name  
 best\_hac\_clust\_count = clusters  
 best\_hac = hac  
 best\_hac\_ds = name  
 }  
 }  
 }  
 }  
}  
  
plot(best\_hac, cex=0.6, hang=-1, main=paste("Figure 43 - HAC with", best\_hac\_clust\_count, "clusters", sep=" "))  
rect.hclust(best\_hac, k=best\_hac\_clust\_count, border=2:5)



print(paste("The best Heirarchical Agglomerative Clustering accuracy was", round(best\_hac\_acc, digits = 1), "with", best\_hac\_clust\_count, "clusters, using", best\_hac\_dist, "distance method and", best\_hac\_clust, "clustering method using the", best\_hac\_ds, "dataset", sep=" "))

## [1] "The best Heirarchical Agglomerative Clustering accuracy was 73.5 with 6 clusters, using maximum distance method and ward.D2 clustering method using the numerical dataset"

# Create a dataframe for the HAC analysis  
laptops\_hac <- dataset\_list[[best\_hac\_ds]]  
  
# add the cluster assignment to laptops\_hac  
laptops\_hac$cluster <- as.factor(cutree(best\_hac, 6))  
  
# Create a bar plot of cost and documents, showing the cluster assignment  
ggplot(data=laptops\_hac, aes(x=CostCategory, fill=cluster)) +  
 geom\_bar(stat="count") +  
 labs(title = "HAC Clustering") +  
 theme(plot.title = element\_text(hjust=0.5), text=element\_text(size=15))



## Decision Tree

The Decision Tree uses 6 different control parameters to test to determine which might perform the best. It uses the original, numerical and normalized laptop data sets. The model will also prune the branch with the highest error value prior to making predictions.

set.seed(374)  
  
dataset\_list <- list("original" = laptops, "numerical" = laptops\_num, "normalized" = laptops\_norm)  
control\_list = vector(mode = "list", length = 6)  
control\_list[[1]]= rpart.control(cp=0)  
control\_list[[2]]= rpart.control(cp=0, minsplit = 2, maxdepth = 5)  
control\_list[[3]]= rpart.control(cp=0, minsplit = 2, maxdepth = 10)  
control\_list[[4]]= rpart.control(cp=0.1, minsplit = 2, maxdepth = 5)  
control\_list[[5]]= rpart.control(cp=0.1, minsplit = 3, maxdepth = 5)  
control\_list[[6]]= rpart.control(cp=0.3, minsplit = 5, maxdepth = 10)  
best\_tree\_acc = 0  
best\_tree\_pos = 0  
best\_tree = NULL  
best\_tree\_ds = ""  
for (name in names(dataset\_list))   
{  
 ds = dataset\_list[[name]]  
 for(pos in 1:6)  
 {  
 tree\_acc = train\_tree(ds, control\_list[[pos]])  
   
 if(tree\_acc > best\_tree\_acc)  
 {  
 best\_tree\_acc = tree\_acc  
 best\_tree\_pos = pos  
 best\_tree = tree\_acc  
 best\_tree\_ds = name  
 }  
 }  
}

## Warning in split.default(sample(1:nrow(data\_set)), 1:kfolds): data length is not  
## a multiple of split variable  
  
## Warning in split.default(sample(1:nrow(data\_set)), 1:kfolds): data length is not  
## a multiple of split variable  
  
## Warning in split.default(sample(1:nrow(data\_set)), 1:kfolds): data length is not  
## a multiple of split variable  
  
## Warning in split.default(sample(1:nrow(data\_set)), 1:kfolds): data length is not  
## a multiple of split variable  
  
## Warning in split.default(sample(1:nrow(data\_set)), 1:kfolds): data length is not  
## a multiple of split variable  
  
## Warning in split.default(sample(1:nrow(data\_set)), 1:kfolds): data length is not  
## a multiple of split variable  
  
## Warning in split.default(sample(1:nrow(data\_set)), 1:kfolds): data length is not  
## a multiple of split variable  
  
## Warning in split.default(sample(1:nrow(data\_set)), 1:kfolds): data length is not  
## a multiple of split variable  
  
## Warning in split.default(sample(1:nrow(data\_set)), 1:kfolds): data length is not  
## a multiple of split variable  
  
## Warning in split.default(sample(1:nrow(data\_set)), 1:kfolds): data length is not  
## a multiple of split variable  
  
## Warning in split.default(sample(1:nrow(data\_set)), 1:kfolds): data length is not  
## a multiple of split variable  
  
## Warning in split.default(sample(1:nrow(data\_set)), 1:kfolds): data length is not  
## a multiple of split variable  
  
## Warning in split.default(sample(1:nrow(data\_set)), 1:kfolds): data length is not  
## a multiple of split variable  
  
## Warning in split.default(sample(1:nrow(data\_set)), 1:kfolds): data length is not  
## a multiple of split variable  
  
## Warning in split.default(sample(1:nrow(data\_set)), 1:kfolds): data length is not  
## a multiple of split variable  
  
## Warning in split.default(sample(1:nrow(data\_set)), 1:kfolds): data length is not  
## a multiple of split variable  
  
## Warning in split.default(sample(1:nrow(data\_set)), 1:kfolds): data length is not  
## a multiple of split variable  
  
## Warning in split.default(sample(1:nrow(data\_set)), 1:kfolds): data length is not  
## a multiple of split variable

print(paste("The best Decision Tree accuracy was", round(best\_tree\_acc, digits = 1), "using control", best\_tree\_pos, "and the", best\_tree\_ds, "dataset", sep=" "))

## [1] "The best Decision Tree accuracy was 99.7 using control 2 and the original dataset"

## Naive Bayes

The Naive Bayes model will train against the original, numerical and normalized laptops dataset. The Laplace parameters didn’t appear to fit here considering there weren’t expected to be any categories with no values.

set.seed(111)  
  
dataset\_list <- list("original" = laptops, "numerical" = laptops\_num, "normalized" = laptops\_norm)  
best\_bayes\_acc = 0  
best\_bayes\_ds = ""  
for (name in names(dataset\_list))   
{  
 ds = dataset\_list[[name]]  
 bayes\_acc = train\_bayes(ds)  
 if(bayes\_acc > best\_bayes\_acc)  
 {  
 best\_bayes\_acc = bayes\_acc  
 best\_bayes\_ds = name  
 }  
}

## Warning in split.default(sample(1:nrow(data\_set)), 1:kfolds): data length is not  
## a multiple of split variable  
  
## Warning in split.default(sample(1:nrow(data\_set)), 1:kfolds): data length is not  
## a multiple of split variable  
  
## Warning in split.default(sample(1:nrow(data\_set)), 1:kfolds): data length is not  
## a multiple of split variable

print(paste("The best Naive Bayes accuracy is", round(best\_bayes\_acc, digits = 1), "using the", best\_bayes\_ds, "dataset", sep=" "))

## [1] "The best Naive Bayes accuracy is 90.3 using the normalized dataset"

## KNN

The K-Nearest Neighbor model will use five different K values to train with and will use the numerical and normalized laptops training set.

set.seed(862)  
  
dataset\_list <- list("numerical" = laptops\_num, "normalized" = laptops\_norm)  
k\_vals = c(2,3,5,7,9)  
best\_knn\_acc = 0  
best\_knn\_pos = 0  
best\_knn\_ds = ""  
for (name in names(dataset\_list))   
{  
 ds\_knn = dataset\_list[[name]]  
 ds\_knn$CostCategory = as.numeric(ds\_knn$CostCategory)  
  
 for(num in k\_vals)  
 {  
 knn\_acc = train\_knn(ds\_knn, num)  
   
 if(knn\_acc > best\_knn\_acc)  
 {  
 best\_knn\_acc = knn\_acc  
 best\_knn\_pos = num  
 best\_knn\_ds = name  
 }  
 }  
}

## Warning in split.default(sample(1:nrow(data\_set)), 1:kfolds): data length is not  
## a multiple of split variable  
  
## Warning in split.default(sample(1:nrow(data\_set)), 1:kfolds): data length is not  
## a multiple of split variable  
  
## Warning in split.default(sample(1:nrow(data\_set)), 1:kfolds): data length is not  
## a multiple of split variable  
  
## Warning in split.default(sample(1:nrow(data\_set)), 1:kfolds): data length is not  
## a multiple of split variable  
  
## Warning in split.default(sample(1:nrow(data\_set)), 1:kfolds): data length is not  
## a multiple of split variable  
  
## Warning in split.default(sample(1:nrow(data\_set)), 1:kfolds): data length is not  
## a multiple of split variable  
  
## Warning in split.default(sample(1:nrow(data\_set)), 1:kfolds): data length is not  
## a multiple of split variable  
  
## Warning in split.default(sample(1:nrow(data\_set)), 1:kfolds): data length is not  
## a multiple of split variable  
  
## Warning in split.default(sample(1:nrow(data\_set)), 1:kfolds): data length is not  
## a multiple of split variable  
  
## Warning in split.default(sample(1:nrow(data\_set)), 1:kfolds): data length is not  
## a multiple of split variable

print(paste("The best K-Nearest Neighbor accuracy is", round(best\_knn\_acc, digits = 1), "where k =", best\_knn\_pos, "using the", best\_knn\_ds, "dataset", sep=" "))

## [1] "The best K-Nearest Neighbor accuracy is 100 where k = 2 using the normalized dataset"

## SVM

The Support Vector Machine model will test against 3 kernels and 4 costs parameters. The model will train against the original, numerical and normalized laptop dataset.

set.seed(213)  
  
best\_svm\_acc = 0  
best\_svm\_kernel = ""  
best\_svm\_cost = 0  
best\_svm\_ds = ""  
dataset\_list <- list("original" = laptops, "numerical" = laptops\_num, "normalized" = laptops\_norm)  
kernel\_list = c("radial", "polynomial", "sigmoid")  
cost\_list = c(30, 60, 90, 120)  
for (name in names(dataset\_list))   
{  
 ds = dataset\_list[[name]]  
 for(kernel in kernel\_list)  
 {  
 for(cost in cost\_list)  
 {  
 svm\_acc = train\_svm(ds, kernel, cost)  
   
 if(svm\_acc > best\_svm\_acc)  
 {  
 best\_svm\_acc = svm\_acc  
 best\_svm\_kernel = kernel  
 best\_svm\_cost = cost  
 best\_svm\_ds = name  
 }   
 }  
 }  
}  
  
print(paste("The best Support Vector Machine accuracy is", round(best\_svm\_acc, digits = 1), "using", best\_svm\_kernel, "kernel with cost", best\_svm\_cost, "using the", best\_svm\_ds, "dataset", sep=" "))

## [1] "The best Support Vector Machine accuracy is 94.8 using radial kernel with cost 120 using the original dataset"

## Random Forest

The Random Forest model trains against the original, numerical and normalized laptops training set 3 times each. It also uses the sample with replacement parameter set to true.

set.seed(77)  
  
best\_rf\_acc = 0  
best\_rf\_bin\_acc = 0  
best\_rf\_ds = ""  
dataset\_list <- list("original" = laptops, "numerical" = laptops\_num, "normalized" = laptops\_norm)  
for (name in names(dataset\_list))   
{  
 ds = dataset\_list[[name]]  
 for (select in 1:3)   
 {  
 forest\_acc = train\_forest(laptops\_num, TRUE)  
 best\_rf\_acc = if(forest\_acc > best\_rf\_acc) forest\_acc else best\_rf\_acc  
 best\_rf\_ds = name  
 }   
}

## Warning in split.default(sample(1:nrow(data\_set)), 1:kfolds): data length is not  
## a multiple of split variable  
  
## Warning in split.default(sample(1:nrow(data\_set)), 1:kfolds): data length is not  
## a multiple of split variable  
  
## Warning in split.default(sample(1:nrow(data\_set)), 1:kfolds): data length is not  
## a multiple of split variable  
  
## Warning in split.default(sample(1:nrow(data\_set)), 1:kfolds): data length is not  
## a multiple of split variable  
  
## Warning in split.default(sample(1:nrow(data\_set)), 1:kfolds): data length is not  
## a multiple of split variable  
  
## Warning in split.default(sample(1:nrow(data\_set)), 1:kfolds): data length is not  
## a multiple of split variable  
  
## Warning in split.default(sample(1:nrow(data\_set)), 1:kfolds): data length is not  
## a multiple of split variable  
  
## Warning in split.default(sample(1:nrow(data\_set)), 1:kfolds): data length is not  
## a multiple of split variable  
  
## Warning in split.default(sample(1:nrow(data\_set)), 1:kfolds): data length is not  
## a multiple of split variable

paste("The best Random Forest accuracy is", round(best\_rf\_acc, digits = 1), "using the", best\_rf\_ds, "dataset", sep=" ")

## [1] "The best Random Forest accuracy is 99.4 using the normalized dataset"

## Aggregated Results

The aggregated best model accuracy results were:

* Association Rules Mining:
* Expectation Maximization: 99.9% (normalized dataset)
* K-Means: 57.8% (numerical dataset)
* Heirarchical Agglomerative Clustering: 73.5% (numerical dataset)
* Decision Tree: 99.7% (original dataset)
* Naive Bayes: 87.5% (numerical dataset)
* K-Nearest Neighbor: 99.7% (normalized dataset)
* Support Vector Machine: 93.7% (original dataset)
* Random Forest: 99% (normalized dataset)

# Conclusion

The exploratory data analysis was fairly robust and there were many valuable findings but this analysis was focused on understanding the price. It was clear from numerous plots that RAM size, disk size and CPU core count were highly correlated with price but more interesting was how the ratings had no significant correlation and cost savings was minor. We attributed the ratings to the website not wanting to risk promoting products that wouldn’t sell, making their brand look bad. The majority of prices were under $1000 but there were a few brands that stood out for higher prices like Alienware (really a subcompany of Dell), Apple and Asus. Apple not only had a higher premium but a higher price floor. Many of the laptops with higher prices also had high-end components which was intuitive. The higher end components had curved prices that rose quite quickly for the best parts. This also followed a steep drop off in the number of them in the dataset. This was attributed to it being cost-prohibitive. The site would offer some but it would be risky to purchase and hold that much inventory unless they thought they could sell it. When looking at the processor manufacturer, Intel made most of the processors in use but AMD had a slight price premium. MRP offered an interesting insight. By comparing the difference with the actual price, a cost savings could be determined. This showed a small but significant number of laptops that had a steep over-charge. It was mostly Apple and Lenovo laptops but only Apple had the high value components. The main cost categories showed that the majority of the laptops were in the bottom three categories: Lightweight, Consumer and Commercial. There was a significant group of Gaming laptops but few of the top tier models.

Rules showed … \* common configurations (ARM results) \* what price points they break down into

Designing the dataset had many opportunities to pursue but at each point where you significantly modify the dataset, there’s the possibility the model may be affected by that change. To allow for the discovery of best combinations of dataset and model without overwhelming the time with discovery, we looped through three datasets: the original, a numerical version and a normalized version. Not all models would accept the original dataset but many would. Nearly every model tested produced a high prediction rate. Three of the four top models used the normalized dataset. This was intuitive and expected given that scaling values would put the variables in context and potentially allow the model to better understand the data. There were concerns that such high training results indicated a possible case of overfitting but given we used 5 fold cross-validation to confirm out results and were confident they were valid. We also considered the affect the correlation between predictors like RAM, DiskGB and Cores. Given they were different component parts with different ranges and despite that they were very highly close in correlation value, we thought keeping them distinct may allow more nuance in the learning process.

# References

[Create an Ordered Factor](https://campus.datacamp.com/courses/introduction-to-r-for-finance/factors-4?ex=8)  
[Finding the Intersection Between Two Vectors](https://stackoverflow.com/questions/45271448/r-finding-intersection-between-two-vectors)  
[Dealing with Special Characters In Regex](https://stackoverflow.com/questions/27721008/how-do-i-deal-with-special-characters-like-in-my-regex)  
[MediaTek mt8788](https://www.mediatek.com/products/tablets/mt8788)  
[MediaTek Kompanio 500](https://www.mediatek.com/products/chromebooks/mediatek-kompanio-500)  
[Apple M1 Processor](https://en.wikipedia.org/wiki/Apple_M1)  
[Apple M2 Processor](https://en.wikipedia.org/wiki/Apple_M2)  
[Intel i3 Processor](https://en.wikipedia.org/wiki/List_of_Intel_Core_i3_processors)  
[Intel i5 Processor](https://en.wikipedia.org/wiki/List_of_Intel_Core_i5_processors)  
[Intel i7 Processor](https://en.wikipedia.org/wiki/List_of_Intel_Core_i7_processors)  
[Intel i9 Processor](https://en.wikipedia.org/wiki/List_of_Intel_Core_i9_processors)  
[Count overlapping points](https://ggplot2.tidyverse.org/reference/geom_count.html)  
[making barplot for distribution of a continuous variable](https://stackoverflow.com/questions/43765618/making-barplot-for-distribution-of-a-continuous-variable)  
[Pairs Plot with GGPairs](https://r-charts.com/correlation/ggpairs/)  
[Increase plot size (width) in ggplot2](https://stackoverflow.com/questions/29587881/increase-plot-size-width-in-ggplot2)  
[R apply function with multiple parameters](https://stackoverflow.com/questions/6827299/r-apply-function-with-multiple-parameters)  
[Does the ternary operator exist in R?](https://stackoverflow.com/questions/8790143/does-the-ternary-operator-exist-in-r)  
[How to sum leading diagonal of table in R](https://stackoverflow.com/questions/5648500/how-to-sum-leading-diagonal-of-table-in-r)