Final_Project

Nicola Cortinovis, Roberta Lamberti, Marta Lucas, Stefano Tumino

2024-02-27

Introduction

The aim of this project is to predict the price of a smartphone based on its features. The dataset used for this analysis is the mobile phones dataset, which contains 13 columns and 1224 rows:

Variables	Description	Type
X	Index of the dataset observation	int
Brand_Name	Name of the phone brand	chr
Model_Name	Name of the phone model	chr
Os	Operating system	chr
Popularity	The popularity of the phone in range 1-1224	int
Best_Price	Best price of the price-range (UAH)	num
Lowest_Price	Highest price of the price-range (UAH)	num
Highest_Price	Lowest price of the price-range (UAH)	num
Sellers_Amount	The amount of sellers who sold the phone	num
Screen_Size	The size of phone's screen (inches).	num
Memory_Size	The size of the phone's memory (GB)	num
Battery_Size	The size of the phone's battery (mAh)	num
Release_Date	The release date (M/Y) of the product on the market	chr

```r
df <- read.csv("phones_data.csv", header=T)
head(df)
```

```
##   X brand_name                                    model_name      os popularity
## 1 0    ALCATEL         1 1/8GB Bluish Black (5033D-2JALUAA) Android        422
## 2 1    ALCATEL 1 5033D 1/16GB Volcano Black (5033D-2LALUAF) Android        323
## 3 2    ALCATEL 1 5033D 1/16GB Volcano Black (5033D-2LALUAF) Android        299
## 4 3    ALCATEL 1 5033D 1/16GB Volcano Black (5033D-2LALUAF) Android        287
## 5 4      Nokia                           1.3 1/16GB Charcoal Android       1047
## 6 5      Honor                               10 6/64GB Black Android         71
##   best_price lowest_price highest_price sellers_amount screen_size memory_size
## 1       1690         1529          1819             36        5.00           8
## 2       1803         1659          2489             36        5.00          16
## 3       1803         1659          2489             36        5.00          16
## 4       1803         1659          2489             36        5.00          16
## 5       1999           NA            NA             10        5.71          16
## 6      10865        10631         11099              2        5.80          64
##   battery_size release_date
## 1         2000      10-2020
## 2         2000       9-2020
## 3         2000       9-2020
```

```
## 4           2000         9-2020
## 5           3000         4-2020
## 6           3400         6-2018
```

Specifically, our objective is to predict the `best_price` variable. Our approach consists of the following steps:

- Data exploration, where we inspect the dataset;
- Data preprocessing, where we clean the dataset and transform the variables;
- Data visualization, where we plot the relationships between the variables;
- Data splitting into training and test set and choice of performance metrics;
- Model building, where we fit different models to the training set and evaluate their performance. The models can be divided into two categories:
  – Linear models (both `lm` and `glm`);
  – Non-linear models (`gam`, `trees`, `ensemble`);
- Model comparison, where we compare the best models.

Data exploration

Data preprocessing

Firstly we remove from the dataset the index column `X`

```
df$X <- NULL
```

Our next step is to briefly explore the `chr` variables and transform the appropriate ones into factors.

- `model_name` won't be transformed into a factor because it has too many levels: there is almost a unique model_name for each row;
- `brand_name` will be transformed into a factor;
- `os` will be transformed into a factor;
- `release_date` won't be transformed into a factor because it will be used to create two new variables: `month` and `year`.

```
length(unique(df$model_name))
```

```
## [1] 1068
```

```
df$brand_name <- factor(df$brand_name)
df$os <- factor(df$os)
```

```
df$month <- as.numeric(sapply(df$release_date, FUN = function(x)
                      {strsplit(x, split = '[-]')[[1]][1]}))
df$year <- as.numeric(sapply(df$release_date, FUN = function(x)
                      {strsplit(x, split = '[-]')[[1]][2]}))-2000
```

For clarity's sake we convert the ukrainian currency (UAH) into euros (€) (27/02/2024 rate) and rename the blank " " `os` class as "other".

```
df$best_price <- df$best_price*0.024
df$lowest_price <- df$lowest_price*0.024
df$highest_price <- df$highest_price*0.024

levels(df$os)[1] <- "other"
```

Our next step is to investigate the `os` variable

```
table(df$os)
```

```
##
##         other      Android         EMUI          iOS        KAIOS     OxygenOS
##           197          915            2          103            1            3
## WindowsPhone
##             3
```

Given the insufficient amount of data for the `EMUI`, `KAIOS`, `OxygenOS` and `WindowsPhone` factor levels, we decide to aggregate them into the `other` and `Android` levels based on their characteristics.

```r
levels(df$os) <- c("other", "Android", "Android", "iOS", "other", "Android", "Android")
```

```r
summary(df)
```

```
##        brand_name   model_name              os        popularity
##  Samsung     :168   Length:1224        other  :198   Min.   :   1.0
##  Xiaomi      :111   Class :character   Android:923   1st Qu.: 306.8
##  Apple       :102   Mode  :character   iOS    :103   Median : 612.5
##  Motorola    : 62                                    Mean   : 612.5
##  Sigma mobile: 52                                    3rd Qu.: 918.2
##  HUAWEI      : 49                                    Max.   :1224.0
##  (Other)     :680
##    best_price        lowest_price      highest_price      sellers_amount
##  Min.   :   5.136   Min.   :   4.752   Min.   :   5.496   Min.   :  1.00
##  1st Qu.:  62.394   1st Qu.:  57.576   1st Qu.:  69.288   1st Qu.:  2.00
##  Median : 113.472   Median : 109.776   Median : 127.812   Median :  8.00
##  Mean   : 190.589   Mean   : 185.184   Mean   : 237.202   Mean   : 16.74
##  3rd Qu.: 223.752   3rd Qu.: 222.294   3rd Qu.: 304.170   3rd Qu.: 26.00
##  Max.   :1345.968   Max.   :1199.976   Max.   :1679.976   Max.   :125.00
##                     NA's   :260        NA's   :260
##   screen_size     memory_size        battery_size   release_date
##  Min.   :1.400   Min.   :3.20e-03   Min.   :  460   Length:1224
##  1st Qu.:5.162   1st Qu.:3.20e+01   1st Qu.: 2900   Class :character
##  Median :6.000   Median :6.40e+01   Median : 3687   Mode  :character
##  Mean   :5.394   Mean   :9.57e+01   Mean   : 3608
##  3rd Qu.:6.400   3rd Qu.:1.28e+02   3rd Qu.: 4400
##  Max.   :8.100   Max.   :1.00e+03   Max.   :18800
##  NA's   :2       NA's   :112        NA's   :10
##      month             year
##  Min.   : 1.000   Min.   :13.00
##  1st Qu.: 4.000   1st Qu.:18.00
##  Median : 8.000   Median :19.00
##  Mean   : 7.016   Mean   :18.85
##  3rd Qu.:10.000   3rd Qu.:20.00
##  Max.   :12.000   Max.   :21.00
##
```

From the summary we notice the presence of some NAs in the `battery_size`, `screen_size`, `memory_size`, `lowest_price` and `highest_price` variables. Given the small amount of NAs in the first two variables, we decide to remove the rows with NAs. Further investigation into the `memory_size` shows that the NAs are present only for the "other" `os` class, so we decide to fill them with the median of the `memory_size` computed on the "other" `os` class. The choice of the median is dictated by the presence of a few outliers that would therefore too influential on a mean.

```r
df <- df[- which(is.na(df$battery_size)),]
df <- df[- which(is.na(df$screen_size)),]
```

```r
tmp <- df[which(df$os == "other"),]$memory_size
df$memory_size[which(is.na(df$memory_size))] <- median(tmp[-which(is.na(tmp))])
```

The variables `lowest_price` and `highest_price` also show the presence of the outliers so their NAs have been substitued with their median.

```r
tmp <- which(is.na(df$lowest_price))
df$lowest_price[tmp] <- median(df$lowest_price[-tmp])
tmp <- which(is.na(df$highest_price))
df$highest_price[tmp] <- median(df$highest_price[-tmp])

summary(df)
```

```
##          brand_name    model_name               os         popularity
##  Samsung      :168   Length:1212        other  :197   Min.   :   1.0
##  Xiaomi       :111   Class :character   Android:918   1st Qu.: 305.8
##  Apple        : 96   Mode  :character   iOS    : 97   Median : 610.5
##  Motorola     : 62                                    Mean   : 611.7
##  Sigma mobile: 51                                     3rd Qu.: 918.2
##  HUAWEI       : 48                                    Max.   :1224.0
##  (Other)      :676
##    best_price        lowest_price      highest_price      sellers_amount
##  Min.   :   5.136   Min.   :   4.752   Min.   :   5.496   Min.   :  1.00
##  1st Qu.:  62.370   1st Qu.:  72.264   1st Qu.:  83.976   1st Qu.:  2.00
##  Median : 113.160   Median : 108.468   Median : 127.176   Median :  8.00
##  Mean   : 189.163   Mean   : 167.963   Mean   : 211.801   Mean   : 16.65
##  3rd Qu.: 216.996   3rd Qu.: 167.976   3rd Qu.: 210.150   3rd Qu.: 25.00
##  Max.   :1328.112   Max.   :1099.176   Max.   :1559.976   Max.   :125.00
##
##   screen_size     memory_size          battery_size    release_date
##  Min.   :1.400   Min.   :   0.0032   Min.   :  460   Length:1212
##  1st Qu.:5.200   1st Qu.:  16.0000   1st Qu.: 2900   Class :character
##  Median :6.000   Median :  64.0000   Median : 3687   Mode  :character
##  Mean   :5.396   Mean   :  86.5264   Mean   : 3610
##  3rd Qu.:6.400   3rd Qu.: 128.0000   3rd Qu.: 4400
##  Max.   :8.100   Max.   :1000.0000   Max.   :18800
##
##      month             year
##  Min.   : 1.000   Min.   :13.00
##  1st Qu.: 4.000   1st Qu.:18.00
##  Median : 8.000   Median :19.00
##  Mean   : 7.001   Mean   :18.86
##  3rd Qu.:10.000   3rd Qu.:20.00
##  Max.   :12.000   Max.   :21.00
##
```

The dataset contains several duplicates rows, where phones share the same characteristics but have different popularity levels. We decide to eliminate these duplicate observations as they provide redundant information. This process involves retaining only the first occurrence of each duplicate and replacing its popularity with the average popularity of the duplicates. An example of duplicates is given below:

```r
df[2:4,]
```

```
##   brand_name                                  model_name      os popularity
## 2    ALCATEL 1 5033D 1/16GB Volcano Black (5033D-2LALUAF) Android        323
## 3    ALCATEL 1 5033D 1/16GB Volcano Black (5033D-2LALUAF) Android        299
```

```
## 4    ALCATEL 1 5033D 1/16GB Volcano Black (5033D-2LALUAF) Android        287
##   best_price lowest_price highest_price sellers_amount screen_size memory_size
## 2     43.272       39.816        59.736             36           5          16
## 3     43.272       39.816        59.736             36           5          16
## 4     43.272       39.816        59.736             36           5          16
##   battery_size release_date month year
## 2         2000       9-2020     9   20
## 3         2000       9-2020     9   20
## 4         2000       9-2020     9   20
```

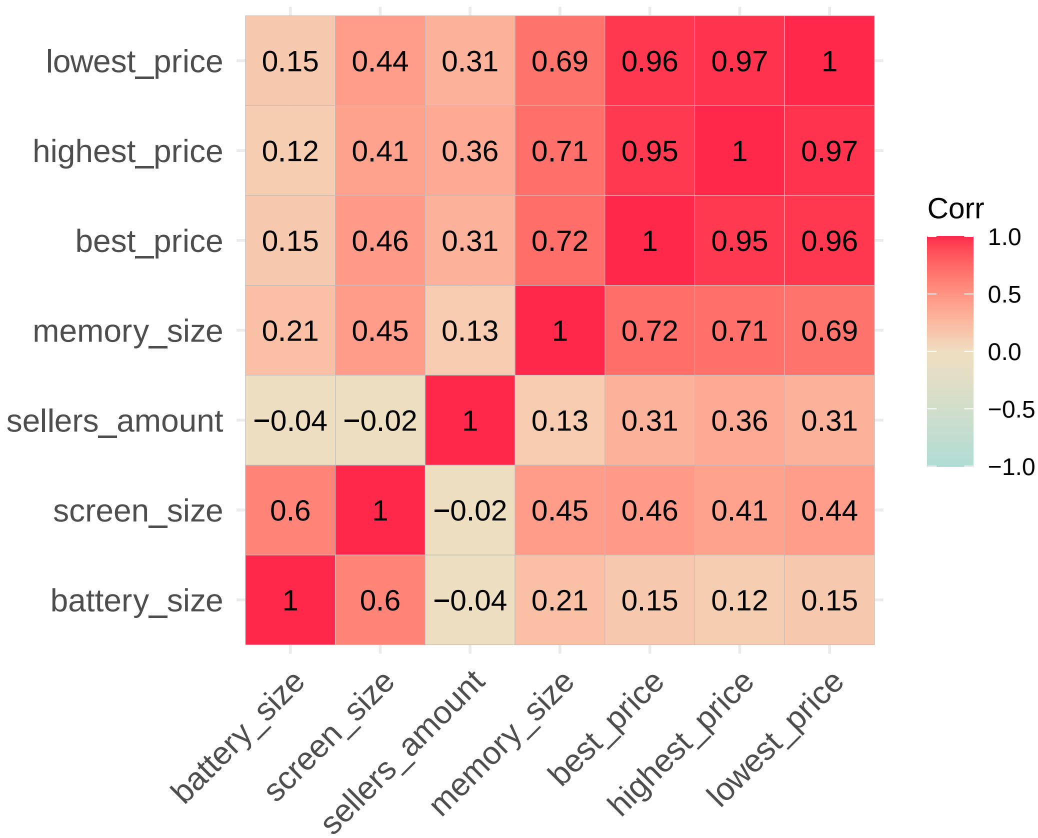
```r
# Find the indices of duplicate rows
idxs <- which(duplicated(df[,-c(2, 4)]))

# Check if each index is succeeded by the next one in the sequence
succ <- c(idxs[-1] - idxs[-length(idxs)] == 1, FALSE)

i = 1
while (i <= length(idxs)){
  start = idxs[i]
  sum <- c(df$popularity[start])
  while (succ[i]){
    i = i + 1
    sum <- c(sum, df$popularity[idxs[i]])
  }
  df$popularity[start] <- mean(sum)
  i = i + 1
}

# Remove the duplicate rows
df <- df[-idxs, ]

# Remove the model_name column
df$model_name <- NULL
```

The `popularity` variable is unique for each row, therefore we decided to create a new variable `popularity_levels` which divides the popularity into 4 classes: "low", "medium", "high" and "very high" based on the quartiles of the `popularity` variable.

```r
df$popularity <- as.numeric(df$popularity)

tag <- quantile(df$popularity)

df$popularity_levels <- cut(df$popularity, breaks = tag,
labels=c("low", "medium", "high", "very high"), include.lowest=TRUE)

df$popularity <- NULL
```

Data visualization

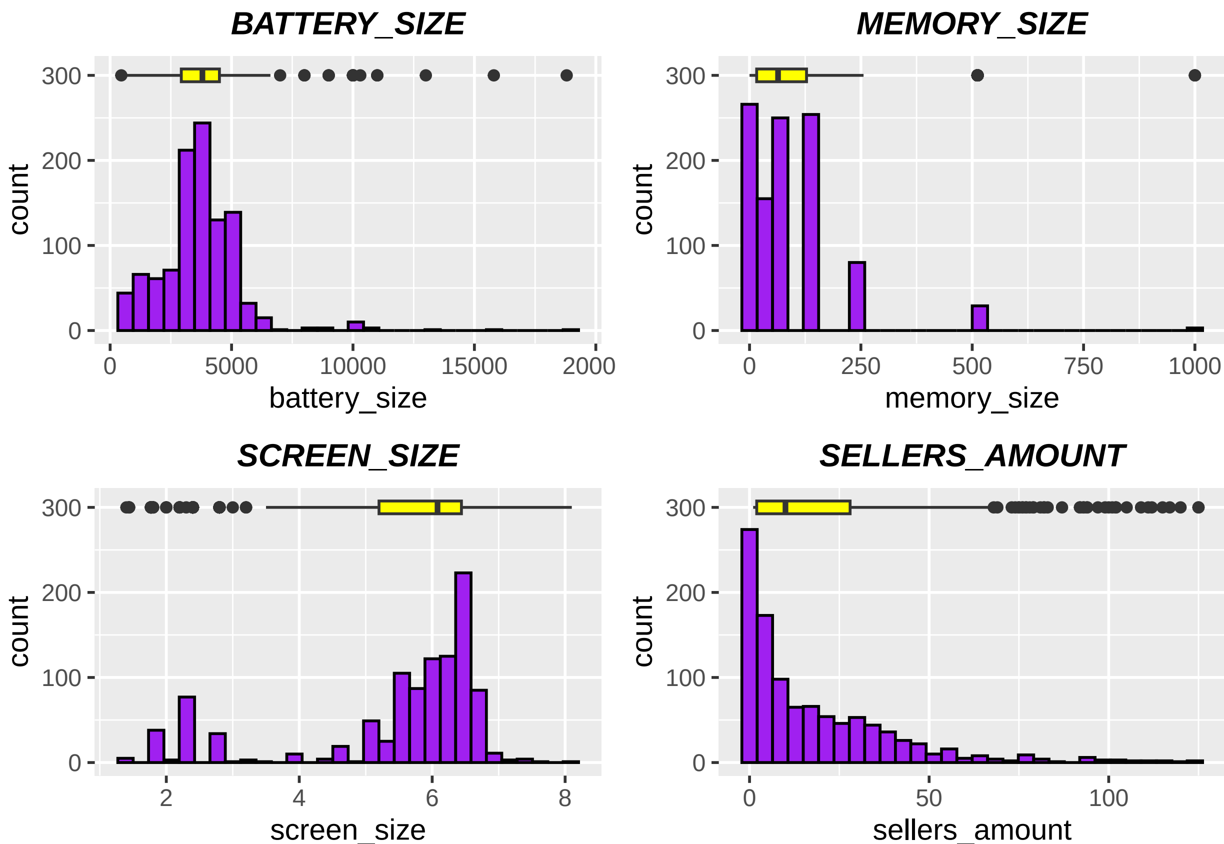
```r
corr <- cor(df[, c("battery_size", "memory_size", "screen_size", "best_price", "highest_price", "lowest_

ggcorrplot(corr, hc.order = TRUE, lab = TRUE, colors = c("#AFDDD5", "#EFDEC0", "#FF284B"))
```



From the correlation plot, we observe almost a linear correlation between `best_price` and the variables `highest_price` and `lowest_price`. For this reason we don't consider them in our analysis.

```
df$highest_price <- NULL
df$lowest_price <- NULL
```

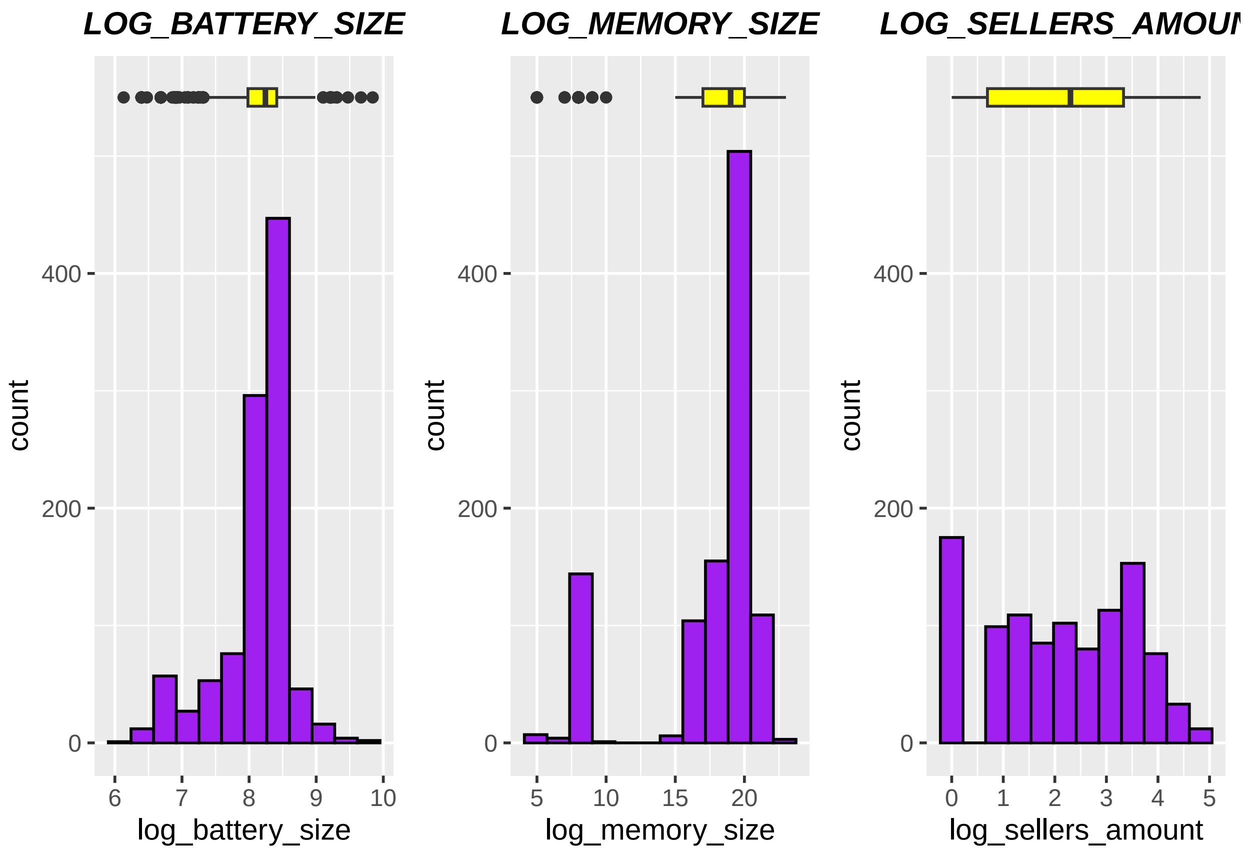
```
cols <- c("battery_size", "memory_size", "screen_size", "sellers_amount")

plots = list()

for (i in cols) {
  p1 <- ggplot(df, aes_string(x=i)) +
    geom_boxplot(fill="yellow", width= 15, position = position_nudge(y=300)) +
    geom_histogram(fill="purple", color = "black") +
    ggtitle(toupper(i)) + theme(plot.title = element_text(hjust = 0.5, size=12, face="bold.italic"))

  plots <- c(plots, list(p1))}

grid.arrange(grobs=plots, ncol=2, nrow=2)
```



The `battery_size`, `memory_size` and `sellers_amount` covariates plots all show the presence of right-skewed distributions. For those we decided to apply a logarithmic transformation, notably for `memory_size` we choose a logarithm of base 2. The `screen_size` covariate too displays a certain degree of left-skewness, but also a clear bi-modal distribution. For this reason we decided to leave it as it is.

```
df$log_battery_size <- log(df$battery_size)
df$log_memory_size <- floor(log2(df$memory_size*1e4))
df$log_sellers_amount <- log(df$sellers_amount)
```

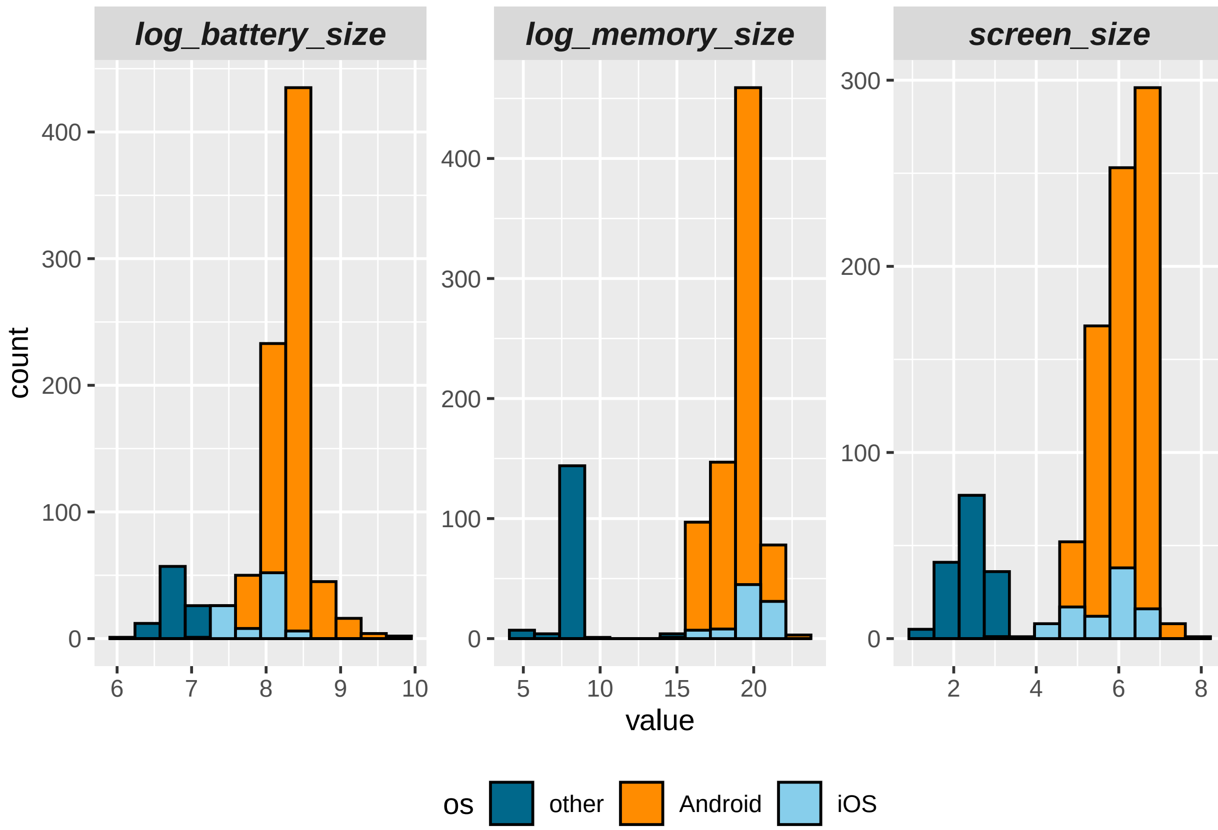
```
cols <- c("log_battery_size", "log_memory_size", "log_sellers_amount")

plots = list()

for (i in cols) {
  p1 <- ggplot(df, aes_string(x=i)) +
    geom_boxplot(fill="yellow", width= 15, position = position_nudge(y=550)) +
    geom_histogram(fill="purple", color = "black", bins = 12) + ggtitle(toupper(i)) +
    theme(plot.title = element_text(hjust = 0.5, size= 12, face="bold.italic"))

  plots <- c(plots, list(p1))}

grid.arrange(grobs=plots, ncol=3, nrow=1)
```



The transformations seem to handle the right-skewness of the covariates. The `log_battery_size` covariate is now approximately normally distributed. The `log_memory_size` covariate exhibits a bimodal behavior, but the transformation has reduced the right-skewness while introducing a degree of left-skewness. The `log_sellers_amount` is now approximately uniform.

We further investigate the bimodal behavior by plotting the three variables we're interested in by factoring the `os` type as well.

```
bimodal_vars <- c("log_battery_size", "log_memory_size", "screen_size")
df_long <- reshape2::melt(df, id.vars = "os", measure.vars = bimodal_vars)

colors <- c("other" = "deepskyblue4", "Android" = "darkorange", "iOS" = "skyblue")

# Create the histograms
p <- ggplot(df_long, aes(x = value, fill = os)) +
  geom_histogram(color = "black", position = "identity", bins = 12) +
  scale_fill_manual(values = colors) +
  facet_wrap(~variable, scales = "free") +
  theme(legend.position = "bottom", strip.text = element_text(size = 12, face = "bold.italic"))

# Print the plot
print(p)
```

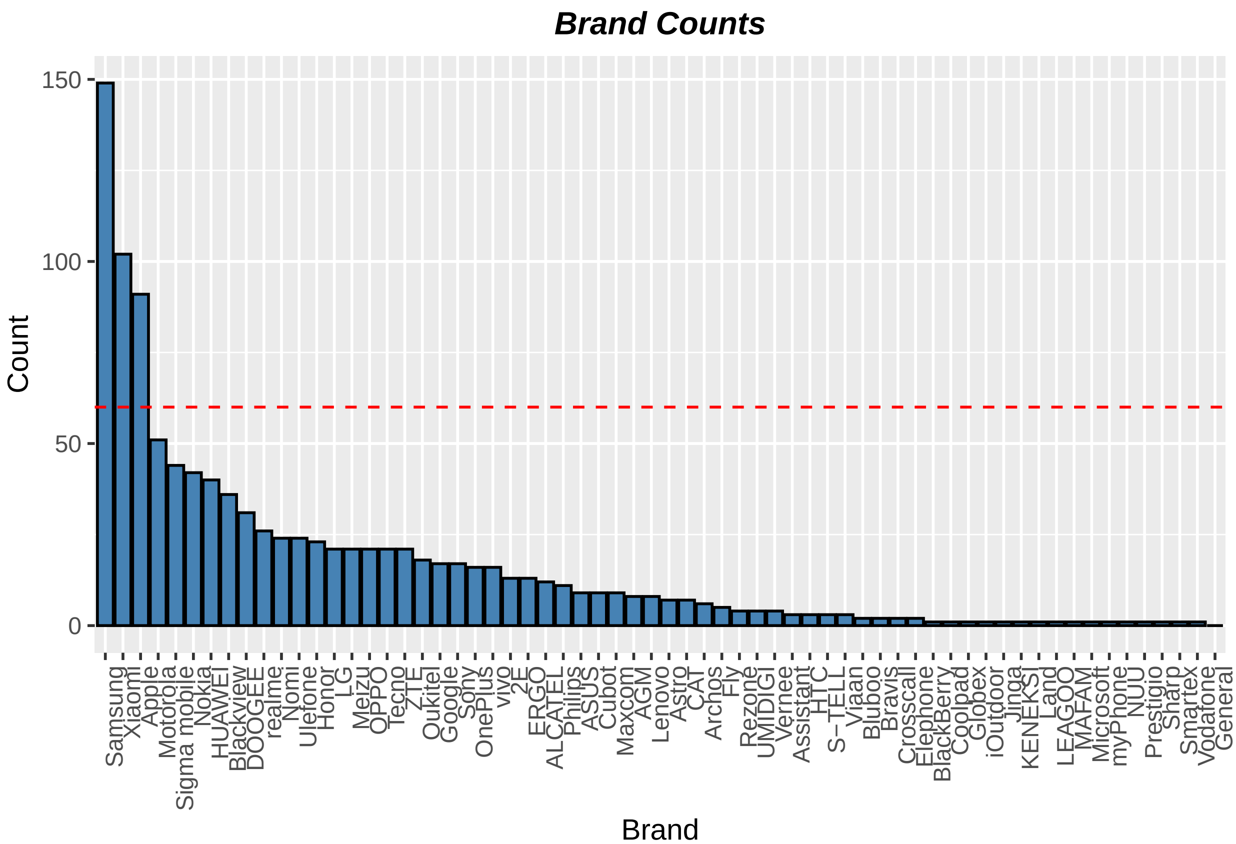


From these plots we see that there exist a clear difference in these variables distribution that's based on the type of operating system (`os`). `Android` and `iOs` categories refer to smartphones with more modern features, while the `other` category refers to old-fashioned mobile phones.

Categorical variables exploration

We begin by exploring the `brand_name` variable

```r
tmp <- sort(table(df$brand_name))

# Convert the table to a data frame for plotting
tmp_df <- data.frame(Brand = names(tmp), Count = as.vector(tmp))

# Create a bar plot
p <- ggplot(tmp_df, aes(x = reorder(Brand, -Count), y = Count)) +
  geom_bar(stat = "identity", fill = "steelblue", color = "black") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  xlab("Brand") +
  ylab("Count") +
  ggtitle("Brand Counts") + geom_hline(yintercept = 60, color = "red", linetype = "dashed") +
  theme(plot.title = element_text(hjust = 0.5, size=12, face="bold.italic"))

# Print the plot
print(p)
```



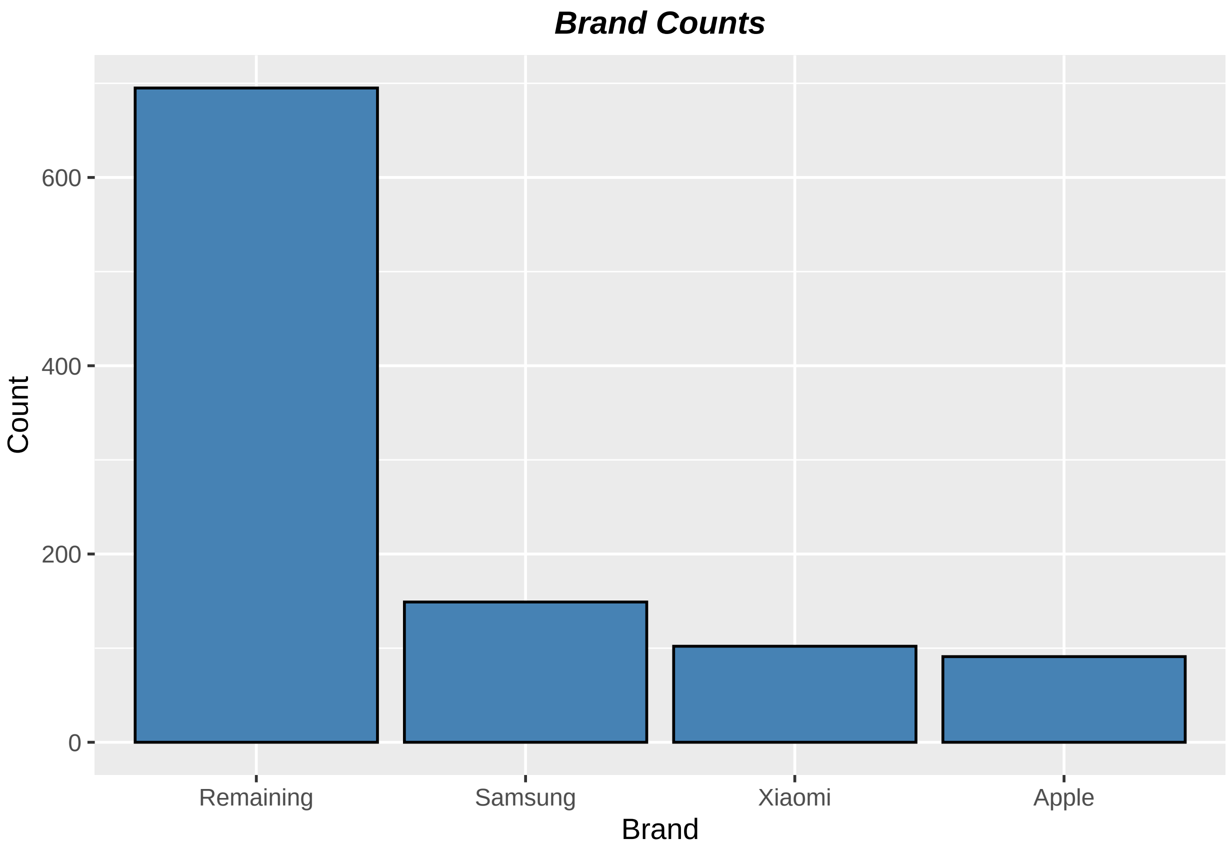
As there are too many classes with an exiguous amount of observations, we decided to group the brands into 4 categories: `Samsung`, `Xiaomi`, `Apple` and `Remaining`.

```
tmp <- sort(table(df$brand_name))
cut_line <- 60
to_remove <- names(tmp[tmp <= cut_line])

vals <- c()
for (i in 1:length(levels(df$brand_name))){
  if (any(levels(df$brand_name)[i] == to_remove)){
    vals <- c(vals, "Remaining")
  } else {
    vals <- c(vals, levels(df$brand_name)[i])
  }
}

levels(df$brand_name) <- vals

table(vals)
```

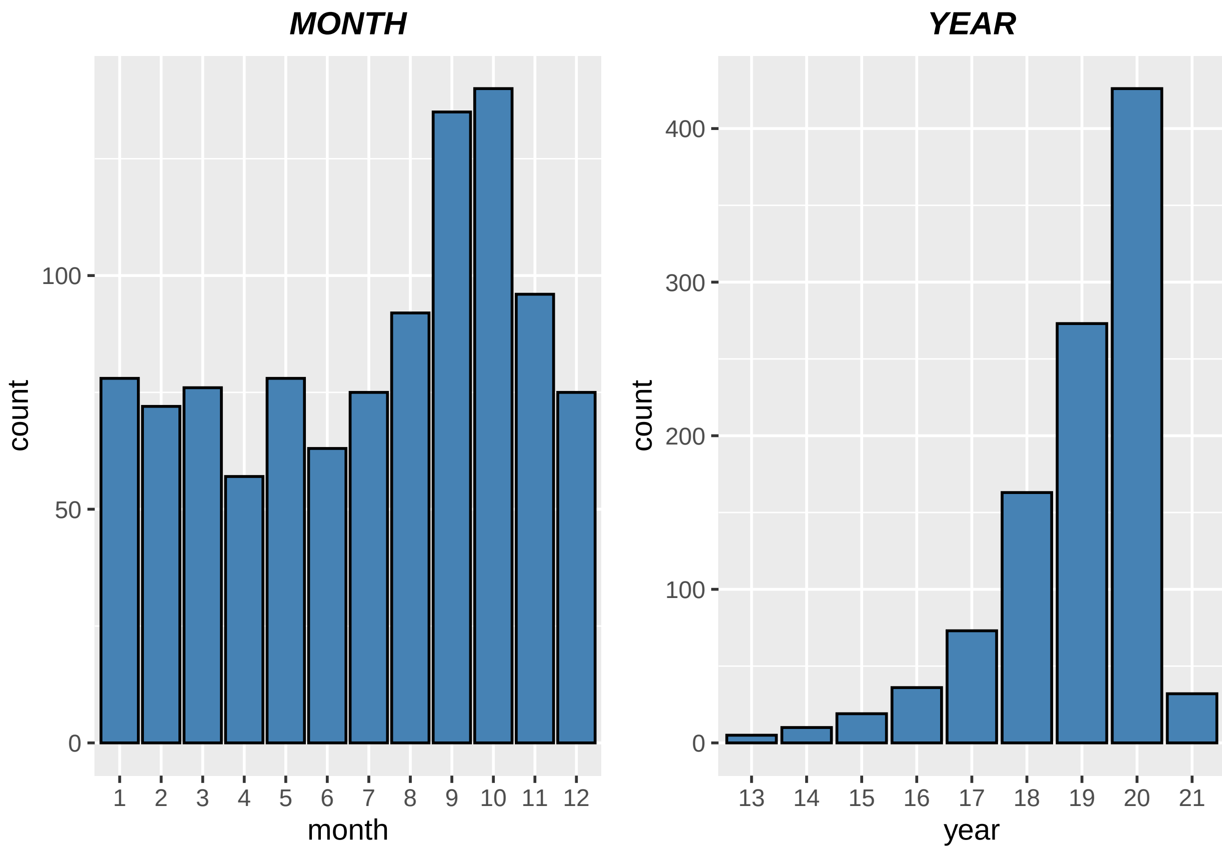
```
## vals
##     Apple Remaining   Samsung    Xiaomi
##         1        61         1         1
```

```
tmp <- sort(table(df$brand_name))

# Convert the table to a data frame for plotting
tmp_df <- data.frame(Brand = names(tmp), Count = as.vector(tmp))
```

```r
# Create a bar plot
p <- ggplot(tmp_df, aes(x = reorder(Brand, -Count), y = Count)) +
  geom_bar(stat = "identity", fill = "steelblue", color = "black") +
  theme(axis.text.x = element_text(hjust = 0.5)) +
  xlab("Brand") +
  ylab("Count") +
  ggtitle("Brand Counts") + theme(plot.title = element_text(hjust = 0.5, size=12, face="bold.italic"))

# Print the plot
print(p)
```



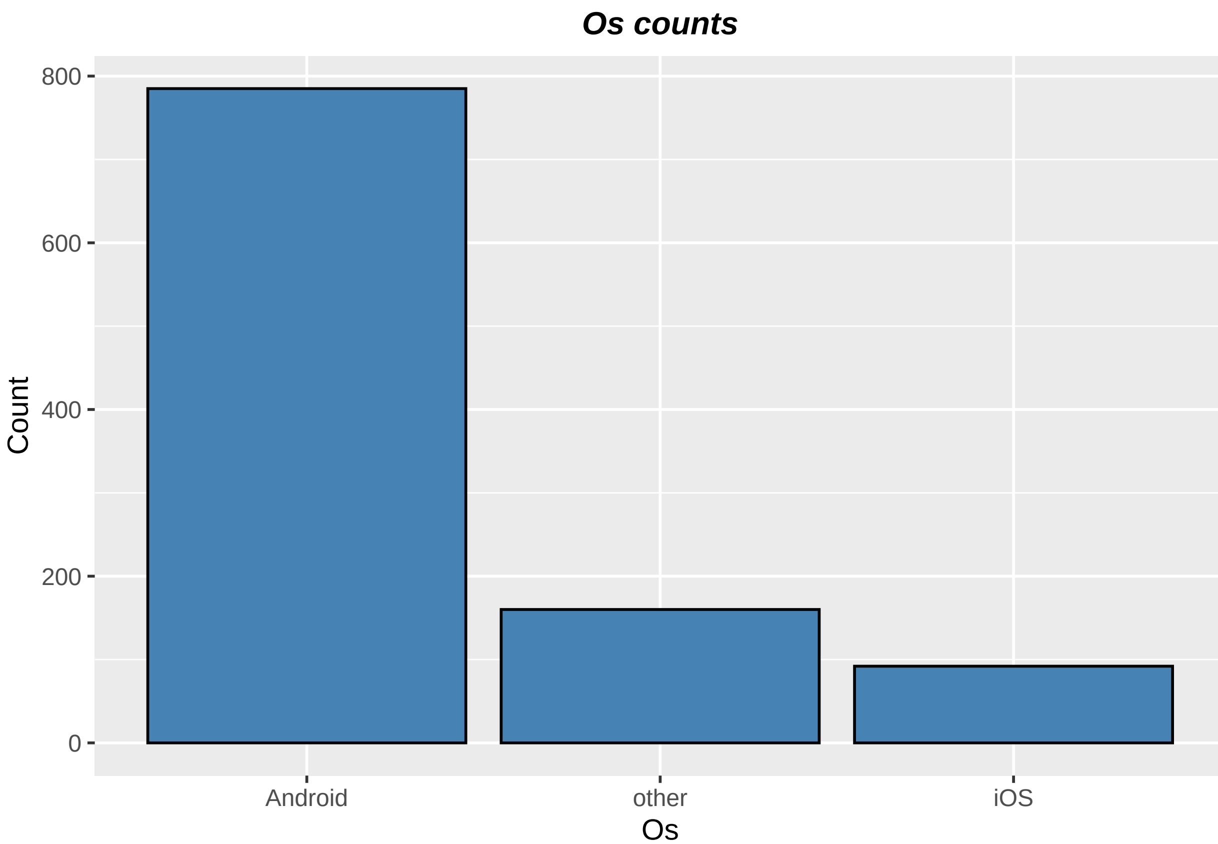

```r
plots = list()
for (i in c("month","year")) {
  p1 <- ggplot(data = data.frame(x = factor(sort(as.numeric(names(table(df[, i]))))), y = as.numeric(ta
        aes(x = x, y = y, fill = x)) +
        geom_bar(fill= "steelblue",color = "black", stat = "identity") +
        ggtitle(toupper(i)) + labs(x = i, y = "count") +
        theme(plot.title = element_text(hjust = 0.5, size=12, face="bold.italic"))
  plots <- c(plots, list(p1))}


grid.arrange(grobs=plots, ncol=2, nrow=1, common.legend = TRUE, legend="bottom")
```



In analyzing the `month` plot, we observe a notable increase in the number of phones sold during the months of September (9) and October (10). This surge aligns with the annual release of iPhones, suggesting a pattern linked to this event.

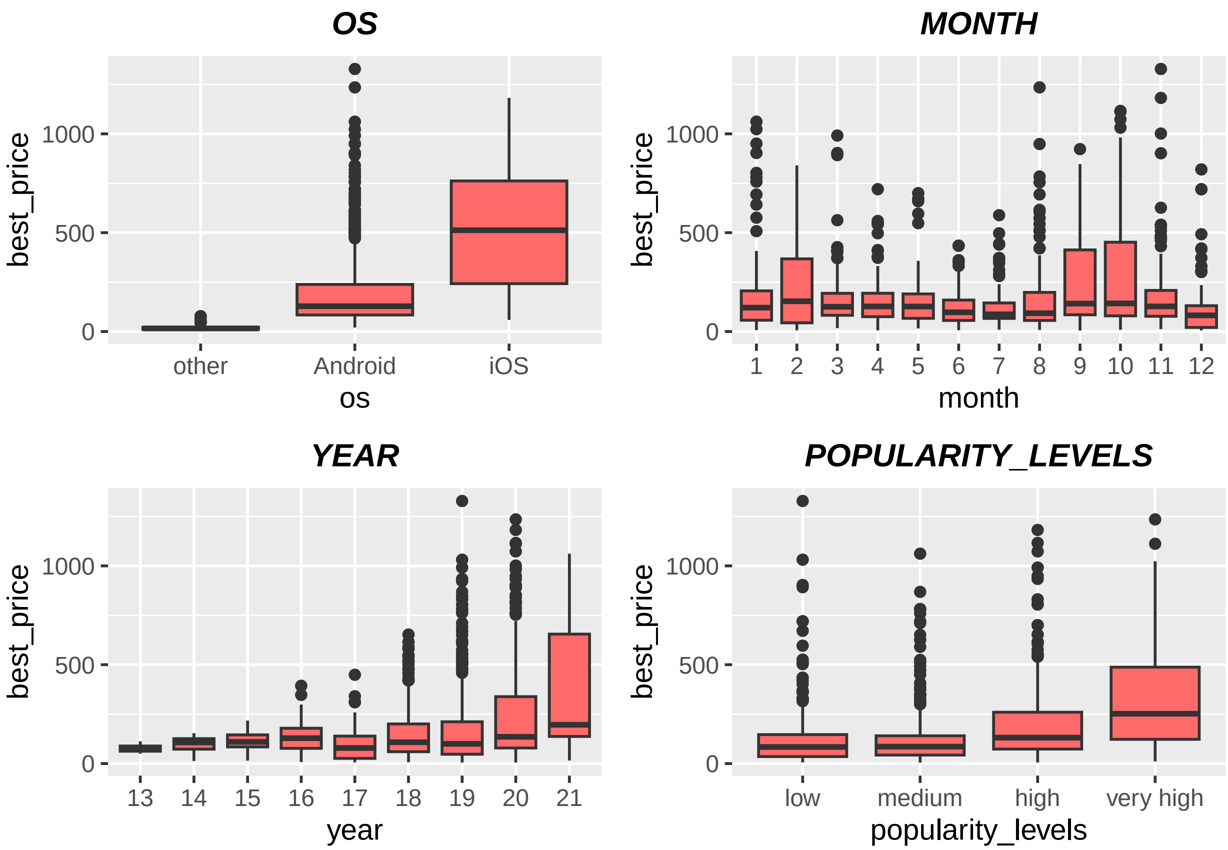
Additionally, the exponential growth trend evident in the `year` plot likely reflects the increasing demand for smartphones on a yearly basis. The peak observed in 2020 may be attributed to the COVID-19 pandemic, which significantly heightened the need for remote communication, during isolation for each member within most households. The count for 2021 appears lower since our dataset registers data until February.

```r
tmp <- sort(table(df$os))

# Convert the table to a data frame for plotting
tmp_df <- data.frame(Os = names(tmp), Count = as.vector(tmp))

# Create a bar plot
p <- ggplot(tmp_df, aes(x = reorder(Os, -Count), y = Count)) +
  geom_bar(stat = "identity", fill = "steelblue", color = "black") +
  theme(axis.text.x = element_text(hjust = 0.5)) +
  xlab("Os") +
  ylab("Count") +
  ggtitle("Os counts") + theme(plot.title = element_text(hjust = 0.5, size=12, face="bold.italic"))

# Print the plot
print(p)
```



It's evident that the `Android` operating system is the most chosen in the market.

```
cat_variables <- c("os", "month", "year", "popularity_levels")

plots = list()

for (i in cat_variables) {
  p1 <- ggplot(df, aes_string(x= factor(df[,i]), y = df$best_price)) +
        geom_boxplot(fill="indianred1") +
        labs(x = i, y = "best_price") +
        theme(plot.title = element_text(hjust = 0.5, size= 12, face="bold.italic")) +
        ggtitle(toupper(i))

  plots <- c(plots, list(p1))}

grid.arrange(grobs=plots, ncol=2, nrow=2)
```



From the boxplots we observe that:

- For the `os` variable the `iOS` operating system shows a significantly higher `best_price` than the others. We also notice that the `Android` level has several outliers;
- For the `month` variable, the `best_price` is higher in the months of September and October, which aligns with the release of new iPhones. In the month of February, the `best_price` is also higher than the others because of the release of the flagships Samsung smartphones;
- For the `year` variable, the `best_price` exhibits a growing trend over the years;
- For the `popularity_levels` variable, the `best_price` grows with the popularity, except for some outliers.

Training and testing datasets split

We decide to split the dataset into a training and test set using a 80/20 split. While this split ignores the temporal data of `month` and `year`, empirical results show no practical differences between an arbitrary and a time-based split. A seed was set to ensure the results reproducibility.

```
set.seed(15)

split <- initial_split(df, prop = 0.8)
train <- training(split)
test <- testing(split)
```

Linear models

We begin by defining the metrics on which our models will be evaluated:

- RMSE on the training set;

- R2 on the training set;
- RMSE on the test set;
- R2 on the test set;
- AIC;
- BIC;
- Training time.

Our analysis begins by fitting a linear model to the data, following a top-down approach to select the most significant variables.

Model 1

Our first model includes all the variables with the applied transformations.

```
start <- Sys.time()
lm_model_1 <- lm(best_price ~ brand_name + os + log_sellers_amount + screen_size +
                 log_memory_size + log_battery_size + month + year + popularity_levels,
                 data = train)

end <- Sys.time()

summary(lm_model_1)
```

```
##
## Call:
## lm(formula = best_price ~ brand_name + os + log_sellers_amount +
##     screen_size + log_memory_size + log_battery_size + month +
##     year + popularity_levels, data = train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -452.30  -69.93  -11.05   48.35  701.39
##
## Coefficients:
##                              Estimate Std. Error t value Pr(>|t|)
## (Intercept)                  -722.535    111.647  -6.472 1.67e-10 ***
## brand_nameApple                26.415    126.691   0.209 0.834889
## brand_nameSamsung             115.252     14.132   8.155 1.31e-15 ***
## brand_nameXiaomi              -52.899     15.142  -3.494 0.000502 ***
## osAndroid                    -728.698     46.583 -15.643  < 2e-16 ***
## osiOS                        -446.959    125.474  -3.562 0.000389 ***
## log_sellers_amount              2.163      4.162   0.520 0.603342
## screen_size                    41.561     11.457   3.628 0.000304 ***
## log_memory_size                68.367      4.187  16.327  < 2e-16 ***
## log_battery_size              -25.059     13.928  -1.799 0.072353 .
## month                           1.144      1.324   0.864 0.387713
## year                           14.740      3.989   3.696 0.000234 ***
## popularity_levelsmedium       -19.161     12.279  -1.560 0.119049
## popularity_levelshigh         -25.808     13.087  -1.972 0.048937 *
## popularity_levelsvery high     17.329     15.480   1.119 0.263299
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 121.9 on 814 degrees of freedom
## Multiple R-squared:  0.6755, Adjusted R-squared:  0.6699
```

```
## F-statistic:   121 on 14 and 814 DF,  p-value: < 2.2e-16
```

```r
y_hat <- predict(lm_model_1, newdata = train)
pred <- predict(lm_model_1, newdata = test)

result1 <- c("RMSE"=RMSE(y_hat, train$best_price), "R2"=R2(y_hat, train$best_price),
             "RMSE_test"=RMSE(pred, test$best_price), "R2_test"=R2(pred, test$best_price),
             "AIC"=AIC(lm_model_1), "BIC" = BIC(lm_model_1), "Time"=end-start)

result1
```

```
##         RMSE           R2    RMSE_test      R2_test          AIC          BIC
## 1.207527e+02 6.755285e-01 1.507590e+02 6.214815e-01 1.033263e+04 1.040815e+04
##         Time
## 8.482695e-03
```

The results show that the variables: `month`, `popularity_levels`, `log_sellers_amount` and `log_battery_size` don't affect the model as much. Our next model is the result of dropping each variable once at the time until no variable can be dropped without a significant loss in the model's performance.

Model 2

```r
start <- Sys.time()
lm_model_2 <- lm(best_price ~ brand_name + os + screen_size + log_memory_size +
                 year, data = train)

end <- Sys.time()

summary(lm_model_2)
```

```
##
## Call:
## lm(formula = best_price ~ brand_name + os + screen_size + log_memory_size +
##     year, data = train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -474.71  -71.16  -11.75   48.28  723.36
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)       -906.829     63.060 -14.380  < 2e-16 ***
## brand_nameApple     39.992    127.503   0.314 0.753865
## brand_nameSamsung  122.612     13.549   9.049  < 2e-16 ***
## brand_nameXiaomi   -49.462     14.952  -3.308 0.000980 ***
## osAndroid         -751.124     46.165 -16.270  < 2e-16 ***
## osiOS             -460.337    126.354  -3.643 0.000286 ***
## screen_size         40.410     10.512   3.844 0.000130 ***
## log_memory_size     68.317      4.179  16.347  < 2e-16 ***
## year                15.262      3.672   4.156 3.58e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 122.9 on 820 degrees of freedom
## Multiple R-squared:  0.6676, Adjusted R-squared:  0.6644
```

```
## F-statistic: 205.9 on 8 and 820 DF,  p-value: < 2.2e-16
```

```
y_hat <- predict(lm_model_2, newdata = train)
pred <- predict(lm_model_2, newdata = test)

result2 <- c("RMSE"=RMSE(y_hat, train$best_price), "R2"=R2(y_hat, train$best_price),
             "RMSE_test"=RMSE(pred, test$best_price), "R2_test"=R2(pred, test$best_price),
             "AIC"=AIC(lm_model_2), "BIC" = BIC(lm_model_2), "Time"=end-start)

result2
```

```
##         RMSE           R2    RMSE_test      R2_test          AIC          BIC
## 1.222165e+02 6.676144e-01 1.514706e+02 6.178734e-01 1.034061e+04 1.038781e+04
##         Time
## 6.316900e-03
```

The results show that the model's performance is only slightly worsened by excluding the variables `log_sellers_amount`, `log_battery_size`, `popularity_levels` and `month`.

Model 3

The third model includes interactions between some variables to check if it's possible to improve the model's performance.

```
start <- Sys.time()

lm_model_3 <- lm(best_price ~ brand_name + os + screen_size + year +
                 os:log_memory_size  + os:screen_size, data = train)


end <- Sys.time()

summary(lm_model_3)
```

```
##
## Call:
## lm(formula = best_price ~ brand_name + os + screen_size + year +
##     os:log_memory_size + os:screen_size, data = train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -247.25  -65.88  -11.29   42.92  769.28
##
## Coefficients:
##                           Estimate Std. Error t value Pr(>|t|)
## (Intercept)               -205.391    101.062  -2.032 0.042444 *
## brand_nameApple           -486.406    126.678  -3.840 0.000133 ***
## brand_nameSamsung          124.295     12.328  10.083  < 2e-16 ***
## brand_nameXiaomi           -42.183     13.519  -3.120 0.001871 **
## osAndroid                -1402.493    109.593 -12.797  < 2e-16 ***
## osiOS                    -1705.574    204.953  -8.322 3.63e-16 ***
## screen_size                 10.799     23.565   0.458 0.646884
## year                        10.020      3.445   2.909 0.003726 **
## osother:log_memory_size      1.267      8.846   0.143 0.886118
## osAndroid:log_memory_size   87.200      4.516  19.308  < 2e-16 ***
## osiOS:log_memory_size       81.545     11.559   7.055 3.69e-12 ***
```

```
## osAndroid:screen_size       -21.620     26.328  -0.821 0.411781
## osiOS:screen_size           188.530     31.674   5.952 3.93e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 110.7 on 816 degrees of freedom
## Multiple R-squared:  0.7315, Adjusted R-squared:  0.7275
## F-statistic: 185.2 on 12 and 816 DF,  p-value: < 2.2e-16
```

```
y_hat <- predict(lm_model_3, newdata = train)
pred <- predict(lm_model_3, newdata = test)

result3 <- c("RMSE"=RMSE(y_hat, train$best_price), "R2"=R2(y_hat, train$best_price),
             "RMSE_test"=RMSE(pred, test$best_price), "R2_test"=R2(pred, test$best_price),
             "AIC"=AIC(lm_model_3), "BIC" = BIC(lm_model_3),  "Time"=end-start)

result3
```

```
##         RMSE           R2    RMSE_test      R2_test          AIC          BIC
## 1.098472e+02 7.314897e-01 1.384910e+02 6.831086e-01 1.017169e+04 1.023778e+04
##         Time
## 7.163763e-03
```

The results show that the model's performance is improved by including these interactions.

Model 4

The previous modelstill has a non-significant variable, namely `screen_size`. The fourth model will therefore exclude it.

```
start <- Sys.time()

lm_model_4 <- lm(best_price ~ brand_name + os + year + os:log_memory_size +
                os:screen_size, data = train)

end <- Sys.time()

summary(lm_model_4)
```

```
##
## Call:
## lm(formula = best_price ~ brand_name + os + year + os:log_memory_size +
##     os:screen_size, data = train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -247.25  -65.88  -11.29   42.92  769.28
##
## Coefficients:
##                              Estimate Std. Error t value Pr(>|t|)
## (Intercept)                  -205.391    101.062  -2.032 0.042444 *
## brand_nameApple              -486.406    126.678  -3.840 0.000133 ***
## brand_nameSamsung             124.295     12.328  10.083  < 2e-16 ***
## brand_nameXiaomi              -42.183     13.519  -3.120 0.001871 **
## osAndroid                   -1402.493    109.593 -12.797  < 2e-16 ***
## osiOS                       -1705.574    204.953  -8.322 3.63e-16 ***
```

```
## year                         10.020      3.445   2.909 0.003726 **
## osother:log_memory_size       1.267      8.846   0.143 0.886118
## osAndroid:log_memory_size    87.200      4.516  19.308  < 2e-16 ***
## osiOS:log_memory_size        81.545     11.559   7.055 3.69e-12 ***
## osother:screen_size          10.799     23.565   0.458 0.646884
## osAndroid:screen_size       -10.821     11.517  -0.940 0.347723
## osiOS:screen_size           199.330     21.008   9.488  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 110.7 on 816 degrees of freedom
## Multiple R-squared:  0.7315, Adjusted R-squared:  0.7275
## F-statistic: 185.2 on 12 and 816 DF,  p-value: < 2.2e-16
```

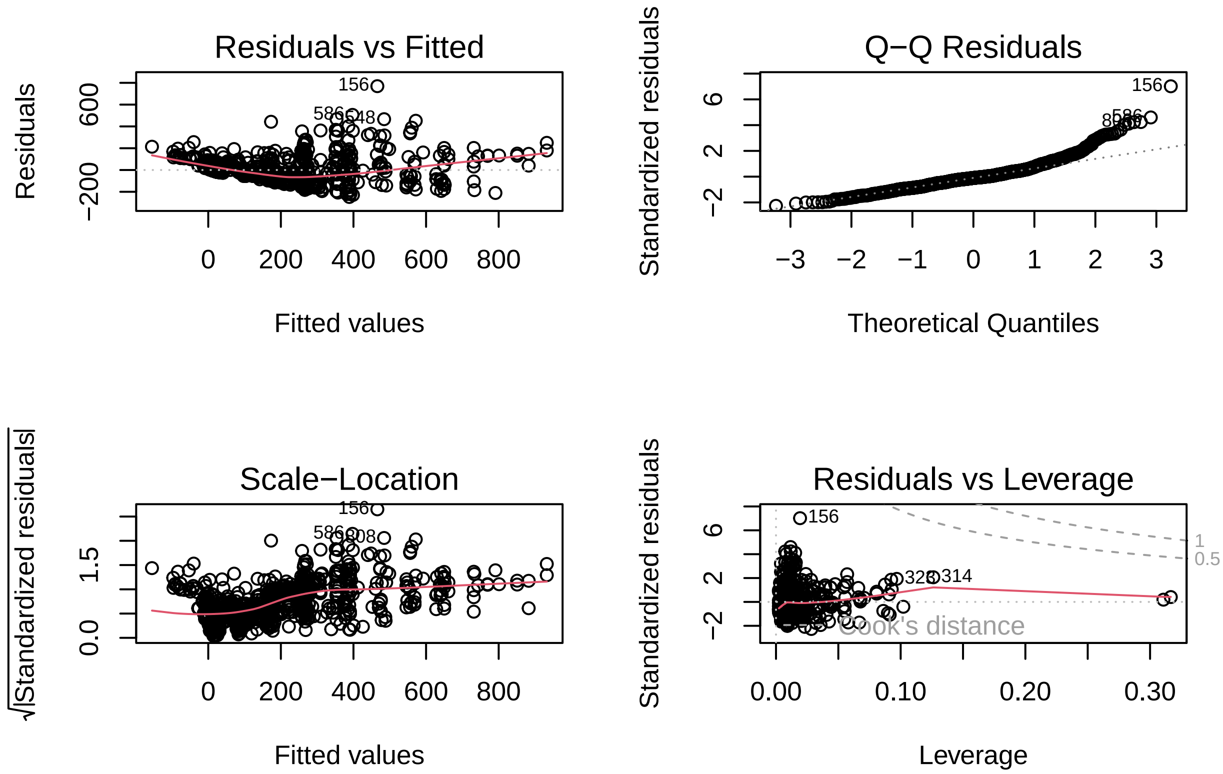
```r
y_hat <- predict(lm_model_4, newdata = train)
pred <- predict(lm_model_4, newdata = test)

result4 <- c("RMSE"=RMSE(y_hat, train$best_price), "R2"=R2(y_hat, train$best_price),
             "RMSE_test"=RMSE(pred, test$best_price), "R2_test"=R2(pred, test$best_price),
             "AIC"=AIC(lm_model_4), "BIC" = BIC(lm_model_4), "Time"=end-start)

result4
```

```
##         RMSE           R2    RMSE_test      R2_test          AIC          BIC
## 1.098472e+02 7.314897e-01 1.384910e+02 6.831086e-01 1.017169e+04 1.023778e+04
##         Time
## 6.441116e-03
```

The results for this final model are unchanged compared to the previous one. Since it is our best model so far we decide to further analyze it by checking its residuals.

```r
par(mfrow=c(2,2))
plot(lm_model_4)
```

```
## Warning: not plotting observations with leverage one:
##   581
```



From the residuals of the model we can see that there is heteroschedasticity. This means that the variance of the residuals is not constant. This is a violation of the assumption of homoscedasticity. We can also see that the residuals are not normally distributed. Knowing that all the dependent variables are positive, we will try to fix these issues by applying a logarithmic transformation to the response variable.

Linear models with logarithmic response variable

```r
df$log_best_price <- log(df$best_price)
train$log_best_price <- log(train$best_price)
test$log_best_price <- log(test$best_price)
```

Model 5

We now test the fourth model only adding a logarithmic transformation on the `best_price` variable.

```r
start <- Sys.time()

lm_model_5 <- lm(log_best_price ~ brand_name + os + month + year +
                 os:log_memory_size + os:log_battery_size, data = train)

end <- Sys.time()

summary(lm_model_5)
```

```
##
## Call:
## lm(formula = log_best_price ~ brand_name + os + month + year +
##     os:log_memory_size + os:log_battery_size, data = train)
##
## Residuals:
```

```
##      Min       1Q   Median       3Q      Max
## -1.01655 -0.28737 -0.03299  0.24005  1.54266
##
## Coefficients:
##                              Estimate Std. Error t value Pr(>|t|)
## (Intercept)                 -0.774371   0.521232  -1.486  0.13776
## brand_nameApple             -0.139240   0.449233  -0.310  0.75668
## brand_nameSamsung            0.396515   0.046326   8.559  < 2e-16 ***
## brand_nameXiaomi            -0.133818   0.049399  -2.709  0.00689 **
## osAndroid                   -3.053416   0.710840  -4.296 1.95e-05 ***
## osiOS                      -10.478699   1.540596  -6.802 2.00e-11 ***
## month                       -0.008322   0.004394  -1.894  0.05859 .
## year                         0.006435   0.012079   0.533  0.59439
## osother:log_memory_size      0.081950   0.032362   2.532  0.01152 *
## osAndroid:log_memory_size    0.442696   0.014465  30.604  < 2e-16 ***
## osiOS:log_memory_size        0.243337   0.041596   5.850 7.11e-09 ***
## osother:log_battery_size     0.388643   0.058810   6.608 7.01e-11 ***
## osAndroid:log_battery_size   0.029390   0.064885   0.453  0.65070
## osiOS:log_battery_size       1.594637   0.216560   7.363 4.39e-13 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4071 on 815 degrees of freedom
## Multiple R-squared:  0.8759, Adjusted R-squared:  0.874
## F-statistic: 442.6 on 13 and 815 DF,  p-value: < 2.2e-16
```

```r
y_hat <- predict(lm_model_5, newdata = train)
pred <- predict(lm_model_5, newdata = test)

result_log_5 <- c("RMSE"=RMSE(exp(y_hat), train$best_price), "R2"=R2(exp(y_hat), train$best_price),
                  "RMSE_test"=RMSE(exp(pred), test$best_price), "R2_test"=R2(exp(pred), test$best_price
                  "AIC"=AIC(lm_model_5), "BIC" = BIC(lm_model_5),  "Time"=end-start)

result_log_5
```

```
##         RMSE           R2    RMSE_test      R2_test          AIC          BIC
## 1.113002e+02 7.298519e-01 1.463461e+02 6.520121e-01 8.784421e+02 9.492454e+02
##         Time
## 3.336191e-03
```

Model 6

We modify the previous model by removing the non-significant variable `year`.

```r
start <- Sys.time()

lm_model_6 <- lm(log_best_price ~ brand_name + os + month +
                 os:log_memory_size + os:log_battery_size, data = train)

end <- Sys.time()

summary(lm_model_6)
```

```
##
## Call:
## lm(formula = log_best_price ~ brand_name + os + month + os:log_memory_size +
```

```
##     os:log_battery_size, data = train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.04089 -0.28854 -0.03606  0.23881  1.53486
##
## Coefficients:
##                              Estimate Std. Error t value Pr(>|t|)
## (Intercept)                 -0.658002   0.473038  -1.391  0.16460
## brand_nameApple             -0.179461   0.442648  -0.405  0.68527
## brand_nameSamsung            0.395929   0.046293   8.553  < 2e-16 ***
## brand_nameXiaomi            -0.134632   0.049354  -2.728  0.00651 **
## osAndroid                   -3.164253   0.679408  -4.657 3.74e-06 ***
## osiOS                      -10.724059   1.469485  -7.298 6.94e-13 ***
## month                       -0.008587   0.004364  -1.968  0.04944 *
## osother:log_memory_size      0.083120   0.032273   2.576  0.01018 *
## osAndroid:log_memory_size    0.444173   0.014191  31.300  < 2e-16 ***
## osiOS:log_memory_size        0.246881   0.041042   6.015 2.71e-09 ***
## osother:log_battery_size     0.388250   0.058779   6.605 7.15e-11 ***
## osAndroid:log_battery_size   0.040345   0.061513   0.656  0.51209
## osiOS:log_battery_size       1.622049   0.210265   7.714 3.55e-14 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4069 on 816 degrees of freedom
## Multiple R-squared:  0.8759, Adjusted R-squared:  0.8741
## F-statistic: 479.9 on 12 and 816 DF,  p-value: < 2.2e-16
```

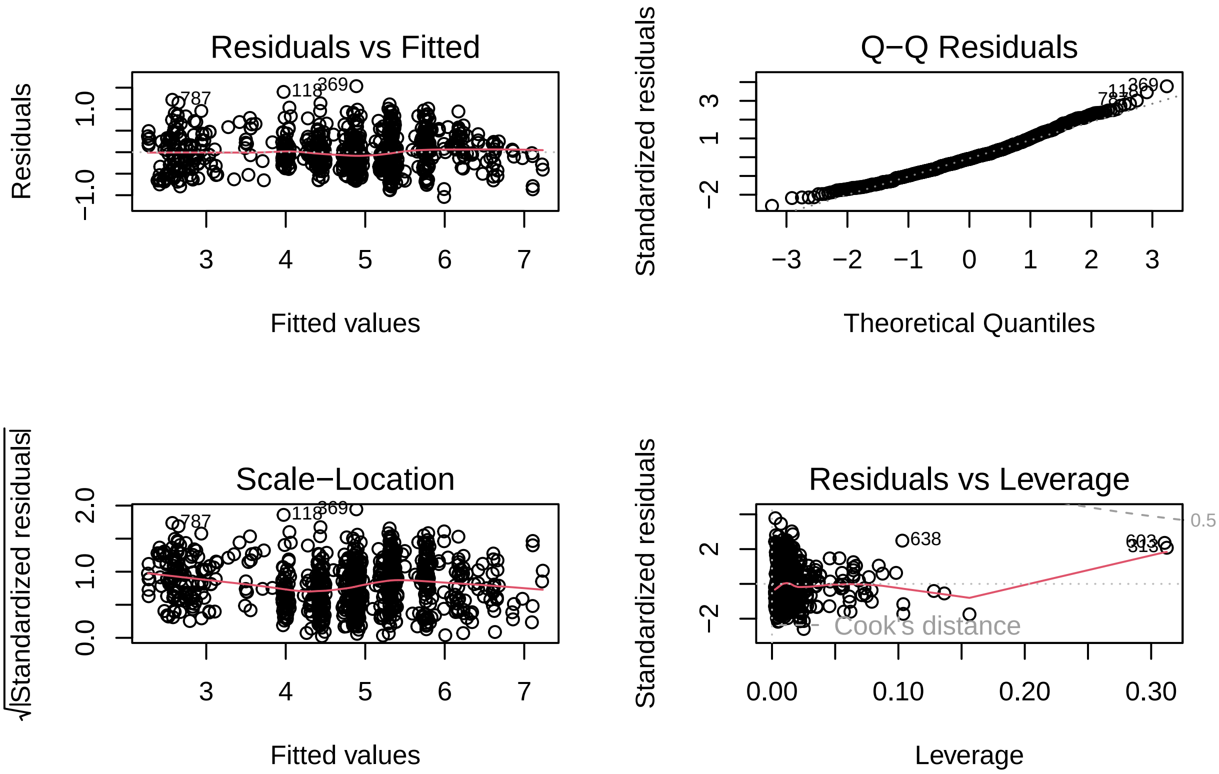
```r
y_hat <- predict(lm_model_6, newdata = train)
pred <- predict(lm_model_6, newdata = test)

result_log_6 <- c("RMSE"=RMSE(exp(y_hat), train$best_price), "R2"=R2(exp(y_hat), train$best_price),
                  "RMSE_test"=RMSE(exp(pred), test$best_price), "R2_test"=R2(exp(pred), test$best_price
                  "AIC"=AIC(lm_model_6), "BIC" = BIC(lm_model_6),  "Time"=end-start)

result_log_6
```

```
##         RMSE           R2    RMSE_test      R2_test          AIC          BIC
## 1.121879e+02 7.256394e-01 1.474927e+02 6.468979e-01 8.767307e+02 9.428138e+02
##         Time
## 6.474972e-03
```

The results show that the model's performance is not improved by including these interactions, but we decide to further analyze it by checking its residuals to compare it with the previous model.

```r
par(mfrow=c(2,2))
plot(lm_model_6)
```

```
## Warning: not plotting observations with leverage one:
##   581
```



The logarithmic model exibits less heteroschedasticity, the residuals are more normally distributed and they are also less right-skewed. The R2 and RMSE performance indexes are slightly worse compared to the linear model ones.

Results comparison

```
results_lm <- rbind(result1, result2, result3, result4, result_log_5, result_log_6)
rownames(results_lm) <- c("Model 1", "Model 2", "Model 3", "Model 4", "Log_Model 5", "Log_Model 6")
round(results_lm, 3)
```

```
##                RMSE    R2 RMSE_test R2_test       AIC       BIC  Time
## Model 1     120.753 0.676   150.759   0.621 10332.629 10408.153 0.008
## Model 2     122.216 0.668   151.471   0.618 10340.606 10387.809 0.006
## Model 3     109.847 0.731   138.491   0.683 10171.693 10237.776 0.007
## Model 4     109.847 0.731   138.491   0.683 10171.693 10237.776 0.006
## Log_Model 5 111.300 0.730   146.346   0.652   878.442   949.245 0.003
## Log_Model 6 112.188 0.726   147.493   0.647   876.731   942.814 0.006
```

The results show that the best model is the Log_Model_6, which has similar R2 and RMSE to the best linear model on both training and test sets, while better satisfying the homoschedasticity and normality assumptions on the residuals.

Generalized linear models

We next try to fit a generalized linear model to the data. We will use the Gaussian and Gamma family with log and identity link functions.

We begin with a gaussian family with a logarithmic link function since the response variable is right skewed and positive.

```r
# Gaussian with log link
start <- Sys.time()
glm_1 <- glm(best_price ~ os + sellers_amount + screen_size + memory_size + battery_size +
            month + year + popularity_levels + brand_name,
            data = train, family = gaussian(link = "log"))
end <- Sys.time()

summary(glm_1)
```

```
##
## Call:
## glm(formula = best_price ~ os + sellers_amount + screen_size +
##     memory_size + battery_size + month + year + popularity_levels +
##     brand_name, family = gaussian(link = "log"), data = train)
##
## Coefficients:
##                             Estimate Std. Error t value Pr(>|t|)
## (Intercept)               -8.799e-01  6.141e-01  -1.433 0.152308
## osAndroid                  2.946e-02  5.510e-01   0.053 0.957365
## osiOS                      9.258e-01  1.842e+00   0.503 0.615330
## sellers_amount            -2.654e-03  6.690e-04  -3.967 7.91e-05 ***
## screen_size                5.843e-01  4.738e-02  12.333  < 2e-16 ***
## memory_size                1.319e-03  7.966e-05  16.561  < 2e-16 ***
## battery_size              -8.886e-05  2.385e-05  -3.726 0.000208 ***
## month                     -1.341e-03  5.549e-03  -0.242 0.809138
## year                       1.336e-01  1.824e-02   7.326 5.70e-13 ***
## popularity_levelsmedium    4.978e-02  6.267e-02   0.794 0.427230
## popularity_levelshigh      9.465e-02  5.213e-02   1.816 0.069811 .
## popularity_levelsvery high  3.215e-01  5.299e-02   6.068 1.98e-09 ***
## brand_nameApple            2.012e-01  1.771e+00   0.114 0.909540
## brand_nameSamsung          4.249e-01  4.426e-02   9.600  < 2e-16 ***
## brand_nameXiaomi          -3.064e-01  8.135e-02  -3.766 0.000178 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 11480.23)
##
##     Null deviance: 37253913  on 828  degrees of freedom
## Residual deviance:  9344844  on 814  degrees of freedom
## AIC: 10119
##
## Number of Fisher Scoring iterations: 8
```

```r
y_hat <- exp(predict(glm_1, newdata = train))
pred <- exp(predict(glm_1, newdata = test))
results_glm1 <- c("RMSE"=RMSE(y_hat, train$best_price), "R2"=R2(y_hat, train$best_price),
                  "RMSE_test"=RMSE(pred, test$best_price), "R2_test"=R2(pred, test$best_price),
                  "AIC"=AIC(glm_1), "BIC"= BIC(glm_1), "Time"=end-start)

results_glm1
```

```
##         RMSE           R2    RMSE_test      R2_test          AIC          BIC
## 1.061717e+02 7.491640e-01 1.256508e+02 7.430499e-01 1.011927e+04 1.019479e+04
##         Time
```

```
## 1.695323e-02
```

We then removed the non-significant variables and tried to fit the model again.

```r
# Gaussian with log link function (removed vars)
start <- Sys.time()
glm_2 <- glm(best_price ~sellers_amount + screen_size + memory_size + battery_size +
             year + popularity_levels + brand_name,
             data = train, family = gaussian(link = "log"))
end <- Sys.time()

summary(glm_2)
```

```
##
## Call:
## glm(formula = best_price ~ sellers_amount + screen_size + memory_size +
##     battery_size + year + popularity_levels + brand_name, family = gaussian(link = "log"),
##     data = train)
##
## Coefficients:
##                             Estimate Std. Error t value Pr(>|t|)
## (Intercept)               -8.592e-01  3.282e-01  -2.618 0.009001 **
## sellers_amount            -2.655e-03  6.677e-04  -3.977 7.61e-05 ***
## screen_size                5.816e-01  4.389e-02  13.252  < 2e-16 ***
## memory_size                1.323e-03  7.822e-05  16.914  < 2e-16 ***
## battery_size              -8.845e-05  2.373e-05  -3.726 0.000208 ***
## year                       1.344e-01  1.775e-02   7.572 9.95e-14 ***
## popularity_levelsmedium    4.850e-02  6.231e-02   0.778 0.436585
## popularity_levelshigh      9.312e-02  5.157e-02   1.806 0.071329 .
## popularity_levelsvery high  3.210e-01  5.282e-02   6.077 1.87e-09 ***
## brand_nameApple            1.094e+00  5.230e-02  20.914  < 2e-16 ***
## brand_nameSamsung          4.283e-01  4.167e-02  10.279  < 2e-16 ***
## brand_nameXiaomi          -3.061e-01  8.119e-02  -3.770 0.000175 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 11440.47)
##
##     Null deviance: 37253913  on 828  degrees of freedom
## Residual deviance:  9346817  on 817  degrees of freedom
## AIC: 10113
##
## Number of Fisher Scoring iterations: 8
```

```r
y_hat <- exp(predict(glm_2, newdata = train))
pred <- exp(predict(glm_2, newdata = test))

results_glm2 <- c("RMSE"=RMSE(y_hat, train$best_price), "R2"=R2(y_hat, train$best_price),
                  "RMSE_test"=RMSE(pred, test$best_price), "R2_test"=R2(pred, test$best_price),
                  "AIC"=AIC(glm_2), "BIC"= BIC(glm_2), "Time"=end-start)

results_glm2
```

```
##         RMSE           R2    RMSE_test      R2_test          AIC          BIC
## 1.061829e+02 7.491129e-01 1.254607e+02 7.440790e-01 1.011344e+04 1.017480e+04
##         Time
```

```
## 1.258206e-02
```

We now try a Gamma family with an identity link function since the variable `best_price` only takes positive values.

```r
# Gamma with identity link function
start <- Sys.time()
glm_3 <- glm(best_price ~ screen_size + memory_size + battery_size + popularity_levels + brand_name,
             data = train, family = Gamma(link = "identity"))
end <- Sys.time()

summary(glm_3)
```

```
##
## Call:
## glm(formula = best_price ~ screen_size + memory_size + battery_size +
##     popularity_levels + brand_name, family = Gamma(link = "identity"),
##     data = train)
##
## Coefficients:
##                               Estimate Std. Error t value Pr(>|t|)
## (Intercept)                  -1.146567   2.050390  -0.559  0.57618
## screen_size                   5.960672   0.978836   6.090 1.74e-09 ***
## memory_size                   1.460271   0.062690  23.293  < 2e-16 ***
## battery_size                  0.001992   0.000712   2.798  0.00526 **
## popularity_levelsmedium       1.427164   1.382100   1.033  0.30209
## popularity_levelshigh         2.728763   1.825022   1.495  0.13525
## popularity_levelsvery high  12.170659   5.520270   2.205  0.02775 *
## brand_nameApple            215.188935  23.750394   9.060  < 2e-16 ***
## brand_nameSamsung           62.837848  11.388822   5.518 4.61e-08 ***
## brand_nameXiaomi           -16.901760   6.412156  -2.636  0.00855 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for Gamma family taken to be 0.2271516)
##
##     Null deviance: 928.26  on 828  degrees of freedom
## Residual deviance: 156.50  on 819  degrees of freedom
## AIC: 8830
##
## Number of Fisher Scoring iterations: 9
```

```r
y_hat <- predict(glm_3, newdata = train)
pred <- predict(glm_3, newdata = test)
results_glm3 <- c("RMSE"=RMSE(y_hat, train$best_price), "R2"=R2(y_hat, train$best_price),
                  "RMSE_test"=RMSE(pred, test$best_price), "R2_test"=R2(pred, test$best_price),
                  "AIC"=AIC(glm_3), "BIC"= BIC(glm_3), "Time"=end-start)

results_glm3
```

```
##         RMSE           R2    RMSE_test      R2_test          AIC          BIC
## 1.256040e+02 6.553434e-01 1.525585e+02 6.304260e-01 8.829985e+03 8.881908e+03
##         Time
## 1.431394e-02
```

In search for better results we also tried a Gamma family with a logarithmic link function.

```r
# Gamma with log link function
start <- Sys.time()
glm_4 <- glm(best_price ~ os +screen_size + log_memory_size +  popularity_levels +
             brand_name , data = train, family = Gamma(link = "log"))
end <- Sys.time()

summary(glm_4)
```

```
##
## Call:
## glm(formula = best_price ~ os + screen_size + log_memory_size +
##     popularity_levels + brand_name, family = Gamma(link = "log"),
##     data = train)
##
## Coefficients:
##                             Estimate Std. Error t value Pr(>|t|)
## (Intercept)                 -0.14946    0.13444  -1.112 0.266581
## osAndroid                   -2.32493    0.18316 -12.694  < 2e-16 ***
## osiOS                       -1.24071    0.53985  -2.298 0.021800 *
## screen_size                  0.19918    0.04145   4.806 1.83e-06 ***
## log_memory_size              0.32849    0.01797  18.280  < 2e-16 ***
## popularity_levelsmedium     -0.08095    0.05189  -1.560 0.119099
## popularity_levelshigh        0.01132    0.05269   0.215 0.829935
## popularity_levelsvery high  0.17656    0.05794   3.047 0.002383 **
## brand_nameApple             -0.20813    0.53942  -0.386 0.699723
## brand_nameSamsung            0.37917    0.05808   6.528 1.17e-10 ***
## brand_nameXiaomi            -0.23418    0.06437  -3.638 0.000292 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for Gamma family taken to be 0.2753884)
##
##     Null deviance: 928.26  on 828  degrees of freedom
## Residual deviance: 167.49  on 818  degrees of freedom
## AIC: 8890.1
##
## Number of Fisher Scoring iterations: 7
```

```r
y_hat <- exp(predict(glm_4, newdata = train))
pred <- exp(predict(glm_4, newdata = test))

results_glm4 <- c("RMSE"=RMSE(y_hat, train$best_price), "R2"=R2(y_hat, train$best_price),
                  "RMSE_test"=RMSE(pred, test$best_price), "R2_test"=R2(pred, test$best_price),
                  "AIC"=AIC(glm_4), "BIC"= BIC(glm_4), "Time"=end-start)

results_glm4
```

```
##         RMSE           R2    RMSE_test      R2_test          AIC          BIC
## 1.033836e+02 7.647832e-01 1.322352e+02 7.102159e-01 8.890094e+03 8.946737e+03
##         Time
## 1.406407e-02
```

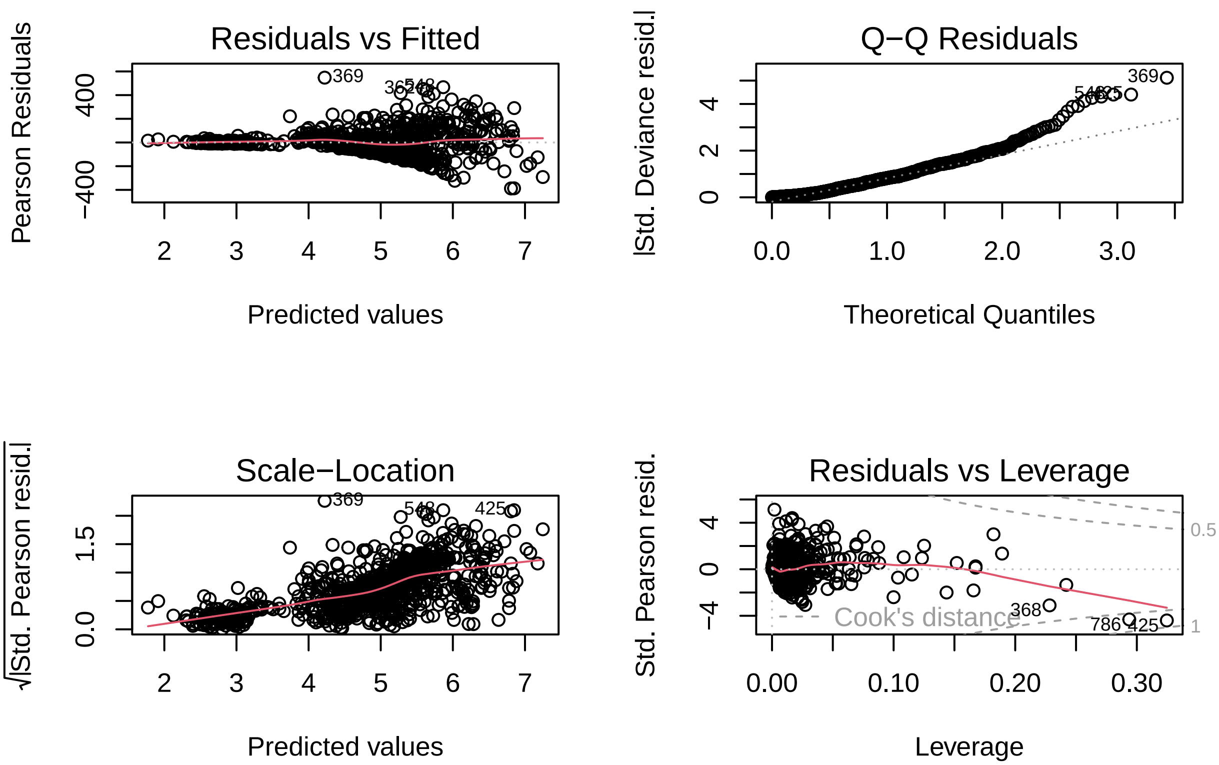
We compare the results of the four models

```r
glm_results <- rbind(results_glm1, results_glm2, results_glm3, results_glm4)
rownames(glm_results) <- c("GLM 1", "GLM 2", "GLM 3", "GLM 4")

glm_results
```

```
##             RMSE        R2 RMSE_test   R2_test       AIC       BIC       Time
## GLM 1 106.1717 0.7491640  125.6508 0.7430499 10119.266 10194.789 0.01695323
## GLM 2 106.1829 0.7491129  125.4607 0.7440790 10113.441 10174.803 0.01258206
## GLM 3 125.6040 0.6553434  152.5585 0.6304260  8829.985  8881.908 0.01431394
## GLM 4 103.3836 0.7647832  132.2352 0.7102159  8890.094  8946.737 0.01406407
```

From the results we observe that the `GLM 4` model performs best overall when considering the performance indexes. Of this model we plot the residuals

```r
par(mfrow=c(2,2))
plot(glm_2)
```



The residuals of the model with gaussian family and log link function are not normally distributed and exhibit a strong heteroschedasticity. We also observe two strong leverage points. However the R2 and RMSE performance indexes are slightly better when compared to linear models.

Non-linear models

We decide to explore if non-linear models can better capture the relationship between the predictors and the response variable. We will therefore use the Generalized Additive Models (GAM) and the Random Forest (RF) algorithms.

Generalized Additive Models (GAM)

We start by fitting a GAM model with splines on the continuous predictors. We will use the same predictors as in the linear models.

```
start <- Sys.time()
fit_gam1 <- gam(best_price ~ os + s(screen_size) + s(memory_size) + s(battery_size)
                + s(sellers_amount) + s(month)+ year+ popularity_levels +
                  brand_name, data = train, method = "REML")
end <- Sys.time()

summary(fit_gam1)
```

```
##
## Family: gaussian
## Link function: identity
##
## Formula:
## best_price ~ os + s(screen_size) + s(memory_size) + s(battery_size) +
##     s(sellers_amount) + s(month) + year + popularity_levels +
##     brand_name
##
## Parametric coefficients:
##                              Estimate Std. Error t value Pr(>|t|)
## (Intercept)                   -77.586     93.688  -0.828 0.407840
## osAndroid                      10.057     81.414   0.124 0.901716
## osiOS                          18.343     97.569   0.188 0.850927
## year                           12.220      3.332   3.667 0.000262 ***
## popularity_levelsmedium        -6.705      9.772  -0.686 0.492840
## popularity_levelshigh          -3.042     10.469  -0.291 0.771452
## popularity_levelsvery high     29.565     12.773   2.315 0.020887 *
## brand_nameApple               259.771    125.857   2.064 0.039341 *
## brand_nameSamsung              68.126     11.618   5.864 6.63e-09 ***
## brand_nameXiaomi              -36.186     12.305  -2.941 0.003369 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##                     edf Ref.df       F  p-value
## s(screen_size)    8.097  8.780  16.289  < 2e-16 ***
## s(memory_size)    3.959  4.495 132.067  < 2e-16 ***
## s(battery_size)   5.637  6.805   2.819  0.00966 **
## s(sellers_amount) 6.038  7.059   4.751 2.97e-05 ***
## s(month)          1.784  2.225   1.074  0.32600
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.796   Deviance explained = 80.5%
## -REML = 4936.8  Scale est. = 9165.3    n = 829
```

```
y_hat <- predict(fit_gam1, newdata = train)
pred <- predict(fit_gam1, newdata = test)
results1 <- c("RMSE"=RMSE(y_hat, train$best_price), "R2"=R2(y_hat, train$best_price),
              "RMSE_test"=RMSE(pred, test$best_price), "R2_test"=R2(pred, test$best_price),
              "AIC"=AIC(fit_gam1), "BIC"= BIC(fit_gam1),  "Time"=end-start)
```

```r
results1
```

```
##         RMSE           R2    RMSE_test      R2_test          AIC          BIC
## 9.366259e+01 8.048163e-01 1.069968e+02 8.095120e-01 9.960149e+03 1.015068e+04
##         Time
## 7.346454e-01
```

We can remove the non significant variables from the model, in this case `os`. Regarding the splines we can see that the `month` variable has a linear effect on the model, so we can remove the spline term from it.

```r
start <- Sys.time()
fit_gam2 <-gam(best_price ~  s(screen_size) + s(memory_size) + s(battery_size) +
                 s(sellers_amount) + year + brand_name,
               data = train, method = "REML")
end <- Sys.time()

summary(fit_gam2)
```

```
##
## Family: gaussian
## Link function: identity
##
## Formula:
## best_price ~ s(screen_size) + s(memory_size) + s(battery_size) +
##     s(sellers_amount) + year + brand_name
##
## Parametric coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)        -83.214     62.526  -1.331   0.1836
## year                13.153      3.267   4.026 6.21e-05 ***
## brand_nameApple    267.989     17.598  15.228  < 2e-16 ***
## brand_nameSamsung   74.216     11.265   6.588 8.08e-11 ***
## brand_nameXiaomi   -30.937     12.125  -2.551   0.0109 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##                     edf Ref.df       F  p-value
## s(screen_size)    8.166  8.804  16.339  < 2e-16 ***
## s(memory_size)    3.990  4.522 132.106  < 2e-16 ***
## s(battery_size)   5.723  6.895   3.000  0.00539 **
## s(sellers_amount) 6.217  7.226   5.844 1.82e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.794   Deviance explained = 80.1%
## -REML =   4966  Scale est. = 9248.2    n = 829
```

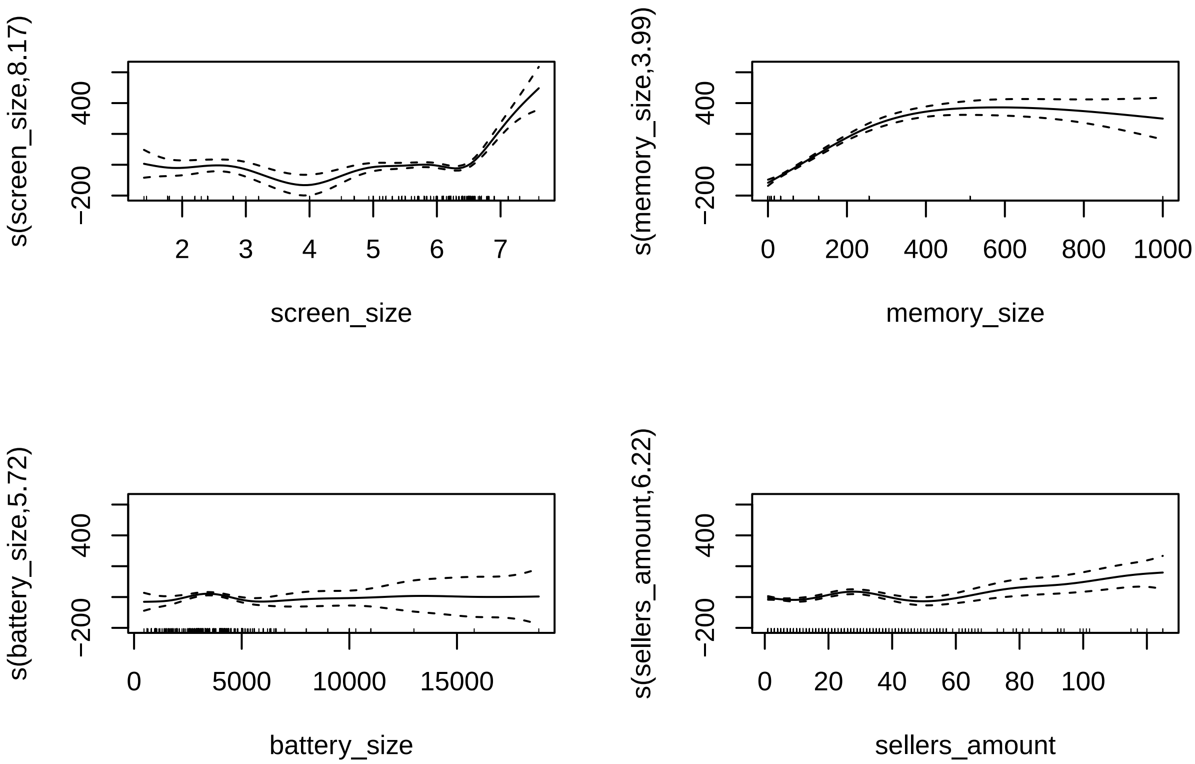
```r
y_hat <- predict(fit_gam2, newdata = train)
pred <- predict(fit_gam2, newdata = test)

results2 <- c("RMSE"=RMSE(y_hat, train$best_price), "R2"=R2(y_hat, train$best_price),
              "RMSE_test"=RMSE(pred, test$best_price), "R2_test"=R2(pred, test$best_price),
              "AIC"=AIC(fit_gam2), "BIC"= BIC(fit_gam2), "Time"=end-start)
```

```r
results2
```

```
##         RMSE           R2    RMSE_test      R2_test          AIC          BIC
## 9.446491e+01 8.014568e-01 1.080966e+02 8.057500e-01 9.958813e+03 1.011281e+04
##         Time
## 3.031719e-01
```

We plot the smooth terms to better understand the relationship between the predictors and the response variable.

```r
par(mfrow=c(2,2))
plot(fit_gam2)
```



The smooth terms for `screen_size`,`battery_size`, `memory_size` and `sellers_amount` show a clear non-linear relationship with the `best_price` variable, as confirmed by the edf values in the summary of the model.

Taking into consideration the important interactions, previously discovered in the linear models, we decided to include them in the GAM model.

```r
start <- Sys.time()
fit_gam3 <- gam(best_price ~  brand_name+ year+ s(screen_size, by = os)+ s(memory_size) +
                s(sellers_amount), data = train, method = "REML")
end <- Sys.time()

summary(fit_gam3)
```

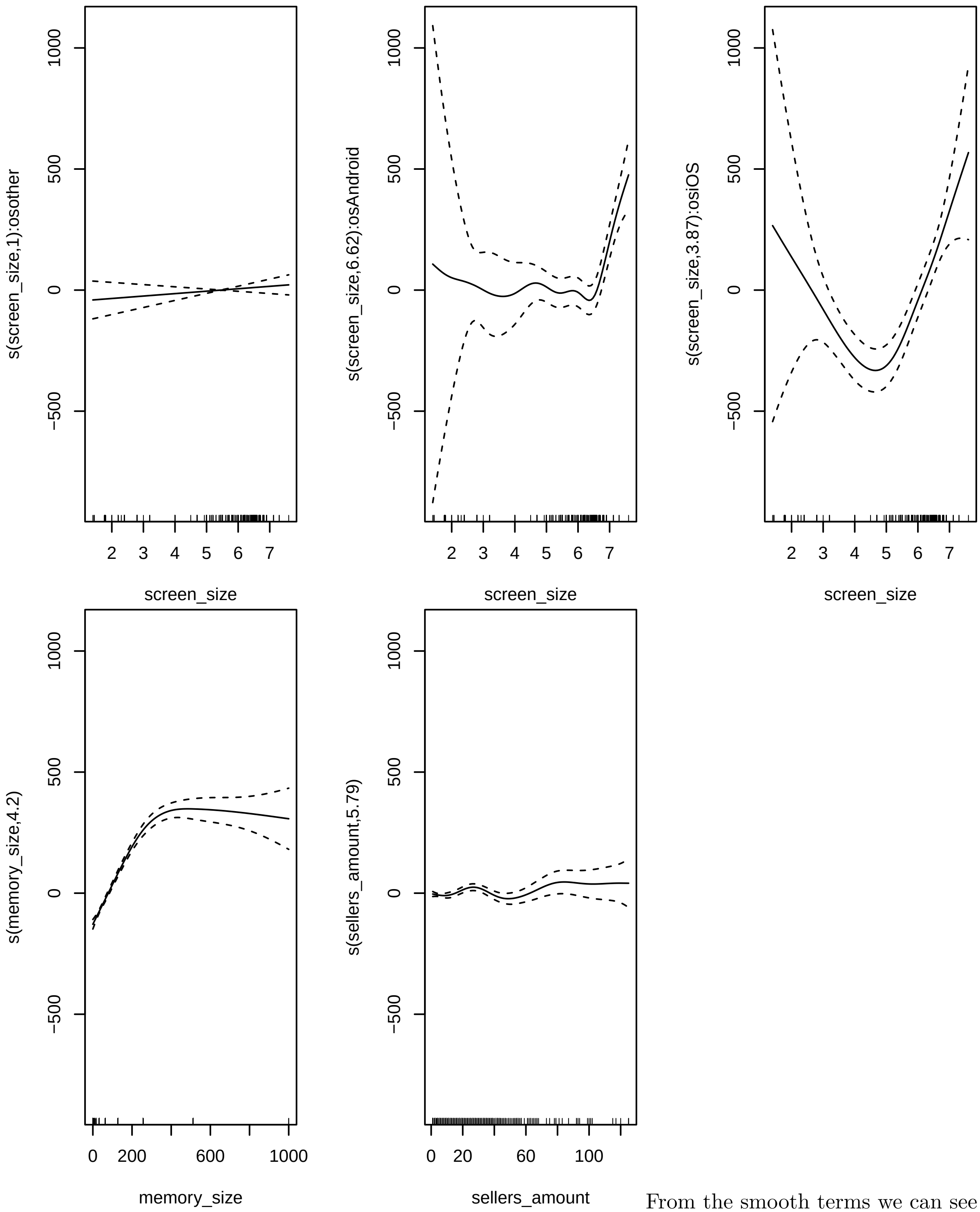
```
##
## Family: gaussian
## Link function: identity
##
## Formula:
## best_price ~ brand_name + year + s(screen_size, by = os) + s(memory_size) +
##     s(sellers_amount)
```

```
##
## Parametric coefficients:
##                    Estimate Std. Error t value Pr(>|t|)
## (Intercept)          54.987     65.449   0.840  0.40107
## brand_nameApple     334.115     50.192   6.657 5.19e-11 ***
## brand_nameSamsung    87.255     10.349   8.431  < 2e-16 ***
## brand_nameXiaomi    -30.542     11.130  -2.744  0.00621 **
## year                  6.399      3.035   2.109  0.03527 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##                              edf Ref.df       F p-value
## s(screen_size):osother     1.002  1.004   1.083 0.29745
## s(screen_size):osAndroid 6.620  7.239  20.819 < 2e-16 ***
## s(screen_size):osiOS     3.869  4.591  38.856 < 2e-16 ***
## s(memory_size)           4.203  4.708 160.751 < 2e-16 ***
## s(sellers_amount)        5.790  6.814   2.807 0.00696 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.823   Deviance explained = 82.9%
## -REML = 4891.5  Scale est. = 7958.9    n = 829
```

```
y_hat <- predict(fit_gam3, newdata = train)
pred <- predict(fit_gam3, newdata = test)
results3 <- c("RMSE"=RMSE(y_hat, train$best_price), "R2"=R2(y_hat, train$best_price),
              "RMSE_test"=RMSE(pred, test$best_price), "R2_test"=R2(pred, test$best_price),
              "AIC"=AIC(fit_gam3), "BIC"= BIC(fit_gam3),  "Time"=end-start)

results3
```

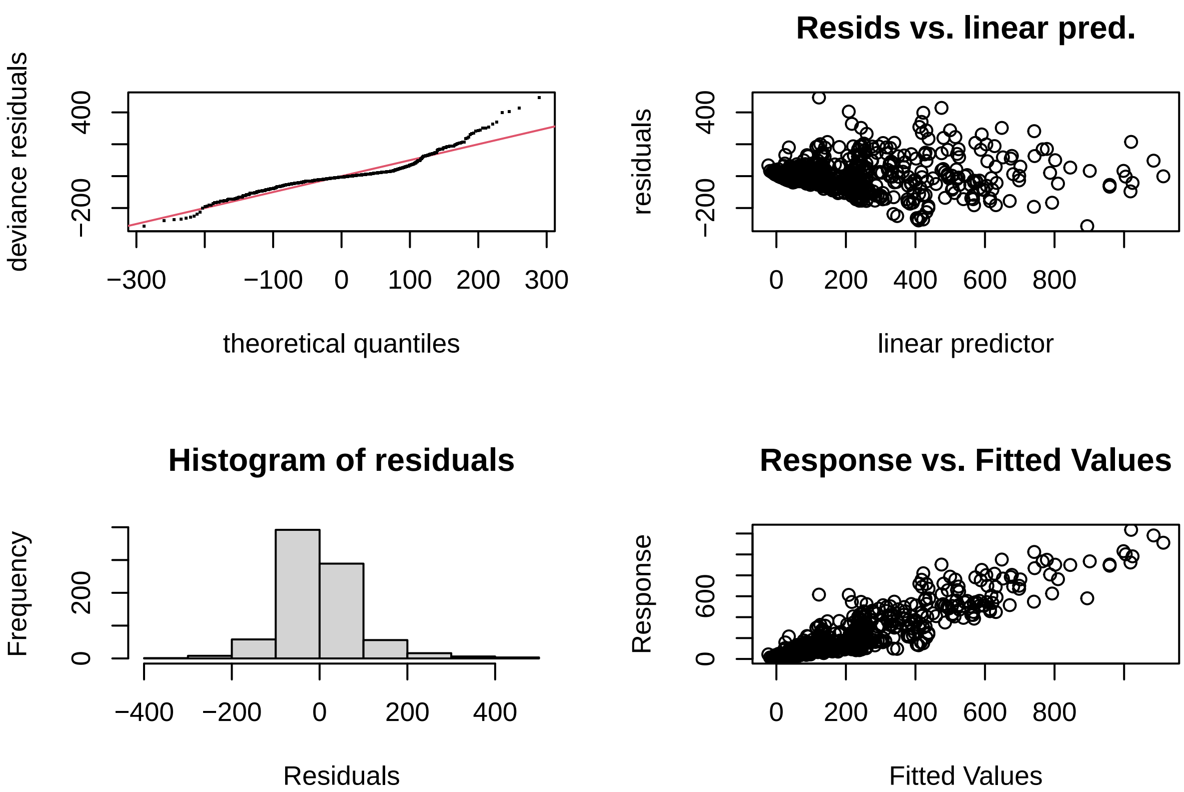
```
##         RMSE           R2     RMSE_test      R2_test          AIC          BIC
##   87.7762055    0.8285656  107.1938184    0.8088495 9830.5044610 9969.0471659
##         Time
##    0.2435613
```

```
par(mfrow=c(1,3))
plot(fit_gam3)
```



From the smooth terms we can see that `screen_size` is different for each `os` level. The only `os` level that exhibits a linear behaviour is `other`, we still decide to keep the spline on this interaction in the model since the non-linearity is strong in the other two cases. For the `memory_size` and `sellers_amount` we can see that the non-linear effect is present, therefore we decide to keep the smooth terms in the model.

```
par(mfrow=c(2,2))
gam.check(fit_gam3)
```




```
##
## Method: REML   Optimizer: outer newton
## full convergence after 5 iterations.
## Gradient range [-0.0009907786,0.0001142986]
## (score 4891.547 & scale 7958.927).
## Hessian positive definite, eigenvalue range [0.0009889589,409.5447].
## Model rank =  50 / 50
##
## Basis dimension (k) checking results. Low p-value (k-index<1) may
## indicate that k is too low, especially if edf is close to k'.
##
##                              k'  edf k-index p-value
## s(screen_size):osother    9.00 1.00    0.93   0.020 *
## s(screen_size):osAndroid 9.00 6.62    0.93   0.015 *
## s(screen_size):osiOS     9.00 3.87    0.93   0.015 *
## s(memory_size)           9.00 4.20    0.97   0.270
## s(sellers_amount)        9.00 5.79    1.06   0.960
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

From the residuals plots we can see that the residuals are not normally distributed and still exhibit heteroschedasticity. The R2 and RMSE performance indexes definitely improved when compared to the linear models.

```
gam_results <- rbind(results1, results2, results3)
rownames(gam_results) <- c("GAM 1", "GAM 2", "GAM 3")
round(gam_results, 3)
```

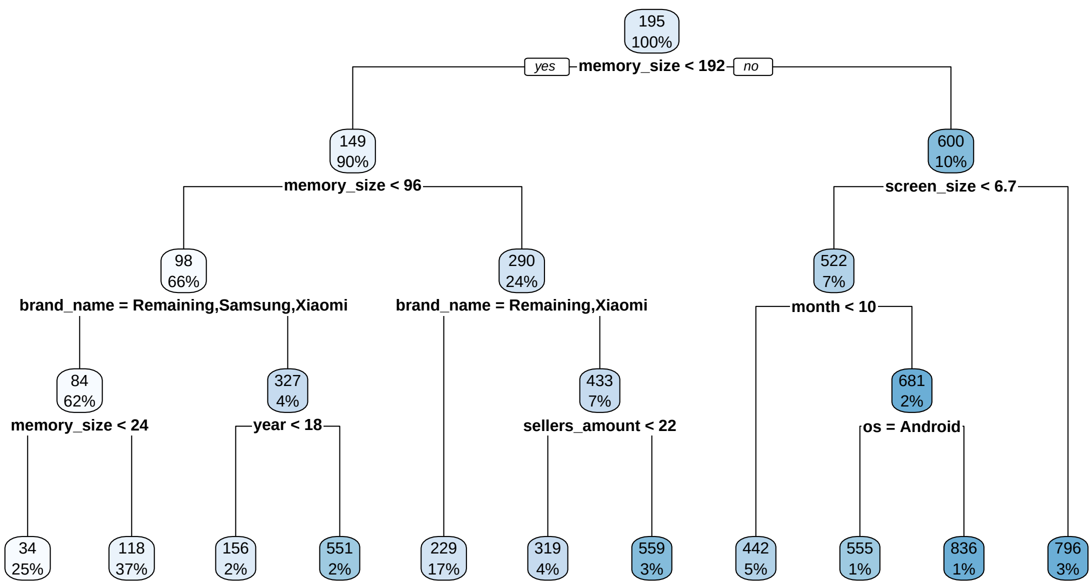
```
##         RMSE    R2 RMSE_test R2_test      AIC       BIC  Time
## GAM 1 93.663 0.805   106.997   0.810 9960.149 10150.678 0.735
```

```
## GAM 2 94.465 0.801   108.097   0.806 9958.813 10112.811 0.303
## GAM 3 87.776 0.829   107.194   0.809 9830.504  9969.047 0.244
```

Trees

We will now fit a decision tree model to the data. We will use the same predictors as in the linear models.

```r
start <- Sys.time()
tree_1 <- rpart(best_price ~ os + sellers_amount + screen_size +
                  memory_size + battery_size + month + year +
                  popularity_levels + brand_name , data = train)
end <- Sys.time()

rpart.plot(tree_1)
```




```r
y_hat <- predict(tree_1, newdata = train)
pred <- predict(tree_1, newdata = test)

results_tree1 <- c("RMSE"=RMSE(y_hat, train$best_price), "R2"=R2(y_hat, train$best_price),
                   "RMSE_test"=RMSE(pred, test$best_price), "R2_test"=R2(pred, test$best_price),
                   "AIC"=NA, "BIC"= NA, "Time"=end-start)

results_tree1
```

```
##         RMSE           R2    RMSE_test      R2_test          AIC          BIC
##  96.88328296   0.79112795 126.82073514   0.73204428           NA           NA
##         Time
##   0.01530981
```

The results show little if any improvement when compared to the previous models. From the tree model we confirm the central role of the `memory_size`, `brand_name` and `screen_size` variables for predicting `best_price`.

Bagging

We now fit a bagging model to the data. We use all the available predictors. The hyperparameters are chosen based on some empirical trials.

```r
start <- Sys.time()
bag <- bagging(best_price ~ os + sellers_amount + screen_size + memory_size +
    battery_size + month + year + popularity_levels + brand_name,
  data=train,
  nbagg=150,
  coob=TRUE,
  control=rpart.control(cp=0.001, minsplit = 2, maxdepth = 5)
)
end <- Sys.time()

y_hat <- predict(bag, newdata = train)
pred <- predict(bag, newdata = test)

results_bag <- c("RMSE"=RMSE(y_hat, train$best_price), "R2"=R2(y_hat, train$best_price),
                 "RMSE_test"=RMSE(pred, test$best_price), "R2_test"=R2(pred, test$best_price),
                 "AIC"=NA, "BIC"= NA, "Time"=end-start)

results_bag
```

```
##        RMSE          R2   RMSE_test     R2_test         AIC         BIC
##  73.0943461   0.8840002 111.2339460   0.7964865          NA          NA
##        Time
##   1.6241283
```

The bagging model fits the data almost as well as the best previous model, but takes much more time to run.

Boosting

We now fit a boosting model to the data. We use all the available predictors. The hyperparameters are chosen based on some empirical trials.

```r
library(gbm)

start <- Sys.time()
boost <- gbm(
  best_price ~ os + sellers_amount + screen_size + memory_size +
    battery_size + month + year + popularity_levels + brand_name,
  distribution="gaussian",
  data=train,
  bag.fraction = 1,
  interaction.depth = 4,
  shrinkage = 0.01,
  n.trees=1000)
end <- Sys.time()

y_hat <- predict(boost, newdata = train)
pred <- predict(boost, newdata = test)

results_boost <- c("RMSE"=RMSE(y_hat, train$best_price), "R2"=R2(y_hat, train$best_price),
                   "RMSE_test"=RMSE(pred, test$best_price), "R2_test"=R2(pred, test$best_price),
                   "AIC"=NA, "BIC"= NA, "Time"=end-start)

results_boost
```

```
##        RMSE          R2   RMSE_test     R2_test         AIC         BIC
```

```
## 67.4279753   0.8994728 104.5962776   0.8214001          NA          NA
##        Time
##   0.2931027
```

The boosting model shows the best results so far, with a relatively low RMSE and a high R2 for both the training and the test set. The model takes a more time to run when compared to simpler models, but the results are worth it.

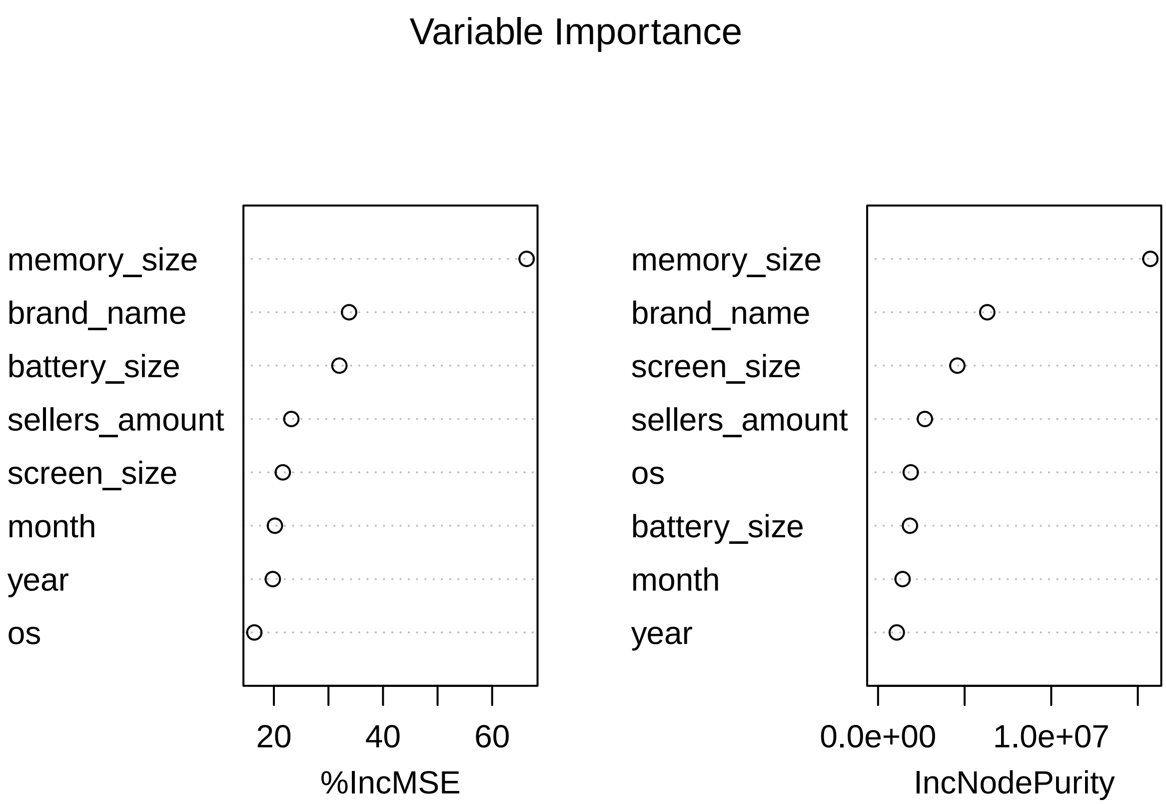
Random Forest

We now fit two random forest models to the data. In this first fit we apply a classical random forest model with all the available predictors setting the hyperparameters based on some empirical trials.

```
start <- Sys.time()
rf_1 <- randomForest(best_price ~ os + sellers_amount + screen_size +
                     memory_size + battery_size + month + year +
                    popularity_levels + brand_name,
                    data = train, importance=T, proximitry=T, mtry=5)
end <- Sys.time()

y_hat <- predict(rf_1, newdata = train)
pred <- predict(rf_1, newdata = test)

results_rf1 <- c("RMSE"=RMSE(y_hat, train$best_price), "R2"=R2(y_hat, train$best_price),
                 "RMSE_test"=RMSE(pred, test$best_price), "R2_test"=R2(pred, test$best_price),
                 "AIC"=NA, "BIC"= NA,  "Time"=end-start)

results_rf1
```

```
##        RMSE          R2   RMSE_test     R2_test         AIC         BIC
##  38.1805301   0.9705038 103.3818863   0.8286013          NA          NA
##        Time
##   2.0274677
```

```
varImpPlot(rf_1, sort=T, n.var=8, main="Variable Importance")
```



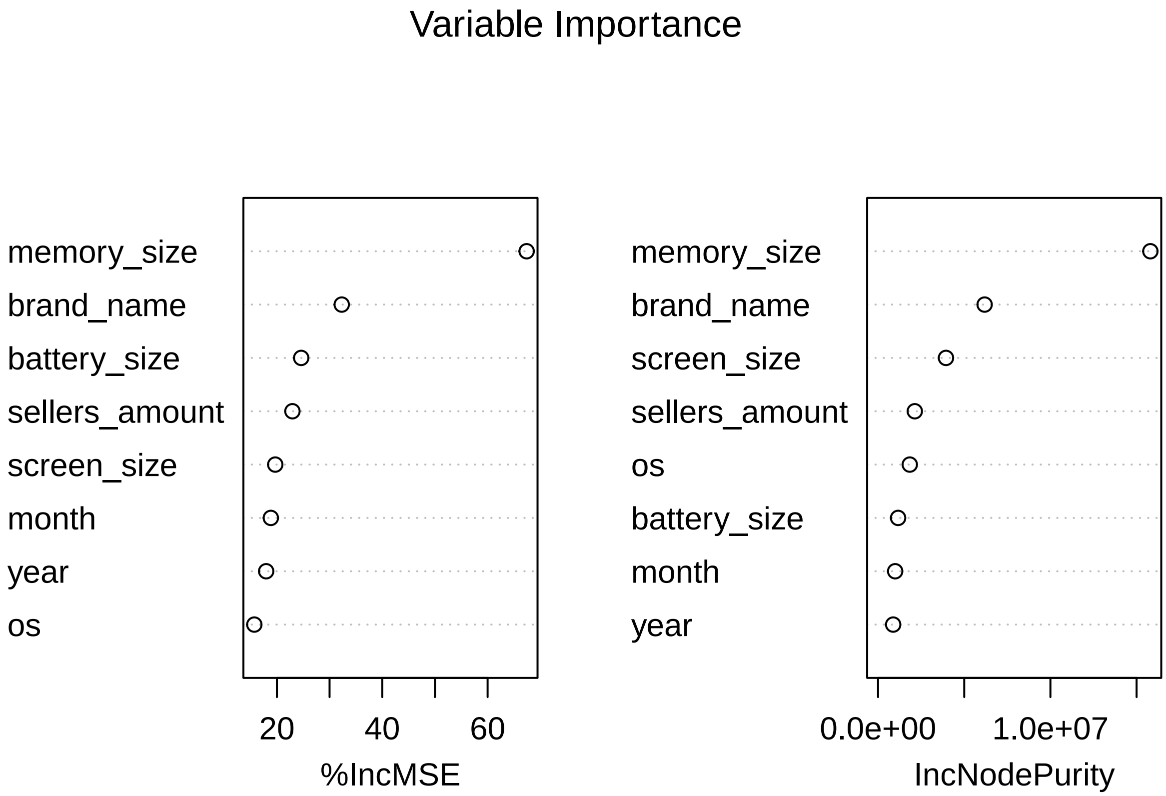
From the variable importance plots we can see that the `memory_size`, `brand_name` and `screen_size` variables are the most important for predicting `best_price`, a result consistent with our previous findings.

In the second random forest model we apply some pre-pruning, again the hyperparameters are chosen based on some empirical trials.

```
start <- Sys.time()
rf_2 <- randomForest(best_price ~ os + sellers_amount + screen_size +
                     memory_size + battery_size + month + year +
                     popularity_levels + brand_name,
                     data = train,
                     importance=T, proximitry=T, mtry=5, maxnodes=52)
end <- Sys.time()

y_hat <- predict(rf_2, newdata = train)
pred <- predict(rf_2, newdata = test)

results_rf2 <- c("RMSE"=RMSE(y_hat, train$best_price), "R2"=R2(y_hat, train$best_price),
                 "RMSE_test"=RMSE(pred, test$best_price), "R2_test"=R2(pred, test$best_price),
                 "AIC"=NA, "BIC"= NA,  "Time"=end-start)

results_rf2
```

```
##        RMSE          R2   RMSE_test     R2_test         AIC         BIC
##  63.7933263   0.9138173 106.2121372   0.8200838          NA          NA
##        Time
##   0.6641870
```

```r
varImpPlot(rf_2, sort=T, n.var=8, main="Variable Importance")
```



Again the variance importance plots show that the `memory_size`, `brand_name` and `screen_size` variables are the most important for predicting `best_price`.

```r
non_linear_results <- rbind(results1, results2, results3, results_tree1, results_bag, results_boost, res
rownames(non_linear_results) <- c("GAM 1", "GAM 2", "GAM 3", "Tree 1",
                                  "Bagging", "Boosting", "RF 1", "RF 2")
colnames(non_linear_results) <- c("RMSE train", "R2 train", "RMSE test",
                                  "R2 test", "AIC", "BIC", "Time")
round(non_linear_results, 3)
```

```
##          RMSE train R2 train RMSE test R2 test      AIC       BIC  Time
## GAM 1        93.663    0.805   106.997   0.810 9960.149 10150.678 0.735
## GAM 2        94.465    0.801   108.097   0.806 9958.813 10112.811 0.303
## GAM 3        87.776    0.829   107.194   0.809 9830.504  9969.047 0.244
## Tree 1       96.883    0.791   126.821   0.732       NA        NA 0.015
## Bagging      73.094    0.884   111.234   0.796       NA        NA 1.624
## Boosting     67.428    0.899   104.596   0.821       NA        NA 0.293
## RF 1         38.181    0.971   103.382   0.829       NA        NA 2.027
## RF 2         63.793    0.914   106.212   0.820       NA        NA 0.664
```

From this table the random forest model without pre-pruning seem to show the best results, with the lowest RMSE and the highest R2 for both the training and the test set. However a R2 of 97.1% is an extremely high value and may indicate overfitting. This led us to choosing the boosting model and the third GAM model as the candidate representatives for the best non-linear models.

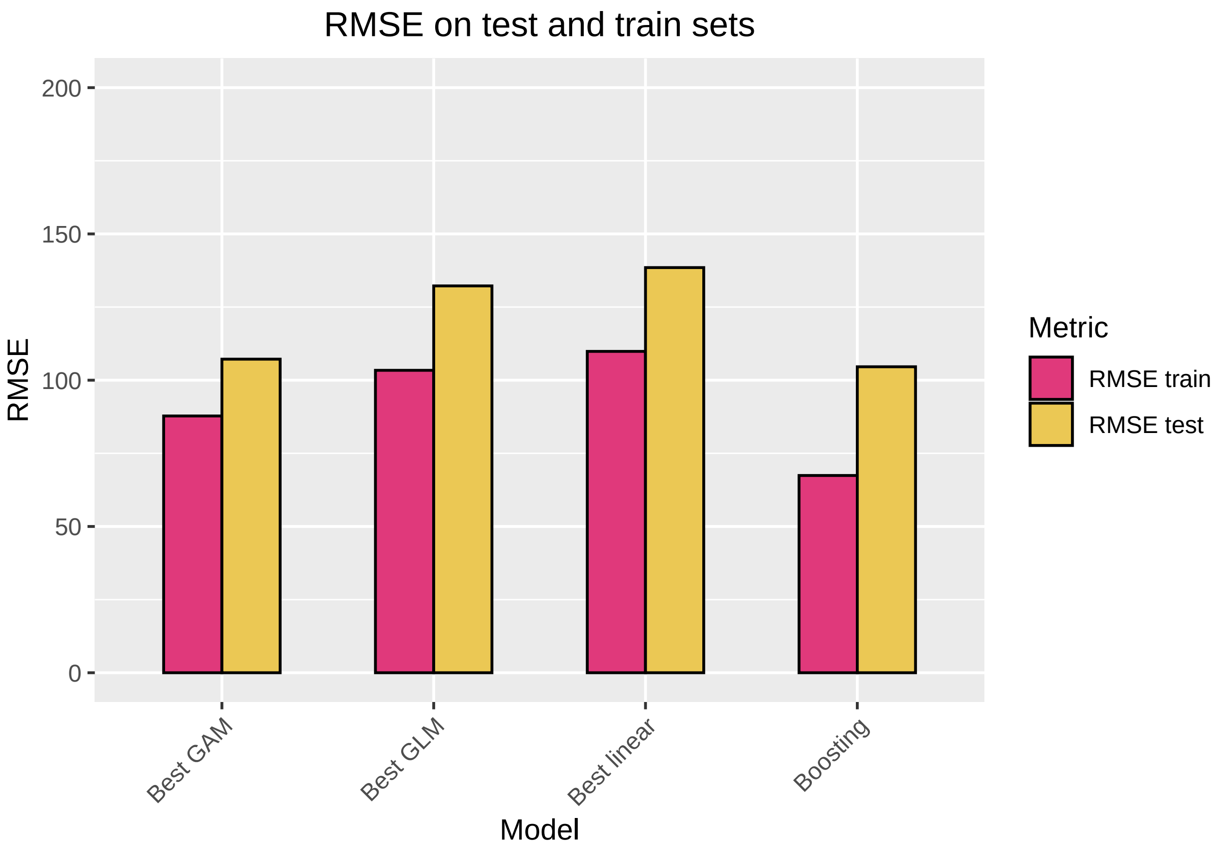
```r
best_models_results <- rbind(result4, results_glm4, results3, results_boost)

rownames(best_models_results) <- c("Best linear", "Best GLM", "Best GAM", "Boosting")
colnames(best_models_results) <- c("RMSE train", "R2 train", "RMSE test",
```

```
                                    "R2 test", "AIC", "BIC", "Time")

best_models_results
```

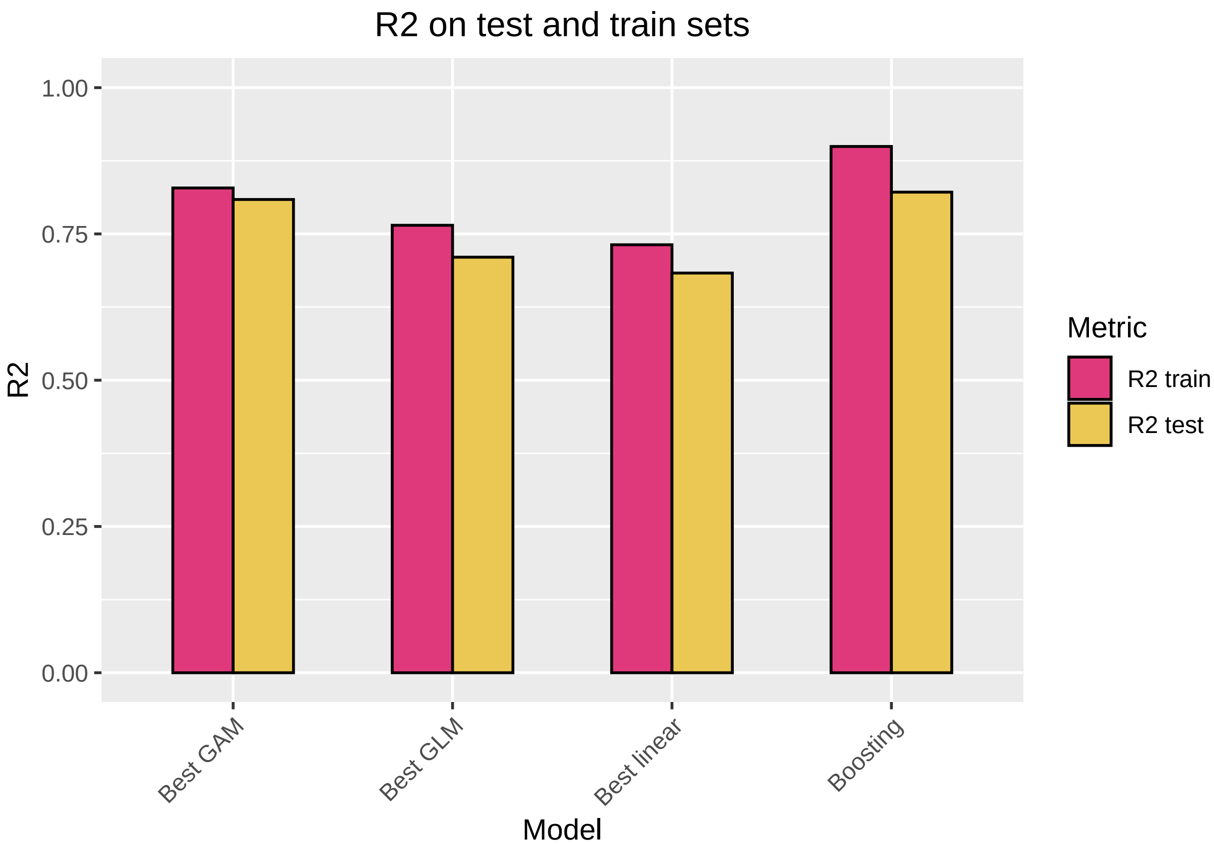
```
##              RMSE train  R2 train RMSE test   R2 test       AIC       BIC
## Best linear  109.84725 0.7314897  138.4910 0.6831086 10171.693 10237.776
## Best GLM     103.38364 0.7647832  132.2352 0.7102159  8890.094  8946.737
## Best GAM      87.77621 0.8285656  107.1938 0.8088495  9830.504  9969.047
## Boosting      67.42798 0.8994728  104.5963 0.8214001        NA        NA
##                     Time
## Best linear 0.006441116
## Best GLM    0.014064074
## Best GAM    0.243561268
## Boosting    0.293102741
```

We now compare the best results we have obtained so far for each class of model

```
# Convert the data frame to a long format
df_long <- data.frame(Model = rep(rownames(best_models_results), each = 2),
                      Metric = c("RMSE train", "RMSE test", "RMSE train", "RMSE test",
                                 "RMSE train", "RMSE test", "RMSE train", "RMSE test"),
                      Value = c(c(best_models_results[1,"RMSE test"], best_models_results[1,"RMSE train
                      c(best_models_results[2,"RMSE test"], best_models_results[2,"RMSE train"]),
                      c(best_models_results[3,"RMSE test"], best_models_results[3,"RMSE train"]),
                      c(best_models_results[4,"RMSE test"], best_models_results[4,"RMSE train"])))

df_long$Metric <- factor(df_long$Metric, levels = c("RMSE train", "RMSE test"))


# Create the bar plot
ggplot(df_long, aes(x = Model, y = Value, fill = Metric)) +
  geom_bar(stat = "identity", position = "dodge", color = "black", width=0.55) +
  scale_fill_manual(values = c("RMSE train" = "#e0397b", "RMSE test" = "#ebc854")) +
  labs(title = "RMSE on test and train sets", x = "Model", y = "RMSE") +
  ylim(0, 200) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1), plot.title = element_text(hjust = 0.5))
```




```r
df_long <- data.frame(Model = rep(rownames(best_models_results), each = 2),
                      Metric = c("R2 train", "R2 test", "R2 train", "R2 test",
                                 "R2 train", "R2 test", "R2 train", "R2 test"),
                      Value = c(c(best_models_results[1,"R2 test"], best_models_results[1,"R2 train"]),
                      c(best_models_results[2,"R2 test"], best_models_results[2,"R2 train"]),
                      c(best_models_results[3,"R2 test"], best_models_results[3,"R2 train"]),
                      c(best_models_results[4,"R2 test"], best_models_results[4,"R2 train"])))

df_long$Metric <- factor(df_long$Metric, levels = c("R2 train", "R2 test"))

# Create the bar plot

ggplot(df_long, aes(x = Model, y = Value, fill = Metric)) +
  geom_bar(stat = "identity", position = "dodge", color = "black", width=0.55) +
  scale_fill_manual(values = c("R2 train" = "#e0397b", "R2 test" = "#ebc854")) +
  labs(title = "R2 on test and train sets", x = "Model", y = "R2") +
  ylim(0, 1) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1), plot.title = element_text(hjust = 0.5))
```



The plots show that the boosting model has the lowest RMSE and the highest R2 for both the test and the training sets. With this we conclude our report choosing the boosting model as the best model overall for predicting the `best_price` variable.