PSTAT 134 - Final Project

Jonathan Palada Rosal, Dana Lee

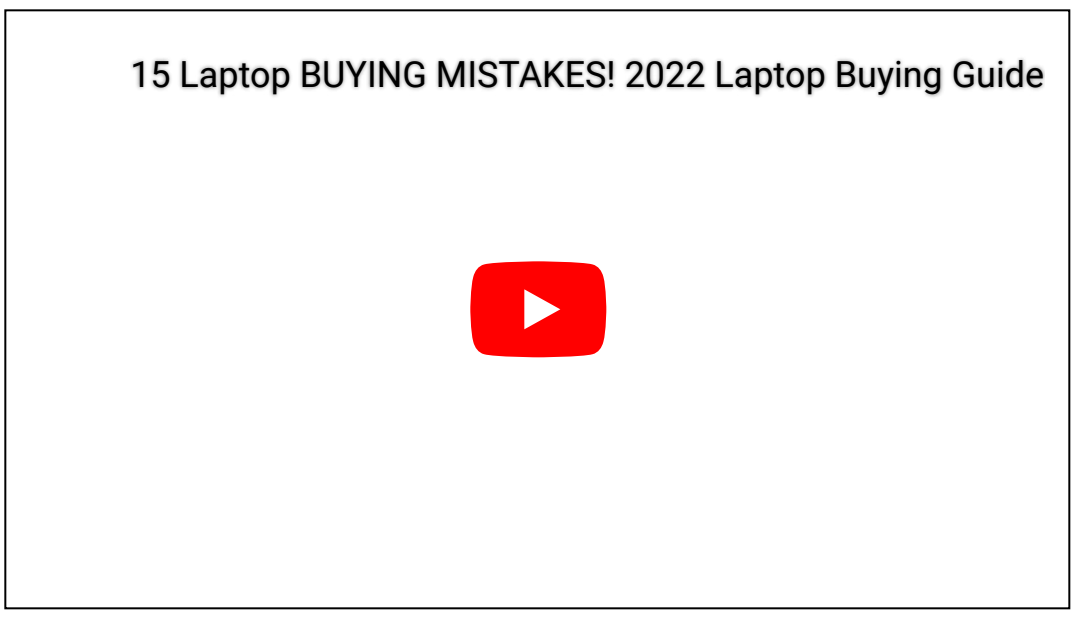
2022-12-06

Introduction

Laptop computers are useful electronic devices that are primarily used for both entertainment and work purposes. A laptop allows you to take your work with you wherever you go, yet it may come with unnecessary features that not everyone may require. For this project, we aimed to create an optimal model that would allow us to forecast a price for a laptop that only includes the features that the customer desires.

Our Dataset

The data we will be using for this project is laptop specs and their prices. (https://www.kaggle.com/datasets/kuchhbhi/latest-laptop-price-list (https://www.kaggle.com/datasets/kuchhbhi/latest-laptop-price-list)) Our dataset is from Kaggle with 23 variables and 1,000 observations.The dataset contains many variables, but we will focus on these 16 factors to make the prediction: Processor Brand, Processor Name, Processor Generation, Ram (GB), Ram Type, SSD (GB), HDD (GB), Operating System, Os_bit, Graphic Card (GB), Weight, Display Size, Warranty, Touchscreen, MSOffice, and Latest Price.



How our model can be helpful

A college student with an essay due every week will undoubtedly require a laptop with different features than a software developer who writes code every day. This model can be useful in assisting customers in selecting the features they want in their laptops and determining estimated prices based on what they require. They can also use the model to see what options are available if they have a specific budget in mind for their laptop.

In order for our model to be helpful, we need it to be accurate. Therefore, our goal will be to get a model that has an accuracy of over 70%. If we accomplish this we believe that our model will be accurate enough to produce helpful predictions to consumers.

Reading/Cleaning the dataset

```
laptop <- read.csv("Cleaned_Laptop_data.csv")
laptop <- laptop %>%
  clean_names()
laptop <- laptop[-19:-23] # removed the variables not needed for the study
laptop <- laptop[-1:-2] # removed the brand and model variable
```

Removed the variables that are not needed for the study.

```
laptop <- subset(laptop, processor_brand != "Missing")
laptop <- subset(laptop, processor_name != "Missing")
laptop <- subset(laptop, processor_gnrtn != "Missing")
laptop <- subset(laptop, ram_gb != "Missing")
laptop <- subset(laptop, ram_type != "Missing")
laptop <- subset(laptop, ssd != "Missing")
laptop <- subset(laptop, hdd != "Missing")
laptop <- subset(laptop, os != "Missing")
laptop <- subset(laptop, os_bit != "Missing")
laptop <- subset(laptop, graphic_card_gb != "Missing")
laptop <- subset(laptop, weight != "Missing")
laptop <- subset(laptop, display_size != "Missing")
laptop <- subset(laptop, warranty != "Missing")
laptop <- subset(laptop, touchscreen != "Missing")
laptop <- subset(laptop, msoffice != "Missing")
laptop <- subset(laptop, latest_price != "Missing") # 415 observations now
laptop$display_size <- as.numeric(laptop$display_size)
laptop$latest_price <- laptop$latest_price/81.41 # Turn the currency into U.S. dollars
```

Deleted any rows with missing information. Also converted the price variable into U.S. dollars.

```
table(laptop$ram_gb)
```

```
##
## 16 GB GB 32 GB GB  4 GB GB  8 GB GB
##      128        3       27      257
```

```
laptop['ram_gb'][laptop['ram_gb'] == '16 GB GB'] <- 16
laptop['ram_gb'][laptop['ram_gb'] == '32 GB GB'] <- 32
laptop['ram_gb'][laptop['ram_gb'] == '4 GB GB'] <- 4
laptop['ram_gb'][laptop['ram_gb'] == '8 GB GB'] <- 8
laptop$ram_gb <- as.numeric(laptop$ram_gb)
```

```
table(laptop$ssd)
```

```
##
##    0 GB 1024 GB  128 GB 2048 GB  256 GB   32 GB  512 GB
##      45      58       3       2      94       1     212
```

```
laptop['ssd'][laptop['ssd'] == '0 GB'] <- 0
laptop['ssd'][laptop['ssd'] == '1024 GB'] <- 1024
laptop['ssd'][laptop['ssd'] == '128 GB'] <- 128
laptop['ssd'][laptop['ssd'] == '2048 GB'] <- 2048
laptop['ssd'][laptop['ssd'] == '256 GB'] <- 256
laptop['ssd'][laptop['ssd'] == '32 GB'] <- 32
laptop['ssd'][laptop['ssd'] == '512 GB'] <- 512
laptop$ssd <- as.numeric(laptop$ssd)
```

```
table(laptop$hdd)
```

```
##
##    0 GB 1024 GB 2048 GB  512 GB
##     333      74       1       7
```

```
laptop['hdd'][laptop['hdd'] == '0 GB'] <- 0
laptop['hdd'][laptop['hdd'] == '1024 GB'] <- 1024
laptop['hdd'][laptop['hdd'] == '2048 GB'] <- 2048
laptop['hdd'][laptop['hdd'] == '512 GB'] <- 512
laptop$hdd <- as.numeric(laptop$hdd)
```

We made the variables that had numbers as values into actual numerical data.

An overview of the dataset

	processor_brand <chr>	processor_name <chr>	processor_gnrtn <chr>	ram_gb <dbl>	ram_type <chr>	s… <dbl>	hdd <dbl>	os <chr>	os_bit <chr>
6	AMD	APU Dual	10th	8	DDR4	256	0	Windows	64-bit
7	AMD	APU Dual	10th	4	DDR4	0	1024	Windows	32-bit
9	AMD	Athlon Dual	10th	32	DDR4	32	0	Windows	32-bit
11	Intel	Core i3	10th	4	DDR4	0	1024	Windows	64-bit
13	Intel	Core i3	10th	4	DDR4	0	1024	Windows	64-bit
18	Intel	Core i5	10th	8	DDR4	0	1024	Windows	32-bit

6 rows | 1-10 of 17 columns

These are the variables we will be using in our dataset:

`processor_brand` : Processor Brand

`processor_name` : Processor Name

`processor_gnrtn` : Processor Generation

`ram_gb` : Random Access Memory in GB

`ram_type` : Random Access Memory type

`ssd` : Solid State Drive in GB

`hdd` : Hard Disk Drive in GB

`os` : Operating system

`os_bit` : Operating system bit

`graphic_card_gb` : Graphic Card in GB

`weight` : Weight of laptop in categories of: Casual, ThinNlight, Gaming

`display_size` : Size in inches

`Warranty` : Years of warranty after purchase

`touchscreen` : If it has a touchscreen feature

`msoffice` : If it has Microsoft office

```
##  processor_brand    processor_name     processor_gnrtn        ram_gb
##  Length:415         Length:415         Length:415         Min.   : 4.00
##  Class :character   Class :character   Class :character   1st Qu.: 8.00
##  Mode  :character   Mode  :character   Mode  :character   Median : 8.00
##                                                           Mean   :10.38
##                                                           3rd Qu.:16.00
##                                                           Max.   :32.00
##    ram_type              ssd              hdd              os
##  Length:415         Min.   :   0.0   Min.   :   0.0   Length:415
##  Class :character   1st Qu.: 256.0   1st Qu.:   0.0   Class :character
##  Mode  :character   Median : 512.0   Median :   0.0   Mode  :character
##                     Mean   : 473.5   Mean   : 196.2
##                     3rd Qu.: 512.0   3rd Qu.:   0.0
##                     Max.   :2048.0   Max.   :2048.0
##     os_bit          graphic_card_gb    weight           display_size
##  Length:415         Min.   :0.000   Length:415         Min.   :12.20
##  Class :character   1st Qu.:0.000   Class :character   1st Qu.:14.00
##  Mode  :character   Median :0.000   Mode  :character   Median :15.60
##                     Mean   :1.171                      Mean   :15.04
##                     3rd Qu.:2.000                      3rd Qu.:15.60
##                     Max.   :8.000                      Max.   :17.30
##     warranty     touchscreen          msoffice          latest_price
##  Min.   :0.000   Length:415         Length:415         Min.   : 256.7
##  1st Qu.:0.000   Class :character   Class :character   1st Qu.: 595.7
##  Median :1.000   Mode  :character   Mode  :character   Median : 805.3
##  Mean   :0.759                                         Mean   :1001.5
##  3rd Qu.:1.000                                         3rd Qu.:1185.2
##  Max.   :3.000                                         Max.   :4242.6
```

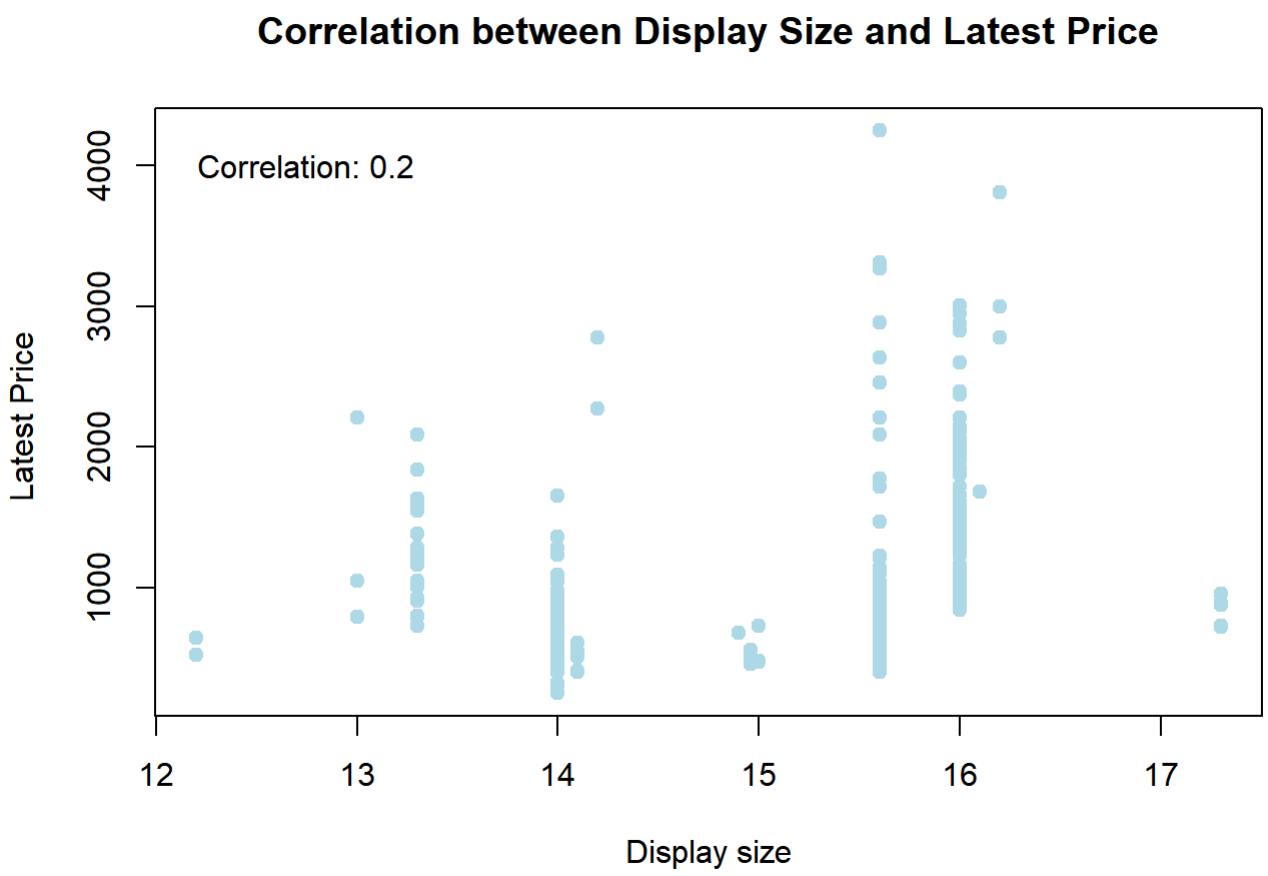
After using the read.csv function to read the dataset, we cleaned the data by removing observations containing missing values. We also removed two variables, brand and model, that were not required for our model, leaving a total of 415 observations with 16 variables. When looking at the summary we can conclude that most our variables are categorical.

Exploratory Data Analysis

Display Size

Fig 1. Size differences between laptops

Display size is one of the most important features that most consumers consider. The display size’s range from 12.2 (inches) - 17.3 (inches). Some consumers may want a 12.2 for easier transportation. Other consumers may want a 17.3 for a bigger screen to see more details in visualizations. On average most students pick a display size between 13 and 15 inches. The perfect in between for students who need easy transport and a decently size screen.



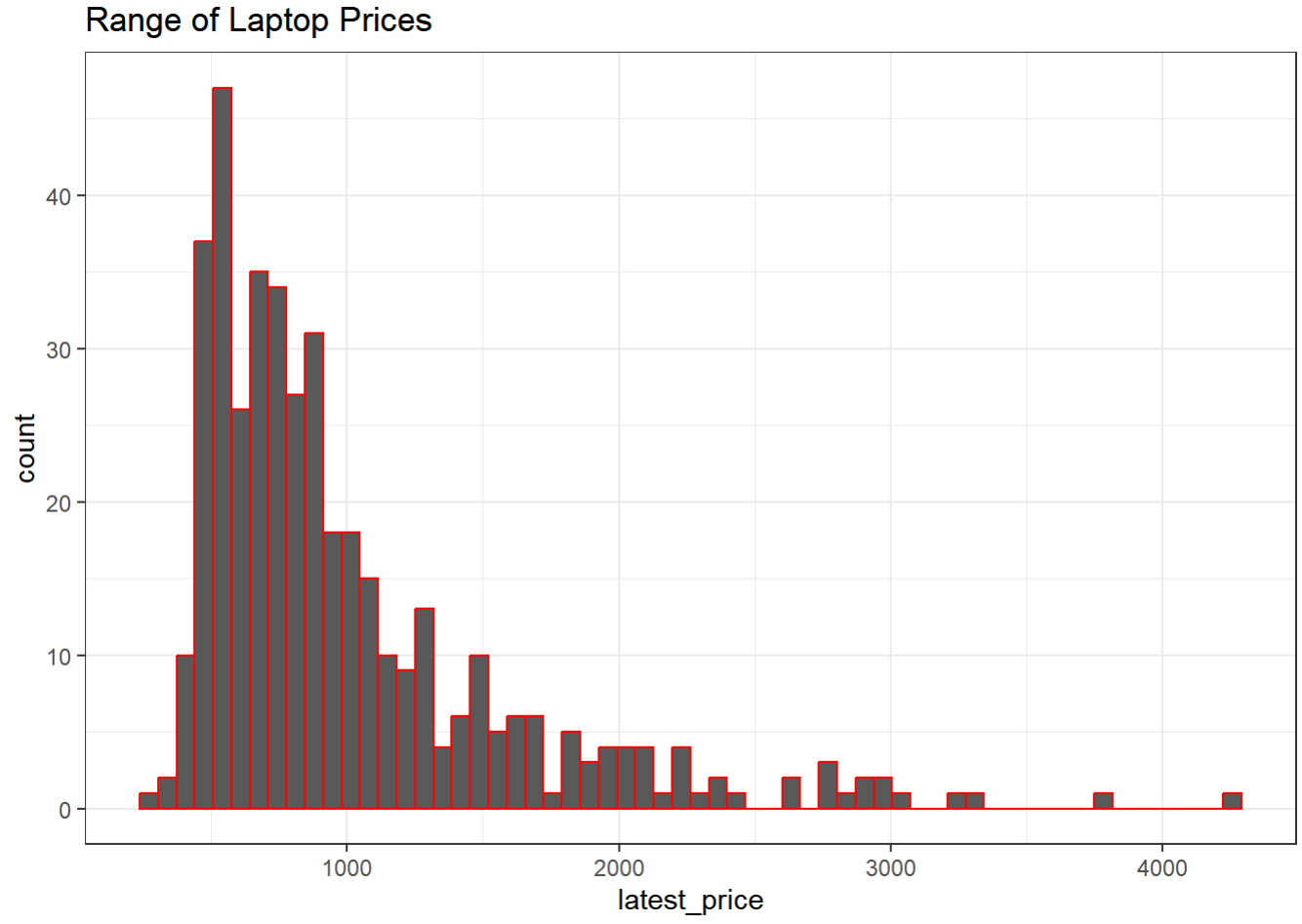
Looking at the plot we can see that there is a 20% positive correlation between the price of the laptop and the display size. Therefore, if a consumer wants a bigger screen, you will most likely pay more for that bigger screen.

Price

	Item	Price	Discount
1	Samsung Chromebook (Wi-Fi, 11.6-Inch)	$244	2%
2	ASUS Transformer Book T100TA	$399	0%
3	Acer C720 Chromebook (11.6-Inch, Haswell micro-architecture, 2GB)	$199	13%
4	ASUS VivoBook X200CA-DB01T 11.6-Inch Touchscreen Laptop	$300	9%
5	Dell Inspiron 15 i15RV-953BLK 15.6-Inch Laptop (Black)	$299	25%
6	Dell Inspiron 15 i15RV-6144BLK 15.6-Inch Touchscreen Laptop	$479	13%
7	ASUS 1015E-DS03 10.1-Inch Laptop	$249	0%
8	Toshiba Satellite C55-A5300 15.6" Laptop PC	$326	7%
9	ASUS Q200E-BSI3T08 11.6-Inch Touchscreen Laptop (Slate Grey)	$420	16%
10	ASUS 1015E-DS01 10.1-Inch Laptop	$259	13%
11	ASUS D450CA-AH21 14-Inch Laptop (Black)	$299	25%
12	Acer C710-2834 11.6-Inch Chromebook	$199	1%

Fig 2. Amazon: Laptop Pricing

When purchasing laptops, one of the most important factors customers consider is price. The laptops users require vary greatly depending on their purpose and budget, and navigating the manufacturers' websites to find the model that perfectly fits can be overwhelming. Therefore, we decided to use the price as our responsive variable in this project in order to focus on selecting the best laptop.



According to the plot, the majority of laptops in our dataset are priced between $250 and $4300. Majority of the laptops range from $400 - $1250. A reason why most laptops are priced within this range is because this range is the typical budget for most buyers. The higher-priced laptops are probably larger and heavier and have far more sophisticated features.

Processor

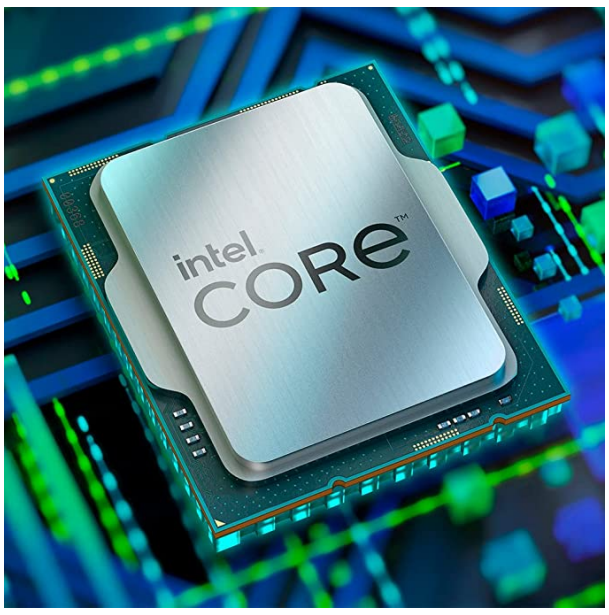


Fig 3. Intel Processor

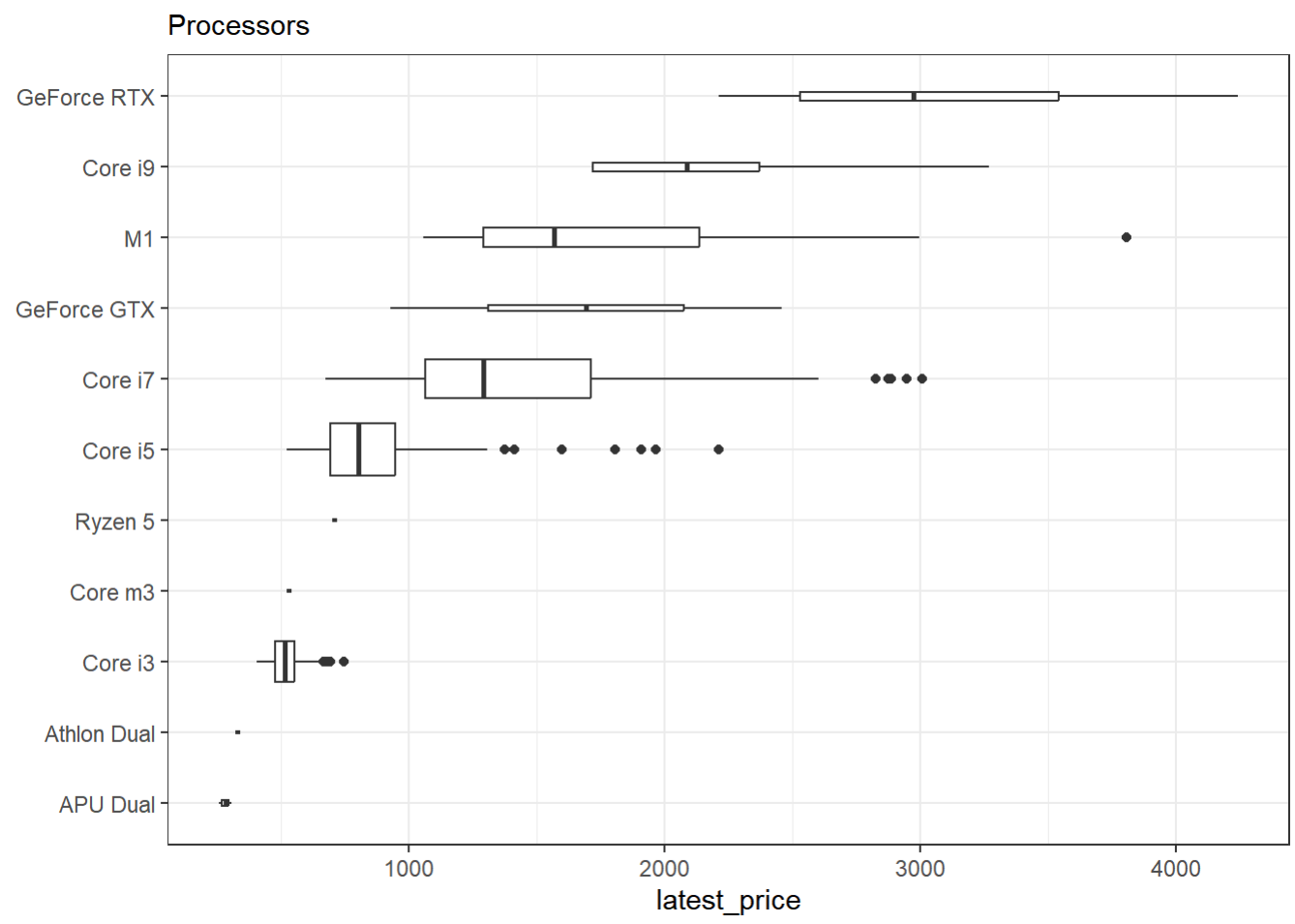
The key element that customers most closely associate with high-performing, quick technology is the computer processor. When comparing computers, the most important factor to consider is the computer processor speed. Making sure the processor works properly is critical to the longevity and functionality of each laptop. Our processor variable contains CPU's, GPU's, APU's and Graphic Cards.

`CPU` - Designed for general purpose tasks, most common applications, and is very compute intensive

`GPU` - Designed for specialized tasks, such as graphics and video, most visual applications, and data processing in parallel.

`APU` - Combining CPU & GPU elements into a single architecture to form a single chip. As a result they consume less power than a CPU + GPU resulting in a better battery life for the laptop.

`Graphic Card` - Contains one or two GPUs, a cooling system, dedicated RAM, and a dedicated source of power. Graphic cards are designed for being in charge of generating images and videos to display then on an output device, such as a monitor/tv.



Observing the box plot created above, the top 3 processors with the highest prices are GeForce RTX, Core i9, and M1. These are the top-ranked processors that are known for their high-speed, and video gamers who require advanced programs use them to deliver realistic graphics with incredibly fast performance or cutting-edge new AI features like NVIDIA DLSS and NVIDIA Broadcast. A customer with a lower budget should purchase a laptop with an APU since it can give a great price to performance ratio. Based on this plot we can see that the faster the processor, the more expensive it is.

Features Significantly Impacting Price

	MacBook	11" MacBook Air	13" MacBook Air	13" MacBook Pro	15" MacBook Pro	17" MacBook Pro
Price	2.4GHz, 250GB **$1,049.00**	1.4GHz, 64GB flash **$1,049.00** 1.4GHz, 128GB flash **$1,249.00**	1.86GHz, 128GB flash **$1,349.00** 1.86GHz, 256GB flash **$1,649.00**	2.3GHz, 320GB **$1,249.00** 2.7GHz, 500GB **$1,549.00**	2.0GHz, 500GB **$1,849.00** 2.2GHz, 750GB **$2,249.00**	2.2GHz, 750GB **$2,499.00**
Display	13.3-inch (viewable) LED-backlit glossy widescreen 1280x800 pixels	11.6-inch (viewable) high-resolution, LED-backlit glossy widescreen 1366x768 pixels	13.3-inch (viewable) high-resolution, LED-backlit glossy widescreen 1440x900 pixels	13.3-inch (viewable) LED-backlit glossy widescreen 1280x800 pixels	15.4-inch (viewable) LED-backlit glossy widescreen 1440x900 resolution Option: 1680x1050 high-resolution glossy or antiglare	17-inch (viewable) high-resolution LED-backlit glossy widescreen 1920x1200 resolution Option: Antiglare
Processor	2.4 GHz Intel Core 2 Duo 1066MHz frontside bus 3MB shared L2 cache	**1.4GHz model:** Intel Core 2 Duo 800MHz frontside bus 3MB shared L2 cache	**1.86GHz model:** Intel Core 2 Duo 1066MHz frontside bus 6MB shared L2 cache	**2.3GHz model:** Dual-core Intel Core i5 3MB shared L3 cache	**2.0GHz model:** Quad-core Intel Core i7 6MB shared L3 cache	**2.2GHz model:** Quad-core Intel Core i7 6MB shared L3 cache

Fig 4. Specifications

We have many different variables in our dataset. On top of those variables, there are many different options that a laptop can have. With so many different options, you can make a huge amount of possible combinations with so many options. Some of these options could increase the price significantly. Therefore we will make a model that can list which features significantly impact the price.

```
##
## Call:
## lm(formula = latest_price ~ ., data = laptop)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -908.48 -128.93   -1.09   93.05 1423.19
##
## Coefficients: (3 not defined because of singularities)
##                             Estimate Std. Error t value Pr(>|t|)
## (Intercept)               -1.567e+03  3.503e+02  -4.473 1.02e-05 ***
## processor_brandIntel       2.272e+03  3.280e+02   6.927 1.82e-11 ***
## processor_brandM1          1.594e+02  2.666e+02   0.598 0.550259
## processor_nameAthlon Dual -8.216e+02  3.649e+02  -2.251 0.024925 *
## processor_nameCore i3     -2.197e+03  2.685e+02  -8.185 4.09e-15 ***
## processor_nameCore i5     -2.081e+03  2.660e+02  -7.821 5.15e-14 ***
## processor_nameCore i7     -1.939e+03  2.644e+02  -7.334 1.34e-12 ***
## processor_nameCore i9     -1.442e+03  2.899e+02  -4.975 9.89e-07 ***
## processor_nameCore m3     -2.010e+03  4.207e+02  -4.778 2.53e-06 ***
## processor_nameGeForce GTX -1.311e+03  2.399e+02  -5.462 8.47e-08 ***
## processor_nameGeForce RTX         NA         NA      NA       NA
## processor_nameM1                  NA         NA      NA       NA
## processor_nameRyzen 5      2.143e+02  3.388e+02   0.632 0.527455
## processor_gnrtn11th       -5.893e+01  3.682e+01  -1.601 0.110294
## processor_gnrtn12th        8.414e+00  2.816e+02   0.030 0.976180
## processor_gnrtn7th         1.227e+02  1.086e+02   1.130 0.259271
## processor_gnrtn8th         2.190e+02  7.512e+01   2.916 0.003756 **
## processor_gnrtn9th         1.320e+02  1.295e+02   1.019 0.308962
## ram_gb                     3.085e+01  4.815e+00   6.408 4.34e-10 ***
## ram_typeDDR4               5.123e+01  1.704e+02   0.301 0.763811
## ram_typeLPDDR3             7.000e+02  1.992e+02   3.514 0.000494 ***
## ram_typeLPDDR4             1.308e+02  2.608e+02   0.502 0.616174
## ram_typeLPDDR4X            1.399e+02  1.775e+02   0.788 0.430983
## ssd                        5.227e-01  7.684e-02   6.802 3.97e-11 ***
## hdd                        8.777e-02  4.475e-02   1.961 0.050590 .
## osMac                      1.087e+03  1.776e+02   6.121 2.31e-09 ***
## osWindows                         NA         NA      NA       NA
## os_bit64-bit              -9.175e+01  5.119e+01  -1.792 0.073856 .
## graphic_card_gb            3.892e+01  1.125e+01   3.459 0.000603 ***
## weightGaming              -2.195e+02  6.986e+01  -3.143 0.001804 **
## weightThinNlight           7.694e+01  3.728e+01   2.064 0.039688 *
## display_size               1.077e+02  1.610e+01   6.687 8.08e-11 ***
## warranty                   3.668e+00  2.724e+01   0.135 0.892959
## touchscreenYes             2.977e+02  4.860e+01   6.124 2.26e-09 ***
## msofficeYes                1.856e+00  3.359e+01   0.055 0.955975
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 275.1 on 383 degrees of freedom
## Multiple R-squared:  0.8054, Adjusted R-squared:  0.7897
## F-statistic: 51.14 on 31 and 383 DF,  p-value: < 2.2e-16
```

Going by a significance level of 0.05, the following features have a significantly impact the price:

`Processor Brand` : AMD, Intel

`Processor Name` : The type of processor is significant to the price

`Processor Generation` : 8th generation is significant to the price

`RAM GB` : RAM is significant to the price

`RAM Type` : DDR3

`SSD GB` : SSD is significant to the price

`HHD GB` : 512

`Operating System` : The Operating System is significant to the price.

`Weight` : The weight is significant to the price.

`Display Size` : All sizes are significant to the price

`Touchscreen` : Having a touchscreen is significant to the price.

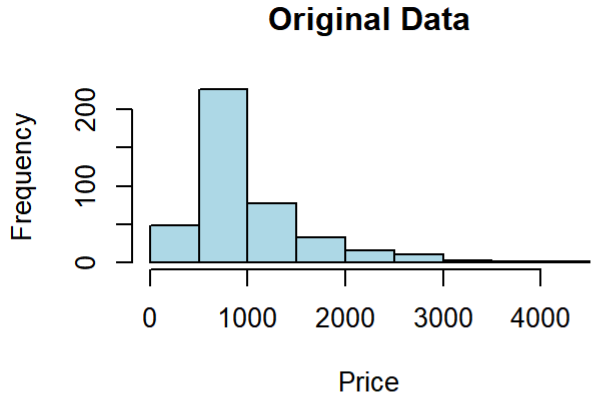
If a consumer was considering any of these features, they will now know that these features could drastically increase or decrease the price of the laptop. These features could be considered a decision-maker for some consumers.

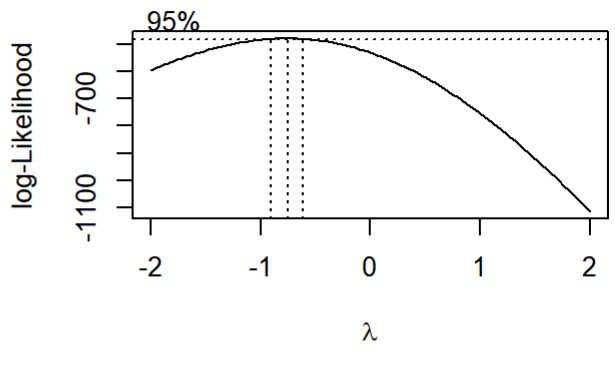
Preparation for modeling

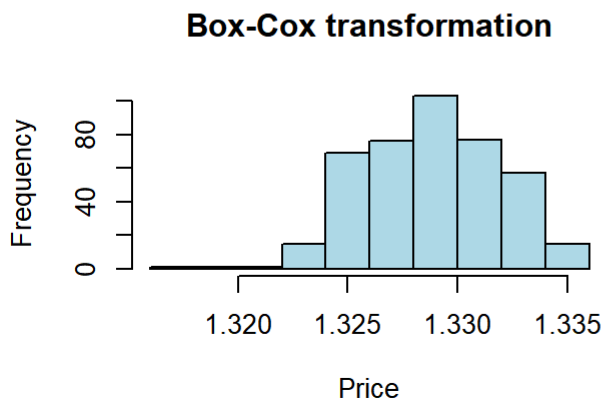
The following models conducted were done in this order and procedure: 1. Building the model 2. Running the model 3. Making predictions using the model

Preparing the data

```
par(mfrow=c(2,2))
laptopbc <- laptop
hist(laptop$latest_price, col="light blue", xlab = "Price", main="Original Data")
bcTransform <- boxcox(laptop$latest_price~ ., data = laptop)
lambda=bcTransform$x[which(bcTransform$y == max(bcTransform$y))]
laptopbc$latest_price <- (1/lambda)*(laptop$latest_price^lambda-1)
hist(laptopbc$latest_price, col="light blue", xlab="Price", main="Box-Cox transformation")  #looks the most symmetric and no
rmally distributed
```







We transformed the `laptop_price` variable by performing a Box-Cox transformation. This transformation should make the data look more like a normal distribution. Looking at the 95% confidence interval for the true λ, we could see that the suggested transformations were $Y_t = \frac{1}{\lambda}(X_t^\lambda - 1)$. When doing a Box-Cox transformation we get a more normal like- distribution.

```
laptop <- laptop %>%
  mutate(processor_brand = factor(processor_brand)) %>%
  mutate(processor_name = factor(processor_name)) %>%
  mutate(processor_gnrtn = factor(processor_gnrtn)) %>%
  mutate(ram_type  = factor(ram_type)) %>%
  mutate(os = factor(os)) %>%
  mutate(os_bit = factor(os_bit)) %>%
  mutate(weight = factor(weight)) %>%
  mutate(touchscreen = factor(touchscreen)) %>%
  mutate(msoffice = factor(msoffice))
head(laptop)
```

	processor_brand	processor_name	processor_gnrtn	ram_gb	ram_type	s…	hdd	os	os_bit
	<fct>	<fct>	<fct>	<dbl>	<fct>	<dbl>	<dbl>	<fct>	<fct>
6	AMD	APU Dual	10th	8	DDR4	256	0	Windows	64-bit
7	AMD	APU Dual	10th	4	DDR4	0	1024	Windows	32-bit
9	AMD	Athlon Dual	10th	32	DDR4	32	0	Windows	32-bit
11	Intel	Core i3	10th	4	DDR4	0	1024	Windows	64-bit
13	Intel	Core i3	10th	4	DDR4	0	1024	Windows	64-bit
18	Intel	Core i5	10th	8	DDR4	0	1024	Windows	32-bit

6 rows | 1-10 of 17 columns

```
laptopbc <- laptopbc %>%
  mutate(processor_brand = factor(processor_brand)) %>%
  mutate(processor_name = factor(processor_name)) %>%
  mutate(processor_gnrtn = factor(processor_gnrtn)) %>%
  mutate(ram_type  = factor(ram_type)) %>%
  mutate(os = factor(os)) %>%
  mutate(os_bit = factor(os_bit)) %>%
  mutate(weight = factor(weight)) %>%
  mutate(touchscreen = factor(touchscreen)) %>%
  mutate(msoffice = factor(msoffice))
head(laptopbc)
```

	processor_brand <fct>	processor_name <fct>	processor_gnrtn <fct>	ram_gb <dbl>	ram_type <fct>	s… <dbl>	hdd <dbl>	os <fct>	os_bit <fct>
6	AMD	APU Dual	10th	8	DDR4	256	0	Windows	64-bit
7	AMD	APU Dual	10th	4	DDR4	0	1024	Windows	32-bit
9	AMD	Athlon Dual	10th	32	DDR4	32	0	Windows	32-bit
11	Intel	Core i3	10th	4	DDR4	0	1024	Windows	64-bit
13	Intel	Core i3	10th	4	DDR4	0	1024	Windows	64-bit
18	Intel	Core i5	10th	8	DDR4	0	1024	Windows	32-bit

6 rows | 1-10 of 17 columns

We mutated the variables by factoring the numerical predictors.

Splitting the data

```
rownames(laptop) <- 1:nrow(laptop) #updating index numbers
laptop
```

	processor_brand <fct>	processor_name <fct>	processor_gnrtn <fct>	ram_gb <dbl>	ram_type <fct>	s… <dbl>	hdd <dbl>	os <fct>	os_bit <fct>
1	AMD	APU Dual	10th	8	DDR4	256	0	Windows	64-bit
2	AMD	APU Dual	10th	4	DDR4	0	1024	Windows	32-bit
3	AMD	Athlon Dual	10th	32	DDR4	32	0	Windows	32-bit
4	Intel	Core i3	10th	4	DDR4	0	1024	Windows	64-bit
5	Intel	Core i3	10th	4	DDR4	0	1024	Windows	64-bit
6	Intel	Core i5	10th	8	DDR4	0	1024	Windows	32-bit
7	Intel	Core i5	10th	4	DDR4	0	1024	Windows	32-bit
8	Intel	Core i3	11th	8	DDR4	256	0	Windows	64-bit
9	Intel	Core i5	11th	8	DDR4	256	0	Windows	64-bit
10	Intel	Core i3	7th	4	DDR4	0	1024	Windows	64-bit

1-10 of 415 rows | 1-10 of 17 columns Previous **1** 2 3 4 5 6 … 42 Next

```
rownames(laptopbc) <- 1:nrow(laptopbc) # updating index numbers
laptopbc
```

	processor_brand <fct>	processor_name <fct>	processor_gnrtn <fct>	ram_gb <dbl>	ram_type <fct>	s… <dbl>	hdd <dbl>	os <fct>	os_bit <fct>
1	AMD	APU Dual	10th	8	DDR4	256	0	Windows	64-bit
2	AMD	APU Dual	10th	4	DDR4	0	1024	Windows	32-bit
3	AMD	Athlon Dual	10th	32	DDR4	32	0	Windows	32-bit
4	Intel	Core i3	10th	4	DDR4	0	1024	Windows	64-bit
5	Intel	Core i3	10th	4	DDR4	0	1024	Windows	64-bit
6	Intel	Core i5	10th	8	DDR4	0	1024	Windows	32-bit
7	Intel	Core i5	10th	4	DDR4	0	1024	Windows	32-bit
8	Intel	Core i3	11th	8	DDR4	256	0	Windows	64-bit
9	Intel	Core i5	11th	8	DDR4	256	0	Windows	64-bit
10	Intel	Core i3	7th	4	DDR4	0	1024	Windows	64-bit

1-10 of 415 rows | 1-10 of 17 columns Previous **1** 2 3 4 5 6 … 42 Next

```
set.seed(12)
ec_split <- laptop %>%
  initial_split(prop = 0.80, strata = "latest_price")

ec_train <- training(ec_split)
ec_test <- testing(ec_split)
dim(ec_train) #331 obs. 16 columns
```

```
## [1] 331  16
```

```
dim(ec_test) #84 obs. 16 columns
```

```
## [1] 84 16
```

```
ec_trainbc <- ec_train
ec_trainbc$latest_price <- (1/lambda)*(ec_trainbc$latest_price^lambda-1)
ec_testbc <- ec_test
ec_testbc$latest_price <- (1/lambda)*(ec_testbc$latest_price^lambda-1)
```

We split the data into 80% training and 20% testing as we felt that would be the best way to approach training and testing our models. We had to make two different splits. For ridge and lasso, the engines we use for those models already find the most optimal lambda and perform a transformation while predicting. For the boost and tree models we had to perform the transformations manually.

Making the recipe and folds

```
ec_recipe <- recipe(latest_price ~ processor_brand + processor_name + processor_gnrtn + ram_gb + ram_type + ssd + hdd + os +
os_bit + graphic_card_gb + weight + display_size + warranty + touchscreen + msoffice, data = ec_train) %>%
  step_dummy(all_nominal_predictors()) %>%
  step_zv(all_nominal_predictors()) %>%
  step_normalize(all_predictors()) %>%
  step_novel(all_nominal_predictors())


ec_folds <- vfold_cv(ec_train, strata = latest_price, v = 10, repeats = 5)

ec_recipebc <- recipe(latest_price ~ processor_brand + processor_name + processor_gnrtn + ram_gb + ram_type + ssd + hdd + os
+ os_bit + graphic_card_gb + weight + display_size + warranty + touchscreen + msoffice, data = ec_trainbc) %>%
  step_dummy(all_nominal_predictors()) %>%
  step_normalize(all_predictors()) %>%
  step_novel(all_nominal_predictors()) %>%
  step_zv(all_nominal_predictors())


ec_foldsbc <- vfold_cv(ec_trainbc, strata = latest_price, v = 10, repeats = 5)
```

We made a recipe using the training set. The predictor variables we left out of the recipe are brand and model. We decided to leave them out because we wanted the prediction to be completely based on the features of the laptop. We `step_dummy()` all nominal predictors to encode them as categorical predictors. We also `step_normalize()` to center and scale all the predictors. We `step_novel` and `step_zv` all nominal predictors so it would assign any previously unseen factor level to a new value and to remove any variables that contain only a single value.

The models

Ridge Regression

The first model we decided to create was a Ridge Regression Model. Ridge regression is one of the alternative approaches to modeling. Ridge is one of the main types of the Regularization approach. The goal of the Regularization approach is to shrink the coefficient estimates toward zero, similar to least squares. Ridge minimizes the sum of squared residuals and $\lambda * slope^2$.

```
ridge_spec <- linear_reg(penalty = tune(), mixture = 0) %>%
  set_mode("regression") %>%
  set_engine("glmnet")

ridge_workflow <- workflow() %>%
  add_recipe(ec_recipe) %>%
  add_model(ridge_spec)

set.seed(12)

penalty_grid <- grid_regular(penalty(range = c(1, 11)), levels = 50)
penalty_grid
```

penalty <dbl>
1.000000e+01
1.599859e+01
2.559548e+01
4.094915e+01
6.551286e+01
1.048113e+02
1.676833e+02
2.682696e+02
4.291934e+02
6.866488e+02

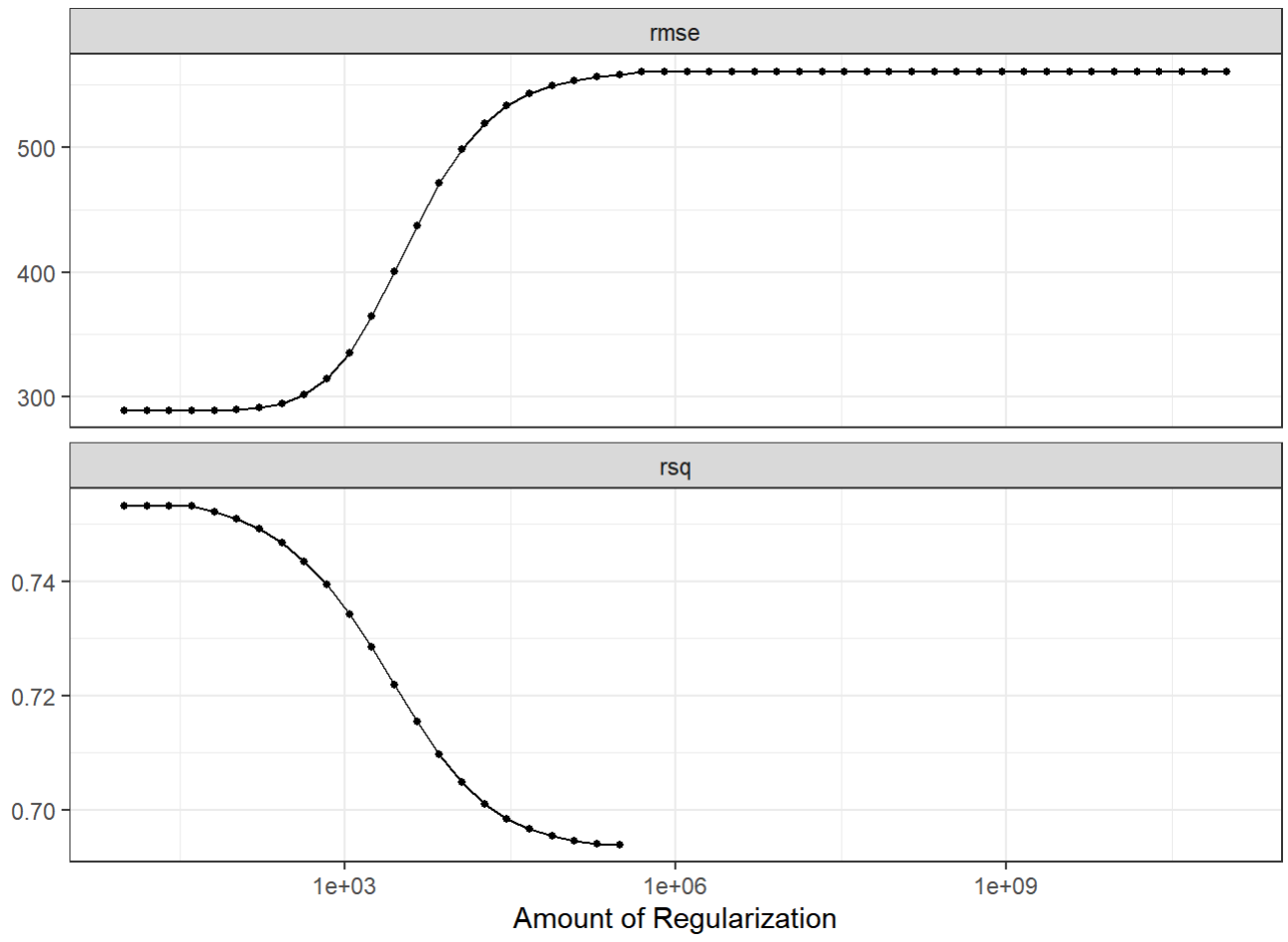
1-10 of 50 rows Previous **1** 2 3 4 5 Next

```
tune_res <- tune_grid(
  ridge_workflow,
  resamples = ec_folds,
  grid = penalty_grid
)
tune_res
```

splits	id	id2	.metrics	.notes
<list>	<chr>	<chr>	<list>	<list>
<S3: vfold_split>	Repeat1	Fold01	<tibble[,5]>	<tibble[,3]>
<S3: vfold_split>	Repeat1	Fold02	<tibble[,5]>	<tibble[,3]>
<S3: vfold_split>	Repeat1	Fold03	<tibble[,5]>	<tibble[,3]>
<S3: vfold_split>	Repeat1	Fold04	<tibble[,5]>	<tibble[,3]>
<S3: vfold_split>	Repeat1	Fold05	<tibble[,5]>	<tibble[,3]>
<S3: vfold_split>	Repeat1	Fold06	<tibble[,5]>	<tibble[,3]>
<S3: vfold_split>	Repeat1	Fold07	<tibble[,5]>	<tibble[,3]>
<S3: vfold_split>	Repeat1	Fold08	<tibble[,5]>	<tibble[,3]>
<S3: vfold_split>	Repeat1	Fold09	<tibble[,5]>	<tibble[,3]>
<S3: vfold_split>	Repeat1	Fold10	<tibble[,5]>	<tibble[,3]>

1-10 of 50 rows Previous **1** 2 3 4 5 Next

```
autoplot(tune_res)
```



In this step we are adding the recipe to the ridge model. We are also making the workflow and grid for the `tune_grid` . We use the folds we did earlier for the `tune_grid` also. We are using the glmnet engine so it will perform the most optimal transformation for the model.

```
Ridge_RMSE <- collect_metrics(tune_res) %>%
  dplyr::select(.metric, mean, std_err) %>%
  head()
```

We collect the metrics of our regression tune and look at the mean and standard error.

```
best_penalty <- select_best(tune_res, metric = "rsq")
best_penalty
```

penalty	.config
<dbl>	<chr>
10	Preprocessor1_Model01

1 row

```
ridge_final <- finalize_workflow(ridge_workflow, best_penalty)
ridge_final_fit <- fit(ridge_final, data = ec_train)

Ridge_Prediction <- predict(ridge_final_fit, new_data = ec_test %>% dplyr::select(-latest_price))
Ridge_Prediction <- bind_cols(Ridge_Prediction, ec_test %>% dplyr::select(latest_price))
Ridge_Prediction <- (1/lambda)*(Ridge_Prediction^lambda-1) #To make the values the same as the boost and tree models


Ridge_Graph <- Ridge_Prediction %>%
  ggplot(aes(x=.pred, y=latest_price)) + geom_point(alpha = 1) + geom_abline(lty = 2) + theme_bw() + coord_obs_pred()

Ridge_Accuracy <- augment(ridge_final_fit, new_data = ec_test) %>%
  rsq(truth = latest_price, estimate = .pred)
```

Here we prepare the predictions, graphs, and plots, for comparison at the end. We converted the predictions and actual values into the same scale as the boxcox data so it will be easier to compare to the boost and tree models.

Lasso Regression

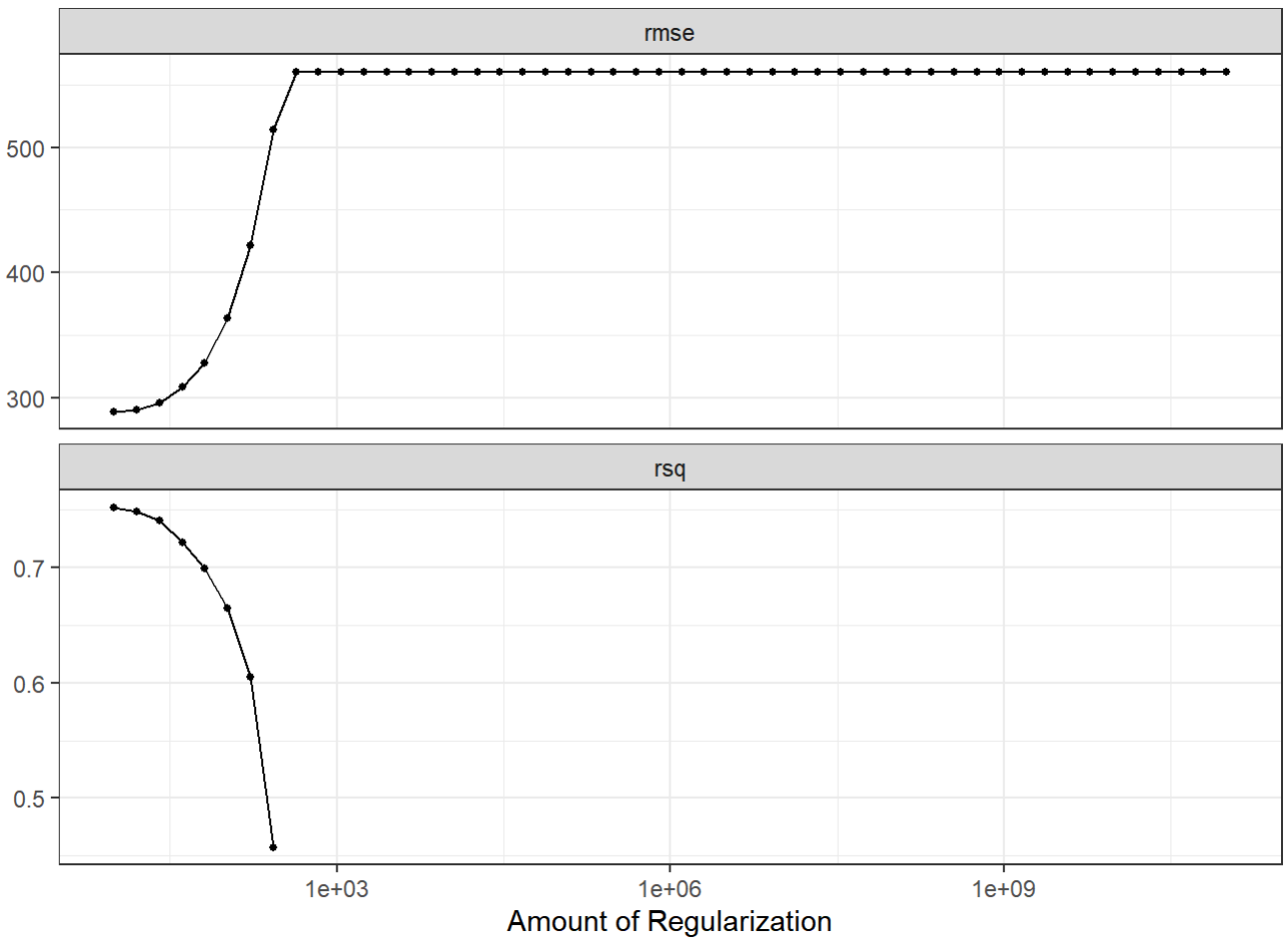
The second model we decided to create was a Lasso Regression Model. Lasso regression is also one of the alternative approaches to modeling. Like Ridge, Lasso is one of the main types of the Regularization approach.The difference is Lasso minimizes the sum of squared residuals and $\lambda * |slope|$.

```
lasso_spec <-
  linear_reg(penalty = tune(), mixture = 1) %>%
  set_mode("regression") %>%
  set_engine("glmnet")

lasso_workflow <- workflow() %>%
  add_recipe(ec_recipe) %>%
  add_model(lasso_spec)

set.seed(12)

tune_res_lasso <- tune_grid(
  lasso_workflow,
  resamples = ec_folds,
  grid = penalty_grid
)
tune_res_lasso
```

splits <list>	id <chr>	id2 <chr>	.metrics <list>	.notes <list>
<S3: vfold_split>	Repeat1	Fold01	<tibble[,5]>	<tibble[,3]>
<S3: vfold_split>	Repeat1	Fold02	<tibble[,5]>	<tibble[,3]>
<S3: vfold_split>	Repeat1	Fold03	<tibble[,5]>	<tibble[,3]>
<S3: vfold_split>	Repeat1	Fold04	<tibble[,5]>	<tibble[,3]>
<S3: vfold_split>	Repeat1	Fold05	<tibble[,5]>	<tibble[,3]>
<S3: vfold_split>	Repeat1	Fold06	<tibble[,5]>	<tibble[,3]>
<S3: vfold_split>	Repeat1	Fold07	<tibble[,5]>	<tibble[,3]>
<S3: vfold_split>	Repeat1	Fold08	<tibble[,5]>	<tibble[,3]>
<S3: vfold_split>	Repeat1	Fold09	<tibble[,5]>	<tibble[,3]>
<S3: vfold_split>	Repeat1	Fold10	<tibble[,5]>	<tibble[,3]>

1-10 of 50 rows Previous **1** 2 3 4 5 Next

```
autoplot(tune_res_lasso)
```



In this step we are adding the recipe to the lasso model. We are also making the workflow and grid for the `tune_grid` . We use the folds we did earlier for the `tune_grid` also. The plots seem to increase/decrease quickly compared to the Ridge Model. We are also using the glmnet engine here as well.

```
Lasso_RMSE <- collect_metrics(tune_res_lasso) %>%
  dplyr::select(.metric, mean, std_err) %>%
  head()
```

We collect the metrics of our regression tune and look at the mean and standard error.

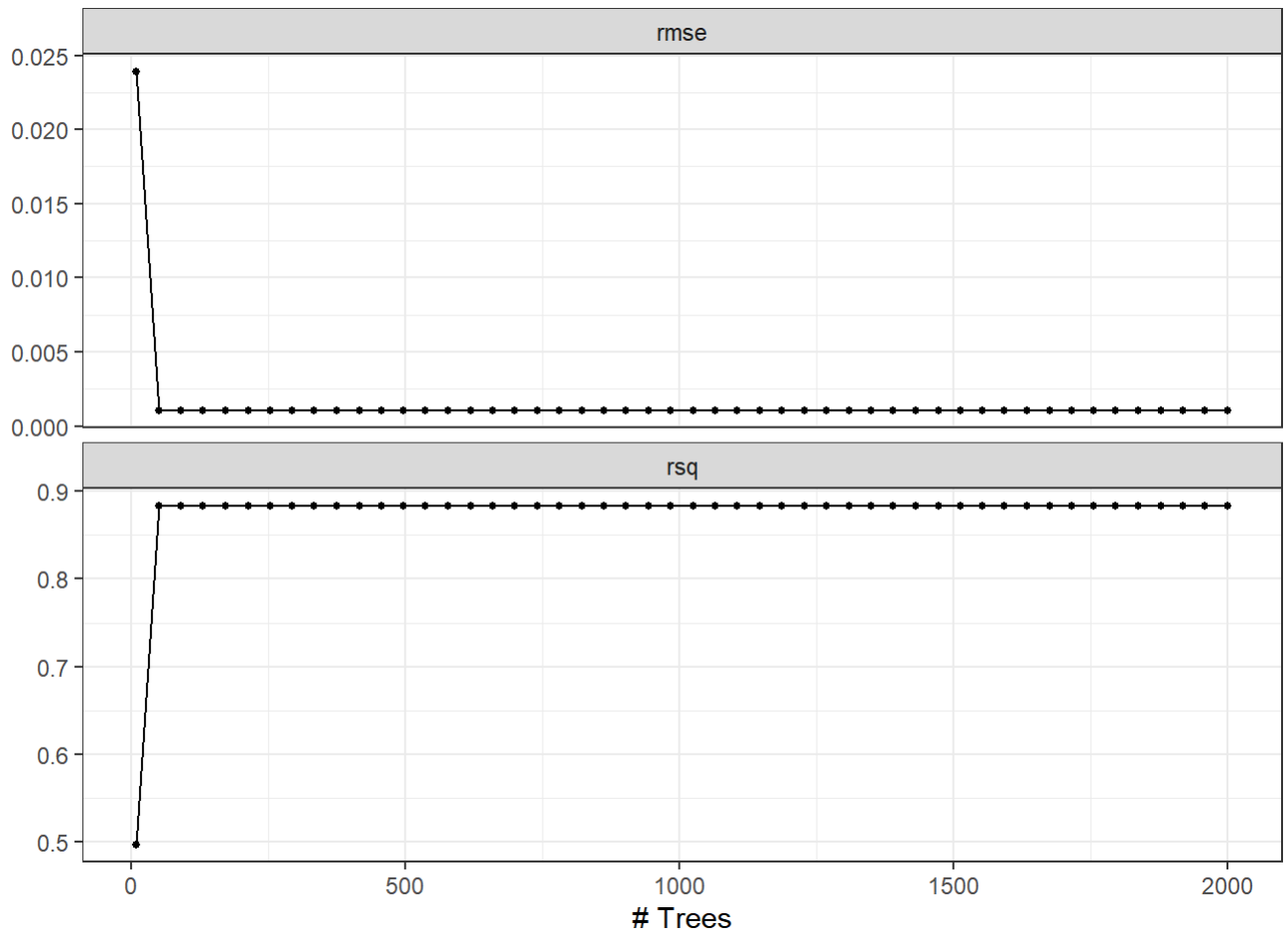
```
best_penalty_lasso <- select_best(tune_res_lasso, metric = "rsq")

lasso_final <- finalize_workflow(lasso_workflow, best_penalty_lasso)
lasso_final_fit <- fit(lasso_final, data = ec_train)

Lasso_Prediction <- predict(lasso_final_fit, new_data = ec_test %>% dplyr::select(-latest_price))
Lasso_Prediction <- bind_cols(Lasso_Prediction, ec_test %>% dplyr::select(latest_price))
Lasso_Prediction <- (1/lambda)*(Lasso_Prediction^lambda-1) #To make the values the same as the boost and tree models

Lasso_Graph <- Lasso_Prediction %>%
  ggplot(aes(x=.pred, y=latest_price)) + geom_point(alpha=1) + geom_abline(lty = 2) + theme_bw() + coord_obs_pred()

Lasso_Accuracy <- augment(lasso_final_fit, new_data = ec_test) %>%
  rsq(truth = latest_price, estimate = .pred)
```

Here we prepare the predictions, graphs, and plots, for comparison at the end. We also converted the predictions and actual values into the same scale as the boxcox data so it will be easier to compare to the boost and tree models.

Boosted Model

The third model we created was a boosted tree model. A boosted model builds a weak decision tree that has low predictive accuracy. Then the model goes through the process of sequentially improving previous decision trees. Doing this, slowly reduces the bias at each step without drastically increasing the variance.

```
boost_spec <- boost_tree() %>%
  set_engine("xgboost") %>%
  set_mode("regression")

boost_wf <- workflow() %>%
  add_model(boost_spec %>%
  set_args(trees = tune())) %>%
  add_recipe(ec_recipebc)

set.seed(12)

boost_grid <- grid_regular(trees(range = c(10, 2000)), levels = 50)

boost_tune_res <- tune_grid(
  boost_wf,
  resamples = ec_foldsbc,
  grid = boost_grid,
)

autoplot(boost_tune_res)
```



In this step we are adding the recipe to the Boost model. We are also making the workflow and grid for the `tune_grid` . We use the folds we did earlier for the `tune_grid` also. The boost rmse plot decreases significantly but then levels out around the 60th tree. The rsq does the exact same thing but increases instead.

```
Boost_RMSE <- collect_metrics(boost_tune_res) %>%
  dplyr::select(.metric, mean, std_err) %>%
  head()
```

We collect the metrics of our regression tune and look at the mean and standard error.

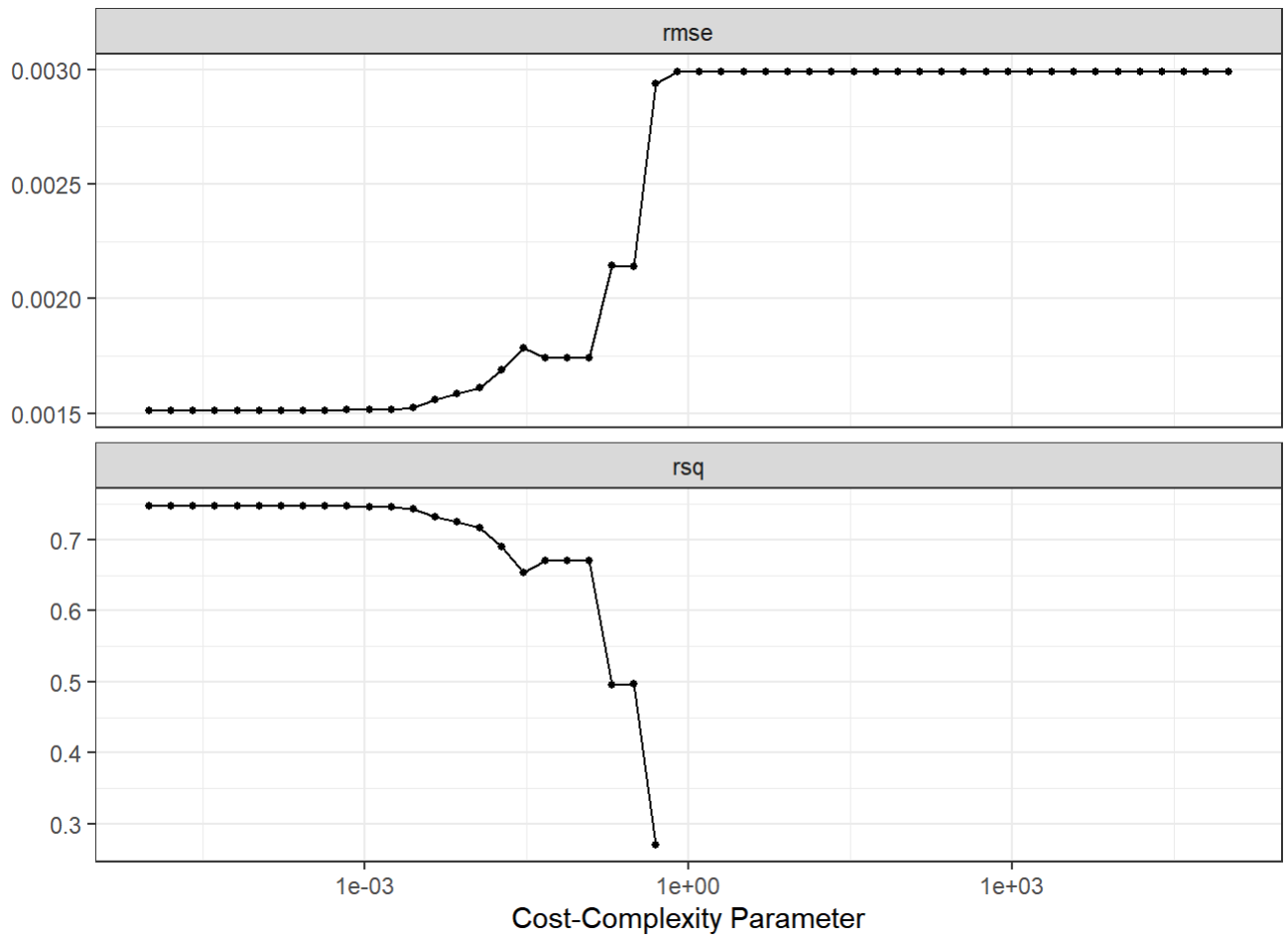
```
best_boost_final <- select_best(boost_tune_res)
best_boost_final_model <- finalize_workflow(boost_wf, best_boost_final)
best_boost_final_model_fit <- fit(best_boost_final_model, data = ec_trainbc)

Boost_Prediction <- predict(best_boost_final_model_fit, new_data = ec_testbc %>% dplyr::select(-latest_price))
Boost_Prediction <- bind_cols(Boost_Prediction, ec_testbc %>% dplyr::select(latest_price))

Boost_Graph <- Boost_Prediction %>%
  ggplot(aes(x=.pred, y=latest_price)) + geom_point(alpha=1) + geom_abline(lty = 2) + theme_bw() + coord_obs_pred()

Boost_Accuracy <- augment(best_boost_final_model_fit, new_data = ec_testbc) %>%
  rsq(truth = latest_price, estimate = .pred)
```

Here we prepare the predictions, graphs, and plots, for comparison at the end.

Decision - Tree model

The fourth and final model we decided to make is a decision tree model. A decision tree model puts the data into classified chunks. Then based on the data from those chunks, the model does it’s best to predict the outcome.

```
tree_spec <-decision_tree() %>%
  set_engine("rpart")

class_tree_spec <- tree_spec %>%
  set_mode("regression")

class_tree_wf <- workflow() %>%
  add_model(class_tree_spec %>% set_args(cost_complexity = tune())) %>%
  add_recipe(ec_recipebc)

set.seed(12)

param_grid <- grid_regular(cost_complexity(range = c(-5, 5)), levels = 50)

tune_res_tree <- tune_grid(
  class_tree_wf,
  resamples = ec_foldsbc,
  grid = param_grid,
)

autoplot(tune_res_tree)
```

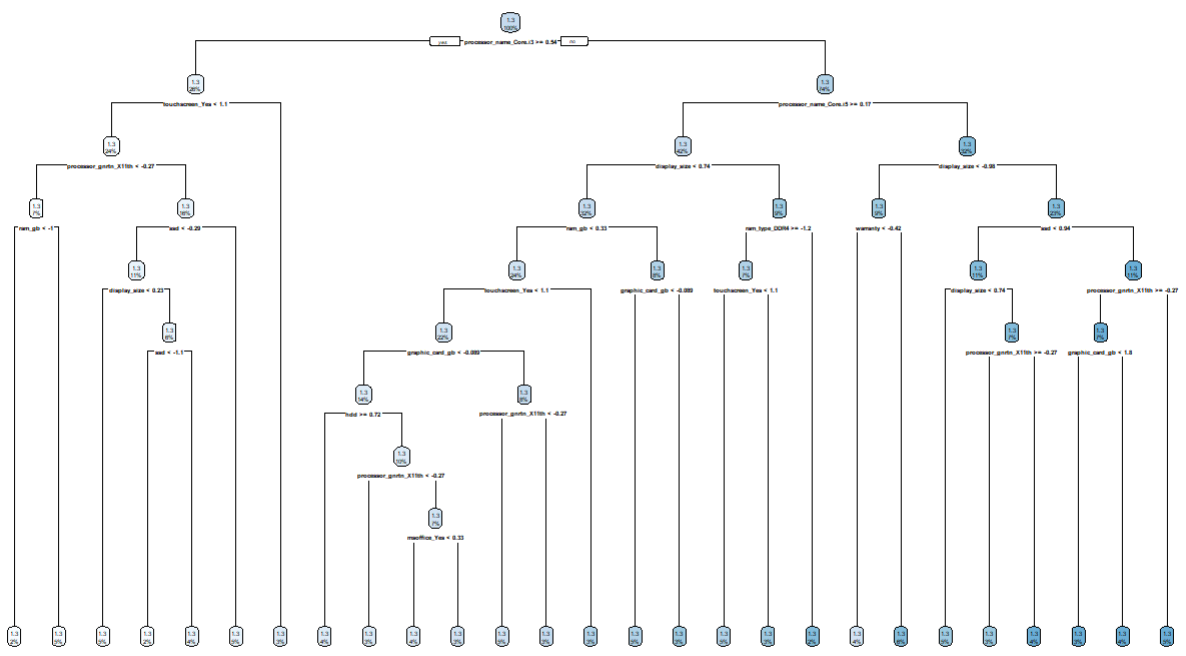


In this step we are adding the recipe to the Tree model. We are also making the workflow and grid for the `tune_grid` . We use the folds we did earlier for the `tune_grid` also. The `cost_complexity` parameter seems to have more similarities to the ridge plots. The difference is the plots are not smooth, and seem to have sudden changes of slope.

```
Tree_RMSE <- collect_metrics(tune_res_tree) %>%
  dplyr::select(.metric, mean, std_err) %>%
  head()
```

We collect the metrics of our regression tune and look at the mean and standard error.

```
best_complexity <- select_best(tune_res_tree)
class_tree_final <- finalize_workflow(class_tree_wf, best_complexity)
class_tree_final_fit <- fit(class_tree_final, data = ec_trainbc)

class_tree_final_fit %>%
  extract_fit_engine() %>%
  rpart.plot()
```



The tree plot asks specific questions. These questions can only be answered yes or no.

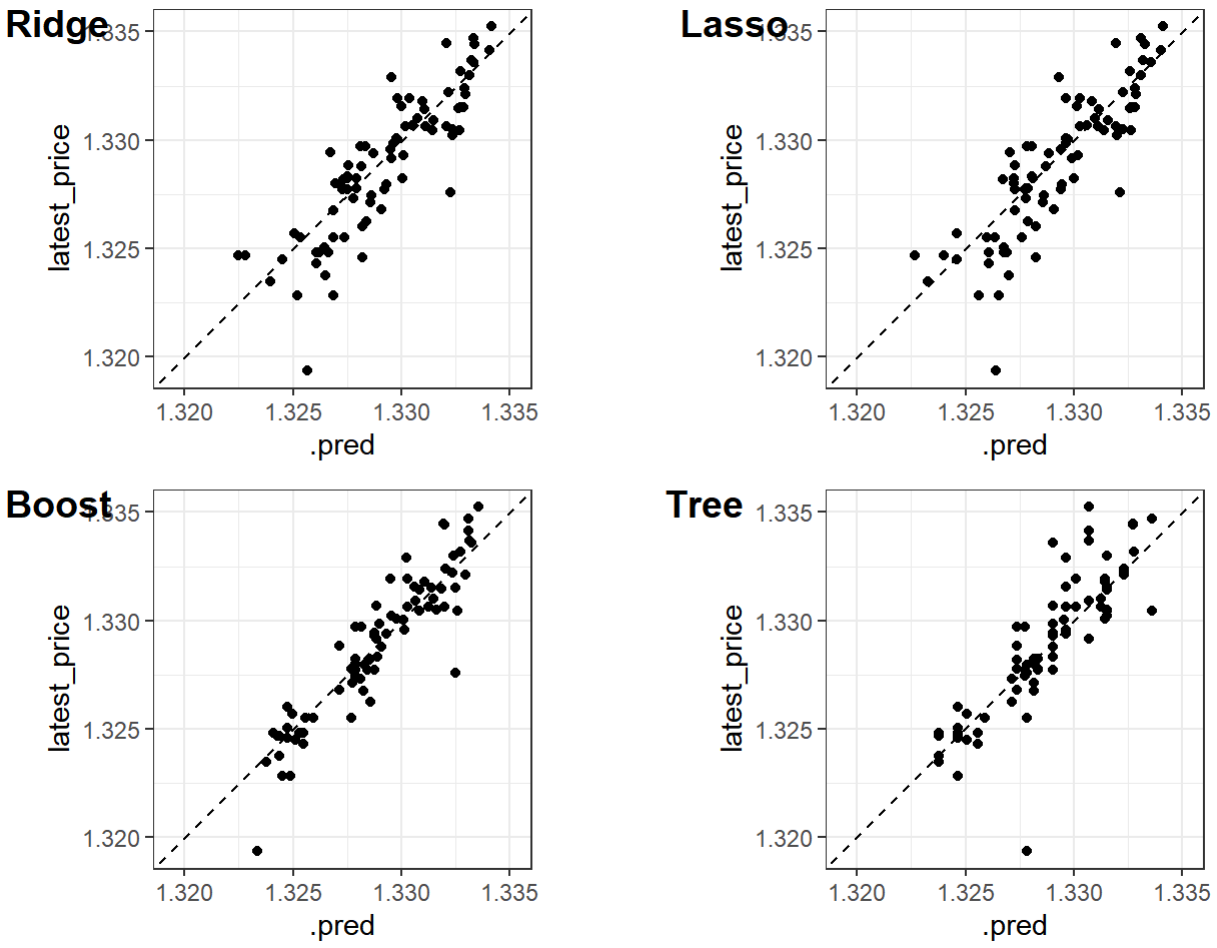
```
Tree_Prediction <- predict(class_tree_final_fit, new_data = ec_testbc %>% dplyr::select(-latest_price))
Tree_Prediction <- bind_cols(Tree_Prediction, ec_testbc %>% dplyr::select(latest_price))

Tree_Graph <- Tree_Prediction %>%
  ggplot(aes(x=.pred, y=latest_price)) + geom_point(alpha=1) + geom_abline(lty = 2) + theme_bw() + coord_obs_pred()

Tree_Accuracy <- augment(class_tree_final_fit, new_data = ec_testbc) %>%
  rsq(truth = latest_price, estimate = .pred)
```

Here we prepare the predictions, graphs, and plots, for comparison at the end.

Results

Comparison of the four different models: We will compare the four different models in this by these factors: - Prediction Graphs - RSQ (R-Squared) from Training Set - RSQ from Testing Set

Graphs



In the plots the dotted line represents where the points would be if the actual price of the laptop was the same number as the prediction. Looking at the plots I would say that the Boost has the points closest to the dotted line meaning they most likely have the highest accuracy between the four models.

RSQ (Training Set)

Ridge

.metric <chr>	mean <dbl>	std_err <dbl>
rmse	288.8631752	9.27203998
rsq	0.7532238	0.01451216
rmse	288.8631752	9.27203998
rsq	0.7532238	0.01451216
rmse	288.8631752	9.27203998
rsq	0.7532238	0.01451216

6 rows

Looking at the mean and standard we get that Ridge has the following values: `RSQ` : mean = 0.753 & standard error = 0.0145

Lasso

.metric <chr>	mean <dbl>	std_err <dbl>
rmse	288.3384497	9.16373032
rsq	0.7522212	0.01370918
rmse	290.3689986	9.20037627
rsq	0.7487383	0.01336527
rmse	295.5909254	9.62396154
rsq	0.7407803	0.01341724

6 rows

Looking at the mean and standard we get that Lasso has the following values: `RSQ` : mean = 0.752 & standard error = 0.0137

Boost

.metric <chr>	mean <dbl>	std_err <dbl>
rmse	0.023923079	2.369457e-05

.metric <chr>	mean <dbl>	std_err <dbl>
rsq	0.496409257	1.706591e-02
rmse	0.001024014	4.998616e-05
rsq	0.883731218	9.778664e-03
rmse	0.001024014	4.998616e-05
rsq	0.883731218	9.778664e-03

6 rows

Looking at the mean and standard we get that Boost has the following values: `RSQ` : mean = 0.884 & standard error = 0.00978

Tree

.metric <chr>	mean <dbl>	std_err <dbl>
rmse	0.001510113	7.243611e-05
rsq	0.747908589	2.047482e-02
rmse	0.001510115	7.243588e-05
rsq	0.747908877	2.047486e-02
rmse	0.001510115	7.243588e-05
rsq	0.747908877	2.047486e-02

6 rows

Looking at the mean and standard we get that Tree has the following values: `RSQ` : mean = 0.748 & standard error = 0.0205

Looking at all the model's rsq, Boost would be the best model to test on the Testing Set. Boost has the highest RSQ mean and the lowest standard error.

R-Squared for (Testing Set)

model <chr>	.estimate <dbl>
Lasso	0.6969994
Ridge	0.7050942
Tree	0.7455435
Boost	0.8492171

4 rows

We combine the predictions of our model with the actual values of the testing set. After doing that we can see the R-Squared value which will tell us the accuracy of the models we used. Looking at the R-Squared of the four different models we see that the Boost model had the highest R-Squared and the tree model had the lowest R-Squared.

Conclusions:

Exploring the dataset we found some variables that significantly that affect the price of the laptop. After we decided to produce four different models to predict the price. These models were: Ridge, Lasso, Boost, and Tree.

Before analyzing the results of the rsq from the training set, we thought the tree model would do best because most of our data was categorical. Based on the results, we can say that the tree model performed well. However, boost came out on top as the highest accuracy. Boost came out on top because it reduced the bias the best without significantly increasing the variance. The lasso and ridge models performed worse than the tree and boost models.

This research shows Boost would be the best model. This model is the best at predicting the price of a laptop, based on the features the laptop has. Predicting with this model, will give the customer a price with around 85% accuracy. The goal of our project was achieved because we got a model with an accuracy over 70%.

Although our tree model didn't have the best accuracy, it can be the most useful model because it offers a solution for every distinct criterion that must be considered when looking for the best laptop. Customers can use the tree model to determine what features to look for based on the yes/no responses for each step of the tree, making it convenient to identify all required features. The tree model is simple to understand and to interpret, and the results are straightforward. Since the tree model can be used for both classification and regression models and can accommodate both categorical and numerical data, it can be the most useful model for our original dataset, which contains both categorical and numerical variables.

In conclusion, the boost model performed with the highest accuracy of 84.9%, but we would recommend our tree model. The tree model does not have the highest accuracy, however it is still over 70% and meets our goal. We think that the usefulness of the tree model overshadows the decrease in accuracy.

What we improved on?

After talking to the professor during our presentation he recommended some changes we should do to improve our project. One tweak we made was to change all our GB variables into numerical. Before the change, our GB variables were categorical. In order to do this change we deleted the GB part in the values then used the `as.numeric()` function to convert it to numeric.

The second change we made was to box-cox our response variable. Before, only lasso and ridge were being transformed because of the glmnet engine we used for those models. However, the boost and tree models were using the original data. So we transformed the data for the boost and tree models in order to get a higher accuracy model.

After doing both of these changes are boost and tree model's accuracy increased significantly while our lasso and ridge model's accuracy slightly decreased.

What can still be improved?

An improvement that can be made for this project is to replace the missing details for most of the laptops. After cleaning the data we lost more than half of the data. If that missing data was filled with updated information then we could have used all of the data. Using the whole dataset would of most likely improved the accuracy of the model because it would of had more observations to train on.