Datacamp Cert Project w/ Text

Connor Hanna

2022-11-10

Data Scientist Associate Case Study

Company Background

EMO is a manufacturer of motorcycles. The company successfully launched its first electric moped in India in 2019. The product team knows how valuable owner reviews are in making improvements to their mopeds.

Unfortunately they often get reviews from people who never owned the moped. They don't want to consider this feedback, so would like to find a way to identify reviews from these people. They have obtained data from other mopeds, where they know if the reviewer owned the moped or not. They think this is equivalent to their own reviews.

Customer Question

Your manager has asked you to answer the following: - Can you predict which reviews come from people who have never owned the moped before?

Dataset

The dataset contains reviews about other mopeds from a local website. The data you will use for this analysis can be accessed here: `"data/moped.csv"`

Column Name	Criteria
Used it for	Character, the purpose of the electric moped for the user, one of "Commuting", "Leisure".
Owned for	Character, duration of ownership of vehicle one of "<= 6 months", "> 6 months", "Never Owned". Rows that indicate ownership should be combined into the category "Owned".
Model name	Character, the name of the electric moped.
Visual Appeal	Numeric, visual appeal rating (on a 5 point scale, replace missing values with 0).
Reliability	Numeric, reliability rating (on a 5 point scale, replace missing values with 0).
Extra Feature	Numeric, extra feature rating (on a 5 point scale, replace missing values with 0).
Comfort	Numeric, comfort rating (on a 5 point scale, replace missing values with 0).
Maintenance cost	Numeric, maintenance cost rating (on a 5 point scale, replace missing values with 0).

Column Name	Criteria
Value for money	Numeric, value for money rating (on a 5 point scale, replace missing values with 0).

Data Scientist Associate Case Study Submission

Use this template to complete your analysis and write up your summary for submission.

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --
```

```
## v ggplot2 3.4.0     v purrr   0.3.4
## v tibble  3.1.8     v dplyr   1.0.7
## v tidyr   1.1.4     v stringr 1.4.0
## v readr   2.1.1     v forcats 0.5.1
```

```
## Warning: package 'ggplot2' was built under R version 4.1.3
```

```
## Warning: package 'tibble' was built under R version 4.1.3
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
## Warning: package 'ggthemes' was built under R version 4.1.3
```

```
## Warning: package 'caret' was built under R version 4.1.3
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:purrr':
##
##     lift
```

```
## Warning: package 'ggrepel' was built under R version 4.1.3
```

```
## Warning: package 'xgboost' was built under R version 4.1.3
```

```
##
## Attaching package: 'xgboost'
```

```
## The following object is masked from 'package:dplyr':
##
##     slice
```

```
## Warning: package 'rsample' was built under R version 4.1.3

## Warning: package 'ggforce' was built under R version 4.1.3

## Warning: package 'vtreat' was built under R version 4.1.3

## Loading required package: wrapr

## Warning: package 'wrapr' was built under R version 4.1.3

##
## Attaching package: 'wrapr'

## The following object is masked from 'package:dplyr':
##
##     coalesce

## The following objects are masked from 'package:tidyr':
##
##     pack, unpack

## The following object is masked from 'package:tibble':
##
##     view

## Warning: package 'WVPlots' was built under R version 4.1.3

## Warning: package 'pROC' was built under R version 4.1.3

## Type 'citation("pROC")' for a citation.

##
## Attaching package: 'pROC'

## The following object is masked from 'package:colorspace':
##
##     coords

## The following objects are masked from 'package:stats':
##
##     cov, smooth, var

## Rows: 713 Columns: 9
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (3): Used it for, Owned for, Model Name
## dbl (6): Visual Appeal, Reliability, Extra Features, Comfort, Maintenance co...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```r
# Inspecting data for variables to ensure we're not missing anything
head(moped)
```

```
## # A tibble: 6 x 9
##   `Used it for` Owned ~1 Model~2 Visua~3 Relia~4 Extra~5 Comfort Maint~6 Value~7
##   <chr>         <chr>    <chr>     <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
## 1 Commuting     Never o~ TVS iQ~       3       4      NA       4      NA       1
## 2 Leisure       > 6 mon~ TVS iQ~       3       1      NA       3      NA       3
## 3 Commuting     <= 6 mo~ TVS iQ~       4       4      NA       5      NA       2
## 4 Commuting     > 6 mon~ TVS iQ~       1       1      NA       1      NA       1
## 5 Commuting     > 6 mon~ TVS iQ~       3       4      NA       3      NA       2
## 6 Commuting     > 6 mon~ TVS iQ~       5       1      NA       5      NA       2
## # ... with abbreviated variable names 1: `Owned for`, 2: `Model Name`,
## #   3: `Visual Appeal`, 4: Reliability, 5: `Extra Features`,
## #   6: `Maintenance cost`, 7: `Value for Money`
```

```r
# all present and accounted for
# varnames all have quotation marks though, those are hideous

# renaming variables
moped <-
moped %>%
  mutate(used_for = `Used it for`,
         duration_owned = `Owned for`,
         model = `Model Name`,
         visual_appeal = `Visual Appeal`,
         extra_features = `Extra Features`,
         maint_cost = `Maintenance cost`,
         value = `Value for Money`,
         reliability = Reliability,
         comfort = Comfort,
         .keep = "unused")

# corrected some capital letters too. naming is now consistent.

# now to correct issues mentioned in the documentation
# "owned for" ownership column should be changed to indicate ownership as a dummy

moped <-
  moped %>%
  mutate(owned = ifelse(duration_owned == "Never owned", 0, 1), .keep = "unused")

# storing the original moped df for when I want the NAs for vtreat
moped_og <- moped

# checking other variables for NA values
# storing for use later validating manipulation prior to analysis
colSums(is.na(moped))
```

```
##       used_for          model  visual_appeal extra_features     maint_cost
##              0              0              0            530            537
##          value    reliability        comfort          owned
##            343              0            203              0
```

```r
# replacing NA values with 0

moped[is.na(moped)] <- 0

colSums(is.na(moped))
```

```
##       used_for          model  visual_appeal extra_features     maint_cost
##              0              0              0              0              0
##          value    reliability        comfort          owned
##              0              0              0              0
```

```r
# checking for entries outside expected values

summary(moped)
```

```
##    used_for            model           visual_appeal   extra_features
##  Length:713         Length:713         Min.   :1.000   Min.   :0.0000
##  Class :character   Class :character   1st Qu.:3.000   1st Qu.:0.0000
##  Mode  :character   Mode  :character   Median :4.000   Median :0.0000
##                                        Mean   :3.769   Mean   :0.7518
##                                        3rd Qu.:5.000   3rd Qu.:1.0000
##                                        Max.   :5.000   Max.   :5.0000
##    maint_cost         value        reliability       comfort
##  Min.   :0.0000   Min.   :0.000   Min.   :1.000   Min.   :0.000
##  1st Qu.:0.0000   1st Qu.:0.000   1st Qu.:2.000   1st Qu.:0.000
##  Median :0.0000   Median :1.000   Median :4.000   Median :3.000
##  Mean   :0.8373   Mean   :1.749   Mean   :3.314   Mean   :2.612
##  3rd Qu.:0.0000   3rd Qu.:4.000   3rd Qu.:5.000   3rd Qu.:5.000
##  Max.   :5.0000   Max.   :5.000   Max.   :5.000   Max.   :5.000
##      owned
##  Min.   :0.0000
##  1st Qu.:1.0000
##  Median :1.0000
##  Mean   :0.8107
##  3rd Qu.:1.0000
##  Max.   :1.0000
```

Data Validation

Describe the validation tasks you completed and what you found. Have you made any changes to the data to enable further analysis? Remember to describe what you did for every column in the data.

I begin the validation by calling `head()` to inspect the variables in the dataframe and make sure I'm not missing anything described in the documentation, and to confirm that the classes of the variables is as described. In some cases, the names were vague or otherwise inappropriate. To correct this, I called `mutate()` to rename the variables for convenience and clarity - in particular `"Owned for"`, where the original variable name was vague with respect to the contents of the variable. I specified the `.keep` argument to remove the original variables.

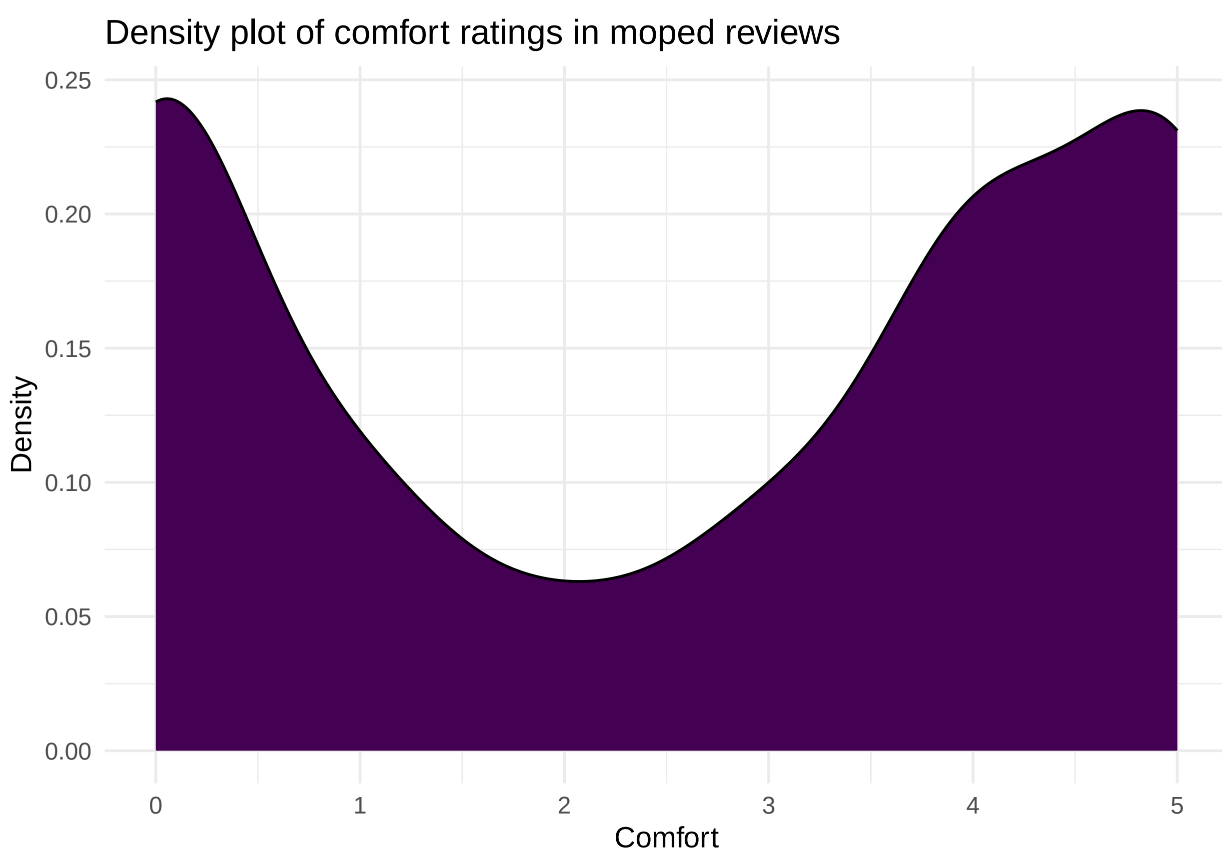
Then as specified in the documentation I used `mutate()` again to transform `duration_owned` into a binary variable reflecting ownership status. Once again I used `.keep` to remove the original variable. I saved this version of the dataframe for use later, since `vtreat` can transform `NA` values into useful binary indicators and synthesize approximate true values.

Then, since the documentation described `NA` values in multiple numerical variables, I checked the dataframe for them. After confirming they were only in the variables described, I used `is.na()` to replace them with 0 as instructed, and rechecked to make sure I hadn't left any stragglers.

I then checked for values outside the expected ranges of the numerical variables, found that there were none, and moved on to the exploratory analysis.

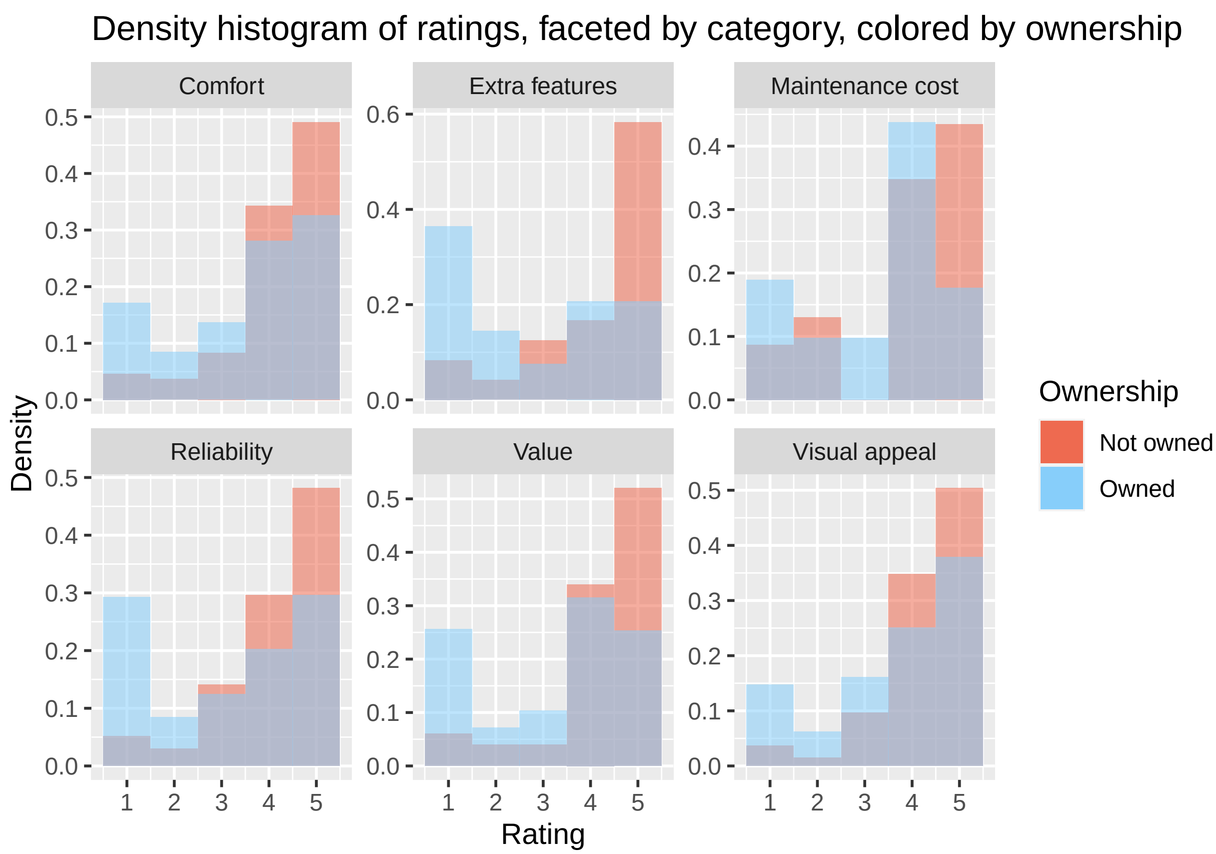
```
# Exploratory Analysis
# Explore the characteristics of the variables in the data

# One density plot
moped |>
  ggplot(aes(
    comfort, fill = "#440154FF"
  )) +
  geom_density() +
  labs(
    title = "Density plot of comfort ratings in moped reviews",
    x = "Comfort",
    y = "Density"
  ) +
  theme_minimal() +
  scale_fill_viridis_d() +
  theme(legend.position = "none")
```



```r
# density bars of all numerical variables, sorted by ownership
  # pivot longer
  moped %>%
    pivot_longer(visual_appeal:comfort) %>%
    select(value, name, owned) %>%
  # recoding ownership values
    mutate(owned = ifelse(owned == 0, "Not owned", "Owned")) %>%
  # recoding names of variables for facet titles
    mutate(name = recode(name,
                         "comfort" = "Comfort",
                         "extra_features" = "Extra features",
                         "maint_cost" = "Maintenance cost",
                         "reliability" = "Reliability",
                         "value" = "Value",
                         "visual_appeal" = "Visual appeal")) %>%
  # only NA values encoded to 0, we'll leave those out
    filter(value > 0) %>%
  # plot generation
  ggplot(aes(
    value, fill = as.factor(owned), alpha = 0.9
  )) +
    geom_histogram(breaks = seq(0.5, 5.5, 1), position = "identity", aes(y = ..density..)) +
    facet_wrap(vars(name), scales = "free_y") +
    labs(
      title = "Density histogram of ratings, faceted by category, colored by ownership",
      x = "Rating",
      y = "Density",
      fill = "Ownership"
    ) +
    guides(fill = "legend", alpha = "none") +
    scale_fill_manual(values = c(
      'Not owned' = '#EE6A50',
      'Owned' = '#87CEFA'
    ))
```

```
## Warning: The dot-dot notation (`..density..`) was deprecated in ggplot2 3.4.0.
## i Please use `after_stat(density)` instead.
```




```r
# proportion of ownership by use
moped %>%
  mutate(commuter = ifelse(used_for == "Commuting", 1, 0), .keep = "unused") |>
  group_by(commuter) %>%
  summarize(prop_owned = mean(owned), n = n()) %>%
  arrange(prop_owned)
```

```
## # A tibble: 2 x 3
##   commuter prop_owned     n
##      <dbl>      <dbl> <int>
## 1        0      0.662   160
## 2        1      0.854   553
```

```r
# proportion of ownership by model
moped %>%
  group_by(model) %>%
  summarize(prop_owned = mean(owned), n = n()) %>%
  arrange(prop_owned)
```

```
## # A tibble: 38 x 3
##    model                  prop_owned     n
##    <chr>                       <dbl> <int>
##  1 Tork Kratos                 0.242    33
##  2 Revolt RV 300               0.333     6
##  3 OLA S1                      0.444    18
```

```
##   4 Revolt RV 400           0.5      44
##   5 Odysse Evoqis           0.667     3
##   6 Bajaj Chetak            0.692    13
##   7 Okinawa i-Praise        0.727    11
##   8 TVS iQube               0.765    17
##   9 Bounce Infinity E1      0.8      10
##  10 Hero Electric Flash     0.809    94
## # ... with 28 more rows
```

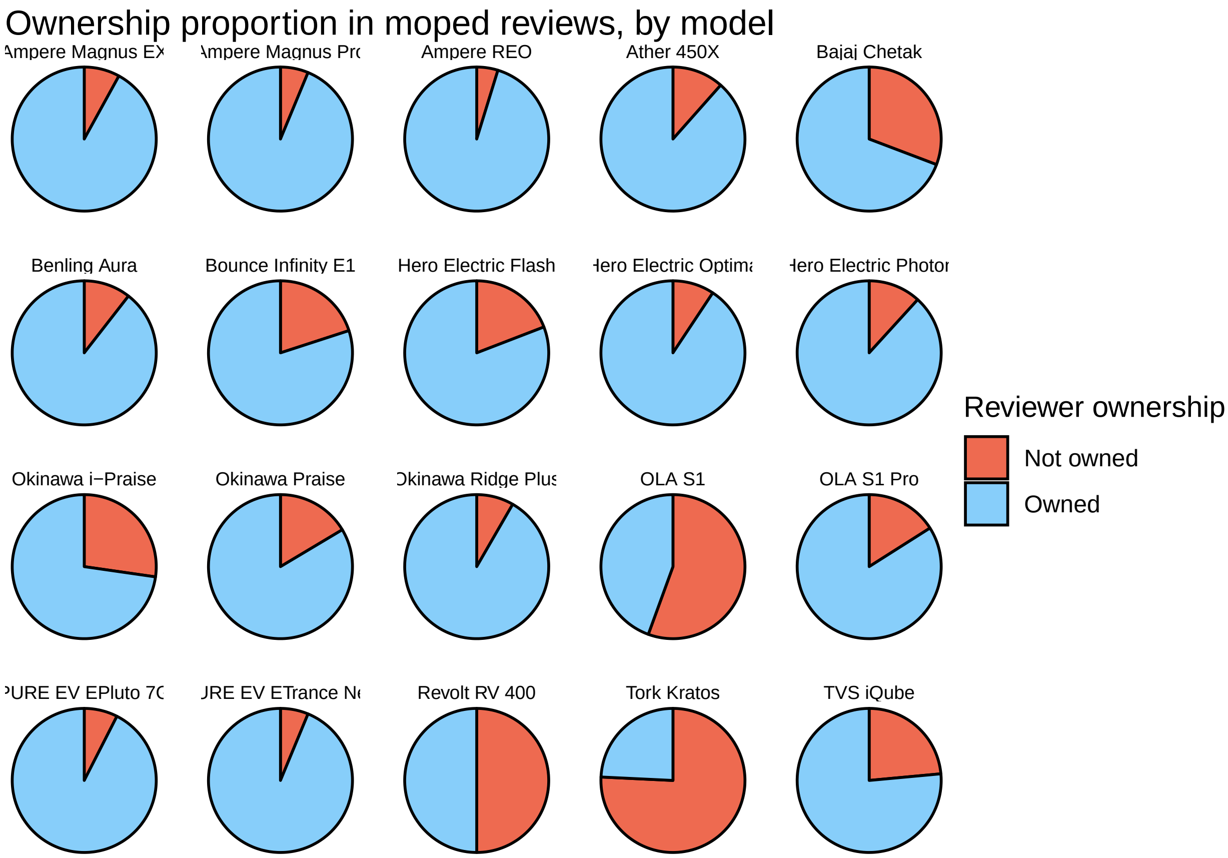
```
# pie plot to investigate ownership rate by model
  # generating a list of desired models
  model_list <-
    moped |>
      group_by(model) |>
      summarize(n = n()) |>
      arrange(desc(n)) |>
      head(n = 20) |>
      pull(var = model)

  # generating a table with clear aesthetic assignments for ggplot
  moped |>
    # culling low-n models using the list from earlier
    filter(model %in% model_list) |>
    # modifying owned to a factor
    mutate(owned = ifelse(owned == 0, "Not owned", "Owned")) |>
    group_by(model, owned) |>
    summarize(n = n()) |>
  # generating pie chart
  ggplot(
    aes(
        x0 = 0, y0 = 0,
        r0 = 0, r = 1,
        amount = n,
        fill = owned,
      )
    ) +
    geom_arc_bar(stat = "pie") +
    theme_void() +
    coord_fixed() +
    labs(title = "Ownership proportion in moped reviews, by model", fill = "Reviewer ownership") +
    facet_wrap(vars(model)) +
    scale_fill_manual(values = c(
      'Not owned' = '#EE6A50',
      'Owned' = '#87CEFA'
    )) +
    theme(
      panel.spacing = unit(0.5, "cm"),
      strip.text = element_text(size = 7)
    )
```

```
## `summarise()` has grouped output by 'model'. You can override using the
## `.groups` argument.

## Warning: Using the `size` aesthetic in this geom was deprecated in ggplot2 3.4.0.
```

```
## i Please use `linewidth` in the `default_aes` field and elsewhere instead.
```



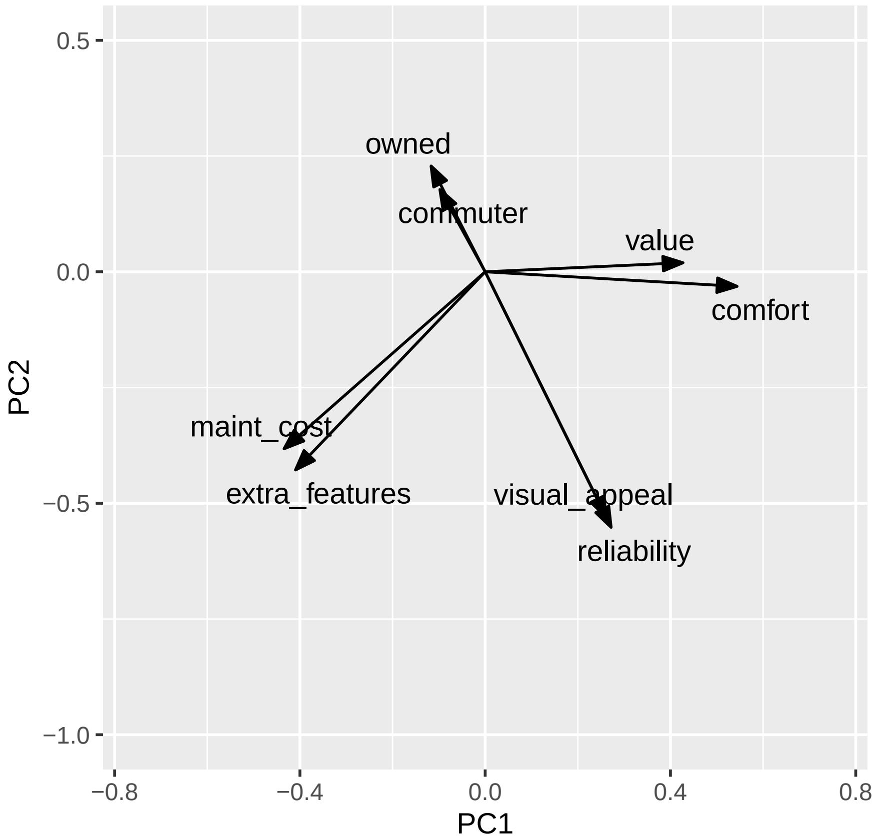

```
# PCA
  # arrow style object
  arrow_style <- arrow(
    angle = 20, length = grid::unit(8, "pt"),
    ends = "first", type = "closed")

  # generating PCA object
  pca_fit <-
    moped %>%
    # recoding the use variable to a dummy
    mutate(commuter = ifelse(used_for == "Commuting", 1, 0)) %>%
    # removing legacy/categorical variables
    select(-c(model, used_for)) %>%
    na.omit() %>%
    scale() %>%
    prcomp()
  pca_fit
```

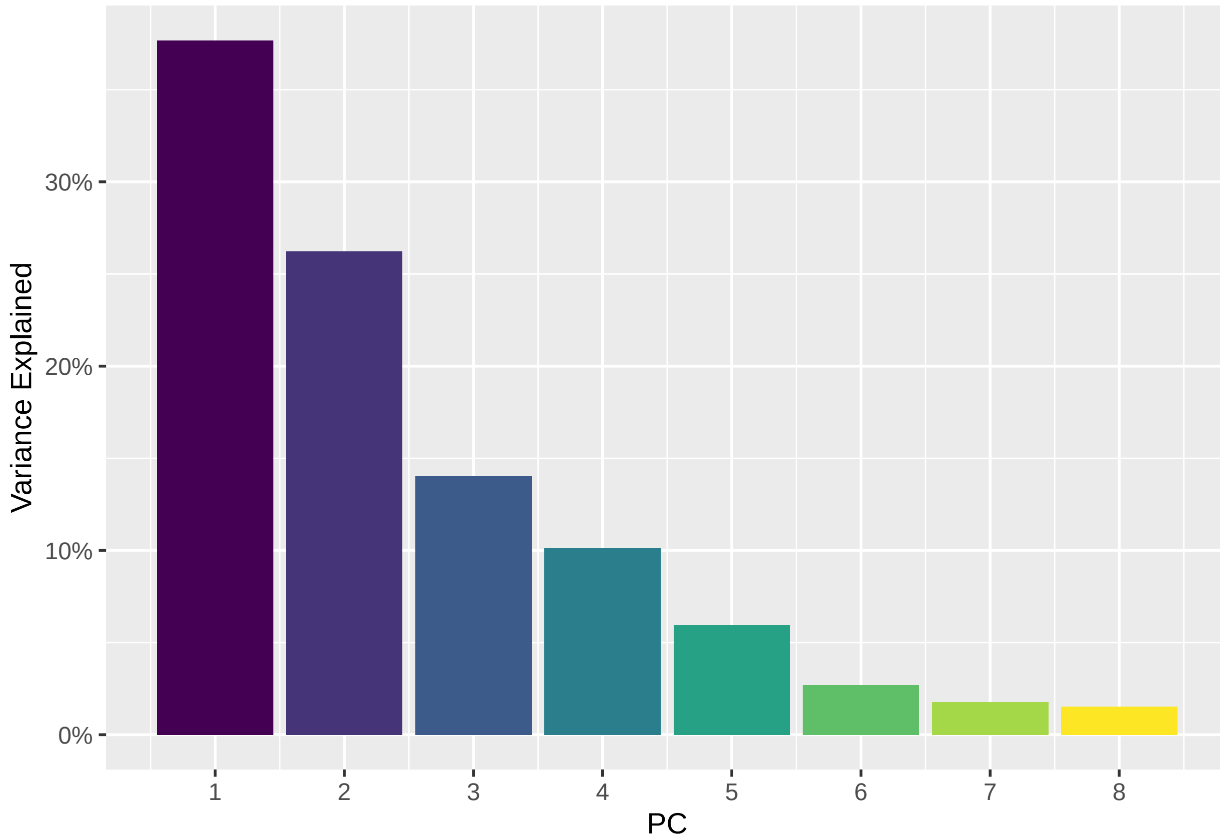
```
## Standard deviations (1, .., p=8):
## [1] 1.7362013 1.4487522 1.0591468 0.8997260 0.6901226 0.4644763 0.3756788
## [8] 0.3496874
##
## Rotation (n x k) = (8 x 8):
```

```
##                       PC1         PC2        PC3          PC4          PC5
## visual_appeal   0.26126418 -0.52972543 0.21039191 -0.029963039  0.318843786
## extra_features -0.40955655 -0.42812724 0.09100086  0.055031595 -0.203374153
## maint_cost     -0.43422465 -0.38231329 0.12060386  0.048861686 -0.208388521
## value           0.42718796  0.01950408 0.29078463  0.186596579 -0.821562611
## reliability     0.27189446 -0.55202501 0.12706360 -0.005249962  0.084013022
## comfort         0.54391735 -0.03149223 0.02603552 -0.040957698  0.197173270
## owned          -0.11680673  0.22858839 0.64157977  0.649731351  0.304310615
## commuter       -0.09771066  0.17770243 0.64800115 -0.731445192  0.006801328
##                       PC6         PC7         PC8
## visual_appeal  -0.69235323  0.02047774 -0.15663894
## extra_features -0.03898281 -0.52349622  0.56633194
## maint_cost      0.08594643  0.77279155  0.01825922
## value          -0.13922820 -0.01220364 -0.05707461
## reliability     0.67267153 -0.18407789 -0.33427418
## comfort         0.17276831  0.30517788  0.73413014
## owned           0.08345277 -0.02138558  0.01736218
## commuter        0.05527946 -0.02647845  0.01239898
```

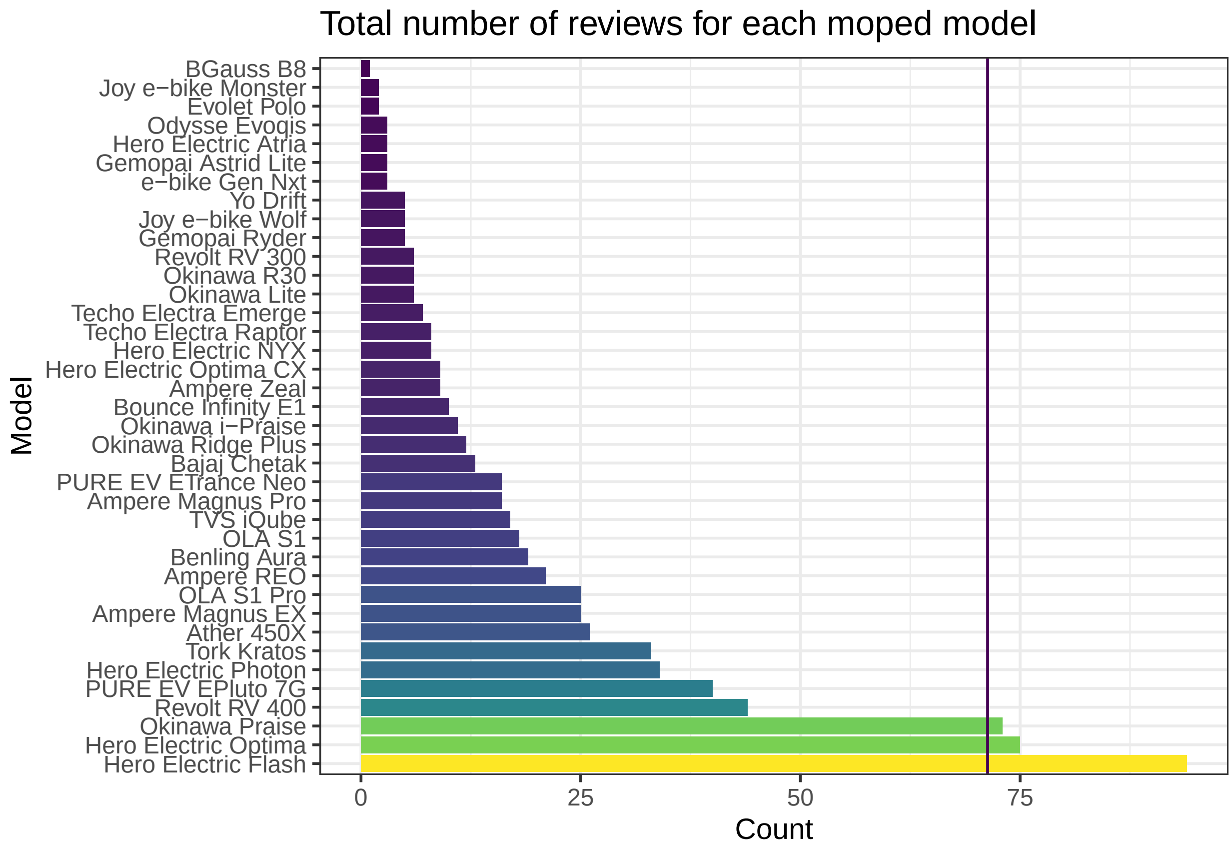
```
# rotation matrix
pca_fit |>
  tidy(matrix = "rotation") |>
  pivot_wider(
    names_from = "PC", values_from = "value",
    names_prefix = "PC"
  ) |>
  # biplot
  ggplot(aes(PC1, PC2)) +
  geom_segment(
    xend = 0, yend = 0,
    arrow = arrow_style
  ) +
  geom_text_repel(aes(label = column)) +
  xlim(-0.75, 0.75) + ylim(-1, 0.5) +
  coord_fixed()
```



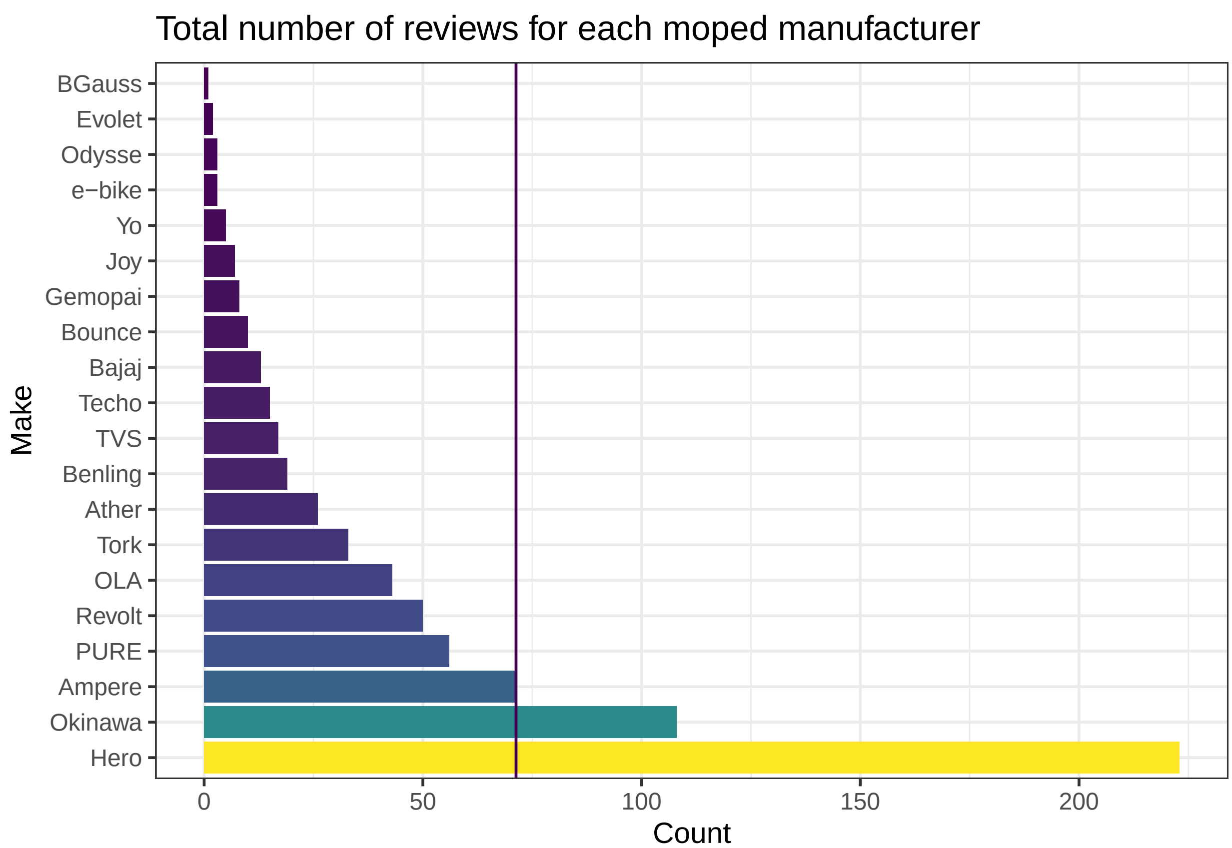

```
# fetching the r-squared values for the principle components via eigenvalue plot
pca_fit |>
  tidy(matrix = "eigenvalues") |>
  # scree plot
  ggplot(aes(PC, percent, fill = PC)) +
  geom_col() +
  scale_x_continuous(
    breaks = 1:8
  ) +
  scale_y_continuous(
    name = "Variance Explained",
    label = scales::label_percent(accuracy = 1)
  ) +
  scale_fill_viridis_c() +
  theme(legend.position = "none")
```




```r
# bar graph of counts by model name
  moped |>
    ggplot(aes(
      fct_infreq(model), fill = after_stat(count)
    )) +
    geom_bar() +
    coord_flip() +
    labs(
      title = "Total number of reviews for each moped model",
      x = "Model",
      y = "Count"
    ) +
    # including line showing the minimum count for inclusion as a group in splitting
    geom_hline(yintercept = 713 * .10, color = "#440154FF") +
    theme_bw() +
    scale_fill_viridis_c() +
    theme(legend.position = "none")
```




```
# counts by brand
  moped |>
    separate(model, into = c("make", "model"), sep = "\\s", extra = "merge") |>
    ggplot(aes(
      fct_infreq(make), fill = after_stat(count)
    )) +
    geom_bar() +
    coord_flip() +
    labs(
      title = "Total number of reviews for each moped manufacturer",
      x = "Make",
      y = "Count"
    ) +
    geom_hline(yintercept = 713 * .10, color = "#440154FF") +
    theme_bw() +
    scale_fill_viridis_c() +
    theme(legend.position = "none")
```

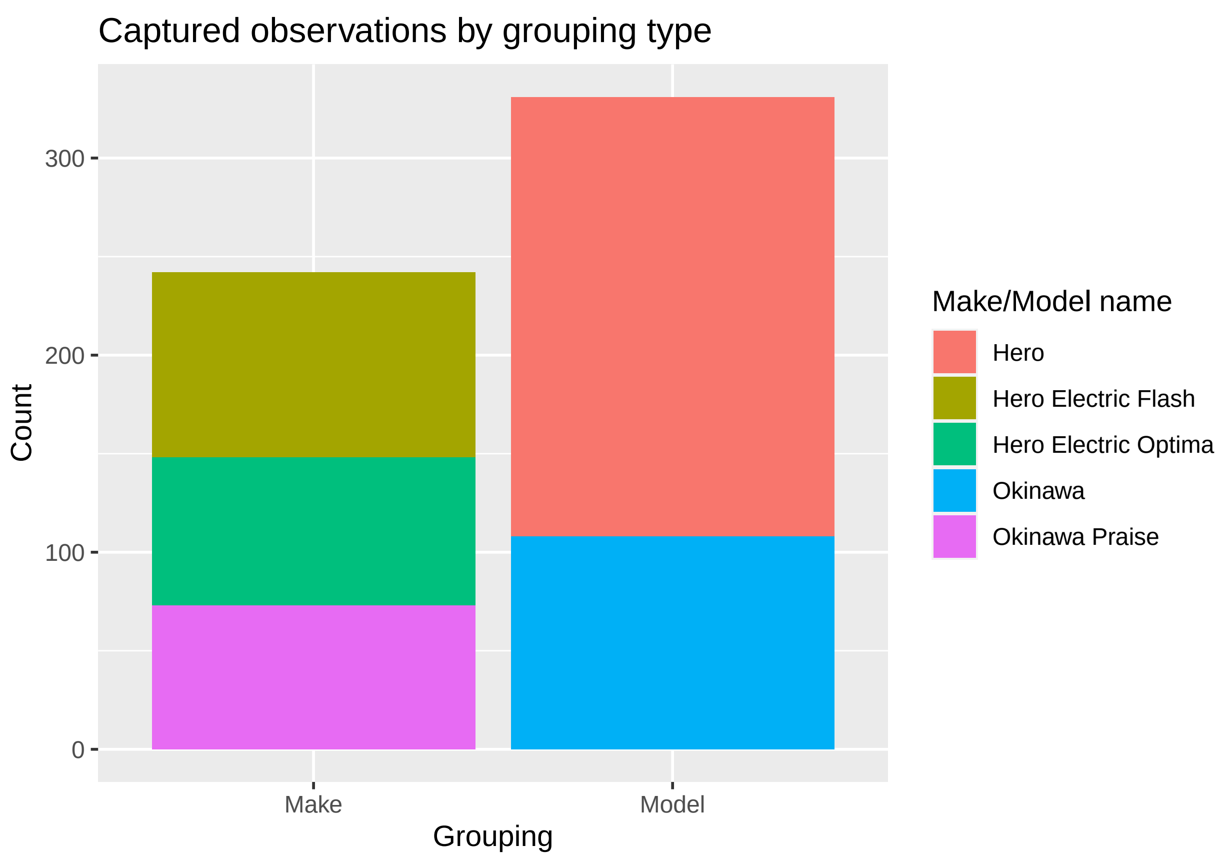



```
# bar graph of observations in makes vs models meeting the requirements
  # generating the desired summary stats
  model_n_1 <-
    moped |>
    group_by(model) |>
    mutate(n = n()) |>
    filter(n > 71.3) |>
    count() |>
    mutate(make = NA)

  model_n_2 <-
    moped |>
    separate(model, into = c("make", "model"), sep = "\\s", extra = "merge") |>
    group_by(make) |>
    mutate(n = n()) |>
    filter(n > 71.3) |>
    count() |>
    mutate(model = NA)

  # forming final matrix of information
  model_n <-
    rbind(model_n_1, model_n_2) |>
    mutate(type = ifelse(is.na(model) == TRUE, 1, 0),
           model = ifelse(type == 1, make, model)) |>
    select(-make)
  # generating the bar graph
```

```
model_n |>
  ggplot(aes(
    as.factor(type), n, fill = model
  )) +
  geom_col() +
  labs(
    x = "Grouping",
    y = "Count",
    title = "Captured observations by grouping type",
    fill = "Make/Model name"
  ) +
  scale_x_discrete(
    labels = c("Make", "Model")
  )
```




```
# examining balance of owned/not owned observations in the data
moped |>
  group_by(owned) |>
  summarize(n = n())
```

```
## # A tibble: 2 x 2
##   owned     n
##   <dbl> <int>
## 1     0   135
## 2     1   578
```

Exploratory Analysis

Describe what you found in the exploratory analysis. In your description you should: - Reference at least two different data visualizations you created above to demonstrate the characteristics of variables - Reference at least one data visualization you created above to demonstrate the relationship between two or more variables - Describe what your exploratory analysis has told you about the data - Describe any changes you have made to the data to enable modeling

characteristics of a single variable

I began the data exploration by modeling a single variable using `geom_density()`, `comfort`. The presence of large numbers of `NA` values as seen in the validation appeared here as a large number of zeroes in the density plot. No modification to the data was needed.

As you can see in the plot, the values reported by reviewers are not normally distributed.

characteristics of multiple variables

I then expanded this analysis, transitioning to a bar plot since the variables are all integers. I first used `pivot_longer()` and `select()` to generate a dataframe that contained only the values for the numerical variables accompanied by the variable name. I then transformed the `owned` variable into a character string for easier aesthetic assignments in the plot, recoded the names to be more appropriate for legend generation, removed the recoded `NA` values, and piped the resulting dataframe into `ggplot()`. Attached to the `ggplot()` call I specified the `identity` position and the `..density..` aesthetic in the `geom_histogram()` function to visualize density, faceted by the variable name with a free vertical axis, manually specified the color scheme to color by ownership, and specified labels as I do throughout. Since the values are all integers I specified breaks to bin the integer values neatly.

The resulting plot shows that non-owner reviewers were more likely to assign higher values to essentially every category, but that no category in particular appeared to be alone capable of identifying a non-owner review. Again none of the distributions of ratings were normal.

I then generated some summary tables using `group_by()` and `summarize()` to explore a potential relationship between ownership and any of the categorical variables. Commuter mopeds were far less likely to be non-owner reviews, and due to the smaller sample size of many model categories some models had 100% or 0% ownership rates. This warrants consideration when expanding these variables into dummy indicators, since any model incorporating those smaller sample models may not have much external validity when applied to new reviews of those small-sample models.

characteristics of multiple variables

To examine this further, I decided to generate a pie chart of the models with the highest number of observations to determine how many could be included in the final analysis. I started by generating `model_list`, a list of the models with the highest number of observations, by grouping on `model`, summarizing on `n()`, calling `head()` with `n = 20`, and pulling the `model` vector from the resulting dataframe. I then generated an appropriate dataframe for aesthetic assignments by filtering for models in `model_list`, converting `owned` back to a character vector, grouping on `model` and `owned`, and summarizing on `n()`. After piping the resulting dataframe into `ggplot()`, I used `geom_arg_bar()` faceted by `model` with colors manually assigned to `owned` using `scale_fill_manual()`. I also manually set panel spacing and text size using `theme()`.

The results showed that for high-n models of moped, no models featured only owners or non-owners in reviews. In the analysis portion I use far fewer models than 20, but I wouldn't risk overfitting my model prior to receiving new data by selecting more within reason.

characteristics of multiple variables

Since the above analysis hadn't given me any clear information about correlation with our outcome variable or correlation between our independent variables, I then decided to run a principle component analysis and generate a scree and biplot to quickly get a better idea for what's going on in the data. When running the

PCA I generated `commuter` as a binary of `used_for`, and dropped the remaining categorical variables that couldn't be easily one-hot encoded manually. I then scaled, omitted NA values, and ran `prcomp()`.

To generate the biplot I used `tidy()` to pull the rotation matrix, transformed it using `pivot_wider()`, and fed the results into `ggplot()` using `geom_segment()` and custom aesthetic assignments stored in `arrow_style`. Limits were set to contain the resulting vectors without excessive white space.

The resulting biplot showed clear correlation between `owned` and `commuter`, and a negative correlation between `owned`, `visual_appeal`, and `reliability`. The latter two also exhibited significant collinearity. The remaining variables were slightly negatively correlated with `owned` and collinear with one another. So moped owners commonly used the moped for commuting, reported that it was ugly as sin, and cited reliability as problematic. It would make sense that reviewers who only rent a moped wouldn't have time to be bothered by these issues - and it would make sense for fake reviews to give positive impressions of the models in question.

To generate the scree plot I applied `tidy()` to the PCA object and piped it directly into `ggplot()` with the `geom_col()` aesthetic. To reach a suitable proportion of variance explained I'd need to include three or more principle components, so PCA is unsuitable for modeling in this case even though the biplot provided useful information about the data.

Based on the results from the pie charts and summary tables, I knew that wrangling the `model` variable correctly was going to be an important factor in avoiding overfitting. Retaining all categories would lead to problematic overfitting, as the many small-n categories aren't representative samples from their respective population means. On the other hand, to maximize retained information for modeling I wanted to retain as much information as possible assigned to their respective observations. Theoretically, I was weighing two possibilities. First, morally onerous marketers from certain brands could be submitting fake reviews. This would lead to a high degree of correlation between the proportion of `owned == 1` values and brand name. Second, specific models could be more likely to be chosen as part of rental fleets - in which case the full model name would be the more appropriate unit for one-hot encoding. Retaining categories only at the brand level might also help retain information, since some brands may field many models such that their total market share comprises a substantial portion of the sample where individually they do not.

characteristics of a single variable

To determine this, I generated two visualizations. First, a `geom_bar()` by the default model name sorted using `fct_infreq()` using a `geom_hline()` to show categories meeting the default sample proportion threshold from `dummyVars()` of 0.10. I used the color palette from `viridis` and `coord_flip()` for readability and ease of interpretation. Then I repeated the visualization, but I first used `separate()` to split the `model` variable so I could examine the `make` independently.

The resulting plots showed that the same brands/models would be captured in either case, and that smaller brands didn't meet the threshold once their total market share was considered.

characteristics of multiple variables

I then decided to compare the total number of observations captured in each treatment plan. To do this, I generated two dataframes named `moped_n_1` and `moped_n_2` containing summary tables of observation counts grouped by model/make respectively and filtered for categories containing the desired proportion. I then appended them using `rbind()` to form `model_n`, and performed additional transformations to ensure neat aesthetic assignments for ggplot. I piped the resulting dataframe into `ggplot()`, factoring on the `type` dummy variable and using `geom_col()` so I could assign the preexisting count information from the summary tables to the y-axis.

The resulting plot showed that although the brands in both treatment plans were the same, one-hot encoding on `make` captured a substantially higher proportion of observations than encoding on `model`.

```
# problem type is binary classification

# seed for replication
```

```r
set.seed(100)
  ###https://win-vector.com/2017/04/15/encoding-categorical-variables-one-hot-and-beyond/

# manually converting one variable to a dummy
moped <-
  moped_og |>
  mutate(commuter = ifelse(used_for == "Commuting", 1, 0), .keep = "unused")

# test/train split
split <-
  initial_split(moped, prop = 0.8, strata = "model")

moped_train <-
  training(split)

moped_test <-
  testing(split)

# storing vtreat plan
treatplan <- designTreatmentsZ(moped_train, colnames(moped_train), minFraction = 1/10)
```

```
## [1] "vtreat 1.6.3 inspecting inputs Thu Nov 10 03:28:02 2022"
## [1] "designing treatments Thu Nov 10 03:28:02 2022"
## [1] " have initial level statistics Thu Nov 10 03:28:02 2022"
## [1] " scoring treatments Thu Nov 10 03:28:02 2022"
## [1] "have treatment plan Thu Nov 10 03:28:02 2022"
```

```r
# inspecting results
View(treatplan[["scoreFrame"]])

# executing treatment
train_treated <-
  prepare(treatplan, moped_train) |>
  select(-model_catP)

test_treated <-
  prepare(treatplan, moped_test)  |>
  select(-model_catP)

# log reg

  # model definition/training
  logreg_model_1 <-
    glm(owned ~ ., data = train_treated, family = "binomial")

  logreg_model_1
```

```
##
## Call:  glm(formula = owned ~ ., family = "binomial", data = train_treated)
##
## Coefficients:
##                      (Intercept)                     visual_appeal
```

```
##                          4.07279                            0.27647
##                   extra_features               extra_features_isBAD
##                         -0.51150                           -0.41112
##                       maint_cost                   maint_cost_isBAD
##                          0.31356                           -0.42311
##                            value                        value_isBAD
##                         -0.24639                           -1.63602
##                      reliability                            comfort
##                         -0.43664                            0.02433
##                    comfort_isBAD                           commuter
##                          0.90676                            0.80406
##  model_lev_x_Hero_Electric_Flash  model_lev_x_Hero_Electric_Optima
##                         -0.43671                            0.45999
##       model_lev_x_Okinawa_Praise
##                         -0.21299
##
## Degrees of Freedom: 568 Total (i.e. Null);  554 Residual
## Null Deviance:       553
## Residual Deviance: 462.5     AIC: 492.5
```

```
# summary of model
summary(logreg_model_1)
```

```
##
## Call:
## glm(formula = owned ~ ., family = "binomial", data = train_treated)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.3805   0.2196   0.3989   0.6512   1.8724
##
## Coefficients:
##                                  Estimate Std. Error z value Pr(>|z|)
## (Intercept)                       4.07279    1.58744   2.566  0.01030 *
## visual_appeal                     0.27647    0.18705   1.478  0.13939
## extra_features                   -0.51150    0.25365  -2.017  0.04374 *
## extra_features_isBAD             -0.41112    1.02727  -0.400  0.68900
## maint_cost                        0.31356    0.27640   1.134  0.25661
## maint_cost_isBAD                 -0.42311    0.88725  -0.477  0.63344
## value                            -0.24639    0.19081  -1.291  0.19661
## value_isBAD                      -1.63602    0.35816  -4.568 4.93e-06 ***
## reliability                      -0.43664    0.16786  -2.601  0.00929 **
## comfort                           0.02433    0.18104   0.134  0.89311
## comfort_isBAD                     0.90676    0.86185   1.052  0.29275
## commuter                          0.80406    0.25270   3.182  0.00146 **
## model_lev_x_Hero_Electric_Flash  -0.43671    0.40386  -1.081  0.27955
## model_lev_x_Hero_Electric_Optima  0.45999    0.56447   0.815  0.41512
## model_lev_x_Okinawa_Praise       -0.21299    0.49217  -0.433  0.66519
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 553.00  on 568  degrees of freedom
```

```
## Residual deviance: 462.54  on 554  degrees of freedom
## AIC: 492.54
##
## Number of Fisher Scoring iterations: 5
```

```
  # test model
  test_treated$pred <-
    predict(logreg_model_1, test_treated, type = "response")

  # saving this dataframe for later
  logreg_pred_1 <-
    test_treated

# xgboost
  # defining dataframes sans outcome
    xgb_train <-
      train_treated |>
      select(-owned) |>
      as.matrix()

    xgb_test <-
      test_treated |>
      select(-c(pred, owned)) |>
      as.matrix()

  # running cross validation to find the ideal parameters
    cv <- xgb.cv(data = xgb_train,
                 label = train_treated$owned,
                 nrounds = 100,
                 nfold = 5,
                 objective = "binary:logistic",
                 max_depth = 5,
                 early_stopping_rounds = 5,
                 verbose = FALSE   # silent
      )
  # fetching evaluation log
    cv$evaluation_log |>
      summarize(ntrees.train = which.min(train_logloss_mean),
                ntrees.test = which.min(test_logloss_mean))
```

```
##   ntrees.train ntrees.test
## 1           14           9
```

```
  # checking cross validation results using xgb.train()
  # generating appropriate matrices
  xgbDM_train <-
    xgb.DMatrix(data = xgb_train, label = train_treated$owned)

  xgbDM_test <-
    xgb.DMatrix(data = xgb_test, label = test_treated$owned)

  # generating watchlist
  watchlist <-
```

```r
    list(train = xgbDM_train, test = xgbDM_test)

  # running xgb.train()
  xgb_training <-
    xgb.train(
      data = xgbDM_train,
      max.depth = 5,
      objective = "binary:logistic",
      watchlist = watchlist,
      nrounds = 100,
      verbose = 0
    )

  # obtaining evaluation log
  xgb_training$evaluation_log |>
    summarize(ntrees.train = which.min(train_logloss),
              ntrees.test = which.min(test_logloss))
```

```
##   ntrees.train ntrees.test
## 1          100          14
```

```r
  # defining final model
  xgb_model_1 <- xgboost(data = xgb_train,
                         label = train_treated$owned,
                         objective = "binary:logistic",
                         max.depth = 5,
                         nrounds = 13,
                         verbose = FALSE
                         )

  # predictions
  test_treated$pred <-
    predict(xgb_model_1, xgb_test, nrounds = 8)

  # saving for evaluation
  xgb_pred_1 <-
    test_treated

  # saving colnames for evaluation
  colnames_1 <-
    colnames(xgb_train)
```

```r
# models without stratification
  # test/train split
  split <-
    initial_split(moped, prop = 0.8)

  moped_train <-
    training(split)

  moped_test <-
    testing(split)
```

```r
# storing new vtreat plan
treatplan <- designTreatmentsZ(moped_train, colnames(moped_train), minFraction = 1/10)
```

```
## [1] "vtreat 1.6.3 inspecting inputs Thu Nov 10 03:28:02 2022"
## [1] "designing treatments Thu Nov 10 03:28:02 2022"
## [1] " have initial level statistics Thu Nov 10 03:28:02 2022"
## [1] " scoring treatments Thu Nov 10 03:28:02 2022"
## [1] "have treatment plan Thu Nov 10 03:28:02 2022"
```

```r
# inspecting results
View(treatplan[["scoreFrame"]])

# executing treatment
train_treated <-
  prepare(treatplan, moped_train) |>
  select(-model_catP)

test_treated <-
  prepare(treatplan, moped_test)  |>
  select(-model_catP)

# log reg
# model definition/training
logreg_model_2 <-
  glm(owned ~ ., data = train_treated, family = "binomial")

logreg_model_2
```

```
##
## Call:  glm(formula = owned ~ ., family = "binomial", data = train_treated)
##
## Coefficients:
##                       (Intercept)                      visual_appeal
##                            4.8544                             0.0320
##                    extra_features               extra_features_isBAD
##                           -0.5791                             0.1897
##                        maint_cost                   maint_cost_isBAD
##                            0.3333                            -1.0609
##                             value                        value_isBAD
##                           -0.4045                            -1.5976
##                       reliability                            comfort
##                           -0.2900                             0.1029
##                     comfort_isBAD                           commuter
##                            0.8675                             0.6907
##  model_lev_x_Hero_Electric_Flash  model_lev_x_Hero_Electric_Optima
##                           -0.0391                             0.5113
##
## Degrees of Freedom: 569 Total (i.e. Null);  556 Residual
## Null Deviance:       567.7
## Residual Deviance: 476.7     AIC: 504.7
```

```r
# summary of model
summary(logreg_model_2)
```

```
##
## Call:
## glm(formula = owned ~ ., family = "binomial", data = train_treated)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.4311   0.2115   0.4084   0.7021   1.4881
##
## Coefficients:
##                                  Estimate Std. Error z value Pr(>|z|)
## (Intercept)                        4.8544     1.6285   2.981  0.00287 **
## visual_appeal                      0.0320     0.1837   0.174  0.86167
## extra_features                    -0.5791     0.2631  -2.201  0.02772 *
## extra_features_isBAD               0.1898     1.0654   0.178  0.85865
## maint_cost                         0.3333     0.2870   1.162  0.24540
## maint_cost_isBAD                  -1.0609     0.9266  -1.145  0.25228
## value                             -0.4045     0.1988  -2.035  0.04186 *
## value_isBAD                       -1.5977     0.3286  -4.861 1.17e-06 ***
## reliability                       -0.2900     0.1628  -1.782  0.07482 .
## comfort                            0.1029     0.1828   0.563  0.57368
## comfort_isBAD                      0.8675     0.8537   1.016  0.30957
## commuter                           0.6907     0.2519   2.742  0.00610 **
## model_lev_x_Hero_Electric_Flash   -0.0391     0.3846  -0.102  0.91903
## model_lev_x_Hero_Electric_Optima   0.5113     0.5029   1.017  0.30931
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 567.68  on 569  degrees of freedom
## Residual deviance: 476.71  on 556  degrees of freedom
## AIC: 504.71
##
## Number of Fisher Scoring iterations: 5
```

```r
# test model
test_treated$pred <-
  predict(logreg_model_2, test_treated, type = "response")

# saving this dataframe for later
logreg_pred_2 <-
  test_treated

# xgboost
# defining dataframes sans outcome
xgb_train <-
  train_treated |>
  select(-owned) |>
  as.matrix()
```

```r
xgb_test <-
  test_treated |>
  select(-c(pred, owned)) |>
  as.matrix()

# generating appropriate matrices
xgbDM_train <-
  xgb.DMatrix(data = xgb_train, label = train_treated$owned)

xgbDM_test <-
  xgb.DMatrix(data = xgb_test, label = test_treated$owned)

# running cross validation to find the ideal parameters
cv <- xgb.cv(data = xgbDM_train,
             nrounds = 100,
             nfold = 5,
             objective = "binary:logistic",
             max_depth = 5,
             early_stopping_rounds = 5,
             verbose = FALSE   # silent
)
# fetching evaluation log
cv$evaluation_log |>
  summarize(ntrees.train = which.min(train_logloss_mean),
            ntrees.test = which.min(test_logloss_mean))
```

```
##   ntrees.train ntrees.test
## 1           12           7
```

```r
# checking cross validation results using xgb.train()
# generating watchlist
watchlist <-
  list(train = xgbDM_train, test = xgbDM_test)

# running xgb.train()
xgb_training <-
  xgb.train(
    data = xgbDM_train,
    max.depth = 5,
    objective = "binary:logistic",
    watchlist = watchlist,
    nrounds = 100,
    verbose = 0
  )

# obtaining evaluation log
xgb_training$evaluation_log |>
  summarize(ntrees.train = which.min(train_logloss),
            ntrees.test = which.min(test_logloss))
```

```
##   ntrees.train ntrees.test
## 1          100          11
```

```r
  # defining final model
  xgb_model_2 <- xgboost(data = xgbDM_train,
                              objective = "binary:logistic",
                              max.depth = 5,
                              nrounds = 14,
                              verbose = FALSE
  )

  # predictions
  test_treated$pred <-
    predict(xgb_model_2, xgb_test, nrounds = 9)

  # saving for evaluation
  xgb_pred_2 <-
    test_treated

  # saving colnames for evaluation
  colnames_2 <-
    colnames(xgb_train)
```

```r
# models without `model`
  # test/train split
  split <-
    moped |>
    select(-model) |>
    initial_split(prop = 0.8)

  moped_train <-
    training(split)

  moped_test <-
    testing(split)

  # storing new vtreat plan
  treatplan <- designTreatmentsZ(moped_train, colnames(moped_train), minFraction = 1/10)
```

```
## [1] "vtreat 1.6.3 inspecting inputs Thu Nov 10 03:28:03 2022"
## [1] "designing treatments Thu Nov 10 03:28:03 2022"
## [1] " have initial level statistics Thu Nov 10 03:28:03 2022"
## [1] " scoring treatments Thu Nov 10 03:28:03 2022"
## [1] "have treatment plan Thu Nov 10 03:28:03 2022"
```

```r
  # inspecting results
  View(treatplan[["scoreFrame"]])

  # executing treatment
  train_treated <-
    prepare(treatplan, moped_train)

  test_treated <-
    prepare(treatplan, moped_test)

# log reg
```

```r
# model definition/training
logreg_model_3 <-
  glm(owned ~ ., data = train_treated, family = "binomial")

logreg_model_3
```

```
##
## Call:  glm(formula = owned ~ ., family = "binomial", data = train_treated)
##
## Coefficients:
##         (Intercept)         visual_appeal        extra_features
##             4.38996               0.35579              -0.55521
## extra_features_isBAD            maint_cost      maint_cost_isBAD
##             0.24099               0.32133              -1.18879
##               value           value_isBAD           reliability
##            -0.16691              -1.40566              -0.55032
##             comfort         comfort_isBAD              commuter
##            -0.09271               0.66584               0.81104
##
## Degrees of Freedom: 569 Total (i.e. Null);  558 Residual
## Null Deviance:       550.5
## Residual Deviance: 456.7     AIC: 480.7
```

```r
# summary of model
summary(logreg_model_3)
```

```
##
## Call:
## glm(formula = owned ~ ., family = "binomial", data = train_treated)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.4501   0.2112   0.4090   0.6536   1.9542
##
## Coefficients:
##                      Estimate Std. Error z value Pr(>|z|)
## (Intercept)           4.38996    1.55325   2.826  0.00471 **
## visual_appeal         0.35579    0.19816   1.795  0.07258 .
## extra_features       -0.55521    0.27037  -2.053  0.04002 *
## extra_features_isBAD  0.24099    1.05232   0.229  0.81886
## maint_cost            0.32133    0.30034   1.070  0.28467
## maint_cost_isBAD     -1.18879    0.92949  -1.279  0.20091
## value                -0.16691    0.17732  -0.941  0.34656
## value_isBAD          -1.40566    0.29223  -4.810 1.51e-06 ***
## reliability          -0.55032    0.17678  -3.113  0.00185 **
## comfort              -0.09271    0.18489  -0.501  0.61605
## comfort_isBAD         0.66584    0.83211   0.800  0.42360
## commuter              0.81104    0.25210   3.217  0.00129 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
```

```
##
##     Null deviance: 550.50  on 569  degrees of freedom
## Residual deviance: 456.71  on 558  degrees of freedom
## AIC: 480.71
##
## Number of Fisher Scoring iterations: 5
```

```
  # test model
  test_treated$pred <-
    predict(logreg_model_3, test_treated, type = "response")

  # saving this dataframe for evaluation
  logreg_pred_3 <-
    test_treated

# xgboost
  # defining dataframes sans outcome
  xgb_train <-
    train_treated |>
    select(-owned) |>
    as.matrix()

  xgb_test <-
    test_treated |>
    select(-c(pred, owned)) |>
    as.matrix()

  # generating appropriate matrices
  xgbDM_train <-
    xgb.DMatrix(data = xgb_train, label = train_treated$owned)

  xgbDM_test <-
    xgb.DMatrix(data = xgb_test, label = test_treated$owned)


  # running cross validation to find the ideal parameters
  cv <- xgb.cv(data = xgbDM_train,
               nrounds = 100,
               nfold = 5,
               objective = "binary:logistic",
               max_depth = 5,
               early_stopping_rounds = 5,
               verbose = FALSE   # silent
  )
  # fetching evaluation log
  cv$evaluation_log |>
    summarize(ntrees.train = which.min(train_logloss_mean),
              ntrees.test = which.min(test_logloss_mean))
```

```
##   ntrees.train ntrees.test
## 1           18          13
```

```r
# checking cross validation results using xgb.train()
# generating watchlist
watchlist <-
  list(train = xgbDM_train, test = xgbDM_test)

# running xgb.train()
xgb_training <-
  xgb.train(
    data = xgbDM_train,
    max.depth = 5,
    objective = "binary:logistic",
    watchlist = watchlist,
    nrounds = 100,
    verbose = 0
  )

# obtaining evaluation log
xgb_training$evaluation_log |>
  summarize(ntrees.train = which.min(train_logloss),
            ntrees.test = which.min(test_logloss))
```

```
##   ntrees.train ntrees.test
## 1          100          11
```

```r
# defining final model
xgb_model_3 <- xgboost(data = xgbDM_train,
                       objective = "binary:logistic",
                       max.depth = 5,
                       nrounds = 15,
                       verbose = FALSE
)

# predictions
test_treated$pred <-
  predict(xgb_model_3, xgb_test, nrounds = 10)

# saving for evaluation
xgb_pred_3 <-
  test_treated

# saving variable names for evaluation
colnames_3 <-
  colnames(xgb_train)
```

Model Fitting

Describe your approach to the model fitting. In your description you should: - Describe what type of machine learning problem you are working on - Describe which method you selected for the baseline model and explain why you chose this model - Describe which method you selected for the comparison model and explain why you chose this model

Since `owned` is a binary classifier, it was clear that I was dealing with a binary classification problem.

For the base model I selected logistic regression. Although I didn't feel it was likely to show exceptional performance based on the results from the data exploration and the simplicity of the methodology, it would be easy to examine and evaluate and its simplicity makes it useful as a baseline. Another advantage of this simplicity is that although logistic regression is unlikely to generate highly accurate predictions, the risk of overfitting or other serious problems is low.

For the comparison model I selected Xtreme Gradient Boosting. I chose `xgboost()` over random forest modeling for a number of reasons. Gradient boosting is better with unbalanced data, better at handling large numbers of categorical variables, and generally has a slight edge in performance. Random forest models are easier to tune, but my priority was model performance rather than speed.

For the test/train split I used the standard 80/20 proportion, `initial_split()`, and the `vtreat` toolkit. The function `designTreatmentsZ()` has additional features for handling of `NA` values not available in `dummyVars()` from `caret`, so given the large number of `NA` values in the data I decided to use `vtreat`. I specified the `initial_split()` to stratify on `model` in order to avoid problems with non-representative sampling of models, and I removed `model_catP` after treatment to avoid metadata about `model` category proportion leading to overfitting.

After splitting and preparing the data, I generated `logreg_model` by running `glm()` on the training data. I then used `predict()` to generate predicted values for the test dataframe and saved the results for model evaluation.

For the gradient boosting model I first used `select()` and `as.matrix()` to generate matrices of data appropriate for the `xgboost` toolkit, without their outcome variables.

I then ran a k-fold cross validation using `xgb.cv()` in conjunction with `xgb.train()` to find the highest performing parameters for the model. For `xgb.train()` I formatted the existing dataframes into `xgb.DMatrix` objects using the corresponding function and defined a `watchlist` of the test and train datasets. I then fed the resulting parameters into `xgboost()` using `objective = "binary:logistic"`, with a `max.depth` of 5.

I experimented with parameter settings to ensure optimal performance, but was unable to improve AUC scores. The results are in the model evalution section.

I then decided to experiment with a number of variations to better estimate performance in modeling future data.

First, I reran the logistic regression and gradient boosting algorithms with unstratified samples. Depending on the use case for the model, this may be more descriptive of performance in future applications if the algorithm is being used to e.g. predict ownership on individual incoming reviews. The proportions of models prevalent in future reviews may not reflect the proportion of past data, and so the model may need to perform on data that does not fully resemble the training set.

Second, I built models to perform estimations without using `model` to compare to the unstratified performance. Although the feature importance indicates that the stratified model places little relative importance on the results of categorical variables, the model is likely to precondition some trees on model membership and may improve in performance on unstratified samples without that information.

The code for these models is contained in the second and third code chunks above, and their performance metrics are contained in the model evaluation section below.

```
# Model Evaluation
# Choose a metric and evaluate the performance of the two models

# model set 1 evaluation
# logreg evaluation
  # glance to get model stats
  (perf <- glance(logreg_model_1))
```

```
## # A tibble: 1 x 8
```

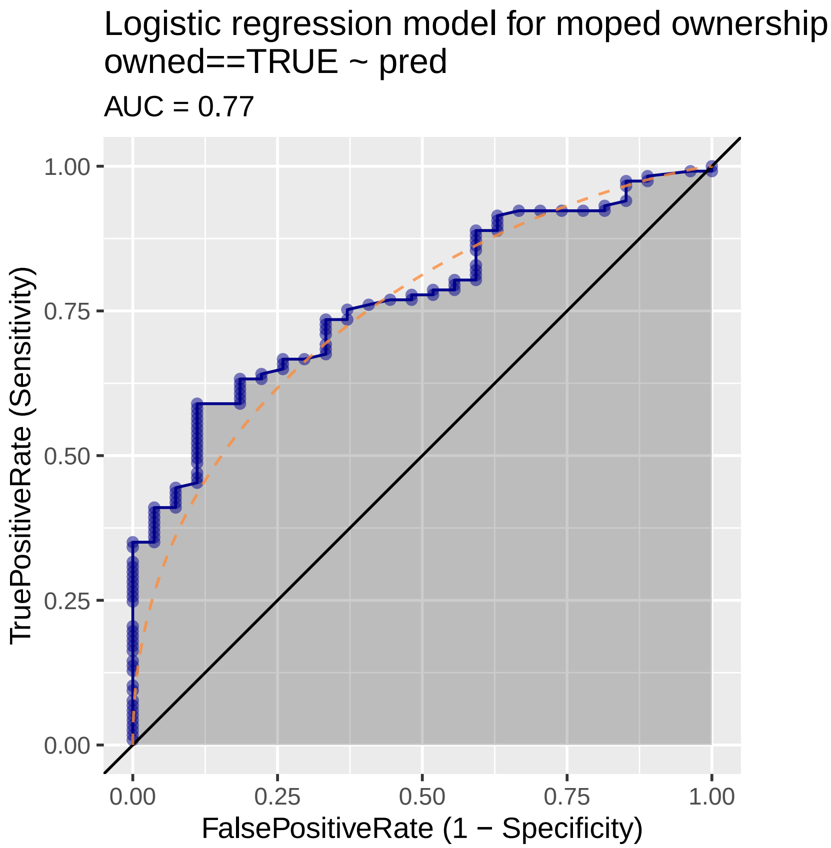
```
##   null.deviance df.null logLik   AIC   BIC deviance df.residual  nobs
##           <dbl>   <int>  <dbl> <dbl> <dbl>    <dbl>       <int> <int>
## 1          553.     568  -231.  493.  558.     463.         554   569
```

```r
# calculating pseudo-R-squared
(pseudoR2 <- 1 - perf$deviance/perf$null.deviance)
```

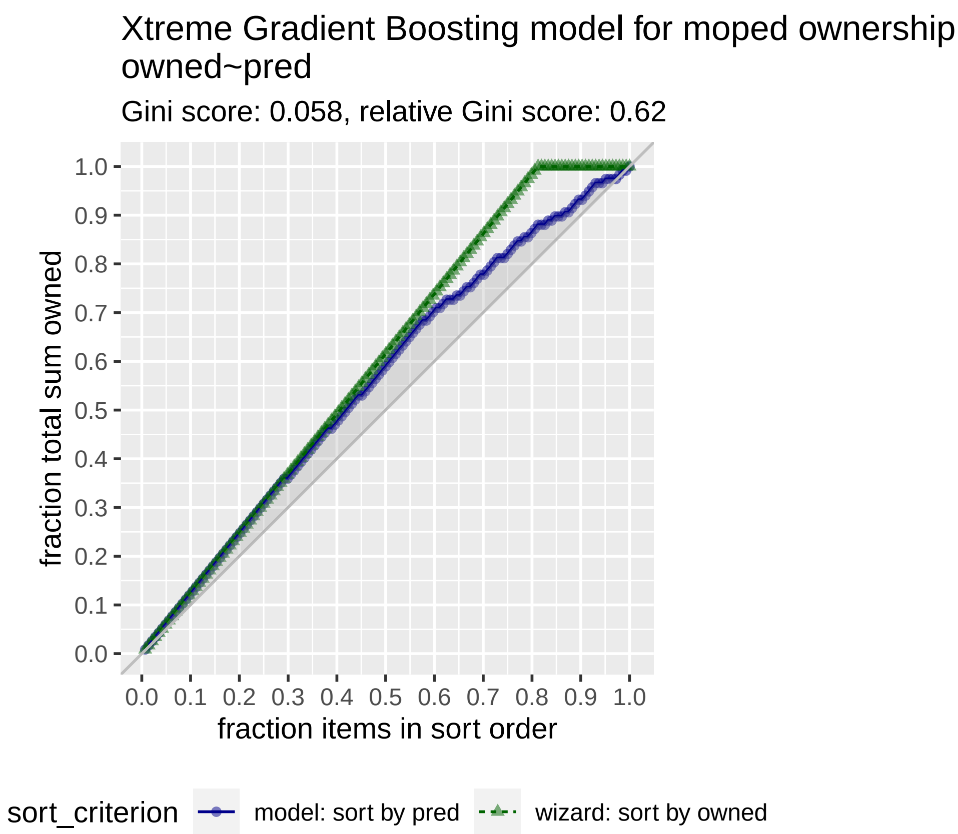
```
## [1] 0.1635854
```

```r
# gain curve plot
GainCurvePlot(logreg_pred_1, xvar = "pred", "owned", "Logistic regression model for moped ownership")
```

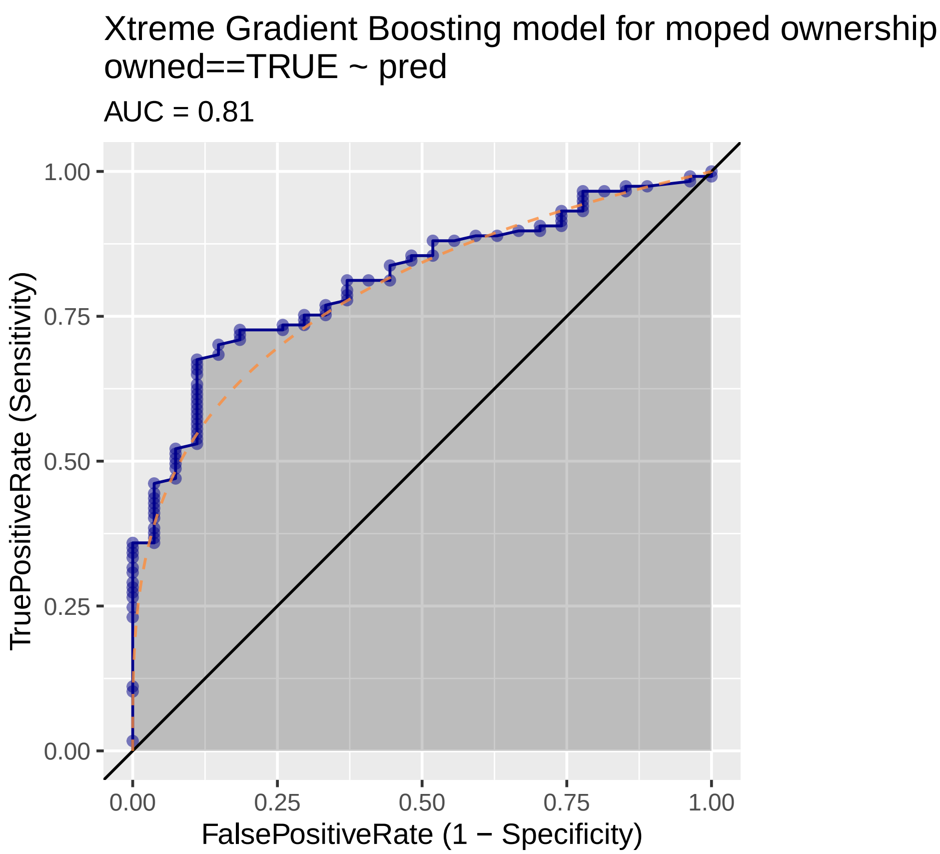

```r
# ROC curve
ROCPlot(logreg_pred_1,
        xvar = "pred",
        truthVar = "owned",
        truthTarget = TRUE,
        title = "Logistic regression model for moped ownership",
        add_beta_ideal_curve = TRUE)
```




```
# xgb evaluation
  # gain curve plot
  GainCurvePlot(xgb_pred_1, xvar = "pred", "owned", "Xtreme Gradient Boosting model for moped ownership
```



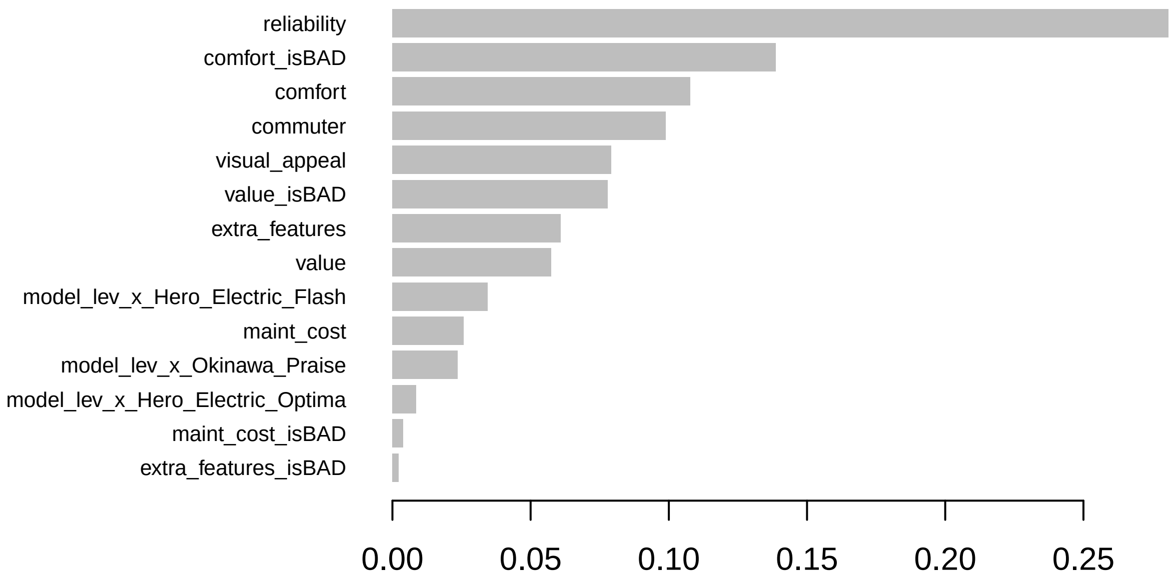

```
# ROC curve #2
ROCPlot(xgb_pred_1,
        xvar = "pred",
        truthVar = "owned",
        truthTarget = TRUE,
        title = "Xtreme Gradient Boosting model for moped ownership",
        add_beta_ideal_curve = TRUE)
```




```r
# inspecting feature importance
(importance_matrix <-
    xgb.importance(feature_names = colnames_1,
                   model = xgb_model_1))
```

```
##                               Feature        Gain       Cover   Frequency
##  1:                       reliability 0.280722387 0.261067689 0.154255319
##  2:                    comfort_isBAD 0.138711492 0.031999076 0.026595745
##  3:                           comfort 0.107803167 0.120758074 0.122340426
##  4:                          commuter 0.098919755 0.107213569 0.122340426
##  5:                     visual_appeal 0.079136047 0.072568630 0.117021277
##  6:                      value_isBAD 0.077958469 0.108844290 0.047872340
##  7:                    extra_features 0.060958420 0.091367803 0.117021277
##  8:                             value 0.057425564 0.113256846 0.090425532
##  9:  model_lev_x_Hero_Electric_Flash 0.034492626 0.032480093 0.042553191
## 10:                        maint_cost 0.025698850 0.017730371 0.095744681
## 11:        model_lev_x_Okinawa_Praise 0.023556153 0.023615368 0.037234043
## 12: model_lev_x_Hero_Electric_Optima 0.008517928 0.008430003 0.015957447
## 13:                 maint_cost_isBAD 0.003835525 0.006503444 0.005319149
## 14:             extra_features_isBAD 0.002263616 0.004164744 0.005319149
```

```r
# visualizing feature importance
xgb.plot.importance(importance_matrix[1:14,])
```



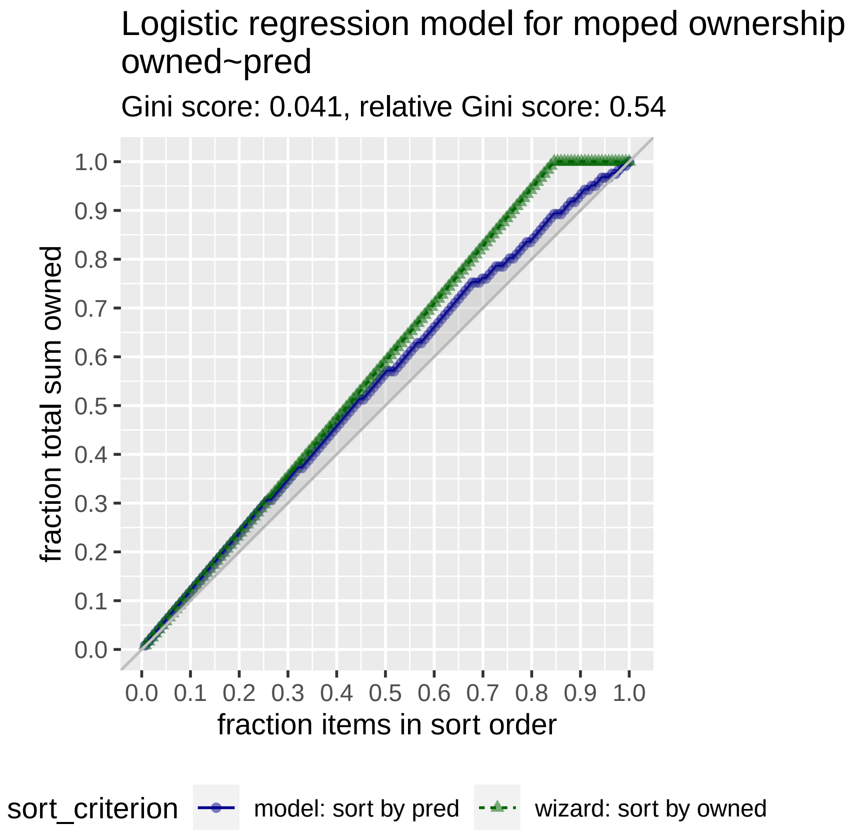

```r
# model set 2 evaluation
  # logreg evaluation
  # glance to get model stats
  (perf <- glance(logreg_model_2))
```

```
## # A tibble: 1 x 8
##   null.deviance df.null logLik   AIC   BIC deviance df.residual  nobs
##           <dbl>   <int>  <dbl> <dbl> <dbl>    <dbl>       <int> <int>
## 1          568.     569  -238.  505.  566.     477.         556   570
```

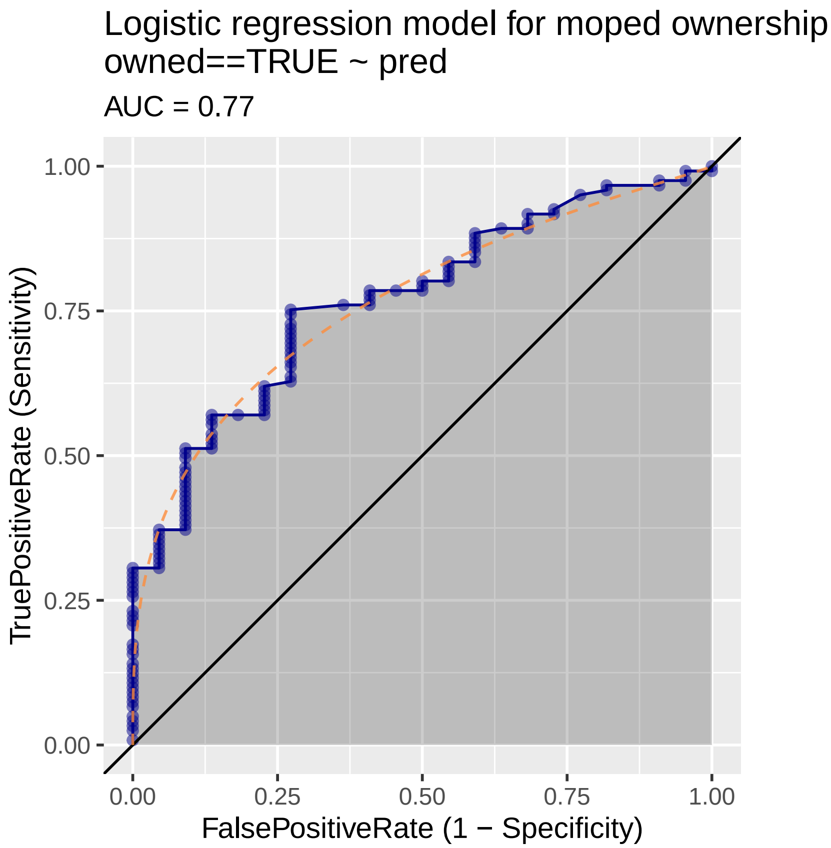
```r
  # calculating pseudo-R-squared
  (pseudoR2 <- 1 - perf$deviance/perf$null.deviance)
```

```
## [1] 0.160235
```

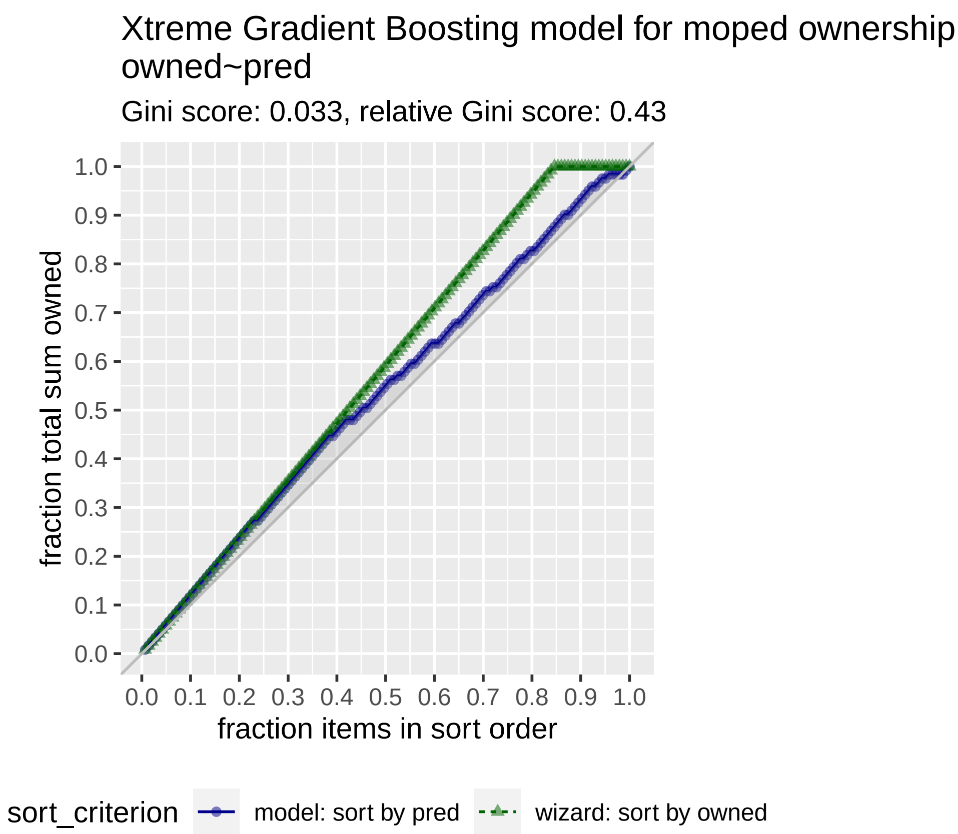
```r
  # gain curve plot
  GainCurvePlot(logreg_pred_2, xvar = "pred", "owned", "Logistic regression model for moped ownership")
```



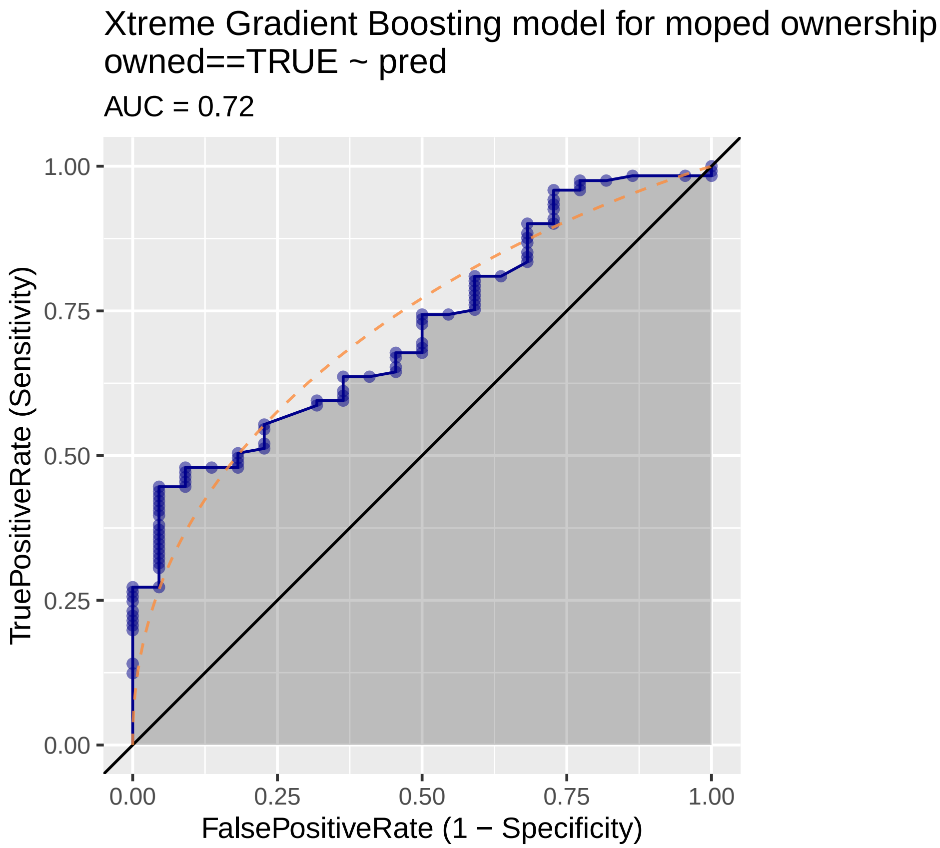

```
# ROC curve
ROCPlot(logreg_pred_2,
        xvar = "pred",
        truthVar = "owned",
        truthTarget = TRUE,
        title = "Logistic regression model for moped ownership",
        add_beta_ideal_curve = TRUE)
```




```
# xgb evaluation
# gain curve plot
GainCurvePlot(xgb_pred_2, xvar = "pred", "owned", "Xtreme Gradient Boosting model for moped ownership
```



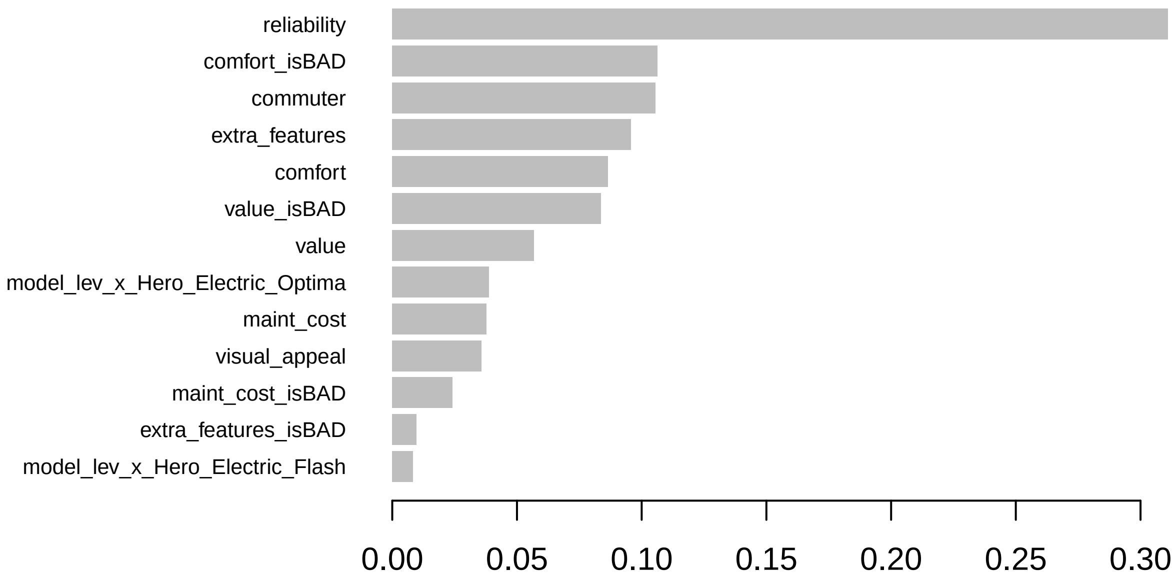

```r
# ROC curve #2
ROCPlot(xgb_pred_2,
        xvar = "pred",
        truthVar = "owned",
        truthTarget = TRUE,
        title = "Xtreme Gradient Boosting model for moped ownership",
        add_beta_ideal_curve = TRUE)
```




```r
# inspecting feature importance
(importance_matrix <-
    xgb.importance(feature_names = colnames_2,
                   model = xgb_model_2))
```

```
##                              Feature        Gain       Cover   Frequency
##  1:                      reliability 0.310958676 0.185086283 0.126315789
##  2:                    comfort_isBAD 0.106266669 0.015067493 0.010526316
##  3:                         commuter 0.105467984 0.106198016 0.163157895
##  4:                   extra_features 0.095637160 0.154827726 0.115789474
##  5:                          comfort 0.086518924 0.087757277 0.147368421
##  6:                      value_isBAD 0.083654625 0.111880806 0.063157895
##  7:                            value 0.056924471 0.111506771 0.100000000
##  8: model_lev_x_Hero_Electric_Optima 0.038828795 0.074525764 0.068421053
##  9:                       maint_cost 0.037869854 0.023542361 0.100000000
## 10:                    visual_appeal 0.035711657 0.083787902 0.068421053
## 11:                 maint_cost_isBAD 0.024142360 0.030051043 0.021052632
## 12:             extra_features_isBAD 0.009693987 0.006648917 0.005263158
## 13:  model_lev_x_Hero_Electric_Flash 0.008324840 0.009119640 0.010526316
```

```r
# visualizing feature importance
xgb.plot.importance(importance_matrix[1:13,])
```




```
# model set 3 evaluation
  # logreg evaluation
  # glance to get model stats
  (perf <- glance(logreg_model_3))
```

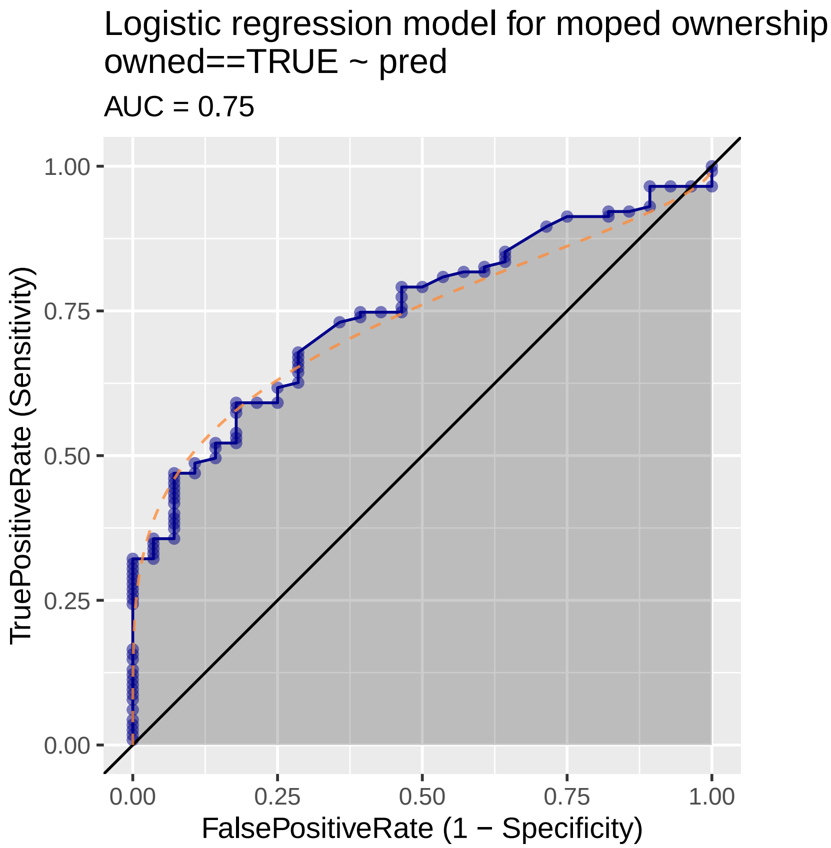
```
## # A tibble: 1 x 8
##   null.deviance df.null logLik   AIC   BIC deviance df.residual  nobs
##           <dbl>   <int>  <dbl> <dbl> <dbl>    <dbl>       <int> <int>
## 1          551.     569  -228.  481.  533.     457.         558   570
```

```
  # calculating pseudo-R-squared
  (pseudoR2 <- 1 - perf$deviance/perf$null.deviance)
```

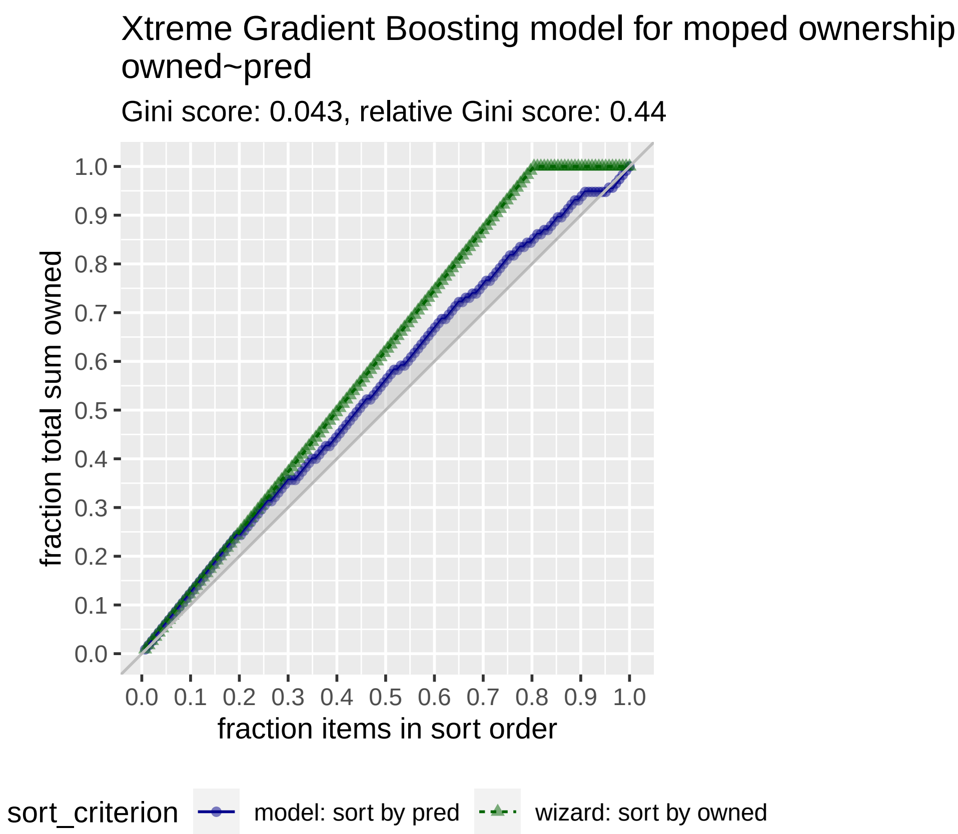
```
## [1] 0.1703809
```

```
  # gain curve plot
  GainCurvePlot(logreg_pred_3, xvar = "pred", "owned", "Logistic regression model for moped ownership")
```

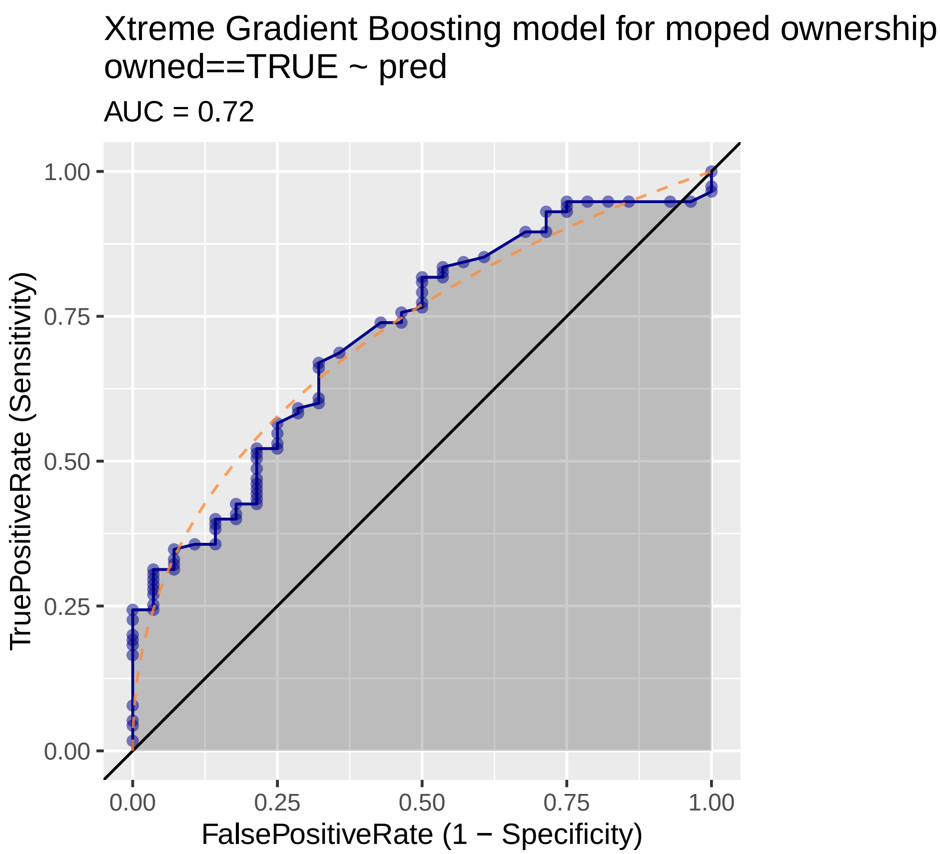

```
# ROC curve
ROCPlot(logreg_pred_3,
        xvar = "pred",
        truthVar = "owned",
        truthTarget = TRUE,
        title = "Logistic regression model for moped ownership",
        add_beta_ideal_curve = TRUE)
```




```
# xgb evaluation
# gain curve plot
GainCurvePlot(xgb_pred_3, xvar = "pred", "owned", "Xtreme Gradient Boosting model for moped ownership
```



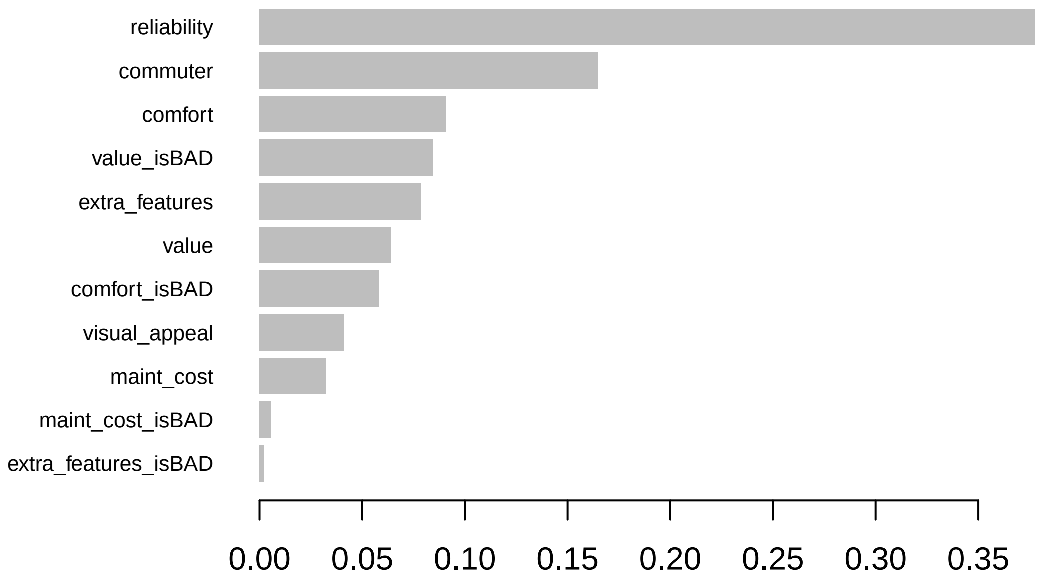

```r
# ROC curve #2
ROCPlot(xgb_pred_3,
        xvar = "pred",
        truthVar = "owned",
        truthTarget = TRUE,
        title = "Xtreme Gradient Boosting model for moped ownership",
        add_beta_ideal_curve = TRUE)
```




```r
# inspecting feature importance
(importance_matrix <-
    xgb.importance(feature_names = colnames_3,
                   model = xgb_model_3))
```

```
##                 Feature        Gain       Cover  Frequency
##  1:         reliability 0.377672316 0.269049279 0.19897959
##  2:            commuter 0.165045427 0.121648789 0.11224490
##  3:             comfort 0.090635409 0.158366960 0.17346939
##  4:         value_isBAD 0.084323314 0.076941085 0.07653061
##  5:      extra_features 0.078835489 0.082149021 0.09693878
##  6:               value 0.064100950 0.133864872 0.11734694
##  7:       comfort_isBAD 0.058018125 0.018619362 0.03061224
##  8:       visual_appeal 0.040935475 0.081284408 0.07142857
##  9:          maint_cost 0.032464469 0.050222740 0.09693878
## 10:    maint_cost_isBAD 0.005550685 0.006311957 0.01020408
## 11: extra_features_isBAD 0.002418342 0.001541527 0.01530612
```

```r
# visualizing feature importance
xgb.plot.importance(importance_matrix[1:11,])
```



Model Evaluation

Explain what the results of your evaluation tell you. You should: - Describe which metric you have selected to compare the models and why - Explain what the outcome of this evaluation tells you about the performance of your models - Identify, based on the evaluation, which you would consider to be the better performing approach

I decided on gain curve plots and ROC curves for comparison of model performance. I also examine pseudo-R^2 values and feature importance for the logistic regression and gradient boosting models respectively. However, gain curve plots and ROC plots are easily generated for both modeling types and thus best suited for comparison.

In scenario 1, gradient boosting performs better in comparisons of both gain curves and ROC plots. The AUC of the gradient boosted model, 0.81, outperforms the regression model (AUC of 0.77) by a narrow margin of 0.04. This relative performance gain holds across the ROC plot - that is, the gradient boosted model performs better regardless of whether specificity or sensitivity is desired. The regression model does have a slight edge in Gini coefficients, but otherwise gradient boosting is a strictly better choice. Feature importance shows that the gradient boosted model relies heavily on reliability and comfort ratings to determine ownership status, indicating that the model is following the data trends previously explored in the faceted density histogram.

Model sets 2 and 3 are the opposite. Logistic regression outperforms gradient boosting in terms of AUC in both cases, with scores of 0.77 vs. 0.72 and 0.75 vs. 0.72 respectively. This edge in performance comes primarily from high-specificity optimizations where logistic regression seems to perform best. For problems where a high degree of sensitivity is desirable, gradient boosting can outperform regression. In model set 3 this is particularly apparent - but when compared to the performance of regression in model set 2 for sensitivity, gradient boosting is at best equal. Both sets of models do show a lower Gini coefficient for gradient boosting even when the third boosted model is compared to the second regression model, but when the population distribution of models is unknown logistic regression has a clear edge in most performance metrics. Also noteworthy is that both models in set 3 are outperformed by their corresponding algorithm in set 2, so that even if the sample isn't proportional to the population including model characteristics improves predictive ability.

So the highest-performing model depends on background information about deployment circumstances and data collection. If information about market share of moped models is unavailable, or if the sample of moped reviews isn't representative of the model concentration in the population, logistic regression with model characteristics is probably the best algorithm for prediction. If the sample of moped reviews examined here is representative, then the gradient boosting model from set 1 is likely to perform the best. The large sample size available in the data suggests that the sample is likely representative of the broader population, but without information on data collection it is difficult to know for certain.

Then there is the question of desired model behavior. The verbiage "... reviews... from people who have never owned the moped..." suggests that *true negatives* and thus *specificity* are the modeling priority.

In that case, gradient boosting outperforms regression **only** when the sample has `model` proportions matching the population. Note that the average AUC of gradient boosted models 1 and 2 is roughly equivalent to the average AUC of the corresponding regression models, but that the regression model *doesn't change in predictive ability* when sample stratification is not performed. So if there are questions about data collection practices, it may behoove a risk-averse firm to opt for the regression model instead.