Preparatory Work

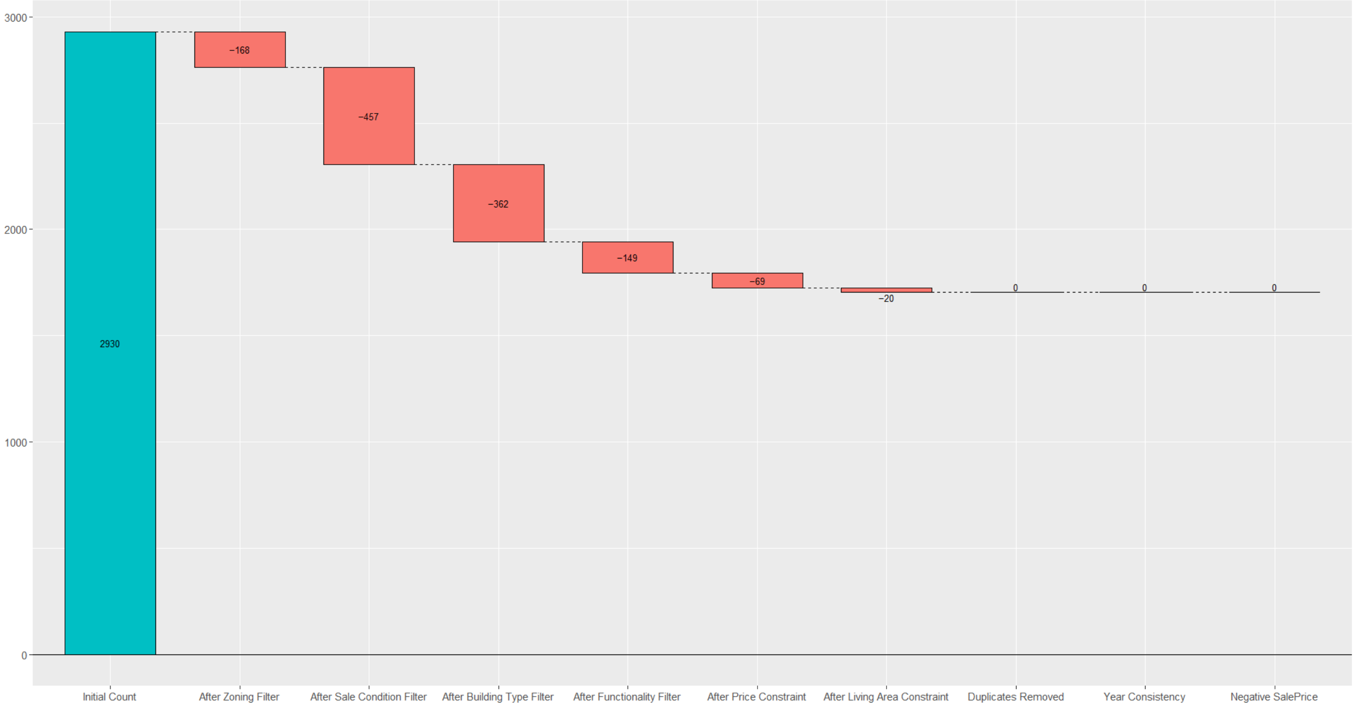
As in Module 3, the AMES data set has been filtered has been filtered to create the typical_homes dataframe, as per the following conditions:

1. Zoning Condition: The dataset was filtered to include only homes with Zoning classifications of "RH", "RL", or "RM".
2. Sale Condition: Only homes with a Sale Condition of "Normal" were retained.
3. Building Type: The data was further narrowed down to include only single-family homes, indicated by the "1Fam" value in the `BldgType` column.
4. Functionality: Homes that were tagged as typical functionality, represented by the value "Typ" in the `Functional` column, were selected.
5. Sale Price Constraint: Homes that had a `SalePrice` less than or equal to 339,500 were included.
6. Living Area Constraint: Only homes with a `GrLivArea` (Above grade living area) less than or equal to 2,667 sq. ft. were considered.

Data Cleaning and Refinement Steps:

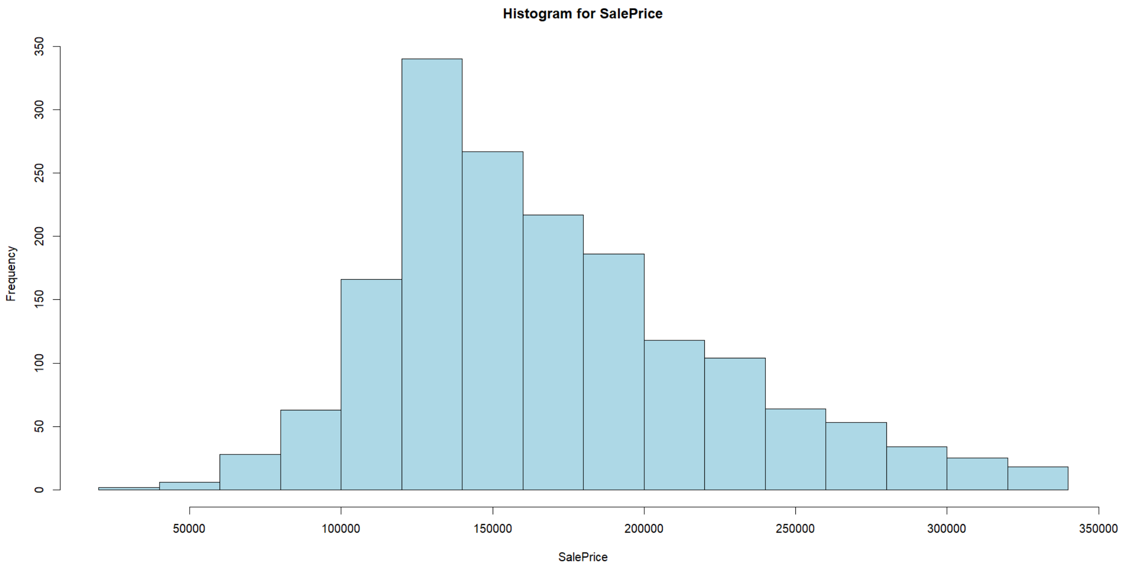
1. Duplicate Rows: Any duplicate rows in the dataset were identified and removed.
2. Year Consistency: Rows where the `YearBuilt` was greater than the `YearRemodAdd` (Remodel date) were deemed inconsistent and thus removed.
3. Sale Price Anomalies: Rows with a negative value for `SalePrice` were identified as anomalies and excluded from the dataset.
4. Column Removal: The columns `LotFrontage` and `GarageYrBlt` were dropped from the dataset.
5. Handling Missing Values: Rows containing missing values in the following columns were removed:
   - `MasVnrType`: Masonry veneer type
   - `BsmtExposure`: Walkout or garden level walls in basement
   - `BsmtFinType2`: Quality of the second finished area (if present)
   - `Electrical`: Electrical system

The waterfall depicts the number of rows deleted at every step.



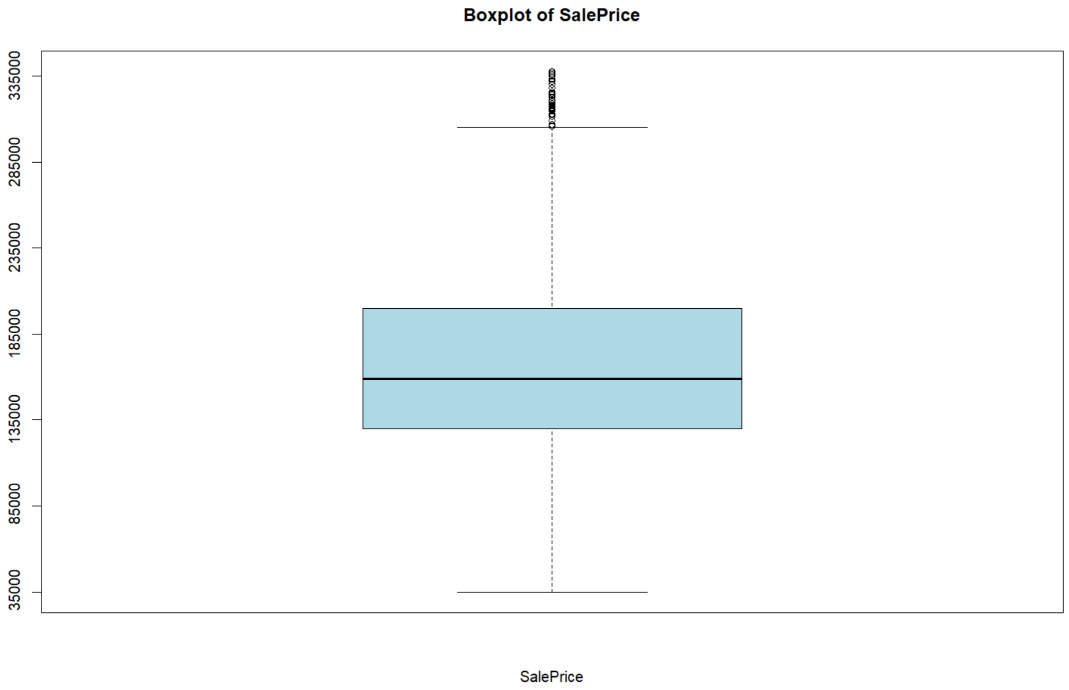
The filtered dataframe has 1691 rows and 80 columns.

This is the Histogram of the SalePrice:



The bulk of the properties have sale prices in the lower to mid ranges, with fewer properties in the higher price ranges.

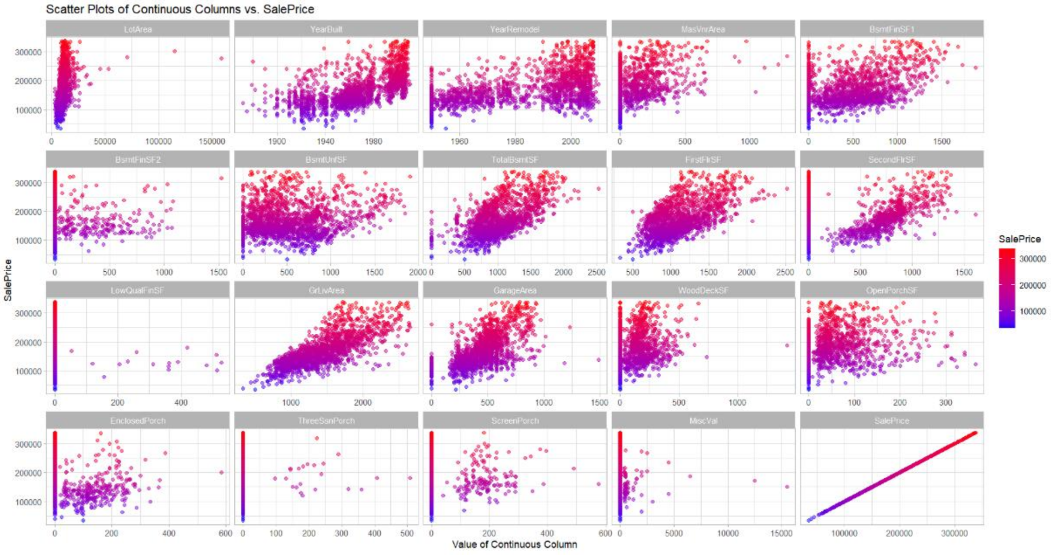
Boxplot of SalePrice

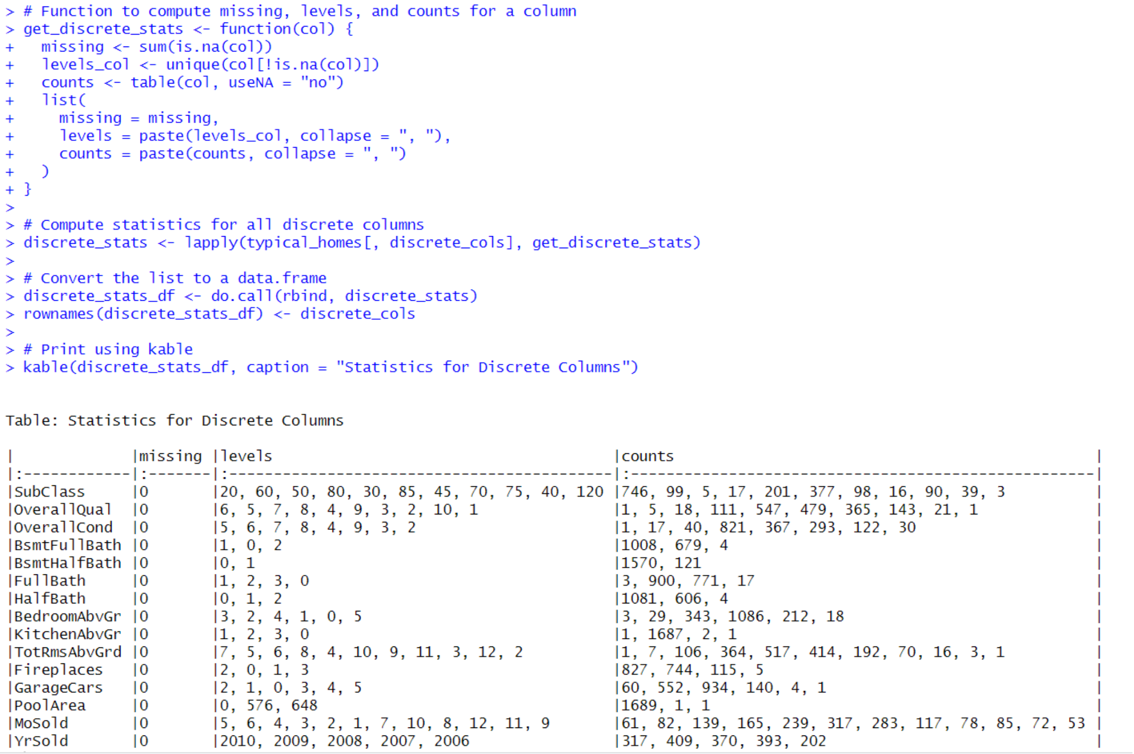


From the boxplot, we can infer that there's a considerable spread in the SalePrice data, with several higher-priced outliers.

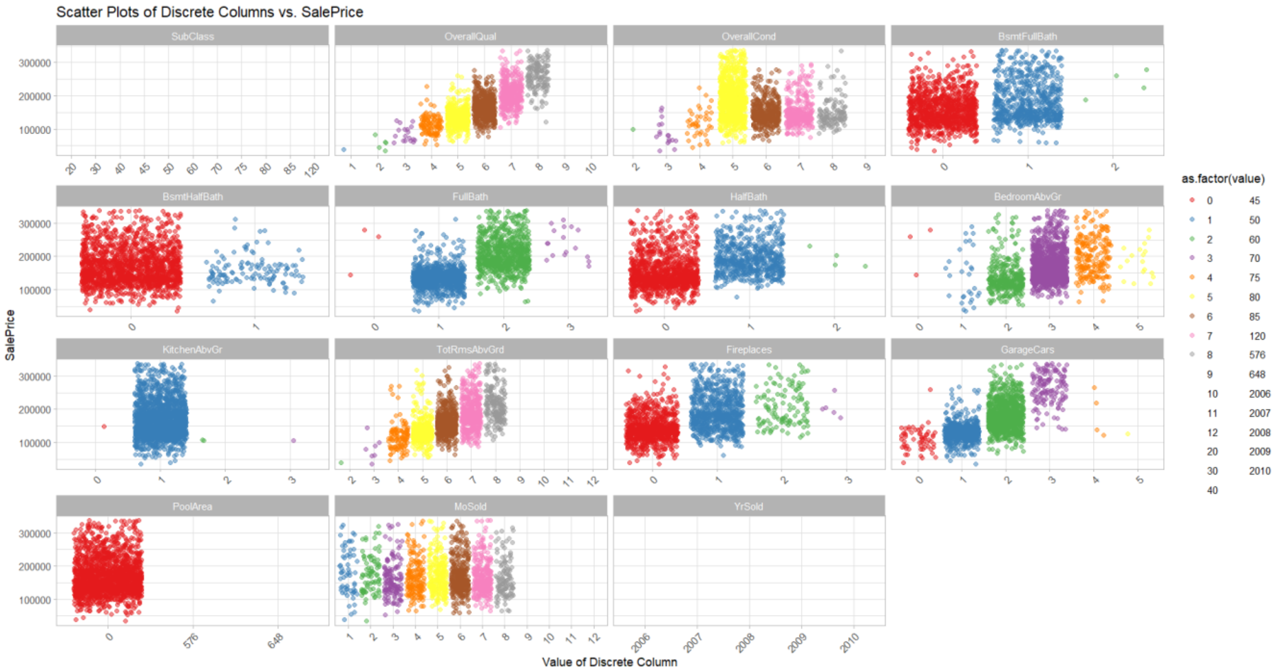
```
> # Extract numeric columns
> numeric_cols <- names(typical_homes)[sapply(typical_homes, is.numeric)]
>
> # Identify discrete columns based on a threshold of unique values
> # (e.g., if a column has 15 or fewer unique values, consider it discrete)
> threshold <- 15
> discrete_cols <- numeric_cols[sapply(typical_homes[, numeric_cols], function(col) {
+   length(unique(col)) <= threshold
+ })]
>
> # Continuous columns are the numeric columns that are not discrete
> continuous_cols <- setdiff(numeric_cols, discrete_cols)
> continuous_cols <- setdiff(continuous_cols, "SID")
> continuous_cols <- setdiff(continuous_cols, "PID")
>
>
> # Print results
> print("Continuous columns:")
[1] "Continuous columns:"
> print(continuous_cols)
 [1] "LotArea"       "YearBuilt"     "YearRemodel"   "MasVnrArea"    "BsmtFinSF1"
 [6] "BsmtFinSF2"    "BsmtUnfSF"     "TotalBsmtSF"   "FirstFlrSF"    "SecondFlrSF"
[11] "LowQualFinSF"  "GrLivArea"     "GarageArea"    "WoodDeckSF"    "OpenPorchSF"
[16] "EnclosedPorch" "ThreeSsnPorch" "ScreenPorch"   "MiscVal"       "SalePrice"
>
> print("Discrete columns:")
[1] "Discrete columns:"
> print(discrete_cols)
 [1] "SubClass"     "OverallQual"  "OverallCond"  "BsmtFullBath" "BsmtHalfBath" "FullBath"
 [7] "HalfBath"     "BedroomAbvGr" "KitchenAbvGr" "TotRmsAbvGrd" "Fireplaces"   "GarageCars"
[13] "PoolArea"     "MoSold"       "YrSold"
```

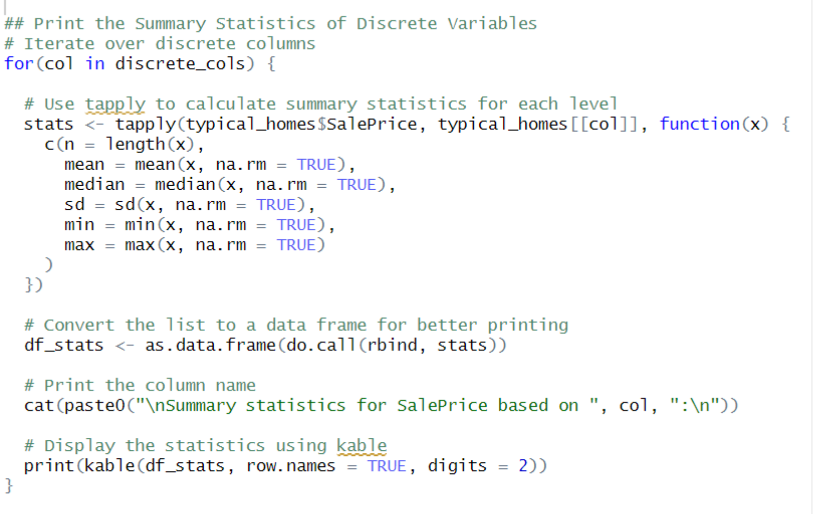
```
> # Compute summary statistics for continuous columns
> summary_stats <- sapply(typical_homes[, continuous_cols], function(col) {
+   c(
+     n = length(col[!is.na(col)]),
+     missing = sum(is.na(col)),  # Number of missing values
+     mean = mean(col, na.rm = TRUE),
+     median = median(col, na.rm = TRUE),
+     sd = sd(col, na.rm = TRUE),
+     min = min(col, na.rm = TRUE),
+     max = max(col, na.rm = TRUE)
+   )
+ })
>
> # Print the summary using kable
> kable(t(summary_stats), caption = "Statistical Summary of Continuous Variables with Missing Values")


Table: Statistical Summary of Continuous Variables with Missing Values

|              |    n| missing|          mean| median|          sd|   min|    max|
|:-------------|----:|-------:|-------------:|------:|-----------:|-----:|------:|
|LotArea       | 1691|       0|  10345.958013|   9600|  5995.77162|  2887| 159000|
|YearBuilt     | 1691|       0|   1966.936132|   1967|    29.19211|  1872|   2010|
|YearRemodel   | 1691|       0|   1982.057362|   1990|    20.70879|  1950|   2010|
|MasVnrArea    | 1691|       0|     82.425192|      0|   148.12114|     0|   1290|
|BsmtFinSF1    | 1691|       0|    424.367830|    386|   389.48403|     0|   1880|
|BsmtFinSF2    | 1691|       0|     53.678297|      0|   170.14493|     0|   1526|
|BsmtUnfSF     | 1691|       0|    530.255470|    461|   383.30830|     0|   1905|
|TotalBsmtSF   | 1691|       0|   1008.301597|    955|   340.26320|     0|   2524|
|FirstFlrSF    | 1691|       0|   1096.342401|   1038|   309.97337|   334|   2524|
|SecondFlrSF   | 1691|       0|    325.545239|      0|   408.88924|     0|   1611|
|LowQualFinSF  | 1691|       0|      3.112951|      0|    34.74899|     0|    528|
|GrLivArea     | 1691|       0|   1425.000591|   1400|   427.40733|   334|   2654|
|GarageArea    | 1691|       0|    450.952691|    461|   187.75401|     0|   1488|
|WoodDeckSF    | 1691|       0|     93.315198|      0|   127.53374|     0|   1424|
|OpenPorchSF   | 1691|       0|     42.212300|     21|    58.84031|     0|    365|
|EnclosedPorch | 1691|       0|     25.078652|      0|    63.35234|     0|    584|
|ThreeSsnPorch | 1691|       0|      2.915435|      0|    27.70405|     0|    508|
|ScreenPorch   | 1691|       0|     16.832052|      0|    58.20609|     0|    576|
|MiscVal       | 1691|       0|     55.934358|      0|   565.72139|     0|  15500|
|SalePrice     | 1691|       0| 169722.547605| 159000| 54834.31924| 35000| 337500|
```



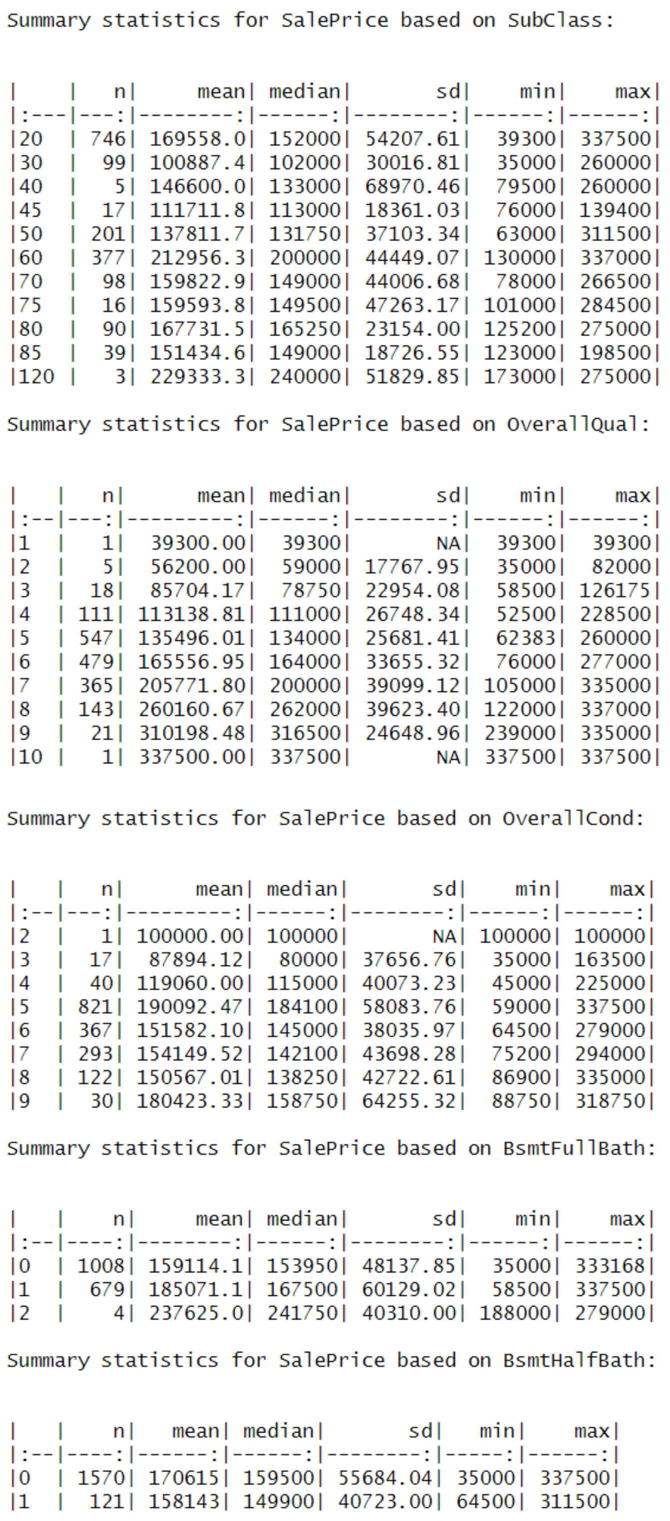


```
> # Function to compute missing, levels, and counts for a column
> get_discrete_stats <- function(col) {
+   missing <- sum(is.na(col))
+   levels_col <- unique(col[!is.na(col)])
+   counts <- table(col, useNA = "no")
+   list(
+     missing = missing,
+     levels = paste(levels_col, collapse = ", "),
+     counts = paste(counts, collapse = ", ")
+   )
+ }
>
> # Compute statistics for all discrete columns
> discrete_stats <- lapply(typical_homes[, discrete_cols], get_discrete_stats)
>
> # Convert the list to a data.frame
> discrete_stats_df <- do.call(rbind, discrete_stats)
> rownames(discrete_stats_df) <- discrete_cols
>
> # Print using kable
> kable(discrete_stats_df, caption = "Statistics for Discrete Columns")


Table: Statistics for Discrete Columns

|             |missing |levels                                      |counts                                                    |
|:------------|:-------|:-------------------------------------------|:---------------------------------------------------------|
|SubClass     |0       |20, 60, 50, 80, 30, 85, 45, 70, 75, 40, 120 |746, 99, 5, 17, 201, 377, 98, 16, 90, 39, 3               |
|OverallQual  |0       |6, 5, 7, 8, 4, 9, 3, 2, 10, 1               |1, 5, 18, 111, 547, 479, 365, 143, 21, 1                  |
|OverallCond  |0       |5, 6, 7, 8, 4, 9, 3, 2                      |1, 17, 40, 821, 367, 293, 122, 30                         |
|BsmtFullBath |0       |1, 0, 2                                     |1008, 679, 4                                              |
|BsmtHalfBath |0       |0, 1                                        |1570, 121                                                 |
|FullBath     |0       |1, 2, 3, 0                                  |3, 900, 771, 17                                           |
|HalfBath     |0       |0, 1, 2                                     |1081, 606, 4                                              |
|BedroomAbvGr |0       |3, 2, 4, 1, 0, 5                            |3, 29, 343, 1086, 212, 18                                 |
|KitchenAbvGr |0       |1, 2, 3, 0                                  |1, 1687, 2, 1                                             |
|TotRmsAbvGrd |0       |7, 5, 6, 8, 4, 10, 9, 11, 3, 12, 2          |1, 7, 106, 364, 517, 414, 192, 70, 16, 3, 1               |
|Fireplaces   |0       |2, 0, 1, 3                                  |827, 744, 115, 5                                          |
|GarageCars   |0       |2, 1, 0, 3, 4, 5                            |60, 552, 934, 140, 4, 1                                   |
|PoolArea     |0       |0, 576, 648                                 |1689, 1, 1                                                |
|MoSold       |0       |5, 6, 4, 3, 2, 1, 7, 10, 8, 12, 11, 9       |61, 82, 139, 165, 239, 317, 283, 117, 78, 85, 72, 53      |
|YrSold       |0       |2010, 2009, 2008, 2007, 2006                |317, 409, 370, 393, 202                                   |
```

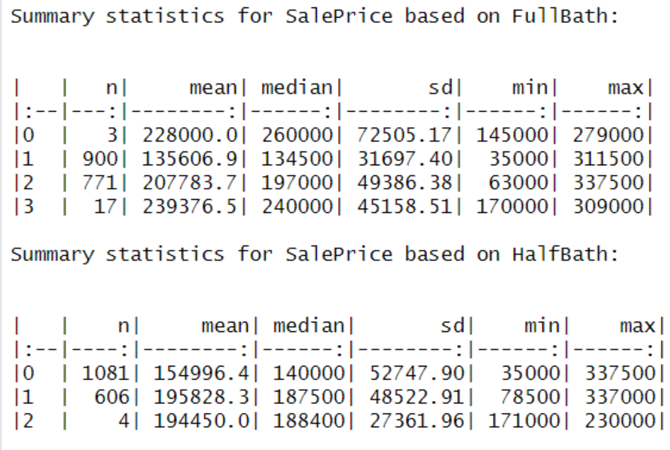




```
## Print the Summary Statistics of Discrete Variables
# Iterate over discrete columns
for(col in discrete_cols) {

  # Use tapply to calculate summary statistics for each level
  stats <- tapply(typical_homes$SalePrice, typical_homes[[col]], function(x) {
    c(n = length(x),
      mean = mean(x, na.rm = TRUE),
      median = median(x, na.rm = TRUE),
      sd = sd(x, na.rm = TRUE),
      min = min(x, na.rm = TRUE),
      max = max(x, na.rm = TRUE)
    )
  })

  # Convert the list to a data frame for better printing
  df_stats <- as.data.frame(do.call(rbind, stats))

  # Print the column name
  cat(paste0("\nSummary statistics for SalePrice based on ", col, ":\n"))

  # Display the statistics using kable
  print(kable(df_stats, row.names = TRUE, digits = 2))
}
```

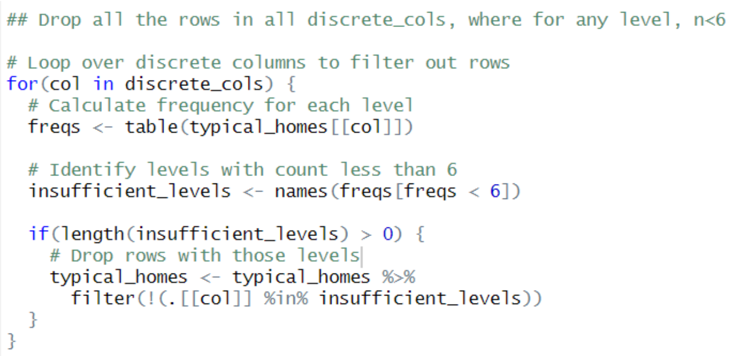


```
Summary statistics for SalePrice based on SubClass:


|    |   n|     mean| median|       sd|    min|    max|
|:---|---:|--------:|------:|--------:|------:|------:|
|20  | 746| 169558.0| 152000| 54207.61|  39300| 337500|
|30  |  99| 100887.4| 102000| 30016.81|  35000| 260000|
|40  |   5| 146600.0| 133000| 68970.46|  79500| 260000|
|45  |  17| 111711.8| 113000| 18361.03|  76000| 139400|
|50  | 201| 137811.7| 131750| 37103.34|  63000| 311500|
|60  | 377| 212956.3| 200000| 44449.07| 130000| 337000|
|70  |  98| 159822.9| 149000| 44006.68|  78000| 266500|
|75  |  16| 159593.8| 149500| 47263.17| 101000| 284500|
|80  |  90| 167731.5| 165250| 23154.00| 125200| 275000|
|85  |  39| 151434.6| 149000| 18726.55| 123000| 198500|
|120 |   3| 229333.3| 240000| 51829.85| 173000| 275000|

Summary statistics for SalePrice based on OverallQual:


|   |   n|      mean| median|       sd|    min|    max|
|:--|---:|---------:|------:|--------:|------:|------:|
|1  |   1|  39300.00|  39300|       NA|  39300|  39300|
|2  |   5|  56200.00|  59000| 17767.95|  35000|  82000|
|3  |  18|  85704.17|  78750| 22954.08|  58500| 126175|
|4  | 111| 113138.81| 111000| 26748.34|  52500| 228500|
|5  | 547| 135496.01| 134000| 25681.41|  62383| 260000|
|6  | 479| 165556.95| 164000| 33655.32|  76000| 277000|
|7  | 365| 205771.80| 200000| 39099.12| 105000| 335000|
|8  | 143| 260160.67| 262000| 39623.40| 122000| 337000|
|9  |  21| 310198.48| 316500| 24648.96| 239000| 335000|
|10 |   1| 337500.00| 337500|       NA| 337500| 337500|


Summary statistics for SalePrice based on OverallCond:


|   |   n|      mean| median|       sd|    min|    max|
|:--|---:|---------:|------:|--------:|------:|------:|
|2  |   1| 100000.00| 100000|       NA| 100000| 100000|
|3  |  17|  87894.12|  80000| 37656.76|  35000| 163500|
|4  |  40| 119060.00| 115000| 40073.23|  45000| 225000|
|5  | 821| 190092.47| 184100| 58083.76|  59000| 337500|
|6  | 367| 151582.10| 145000| 38035.97|  64500| 279000|
|7  | 293| 154149.52| 142100| 43698.28|  75200| 294000|
|8  | 122| 150567.01| 138250| 42722.61|  86900| 335000|
|9  |  30| 180423.33| 158750| 64255.32|  88750| 318750|

Summary statistics for SalePrice based on BsmtFullBath:


|   |    n|     mean| median|       sd|    min|    max|
|:--|----:|--------:|------:|--------:|------:|------:|
|0  | 1008| 159114.1| 153950| 48137.85|  35000| 333168|
|1  |  679| 185071.1| 167500| 60129.02|  58500| 337500|
|2  |    4| 237625.0| 241750| 40310.00| 188000| 279000|

Summary statistics for SalePrice based on BsmtHalfBath:


|   |    n|   mean| median|       sd|   min|    max|
|:--|----:|------:|------:|--------:|-----:|------:|
|0  | 1570| 170615| 159500| 55684.04| 35000| 337500|
|1  |  121| 158143| 149900| 40723.00| 64500| 311500|
```

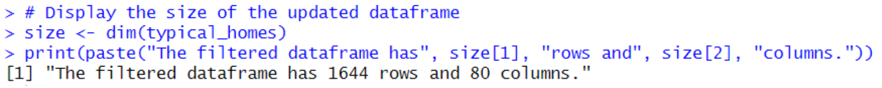


```
Summary statistics for SalePrice based on FullBath:


|   |   n|     mean| median|       sd|    min|    max|
|:--|---:|--------:|------:|--------:|------:|------:|
|0  |   3| 228000.0| 260000| 72505.17| 145000| 279000|
|1  | 900| 135606.9| 134500| 31697.40|  35000| 311500|
|2  | 771| 207783.7| 197000| 49386.38|  63000| 337500|
|3  |  17| 239376.5| 240000| 45158.51| 170000| 309000|

Summary statistics for SalePrice based on HalfBath:


|   |    n|     mean| median|       sd|    min|    max|
|:--|----:|--------:|------:|--------:|------:|------:|
|0  | 1081| 154996.4| 140000| 52747.90|  35000| 337500|
|1  |  606| 195828.3| 187500| 48522.91|  78500| 337000|
|2  |    4| 194450.0| 188400| 27361.96| 171000| 230000|
Summary statistics for SalePrice based on BedroomAbvGr:


|    |    n|     mean| median|       sd|    min|    max|
|:---|----:|--------:|------:|--------:|------:|------:|
|0   |    3| 228000.0| 260000| 72505.17| 145000| 279000|
|1   |   29| 140722.4| 143000| 77958.13|  35000| 289000|
|2   |  343| 137570.5| 127500| 48955.67|  52500| 325000|
|3   | 1086| 173943.9| 163500| 49682.30|  62383| 337500|
|4   |  212| 201884.3| 198000| 58862.68|  63000| 336000|
|5   |   18| 185927.8| 185750| 48152.72| 117500| 279900|

Summary statistics for SalePrice based on KitchenAbvGr:


|    |    n|     mean| median|       sd|    min|    max|
|:---|----:|--------:|------:|--------:|------:|------:|
|0   |    1| 148000.0| 148000|       NA| 148000| 148000|
|1   | 1687| 169848.5| 159000| 54831.05|  35000| 337500|
|2   |    2| 106200.0| 106200|  1697.06| 105000| 107400|
|3   |    1| 106000.0| 106000|       NA| 106000| 106000|

Summary statistics for SalePrice based on TotRmsAbvGrd:


|    |   n|     mean| median|        sd|    min|    max|
|:---|---:|--------:|------:|---------:|------:|------:|
|2   |   1|  39300.0|  39300|        NA|  39300|  39300|
|3   |   7|  81700.0|  80500|  35285.22|  35000| 145000|
|4   | 106| 119537.7| 116250|  41347.60|  45000| 270000|
|5   | 364| 135953.4| 129500|  38048.82|  55000| 318750|
|6   | 517| 159210.2| 152000|  38070.89|  63000| 325000|
|7   | 414| 191221.6| 184900|  50118.93|  87500| 337500|
|8   | 192| 214386.7| 210000|  50303.49| 107400| 337000|
|9   |  70| 238825.7| 240000|  51422.51| 111500| 335000|
|10  |  16| 252540.5| 255000|  55799.84| 118500| 335000|
|11  |   3| 184666.7| 123000| 121829.12| 106000| 325000|
|12  |   1| 279500.0| 279500|        NA| 279500| 279500|


Summary statistics for SalePrice based on Fireplaces:


|    |   n|     mean| median|       sd|    min|    max|
|:---|---:|--------:|------:|--------:|------:|------:|
|0   | 827| 142681.3| 136000| 39941.54|  35000| 328000|
|1   | 744| 193343.2| 184000| 54233.66|  62383| 337500|
|2   | 115| 209810.9| 201800| 56464.15| 116900| 335000|
|3   |   5| 205570.0| 201000| 31033.84| 174850| 257000|

Summary statistics for SalePrice based on GarageCars:


|    |   n|     mean| median|       sd|    min|    max|
|:---|---:|--------:|------:|--------:|------:|------:|
|0   |  60| 109368.3| 110000| 33776.50|  39300| 260000|
|1   | 552| 131148.7| 130000| 29382.87|  35000| 266500|
|2   | 934| 182263.3| 178000| 43755.26|  68000| 335000|
|3   | 140| 263844.5| 270500| 47117.89| 140000| 337500|
|4   |   4| 186494.8| 178500| 67324.24| 123000| 265979|
|5   |   1| 126500.0| 126500|       NA| 126500| 126500|

Summary statistics for SalePrice based on PoolArea:


|     |    n|     mean| median|      sd|    min|    max|
|:----|----:|--------:|------:|-------:|------:|------:|
|0    | 1689| 169715.1| 159000| 54866.1|  35000| 337500|
|576  |    1| 171000.0| 171000|      NA| 171000| 171000|
|648  |    1| 181000.0| 181000|      NA| 181000| 181000|




Summary statistics for SalePrice based on MoSold:


|    |   n|     mean| median|       sd|   min|    max|
|:---|---:|--------:|------:|--------:|-----:|------:|
|1   |  61| 181723.2| 179900| 69256.14| 39300| 322500|
|2   |  82| 176857.9| 172250| 56333.58| 35000| 320000|
|3   | 139| 161154.1| 149500| 54657.64| 52500| 326000|
|4   | 165| 167178.9| 157900| 54307.66| 64500| 335000|
|5   | 239| 171144.4| 160000| 53090.84| 68500| 335000|
|6   | 317| 173155.3| 159950| 55091.54| 58500| 335000|
|7   | 283| 166914.1| 154900| 51369.20| 64000| 337500|
|8   | 117| 164575.0| 149000| 56991.32| 64000| 315750|
|9   |  78| 180369.2| 175000| 63911.77| 67000| 315500|
|10  |  85| 155806.4| 147000| 42416.31| 45000| 317000|
|11  |  72| 173667.4| 169750| 53733.42| 78500| 332000|
|12  |  53| 175968.5| 170000| 54938.94| 62383| 290000|

Summary statistics for SalePrice based on YrSold:


|     |   n|     mean|   median|       sd|   min|    max|
|:----|---:|--------:|--------:|--------:|-----:|------:|
|2006 | 317| 168509.2| 157500.0| 54074.57| 35000| 335000|
|2007 | 409| 166914.8| 156000.0| 52404.59| 39300| 330000|
|2008 | 370| 168416.6| 159897.5| 53198.66| 55000| 325000|
|2009 | 393| 173892.9| 162000.0| 58624.19| 45000| 337500|
|2010 | 202| 171590.1| 159700.0| 56146.39| 58500| 335000|
```

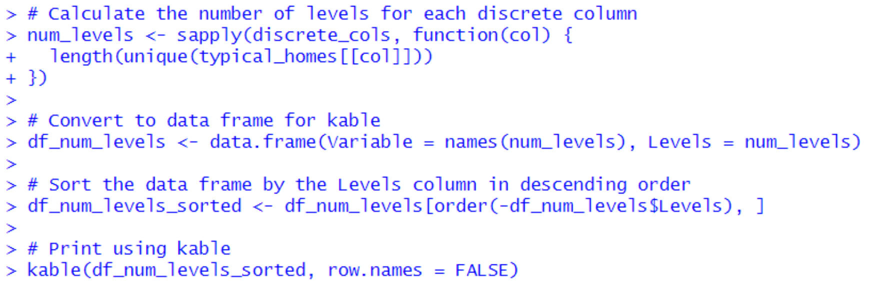
There are quite a few variables with levels that have only one value. We delete all such rows.



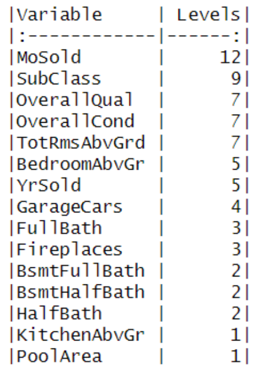
```
## Drop all the rows in all discrete_cols, where for any level, n<6

# Loop over discrete columns to filter out rows
for(col in discrete_cols) {
  # Calculate frequency for each level
  freqs <- table(typical_homes[[col]])

  # Identify levels with count less than 6
  insufficient_levels <- names(freqs[freqs < 6])

  if(length(insufficient_levels) > 0) {
    # Drop rows with those levels
    typical_homes <- typical_homes %>%
      filter(!(.[[col]] %in% insufficient_levels))
  }
}
```



```
> # Display the size of the updated dataframe
> size <- dim(typical_homes)
> print(paste("The filtered dataframe has", size[1], "rows and", size[2], "columns."))
[1] "The filtered dataframe has 1644 rows and 80 columns."
```

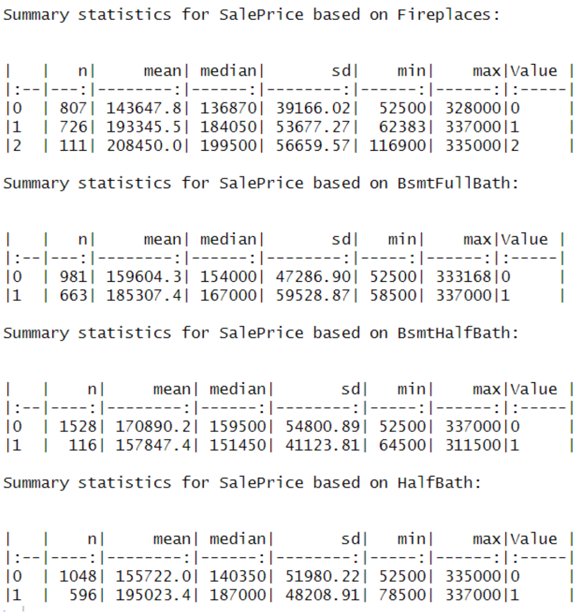


```
> # Calculate the number of levels for each discrete column
> num_levels <- sapply(discrete_cols, function(col) {
+   length(unique(typical_homes[[col]]))
+ })
>
> # Convert to data frame for kable
> df_num_levels <- data.frame(Variable = names(num_levels), Levels = num_levels)
>
> # Sort the data frame by the Levels column in descending order
> df_num_levels_sorted <- df_num_levels[order(-df_num_levels$Levels), ]
>
> # Print using kable
> kable(df_num_levels_sorted, row.names = FALSE)
```



```
|Variable     | Levels|
|:------------|------:|
|MoSold       |     12|
|SubClass     |      9|
|OverallQual  |      7|
|OverallCond  |      7|
|TotRmsAbvGrd |      7|
|BedroomAbvGr |      5|
|YrSold       |      5|
|GarageCars   |      4|
|FullBath     |      3|
|Fireplaces   |      3|
|BsmtFullBath |      2|
|BsmtHalfBath |      2|
|HalfBath     |      2|
|KitchenAbvGr |      1|
|PoolArea     |      1|
```

We chose to explore the variables with more than 1 level and less than 4 levels.

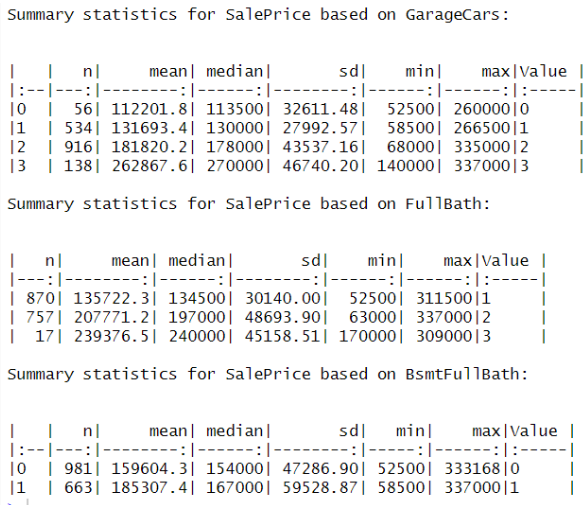


```
Summary statistics for SalePrice based on Fireplaces:


|   |   n|     mean| median|       sd|    min|    max|Value |
|:--|---:|--------:|------:|--------:|------:|------:|:-----|
|0  | 807| 143647.8| 136870| 39166.02|  52500| 328000|0     |
|1  | 726| 193345.5| 184050| 53677.27|  62383| 337000|1     |
|2  | 111| 208450.0| 199500| 56659.57| 116900| 335000|2     |

Summary statistics for SalePrice based on BsmtFullBath:


|   |   n|     mean| median|       sd|   min|    max|Value |
|:--|---:|--------:|------:|--------:|-----:|------:|:-----|
|0  | 981| 159604.3| 154000| 47286.90| 52500| 333168|0     |
|1  | 663| 185307.4| 167000| 59528.87| 58500| 337000|1     |

Summary statistics for SalePrice based on BsmtHalfBath:


|   |    n|     mean| median|       sd|   min|    max|Value |
|:--|----:|--------:|------:|--------:|-----:|------:|:-----|
|0  | 1528| 170890.2| 159500| 54800.89| 52500| 337000|0     |
|1  |  116| 157847.4| 151450| 41123.81| 64500| 311500|1     |

Summary statistics for SalePrice based on HalfBath:


|   |    n|     mean| median|       sd|   min|    max|Value |
|:--|----:|--------:|------:|--------:|-----:|------:|:-----|
|0  | 1048| 155722.0| 140350| 51980.22| 52500| 335000|0     |
|1  |  596| 195023.4| 187000| 48208.91| 78500| 337000|1     |
```

We explore the variables with the Mean Difference of SalePrice, within levels, and take into account the number of levels:

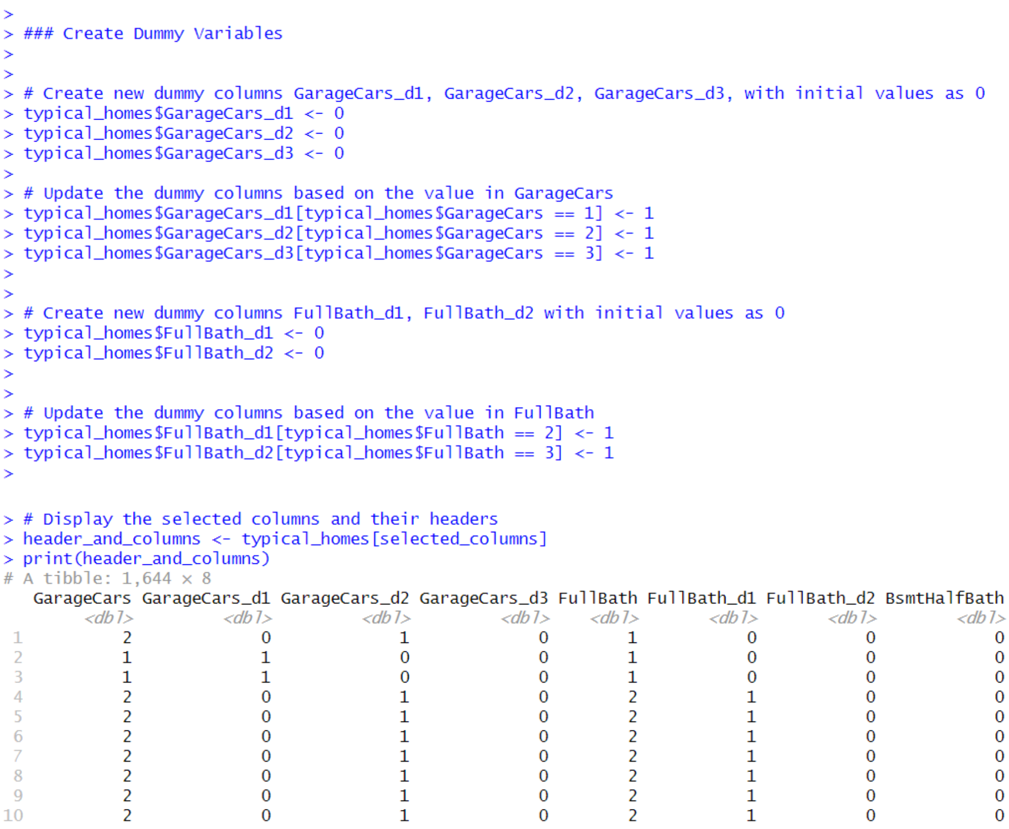
Predictor	Minimum Mean SalePrice	Maximum Mean SalePrice	Difference	# Levels
OverallQuality	39300	337500	298200.00	7
TotRmsAbvGrd	39300	279500	240200.00	7
GarageCars	**109368.3**	**263844.5**	**154476.20**	**4**
Sub-Class	100887.4	212956.3	112068.90	9
FullBath	**135606.9**	**239376.5**	**103769.60**	**3**
OverallCond	87894.12	180423.33	92529.21	7
BedroomAbvGr	137570.5	228000	90429.50	5
BsmtFullBath	**159114.1**	**237625**	**78510.90**	**2**
Fireplaces	142681.3	209810.9	67129.60	3
KitchenAbvGr	106000	169848.5	63848.50	1
HalfBath	154996.4	195828.3	40831.90	2
MoSold	161154.1	181723.2	20569.10	1
	158146	170615	12469.00	2
YrSold	166914.8	173892.9	6978.10	5

Given the limitations of computing power, and considering the SalePrice differences within levels, we decide to go choose GarageCars, FullBath, and BsmtFullBath.

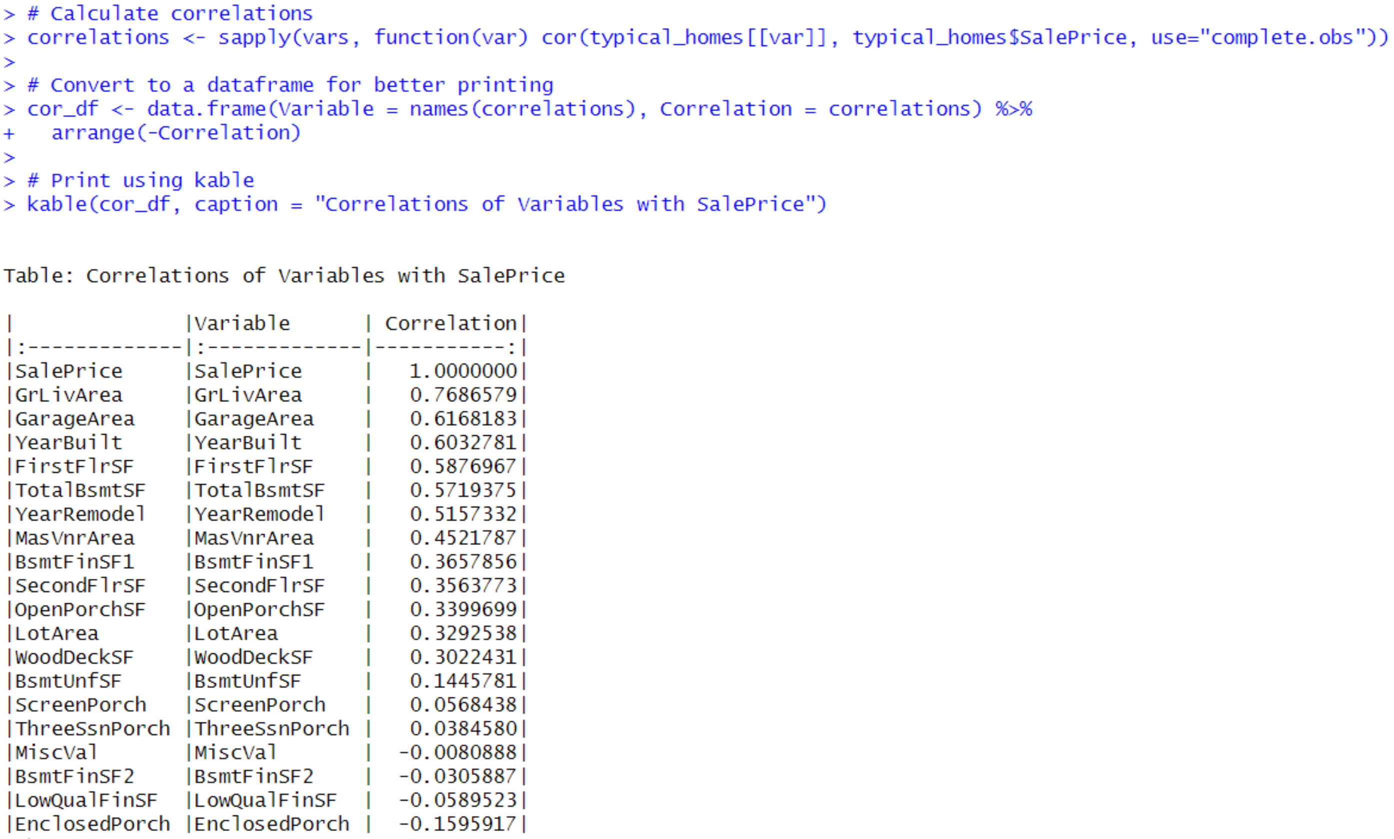


```
Summary statistics for SalePrice based on GarageCars:


|   |   n|     mean| median|       sd|    min|    max|Value |
|:--|---:|--------:|------:|--------:|------:|------:|:-----|
|0  |  56| 112201.8| 113500| 32611.48|  52500| 260000|0     |
|1  | 534| 131693.4| 130000| 27992.57|  58500| 266500|1     |
|2  | 916| 181820.2| 178000| 43537.16|  68000| 335000|2     |
|3  | 138| 262867.6| 270000| 46740.20| 140000| 337000|3     |

Summary statistics for SalePrice based on FullBath:


|   n|     mean| median|       sd|    min|    max|Value |
|---:|--------:|------:|--------:|------:|------:|:-----|
| 870| 135722.3| 134500| 30140.00|  52500| 311500|1     |
| 757| 207771.2| 197000| 48693.90|  63000| 337000|2     |
|  17| 239376.5| 240000| 45158.51| 170000| 309000|3     |

Summary statistics for SalePrice based on BsmtFullBath:


|   |   n|     mean| median|       sd|   min|    max|Value |
|:--|---:|--------:|------:|--------:|-----:|------:|:-----|
|0  | 981| 159604.3| 154000| 47286.90| 52500| 333168|0     |
|1  | 663| 185307.4| 167000| 59528.87| 58500| 337000|1     |
```

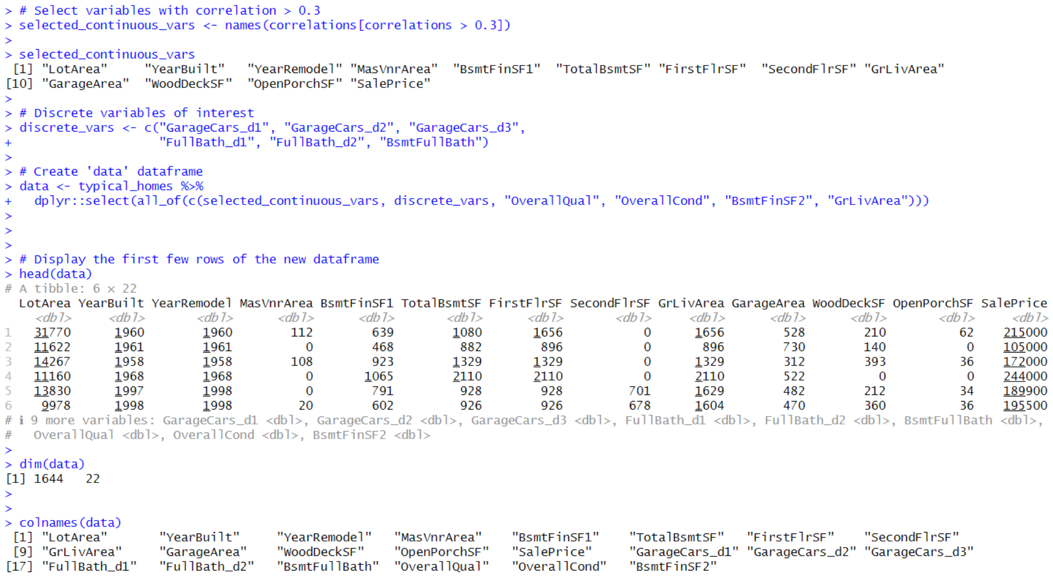
We create dummy variables for GarageCars and FullBath, but not for BsmtFullBath because the only values in BsmtFullBath are 0 and 1.



```
>
> ### Create Dummy Variables
>
>
> # Create new dummy columns GarageCars_d1, GarageCars_d2, GarageCars_d3, with initial values as 0
> typical_homes$GarageCars_d1 <- 0
> typical_homes$GarageCars_d2 <- 0
> typical_homes$GarageCars_d3 <- 0
>
> # Update the dummy columns based on the value in GarageCars
> typical_homes$GarageCars_d1[typical_homes$GarageCars == 1] <- 1
> typical_homes$GarageCars_d2[typical_homes$GarageCars == 2] <- 1
> typical_homes$GarageCars_d3[typical_homes$GarageCars == 3] <- 1
>
>
> # Create new dummy columns FullBath_d1, FullBath_d2 with initial values as 0
> typical_homes$FullBath_d1 <- 0
> typical_homes$FullBath_d2 <- 0
>
>
> # Update the dummy columns based on the value in FullBath
> typical_homes$FullBath_d1[typical_homes$FullBath == 2] <- 1
> typical_homes$FullBath_d2[typical_homes$FullBath == 3] <- 1
>

> # Display the selected columns and their headers
> header_and_columns <- typical_homes[selected_columns]
> print(header_and_columns)
# A tibble: 1,644 × 8
   GarageCars GarageCars_d1 GarageCars_d2 GarageCars_d3 FullBath FullBath_d1 FullBath_d2 BsmtHalfBath
        <dbl>         <dbl>         <dbl>         <dbl>    <dbl>       <dbl>       <dbl>        <dbl>
 1          2             0             1             0        1           0           0            0
 2          1             1             0             0        1           0           0            0
 3          1             1             0             0        1           0           0            0
 4          2             0             1             0        2           1           0            0
 5          2             0             1             0        2           1           0            0
 6          2             0             1             0        2           1           0            0
 7          2             0             1             0        2           1           0            0
 8          2             0             1             0        2           1           0            0
 9          2             0             1             0        2           1           0            0
10          2             0             1             0        2           1           0            0
```



```
> # Calculate correlations
> correlations <- sapply(vars, function(var) cor(typical_homes[[var]], typical_homes$SalePrice, use="complete.obs"))
>
> # Convert to a dataframe for better printing
> cor_df <- data.frame(Variable = names(correlations), Correlation = correlations) %>%
+   arrange(-Correlation)
>
> # Print using kable
> kable(cor_df, caption = "Correlations of Variables with SalePrice")


Table: Correlations of Variables with SalePrice

|              |Variable      | Correlation|
|:-------------|:-------------|-----------:|
|SalePrice     |SalePrice     |   1.0000000|
|GrLivArea     |GrLivArea     |   0.7686579|
|GarageArea    |GarageArea    |   0.6168183|
|YearBuilt     |YearBuilt     |   0.6032781|
|FirstFlrSF    |FirstFlrSF    |   0.5876967|
|TotalBsmtSF   |TotalBsmtSF   |   0.5719375|
|YearRemodel   |YearRemodel   |   0.5157332|
|MasVnrArea    |MasVnrArea    |   0.4521787|
|BsmtFinSF1    |BsmtFinSF1    |   0.3657856|
|SecondFlrSF   |SecondFlrSF   |   0.3563773|
|OpenPorchSF   |OpenPorchSF   |   0.3399699|
|LotArea       |LotArea       |   0.3292538|
|WoodDeckSF    |WoodDeckSF    |   0.3022431|
|BsmtUnfSF     |BsmtUnfSF     |   0.1445781|
|ScreenPorch   |ScreenPorch   |   0.0568438|
|ThreeSsnPorch |ThreeSsnPorch |   0.0384580|
|MiscVal       |MiscVal       |  -0.0080888|
|BsmtFinSF2    |BsmtFinSF2    |  -0.0305887|
|LowQualFinSF  |LowQualFinSF  |  -0.0589523|
|EnclosedPorch |EnclosedPorch |  -0.1595917|
```



```
> # Select variables with correlation > 0.3
> selected_continuous_vars <- names(correlations[correlations > 0.3])
>
> selected_continuous_vars
 [1] "LotArea"     "YearBuilt"   "YearRemodel" "MasVnrArea"  "BsmtFinSF1"  "TotalBsmtSF" "FirstFlrSF"  "SecondFlrSF" "GrLivArea"
[10] "GarageArea"  "WoodDeckSF"  "OpenPorchSF" "SalePrice"
>
> # Discrete variables of interest
> discrete_vars <- c("GarageCars_d1", "GarageCars_d2", "GarageCars_d3",
+                    "FullBath_d1", "FullBath_d2", "BsmtFullBath")
>
> # Create 'data' dataframe
> data <- typical_homes %>%
+   dplyr::select(all_of(c(selected_continuous_vars, discrete_vars, "OverallQual", "OverallCond", "BsmtFinSF2", "GrLivArea")))
>
>
>
> # Display the first few rows of the new dataframe
> head(data)
# A tibble: 6 × 22
  LotArea YearBuilt YearRemodel MasVnrArea BsmtFinSF1 TotalBsmtSF FirstFlrSF SecondFlrSF GrLivArea GarageArea WoodDeckSF OpenPorchSF SalePrice
    <dbl>     <dbl>       <dbl>      <dbl>      <dbl>       <dbl>      <dbl>       <dbl>     <dbl>      <dbl>      <dbl>       <dbl>     <dbl>
1   31770      1960        1960        112        639        1080       1656           0      1656        528        210          62    215000
2   11622      1961        1961          0        468         882        896           0       896        730        140           0    105000
3   14267      1958        1958        108        923        1329       1329           0      1329        312        393          36    172000
4   11160      1968        1968          0       1065        2110       2110           0      2110        522          0           0    244000
5   13830      1997        1998          0        791         928        928         701      1629        482        212          34    189900
6    9978      1998        1998         20        602         926        926         678      1604        470        360          36    195500
# ℹ 9 more variables: GarageCars_d1 <dbl>, GarageCars_d2 <dbl>, GarageCars_d3 <dbl>, FullBath_d1 <dbl>, FullBath_d2 <dbl>, BsmtFullBath <dbl>,
#   OverallQual <dbl>, OverallCond <dbl>, BsmtFinSF2 <dbl>
>
> dim(data)
[1] 1644   22
>
>
> colnames(data)
 [1] "LotArea"       "YearBuilt"     "YearRemodel"   "MasVnrArea"    "BsmtFinSF1"    "TotalBsmtSF"   "FirstFlrSF"    "SecondFlrSF"
 [9] "GrLivArea"     "GarageArea"    "WoodDeckSF"    "OpenPorchSF"   "SalePrice"     "GarageCars_d1" "GarageCars_d2" "GarageCars_d3"
[17] "FullBath_d1"   "FullBath_d2"   "BsmtFullBath"  "OverallQual"   "OverallCond"   "BsmtFinSF2"
```

Task 1

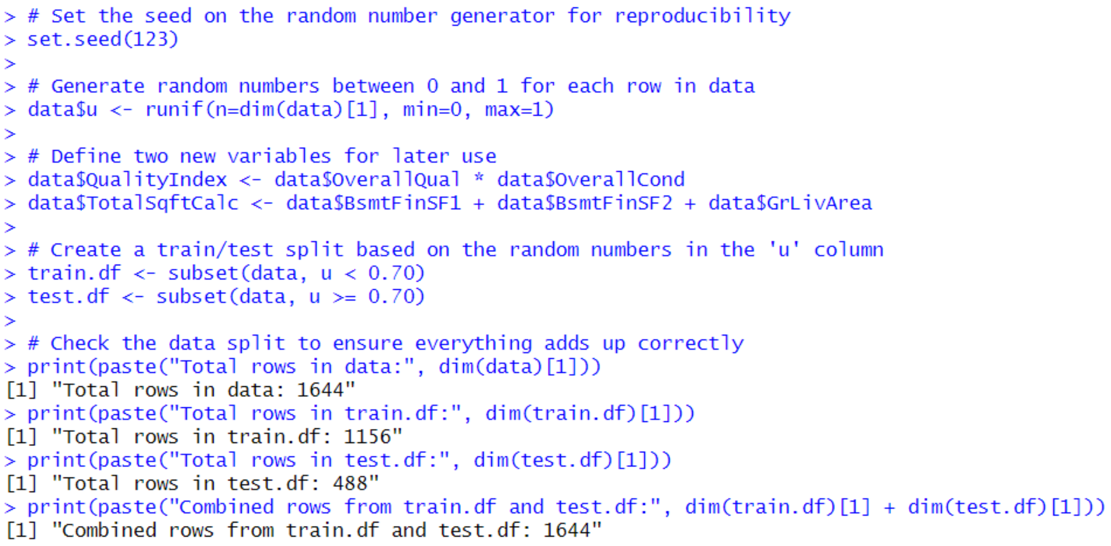
The Predictive Modeling Framework

A defining feature of predictive modeling is assessing model performance out-of-sample. We will use uniform random number to split the sample into a 70/30 train/test split. With a train/test split we now have two data sets: one for in-sample model development and one for out-of-sample model assessment.

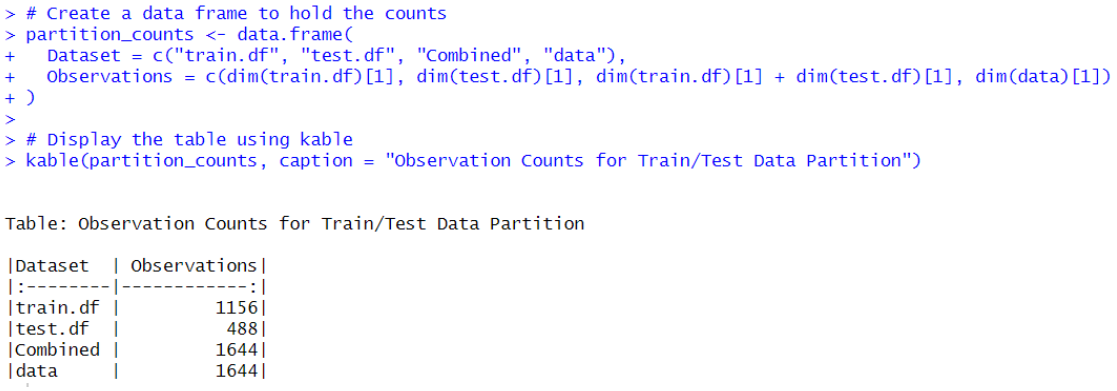
```
# Set the seed on the random number generator so you get the same split every time that
you run the code.
set.seed(123)
my.data$u <- runif(n=dim(my.data)[1],min=0,max=1);

# Define these two variables for later use;
my.data$QualityIndex <- my.data$OverallQual*my.data$OverallCond;
my.data$TotalSqftCalc <- my.data$BsmtFinSF1+my.data$BsmtFinSF2+my.data$GrLivArea;

# Create train/test split;
train.df <- subset(my.data, u<0.70);
test.df  <- subset(my.data, u>=0.70);

# Check your data split. The sum of the parts should equal the whole.
# Do your totals add up?
dim(my.data)[1]
dim(train.df)[1]
dim(test.df)[1]
dim(train.df)[1]+dim(test.df)[1]
```

Our 70/30 training/test split is the most basic form of cross-validation. We will 'train' each model by estimating the models on the 70% of the data identified as the training data set, and we will 'test' each model by examining the predictive accuracy on the 30% of the data. In R will estimate our models using the lm() function, and we will be able to apply those linear models using the R function predict(). You will want to read the R help page for the R function predict(). In particular, pay attention to the newdata argument. Your test data set is your new data.

Show a table of observation counts for your train/test data partition in your data section.



```
> # Set the seed on the random number generator for reproducibility
> set.seed(123)
>
> # Generate random numbers between 0 and 1 for each row in data
> data$u <- runif(n=dim(data)[1], min=0, max=1)
>
> # Define two new variables for later use
> data$QualityIndex <- data$OverallQual * data$OverallCond
> data$TotalSqftCalc <- data$BsmtFinSF1 + data$BsmtFinSF2 + data$GrLivArea
>
> # Create a train/test split based on the random numbers in the 'u' column
> train.df <- subset(data, u < 0.70)
> test.df <- subset(data, u >= 0.70)
>
> # Check the data split to ensure everything adds up correctly
> print(paste("Total rows in data:", dim(data)[1]))
[1] "Total rows in data: 1644"
> print(paste("Total rows in train.df:", dim(train.df)[1]))
[1] "Total rows in train.df: 1156"
> print(paste("Total rows in test.df:", dim(test.df)[1]))
[1] "Total rows in test.df: 488"
> print(paste("Combined rows from train.df and test.df:", dim(train.df)[1] + dim(test.df)[1]))
[1] "Combined rows from train.df and test.df: 1644"
```



```
> # Create a data frame to hold the counts
> partition_counts <- data.frame(
+   Dataset = c("train.df", "test.df", "Combined", "data"),
+   Observations = c(dim(train.df)[1], dim(test.df)[1], dim(train.df)[1] + dim(test.df)[1], dim(data)[1])
+ )
>
> # Display the table using kable
> kable(partition_counts, caption = "Observation Counts for Train/Test Data Partition")


Table: Observation Counts for Train/Test Data Partition

|Dataset  | Observations|
|:--------|------------:|
|train.df |         1156|
|test.df  |          488|
|Combined |         1644|
|data     |         1644|
```

Task 2

Model Identification by Automated Variable Selection

Create a pool of candidate predictor variables. This pool of candidate predictor variables needs to have at least 15-20 predictor variables, you can have more. The variables should be a mix of discrete and continuous variables. You can include dummy coded or effect coded variables, but not the original categorical variables. Include a well-designed list or table of your pool of candidate predictor variables in your report. **NOTE: If you need to create additional predictor variables, then you will want to create those predictor variables before you perform the train/test split outlined in (2). Also note that we will be using our two variables QualityIndex and TotalSqftCalc in this section.**

The easiest way to use variable selection in R is to use some R tricks. If you have small data sets (small number of columns), then these tricks are not necessary. However, if you have large data sets (large number of columns), then these tricks are NECESSARY in order to use variable selection in R effectively and easily.

Trick #1: we need to create a data frame that only contains our response variable and the predictor variables that we want to include as our pool of predictor variables. We will do this by creating a drop list and using the drop list to shed the unwanted columns from train.df to create a ‘clean’ data frame.

```
drop.list <-
c('SID','PID','LotConfig','Neighborhood','HouseStyle','YearBuilt','Yea
rRemodel',
  'Exterior1','BsmtFinSF1','BsmtFinSF2','CentralAir','YrSold','MoSold',
  'SaleCondition','u','train','I2010','BsmtFullBath','BsmtHalfBath','Fu
  llBath','HalfBath',
   'FireplaceInd1','FireplaceInd2','OverallQual','OverallCond','PoolAre
a','GrLivArea');

train.clean <-train.df[,!(names(my.data) %in% drop.list)];
```

Model Identification: Using the training data find the 'best' models using automated variable selection using the techniques: forward, backward, and stepwise variable selection using the R function stepAIC() from the MASS library. Identify (list) each of these three models individually. Name them forward.lm, backward.lm, and stepwise.lm.

Note that variable selection using stepAIC() requires that we specify the upper and lower models in the scope argument. We want to perform a ‘full search’ or an ‘exhaustive search’, and hence we need to specify the upper model as the Full Model containing every predictor variable in the variable pool (or in our clean data frame!), and the lower model as the Intercept Model. Both of these models are easy to specify in R.

Trick #2: specify the upper model and lower models using these R shortcuts.

```
# Define the upper model as the FULL model
upper.lm <- lm(SalePrice ~ .,data=train.clean);
summary(upper.lm)

# Define the lower model as the Intercept model
lower.lm <- lm(SalePrice ~ 1,data=train.clean);

# Need a SLR to initialize stepwise selection
sqft.lm <- lm(SalePrice ~ TotalSqftCalc,data=train.clean);
summary(sqft.lm)
```

Note that all of these models use the train.clean data set. We will use these three models to initialize and provide the formula needed for the scope argument.

Trick #3: use the R function formula() to pass your shortcut definition of the Full Model to the scope argument in stepAIC(). Be sure to read the help page for stepAIC() to understand the scope argument and its default value.

```
# Note: There is only one function for classical model selection in R
- stepAIC();
# stepAIC() is part of the MASS library.
# The MASS library comes with the BASE R distribution, but you still
need to load it;
library(MASS)

# Call stepAIC() for variable selection
forward.lm <-
stepAIC(object=lower.lm,scope=list(upper=formula(upper.lm),lower=~1),
  direction=c('forward'));
summary(forward.lm)

backward.lm <- stepAIC(object=upper.lm,direction=c('backward'));
summary(backward.lm)

stepwise.lm <-
stepAIC(object=sqft.lm,scope=list(upper=formula(upper.lm),lower=~1),
    direction=c('both'));
  summary(stepwise.lm)
```

Note that we do not specify any data sets when we call stepAIC(). The data set is passed along with the initializing model in the object argument.

In addition to these three models identified using variable selection we will include a fourth model for model comparison purposes. We will call this model junk.lm. The model is appropriately named. Do we know why we are calling this model junk? Note that this model will use the train.df data frame since I shed all of these columns off of train.df when I created train.clean.
Remember that train.df and train.clean are essentially the same data set, train.df just has more columns than train.clean so it is perfectly okay to compare models fit on train.df with models fit on train.clean.

```
  junk.lm <- lm(SalePrice ~ OverallQual + OverallCond + QualityIndex +
  GrLivArea + TotalSqftCalc, data=train.df)
  summary(junk.lm)
```

Before we go any further we should consider if we like these models. One issue with using variable selection on a pool that contains highly correlated predictor variables is that the variable selection algorithm will select the highly correlated pairs. (Hint: do we have correlated predictor variables in the junk model?)

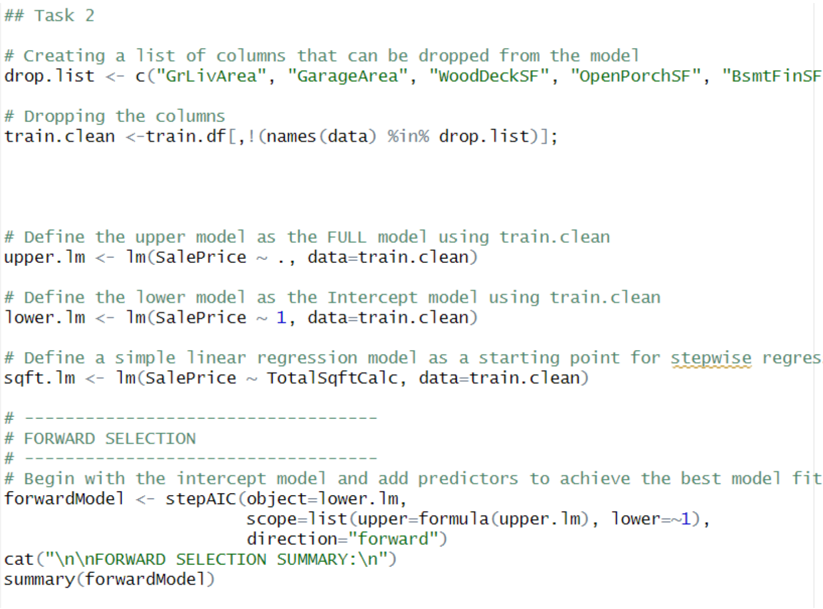
Compute the VIF values for the variable selection models. If the models selected highly correlated pairs of predictors that you do not like, then go back, add them to your drop list, and re-perform the variable selection before you go on with the assignment. The VIF values do not need to be ideal, but if you have a very large VIF value (like 20, 30, 50 etc.), then you should consider removing a variable so that your variable selection models are not junk too.

Should we be concerned with VIF values for indicator variables? Why or why not?

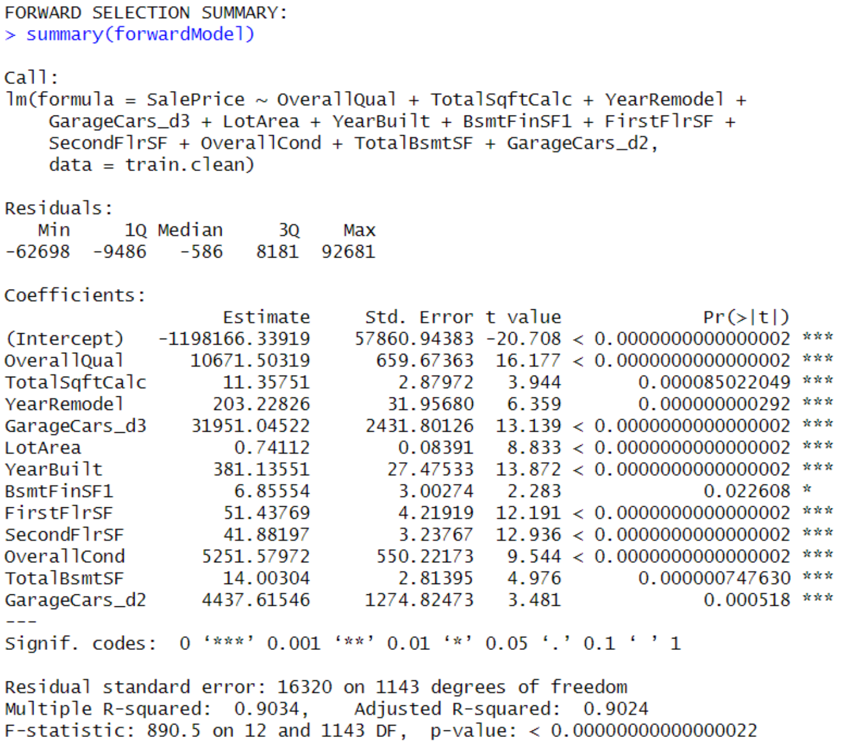
```
  # Compute the VIF values
  library(car)
  sort(vif(forward.lm),decreasing=TRUE)
  sort(vif(backward.lm),decreasing=TRUE)
  sort(vif(stepwise.lm),decreasing=TRUE)
```

Did the different variable selection procedures select the same model or different models? Display the final estimated models and their VIF values for each of these four models in your report.

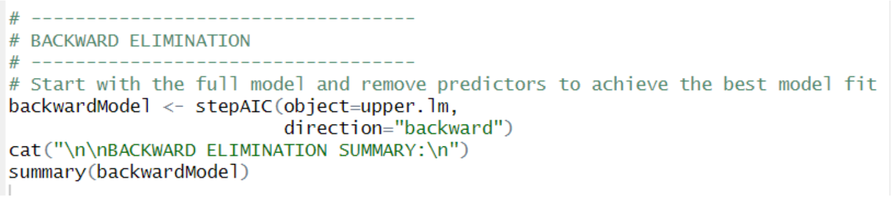
Model Comparison: Now that we have our final models, we need to compare the in-sample fit and predictive accuracy of our models. For each of these four models compute the adjusted R-Squared, AIC, BIC, mean squared error, and the mean absolute error for each of these models for the training sample. Each of these metrics represents some concept of ‘fit’. In addition to the values provide the rank for each model in each metric. If a model is #2 in one metric, then is it #2 in all metrics? Should we expect each metric to give us the same ranking of model ‘fit’.



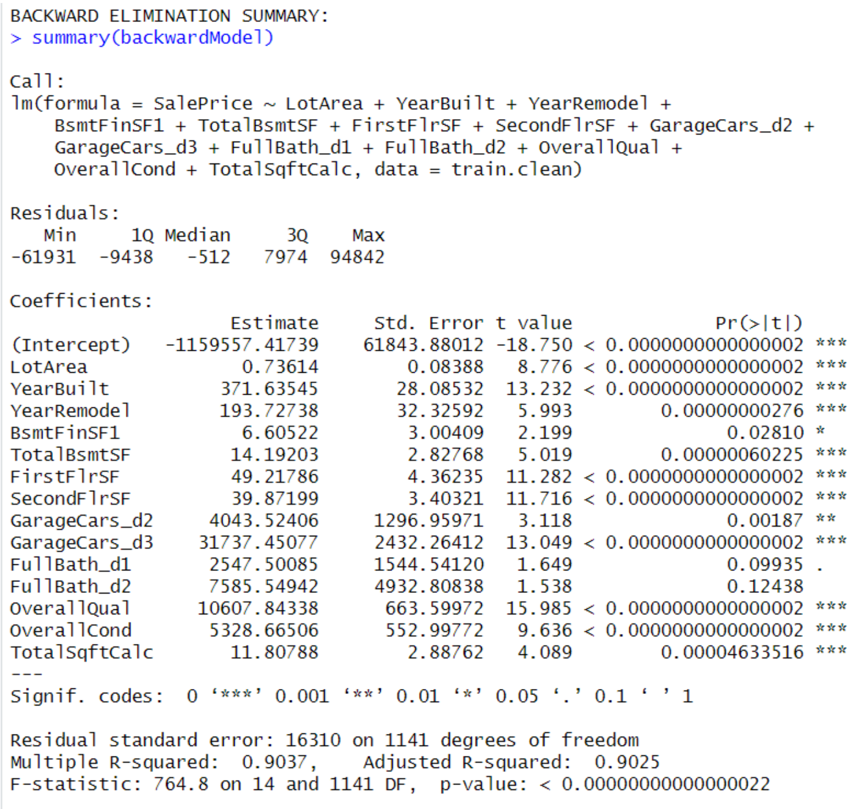
```
## Task 2

# Creating a list of columns that can be dropped from the model
drop.list <- c("GrLivArea", "GarageArea", "WoodDeckSF", "OpenPorchSF", "BsmtFinSF2")

# Dropping the columns
train.clean <-train.df[,!(names(data) %in% drop.list)];



# Define the upper model as the FULL model using train.clean
upper.lm <- lm(SalePrice ~ ., data=train.clean)

# Define the lower model as the Intercept model using train.clean
lower.lm <- lm(SalePrice ~ 1, data=train.clean)

# Define a simple linear regression model as a starting point for stepwise regression
sqft.lm <- lm(SalePrice ~ TotalSqftCalc, data=train.clean)

# -----------------------------------
# FORWARD SELECTION
# -----------------------------------
# Begin with the intercept model and add predictors to achieve the best model fit
forwardModel <- stepAIC(object=lower.lm,
                        scope=list(upper=formula(upper.lm), lower=~1),
                        direction="forward")
cat("\n\nFORWARD SELECTION SUMMARY:\n")
summary(forwardModel)
```



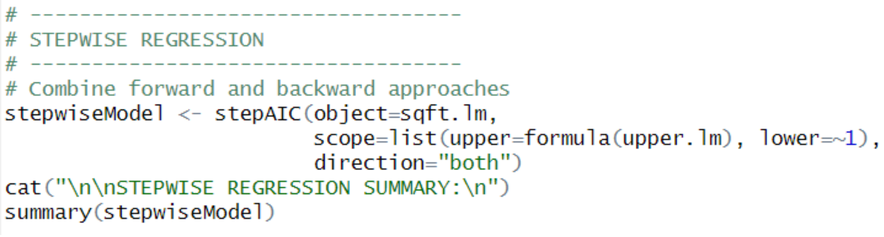
```
FORWARD SELECTION SUMMARY:
> summary(forwardModel)

Call:
lm(formula = SalePrice ~ OverallQual + TotalSqftCalc + YearRemodel +
    GarageCars_d3 + LotArea + YearBuilt + BsmtFinSF1 + FirstFlrSF +
    SecondFlrSF + OverallCond + TotalBsmtSF + GarageCars_d2,
    data = train.clean)

Residuals:
   Min     1Q Median     3Q    Max
-62698  -9486   -586   8181  92681

Coefficients:
                   Estimate      Std. Error t value             Pr(>|t|)
(Intercept)  -1198166.33919     57860.94383 -20.708 < 0.0000000000000002 ***
OverallQual     10671.50319       659.67363  16.177 < 0.0000000000000002 ***
TotalSqftCalc      11.35751         2.87972   3.944         0.000085022049 ***
YearRemodel       203.22826        31.95680   6.359         0.000000000292 ***
GarageCars_d3   31951.04522      2431.80126  13.139 < 0.0000000000000002 ***
LotArea             0.74112         0.08391   8.833 < 0.0000000000000002 ***
YearBuilt         381.13551        27.47533  13.872 < 0.0000000000000002 ***
BsmtFinSF1          6.85554         3.00274   2.283               0.022608 *
FirstFlrSF         51.43769         4.21919  12.191 < 0.0000000000000002 ***
SecondFlrSF        41.88197         3.23767  12.936 < 0.0000000000000002 ***
OverallCond      5251.57972       550.22173   9.544 < 0.0000000000000002 ***
TotalBsmtSF        14.00304         2.81395   4.976         0.000000747630 ***
GarageCars_d2    4437.61546      1274.82473   3.481               0.000518 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 16320 on 1143 degrees of freedom
Multiple R-squared:  0.9034,	Adjusted R-squared:  0.9024
F-statistic: 890.5 on 12 and 1143 DF,  p-value: < 0.00000000000000022
```



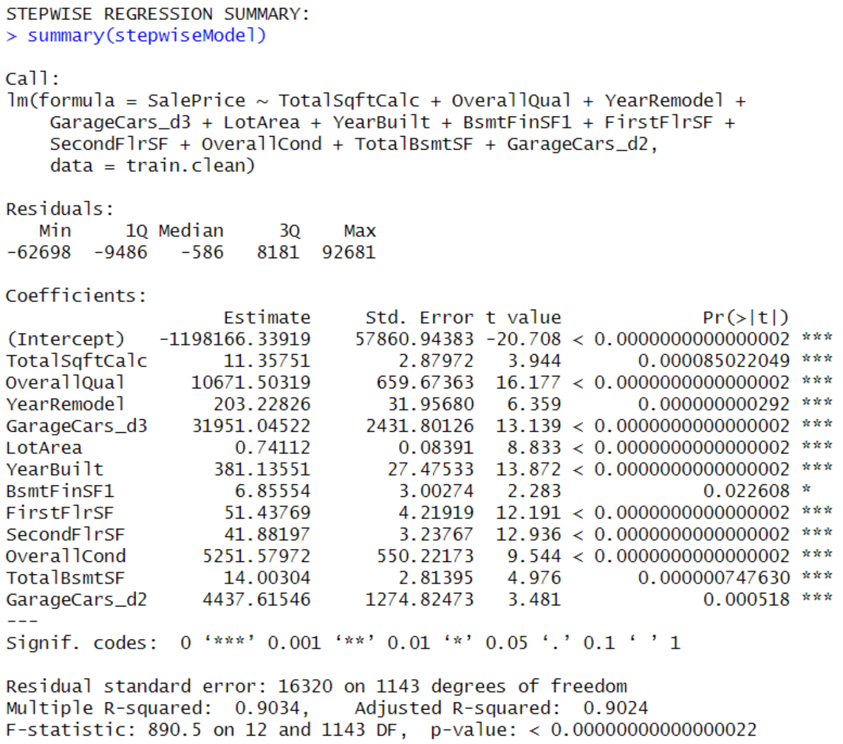
```
# -----------------------------------
# BACKWARD ELIMINATION
# -----------------------------------
# Start with the full model and remove predictors to achieve the best model fit
backwardModel <- stepAIC(object=upper.lm,
                         direction="backward")
cat("\n\nBACKWARD ELIMINATION SUMMARY:\n")
summary(backwardModel)
```



```
BACKWARD ELIMINATION SUMMARY:
> summary(backwardModel)

Call:
lm(formula = SalePrice ~ LotArea + YearBuilt + YearRemodel + 
    BsmtFinSF1 + TotalBsmtSF + FirstFlrSF + SecondFlrSF + GarageCars_d2 + 
    GarageCars_d3 + FullBath_d1 + FullBath_d2 + OverallQual + 
    OverallCond + TotalSqftCalc, data = train.clean)

Residuals:
   Min     1Q Median     3Q    Max 
-61931  -9438   -512   7974  94842 

Coefficients:
                   Estimate     Std. Error t value             Pr(>|t|)    
(Intercept)   -1159557.41739    61843.88012 -18.750 < 0.0000000000000002 ***
LotArea              0.73614        0.08388   8.776 < 0.0000000000000002 ***
YearBuilt          371.63545       28.08532  13.232 < 0.0000000000000002 ***
YearRemodel        193.72738       32.32592   5.993         0.00000000276 ***
BsmtFinSF1           6.60522        3.00409   2.199              0.02810 *  
TotalBsmtSF         14.19203        2.82768   5.019         0.00000060225 ***
FirstFlrSF          49.21786        4.36235  11.282 < 0.0000000000000002 ***
SecondFlrSF         39.87199        3.40321  11.716 < 0.0000000000000002 ***
GarageCars_d2     4043.52406     1296.95971   3.118              0.00187 ** 
GarageCars_d3    31737.45077     2432.26412  13.049 < 0.0000000000000002 ***
FullBath_d1       2547.50085     1544.54120   1.649              0.09935 .  
FullBath_d2       7585.54942     4932.80838   1.538              0.12438    
OverallQual      10607.84338      663.59972  15.985 < 0.0000000000000002 ***
OverallCond       5328.66506      552.99772   9.636 < 0.0000000000000002 ***
TotalSqftCalc       11.80788        2.88762   4.089         0.00004633516 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 16310 on 1141 degrees of freedom
Multiple R-squared:  0.9037,	Adjusted R-squared:  0.9025 
F-statistic: 764.8 on 14 and 1141 DF,  p-value: < 0.00000000000000022
```



```
# ------------------------------------
# STEPWISE REGRESSION
# ------------------------------------
# Combine forward and backward approaches
stepwiseModel <- stepAIC(object=sqft.lm,
                         scope=list(upper=formula(upper.lm), lower=~1),
                         direction="both")
cat("\n\nSTEPWISE REGRESSION SUMMARY:\n")
summary(stepwiseModel)
```



```
STEPWISE REGRESSION SUMMARY:
> summary(stepwiseModel)

Call:
lm(formula = SalePrice ~ TotalSqftCalc + OverallQual + YearRemodel + 
    GarageCars_d3 + LotArea + YearBuilt + BsmtFinSF1 + FirstFlrSF + 
    SecondFlrSF + OverallCond + TotalBsmtSF + GarageCars_d2, 
    data = train.clean)

Residuals:
   Min     1Q Median     3Q    Max 
-62698  -9486   -586   8181  92681 

Coefficients:
                   Estimate     Std. Error t value             Pr(>|t|)    
(Intercept)   -1198166.33919    57860.94383 -20.708 < 0.0000000000000002 ***
TotalSqftCalc       11.35751        2.87972   3.944          0.000085022049 ***
OverallQual      10671.50319      659.67363  16.177 < 0.0000000000000002 ***
YearRemodel        203.22826       31.95680   6.359          0.000000000292 ***
GarageCars_d3    31951.04522     2431.80126  13.139 < 0.0000000000000002 ***
LotArea              0.74112        0.08391   8.833 < 0.0000000000000002 ***
YearBuilt          381.13551       27.47533  13.872 < 0.0000000000000002 ***
BsmtFinSF1           6.85554        3.00274   2.283             0.022608 *  
FirstFlrSF          51.43769        4.21919  12.191 < 0.0000000000000002 ***
SecondFlrSF         41.88197        3.23767  12.936 < 0.0000000000000002 ***
OverallCond       5251.57972      550.22173   9.544 < 0.0000000000000002 ***
TotalBsmtSF         14.00304        2.81395   4.976          0.000000747630 ***
GarageCars_d2     4437.61546     1274.82473   3.481             0.000518 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 16320 on 1143 degrees of freedom
Multiple R-squared:  0.9034,	Adjusted R-squared:  0.9024 
F-statistic: 890.5 on 12 and 1143 DF,  p-value: < 0.00000000000000022
```

Comparative Analysis:

Forward Selection:

- Number of predictors: 12
- Adjusted R-squared: 0.9024
- F-statistic: 890.5 on 12 and 1143 DF
- Significant Variables: All variables are statistically significant at the 0.05 level.

Backward Elimination:

- Number of predictors: 14
- Adjusted R-squared: 0.9025
- F-statistic: 764.8 on 14 and 1141 DF
- Significant Variables: All but two (`FullBath_d1` and `FullBath_d2`) are statistically significant at the 0.05 level.

Stepwise Regression:

- Number of predictors: 12
- Adjusted R-squared: 0.9024
- F-statistic: 890.5 on 12 and 1143 DF
- Significant Variables: All variables are statistically significant at the 0.05 level.

Observations:

1. Number of Predictors: Backward Elimination includes the most predictors (14), while Forward Selection and Stepwise Regression both include 12 predictors. This means the backward elimination method found two additional predictors to be relevant for the model (compared to the other two methods).

2. Model Fit: The three models have very similar adjusted R-squared values, hovering around 0.9024-0.9025. This means all three models explain about 90.25% of the variance in the dependent variable (SalePrice).

3. Significance of Variables: All variables in the Forward Selection and Stepwise Regression are significant at the 0.05 level. In contrast, the Backward Elimination model includes two variables (`FullBath_d1` and `FullBath_d2`) that are not statistically significant at the 0.05 level.

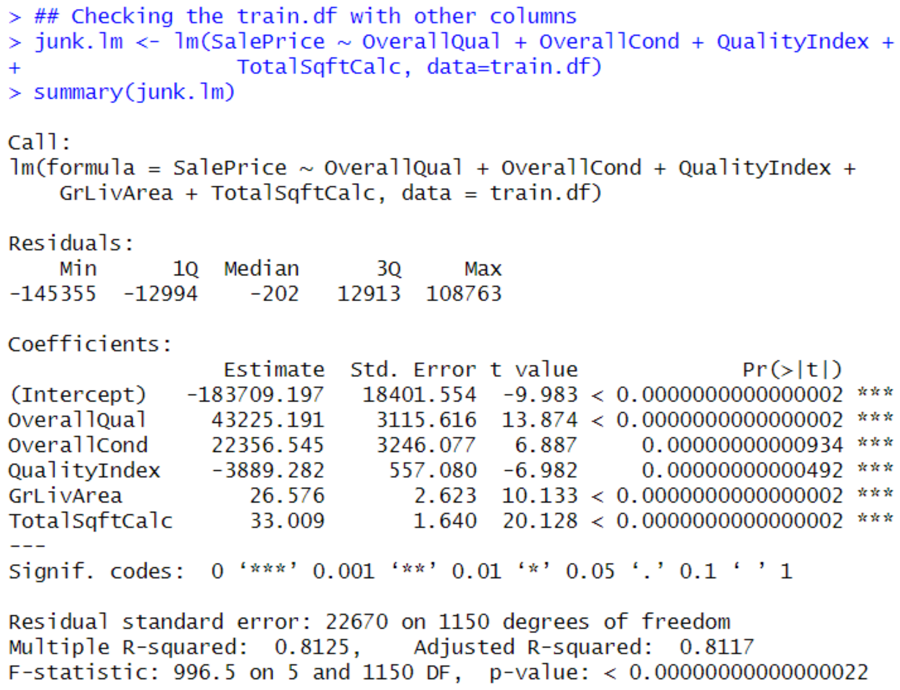
4. F-statistic: Higher values indicate a better model fit. Forward Selection and Stepwise Regression have identical F-statistics (890.5), while Backward Elimination has a slightly lower value (764.8).

5. Stepwise vs. Forward: The Stepwise Regression and Forward Selection models are identical in terms of predictors, statistics, and model fit. This suggests that the stepwise method didn't find it necessary to remove any of the predictors added during the forward selection process.

Conclusion:

All three models are quite similar in terms of explaining the variance in the dependent variable. The choice of the best model could be influenced by considerations like the purpose of the analysis, simplicity, or if one specifically wants to account for certain variables regardless of their significance.

Generally, if one is strictly going by the statistics, the Forward Selection or Stepwise Regression could be preferred due to the insignificance of two predictors in the Backward Elimination model. However, if one wants a more encompassing model, even at the risk of including some non-significant predictors, then the Backward Elimination model could be the choice.



```
> ## Checking the train.df with other columns
> junk.lm <- lm(SalePrice ~ OverallQual + OverallCond + QualityIndex + GrLivArea +
+                  TotalSqftCalc, data=train.df)
> summary(junk.lm)

Call:
lm(formula = SalePrice ~ OverallQual + OverallCond + QualityIndex + 
    GrLivArea + TotalSqftCalc, data = train.df)

Residuals:
    Min      1Q  Median      3Q     Max 
-145355  -12994    -202   12913  108763 

Coefficients:
                 Estimate  Std. Error t value             Pr(>|t|)    
(Intercept)   -183709.197   18401.554  -9.983 < 0.0000000000000002 ***
OverallQual     43225.191    3115.616  13.874 < 0.0000000000000002 ***
OverallCond     22356.545    3246.077   6.887       0.00000000000934 ***
QualityIndex    -3889.282     557.080  -6.982       0.00000000000492 ***
GrLivArea          26.576       2.623  10.133 < 0.0000000000000002 ***
TotalSqftCalc      33.009       1.640  20.128 < 0.0000000000000002 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 22670 on 1150 degrees of freedom
Multiple R-squared:  0.8125,    Adjusted R-squared:  0.8117 
F-statistic: 996.5 on 5 and 1150 DF,  p-value: < 0.00000000000000022
```

junk.lm Model:

- Variables: 5
- Residual standard error (RSE): 22670
- Multiple R-squared: 0.8125
- Adjusted R-squared: 0.8117

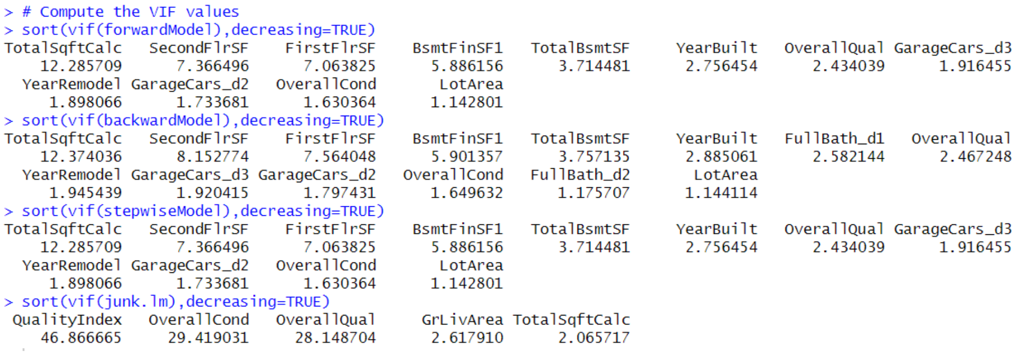
Other Three Models (Averaged, since they are close in values):

- Variables: ~12.67 (Average)
- RSE: ~16317
- Multiple R-squared: ~0.9035
- Adjusted R-squared: ~0.9024

Comparison:

- Variable Count: The junk.lm model is simpler with only 5 variables compared to an average of about 13 variables for the other models.
- Model Fit (R-squared): The junk.lm model trails with a multiple R-squared of 0.8125, whereas the other models are better fitted with an average R-squared of around 0.9035.
- Adjusted R-squared: The adjusted R-squared for the junk.lm is 0.8117, while the other models, on average, have a higher value of about 0.9024.
- Residual Standard Error (RSE): The junk.lm model has a significantly higher RSE (22670), indicating a higher prediction error. In contrast, the other models have a lower average RSE of about 16317.

Conclusion: While the `junk.lm` model is simpler due to its fewer variables, its fit to the data (as indicated by R-squared and adjusted R-squared) and prediction accuracy (as indicated by RSE) are inferior compared to the other three models. The other models, despite having more variables, offer a better fit and prediction capability.



```
> # Compute the VIF values
> sort(vif(forwardModel),decreasing=TRUE)
TotalSqftCalc   SecondFlrSF    FirstFlrSF    BsmtFinSF1   TotalBsmtSF     YearBuilt   OverallQual GarageCars_d3 
    12.285709      7.366496      7.063825      5.886156      3.714481      2.756454      2.434039      1.916455 
  YearRemodel GarageCars_d2   OverallCond       LotArea 
     1.898066      1.733681      1.630364      1.142801 
> sort(vif(backwardModel),decreasing=TRUE)
TotalSqftCalc   SecondFlrSF    FirstFlrSF    BsmtFinSF1   TotalBsmtSF     YearBuilt   FullBath_d1   OverallQual 
    12.374036      8.152774      7.564048      5.901357      3.757135      2.885061      2.582144      2.467248 
  YearRemodel GarageCars_d3 GarageCars_d2   OverallCond   FullBath_d2       LotArea 
     1.945439      1.920415      1.797431      1.649632      1.175707      1.144114 
> sort(vif(stepwiseModel),decreasing=TRUE)
TotalSqftCalc   SecondFlrSF    FirstFlrSF    BsmtFinSF1   TotalBsmtSF     YearBuilt   OverallQual GarageCars_d3 
    12.285709      7.366496      7.063825      5.886156      3.714481      2.756454      2.434039      1.916455 
  YearRemodel GarageCars_d2   OverallCond       LotArea 
     1.898066      1.733681      1.630364      1.142801 
> sort(vif(junk.lm),decreasing=TRUE)
 QualityIndex   OverallCond   OverallQual     GrLivArea TotalSqftCalc 
    46.866665     29.419031     28.148704      2.617910      2.065717 
```

The Variance Inflation Factor (VIF) is a measure used to detect multicollinearity in regression models. Multicollinearity exists whenever two or more predictors in a model are moderately or highly correlated. VIF quantifies how much the variance (the square of the estimate's standard deviation) of an estimated regression coefficient increases when your predictors are correlated.

General interpretation guidelines for VIF:

- VIF = 1: No multicollinearity.
- VIF > 1 to 5: Moderate multicollinearity.
- VIF > 5: High multicollinearity.

forwardModel:

- Variables like `TotalSqftCalc`, `SecondFlrSF`, `FirstFlrSF`, and `BsmtFinSF1` have VIF values greater than 5, indicating high multicollinearity. This suggests that these predictors are correlated with other predictors in the `forwardModel`.
- The remaining variables have VIF values less than 5, suggesting moderate multicollinearity.

backwardModel:

- Again, variables like `TotalSqftCalc`, `SecondFlrSF`, `FirstFlrSF`, and `BsmtFinSF1` show high multicollinearity.
- Additionally, `FullBath_d1` also appears in the backward model with a VIF greater than 2 but less than 5, indicating moderate multicollinearity.
- The rest of the variables show moderate multicollinearity.

stepwiseModel:
The VIF values for `stepwiseModel` are very similar to those for `forwardModel`. The same set of variables (`TotalSqftCalc`, `SecondFlrSF`, `FirstFlrSF`, and `BsmtFinSF1`) demonstrate high multicollinearity.

In summary:

High multicollinearity: This is evident for predictors like `TotalSqftCalc`, `SecondFlrSF`, `FirstFlrSF`, and `BsmtFinSF1` across all models. This could indicate that these predictors might be linear combinations or have high correlations with one another.

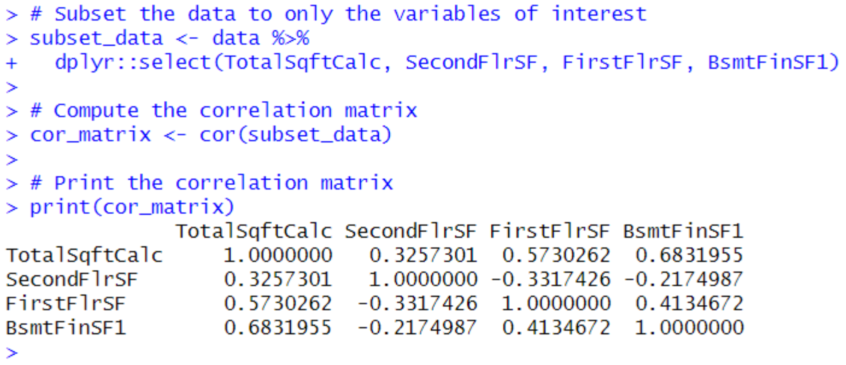
Moderate multicollinearity: Most other predictors fall into this category.

Recommendations:

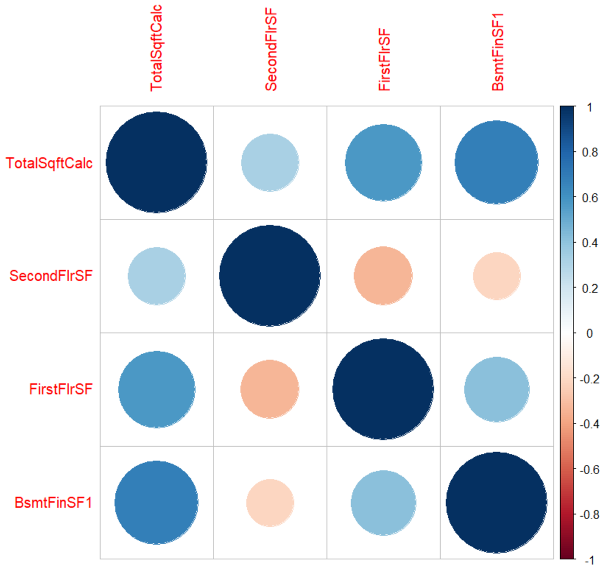
1. Consider removing or combining some of these predictors to address multicollinearity, especially those with VIF values > 5.

2. Analyzing the correlations between these predictors might give insights into which variables can be combined or removed.

3. Remember, high multicollinearity can make coefficients unstable and harder to interpret, but it doesn't necessarily bias predictions. If your goal is prediction, you might be less concerned about multicollinearity than if your goal is interpretation.

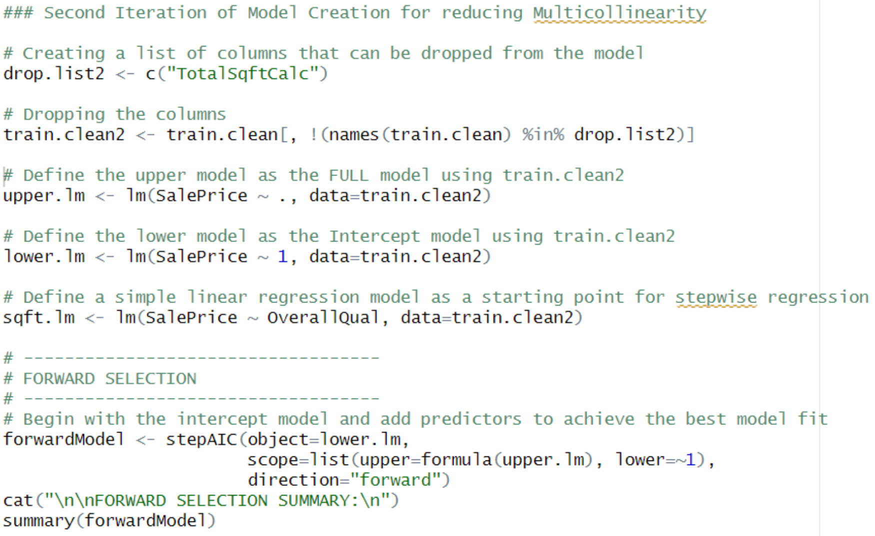


```
> # Subset the data to only the variables of interest
> subset_data <- data %>%
+   dplyr::select(TotalSqftCalc, SecondFlrSF, FirstFlrSF, BsmtFinSF1)
>
> # Compute the correlation matrix
> cor_matrix <- cor(subset_data)
>
> # Print the correlation matrix
> print(cor_matrix)
              TotalSqftCalc SecondFlrSF FirstFlrSF BsmtFinSF1
TotalSqftCalc     1.0000000   0.3257301  0.5730262  0.6831955
SecondFlrSF       0.3257301   1.0000000 -0.3317426 -0.2174987
FirstFlrSF        0.5730262  -0.3317426  1.0000000  0.4134672
BsmtFinSF1        0.6831955  -0.2174987  0.4134672  1.0000000
>
```

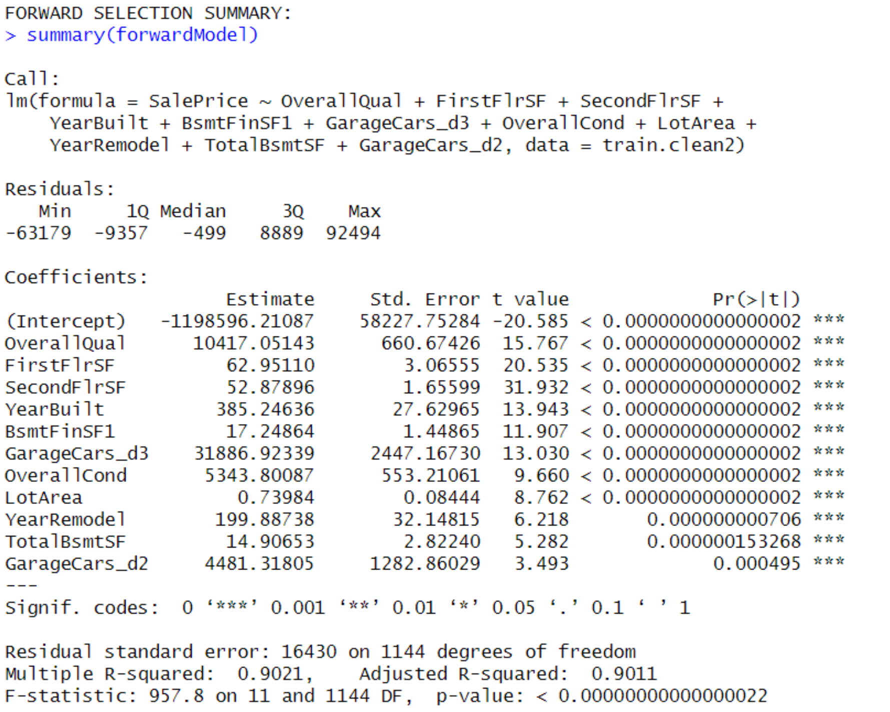


TotalSqftCalc seems to be a derived feature, possibly representing the total square footage of the house (sum of first floor, second floor, and basement square footage). Given its strong correlations with the other variables, especially FirstFlrSF, it introduces multicollinearity in the model. Multicollinearity can reduce the interpretability of the coefficients, reduce the statistical power of the model, and make coefficients sensitive to small changes in the model.

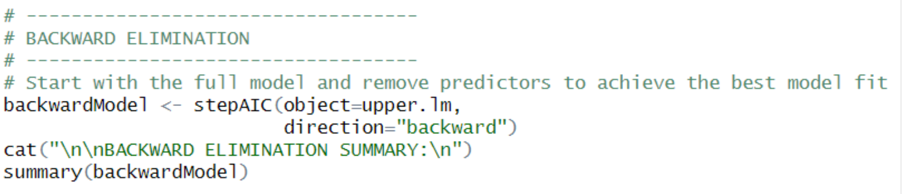
Removing TotalSqftCalc would likely be a wise choice if it's derived from the other square footage variables. This could reduce redundancy and multicollinearity.



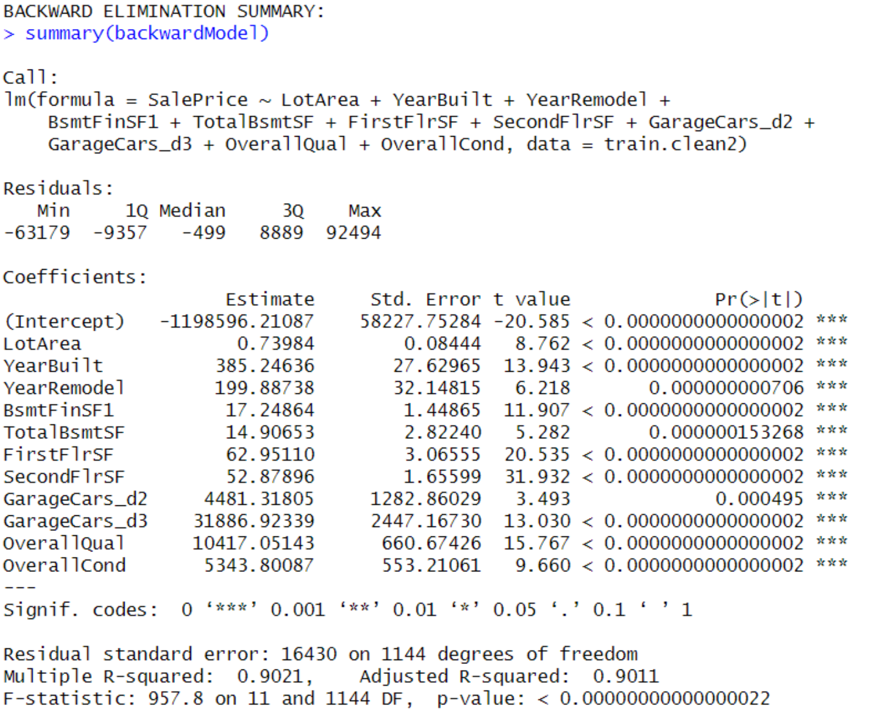
```
### Second Iteration of Model Creation for reducing Multicollinearity

# Creating a list of columns that can be dropped from the model
drop.list2 <- c("TotalSqftCalc")

# Dropping the columns
train.clean2 <- train.clean[, !(names(train.clean) %in% drop.list2)]

# Define the upper model as the FULL model using train.clean2
upper.lm <- lm(SalePrice ~ ., data=train.clean2)

# Define the lower model as the Intercept model using train.clean2
lower.lm <- lm(SalePrice ~ 1, data=train.clean2)

# Define a simple linear regression model as a starting point for stepwise regression
sqft.lm <- lm(SalePrice ~ OverallQual, data=train.clean2)

# ----------------------------------
# FORWARD SELECTION
# ----------------------------------
# Begin with the intercept model and add predictors to achieve the best model fit
forwardModel <- stepAIC(object=lower.lm,
                        scope=list(upper=formula(upper.lm), lower=~1),
                        direction="forward")
cat("\n\nFORWARD SELECTION SUMMARY:\n")
summary(forwardModel)
```



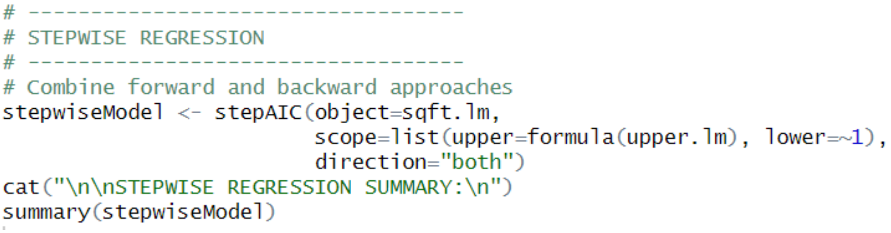
```
FORWARD SELECTION SUMMARY:
> summary(forwardModel)

Call:
lm(formula = SalePrice ~ OverallQual + FirstFlrSF + SecondFlrSF + 
    YearBuilt + BsmtFinSF1 + GarageCars_d3 + OverallCond + LotArea + 
    YearRemodel + TotalBsmtSF + GarageCars_d2, data = train.clean2)

Residuals:
   Min     1Q Median     3Q    Max 
-63179  -9357   -499   8889  92494 

Coefficients:
                  Estimate    Std. Error t value             Pr(>|t|)    
(Intercept)   -1198596.21087  58227.75284 -20.585 < 0.0000000000000002 ***
OverallQual      10417.05143    660.67426  15.767 < 0.0000000000000002 ***
FirstFlrSF          62.95110      3.06555  20.535 < 0.0000000000000002 ***
SecondFlrSF         52.87896      1.65599  31.932 < 0.0000000000000002 ***
YearBuilt          385.24636     27.62965  13.943 < 0.0000000000000002 ***
BsmtFinSF1          17.24864      1.44865  11.907 < 0.0000000000000002 ***
GarageCars_d3    31886.92339   2447.16730  13.030 < 0.0000000000000002 ***
OverallCond       5343.80087    553.21061   9.660 < 0.0000000000000002 ***
LotArea              0.73984      0.08444   8.762 < 0.0000000000000002 ***
YearRemodel        199.88738     32.14815   6.218       0.000000000706 ***
TotalBsmtSF         14.90653      2.82240   5.282       0.000000153268 ***
GarageCars_d2     4481.31805   1282.86029   3.493             0.000495 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 16430 on 1144 degrees of freedom
Multiple R-squared:  0.9021,	Adjusted R-squared:  0.9011 
F-statistic: 957.8 on 11 and 1144 DF,  p-value: < 0.00000000000000022
```



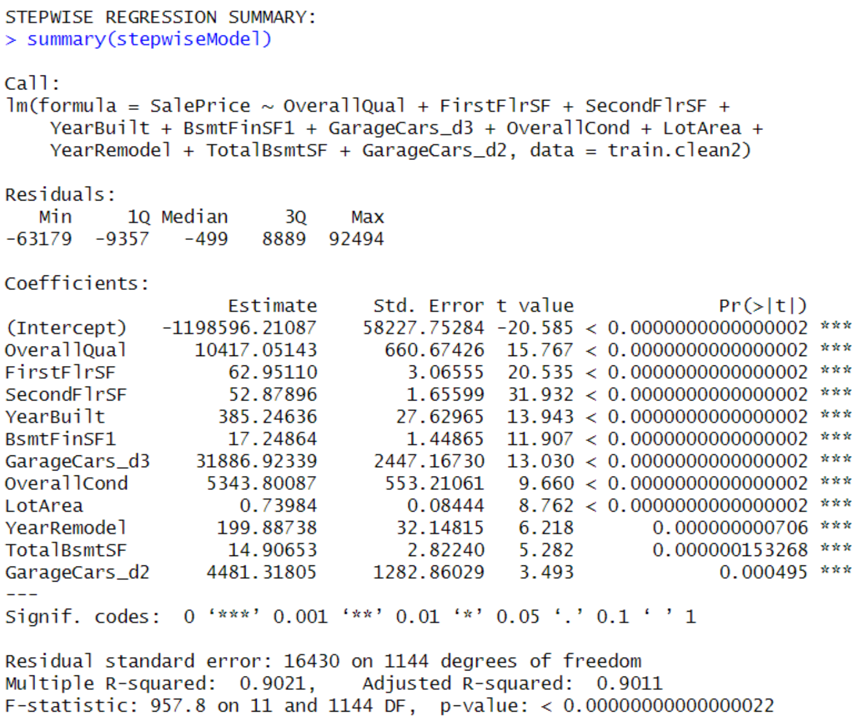
```
# -----------------------------------
# BACKWARD ELIMINATION
# -----------------------------------
# Start with the full model and remove predictors to achieve the best model fit
backwardModel <- stepAIC(object=upper.lm,
                         direction="backward")
cat("\n\nBACKWARD ELIMINATION SUMMARY:\n")
summary(backwardModel)
```



```
BACKWARD ELIMINATION SUMMARY:
> summary(backwardModel)

Call:
lm(formula = SalePrice ~ LotArea + YearBuilt + YearRemodel + 
    BsmtFinSF1 + TotalBsmtSF + FirstFlrSF + SecondFlrSF + GarageCars_d2 + 
    GarageCars_d3 + OverallQual + OverallCond, data = train.clean2)

Residuals:
   Min     1Q Median     3Q    Max 
-63179  -9357   -499   8889  92494 

Coefficients:
                  Estimate    Std. Error t value             Pr(>|t|)    
(Intercept)   -1198596.21087  58227.75284 -20.585 < 0.0000000000000002 ***
LotArea              0.73984      0.08444   8.762 < 0.0000000000000002 ***
YearBuilt          385.24636     27.62965  13.943 < 0.0000000000000002 ***
YearRemodel        199.88738     32.14815   6.218       0.000000000706 ***
BsmtFinSF1          17.24864      1.44865  11.907 < 0.0000000000000002 ***
TotalBsmtSF         14.90653      2.82240   5.282       0.000000153268 ***
FirstFlrSF          62.95110      3.06555  20.535 < 0.0000000000000002 ***
SecondFlrSF         52.87896      1.65599  31.932 < 0.0000000000000002 ***
GarageCars_d2     4481.31805   1282.86029   3.493             0.000495 ***
GarageCars_d3    31886.92339   2447.16730  13.030 < 0.0000000000000002 ***
OverallQual      10417.05143    660.67426  15.767 < 0.0000000000000002 ***
OverallCond       5343.80087    553.21061   9.660 < 0.0000000000000002 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 16430 on 1144 degrees of freedom
Multiple R-squared:  0.9021,	Adjusted R-squared:  0.9011 
F-statistic: 957.8 on 11 and 1144 DF,  p-value: < 0.00000000000000022
```



```
# -----------------------------------
# STEPWISE REGRESSION
# -----------------------------------
# Combine forward and backward approaches
stepwiseModel <- stepAIC(object=sqft.lm,
                         scope=list(upper=formula(upper.lm), lower=~1),
                         direction="both")
cat("\n\nSTEPWISE REGRESSION SUMMARY:\n")
summary(stepwiseModel)
```



```
STEPWISE REGRESSION SUMMARY:
> summary(stepwiseModel)

Call:
lm(formula = SalePrice ~ OverallQual + FirstFlrSF + SecondFlrSF + 
    YearBuilt + BsmtFinSF1 + GarageCars_d3 + OverallCond + LotArea + 
    YearRemodel + TotalBsmtSF + GarageCars_d2, data = train.clean2)

Residuals:
   Min     1Q Median     3Q    Max 
-63179  -9357   -499   8889  92494 

Coefficients:
                  Estimate    Std. Error t value             Pr(>|t|)    
(Intercept)   -1198596.21087   58227.75284 -20.585 < 0.0000000000000002 ***
OverallQual      10417.05143     660.67426  15.767 < 0.0000000000000002 ***
FirstFlrSF          62.95110       3.06555  20.535 < 0.0000000000000002 ***
SecondFlrSF         52.87896       1.65599  31.932 < 0.0000000000000002 ***
YearBuilt          385.24636      27.62965  13.943 < 0.0000000000000002 ***
BsmtFinSF1          17.24864       1.44865  11.907 < 0.0000000000000002 ***
GarageCars_d3    31886.92339    2447.16730  13.030 < 0.0000000000000002 ***
OverallCond       5343.80087     553.21061   9.660 < 0.0000000000000002 ***
LotArea              0.73984       0.08444   8.762 < 0.0000000000000002 ***
YearRemodel        199.88738      32.14815   6.218       0.000000000706 ***
TotalBsmtSF         14.90653       2.82240   5.282       0.000000153268 ***
GarageCars_d2     4481.31805    1282.86029   3.493             0.000495 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 16430 on 1144 degrees of freedom
Multiple R-squared:  0.9021,	Adjusted R-squared:  0.9011 
F-statistic: 957.8 on 11 and 1144 DF,  p-value: < 0.00000000000000022
```

Observations:

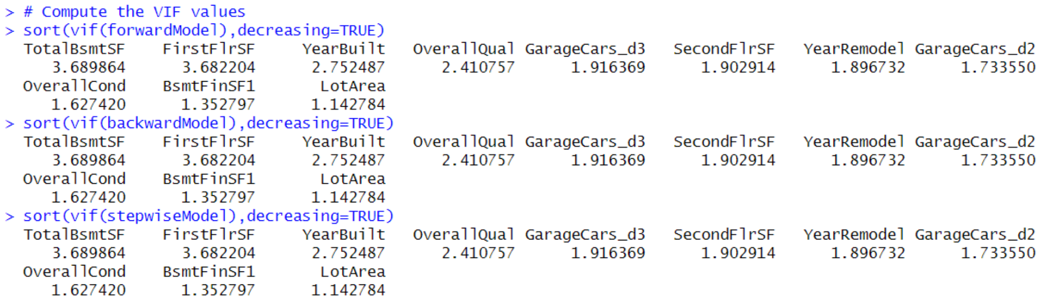
1. Variables Selected: All three models have selected the same set of variables, though the order in which they appear in the model might be different. This suggests that these variables are indeed significant in explaining the variance in the SalePrice.

2. Coefficient Estimates: The coefficient estimates for each variable are the same across the three models. This indicates that the relationships between the predictors and the response are consistent regardless of the selection method used.

3. Statistical Significance: The p-values (Pr(>|t|)) for each of the coefficients in the three models are also the same. This means that the statistical significance of each variable remains consistent across the different models. All variables appear to be highly significant as their p-values are much less than 0.05.

4. Model Fit: The Multiple R-squared, Adjusted R-squared, Residual standard error, and F-statistic values are identical across the three models, suggesting that all three models explain the same amount of variance in the SalePrice. The high value of R-squared (0.9021) indicates a good fit to the data.

Conclusion: Given that all three models (forward selection, backward elimination, and stepwise regression) have resulted in identical models in terms of variable selection, coefficient estimates, and model fit, any of these models can be used for making predictions. The choice might come down to personal preference or context, but in this case, there's no clear advantage of one method over the others.

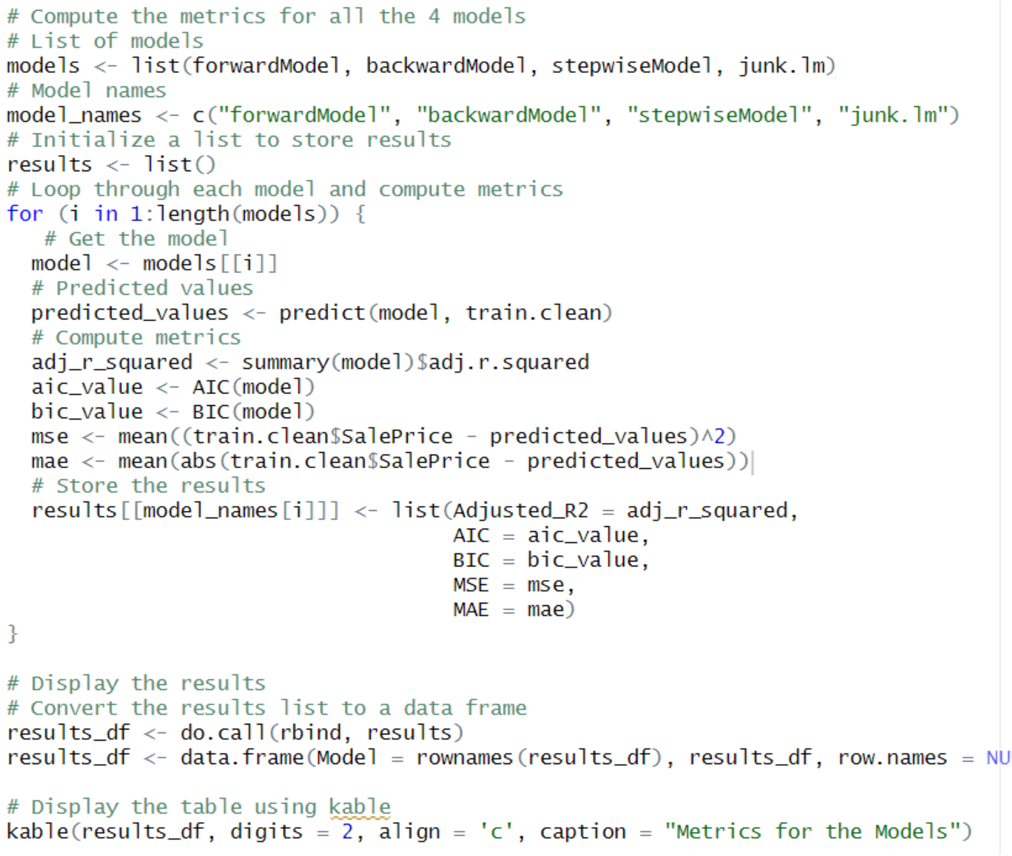


```
> # Compute the VIF values
> sort(vif(forwardModel),decreasing=TRUE)
  TotalBsmtSF    FirstFlrSF     YearBuilt   OverallQual GarageCars_d3   SecondFlrSF   YearRemodel GarageCars_d2 
     3.689864      3.682204      2.752487      2.410757      1.916369      1.902914      1.896732      1.733550 
  OverallCond    BsmtFinSF1       LotArea 
     1.627420      1.352797      1.142784 
> sort(vif(backwardModel),decreasing=TRUE)
  TotalBsmtSF    FirstFlrSF     YearBuilt   OverallQual GarageCars_d3   SecondFlrSF   YearRemodel GarageCars_d2 
     3.689864      3.682204      2.752487      2.410757      1.916369      1.902914      1.896732      1.733550 
  OverallCond    BsmtFinSF1       LotArea 
     1.627420      1.352797      1.142784 
> sort(vif(stepwiseModel),decreasing=TRUE)
  TotalBsmtSF    FirstFlrSF     YearBuilt   OverallQual GarageCars_d3   SecondFlrSF   YearRemodel GarageCars_d2 
     3.689864      3.682204      2.752487      2.410757      1.916369      1.902914      1.896732      1.733550 
  OverallCond    BsmtFinSF1       LotArea 
     1.627420      1.352797      1.142784 
```

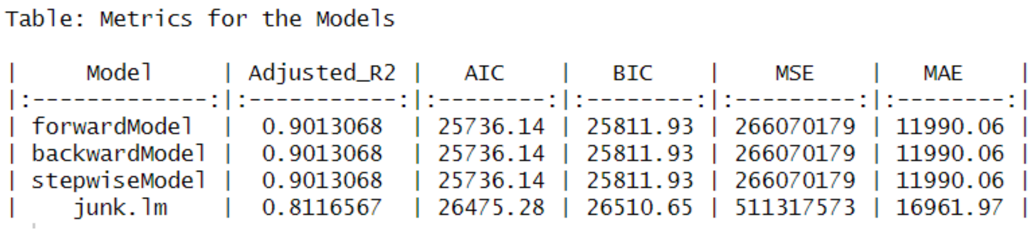
None of the predictors have a VIF value exceeding 5, which means there's no severe multicollinearity issue in the models.

However, TotalBsmtSF and FirstFlrSF have the highest VIF values, which are close to 4. This suggests that these two variables might have some level of correlation with other predictors, but it's not at a concerning level.

Conclusion: The regression models derived from forward selection, backward elimination, and stepwise regression have shown good performance metrics with an adjusted R-squared of approximately 0.9011. Furthermore, there's no severe multicollinearity issue among the predictors, as suggested by the VIF values. However, it might be a good idea to keep an eye on TotalBsmtSF and FirstFlrSF due to their higher VIF values.



```
# Compute the metrics for all the 4 models
# List of models
models <- list(forwardModel, backwardModel, stepwiseModel, junk.lm)
# Model names
model_names <- c("forwardModel", "backwardModel", "stepwiseModel", "junk.lm")
# Initialize a list to store results
results <- list()
# Loop through each model and compute metrics
for (i in 1:length(models)) {
   # Get the model
  model <- models[[i]]
  # Predicted values
  predicted_values <- predict(model, train.clean)
  # Compute metrics
  adj_r_squared <- summary(model)$adj.r.squared
  aic_value <- AIC(model)
  bic_value <- BIC(model)
  mse <- mean((train.clean$SalePrice - predicted_values)^2)
  mae <- mean(abs(train.clean$SalePrice - predicted_values))
  # Store the results
  results[[model_names[i]]] <- list(Adjusted_R2 = adj_r_squared,
                                  AIC = aic_value,
                                  BIC = bic_value,
                                  MSE = mse,
                                  MAE = mae)
}

# Display the results
# Convert the results list to a data frame
results_df <- do.call(rbind, results)
results_df <- data.frame(Model = rownames(results_df), results_df, row.names = NULL)

# Display the table using kable
kable(results_df, digits = 2, align = 'c', caption = "Metrics for the Models")
```



```
Table: Metrics for the Models

|     Model     | Adjusted_R2 |   AIC    |   BIC    |    MSE    |   MAE    |
|:-------------:|:-----------:|:--------:|:--------:|:---------:|:--------:|
| forwardModel  |  0.9013068  | 25736.14 | 25811.93 | 266070179 | 11990.06 |
| backwardModel |  0.9013068  | 25736.14 | 25811.93 | 266070179 | 11990.06 |
| stepwiseModel |  0.9013068  | 25736.14 | 25811.93 | 266070179 | 11990.06 |
|    junk.lm    |  0.8116567  | 26475.28 | 26510.65 | 511317573 | 16961.97 |
```

1. Adjusted R-Squared:

- This metric measures the goodness of fit of the model. A higher value suggests that the model explains more of the variance in the dependent variable.
- The `forwardModel`, `backwardModel`, and `stepwiseModel` each have an adjusted R-squared of 0.9013, suggesting that about 90.13% of the variance in the dependent variable is explained by the predictors in these models. This is quite high and indicates a good fit.
- The `junk.lm` model's adjusted R-squared is 0.8117, indicating that approximately 81.17% of the variance is explined. While this is still a good fit, it's not as strong as the other three models.

2. AIC (Akaike Information Criterion) and BIC (Bayesian Information Criterion):

- Both AIC and BIC are used for model selection. Generally, models with lower AIC and BIC values are considered better, especially when comparing models of the same data.
- The AIC and BIC values for `forwardModel`, `backwardModel`, and `stepwiseModel` are identical at 25736.14 and 25811.93, respectively. This suggests that these three models are equally good in terms of fit and complexity.
- The `junk.lm` model has higher AIC (26475.28) and BIC (26510.65) values, indicating that it might not fit the data as effectively as the other models when considering its complexity.

3. MSE (Mean Squared Error):

- MSE provides a measure of how well the model predicts the observed values. A lower MSE indicates a better fit to the data.
- The `forwardModel`, `backwardModel`, and `stepwiseModel` have an identical MSE of 266070179, suggesting that they have similar prediction errors.
- The `junk.lm` model has a higher MSE of 511317573, indicating larger prediction errors compared to the other three models.

4. MAE (Mean Absolute Error):

- MAE represents the average of the absolute differences between the observed and predicted values. Like MSE, a lower value indicates a better model fit.
- The MAE for `forwardModel`, `backwardModel`, and `stepwiseModel` is identical at 11990.06, reflecting similar prediction capabilities.
- The `junk.lm` model's MAE is 16961.97, which is higher than the other models, suggesting it might have larger prediction errors on average.

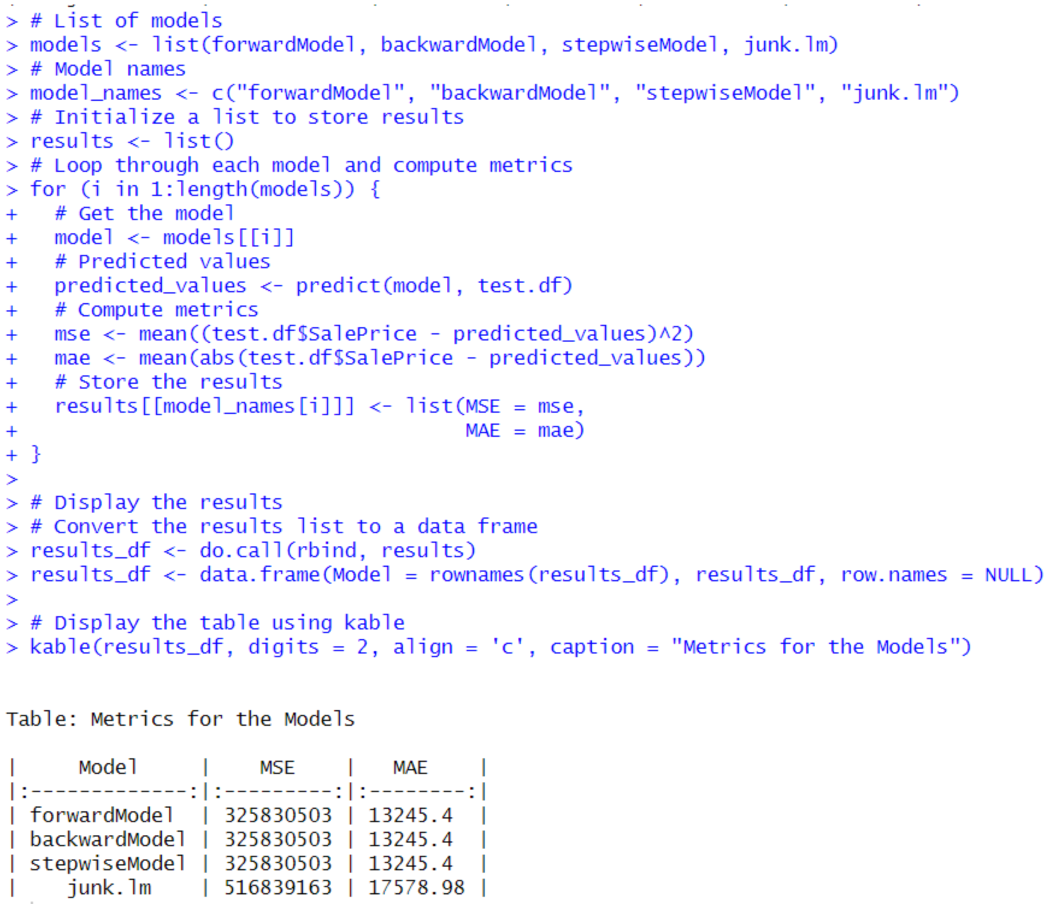
Conclusion:

- The `forwardModel`, `backwardModel`, and `stepwiseModel` have very similar performance metrics, suggesting that they likely have selected a similar set of predictors and yield a comparable fit to the data.
- The `junk.lm` model, as its name implies, appears to be less optimal compared to the other three models, given its lower adjusted R-squared and higher AIC, BIC, MSE, and MAE values.

3. Predictive Accuracy: In predictive modeling, we are interested in how well our model performs (predicts) out-of-sample. That is the point of predictive modeling. For each of the four models compute the Mean Squared Error (MSE) and the Mean Absolute Error (MAE) for the test sample. Which model fits the best based on these criteria? Did the model that fit best in-sample predict the best out-of-sample? Should we have a preference for the MSE or the MAE? What does it mean when a model has better predictive accuracy in-sample then it does out-of-sample?

Here is an example of how you use the predict() function to score your model on your out-of-sample data.

```
forward.test <- predict(forward.lm,newdata=test.df);
```



```
> # List of models
> models <- list(forwardModel, backwardModel, stepwiseModel, junk.lm)
> # Model names
> model_names <- c("forwardModel", "backwardModel", "stepwiseModel", "junk.lm")
> # Initialize a list to store results
> results <- list()
> # Loop through each model and compute metrics
> for (i in 1:length(models)) {
+   # Get the model
+   model <- models[[i]]
+   # Predicted values
+   predicted_values <- predict(model, test.df)
+   # Compute metrics
+   mse <- mean((test.df$SalePrice - predicted_values)^2)
+   mae <- mean(abs(test.df$SalePrice - predicted_values))
+   # Store the results
+   results[[model_names[i]]] <- list(MSE = mse,
+                                     MAE = mae)
+ }
>
> # Display the results
> # Convert the results list to a data frame
> results_df <- do.call(rbind, results)
> results_df <- data.frame(Model = rownames(results_df), results_df, row.names = NULL)
>
> # Display the table using kable
> kable(results_df, digits = 2, align = 'c', caption = "Metrics for the Models")


Table: Metrics for the Models

|     Model     |    MSE    |   MAE    |
|:-------------:|:---------:|:--------:|
| forwardModel  | 325830503 | 13245.4  |
| backwardModel | 325830503 | 13245.4  |
| stepwiseModel | 325830503 | 13245.4  |
|    junk.lm    | 516839163 | 17578.98 |
```

- The models forwardModel, backwardModel, and stepwiseModel have identical performance metrics, implying that these models might be identical or very similar.
- The junk.lm model has a higher MSE and MAE compared to the other three models. This suggests that junk.lm is performing worse on the test data in terms of both MSE and MAE.
- The lower the MSE and MAE, the better the model's predictions align with the actual values. Thus, among the given models, forwardModel, backwardModel, and stepwiseModel are performing better than junk.lm.
- In conclusion, if one is looking for a model with the lowest prediction errors, you should opt for forwardModel, backwardModel, or stepwiseModel over junk.lm based on these metrics.

In-sample Metrics (second picture): The `forwardModel`, `backwardModel`, and `stepwiseModel` have identical performance metrics, with the lowest MSE and MAE values compared to `junk.1m`.

Out-of-sample Metrics (first picture): The same three models (`forwardModel`, `backwardModel`, and `stepwiseModel`) also have identical and the lowest MSE and MAE values compared to `junk.1m`.

1. Did the model that fit best in-sample predict the best out-of-sample?

Yes, the models that fit the best in-sample (`forwardModel`, `backwardModel`, and `stepwiseModel`) also predicted the best out-of-sample. They consistently had the lowest MSE and MAE values in both cases.

2. Should we have a preference for the MSE or the MAE?

Given the presence of outliers in the `SalePrice` data (as evidenced by the boxplot), MAE would be a more appropriate metric. MAE provides a more robust measure of the average prediction error because it's less sensitive to outliers. In the context of house prices, where outliers could represent unique properties or rare market conditions, using MAE will give a more representative measure of model performance on average predictions.

3. What does it mean when a model has better predictive accuracy in-sample than it does out-of-sample?

If a model has better predictive accuracy in-sample than out-of-sample, it may suggest a few things:

- Overfitting: The model may have fit too closely to the quirks or noise in the training data and captured patterns that don't generalize well to new, unseen data.
- Different Data Distributions: The training and test datasets might come from slightly different distributions or periods.
- Model Complexity: Highly complex models might perform exceptionally well on training data but fail to generalize to new data.
- Insufficient Testing Data: If the out-of-sample data is too small or not representative, it might give a skewed performance metric.

In practice, it's quite common for models to perform slightly better in-sample than out-of-sample. The key is to ensure that the drop in performance is not significant, which would suggest potential overfitting. The aim is to create models that generalize well to new, unseen data.

4. Operational Validation

We have validated these models in the statistical sense, but what about the business sense? Do MSE or MAE easily translate to the development of a business policy? Typically, in applications we need to be able to hit defined cut-off points, i.e. we set a policy that we need to be p% accurate. Let's define a variable called PredictionGrade, and consider the predicted value to be 'Grade 1' if it is within ten percent of the actual value, 'Grade 2' if it is not Grade 1 but within fifteen percent of the actual value, Grade 3 if it is not Grade 2 but within twenty-five percent of the actual value, and 'Grade 4' otherwise.

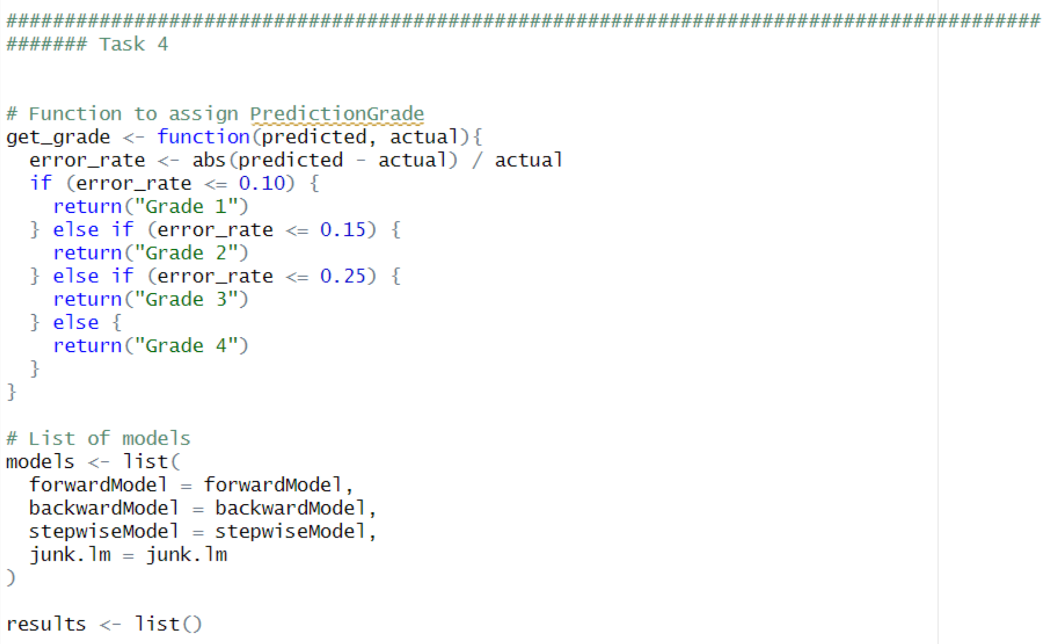
Here is a code snippet to show you how we will produce the prediction grades.

```
# Training Data
# Abs Pct Error
forward.pct <- abs(forward.lm$residuals)/train.clean$SalePrice;

# Assign Prediction Grades;
forward.PredictionGrade <- ifelse(forward.pct<=0.10,'Grade 1: [0.0.10]',
                                 ifelse(forward.pct<=0.15,'Grade 2: (0.10,0.15]',
                                        ifelse(forward.pct<=0.25,'Grade 3: (0.15,0.25]',
                                        'Grade 4: (0.25+]')
                                 )
                         )

forward.trainTable <- table(forward.PredictionGrade)
forward.trainTable/sum(forward.trainTable)


# Test Data
# Abs Pct Error
forward.testPCT <- abs(test.df$SalePrice-forward.test)/test.df$SalePrice;
backward.testPCT <- abs(test.df$SalePrice-backward.test)/test.df$SalePrice;
stepwise.testPCT <- abs(test.df$SalePrice-stepwise.test)/test.df$SalePrice;
junk.testPCT <- abs(test.df$SalePrice-junk.test)/test.df$SalePrice;


# Assign Prediction Grades;
forward.testPredictionGrade <- ifelse(forward.testPCT<=0.10,'Grade 1: [0.0.10]',
                                 ifelse(forward.testPCT<=0.15,'Grade 2: (0.10,0.15]',
                                        ifelse(forward.testPCT<=0.25,'Grade 3: (0.15,0.25]',
                                        'Grade 4: (0.25+]')
                                 )
                         )

forward.testTable <-table(forward.testPredictionGrade)
forward.testTable/sum(forward.testTable)
```

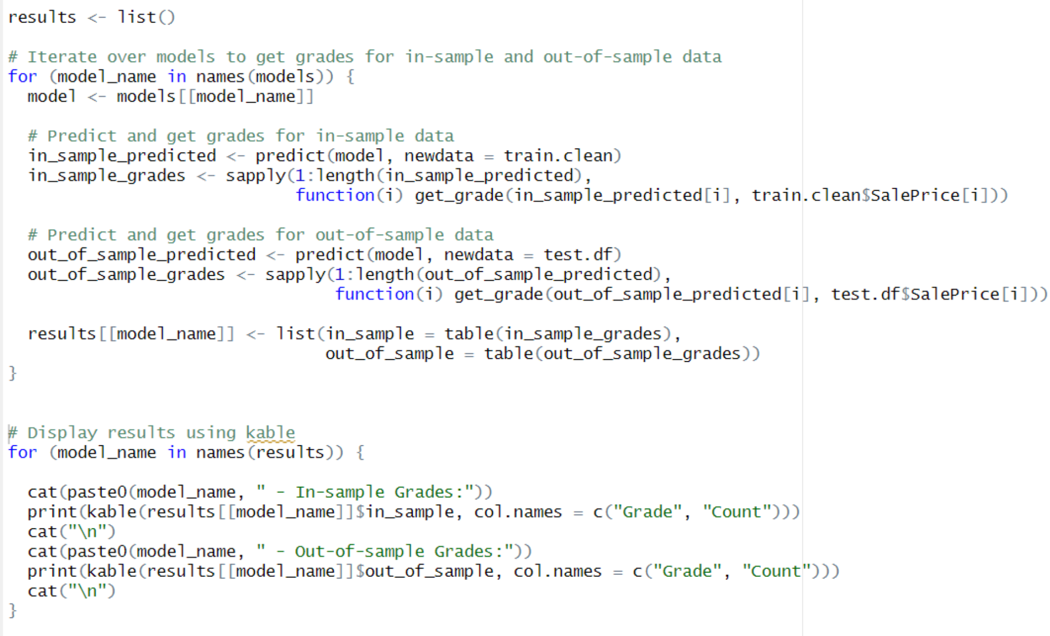
Produce these prediction grades for the in-sample training data and the out-of-sample test data. Note that we want to show these tables in distribution form, not counts. Distribution form is more informative and easier for your reader (and you!) to understand, hence we have normalized the table object.

How accurate are the models under this definition of predictive accuracy? How do these results compare to our predictive accuracy results? Did the model ranking remain the same?

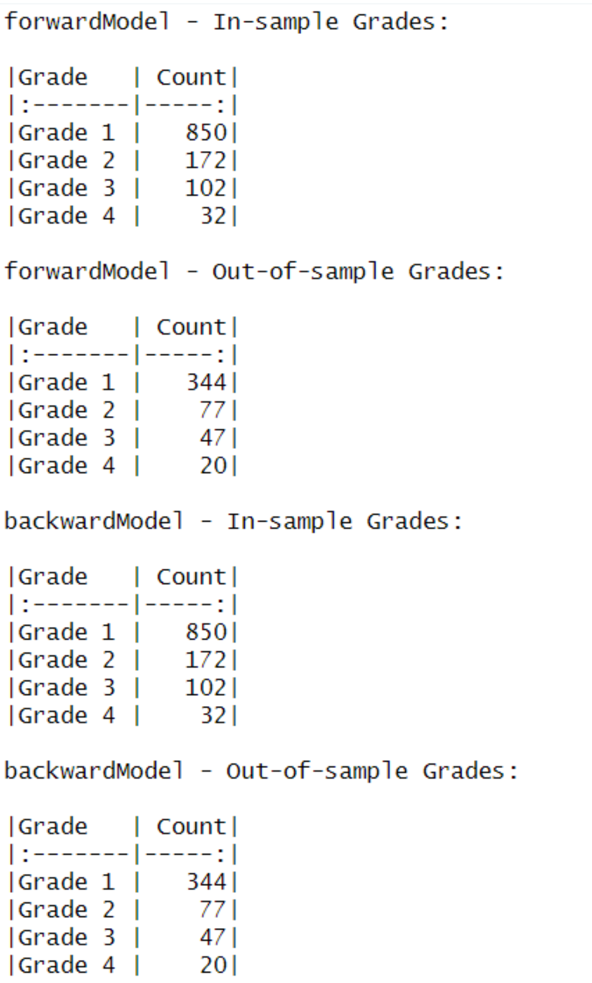
Note: The GSEs (Fannie Mae and Freddie Mac) rate an AVM model as ‘underwriting quality’ if the model is accurate to within ten percent more than fifty percent of the time. Are any of your models ‘underwriting quality’?



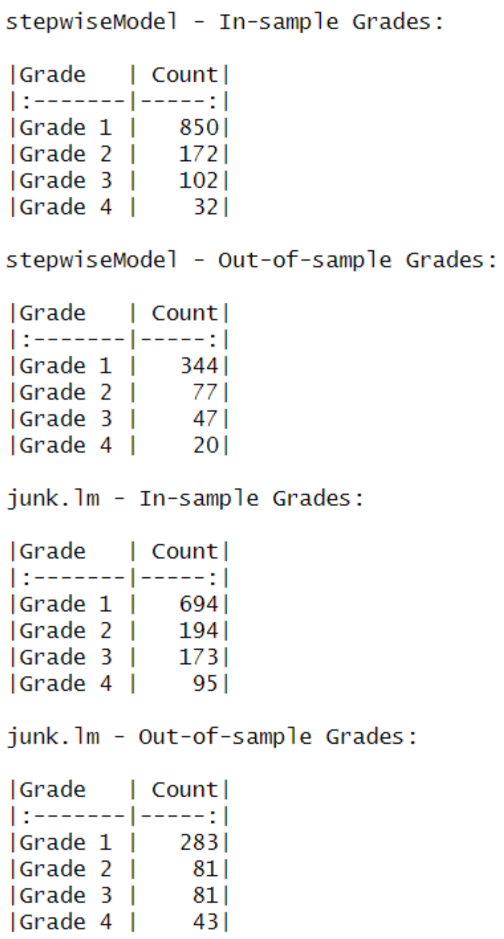
```
###########################################################################################
####### Task 4


# Function to assign PredictionGrade
get_grade <- function(predicted, actual){
  error_rate <- abs(predicted - actual) / actual
  if (error_rate <= 0.10) {
    return("Grade 1")
  } else if (error_rate <= 0.15) {
    return("Grade 2")
  } else if (error_rate <= 0.25) {
    return("Grade 3")
  } else {
    return("Grade 4")
  }
}

# List of models
models <- list(
  forwardModel = forwardModel,
  backwardModel = backwardModel,
  stepwiseModel = stepwiseModel,
  junk.lm = junk.lm
)

results <- list()
```



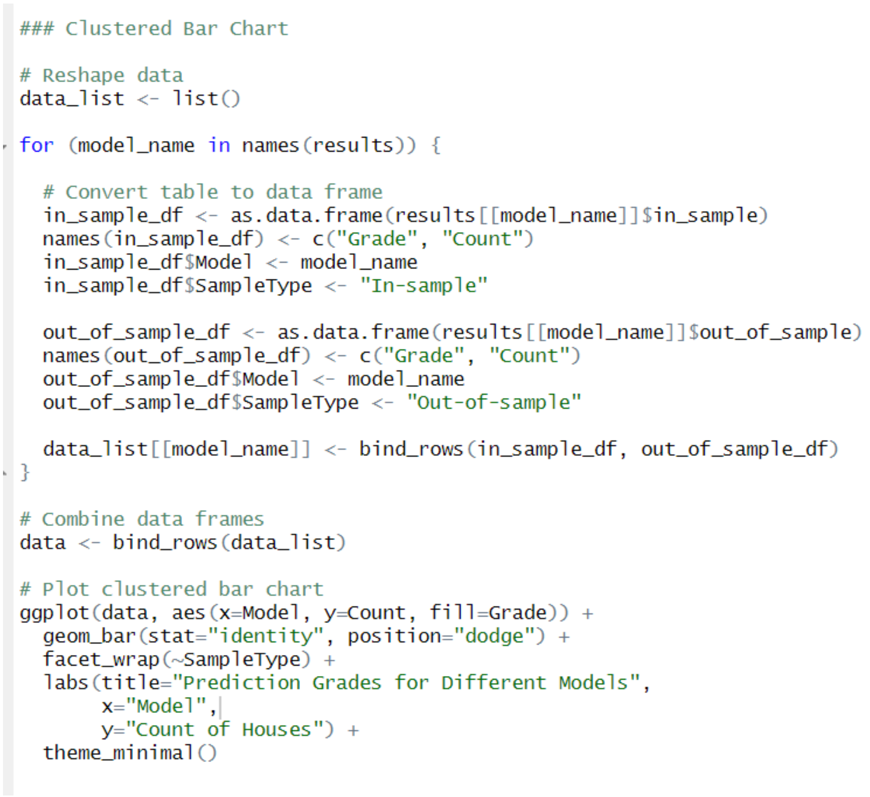
```
results <- list()

# Iterate over models to get grades for in-sample and out-of-sample data
for (model_name in names(models)) {
  model <- models[[model_name]]

  # Predict and get grades for in-sample data
  in_sample_predicted <- predict(model, newdata = train.clean)
  in_sample_grades <- sapply(1:length(in_sample_predicted),
                             function(i) get_grade(in_sample_predicted[i], train.clean$SalePrice[i]))

  # Predict and get grades for out-of-sample data
  out_of_sample_predicted <- predict(model, newdata = test.df)
  out_of_sample_grades <- sapply(1:length(out_of_sample_predicted),
                                 function(i) get_grade(out_of_sample_predicted[i], test.df$SalePrice[i]))

  results[[model_name]] <- list(in_sample = table(in_sample_grades),
                                out_of_sample = table(out_of_sample_grades))
}


# Display results using kable
for (model_name in names(results)) {

  cat(paste0(model_name, " - In-sample Grades:"))
  print(kable(results[[model_name]]$in_sample, col.names = c("Grade", "Count")))
  cat("\n")
  cat(paste0(model_name, " - Out-of-sample Grades:"))
  print(kable(results[[model_name]]$out_of_sample, col.names = c("Grade", "Count")))
  cat("\n")
}
```



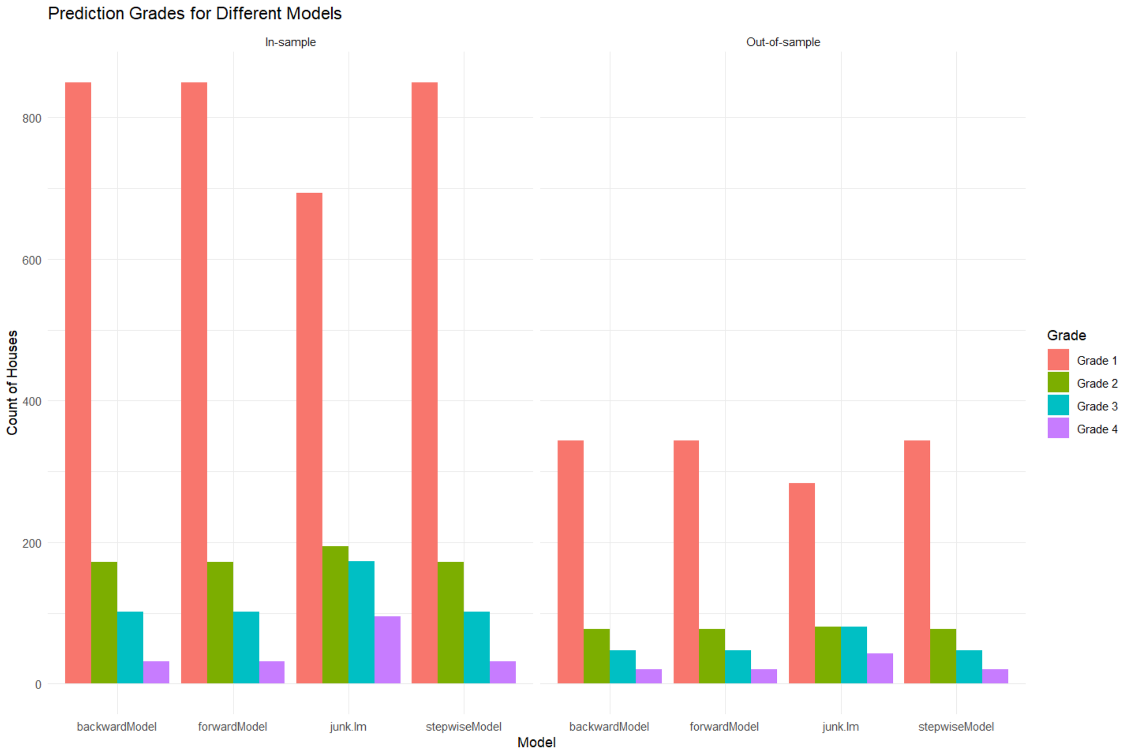
```
forwardModel - In-sample Grades:

|Grade    | Count|
|:-------|-----:|
|Grade 1 |   850|
|Grade 2 |   172|
|Grade 3 |   102|
|Grade 4 |    32|

forwardModel - Out-of-sample Grades:

|Grade    | Count|
|:-------|-----:|
|Grade 1 |   344|
|Grade 2 |    77|
|Grade 3 |    47|
|Grade 4 |    20|

backwardModel - In-sample Grades:

|Grade    | Count|
|:-------|-----:|
|Grade 1 |   850|
|Grade 2 |   172|
|Grade 3 |   102|
|Grade 4 |    32|

backwardModel - Out-of-sample Grades:

|Grade    | Count|
|:-------|-----:|
|Grade 1 |   344|
|Grade 2 |    77|
|Grade 3 |    47|
|Grade 4 |    20|
```



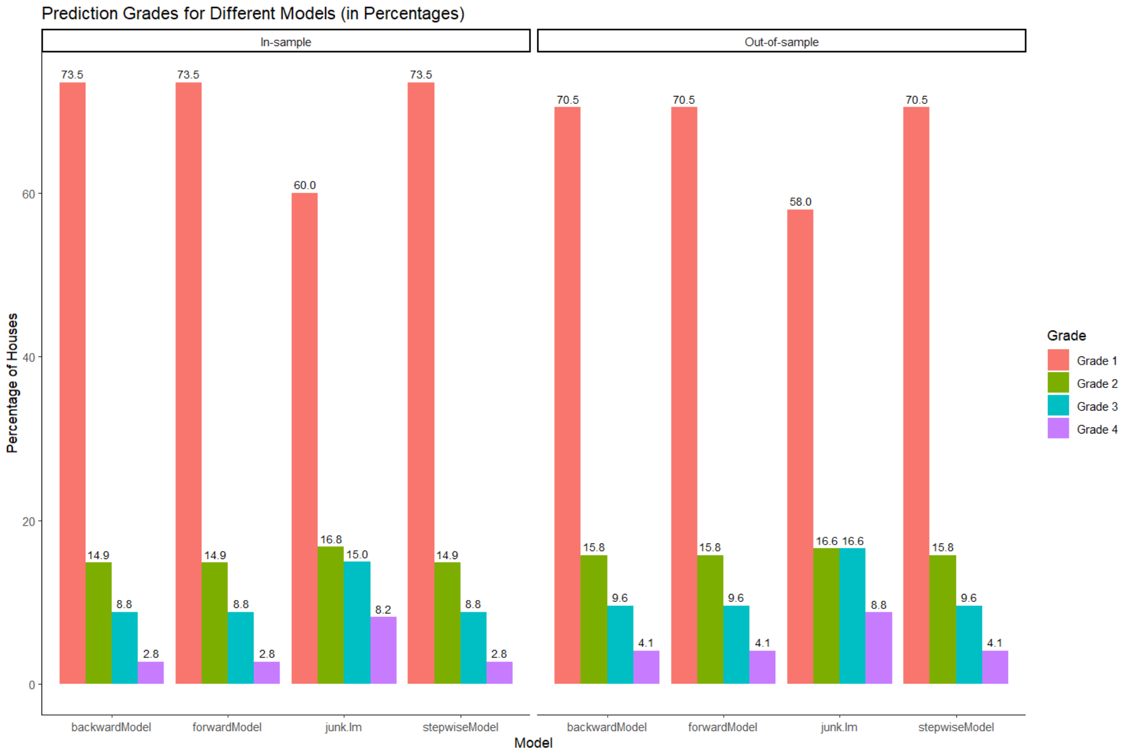
```
stepwiseModel - In-sample Grades:

|Grade    | Count|
|:-------|-----:|
|Grade 1 |   850|
|Grade 2 |   172|
|Grade 3 |   102|
|Grade 4 |    32|

stepwiseModel - Out-of-sample Grades:

|Grade    | Count|
|:-------|-----:|
|Grade 1 |   344|
|Grade 2 |    77|
|Grade 3 |    47|
|Grade 4 |    20|

junk.lm - In-sample Grades:

|Grade    | Count|
|:-------|-----:|
|Grade 1 |   694|
|Grade 2 |   194|
|Grade 3 |   173|
|Grade 4 |    95|

junk.lm - Out-of-sample Grades:

|Grade    | Count|
|:-------|-----:|
|Grade 1 |   283|
|Grade 2 |    81|
|Grade 3 |    81|
|Grade 4 |    43|
```



```r
### Clustered Bar Chart

# Reshape data
data_list <- list()

for (model_name in names(results)) {

  # Convert table to data frame
  in_sample_df <- as.data.frame(results[[model_name]]$in_sample)
  names(in_sample_df) <- c("Grade", "Count")
  in_sample_df$Model <- model_name
  in_sample_df$SampleType <- "In-sample"

  out_of_sample_df <- as.data.frame(results[[model_name]]$out_of_sample)
  names(out_of_sample_df) <- c("Grade", "Count")
  out_of_sample_df$Model <- model_name
  out_of_sample_df$SampleType <- "Out-of-sample"

  data_list[[model_name]] <- bind_rows(in_sample_df, out_of_sample_df)
}

# Combine data frames
data <- bind_rows(data_list)

# Plot clustered bar chart
ggplot(data, aes(x=Model, y=Count, fill=Grade)) +
  geom_bar(stat="identity", position="dodge") +
  facet_wrap(~SampleType) +
  labs(title="Prediction Grades for Different Models",
       x="Model",
       y="Count of Houses") +
  theme_minimal()
```



This clustered-bar chart is plotted for different numbers of datapoints in In-Sample(1156) and Out-of-Sample(488). To get a better perspective, we have plotted the clustered-bar-chart using percentages.



	In-Sample				Out-of-Sample			
	backwardModel	forwardModel	junk.lm	stepwiseModel	backwardModel	forwardModel	junk.lm	stepwiseModel
Grade1	73.5%	73.5%	60.0%	73.5%	70.5%	70.5%	58.0%	70.5%
Grade2	14.9%	14.9%	16.8%	14.9%	15.8%	15.8%	16.6%	15.8%
Grade3	8.8%	8.8%	15.0%	8.8%	9.6%	9.6%	16.6%	9.6%
Grade4	2.8%	2.8%	8.2%	2.8%	4.1%	4.1%	8.8%	4.1%

To rank the models, we'll use a weightage system:

- Grade 1 = 4 points
- Grade 2 = 3 points
- Grade 3 = 2 points
- Grade 4 = 1 point

Let's calculate the weighted scores for each model:

In-Sample Rankings:

1. backwardModel & forwardModel & stepwiseModel:
Weighted Score = (73.5 4) + (14.9 3) + (8.8 2) + (2.8 1)
= 294 + 44.7 + 17.6 + 2.8
= 359.1

2. junk.lm:
Weighted Score = (60 4) + (16.8 3) + (15 2) + (8.2 1)
= 240 + 50.4 + 30 + 8.2
= 328.6

Out-of-Sample Rankings:

1. backwardModel & forwardModel & stepwiseModel:
Weighted Score = (70.5 4) + (15.8 3) + (9.6 2) + (4.1 1)
= 282 + 47.4 + 19.2 + 4.1
= 352.7

2. junk.lm:
Weighted Score = (58 4) + (16.6 3) + (16.6 2) + (8.8 1)
= 232 + 49.8 + 33.2 + 8.8
= 323.8

Final Rankings:

In-Sample:
1. `backwardModel`, `forwardModel`, `stepwiseModel` (tied) with a score of 359.1
2. `junk.lm` with a score of 328.6

Out-of-Sample:
1. `backwardModel`, `forwardModel`, `stepwiseModel` (tied) with a score of 352.7
2. `junk.lm` with a score of 323.8

Summary: For both In-Sample and Out-of-Sample data, the `backwardModel`, `forwardModel`, and `stepwiseModel` perform identically and outperform the `junk.lm` model. While the three models have a tie in terms of their performance score, the `junk.lm` lags behind in both scenarios. The difference in scores between the models suggests that the `backwardModel`, `forwardModel`, and `stepwiseModel` are relatively more reliable in their predictions compared to `junk.lm`.

How accurate are the models under this definition of predictive accuracy? How do these results compare to our predictive accuracy results? Did the model ranking remain the same?

The grading scores and model metrics paint a consistent picture: forwardModel, backwardModel, and stepwiseModel have comparable and superior performance both in terms of grading scores and statistical model metrics.

junk.lm, on the other hand, underperforms in both scenarios.

In summary, the grading scores align well with the model metrics. Both methods of evaluation suggest that forwardModel, backwardModel, and stepwiseModel are more reliable and accurate in their predictions, while junk.lm is less so.

The GSEs (Fannie Mae and Freddie Mac) rate an AVM model as ‘underwriting quality’ if the model is accurate to within ten percent more than fifty percent of the time. Are any of your models ‘underwriting quality’?

backwardModel, forwardModel, and stepwiseModel all qualify as 'underwriting quality' for both in-sample and out-of-sample data, as they are accurate to within ten percent more than 70% of the time.

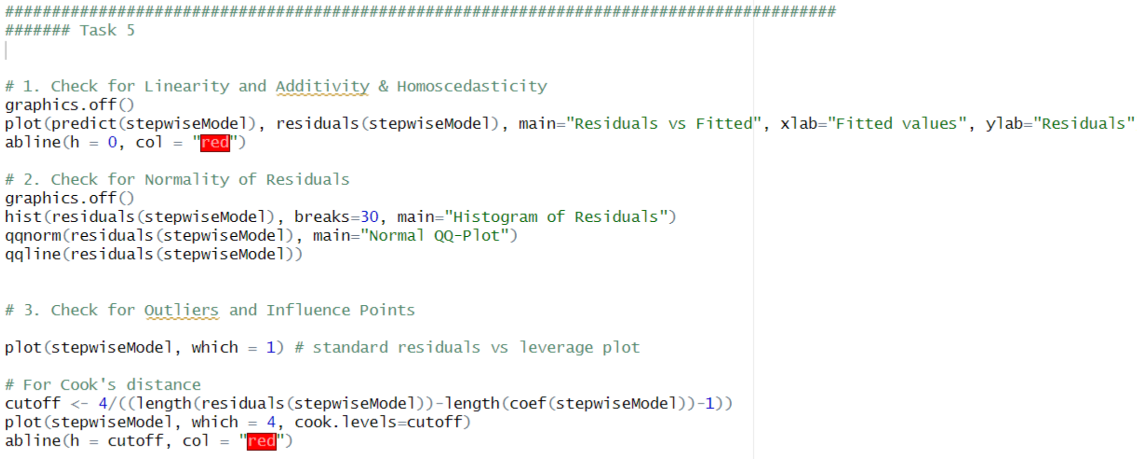
junk.lm also qualifies as 'underwriting quality' for in-sample data (60% accuracy) but is not too far from the 50% cut-off for out-of-sample data (58% accuracy).

In conclusion, backwardModel, forwardModel, and stepwiseModel are 'underwriting quality', and junk.lm could be considered as such for in-sample data but is borderline for out-of-sample data.

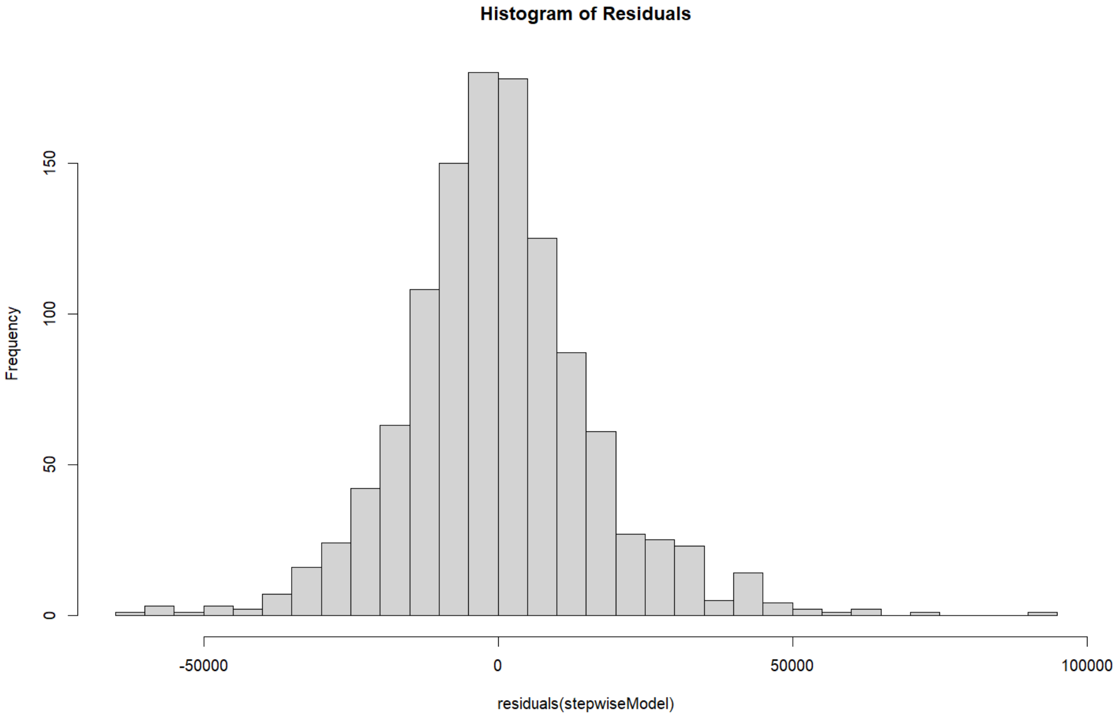
Task 5

For which ever model you find to be “Best” after the automated variable selection procedures and all of these comparisons, you will need to re-visit that model and clean it up, as well as conduct residual diagnostics. Frankly, the end of an automated variable selection process is in many ways a starting point. What kinds of things do you want to check for and “clean up”?

Since backwardModel, forwardModel, and stepwiseModel have returned similar metrics, we would go ahead with stepwiseModel to complete this task.

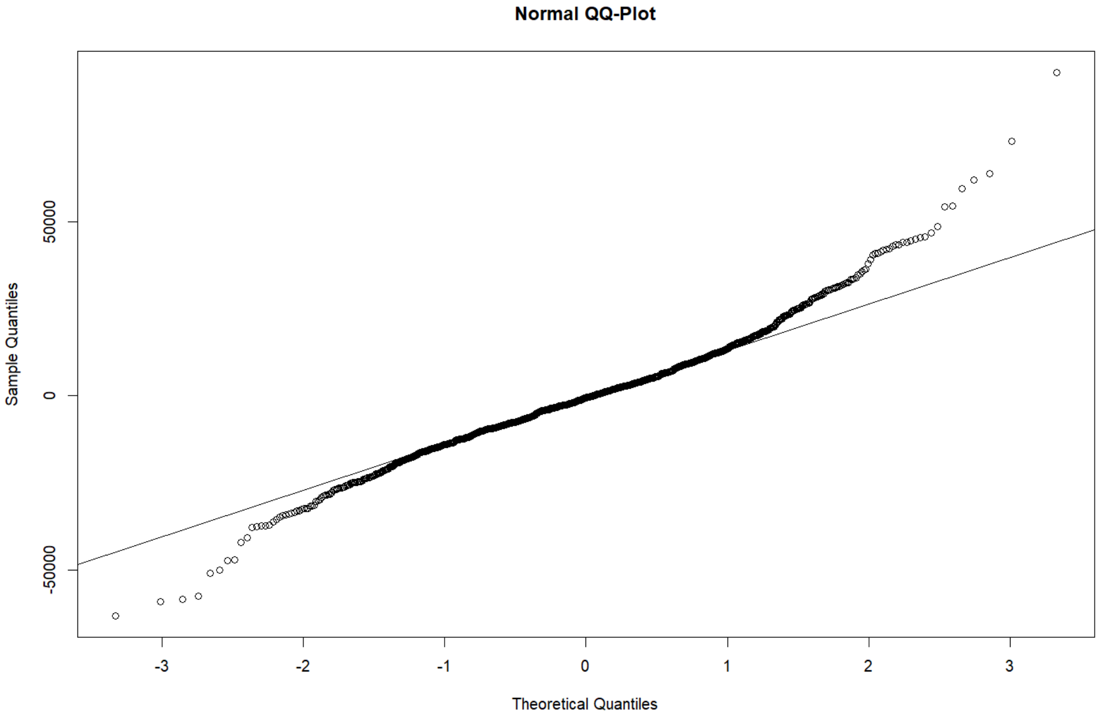


```
#########################################################################################
####### Task 5


# 1. Check for Linearity and Additivity & Homoscedasticity
graphics.off()
plot(predict(stepwiseModel), residuals(stepwiseModel), main="Residuals vs Fitted", xlab="Fitted values", ylab="Residuals"
abline(h = 0, col = "red")

# 2. Check for Normality of Residuals
graphics.off()
hist(residuals(stepwiseModel), breaks=30, main="Histogram of Residuals")
qqnorm(residuals(stepwiseModel), main="Normal QQ-Plot")
qqline(residuals(stepwiseModel))


# 3. Check for Outliers and Influence Points

plot(stepwiseModel, which = 1) # standard residuals vs leverage plot

# For Cook's distance
cutoff <- 4/((length(residuals(stepwiseModel))-length(coef(stepwiseModel))-1))
plot(stepwiseModel, which = 4, cook.levels=cutoff)
abline(h = cutoff, col = "red")
```



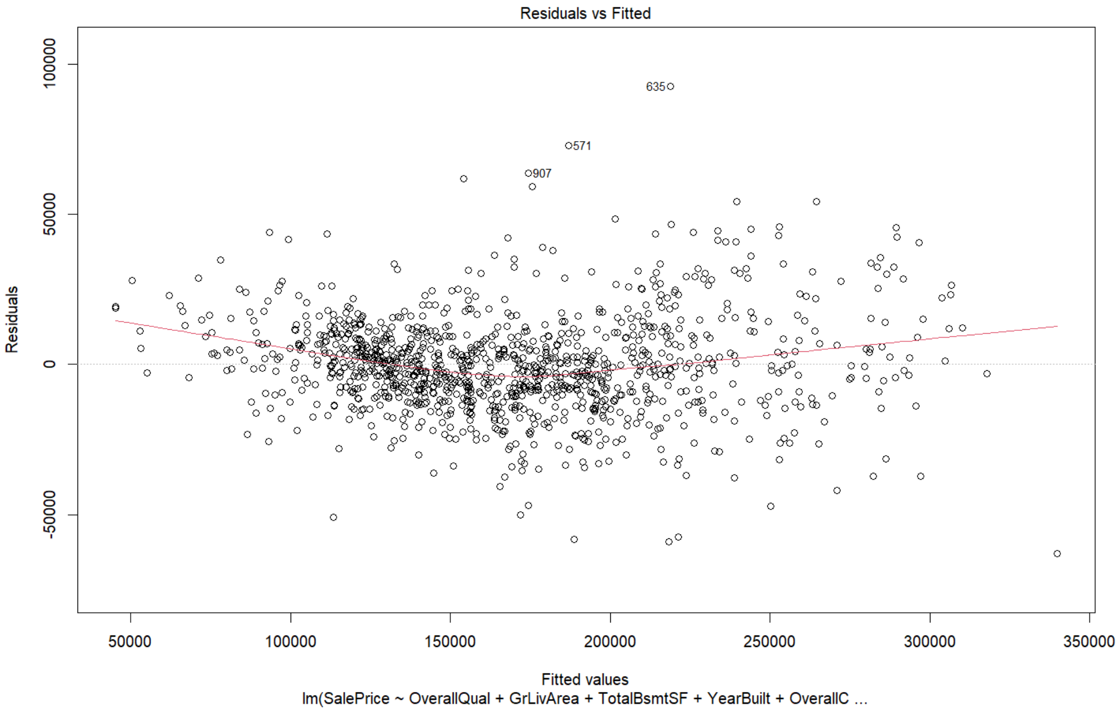
Histogram of Residuals:

- The residuals appear to be centred around zero, which is a good sign.
- The distribution of the residuals seems to be slightly right-skewed. The tail on the right side of the histogram suggests that there are some observations that have higher residuals than expected if the residuals were perfectly normally distributed.



Normal QQ-Plot:

The points in the QQ-plot generally follow the straight line, especially in the middle part, which indicates that the residuals are approximately normally distributed. However, the departure from the line at the top right of the plot indicates that there are some larger residuals that are not explained by a normal distribution. This observation confirms the right-skewness noted in the histogram.



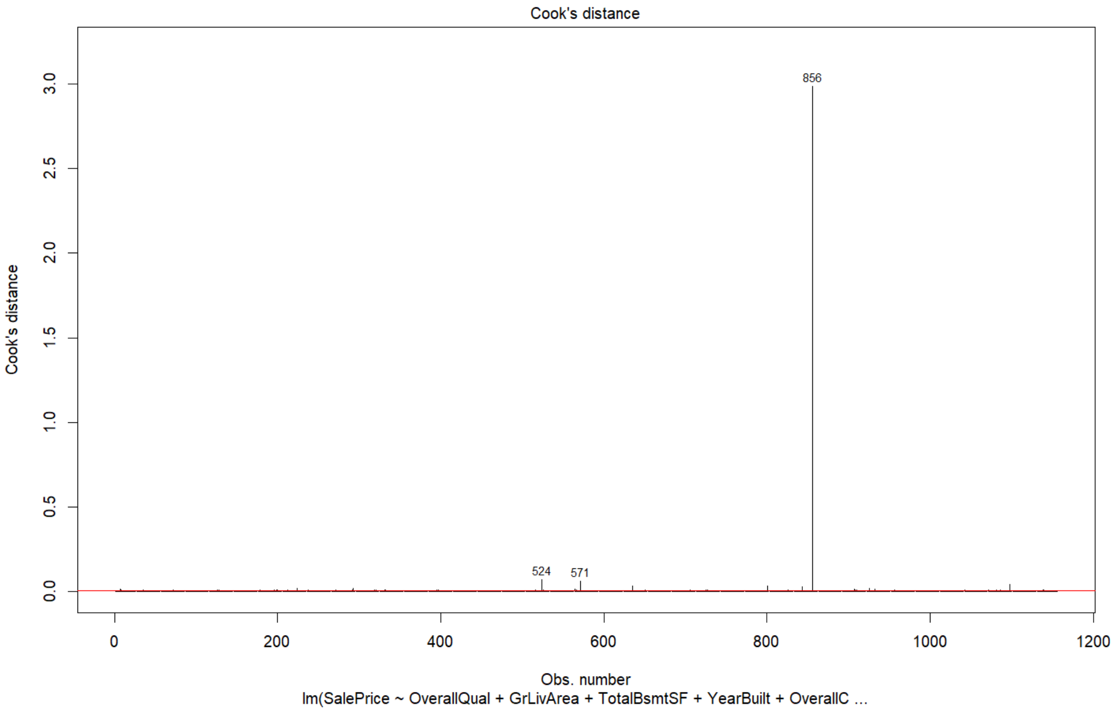
Residuals vs Fitted:

Linearity: The presence of a horizontal red line (usually a loess curve) suggests that on average, the residuals are centered around zero across different levels of the fitted values. If this line were to show a clear pattern (e.g., U-shape or inverted U-shape), it would indicate potential non-linearity in the relationship between predictors and the response variable. In this plot, the red line is relatively straight, suggesting no major issues with linearity.

Homoscedasticity: This assumption means that the variances of the residuals are constant across levels of the independent variables. This plot seems to have a relatively consistent spread of residuals across the range of fitted values, which is a good sign. However, there are a few areas, especially towards the higher end of fitted values, where the spread seems slightly wider.

Outliers: The individual points on the graph represent residuals for each observation. Points that are far from the central cloud might be considered outliers. In the plot, there are several points that are quite far from the main cluster, indicating potential outliers in the data.

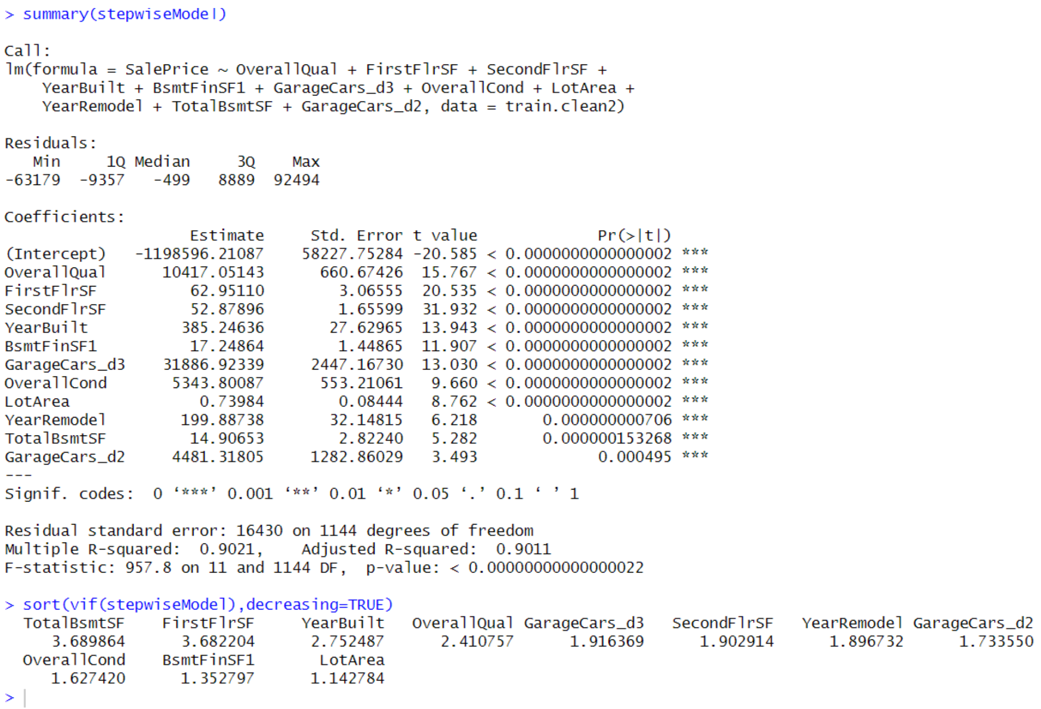
Leverage Points: The numbers on the plot (like 635, 571, 907) are likely identifiers for specific observations. Points with high leverage have the potential to influence the regression line considerably. While these points might not be outliers in the Y dimension (residuals), they might be extreme values in the X dimension and can unduly influence the model.



Cook’s Distance: Most of the observations have a Cook's distance close to 0, suggesting they are not influential. However, there's a very noticeable spike for observation 856. This observation has a much higher Cook's distance than the others, indicating that it is potentially an influential point. Other observations labelled (e.g., 524 and 571) also seem to have higher values than the majority, but they are not as extreme as 856.

Observation 856 is potentially an influential point in this regression analysis, as it has a notably high Cook's distance. One may want to investigate this observation further to understand why it's so influential. It could be an outlier, a data entry error, or it might represent a unique situation that needs to be considered separately. Removing or adjusting influential points can sometimes lead to a better-fitting and more interpretable model, but any such decisions should be made carefully, understanding the context and the implications.

Quantitative variables may have been selected that logically have their coefficients reversed from what it theoretically should be. For example, a final model can have a negative coefficient relating size of home in square feet to price. That wouldn’t make sense. Why would that variable be included in the model – or there is something else going on, like multicollinearity that needs to be accounted for. That has to be fixed some way. Always remember, an easy solution for multicollinearity is to simply leave an offending variable out of the model.



```
> summary(stepwiseModel)

Call:
lm(formula = SalePrice ~ OverallQual + FirstFlrSF + SecondFlrSF + 
    YearBuilt + BsmtFinSF1 + GarageCars_d3 + OverallCond + LotArea + 
    YearRemodel + TotalBsmtSF + GarageCars_d2, data = train.clean2)

Residuals:
   Min     1Q Median     3Q    Max 
-63179  -9357   -499   8889  92494 

Coefficients:
                   Estimate     Std. Error t value              Pr(>|t|)    
(Intercept)   -1198596.21087    58227.75284 -20.585 < 0.0000000000000002 ***
OverallQual      10417.05143      660.67426  15.767 < 0.0000000000000002 ***
FirstFlrSF          62.95110        3.06555  20.535 < 0.0000000000000002 ***
SecondFlrSF         52.87896        1.65599  31.932 < 0.0000000000000002 ***
YearBuilt          385.24636       27.62965  13.943 < 0.0000000000000002 ***
BsmtFinSF1          17.24864        1.44865  11.907 < 0.0000000000000002 ***
GarageCars_d3    31886.92339     2447.16730  13.030 < 0.0000000000000002 ***
OverallCond       5343.80087      553.21061   9.660 < 0.0000000000000002 ***
LotArea              0.73984        0.08444   8.762 < 0.0000000000000002 ***
YearRemodel        199.88738       32.14815   6.218        0.000000000706 ***
TotalBsmtSF         14.90653        2.82240   5.282        0.000000153268 ***
GarageCars_d2     4481.31805     1282.86029   3.493              0.000495 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 16430 on 1144 degrees of freedom
Multiple R-squared:  0.9021,	Adjusted R-squared:  0.9011 
F-statistic: 957.8 on 11 and 1144 DF,  p-value: < 0.00000000000000022

> sort(vif(stepwiseModel),decreasing=TRUE)
  TotalBsmtSF    FirstFlrSF     YearBuilt   OverallQual GarageCars_d3   SecondFlrSF   YearRemodel GarageCars_d2 
     3.689864      3.682204      2.752487      2.410757      1.916369      1.902914      1.896732      1.733550 
  OverallCond    BsmtFinSF1       LotArea 
     1.627420      1.352797      1.142784 
> 
```

In this model, only Intercept has a negative coefficient (-1198596.21087). The intercept in a regression model represents the expected value of the dependent variable (in this case, SalePrice) when all predictor variables are set to zero. However, in many real-world scenarios, especially with complex models, this hypothetical situation might not be practically meaningful or even possible. A house with 0 square footage, 0 rooms, and built in year 0 does not exist.

As models become more complex with more variables, the mathematical relationships can lead to an intercept that may not be directly interpretable in real-world terms. It's more about how the entire equation (intercept + coefficients) models the data rather than the standalone value of the intercept.

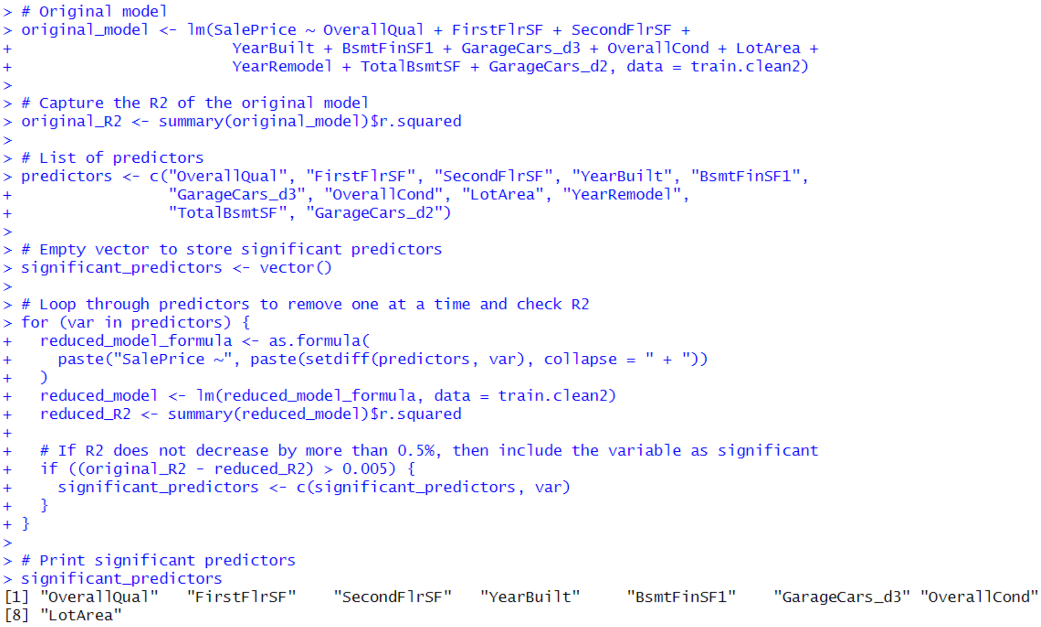
It is essential to remember the role of the intercept in multiple regression models. It serves as an adjustment factor that works in tandem with the other coefficients to provide accurate predictions for data points within the scope of the data. In some cases, especially when none of the predictor variables are close to zero, the intercept can be less interpretable.

The VIF values are indicative of multicollinearity. A VIF value greater than 10 is typically considered a sign of high multicollinearity, though some practitioners might use a lower threshold, such as 5. In this model, none of the variables exceed the stringent threshold of 10.

TotalBsmtSF and FirstFlrSF have the highest VIF values, nearing 3.7, indicating they might have some degree of multicollinearity but not at extreme levels. The other variables have VIF values below this, suggesting that they don't introduce severe multicollinearity.

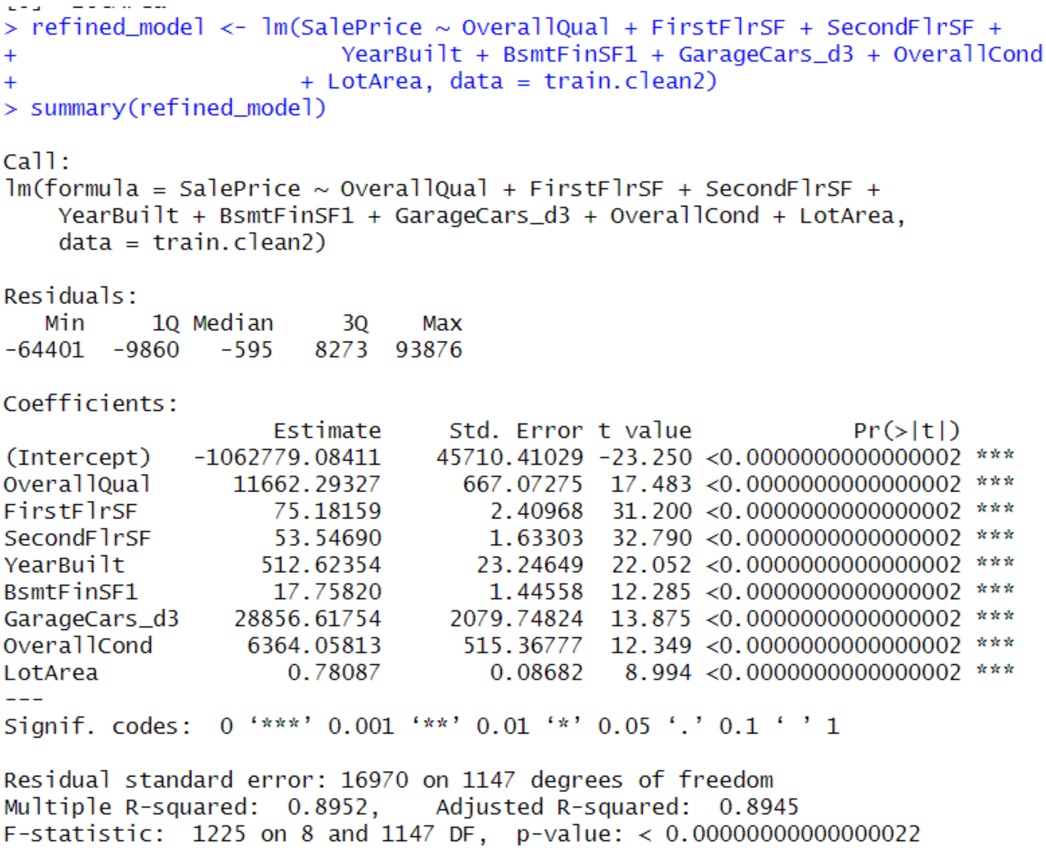
Quantitative variables may be included in the model that are statistically significant, but are not actually predictive. This is mostly an issue when the sample size is large. The issue is large sample size translates into high statistical power. If too much power is present, everything can be statistically significant. Remember, statistical significance does not mean important, it means ruling out chance as the explanation. To guard against an overfit model with too many variables, consider looking at R-squared change. Examine the impact of removing each variable from the final model, one at a time. If R-squared change for any of the retained variables is too small – why include that variable in the model? It is not contributing to the predictive ability of the model. Think about it! Parsimony is important – simpler models are easier to explain and tend to be better in the long run out of sample! Do you really need a max fit model, or a best explanation model? You are welcome to remove variables from the “final model” until all contribute sufficiently well.

We cycle through each variable in the stepwiseModel, temporarily excluding one at a time. After each exclusion, if the R2 value drops by less than 0.5%, that variable is not incorporated into the subsequent model iteration. By the loop's conclusion, we're left with a refined list of variables that substantially influence the accuracy of the stepwiseModel.



```
> # Original model
> original_model <- lm(SalePrice ~ OverallQual + FirstFlrSF + SecondFlrSF +
+                        YearBuilt + BsmtFinSF1 + GarageCars_d3 + OverallCond + LotArea +
+                        YearRemodel + TotalBsmtSF + GarageCars_d2, data = train.clean2)
> 
> # Capture the R2 of the original model
> original_R2 <- summary(original_model)$r.squared
> 
> # List of predictors
> predictors <- c("OverallQual", "FirstFlrSF", "SecondFlrSF", "YearBuilt", "BsmtFinSF1",
+                 "GarageCars_d3", "OverallCond", "LotArea", "YearRemodel",
+                 "TotalBsmtSF", "GarageCars_d2")
> 
> # Empty vector to store significant predictors
> significant_predictors <- vector()
> 
> # Loop through predictors to remove one at a time and check R2
> for (var in predictors) {
+   reduced_model_formula <- as.formula(
+     paste("SalePrice ~", paste(setdiff(predictors, var), collapse = " + "))
+   )
+   reduced_model <- lm(reduced_model_formula, data = train.clean2)
+   reduced_R2 <- summary(reduced_model)$r.squared
+   
+   # If R2 does not decrease by more than 0.5%, then include the variable as significant
+   if ((original_R2 - reduced_R2) > 0.005) {
+     significant_predictors <- c(significant_predictors, var)
+   }
+ }
> 
> # Print significant predictors
> significant_predictors
[1] "OverallQual"   "FirstFlrSF"    "SecondFlrSF"   "YearBuilt"     "BsmtFinSF1"    "GarageCars_d3" "OverallCond"  
[8] "LotArea"
```

We create refined_model, only with significant variables, the and check for R square:

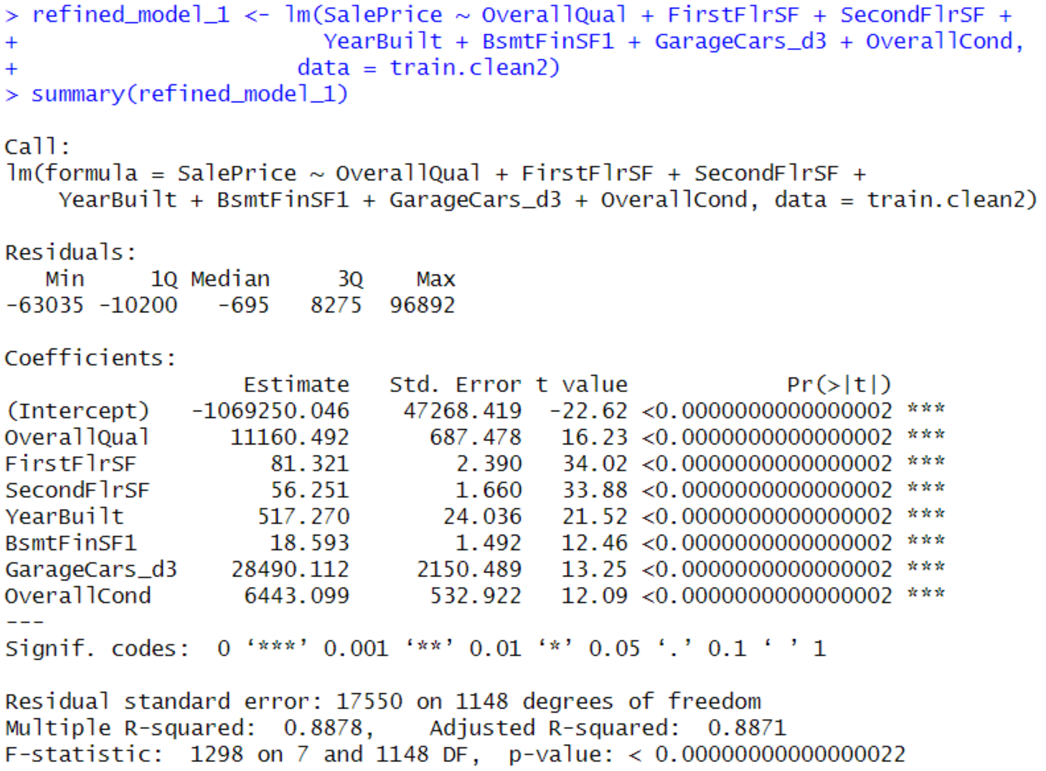


```
> refined_model <- lm(SalePrice ~ OverallQual + FirstFlrSF + SecondFlrSF +
+                       YearBuilt + BsmtFinSF1 + GarageCars_d3 + OverallCond
+                     + LotArea, data = train.clean2)
> summary(refined_model)

Call:
lm(formula = SalePrice ~ OverallQual + FirstFlrSF + SecondFlrSF + 
    YearBuilt + BsmtFinSF1 + GarageCars_d3 + OverallCond + LotArea, 
    data = train.clean2)

Residuals:
   Min     1Q Median     3Q    Max 
-64401  -9860   -595   8273  93876 

Coefficients:
                   Estimate     Std. Error t value            Pr(>|t|)    
(Intercept)   -1062779.08411    45710.41029 -23.250 <0.0000000000000002 ***
OverallQual      11662.29327      667.07275  17.483 <0.0000000000000002 ***
FirstFlrSF          75.18159        2.40968  31.200 <0.0000000000000002 ***
SecondFlrSF         53.54690        1.63303  32.790 <0.0000000000000002 ***
YearBuilt          512.62354       23.24649  22.052 <0.0000000000000002 ***
BsmtFinSF1          17.75820        1.44558  12.285 <0.0000000000000002 ***
GarageCars_d3    28856.61754     2079.74824  13.875 <0.0000000000000002 ***
OverallCond       6364.05813      515.36777  12.349 <0.0000000000000002 ***
LotArea              0.78087        0.08682   8.994 <0.0000000000000002 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 16970 on 1147 degrees of freedom
Multiple R-squared:  0.8952,	Adjusted R-squared:  0.8945 
F-statistic:  1225 on 8 and 1147 DF,  p-value: < 0.00000000000000022
```

R-square has decreased from 0.9021 to 0.8952, which is not significant, given the fact that the number of variables has decreased from 11 to 8.

Variables included in the final model may have coefficients that are essentially zero. These need to be checked to see if those variables are predictive. Use R-squared change to determine if the variable should be retained.

Coefficient of LotArea < 1, we remove this from the model, and create refined_model_1:

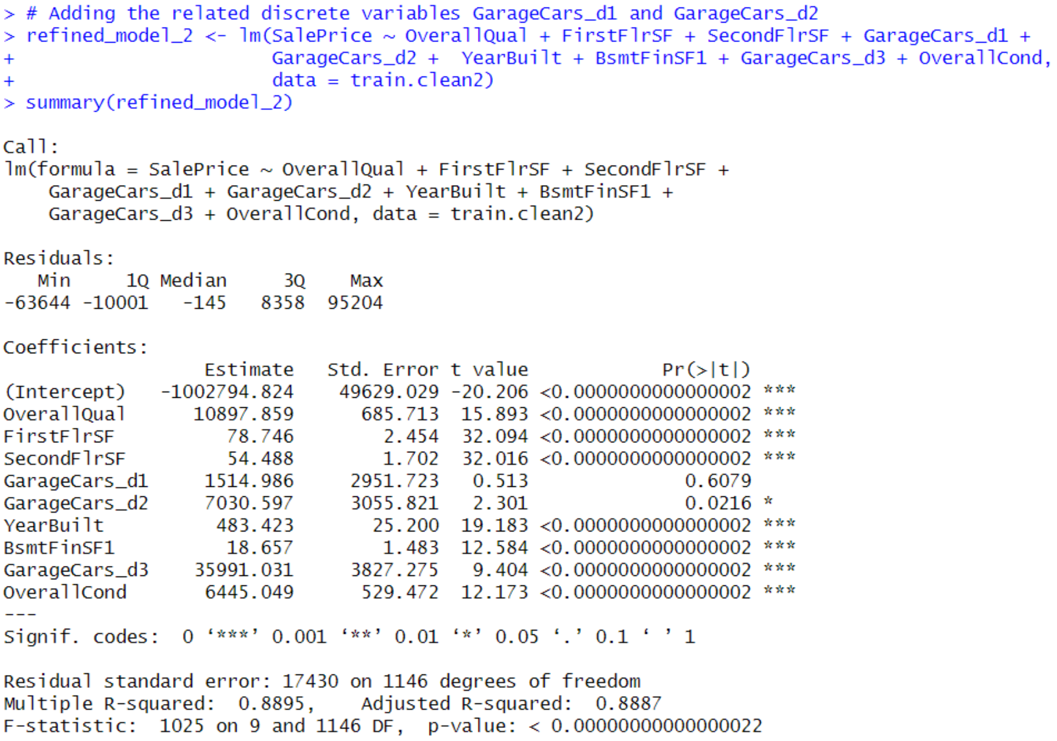


```
> refined_model_1 <- lm(SalePrice ~ OverallQual + FirstFlrSF + SecondFlrSF +
+                       YearBuilt + BsmtFinSF1 + GarageCars_d3 + OverallCond,
+                     data = train.clean2)
> summary(refined_model_1)

Call:
lm(formula = SalePrice ~ OverallQual + FirstFlrSF + SecondFlrSF + 
    YearBuilt + BsmtFinSF1 + GarageCars_d3 + OverallCond, data = train.clean2)

Residuals:
   Min     1Q Median     3Q    Max 
-63035 -10200   -695   8275  96892 

Coefficients:
                 Estimate   Std. Error t value            Pr(>|t|)    
(Intercept)  -1069250.046    47268.419  -22.62 <0.0000000000000002 ***
OverallQual     11160.492      687.478   16.23 <0.0000000000000002 ***
FirstFlrSF         81.321        2.390   34.02 <0.0000000000000002 ***
SecondFlrSF        56.251        1.660   33.88 <0.0000000000000002 ***
YearBuilt         517.270       24.036   21.52 <0.0000000000000002 ***
BsmtFinSF1         18.593        1.492   12.46 <0.0000000000000002 ***
GarageCars_d3   28490.112     2150.489   13.25 <0.0000000000000002 ***
OverallCond      6443.099      532.922   12.09 <0.0000000000000002 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 17550 on 1148 degrees of freedom
Multiple R-squared:  0.8878,	Adjusted R-squared:  0.8871 
F-statistic:  1298 on 7 and 1148 DF,  p-value: < 0.00000000000000022
```

We now have a model (refined_model_1) with R-square value of 0.8878, but with only 7 variables, which seems to be so much better than the model (stepwiseModel) where we had a R-square value of 0.9021 but with 11 variables.

A dummy coded variable may have been selected using the automated procedure. If this has happened, you will want to include all but one of the dummy coded variables for the associated categorical variable in the model. It does not matter whether the dummy coded variables are statistically significant or not. The purpose is for interpretation of coefficients and the inclusion of the entire categorical variable in the predictive model. Think of it as variables are used – all or nothing, not just bits and pieces.

We add GarageCars_d1, GarageCars_d2 to the refined_model_1, to create refined_model_2.



```
> # Adding the related discrete variables GarageCars_d1 and GarageCars_d2
> refined_model_2 <- lm(SalePrice ~ OverallQual + FirstFlrSF + SecondFlrSF + GarageCars_d1 +
+                       GarageCars_d2 +  YearBuilt + BsmtFinSF1 + GarageCars_d3 + OverallCond,
+                       data = train.clean2)
> summary(refined_model_2)

Call:
lm(formula = SalePrice ~ OverallQual + FirstFlrSF + SecondFlrSF + 
    GarageCars_d1 + GarageCars_d2 + YearBuilt + BsmtFinSF1 + 
    GarageCars_d3 + OverallCond, data = train.clean2)

Residuals:
   Min     1Q Median     3Q    Max 
-63644 -10001   -145   8358  95204 

Coefficients:
                 Estimate  Std. Error t value            Pr(>|t|)    
(Intercept)  -1002794.824   49629.029 -20.206 <0.0000000000000002 ***
OverallQual     10897.859     685.713  15.893 <0.0000000000000002 ***
FirstFlrSF         78.746       2.454  32.094 <0.0000000000000002 ***
SecondFlrSF        54.488       1.702  32.016 <0.0000000000000002 ***
GarageCars_d1    1514.986    2951.723   0.513              0.6079    
GarageCars_d2    7030.597    3055.821   2.301              0.0216 *  
YearBuilt         483.423      25.200  19.183 <0.0000000000000002 ***
BsmtFinSF1         18.657       1.483  12.584 <0.0000000000000002 ***
GarageCars_d3   35991.031    3827.275   9.404 <0.0000000000000002 ***
OverallCond      6445.049     529.472  12.173 <0.0000000000000002 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 17430 on 1146 degrees of freedom
Multiple R-squared:  0.8895,	Adjusted R-squared:  0.8887 
F-statistic:  1025 on 9 and 1146 DF,  p-value: < 0.00000000000000022
```

1. Added Variables:

- `GarageCars_d1`: Introduced into refined_model_2 with an estimate of 1514.986. However, its significance level suggests it may not be a strong predictor (p-value: 0.6079).
- `GarageCars_d2`: Added to refined_model_2 with an estimate of 7030.597 and a p-value of 0.0216, indicating some significance.

2. Removed Variables:

- None. All variables present in refined_model_1 remain in refined_model_2.

3. Modified Coefficients:

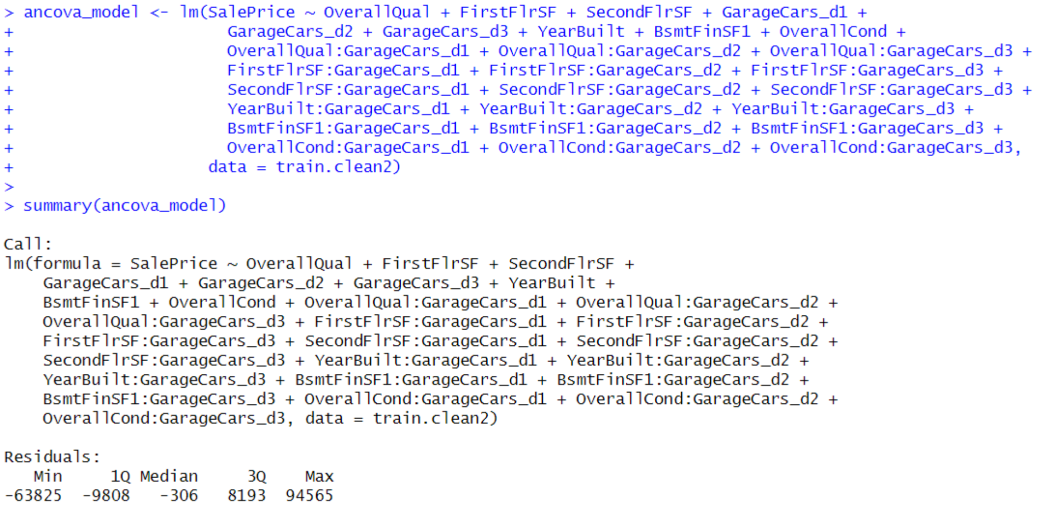
- `OverallQual`: Adjusted from 11160.492 to 10897.859.
- `FirstFlrSF`: Adjusted from 81.321 to 78.746.
- `SecondFlrSF`: Adjusted from 56.251 to 54.488.
- `YearBuilt`: Adjusted from 517.270 to 483.423.
- `BsmtFinSF1`: Very slight adjustment from 18.593 to 18.657.
- `GarageCars_d3`: Adjusted from 28490.112 to 35991.031.
- `OverallCond`: Very slight adjustment from 6443.099 to 6445.049.

4. Model Statistics:

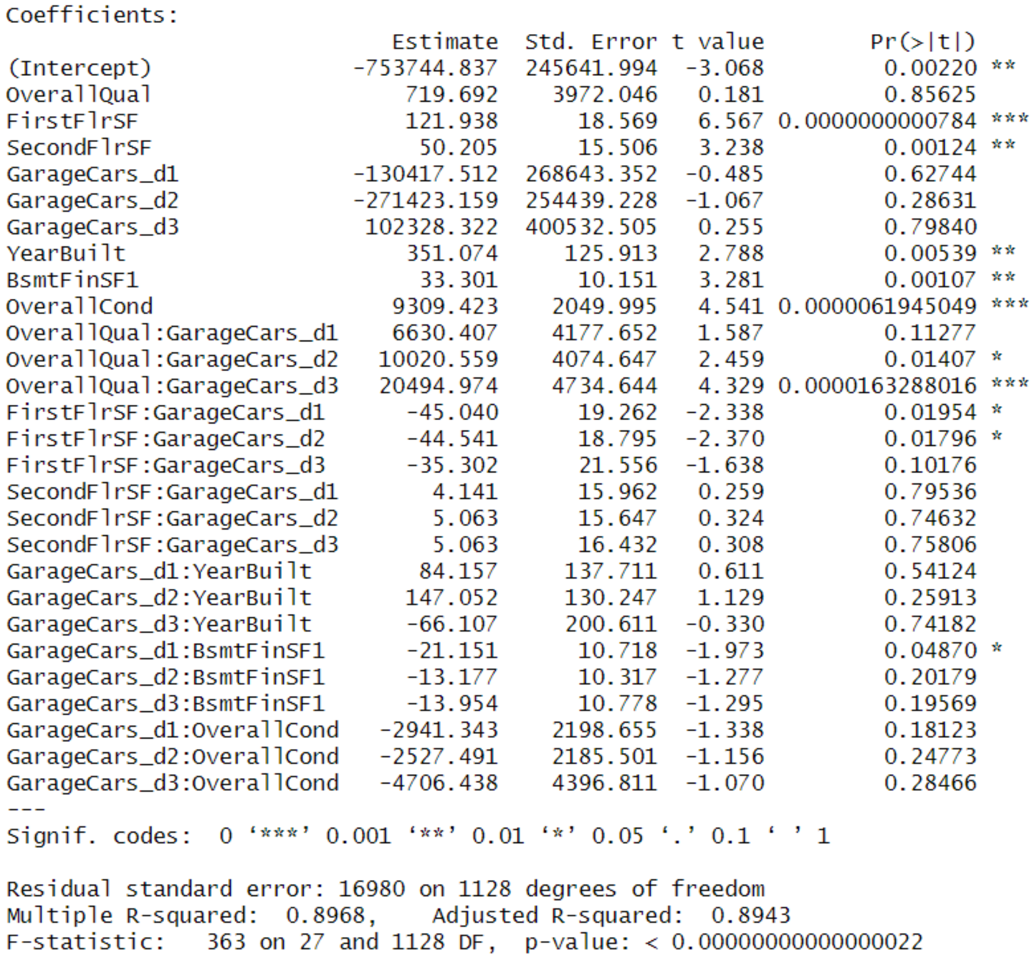
- R-squared: Increased slightly from 0.8878 in refined_model_1 to 0.8895 in refined_model_2.
- Adjusted R-squared: Also increased slightly from 0.8871 to 0.8887.
- Number of Variables: Increased from 7 in refined_model_1 to 9 in refined_model_2.

In summary, refined_model_2 expanded on refined_model_1 by incorporating two more variables, `GarageCars_d1` and `GarageCars_d2`, and slightly adjusting the coefficients of the existing variables. The R-squared value improved marginally, suggesting a slight increase in the explanatory power of the model.

If you retain one or more categorical variables in your final model, you have an ANCOVA model. This means you are modeling Y with parallel planes. You should then be concerned about unequal slopes for these planes. Of concern is the interaction between the categorical variable and any of the quantitative variables. This is a challenging issue if you have a large number of quantitative explanatory variables in your final model. If interaction between the categorical variable and any of the quantitative variables is a logical possibility, you'll need to test for unequal slopes.



```
> ancova_model <- lm(SalePrice ~ OverallQual + FirstFlrSF + SecondFlrSF + GarageCars_d1 +
+                      GarageCars_d2 + GarageCars_d3 + YearBuilt + BsmtFinSF1 + OverallCond +
+                      OverallQual:GarageCars_d1 + OverallQual:GarageCars_d2 + OverallQual:GarageCars_d3 +
+                      FirstFlrSF:GarageCars_d1 + FirstFlrSF:GarageCars_d2 + FirstFlrSF:GarageCars_d3 +
+                      SecondFlrSF:GarageCars_d1 + SecondFlrSF:GarageCars_d2 + SecondFlrSF:GarageCars_d3 +
+                      YearBuilt:GarageCars_d1 + YearBuilt:GarageCars_d2 + YearBuilt:GarageCars_d3 +
+                      BsmtFinSF1:GarageCars_d1 + BsmtFinSF1:GarageCars_d2 + BsmtFinSF1:GarageCars_d3 +
+                      OverallCond:GarageCars_d1 + OverallCond:GarageCars_d2 + OverallCond:GarageCars_d3,
+                    data = train.clean2)
> 
> summary(ancova_model)

Call:
lm(formula = SalePrice ~ OverallQual + FirstFlrSF + SecondFlrSF + 
    GarageCars_d1 + GarageCars_d2 + GarageCars_d3 + YearBuilt + 
    BsmtFinSF1 + OverallCond + OverallQual:GarageCars_d1 + OverallQual:GarageCars_d2 + 
    OverallQual:GarageCars_d3 + FirstFlrSF:GarageCars_d1 + FirstFlrSF:GarageCars_d2 + 
    FirstFlrSF:GarageCars_d3 + SecondFlrSF:GarageCars_d1 + SecondFlrSF:GarageCars_d2 + 
    SecondFlrSF:GarageCars_d3 + YearBuilt:GarageCars_d1 + YearBuilt:GarageCars_d2 + 
    YearBuilt:GarageCars_d3 + BsmtFinSF1:GarageCars_d1 + BsmtFinSF1:GarageCars_d2 + 
    BsmtFinSF1:GarageCars_d3 + OverallCond:GarageCars_d1 + OverallCond:GarageCars_d2 + 
    OverallCond:GarageCars_d3, data = train.clean2)

Residuals:
   Min     1Q Median     3Q    Max 
-63825  -9808   -306   8193  94565 
```



```
Coefficients:
                              Estimate  Std. Error t value        Pr(>|t|)    
(Intercept)                -753744.837  245641.994  -3.068         0.00220 ** 
OverallQual                    719.692    3972.046   0.181         0.85625    
FirstFlrSF                     121.938      18.569   6.567 0.0000000000784 ***
SecondFlrSF                     50.205      15.506   3.238         0.00124 ** 
GarageCars_d1              -130417.512  268643.352  -0.485         0.62744    
GarageCars_d2              -271423.159  254439.228  -1.067         0.28631    
GarageCars_d3               102328.322  400532.505   0.255         0.79840    
YearBuilt                      351.074     125.913   2.788         0.00539 ** 
BsmtFinSF1                      33.301      10.151   3.281         0.00107 ** 
OverallCond                   9309.423    2049.995   4.541 0.0000061945049 ***
OverallQual:GarageCars_d1     6630.407    4177.652   1.587         0.11277    
OverallQual:GarageCars_d2    10020.559    4074.647   2.459         0.01407 *  
OverallQual:GarageCars_d3    20494.974    4734.644   4.329 0.0000163288016 ***
FirstFlrSF:GarageCars_d1       -45.040      19.262  -2.338         0.01954 *  
FirstFlrSF:GarageCars_d2       -44.541      18.795  -2.370         0.01796 *  
FirstFlrSF:GarageCars_d3       -35.302      21.556  -1.638         0.10176    
SecondFlrSF:GarageCars_d1        4.141      15.962   0.259         0.79536    
SecondFlrSF:GarageCars_d2        5.063      15.647   0.324         0.74632    
SecondFlrSF:GarageCars_d3        5.063      16.432   0.308         0.75806    
GarageCars_d1:YearBuilt         84.157     137.711   0.611         0.54124    
GarageCars_d2:YearBuilt        147.052     130.247   1.129         0.25913    
GarageCars_d3:YearBuilt        -66.107     200.611  -0.330         0.74182    
GarageCars_d1:BsmtFinSF1       -21.151      10.718  -1.973         0.04870 *  
GarageCars_d2:BsmtFinSF1       -13.177      10.317  -1.277         0.20179    
GarageCars_d3:BsmtFinSF1       -13.954      10.778  -1.295         0.19569    
GarageCars_d1:OverallCond    -2941.343    2198.655  -1.338         0.18123    
GarageCars_d2:OverallCond    -2527.491    2185.501  -1.156         0.24773    
GarageCars_d3:OverallCond    -4706.438    4396.811  -1.070         0.28466    
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 16980 on 1128 degrees of freedom
Multiple R-squared:  0.8968,	Adjusted R-squared:  0.8943 
F-statistic:   363 on 27 and 1128 DF,  p-value: < 0.00000000000000022
```

1. Overall Model Fit:
   - The model explains approximately 89.68% of the variability in SalePrice (`Multiple R-squared: 0.8968`).
   - The adjusted R-squared, which adjusts for the number of predictors in the model, is 89.43%. This implies that the model fits well with the data.
   - The F-statistic is very large (363), and the associated p-value is extremely close to zero. This indicates that the predictors collectively are significantly explaining the variance in `SalePrice`.

2. Main Effects:
   - Variables like `FirstFlrSF`, `SecondFlrSF`, `YearBuilt`, `BsmtFinSF1`, and `OverallCond` are all statistically significant at various levels.
   - `OverallQual`, `GarageCars_d1`, `GarageCars_d2`, and `GarageCars_d3` are not individually significant in the presence of other predictors and their interactions.

3. Interaction Effects:
   - The interaction `OverallQual:GarageCars_d2` is significant, indicating that the effect of `OverallQual` on `SalePrice` differs when considering different levels of `GarageCars_d2`.
   - `OverallQual:GarageCars_d3` interaction is also significant, which means the effect of `OverallQual` on `SalePrice` changes across different levels of `GarageCars_d3`.
   - Interactions involving `FirstFlrSF` with both `GarageCars_d1` and `GarageCars_d2` are significant. This suggests the effect of `FirstFlrSF` on `SalePrice` differs by levels of `GarageCars_d1` and `GarageCars_d2`.
   - The interaction `GarageCars_d1:BsmtFinSF1` is significant, meaning the effect of `BsmtFinSF1` on `SalePrice` differs based on the level of `GarageCars_d1`.
   - Many other interactions are not significant, indicating that for those combinations, the effect of one predictor on `SalePrice` does not change by levels of the other predictor.

4. Coefficients:

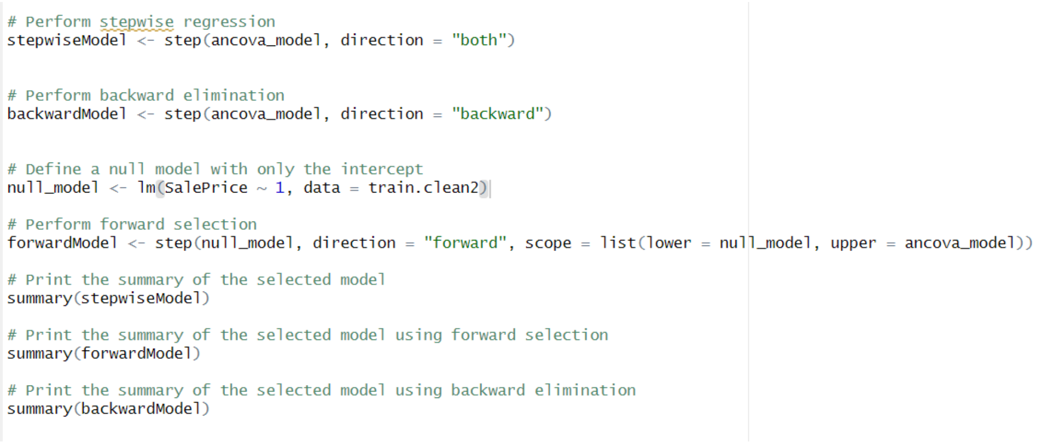
- The coefficient of a term represents the change in the predicted value of `SalePrice` for a one-unit change in the predictor, holding all other predictors constant.
- For example, `YearBuilt` has a coefficient of 351.074. This implies that for each additional year (assuming other variables are constant), the sale price increases by approximately $351.07.

5. Residuals:

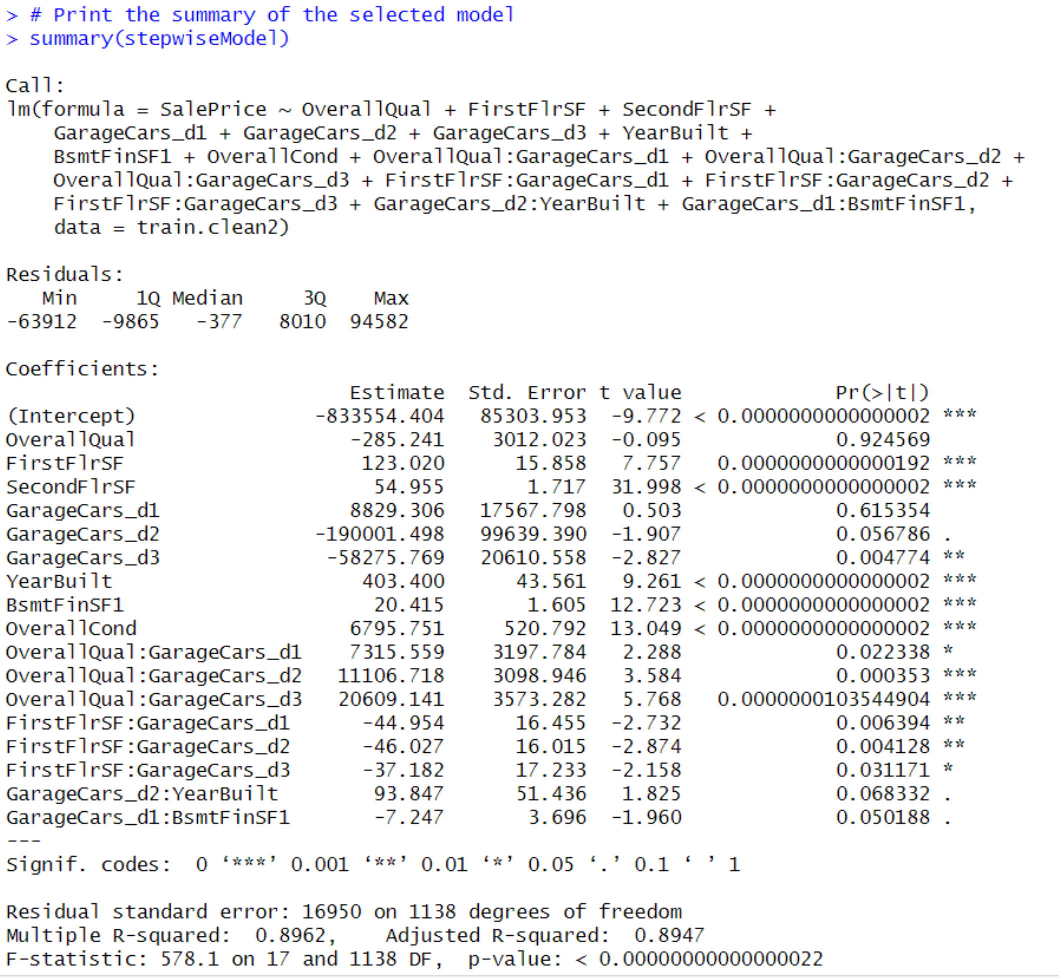
- Residuals represent the differences between the observed and predicted values of `SalePrice`.
- The distribution of residuals (from Min to Max) gives us a sense of how well the model fits the data. If the residuals are symmetrically distributed around zero, it’s an indication that the model is a good fit.

In summary, this model is comprehensive and captures a significant amount of variability in the `SalePrice` data. The presence of significant interaction terms indicates that the effect of certain variables on the sale price is conditional on the levels/values of other variables. It's essential to consider these interactions while making predictions or insights using this model.

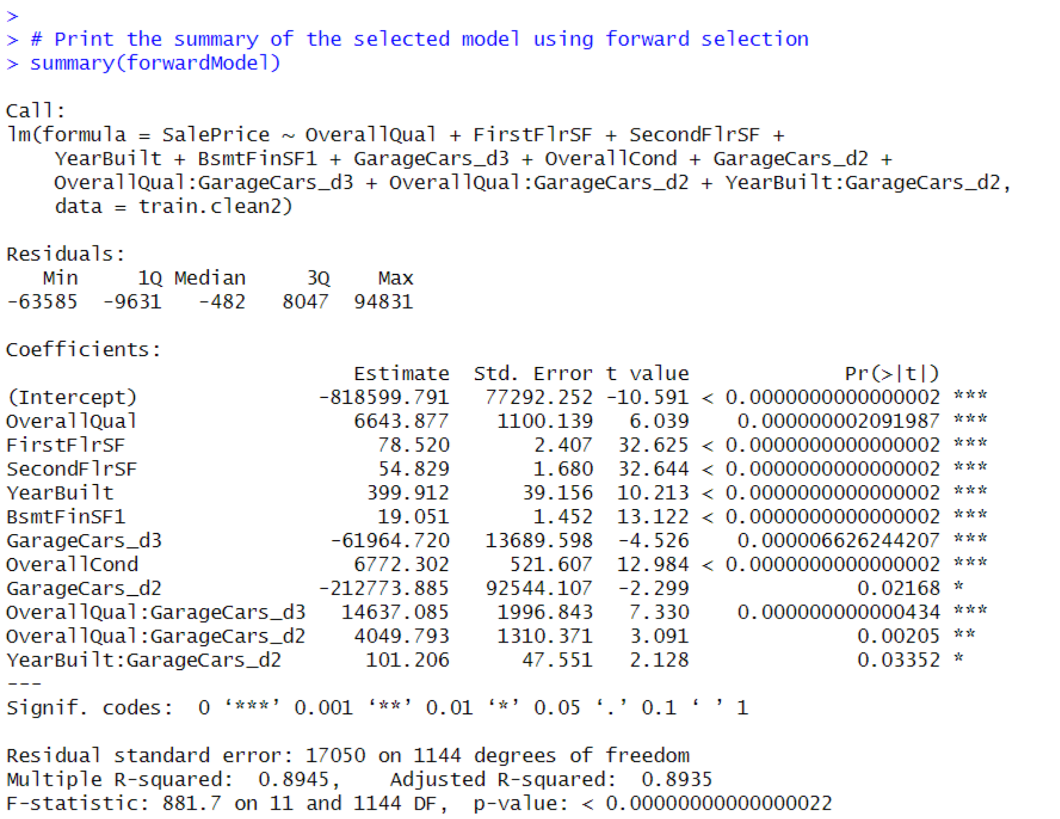
Given the high number of non-significant variables in the model, we have decided to do the automated variable selection using forward, backward, and stepAIC.



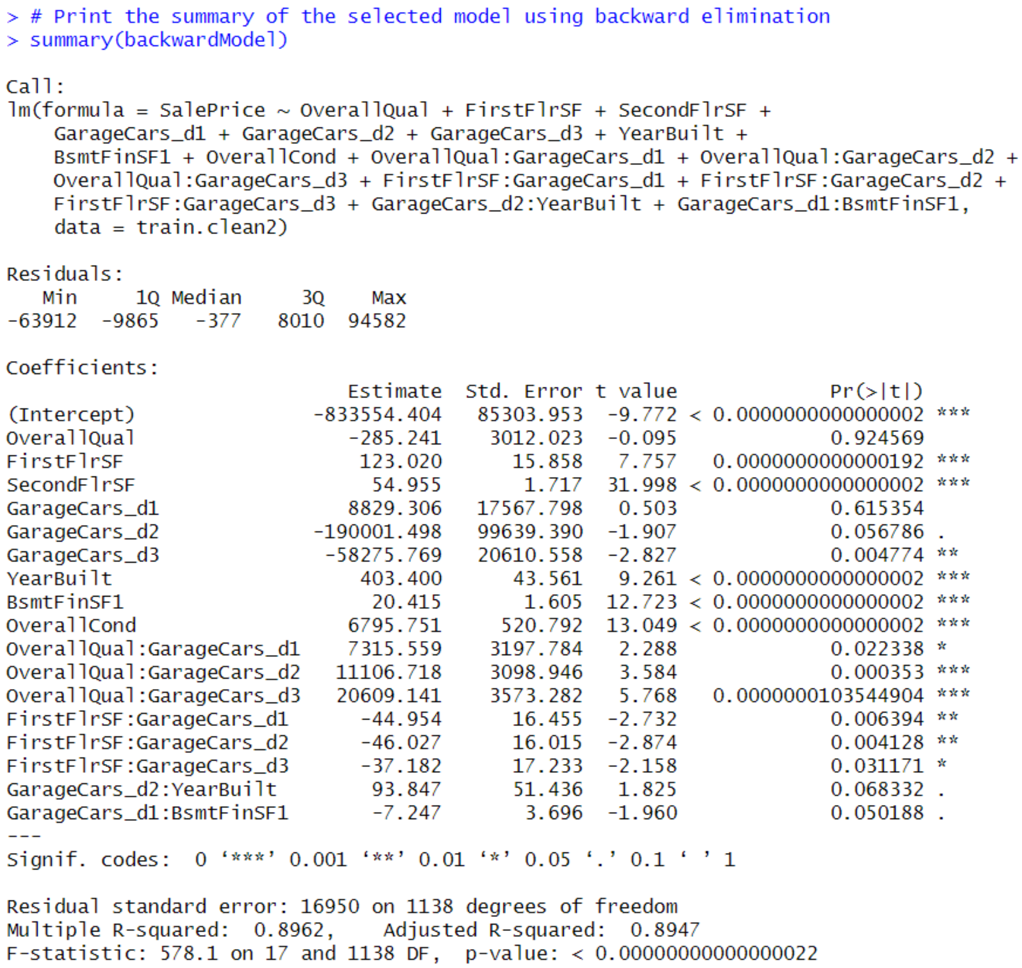
```
# Perform stepwise regression
stepwiseModel <- step(ancova_model, direction = "both")


# Perform backward elimination
backwardModel <- step(ancova_model, direction = "backward")


# Define a null model with only the intercept
null_model <- lm(SalePrice ~ 1, data = train.clean2)

# Perform forward selection
forwardModel <- step(null_model, direction = "forward", scope = list(lower = null_model, upper = ancova_model))

# Print the summary of the selected model
summary(stepwiseModel)

# Print the summary of the selected model using forward selection
summary(forwardModel)

# Print the summary of the selected model using backward elimination
summary(backwardModel)
```



```
> # Print the summary of the selected model
> summary(stepwiseModel)

Call:
lm(formula = SalePrice ~ OverallQual + FirstFlrSF + SecondFlrSF + 
    GarageCars_d1 + GarageCars_d2 + GarageCars_d3 + YearBuilt + 
    BsmtFinSF1 + OverallCond + OverallQual:GarageCars_d1 + OverallQual:GarageCars_d2 + 
    OverallQual:GarageCars_d3 + FirstFlrSF:GarageCars_d1 + FirstFlrSF:GarageCars_d2 + 
    FirstFlrSF:GarageCars_d3 + GarageCars_d2:YearBuilt + GarageCars_d1:BsmtFinSF1, 
    data = train.clean2)

Residuals:
   Min     1Q Median     3Q    Max 
-63912  -9865   -377   8010  94582 

Coefficients:
                             Estimate  Std. Error t value             Pr(>|t|)    
(Intercept)               -833554.404  85303.953  -9.772 < 0.0000000000000002 ***
OverallQual                  -285.241   3012.023  -0.095             0.924569    
FirstFlrSF                    123.020     15.858   7.757   0.0000000000000192 ***
SecondFlrSF                    54.955      1.717  31.998 < 0.0000000000000002 ***
GarageCars_d1                8829.306  17567.798   0.503             0.615354    
GarageCars_d2             -190001.498  99639.390  -1.907             0.056786 .  
GarageCars_d3              -58275.769  20610.558  -2.827             0.004774 ** 
YearBuilt                     403.400     43.561   9.261 < 0.0000000000000002 ***
BsmtFinSF1                     20.415      1.605  12.723 < 0.0000000000000002 ***
OverallCond                  6795.751    520.792  13.049 < 0.0000000000000002 ***
OverallQual:GarageCars_d1    7315.559   3197.784   2.288             0.022338 *  
OverallQual:GarageCars_d2   11106.718   3098.946   3.584             0.000353 ***
OverallQual:GarageCars_d3   20609.141   3573.282   5.768   0.0000000103544904 ***
FirstFlrSF:GarageCars_d1      -44.954     16.455  -2.732             0.006394 ** 
FirstFlrSF:GarageCars_d2      -46.027     16.015  -2.874             0.004128 ** 
FirstFlrSF:GarageCars_d3      -37.182     17.233  -2.158             0.031171 *  
GarageCars_d2:YearBuilt        93.847     51.436   1.825             0.068332 .  
GarageCars_d1:BsmtFinSF1       -7.247      3.696  -1.960             0.050188 .  
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 16950 on 1138 degrees of freedom
Multiple R-squared:  0.8962,	Adjusted R-squared:  0.8947 
F-statistic: 578.1 on 17 and 1138 DF,  p-value: < 0.00000000000000022
```



```
> 
> # Print the summary of the selected model using forward selection
> summary(forwardModel)

Call:
lm(formula = SalePrice ~ OverallQual + FirstFlrSF + SecondFlrSF + 
    YearBuilt + BsmtFinSF1 + GarageCars_d3 + OverallCond + GarageCars_d2 + 
    OverallQual:GarageCars_d3 + OverallQual:GarageCars_d2 + YearBuilt:GarageCars_d2, 
    data = train.clean2)

Residuals:
   Min     1Q Median     3Q    Max 
-63585  -9631   -482   8047  94831 

Coefficients:
                             Estimate  Std. Error t value             Pr(>|t|)    
(Intercept)               -818599.791  77292.252 -10.591 < 0.0000000000000002 ***
OverallQual                  6643.877   1100.139   6.039    0.000000002091987 ***
FirstFlrSF                     78.520      2.407  32.625 < 0.0000000000000002 ***
SecondFlrSF                    54.829      1.680  32.644 < 0.0000000000000002 ***
YearBuilt                     399.912     39.156  10.213 < 0.0000000000000002 ***
BsmtFinSF1                     19.051      1.452  13.122 < 0.0000000000000002 ***
GarageCars_d3              -61964.720  13689.598  -4.526    0.000006626244207 ***
OverallCond                  6772.302    521.607  12.984 < 0.0000000000000002 ***
GarageCars_d2             -212773.885  92544.107  -2.299              0.02168 *  
OverallQual:GarageCars_d3   14637.085   1996.843   7.330    0.000000000000434 ***
OverallQual:GarageCars_d2    4049.793   1310.371   3.091              0.00205 ** 
YearBuilt:GarageCars_d2       101.206     47.551   2.128              0.03352 *  
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 17050 on 1144 degrees of freedom
Multiple R-squared:  0.8945,	Adjusted R-squared:  0.8935 
F-statistic: 881.7 on 11 and 1144 DF,  p-value: < 0.00000000000000022
```



```
> # Print the summary of the selected model using backward elimination
> summary(backwardModel)

Call:
lm(formula = SalePrice ~ OverallQual + FirstFlrSF + SecondFlrSF + 
    GarageCars_d1 + GarageCars_d2 + GarageCars_d3 + YearBuilt + 
    BsmtFinSF1 + OverallCond + OverallQual:GarageCars_d1 + OverallQual:GarageCars_d2 + 
    OverallQual:GarageCars_d3 + FirstFlrSF:GarageCars_d1 + FirstFlrSF:GarageCars_d2 + 
    FirstFlrSF:GarageCars_d3 + GarageCars_d2:YearBuilt + GarageCars_d1:BsmtFinSF1, 
    data = train.clean2)

Residuals:
   Min     1Q Median     3Q    Max 
-63912  -9865   -377   8010  94582 

Coefficients:
                             Estimate  Std. Error t value             Pr(>|t|)    
(Intercept)               -833554.404   85303.953  -9.772 < 0.0000000000000002 ***
OverallQual                  -285.241    3012.023  -0.095             0.924569    
FirstFlrSF                    123.020      15.858   7.757   0.0000000000000192 ***
SecondFlrSF                    54.955       1.717  31.998 < 0.0000000000000002 ***
GarageCars_d1                8829.306   17567.798   0.503             0.615354    
GarageCars_d2             -190001.498   99639.390  -1.907             0.056786 .  
GarageCars_d3              -58275.769   20610.558  -2.827             0.004774 ** 
YearBuilt                     403.400      43.561   9.261 < 0.0000000000000002 ***
BsmtFinSF1                     20.415       1.605  12.723 < 0.0000000000000002 ***
OverallCond                  6795.751     520.792  13.049 < 0.0000000000000002 ***
OverallQual:GarageCars_d1    7315.559    3197.784   2.288             0.022338 *  
OverallQual:GarageCars_d2   11106.718    3098.946   3.584             0.000353 ***
OverallQual:GarageCars_d3   20609.141    3573.282   5.768   0.0000000103544904 ***
FirstFlrSF:GarageCars_d1      -44.954      16.455  -2.732             0.006394 ** 
FirstFlrSF:GarageCars_d2      -46.027      16.015  -2.874             0.004128 ** 
FirstFlrSF:GarageCars_d3      -37.182      17.233  -2.158             0.031171 *  
GarageCars_d2:YearBuilt        93.847      51.436   1.825             0.068332 .  
GarageCars_d1:BsmtFinSF1       -7.247       3.696  -1.960             0.050188 .  
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 16950 on 1138 degrees of freedom
Multiple R-squared:  0.8962,	Adjusted R-squared:  0.8947 
F-statistic: 578.1 on 17 and 1138 DF,  p-value: < 0.00000000000000022
```

To compare the three models (`stepwiseModel`, `forwardModel`, and `backwardModel`), we can look at several criteria:

1. Number of Predictors:
    - `stepwiseModel` and `backwardModel` both have 17 predictors.
    - `forwardModel` has 11 predictors.

2. Adjusted R-squared:
    - `stepwiseModel` and `backwardModel`: 0.8947
    - `forwardModel`: 0.8935
    - The adjusted R-squared value takes into account the number of predictors in the model. Higher adjusted R-squared indicates a better fit for the model, considering the number of predictors. In this case, `stepwiseModel` and `backwardModel` have slightly higher adjusted R-squared than the `forwardModel`, but the difference is minimal.

3. Residual Standard Error:
    - `stepwiseModel` and `backwardModel`: 16950
    - `forwardModel`: 17050

The residual standard error gives an average measure of how far off our predictions are. A lower value is preferable. In this case, `stepwiseModel` and `backwardModel` have a slightly lower value than `forwardModel`.

4. Significance of Predictors:
    - For both `stepwiseModel` and `backwardModel`, some predictors have p-values close to 0.05, suggesting they might not be as statistically significant.

- `forwardModel` predictors seem to have lower p-values overall.

5. Complexity: More predictors in a model can make it more complex and harder to interpret. If two models perform similarly, it might be beneficial to choose the simpler one with fewer predictors.

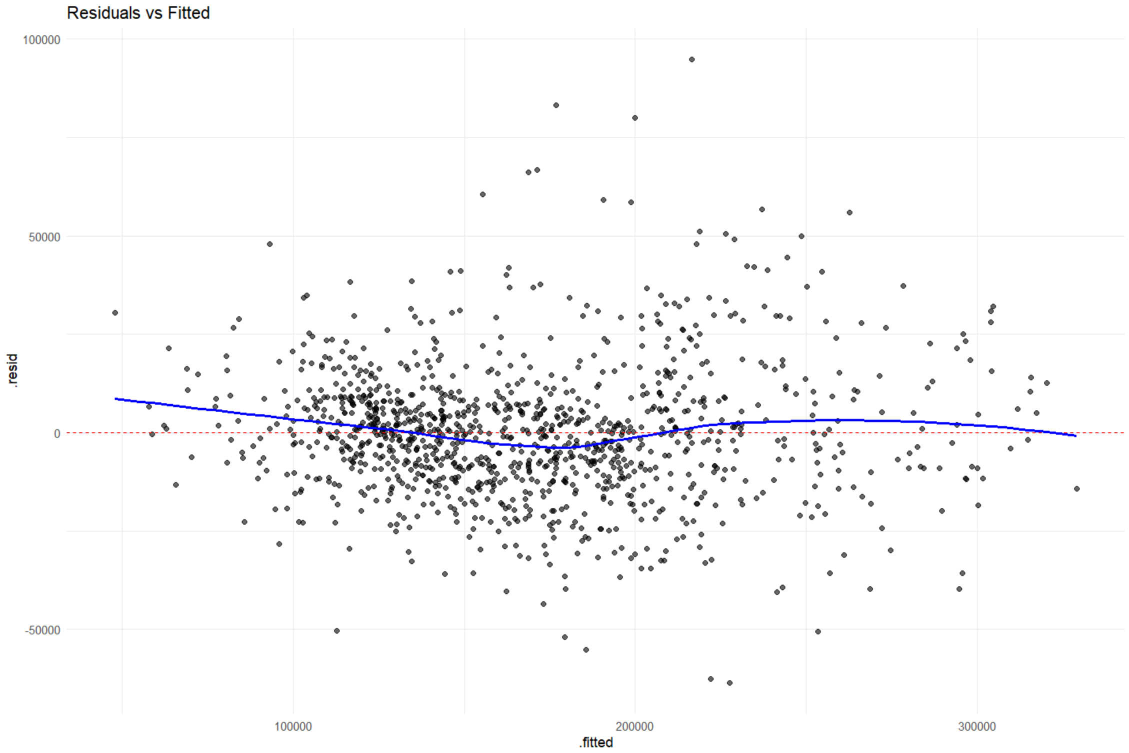
6. Potential for Overfitting: Models with more predictors, especially if they're not statistically significant, can risk overfitting. Overfitted models might perform well on training data but poorly on new, unseen data.

Conclusion:

- The `stepwiseModel` and `backwardModel` are essentially the same based on the provided summaries.
- While the `stepwiseModel` and `backwardModel` have a slightly higher adjusted R-squared and lower residual standard error, the `forwardModel` achieves nearly the same performance metrics with fewer predictors.

The decision to pick a model could depend on the purpose. If interpretability and simplicity are priorities, the `forwardModel` might be preferable.

Goodness for fit and diagnostic analysis on `forwardModel`:

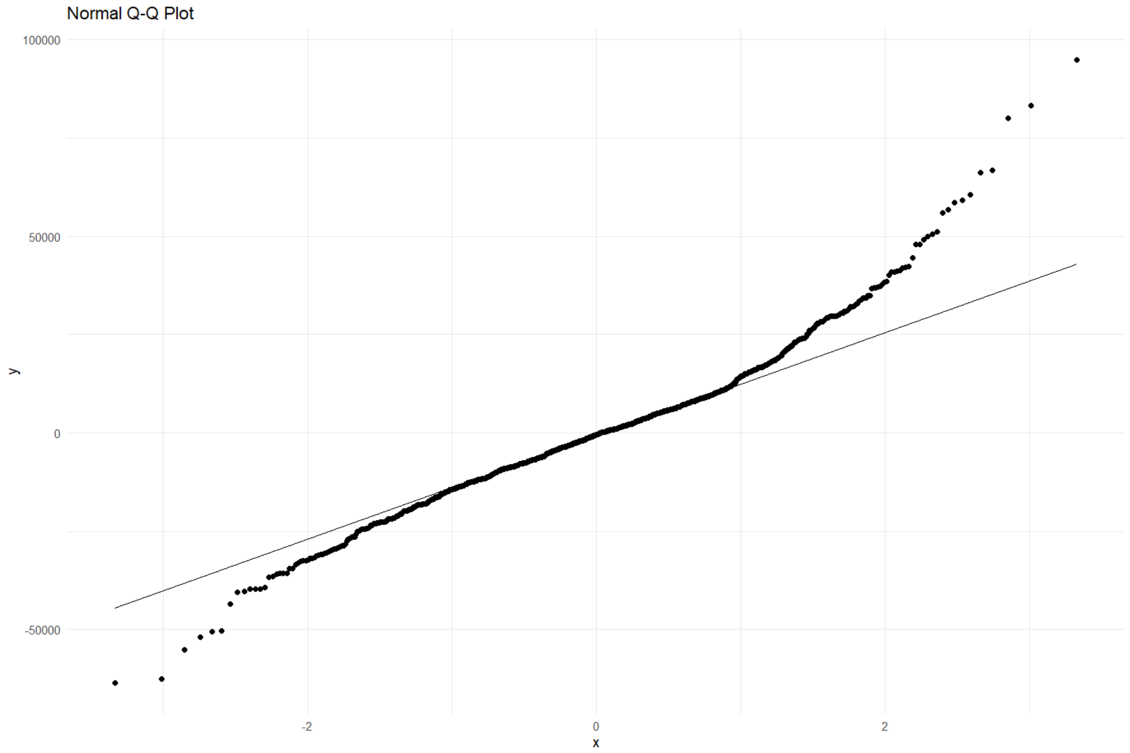


Interpretation:

Pattern in Residuals: The blue curve suggests a slight U-shaped pattern, indicating that the model might be missing some non-linear relationship in the data. This could be due to missing higher-order terms (like quadratic or cubic terms) or interactions between predictors.

Variance of Residuals: The spread of residuals seems to be somewhat constant across the range of fitted values, which is a good sign.

Outliers: There are a few points that stand far away from the horizontal red dashed line, suggesting the presence of potential outliers. These data points might be influential observations that can unduly affect the regression results. One might want to further investigate these points to ensure they're not errors or anomalies.

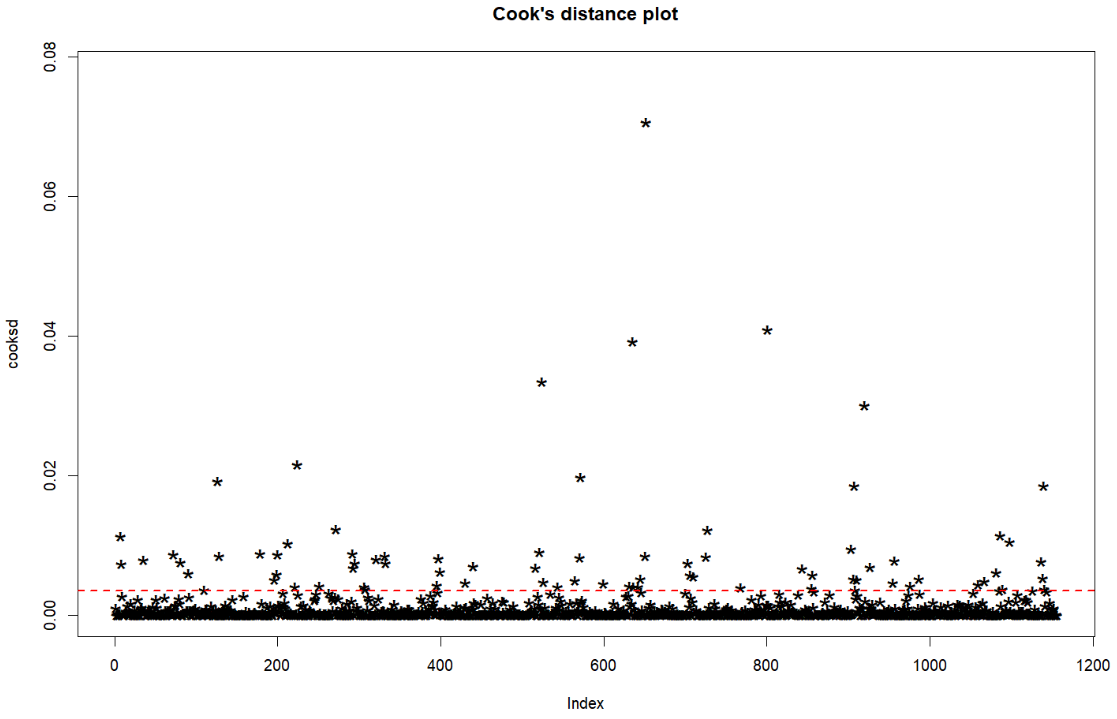


Interpretation:

General Adherence: For a large portion of the plot, the residuals follow the straight line closely, suggesting that the central part of the residual distribution is roughly normal.

Tail Behavior: There's a noticeable deviation from the straight line at both tails, especially the upper right tail. The upward curve at the upper right tail suggests that the residuals have heavier tails than a normal distribution, indicating the presence of potential outliers or that the residuals may have a skewed distribution. The downward curve at the bottom left also indicates potential outliers or deviations from normality, though it's less pronounced than the upper tail.

Outliers: Points that stray far from the diagonal line, especially in the tails, might be considered outliers. They could have a disproportionately large impact on your model, and one may want to further investigate these data points.



Interpretation:

Cook's Distance Threshold: The red dashed line represents the threshold for Cook's distance. Observations with a Cook's distance above this line are typically considered influential points, meaning they have a significant impact on the regression model's coefficients.

Distribution of Data Points: Most data points lie near the x-axis, indicating that they have low Cook's distances and, therefore, a minimal influence on the regression model.

Sparse Region: Towards the right of the plot, the observations are more scattered and less dense. This may indicate that data in that region of the index might be more varied or less consistent than in the dense region.

In conclusion, while the majority of the data seems consistent and non-influential, the few potential influential points need to be investigated further.

For reflection / conclusions: After working on this problem and this data for several weeks, what are the challenges presented by the data? What are your recommendations for improving predictive accuracy? What do you think of the notion of parsimony: simpler models might be preferable over complicated models? Do we really need a max fit model or is a simpler but more interpretable model better?

The Ames Housing dataset is a comprehensive dataset offering a plethora of features for predicting house prices. However, it comes with its challenges and presents an opportunity to explore the broader concepts of model complexity and interpretability.

Challenges:
1. Dimensionality: With over 80 features, it's easy to fall into the trap of the curse of dimensionality, potentially leading to overfitting.

2. Missing Values: Some features have missing data. Deciding how to handle these—whether through imputation, deletion, or other techniques—is a key concern.

3. Diverse Data Types: The dataset combines numerical, ordinal, and nominal features. Proper encoding and scaling are essential for many algorithms.

4. Outliers and Skewness: Certain features contain outliers or have a skewed distribution, requiring transformations or specialized treatment.

Recommendations for Improving Predictive Accuracy on the Ames Dataset:

1. Data Preprocessing: This dataset contains a mix of missing values, skewed distributions, and outliers. Effective imputation strategies, such as median imputation for numerical features and mode imputation for categorical ones, can be employed. For skewed features, transformations (e.g., logarithmic) can help in normalizing them, while robust scalers or IQR-based techniques can be used for handling outliers.

2. Feature Engineering: Given the dataset's rich set of features, derived variables might add predictive power. Interaction terms, polynomial features, or even domain-inspired attributes can be beneficial. Categorical variables should be carefully encoded; while one-hot encoding is common, ordinal or target encoding can be considered, depending on the model being used.

3. Model Selection and Ensemble: While linear regression might be a natural first choice, given the dataset's size and complexity, tree-based models like Random Forests or Gradient Boosting Machines (GBM) might yield better results. Ensembling different model outputs, either through stacking or blending, can enhance predictive accuracy further.

4. Regularization and Hyperparameter Tuning: With so many features, overfitting can easily creep in. Regularization techniques such as Lasso (L1) or Ridge (L2) can help. Additionally, rigorous hyperparameter tuning using cross-validation ensures that the model isn't just performing well on the training set but is expected to generalize well to unseen data.

Parsimony and Model Choice on the Ames Dataset: The principle of parsimony, deeply rooted in scientific endeavors, posits that when two models perform similarly, the simpler one is usually more desirable.

Interpretability: A simpler model, like linear regression, offers clarity. Each coefficient provides direct insight into the relationship between the corresponding feature and the target. In contrast, while a model like GBM might offer better accuracy, it lacks this straightforward interpretability.

Overfitting: The high dimensionality of the Ames dataset increases the risk of overfitting, especially with complex models. A simpler model with fewer parameters might be less prone to capturing noise.

Deployment and Maintenance: Simpler models are generally easier to deploy, maintain, and troubleshoot. Their predictability can be invaluable in real-world applications. However, the Ames dataset, with its nuanced features and relationships, might intrinsically require a more complex model to capture its underlying patterns adequately.

Balancing Fit and Simplicity: The decision between a max fit model and a simpler model hinges on the problem's context and goals. For real-world applications, where model decisions might need justifications or where the model might need frequent updates and maintenance, a simpler, more interpretable model might be more suitable.

In the context of the Ames dataset, one might start with a linear regression model, evaluating its performance and interpretability. If the results are satisfactory, and the primary audience for the model values transparency, this might be the endpoint. However, if accuracy is paramount and the audience is comfortable with a more complex model, progressing to models like GBM or even neural networks could be the next step.

In essence, while the Ames dataset offers opportunities to create highly accurate models, it also underscores the importance of context. Predictive accuracy is just one metric of success; the broader implications of the model's complexity, interpretability, and maintainability must also be weighed. The ideal model would strike a balance, providing actionable insights without sacrificing too much on accuracy or becoming an inscrutable black box.