RMI 8300 Final Project

Asmin Acharya

2024-12-09

```
library(leaps)
library(glmnet)
library(caret)
library(tidyverse)
library(corrplot)
```

Introduction

This report aims to use predictive modeling techniques to evaluate the fair market value (fmv) of variable annuities from an annuity dataset. A variable annuity is essentially a type of financial product that is offered by insurance companies. These financial products are designed to provide individuals with a way to invest their money. Individuals usually receive payments from these investments at retirement and the value of these payments can fluctuate depending on how well the market is performing. This is the key distinction between fixed and variable annuities, where a regular fixed annuity provides fixed-income payments. For this reason, variable annuities can offer potentially higher returns compared to fixed annuities, but at the same time involve more risk.

In the report, we will apply various regression techniques like multiple linear regression, ridge regression, and lasso regression in order to create reliable predictive models. These regression techniques are all supervised learning methods where labeled data is used in order to understand the relationships between our predictors and the response variable. This will allow us to make accurate predictions on new or unseen data.

The first thing we will do is prepare and explore our data before we begin our main analysis. In addition to using supervised learning methods, we will also be using unsupervised learning methods like Hierarchical and K-means clustering. By utilizing a combination of supervised and unsupervised methods, we will leverage the strengths of each approach and in the process reduce model complexity, improve accuracy, and increase efficiency. In the end, a total of 9 models will be created and each model will then be evaluated based on its prediction error.

Our last steps involve obtaining forecasts for fmv by applying the nine models we created and conducting a VaR(Value-at-Risk) analysis at the confidence levels of 90% and 95%. The VaR analysis will provide us with important insights into the potential financial exposure associated with variable annuities. In the end, we should be able to accurately predict the fair market value of variable annuities using predictive modeling and performing risk analysis.

Prepare Data

To prepare our data we first load our two datasets using the read.csv() function. Here, the enforce dataset includes all the variables related to the annuity policies and the fmv dataset contains the fair market values for each policy. Once we have both our datasets loaded, we are going to merge the inforce and fmv datasets using the record ID variable. The resulting dataset (inforce2) now contains the information on each annuity's policy as well as its fmv. We only want to include the relevant variables for our analysis, so we make sure

to specify each variable we want to keep and store it in vNames. Finally, we subset the dataset inforce2 and only keep the columns that were specified in vNames. The final dataset dat10k contains all the relevant variables and will be used going forward for our analysis.

```
inforce <- read.csv("C:/Users/achar/Downloads/inforce10k.csv")
fmv <- read.csv("C:/Users/achar/Downloads/fmv_seriatim.csv")

inforce2 <- merge(inforce, fmv, by.x = "recordID", by.y = "RecordID")

vNames <- c("recordID", "gender", "prodType", "issueDate", "matDate", "age",
            "gmdbAmt",  "gmwbAmt", "gmwbBalance", "gmmbAmt", "withdrawal",
            paste("FundValue", 1:10, sep = ""), "fmv")

dat10k <- inforce2[, vNames]
```

Explore Dataset

Before going into our analysis, we must first explore our dataset to get a general understanding of some of our key variables and also how the dataset is structured. Running a summary statistic, we can find the max, min, mean, and median values of our variables. We will not go over every single variable, but we will look at a few. For the age variable, we can see that the average age of the policyholder is around 49.38 and the age of policyholders range from 34.36 to 64.37. Looking at the fair market value (fmv) of the annuity, we find that it has a mean of 18211 and the fmv of the annuities ranges from -30215 to 285182. This suggests that some annuities have a negative fair market value. Based on the summary statistics, the fund values seem to deviate quite heavily for each fund. This variation indicates that there is a lot of diversity in policyholder investments. Getting a glimpse of the dataset we can see all the variables that are in our dataset, the class type of each variable, and the first few observations for each column. Almost all of the variables are of the class double with some being integer types and a couple being character types. In the dataset, there are a total of 10000 observations and 22 different columns/variables. These variables include recordID, gender, prodType, issueDate, matDate, age, gmdbAmt, gmwbAmt, gmwbBalance, gmmbAmt, withdrawal, FundValue 1-10, and fmv. There are no missing values in the entire dataset so we do not have to worry about missing data in our report. We do have a couple of categorical variables in our data such as gender a prodType. The gender variable represents the gender of the policyholder (male or female) and prodType represents the various types of products for each policy. There are a total of 4071 female policyholders and 5929 male policyholders in the dataset. The average fmv for female policyholders is approximately 19596 and for males, it is around 17260. As for prodType, there are five types of annuity products: DBRP, DBRU, MB, WB, and WBSU with DBRP having the most policies (2028) and MB having the least (1959). WBSU has the highest average fmv (50046) and DBRP has the lowest average fmv (-3065).

A correlation plot has also been created to identify relationships between our numerical variables. The plot follows a color scale where red represents a positive correlation and blue represents a negative correlation between two variables. The correlation coefficient is going to be from +1 to -1, where the darker the color is the stronger the negative or positive relationship is. There seems to be a moderate positive correlation between the issueDate and matDate variables. This suggests that as the issue date of the financial product (variable annuities) increases, the maturity date also tends to increase. There is a moderate negative relationship between the variables gmdbAmt and gmwbAmt. This suggests that as the guaranteed minimum death benefit amount increases, the guaranteed minimum withdrawal benefit amount tends to decrease and vice versa. Most of the FundValue variables seem to show a slight positive correlation with the fmv variable. This means that as the fund values increase, the fmv tends to increase as well. In general, this correlation plot provides us valuable insights into the relationships between different variables in our dataset.

```
summary(dat10k[1:22])
```

```
##     recordID        gender            prodType           issueDate
```

```
##  Min.   :    1   Length:10000       Length:10000       Min.   :36528
##  1st Qu.: 2501   Class :character   Class :character   1st Qu.:37811
##  Median : 5000   Mode  :character   Mode  :character   Median :39077
##  Mean   : 5000                                         Mean   :39079
##  3rd Qu.: 7500                                         3rd Qu.:40350
##  Max.   :10000                                         Max.   :41639
##     matDate           age           gmdbAmt          gmwbAmt
##  Min.   :42009   Min.   :34.36   Min.   :     0   Min.   :    0
##  1st Qu.:45559   1st Qu.:42.08   1st Qu.:     0   1st Qu.:    0
##  Median :47084   Median :49.40   Median :     0   Median :    0
##  Mean   :47106   Mean   :49.38   Mean   :135117   Mean   : 7889
##  3rd Qu.:48639   3rd Qu.:56.82   3rd Qu.:256529   3rd Qu.:15042
##  Max.   :52230   Max.   :64.37   Max.   :986536   Max.   :69404
##   gmwbBalance        gmmbAmt         withdrawal       FundValue1
##  Min.   :     0   Min.   :     0   Min.   :     0   Min.   :      0
##  1st Qu.:     0   1st Qu.:     0   1st Qu.:     0   1st Qu.:      0
##  Median :     0   Median :     0   Median :     0   Median :  12639
##  Mean   : 94152   Mean   : 54715   Mean   : 26349   Mean   :  33325
##  3rd Qu.:149781   3rd Qu.:     0   3rd Qu.:     0   3rd Qu.:  49292
##  Max.   :991482   Max.   :499925   Max.   :418565   Max.   :1030517
##    FundValue2        FundValue3       FundValue4       FundValue5
##  Min.   :      0   Min.   :     0   Min.   :     0   Min.   :     0
##  1st Qu.:      0   1st Qu.:     0   1st Qu.:     0   1st Qu.:     0
##  Median :  18844   Median : 12162   Median : 12466   Median : 11326
##  Mean   :  43224   Mean   : 28624   Mean   : 27479   Mean   : 24225
##  3rd Qu.:  60463   3rd Qu.: 41359   3rd Qu.: 41114   3rd Qu.: 36498
##  Max.   :1094840   Max.   :672927   Max.   :547874   Max.   :477843
##    FundValue6       FundValue7       FundValue8       FundValue9
##  Min.   :     0   Min.   :     0   Min.   :     0   Min.   :     0
##  1st Qu.:     0   1st Qu.:     0   1st Qu.:     0   1st Qu.:     0
##  Median : 14898   Median : 11367   Median : 11289   Median :  7591
##  Mean   : 35305   Mean   : 28904   Mean   : 28745   Mean   : 27191
##  3rd Qu.: 53520   3rd Qu.: 44194   3rd Qu.: 44972   3rd Qu.: 41633
##  Max.   :819144   Max.   :794471   Max.   :726032   Max.   :808214
##   FundValue10          fmv
##  Min.   :     0   Min.   :-30215
##  1st Qu.:     0   1st Qu.: -2593
##  Median :  6525   Median :  7011
##  Mean   : 26666   Mean   : 18211
##  3rd Qu.: 41284   3rd Qu.: 31363
##  Max.   :709233   Max.   :285182
```

```
glimpse(dat10k[1:22])
```

```
## Rows: 10,000
## Columns: 22
## $ recordID    <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17,~
## $ gender      <chr> "M", "M", "M", "F", "M", "F", "F", "M", "M", "F", "F", "F"~
## $ prodType    <chr> "DBRP", "DBRU", "DBRU", "MB", "DBRU", "DBRU", "WBSU", "DBR~
## $ issueDate   <int> 39531, 39797, 41191, 36857, 37011, 38973, 38135, 37349, 40~
## $ matDate     <int> 50123, 50389, 48499, 46353, 45412, 49200, 47631, 43924, 48~
## $ age         <dbl> 41.99726, 36.24657, 44.84931, 61.44658, 52.82466, 64.26301~
## $ gmdbAmt     <dbl> 374146.03, 457353.94, 510003.60, 0.00, 729132.53, 500190.4~
## $ gmwbAmt     <dbl> 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 20023.887, 0.000~
```

```
## $ gmwbBalance <dbl> 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 286055.5, 0.0, 399742.8, 276~
## $ gmmbAmt     <dbl> 0.0, 0.0, 0.0, 239498.4, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 287~
## $ withdrawal  <dbl> 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 2004~
## $ FundValue1  <dbl> 50403.664, 121790.814, 0.000, 0.000, 0.000, 0.000, 30375.4~
## $ FundValue2  <dbl> 58618.597, 0.000, 134450.566, 65660.716, 156811.368, 0.000~
## $ FundValue3  <dbl> 32238.017, 89283.096, 117240.446, 0.000, 0.000, 0.000, 254~
## $ FundValue4  <dbl> 43982.263, 0.000, 0.000, 46861.324, 0.000, 0.000, 25525.17~
## $ FundValue5  <dbl> 36676.919, 0.000, 93373.409, 35654.508, 0.000, 0.000, 2130~
## $ FundValue6  <dbl> 53832.832, 127998.010, 131038.143, 0.000, 0.000, 0.000, 32~
## $ FundValue7  <dbl> 40578.413, 0.000, 122941.426, 37982.317, 101590.018, 0.000~
## $ FundValue8  <dbl> 0.000, 88402.194, 0.000, 45684.692, 112713.620, 409299.896~
## $ FundValue9  <dbl> 38986.644, 102256.200, 0.000, 42854.973, 0.000, 0.000, 284~
## $ FundValue10 <dbl> 45030.63, 88598.72, 0.00, 0.00, 115406.09, 0.00, 28404.10,~
## $ fmv         <dbl> -6092.4623, -26637.2018, -16507.2193, 23539.2665, -3530.13~
```

```
dim(dat10k)
```

```
## [1] 10000    22
```

```
names(dat10k)
```

```
##  [1] "recordID"    "gender"      "prodType"    "issueDate"   "matDate"
##  [6] "age"         "gmdbAmt"     "gmwbAmt"     "gmwbBalance" "gmmbAmt"
## [11] "withdrawal"  "FundValue1"  "FundValue2"  "FundValue3"  "FundValue4"
## [16] "FundValue5"  "FundValue6"  "FundValue7"  "FundValue8"  "FundValue9"
## [21] "FundValue10" "fmv"
```

```
sum(is.na(dat10k))
```

```
## [1] 0
```

```
table(dat10k$gender)
```

```
##
##    F    M
## 4071 5929
```

```
table(dat10k$prodType)
```

```
##
## DBRP DBRU   MB   WB WBSU
## 2028 2018 1959 1991 2004
```

```
aggregate(dat10k$fmv,
          by = list(dat10k$gender),
          FUN = mean)
```

```
##   Group.1        x
## 1       F 19596.07
## 2       M 17259.95
```

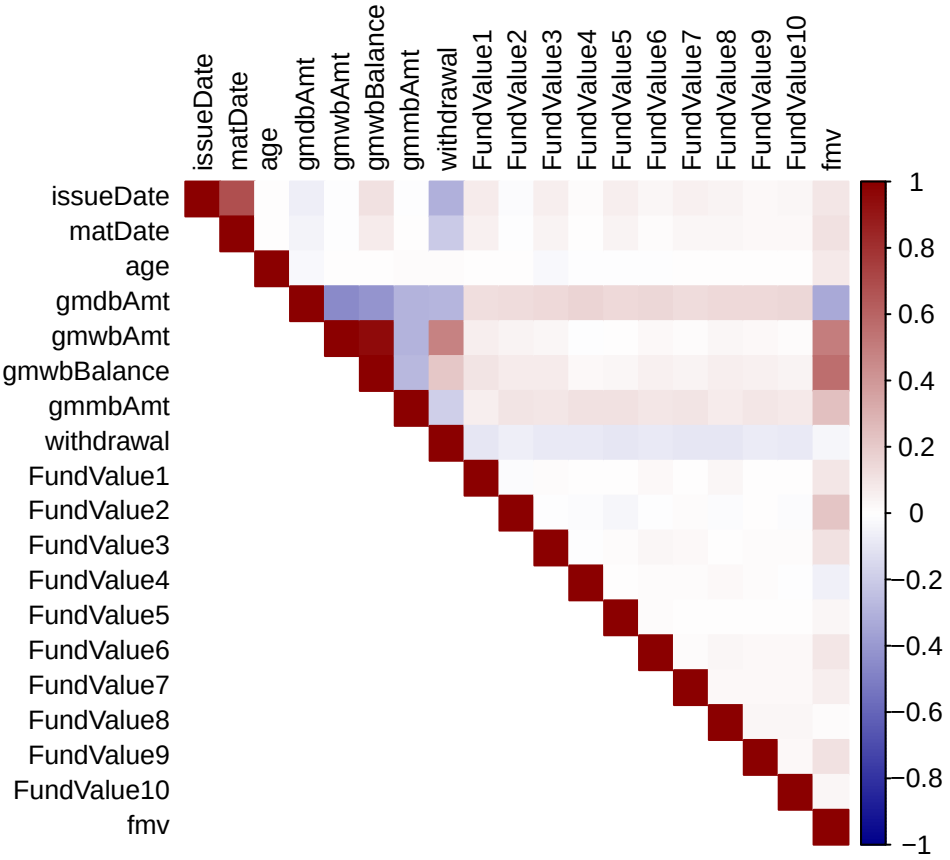
```
aggregate(dat10k$fmv,
          by = list(dat10k$prodType),
          FUN = mean)
```

```
##   Group.1         x
## 1    DBRP -3065.277
## 2    DBRU  7479.611
## 3      MB 28838.170
## 4      WB  8260.669
## 5    WBSU 50045.600
```

```
cor_matrix <- cor(dat10k[, c("issueDate","matDate","age","gmdbAmt","gmwbAmt",
                             "gmwbBalance","gmmbAmt","withdrawal","FundValue1",
                             "FundValue2","FundValue3","FundValue4",
                             "FundValue5","FundValue6","FundValue7",
                             "FundValue8","FundValue9","FundValue10","fmv")])

col <- colorRampPalette(c("darkblue", "white", "darkred"))(200)

corrplot(cor_matrix, method = "color", type = "upper",
         col = col, tl.pos = 'lt', tl.col = 'black',
         number.cex = 0.9, tl.cex = 0.8)
```



Training and Testing Sets

By exploring the data, we have found some general insights. Now, we can move onto splitting our data into training and testing sets which will be needed in order to build and validate our models. We first use the set.seed() function to ensure that the results are reproducible. The dat10k dataset is split where the training data contains 80% of the data and the testing data contains 20% of the remaining data.

At the same time, we have to change our categorical variables into numeric or binary values. We transform our gender variables into a dummy variable where 1 represents a male policyholder while 0 represents a female policyholder. The same thing is done with prodType where a dummy variable is created for each category except for one category. In the end, we will end up creating 4 dummy variables for prodType (DBRU, MB, WB & WBSU) and DBRP will be our reference category which is essentially the baseline against which the other categories are compared. This is done so that we can avoid multicollinearity in our models. It is very important that we transform our categorical variables into binary variables as regression-based models require numeric values. If we do not do this then we will get errors when running our models. Once these conversions are applied to the testing and training data, we are going to prepare a matrix for our predictors and response vector. The variable X contains all the columns in our updated training data except recordID and fmv and y just contains the fmv response variable from the training data. We do the same process again for X_test and y_test which includes the variables from our testing data. Finally, we combine the fmv response and predictor matrix into two separate datasets called train_data_combined and test_data_combined. Since we have already created binary variables for prodType, we can remove them from the updated training and testing sets.

```
set.seed(945)
train_index <- createDataPartition(dat10k$fmv, p = 0.8, list = FALSE)
train_data <- dat10k[train_index, ]
test_data <- dat10k[-train_index, ]
```

```
train_data$gender <- as.numeric(train_data$gender == "M")  # M = 1, F = 0
train_data <- cbind(train_data, model.matrix(~ prodType - 1, train_data)[,-1])

test_data$gender <- as.numeric(test_data$gender == "M")  # M = 1, F = 0
test_data <- cbind(test_data, model.matrix(~ prodType - 1, test_data)[,-1])

X <- as.data.frame(train_data[, !(names(train_data) %in% c("recordID", "fmv"))])
y <- train_data$fmv

X_test <- as.data.frame(test_data[, !(names(test_data)
                                      %in% c("recordID", "fmv"))])

y_test <- test_data$fmv

train_data_combined <- as.data.frame(cbind(fmv = y, X))
train_data_combined <- subset(train_data_combined, select = -prodType)

test_data_combined <- as.data.frame(cbind(fmv = y_test, X_test))
test_data_combined <- subset(test_data_combined, select = -prodType)
```

Multiple Linear Regression

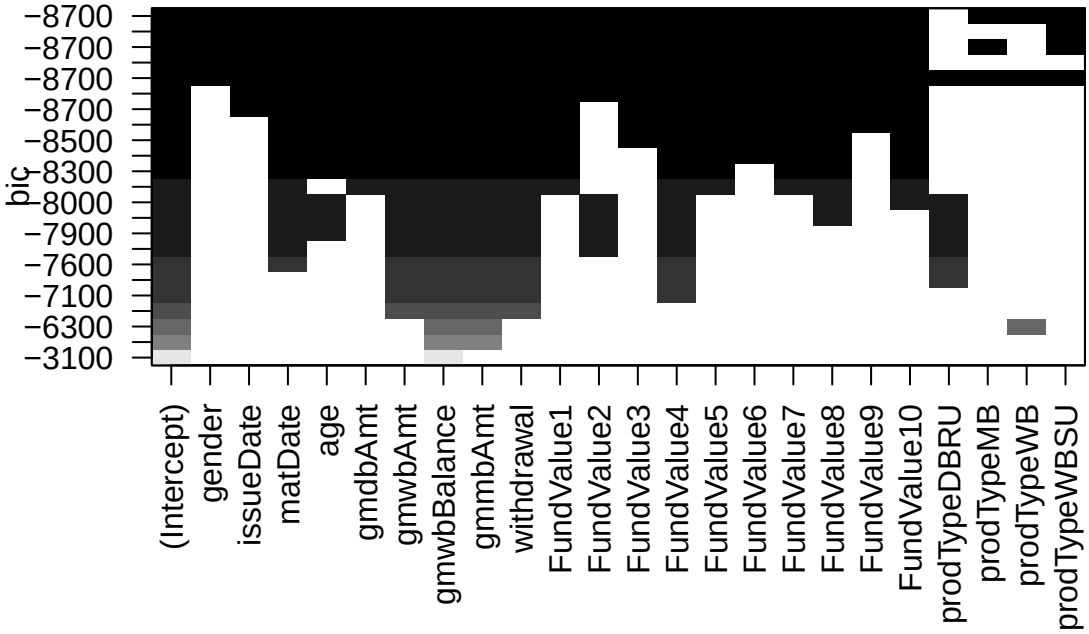
The first model we build is a multiple linear regression using an exhaustive search. First, we have to use the regsubsets() function in order to find the best combination of predictors that optimally predicts our response fmv. The model formula is specified where fmv is our response with all the other variables being our predictors. The exhaustive method will evaluate all possible subsets of predictors and find the one that best fits the model. The nvmax argument limits our number of predictors which in this case will be the total number of columns in train_data_combined. The best predictors with the lowest BIC are given below. The variable best_vars gives us the names of the predictors in our best model. We can also visualize this process by creating a subset selection plot. Here, we can see that the BIC score for each subset model is shown on the y-axis. The best model in this case will be the row that corresponds to the lowest BIC value. This row shows the optimal predictors that are going to be used in our regression model. The plot will show the same number of optimal predictors as those that were stored in the variable best_vars. Once we have found our best predictors, we can go ahead and predict fmv based on the subset of variables we found to be the best predictors. This model is fit to our training data and using the summary() function we can see the estimated coefficient for each predictor. Looking at some of these predictors we find out they change our outcome or response variable. For example, an additional unit of gmdbAmt increases fmv by approximately 0.0896 holding other factors constant. The variable gender has an estimated coefficient of -2137, which suggests that the fmv of a policyholder will decrease by 2137 if they are a male (gender = 1) compared to if they are a female (gender = 0). Most of the FundValue variables have negative coefficient estimates, indicating that larger values in these funds will result in lower fmv.

```
reg_search <- regsubsets(fmv ~ ., data = train_data_combined,
                         method = "exhaustive",
                         nvmax = ncol(train_data_combined))

reg_summary <- summary(reg_search)
best_model_size <- which.min(reg_summary$bic)
best_vars <- names(coef(reg_search, best_model_size))[-1]
best_vars
```

```
##  [1] "gender"       "issueDate"    "matDate"      "age"          "gmdbAmt"
##  [6] "gmwbAmt"      "gmwbBalance"  "gmmbAmt"      "withdrawal"   "FundValue1"
## [11] "FundValue2"   "FundValue3"   "FundValue4"   "FundValue5"   "FundValue6"
## [16] "FundValue7"   "FundValue8"   "FundValue9"   "FundValue10"  "prodTypeMB"
## [21] "prodTypeWB"   "prodTypeWBSU"
```

```
plot(reg_search, scale = "bic")
```




```
formula1 <- as.formula(paste("fmv ~", paste(best_vars, collapse = " + ")))
lm_model1 <- lm(formula1, data = train_data_combined)
summary(lm_model1)
```

```
##
## Call:
## lm(formula = formula1, data = train_data_combined)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -69295  -8476  -2122   5923 269534
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -1.420e+05  6.482e+03 -21.913  < 2e-16 ***
## gender       -2.137e+03  3.975e+02  -5.376 7.82e-08 ***
## issueDate     1.850e+00  2.009e-01   9.209  < 2e-16 ***
## matDate       1.317e+00  1.237e-01  10.641  < 2e-16 ***
## age           2.875e+02  2.276e+01  12.631  < 2e-16 ***
## gmdbAmt       8.959e-02  2.897e-03  30.921  < 2e-16 ***
## gmwbAmt       1.123e+01  4.877e-01  23.037  < 2e-16 ***
## gmwbBalance  -5.336e-01  3.337e-02 -15.992  < 2e-16 ***
## gmmbAmt       2.190e-01  4.954e-03  44.205  < 2e-16 ***
## withdrawal   -6.386e-01  3.043e-02 -20.987  < 2e-16 ***
## FundValue1   -9.232e-02  4.241e-03 -21.767  < 2e-16 ***
```

```
## FundValue2   -2.761e-02  3.431e-03  -8.047 9.70e-16 ***
## FundValue3   -8.559e-02  5.002e-03 -17.113  < 2e-16 ***
## FundValue4   -1.905e-01  5.440e-03 -35.021  < 2e-16 ***
## FundValue5   -1.501e-01  6.131e-03 -24.478  < 2e-16 ***
## FundValue6   -7.228e-02  4.140e-03 -17.457  < 2e-16 ***
## FundValue7   -1.040e-01  4.906e-03 -21.201  < 2e-16 ***
## FundValue8   -1.288e-01  5.201e-03 -24.754  < 2e-16 ***
## FundValue9   -6.796e-02  4.747e-03 -14.318  < 2e-16 ***
## FundValue10  -1.166e-01  5.029e-03 -23.191  < 2e-16 ***
## prodTypeMB   -4.436e+03  1.230e+03  -3.608 0.000311 ***
## prodTypeWB   -4.218e+03  1.243e+03  -3.393 0.000693 ***
## prodTypeWBSU -6.014e+03  1.179e+03  -5.102 3.43e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 17460 on 7977 degrees of freedom
## Multiple R-squared:  0.6733, Adjusted R-squared:  0.6724
## F-statistic: 747.2 on 22 and 7977 DF,  p-value: < 2.2e-16
```

Ridge Regression

The second model we build is going to be a ridge regression model. In order to perform this regression technique, the data for our training and testing set will have to be properly formatted. Our predictors and response variables from the training and testing sets are created separately. The X_train and X_test variables contain the matrix of predictors from the training and testing set while the y_train and y_test variables extract the response variable (fmv) from the the training and testing sets. Now that the data is properly structured, we can go ahead and perform our regression technique.

Ridge regression is very useful when we have multicollinearity or overfitting. This technique is unique in the fact that it includes a regularization term. This regularization term or lambda value adds a penalty that shrinks the size of the coefficients in our model. This will help us deal with problems such as overfitting or multicollinearity as we can control the influence of less relevant predictors. The ridge regression model in the following steps will be created based on our optimal or minimum lambda value. It is called the optimal lambda value because it minimizes the mean cross-validated error.

In our code, the cv.glmnet() function is used to perform the cross-validation for our Ridge regression. In the next step, we build our Ridge regression model using the optimal lambda value that we found in the previous step. The model is built by using the glmnet() function where we reference our X_train and y_train variables, set our alpha = 0 which specifies ridge regression, and set our lambda to the optimal value we found during CV. In addition to building our model, we can also extract the coefficients from the final fitted Ridge regression. The coefficients here are retrieved from the model using the optimal lambda value. Due to the penalty term, some of the coefficients here have gotten smaller compared to our linear regression model. These coefficients can be interpreted based on their relative magnitude. The larger the absolute value of a coefficient, the bigger the impact it has on fmv. It remains that positive coefficients indicate that an increase in the predictors leads to an increase in the expected fmv and negative coefficients indicate that a decrease in the predictors leads to a decrease in the expected fmv. Just as an example, the variable prodTypeWB with a coefficient estimate of -12335.50 has a substantial negative effect compared to most of the other predictors.

```r
X_train <- model.matrix(~ . -fmv, data = train_data_combined)[,-1]
X_test <- model.matrix(~. -fmv, data = test_data_combined)[,-1]
y_train <- train_data$fmv
y_test <- test_data$fmv

ridge_cv <- cv.glmnet(X_train, y_train, alpha = 0)
ridge_cv$lambda.min
```

```
## [1] 1742.951
```

```
ridge_model <- glmnet(X_train, y_train, alpha=0, lambda=ridge_cv$lambda.min)
ridge_coefficients <- coef(ridge_model, s = ridge_cv$lambda.min)
print(ridge_coefficients)
```

```
## 24 x 1 sparse Matrix of class "dgCMatrix"
##                            s1
## (Intercept)  -7.847339e+04
## gender       -2.090150e+03
## issueDate     2.700006e-01
## matDate       1.200754e+00
## age           2.687659e+02
## gmdbAmt       1.958582e-02
## gmwbAmt       1.021939e+00
## gmwbBalance   7.029853e-02
## gmmbAmt       1.063740e-01
## withdrawal   -2.366478e-02
## FundValue1   -2.413081e-02
## FundValue2    2.927486e-02
## FundValue3   -6.502437e-03
## FundValue4   -1.008895e-01
## FundValue5   -5.013820e-02
## FundValue6   -1.027596e-02
## FundValue7   -3.201982e-02
## FundValue8   -5.439560e-02
## FundValue9    4.472746e-04
## FundValue10  -4.035565e-02
## prodTypeDBRU  7.002364e+03
## prodTypeMB    6.399894e+03
## prodTypeWB   -1.233550e+04
## prodTypeWBSU  1.174526e+04
```

LASSO Regression

The last supervised learning method we will use is the LASSO regression. LASSO is similar to Ridge as it also contains a regularization parameter that controls the magnitude of the coefficients, but it is quite different. First of all, the penalty term in LASSO can sometimes shrink coefficients to exactly zero which is not the case for Ridge regression. LASSO is best used when we believe that some of our predictors are completely irrelevant to our model. Ridge is better utilized when we believe that many of our predictors are important for our model. LASSO can also handle multicollinearity issues like Ride and remove irrelevant predictors by setting their coefficients to zero.

The process of building a LASSO regression is very similar to Ridge. Just like Ridge, we find the optimal lambda value by performing the cross-validation for our LASSO regression. The only difference here is that we set the alpha = 1 which specifies that we want to use a LASSO and not a Ridge. Using the glmnet() function, we fit a LASSO model to the training data and set our lambda equal to the optimal or minimum lambda that we found using CV. The coefficients can now be retrieved from the LASSO model using the optimal lambda. The interpretation of these coefficients is the same as the way we interpreted the Ridge regression coefficients. None of our coefficients here have been set to zero, so it seems that all of our variables have some predictive power. All the FundValue variables have negative coefficients indicating that higher fund values lead to a decrease in fmv. Additionally, variables like gender, gmwbBalance, and withdrawal also tend to decrease fmv. Variables like age, gmwbAmt, and gmmbAmt have a positive impact which suggests that they tend to increase fmv.

```r
lasso_cv <- cv.glmnet(X_train, y_train, alpha = 1)
lasso_cv$lambda.min
```

```
## [1] 1.742951
```

```r
lasso_model <- glmnet(X_train, y_train, alpha = 1, lambda = lasso_cv$lambda.min)
lasso_coefficients <- coef(lasso_model, s = lasso_cv$lambda.min)
print(lasso_coefficients)
```

```
## 24 x 1 sparse Matrix of class "dgCMatrix"
##                            s1
## (Intercept)  -1.400155e+05
## gender       -2.119468e+03
## issueDate     1.811295e+00
## matDate       1.314203e+00
## age           2.868156e+02
## gmdbAmt       9.220875e-02
## gmwbAmt       1.026956e+01
## gmwbBalance  -4.662815e-01
## gmmbAmt       2.217994e-01
## withdrawal   -5.788587e-01
## FundValue1   -9.416477e-02
## FundValue2   -2.941163e-02
## FundValue3   -8.807548e-02
## FundValue4   -1.934179e-01
## FundValue5   -1.535064e-01
## FundValue6   -7.422263e-02
## FundValue7   -1.064974e-01
## FundValue8   -1.311991e-01
## FundValue9   -7.027740e-02
## FundValue10  -1.190888e-01
## prodTypeDBRU -9.168229e+02
## prodTypeMB   -4.782177e+03
## prodTypeWB   -5.851906e+03
## prodTypeWBSU -5.443898e+03
```

Hierarchical Clustering Method

In the next sections, we will use both unsupervised and supervised methods in order to build six additional models. Unsupervised techniques allow us to reduce the size of our dataset by selecting a certain sample from each cluster. Overall, this makes our data set more manageable, speeds up model training, and helps us capture underlying patterns in our data. The first unsupervised technique we will look at is Hierarchical clustering. Since we are using this method for the first time, we have to format our data accordingly.

In our code, we create a variable called dist_matrix which is going to standardize the predictor variables in our training data which will ensure that all of our variables are on the same scale. Next, the distance between each pair of observations is computed in the standardized data. This will give us an output or matrix where each element represents the distance between two observations. The hclust() function is used to perform the Hierarchical clustering and the cutree() function cuts the hierarchical tree into exactly 200 clusters. The variable cluster_samples is going to iterate over these 200 clusters and randomly select one observation from each cluster. Once our cluster samples are chosen, new training datasets are going to be created based on these selected observations. The result is a reduced training dataset, where train_selected_h,

X_train_selected_h, and y_train_selected_h contain our 200 selected observations (one from each cluster). We are now ready to use our previous supervised methods (multiple linear, Ridge, and LASSO) and create 3 new models based on our cluster-selected data.

```
dist_matrix <- dist(scale(X_train))
hclust_result <- hclust(dist_matrix)
clusters_h <- cutree(hclust_result, k = 200)

cluster_samples <- sapply(1:200, function(i) {
  cluster_indices <- which(clusters_h == i)
  sample(cluster_indices, 1)
})

train_selected_h <- train_data_combined[cluster_samples, ]
X_train_selected_h <- X_train[cluster_samples, ]
y_train_selected_h <- y_train[cluster_samples]
```

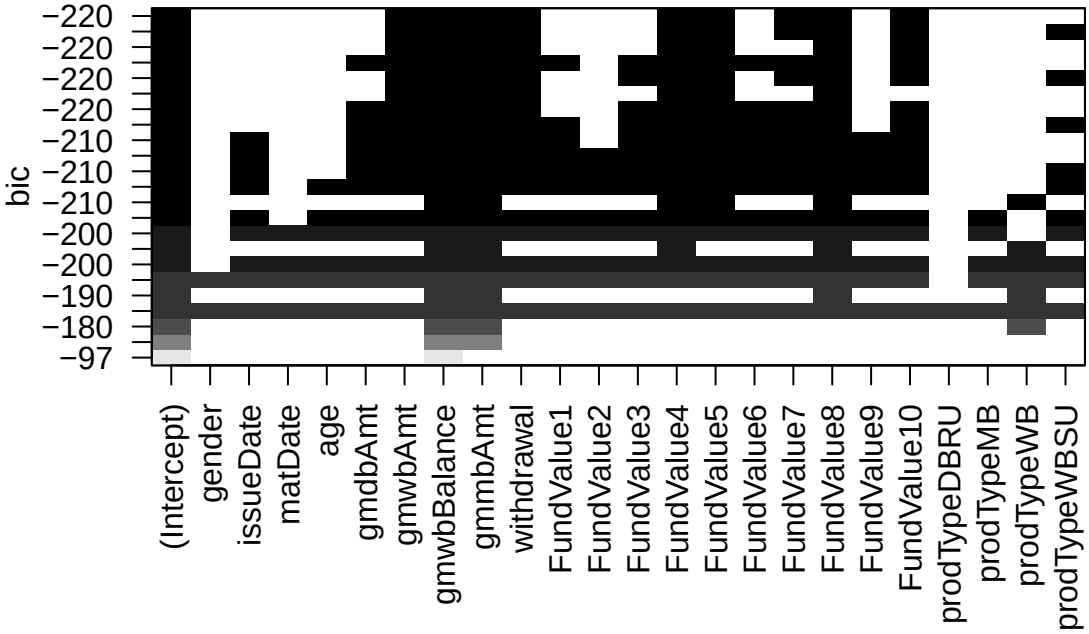
Multiple Linear Regression

The process of creating our three new models is going to be very similar to before when we were only focused on using supervised methods. The only difference is that we have to use the new training data that we created using Hierarchical clustering in order to build our models.

Our multiple linear regression model will be created just like before, where we used an exhaustive search to find the best subset of predictors for predicting fmv. The best subset of predictors is chosen based on the lowest BIC value. The variable best_vars_h gives us the names of the predictors to include in our best model, which happens to be the variables gmwbAmt, gmwbBalance, gmmbAmt, withdrawal, FundValue4, FundValue5 FundValue7, FundValue8, and FundValue10. If we look at the model summary, we can see that there is a significant difference between this linear regression model compared to our first one. First of all, there are far fewer predictors in this model which may suggest hierarchical clustering was able to capture key patterns in the data while at the same time reducing unnecessary complexity. This model also has a higher adjusted R-squared value (0.7319) compared to our first linear regression model (0.6724). This higher adjusted R-squared value suggests that the unsupervised method allowed us to highlight the most significant relationships. Additionally, an exhaustive search was performed on a smaller and more focused dataset. The process is better able to isolate the most significant predictors and leave out more irrelevant variables.

```
reg_search_h <- regsubsets(fmv ~., data = train_selected_h,
                           method = "exhaustive",
                           nvmax = ncol(train_selected_h))

plot(reg_search_h, scale = "bic")
```




```
reg_summary_h <- summary(reg_search_h)
best_model_size_h <- which.min(reg_summary_h$bic)
best_vars_h <- names(coef(reg_search_h, best_model_size_h))[-1]
best_vars_h
```

```
## [1] "gmwbAmt"     "gmwbBalance" "gmmbAmt"     "withdrawal"  "FundValue4"
## [6] "FundValue5"  "FundValue7"  "FundValue8"  "FundValue10"
```

```
formula2 <- as.formula(paste("fmv ~", paste(best_vars_h, collapse = " + ")))
lm_model2 <- lm(formula2, data = train_selected_h)
summary(lm_model2)
```

```
##
## Call:
## lm(formula = formula2, data = train_selected_h)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -43119 -12153  -3145  10046  86119
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.290e+04  2.906e+03   4.441 1.52e-05 ***
## gmwbAmt      1.202e+01  1.856e+00   6.473 7.98e-10 ***
## gmwbBalance -6.578e-01  1.298e-01  -5.067 9.53e-07 ***
```

```
## gmmbAmt      1.068e-01  9.646e-03  11.070  < 2e-16 ***
## withdrawal  -9.020e-01  1.092e-01  -8.260 2.43e-14 ***
## FundValue4  -1.093e-01  2.268e-02  -4.821 2.92e-06 ***
## FundValue5  -9.816e-02  2.216e-02  -4.429 1.60e-05 ***
## FundValue7  -4.138e-02  1.659e-02  -2.494  0.01349 *
## FundValue8  -1.124e-01  2.003e-02  -5.610 7.06e-08 ***
## FundValue10 -5.071e-02  1.732e-02  -2.928  0.00383 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 21910 on 190 degrees of freedom
## Multiple R-squared:  0.744,  Adjusted R-squared:  0.7319
## F-statistic: 61.36 on 9 and 190 DF,  p-value: < 2.2e-16
```

Ridge Regression

A Ridge regression model is created by using our selected observations from Hierarchical clustering. The variable X_train_selected_h contains the matrix of predictors whereas the variable y_train_selected_h contains the response variable. Using the cross-validation function, we find the optimal lambda value to be approximately 2718. The Ridge regression model is then created based on the optimal lambda value. The coefficients from the model can now be retrieved from the fitted model using the optimal lambda. We can see that in this model our best lambda value is higher compared to our first ridge regression model. It is important to understand that we performed the ridge regression on the full dataset, but here we are using Hierarchical clustering. A higher lambda suggests that the second Ridge model needs a stronger penalty to achieve the best predictive accuracy. The Hierarchical clustering method may have improved the quality of our data by focusing on key patterns. The higher lambda value could be an indication of this improved data quality. Looking at the coefficients, we can see that FundValue has a relatively small effect on fmv. The negative coefficient of the variable prodTypeWB suggests that there is a substantial negative effect on fmv compared to the baseline category. The other product type categories show a positive effect on fmv compared to the baseline category.

```
ridge_cv_h <- cv.glmnet(X_train_selected_h, y_train_selected_h, alpha = 0)
ridge_cv_h$lambda.min
```

```
## [1] 2718.075
```

```
ridge_model_h <- glmnet(X_train_selected_h, y_train_selected_h, alpha = 0,
                        lambda = ridge_cv_h$lambda.min)
ridge_coefficients_h <- coef(ridge_model_h, s = ridge_cv_h$lambda.min)
print(ridge_coefficients_h)
```

```
## 24 x 1 sparse Matrix of class "dgCMatrix"
##                          s1
## (Intercept)  -3.164755e+04
## gender       -1.148318e+03
## issueDate     6.692706e-02
## matDate       5.918271e-01
## age           2.628025e+02
## gmdbAmt       2.480678e-03
## gmwbAmt       1.130761e+00
## gmwbBalance   8.525221e-02
## gmmbAmt       7.797317e-02
```

```
## withdrawal   -1.717226e-01
## FundValue1   -1.748491e-02
## FundValue2    1.575589e-02
## FundValue3   -2.451412e-02
## FundValue4   -1.031664e-01
## FundValue5   -8.544195e-02
## FundValue6   -1.387242e-02
## FundValue7   -3.948216e-02
## FundValue8   -1.078036e-01
## FundValue9    7.297825e-03
## FundValue10  -4.905341e-02
## prodTypeDBRU  5.896700e+03
## prodTypeMB    1.128117e+04
## prodTypeWB   -2.637866e+04
## prodTypeWBSU  9.452275e+03
```

LASSO Regression

LASSO Regression is performed on the data that was pre-processed using the Hierarchical clustering method. Similarly, like the previous Ridge regression model, we are going to find the optimal lambda using CV and fit a LASSO regression model utilizing the same optimal lambda. The minimum lambda for this LASSO model was approximately equal to 2.72. Once we fit the LASSO regression model based on this lambda, the coefficients of the model can be retrieved. Comparing the lambdas, we find that the optimal lambda of the first LASSO model was lower than the second LASSO model here. Again, just like the Ridge regression models the higher lambda in the second LASSO model indicates a stronger penalty. The Hierarchical clustering process may have reduced the complexity or variability of the data within each cluster. This may explain the stronger penalty term in the second LASSO model compared to the first. The FundValue coefficients do not seem to have a significant effect on fmv. Variables like age and gmwbAmt have a positive effect on our response variable whereas the product type categorical variables have a negative effect on fmv in comparison to the baseline category. None of our coefficients have been shrunk to zero in this LASSO model. It suggests that most of our predictors here are relevant and informative.

```
lasso_cv_h <- cv.glmnet(X_train_selected_h, y_train_selected_h, alpha = 1)
lasso_cv_h$lambda.min
```

```
## [1] 2.983084
```

```
lasso_model_h <- glmnet(X_train_selected_h, y_train_selected_h, alpha = 1,
                        lambda = lasso_cv_h$lambda.min)
lasso_coefficients_h <- coef(lasso_model_h, s = lasso_cv_h$lambda.min)
print(lasso_coefficients_h)
```

```
## 24 x 1 sparse Matrix of class "dgCMatrix"
##                            s1
## (Intercept)  -9.674921e+04
## gender       -8.068023e+02
## issueDate     1.597944e+00
## matDate       9.567102e-01
## age           3.224491e+02
## gmdbAmt       6.546385e-02
## gmwbAmt       1.270057e+01
## gmwbBalance  -5.936577e-01
```

```
## gmmbAmt       2.451831e-01
## withdrawal   -9.082485e-01
## FundValue1   -8.739255e-02
## FundValue2   -4.143567e-02
## FundValue3   -1.128339e-01
## FundValue4   -2.126627e-01
## FundValue5   -2.026305e-01
## FundValue6   -8.713428e-02
## FundValue7   -1.286688e-01
## FundValue8   -2.047151e-01
## FundValue9   -7.796113e-02
## FundValue10  -1.462529e-01
## prodTypeDBRU -1.029297e+03
## prodTypeMB   -2.610708e+04
## prodTypeWB   -1.502971e+04
## prodTypeWBSU -2.668217e+04
```

K-means Clustering Method

The second unsupervised method we will use in combination with our supervised methods is going be K-means clustering. The purpose of K-means clustering is quite similar to Hierarchical clustering, where we randomly select a data point from each cluster. Like with Hierarchical clustering, this will reduce the size of our dataset while still allowing us to capture the variability within our data. The key difference between the two unsupervised techniques is that K-means clustering divides our data into k number of clusters whereas Hierarchical clustering is a tree-based method. For this reason, K-means is computationally faster when dealing with large datasets compared to Hierarchical.

To use the method, we will first set a seed so that this random sampling process will produce the same results every time we run the code. Using the kmeans() function, we can perform the K-mean clustering on our training predictors dataset (X_train). The scale function will standardize our data and the centers value is going to be set to 200 which indicates that the K-means method will partition the data into 200 clusters. The clusters_samples_k variables will iterate over these 200 clusters and the cluster_indices variable is going to randomly sample one data point from each cluster. The conditional statement makes sure that we select a data point if there are data points in cluster i, if the cluster is empty then it returns a NA value. In this case, there are no missing values in our cluster sample. This new unsupervised method requires that we format our training data if we want to use the K-means cluster sample in our models. The variables train_selected_k, X_train_selected_k, and y_train_selected_k include the data we obtained from the K-means cluster sample. We have successfully formatted our training data so we will now create 3 models using the supervised methods.

```r
set.seed(945)
kmean_result <- kmeans(scale(X_train), centers = 200)

cluster_samples_k <- sapply(1:200, function(i) {
  cluster_indices <- which(kmean_result$cluster == i)
  if(length(cluster_indices) > 0) {
    sample(cluster_indices, 1)
  } else {
    NA
  }
})

cluster_samples_k <- cluster_samples_k[!is.na(cluster_samples_k)]
sum(is.na(cluster_samples_k))
```

```
## [1] 0
```

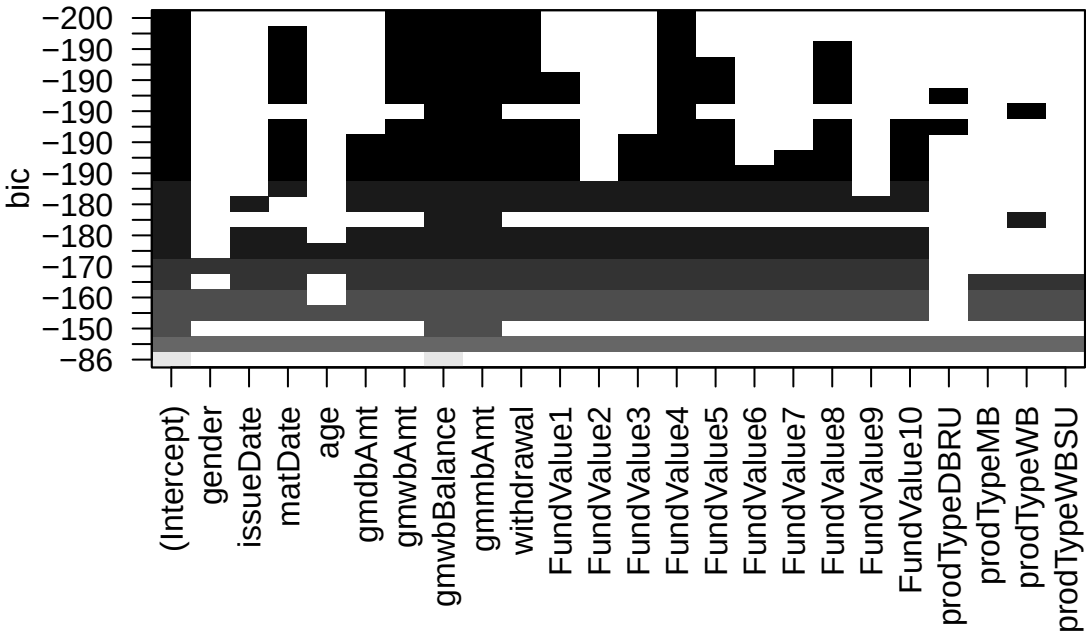
```
train_selected_k <- train_data_combined[cluster_samples_k, ]
X_train_selected_k <- X_train[cluster_samples_k, ]
y_train_selected_k <- y_train[cluster_samples_k]
```

Multiple Linear Regression

The exhaustive search method here can evaluate all possible combinations of predictors to find the best subset. The model is being fitted on the train_selected_k data, which contains our sample observations from the K-means clusters. The nvmax is set to the number of columns that are present in our K-means training data. The best predictors for this linear regression model based on BIC include: gmwbAmt, gmwbBalance, gmmbAmt, withdrawal, and FundValue4. If we look at our model coefficients, we can see that gmwbAmt and gmmbAmt have a positive impact on our response variable (fmv) while the other variables have a negative impact on the response. All the predictors in this model are statistically significant at the 0.01 level, so they could have a meaningful impact on fmv. The adjusted R-squared of this model (0.6707) is lower than the other 2 linear regression models we created earlier. The first linear model had an adjusted R-squared = 0.6724 and the second model that was created using Hierarchical clustering had an adjusted R-squared = 0.7319. The one thing that is obvious to see here is that the number of predictors in our best linear regression model is decreasing as we switch from using just a supervised method to using a combination of unsupervised and supervised methods. The K-means clustering approach resulted in the fewest predictors in our third linear regression model. This suggests that the clusters that were formed by the K-means may have found a more condensed or focused sample data. The K-means clustering method seems to have simplified our model and found a more efficient set of predictors. Even though this model has a lower adjusted R-squared compared to the other two, it has the fewest amount of predictors. This is especially true if we this to the first linear regression model which has many predictors, but a slightly higher adjusted R-squared.

```
reg_search_k <- regsubsets(fmv~., data = train_selected_k,
                           method = "exhaustive",
                           nvmax = ncol(train_selected_k))

plot(reg_search_k, scale = "bic")
```




```
reg_summary_k <- summary(reg_search_k)
best_model_size_k <- which.min(reg_summary_k$bic)
best_vars_k <- names(coef(reg_search_k, best_model_size_k))[-1]
best_vars_k
```

```
## [1] "gmwbAmt"     "gmwbBalance" "gmmbAmt"     "withdrawal"  "FundValue4"
```

```
formula3 <- as.formula(paste("fmv ~", paste(best_vars_k, collapse = " + ")))
lm_model3 <- lm(formula3, data = train_selected_k)
summary(lm_model3)
```

```
##
## Call:
## lm(formula = formula3, data = train_selected_k)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -70939  -8762  -2574   4728  70032
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) 1930.61912 2072.63036   0.931  0.35276
## gmwbAmt        9.60659    1.81168   5.303 3.09e-07 ***
## gmwbBalance   -0.51055    0.12580  -4.059 7.16e-05 ***
## gmmbAmt        0.09962    0.01013   9.834  < 2e-16 ***
```

```
## withdrawal    -0.63226    0.10091  -6.266 2.35e-09 ***
## FundValue4    -0.08332    0.02173  -3.834  0.00017 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 19390 on 194 degrees of freedom
## Multiple R-squared:  0.6789, Adjusted R-squared:  0.6707
## F-statistic: 82.05 on 5 and 194 DF,  p-value: < 2.2e-16
```

Ridge Regression

Using the K-means cluster sample data we will build a Ridge regression and then evaluate our results. Inside the cv.glmnet() function we include our matrix of predictors and response variable from the training data which we obtained using K-means clustering. The optimal lambda value for this model is approximately equal to 2086.52. We then build and extract our Ridge regression coefficients based on our optimal lambda value. If we compare the optimal lambda value across all three of our Ridge regression models, we find that the lowest value belongs to our first Ridge model. In our first model, we did not use a combination of supervised and unsupervised methods it was just the supervised method. The smaller lambda value in our first Ridge model suggests that it is not overly penalizing the coefficients, but applying some shrinkage to them. There is quite a high lambda value where we used the Hierarchical clustering method. This method may have introduced higher multicollinearity between the predictors which would require a higher lambda value to penalize these predictors. The lambda value using the K-means clustering method may have reduced complexity in our model compared to Hierarchical clustering. Looking at coefficients in this model, we can see that variables like age and gmwbAmt seem to have a positive effect on our response variable. The variable prodTypeWB stands out as it has a large negative coefficient, suggesting that this product type negatively influences our outcome in comparison to the baseline category.

```
ridge_cv_k <- cv.glmnet(X_train_selected_k, y_train_selected_k, alpha = 0)
ridge_cv_k$lambda.min
```

```
## [1] 2086.521
```

```
ridge_model_k <- glmnet(X_train_selected_k, y_train_selected_k, alpha = 0,
                        lambda = ridge_cv_k$lambda.min)
ridge_coefficients_k <- coef(ridge_model_k, s = ridge_cv_k$lambda.min)
print(ridge_coefficients_k)
```

```
## 24 x 1 sparse Matrix of class "dgCMatrix"
##                           s1
## (Intercept)  -5.546691e+04
## gender        1.528429e+03
## issueDate     2.105223e-02
## matDate       1.182862e+00
## age           6.159873e+01
## gmdbAmt       4.855219e-03
## gmwbAmt       9.347879e-01
## gmwbBalance   7.448712e-02
## gmmbAmt       8.859283e-02
## withdrawal   -7.407685e-02
## FundValue1   -3.384152e-02
## FundValue2    7.623239e-03
## FundValue3   -3.044297e-02
```

```
## FundValue4   -9.597700e-02
## FundValue5   -5.609614e-02
## FundValue6   -3.323492e-03
## FundValue7   -1.517297e-02
## FundValue8   -5.400429e-02
## FundValue9    2.245010e-02
## FundValue10  -3.088850e-02
## prodTypeDBRU  5.881576e+03
## prodTypeMB    8.160269e+03
## prodTypeWB   -1.459466e+04
## prodTypeWBSU  1.107945e+04
```

LASSO Regression

The last model we build is going to be a LASSO regression model based on our K-means cluster sample data. Using the cv.glmnet() function we find that the optimal lambda value for this LASSO model is approximately equal to 2.087. We then build and extract our LASSO model coefficients based on this optimal lambda value. Again, similar to our Ridge regression models the lowest lambda value belongs to the first LASSO model. In the first LASSO model, we just used the supervised method and did not use a combination of supervised and unsupervised methods like our other two models. The highest optimal lambda value out of our three LASSO models was in the second model. This model was created using a combination of the Hierarchical clustering method and the LASSO regression technique. The larger lambda value indicates that Hierarchical clustering may have introduced higher complexity or multicollinearity, which requires a stronger penalty. This larger penalty will reduce our risk of overfitting, but at the same time, we may have to sacrifice some predictive power. K-means clustering may have provided less complexity compared to the Hierarchical clustering as it had a lower penalty term. If we look at the coefficients, we can see that the LASSO model did not shrink any of them to zero. All of the variables have a non-zero coefficient suggesting that each predictor was useful, even if some of their effects might be relatively small. Variables such as prodTypeMB and prodTypeWB have large negative coefficients, indicating that they play an important role in reducing our response variable. The age and gmwbAmt variables have a moderate, but noticeable positive impact on our response.

```
lasso_cv_k <- cv.glmnet(X_train_selected_k, y_train_selected_k, alpha = 1)
lasso_cv_k$lambda.min
```

```
## [1] 2.086521
```

```
lasso_model_k <- glmnet(X_train_selected_k, y_train_selected_k, alpha = 1,
                        lambda = lasso_cv_k$lambda.min)
lasso_coefficients_k <- coef(lasso_model_k, s = lasso_cv_k$lambda.min)
print(lasso_coefficients_k)
```

```
## 24 x 1 sparse Matrix of class "dgCMatrix"
##                           s1
## (Intercept)  -1.251828e+05
## gender        2.322991e+03
## issueDate     2.032077e+00
## matDate       1.055551e+00
## age           1.358536e+02
## gmdbAmt       8.519120e-02
## gmwbAmt       8.484042e+00
## gmwbBalance  -3.341216e-01
## gmmbAmt       2.299329e-01
```

```
## withdrawal   -5.134309e-01
## FundValue1   -1.227325e-01
## FundValue2   -5.828068e-02
## FundValue3   -1.283040e-01
## FundValue4   -1.940522e-01
## FundValue5   -1.751880e-01
## FundValue6   -7.560449e-02
## FundValue7   -9.097672e-02
## FundValue8   -1.343000e-01
## FundValue9   -5.649582e-02
## FundValue10  -1.092028e-01
## prodTypeDBRU -2.440036e+03
## prodTypeMB   -1.162461e+04
## prodTypeWB   -1.271030e+04
## prodTypeWBSU -9.927408e+03
```

Prediction Errors and Best Model

Now that we have created all of our nine models, we can compute the prediction error for each one. First, a list is created which contains predictions from all nine models. Inside our list, we predict outcomes using the nine models we created based on our supervised and unsupervised methods. For the multiple linear regression models, we predict values for the test data whereas for the Ridge and LASSO regression models, we use our test predictors and the corresponding optimal lambda to make predictions. After computing our predictions, we can now calculate the prediction error for each model. The mse_results and rmse_results variables calculate the Mean Squared Error (MSE) and the Root Mean Squared Error (RMSE) for each model. Once these prediction errors are calculated, test_error_summary is created which provides our prediction errors in an easy-to-interpret data frame. Even though we are given both the MSE and RMSE values, we will evaluate our models using RMSE going forward.

For our original (supervised only) models, we found that the best model from this group was the multiple linear regression which has an RMSE value of 342.69. The best model using both Hierarchical clustering and supervised method was the LASSO regression model. This LASSO model in this group had an RMSE value of 784.67. Lastly, the best model using the K-means clustering and supervised method was the LASSO model again. The LASSO model in this group of models had an RMSE value of 1198.23. Out of our nine models, the best model ended up being our original (supervised only) multiple linear regression model which had the lowest RMSE value (342.69) out of all the models. The worst model ended up being the K-means multiple linear regression model which had the highest RMSE value (2252.76) out of all the models. This means that the predictions from our best model (original linear regression) deviate from the actual values by plus or minus 343 units whereas the predictions from our worst model (K-mean linear regression) deviate from the actual values by plus or minus 2253.

For this particular dataset, it seems that simpler models perform better without the need for unsupervised methods. Both of our clustering methods ended up significantly increasing the prediction errors. This indicates that clustering might be introducing unnecessary complexity. Furthermore, perhaps the data that we are using may not have natural clusters that improve our prediction accuracy. The results show that sticking with our original linear regression model provides us with the best accuracy. If we have to use clustering, then we would prefer using a LASSO model as it has the lowest RMSE when we use Hierarchical or K-means clustering. In general, we should stick with our original (supervised only) models as using clustering methods for this prediction task ends up degrading the performance of our models.

```
predictions <- list(

  lm_predictions = predict(lm_model1, newdata = test_data_combined),
  ridge_predictions = predict(ridge_model, s = ridge_cv$lambda.min,
```

```
                                newx = X_test),
  lasso_predictions = predict(lasso_model, s = lasso_cv$lambda.min,
                                newx = X_test),

  lm_h = predict(lm_model2, newdata = test_data_combined),
  ridge_h = predict(ridge_model_h, s = ridge_cv_h$lambda.min, newx = X_test),
  lasso_h = predict(lasso_model_h, s = lasso_cv_h$lambda.min, newx = X_test),

  lm_k = predict(lm_model3, newdata = test_data_combined),
  ridge_k = predict(ridge_model_k, s = ridge_cv_k$lambda.min, newx = X_test),
  lasso_k = predict(lasso_model_k, s = lasso_cv_k$lambda.min, newx = X_test)
)

mse_results <- sapply(predictions, function(pred) mean(y_test - pred)^2)
rmse_results <- sqrt(mse_results)

test_error_summary <- data.frame(
  Model = names(mse_results),
  MSE = mse_results,
  RMSE = rmse_results,
  row.names = NULL
)

print("Test Error Summary:")
```

```
## [1] "Test Error Summary:"
```

```
print(test_error_summary)
```

```
##                Model       MSE      RMSE
## 1    lm_predictions  117438.6  342.6931
## 2 ridge_predictions  143493.9  378.8058
## 3 lasso_predictions  127410.4  356.9459
## 4              lm_h 3704079.3 1924.5985
## 5           ridge_h 3720242.0 1928.7929
## 6           lasso_h  615713.4  784.6740
## 7              lm_k 5074940.0 2252.7627
## 8           ridge_k 2251655.4 1500.5517
## 9           lasso_k 1435758.8 1198.2315
```

VaR Analysis

For our last analysis, we are going to apply our nine models to the testing data and obtain forecasts for fmv. We will then compute the Value-at-Risk at levels 90% and 95% of the forecasted fmv and observed fmv in the testing data. VaR is a risk metric that is used to estimate the maximum loss a portfolio may encounter in a certain time frame and is specified at various confidence levels. Using VaR analysis allows us to quantify and compare the potential risks predicted by our various models. In the context of this report, we are predicting the fmv or the fair market value of variable annuities using different supervised and unsupervised methods. With the VaR analysis, we will be able to evaluate how well our models predict the fmv under certain scenarios. By comparing the model VaR values to the observed VaR values, we will be able to tell which models align closely with the actual risk profile. This is important as selecting the best model will help in making practical financial and risk-based decisions.

In our code, we will first define a function called calculate_var which will compute the VaR for a set of values at our specified confidence levels (90% & 95%). Then the VaR values for all of our predictions and the actual testing data are calculated. A data frame called var_summary is created which stores our VaR values at the 90% and 95% confidence levels for each model. Our observed VaR values will also be added to the data frame so that we can easily compare the values in our output. Here, VaR at 90% confidence suggests that there is a 10% chance our loss will exceed our VaR value whereas VaR at 95% indicates that there is a 5% chance our loss will exceed our VaR value.

We will not go over every single VaR value in detail. Instead, we will focus on some of the key insights from our VaR analysis. Our results show that the K-means linear regression model has the lowest VaR at 90% confidence compared to all other models. If we were to interpret this value, it would mean that this model predicts that there is a 90% probability that the losses will not exceed 49832.07 dollars. For the same model, the VaR at 95% confidence suggests that there is a 95% probability that the losses will not exceed 69755.86 dollars. The lowest VaR at 95% was from our K-means Ridge regression model which gave us a VaR value of 64527.37. In comparison, our Hierarchical clustering models seem to consistently show higher VaR values, which could be a sign that these models account for more risk. Looking at our observed VaR values, we can see that these values are higher compared to all the other models. The observed VaR at 90% is 60543.27 and the observed VaR at 95% is 82794.99. This means that based on our observed fmv from the test data there is a 90% probability that the losses will not exceed 60543.27 dollars and there is a 95% probability that the losses will not exceed 82794.99 dollars. Since these observed VaR values are higher than our other models, it indicates that our models may be underestimating the potential losses compared to the actual observed data. The K-means clustering models show more conservative VaR estimates, which might be good if we are trying to find relationships that minimize risk. Going even deeper, we find discrepancies between our LASSO and Ridge models. Our Ridge regression models in general seem to show the most conservative risk estimates. These models therefore can be effectively utilized to minimize risk. For conservative-based financial decisions, the Ridge regression models would be the best choice preferably the original or K-means Ridge regression model. If we are focused more on managing frequent small losses, the best choice would be the K-means linear regression model due to it having the lowest VaR at 90% confidence.

```
calculate_var <- function(values, levels = c(0.90, 0.95)) {
  sapply(levels, function(level) quantile(values, level))
}

var_results <- lapply(predictions, calculate_var)
observed_var <- calculate_var(y_test)

var_summary <- data.frame(
  Model = names(var_results),
  VaR_90 = sapply(var_results, function(x) x[1]),
  VaR_95 = sapply(var_results, function(x) x[2]),
  row.names = NULL
)

var_summary <- rbind(var_summary,
                     data.frame(Model = "Observed",
                                VaR_90 = observed_var[1],
                                VaR_95 = observed_var[2]),
                     row.names = NULL)

print("Value at Risk Summary:")
```

```
## [1] "Value at Risk Summary:"
```

```r
print(var_summary)
```

```
##                  Model   VaR_90   VaR_95
## 1      lm_predictions 53988.98 71820.59
## 2   ridge_predictions 51235.21 64728.25
## 3   lasso_predictions 53997.70 71695.46
## 4                lm_h 55368.62 75121.69
## 5             ridge_h 55321.88 73977.95
## 6             lasso_h 54597.67 75995.10
## 7                lm_k 49832.07 69755.86
## 8             ridge_k 50752.60 64527.37
## 9             lasso_k 53805.02 69779.42
## 90%          Observed 60543.27 82794.99
```

Conclusion

With our analysis, we were able to assess the fair market value (fmv) of variable annuities using various models. In the end, a total of nine models were created using a combination of supervised methods (multiple linear regression, Ridge regression, and LASSO regression) and unsupervised methods (Hierarchical and K-means clustering). Each model was evaluated based on prediction errors and by conducting a VaR analysis.

By computing the prediction errors of all of our models, we were able to discover some important insights. The original multiple linear regression model, where we only used a supervised method was our best-performing model. Out of all the other models this model had the lowest RMSE value (342.69). If we go based on our models where we used both supervised and unsupervised techniques, the best-performing model using Hierarchical clustering was the LASSO regression (RMSE = 784.67). The best-performing model using K-means clustering was also the LASSO regression (RMSE = 1198.23). The worst-performing model was the K-means linear regression model which had the highest RMSE value (2252.76). Based on our analysis we find that models which incorporate these clustering methods showed significantly higher prediction errors compared to our original (supervised only) models. For this particular dataset, using supervised methods alone proved to be the most effective at predicting the fmv of variable annuities. The increased prediction errors suggest that using clustering methods may have introduced unnecessary complexity or removed variations in the data that may have been relevant in predicting fmv.

In addition to evaluating the prediction errors, the VaR analysis provided crucial insights into the risk profiles predicted by various models. Some of our smallest potential losses came from our K-means clustering models. The K-means linear regression model had the lowest VaR at 90% confidence (49832.07) and the K-means Ridge regression model had the lowest VaR at 95% confidence (64527.37). Comparing our observed VaR values against our forecasted VaR values, we find that the observed VaR values were higher than our other models. The observed VaR at 90% was 60543.27 whereas the observed VaR at 95% was 82794.99. This indicates that all of our models seem to be underestimating potential losses compared to the actual data. Hierarchical clustering models consistently showed higher VaR values compared to our other models, which means that these models predict larger potential losses. If a policyholder is risk-averse and wants to minimize losses, the K-means Ridge regression model or the K-means linear regression model would be the ideal choice. This is because the VaR values at 90% confidence of these models are the lowest in comparison to our other models. For moderate risk scenarios, we want to focus on 90% VaR values as they tend to capture typical risks. In situations where we see high volatility, the Hierarchical LASSO regression model would be the most suitable. The reason we use this model in high-risk scenarios is because it has the highest VaR value at 95% confidence. It is best to focus on 95% VaR values in scenarios where we see less frequent, but severe losses in the value of our variable annuities. Finally, in our VaR analysis, we were also able to find the best model based on how close it was to our observed VaR values. Here, the Hierarchical clustering models were the closest to our observed VaR values. Even though these Hierarchical models have lower VaR values compared to our observed data, our original and K-means clustering models have even lower VaR values.

The Hierarchical models (lm_h, ridge_h, lasso_h) were the most accurate in predicting the potential losses without severely underestimating the risk.

From our models, we were able to determine some predictors that might be important in predicting fmv of variable annuities. The Hierarchical clustering method produced the multiple linear regression with the highest adjusted R-squared value (0.7319). Based on the results, we find that all of our predictors are statistically significant with the p-values being less than 0.05. In this model, the variable gmwbAmt had a positive coefficient (12.02) which indicates a positive effect on fmv whereas the variable withdrawal had a negative coefficient (-0.902) which indicates a negative effect on fmv. Even though the Hierarchical linear regression model has the highest adjusted R-squared, it does not have the lowest prediction error. Our original linear regression model had the lowest RMSE, but it had a lot more predictors in the model. The model of choice here really depends on whether we want to focus on prediction accuracy or model simplicity with a lower risk of overfitting. If we compare across all our Ridge and LASSO regression models we can see that variables like gender, age, and prodtypes had coefficients that were high in magnitude in comparison to the other variables. The FundValue variables did not seem to have a significant effect on our response since the magnitude of their coefficients in our Ridge and LASSO models was relatively low.

The decision to use certain models depends on our primary objective. While the original multiple linear regression model showed the lowest RMSE (highest predictive accuracy), it achieved this by including a large number of predictors. This in itself has its downside as we may run the risk of overfitting and reducing the interpretability of our model. In contrast, the models that combined supervised and unsupervised techniques, like K-means and Hierarchical clustering, provided us with fewer predictors but higher prediction errors. If our primary goal is high predictive accuracy, then the models with the lowest RMSE value should be chosen which in this case would be our original multiple linear regression model. On the other hand, if our focus is more on reducing the risk of overfitting and model interpretability, simpler models such as those obtained by using clustering methods would prove to be more suitable.