Capstone Project Harvard PH125.9X

Machine Learning Models for analysis of Sun City Real Estate

Herbert Holeman

February 23, 2022

This report is available on Github at: HarvardX-PH125.9X/Holeman_Capstone_2.pdf

Introduction

The general goal of machine learning in prediction is to build models out of the patterns in data to use for making future predictions. To that end, this project concerns the construction of a project-oriented dataset and the building of two such machine learning models relaated to criterion of interest to homebuyers and sellers.

Chosen as the data source for this project are the homes in the 3,000 acres community of Sun City Lincoln Hills in California's Sierra foothills. It is an adult homeowner retirement community of some 11,000 residents with a median age of 74, of which 43% are males, and 57% are female. They are essentially empty nesters who live in 6,703 houses and 80 condominiums. Families occupy no homes with children. Couples reside in 58% of the homes, single-females occupy 34%, and single-males 8%.

Sun City itself is a walled-in community. The residents enjoy dining, entertainment, and a wide range of recreational facilities, including bocce ball, softball, tennis, Wiffle ball, indoor/outdoor swimming pools, parks, and golf courses. Sun City is within the census area of the City of Lincoln, which is ranked nationally as a safe city. Violent crime rank is 14.6 compared to the US average of 22.7. Property crime rank is 19.9; the US average is 35.4.

Homebuilding in Sun City occurred between 1999 and 2008. The homes range from 900 sq ft cottages to 11,000 sq ft estates. Since its construction, Sun City has experienced a steady turnover in homeownership and rising home values. Accordingly, the project deals in home selection and sale price, key issues to homebuyers and sellers.

Methods and Analysis

Two supervised machine learning models are built:

- Decision Tree classification model to predict the home type based on the other feature variables.
- KNN regression model to predict home selling price based on dataset features.

The methodology for doing so involves five stages.

1. Use of a personally webscraped dataset that is exclusive to the project. This decision is in keeping with instructional guidance to avoid using well-known datasets, particularly ones that have been used as examples in previous courses or are similar to them. Accordingly, the project dataset is constructed from public real estate sources on the web of homes sold in Sun City Lincoln Hills California during the years 2019 through 2021 and recorded by the Placer County assessor.
2. Exploratory data analysis (EDA) to provide a high-level understanding of home types, home sale prices, home quality ratings, and home attributes, such as lot size, living area, bedrooms, and bathrooms.
3. Supervised machine learning modeling with R tidymodels, which is more or less a restructuring of the caret framework. It includes these packages:

   rsample to split and sample data,

   recipe for feature engineering and preprocessing data,

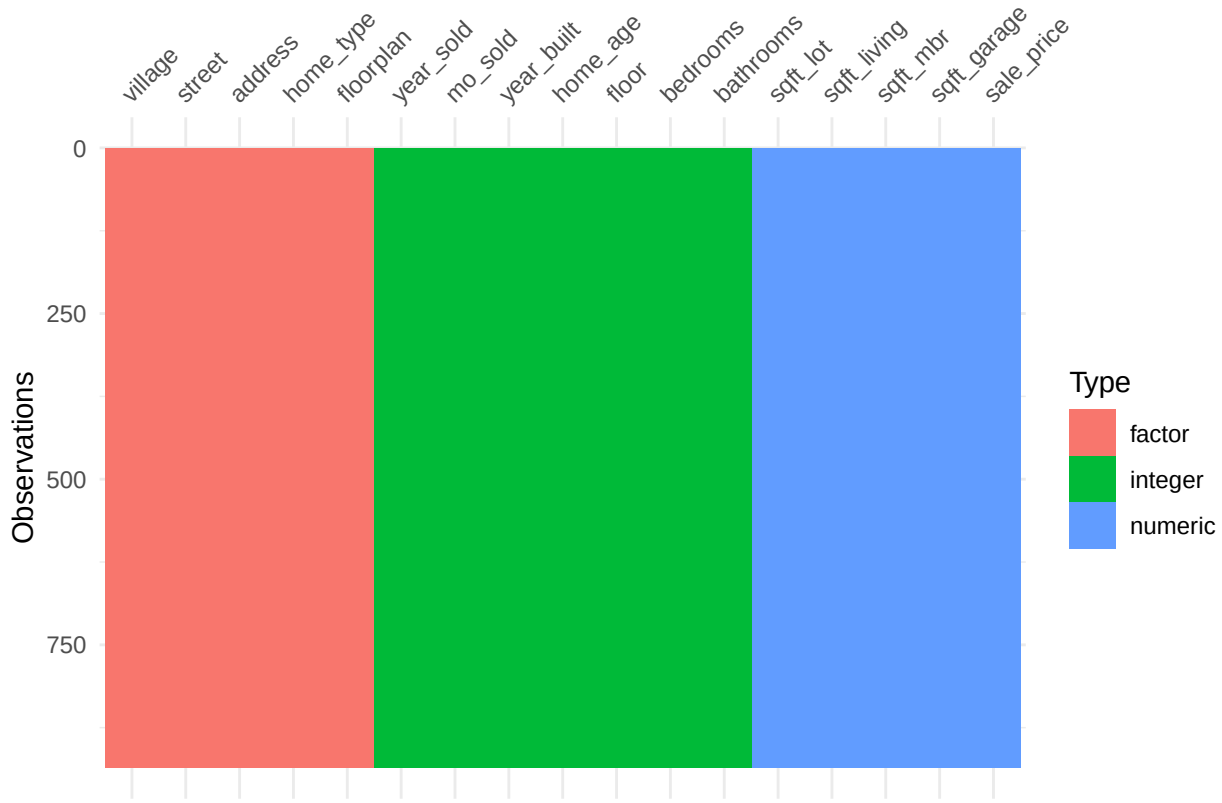
   parsnip to select and specify models,

   workflows and *tune* to adjust and improve model processing, and

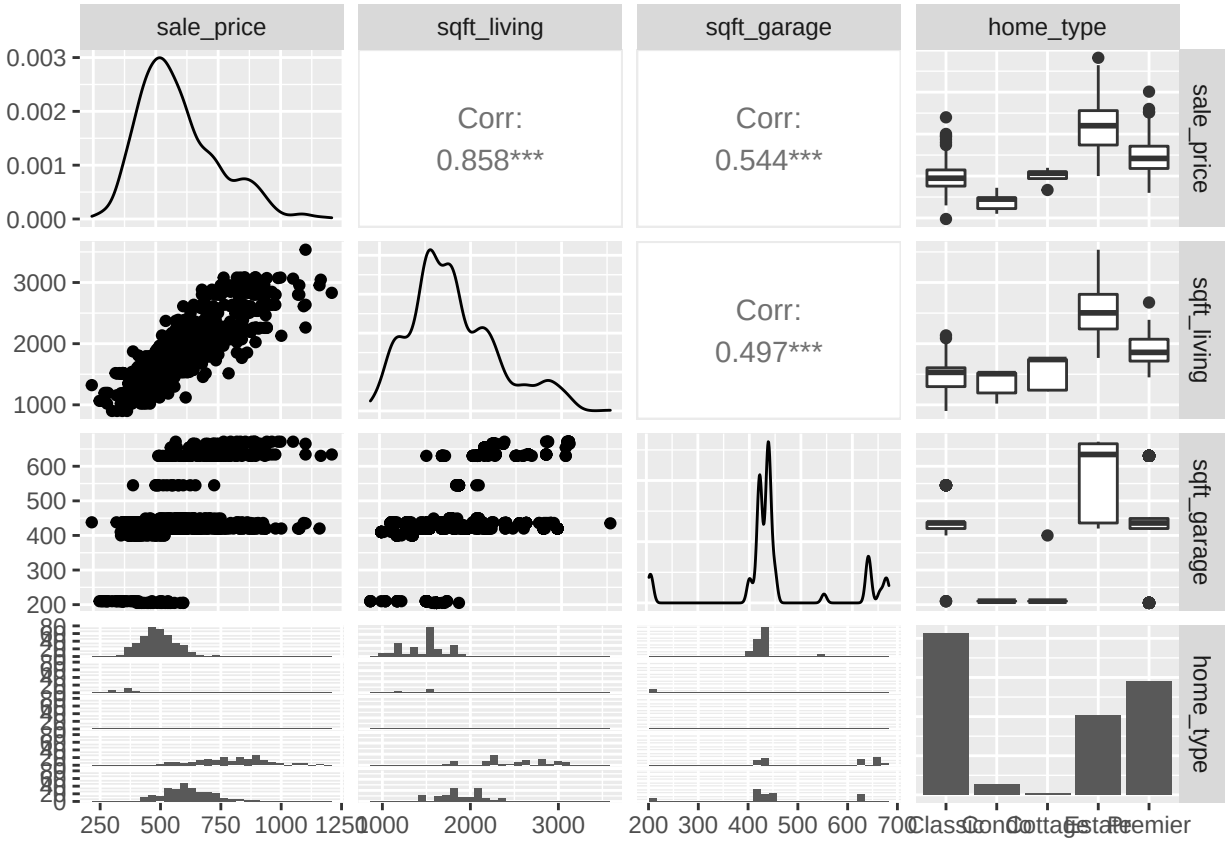
   yardstick to evaluate model performance.
4. Performance results of the Decision Tree and KNN models' predictions.
5. Project summary of the models' utility and their potential for predicting future home selection and sale price in Sun City.

The Sun City dataset.

The project dataset contains features of the 935 homes sold in Sun City Lincoln Hills from 2019 through 2021. Each column entry is a feature, and each row represents a home sold.



In this overview of dataset features are the categorical and numerical variables in the data. Correlation coefficients are Pearson by default.



Exploratory Data Analysis (EDA).

EDA begins with a glimpse of the dataset, and a missing data check.

```
### Dataset glimpse
glimpse(sun)
```

```
## Rows: 935
## Columns: 17
## $ year_sold   <int> 2020, 2021, 2021, 2020, 2020, 2021, 2020, 2020, 2020, 2021~
## $ mo_sold     <int> 5, 7, 10, 10, 2, 8, 10, 5, 8, 11, 5, 10, 8, 6, 1, 7, 7, 1,~
## $ year_built  <int> 2004, 2004, 2004, 2004, 2004, 2004, 2005, 2004, 2004, 2004~
## $ home_age    <int> 18, 18, 18, 18, 18, 18, 17, 18, 18, 18, 20, 18, 18, 18, 17~
## $ village     <fct> Village 1, Village 1, Village 1, Village 1, Village 1, Vil~
## $ street      <fct> Hay Wagon Court, Haywagon Court, Calistoga Lane, Wagon Whe~
## $ address     <fct> 101 Hay Wagon Court, 108 Haywagon Court, 234 Calistoga Lan~
## $ sqft_lot    <dbl> 7039, 8760, 7318, 8934, 4752, 5955, 6242, 7169, 5811, 6639~
## $ sqft_living <dbl> 1531, 1531, 1549, 1549, 1190, 1673, 1775, 1531, 1531, 1190~
## $ sqft_mbr    <dbl> 156, 156, 156, 156, 168, 178, 262, 156, 156, 168, 184, 178~
## $ sqft_garage <dbl> 420, 420, 420, 420, 400, 438, 438, 420, 420, 400, 430, 438~
## $ home_type   <fct> Classic, Classic, Classic, Classic, Classic, Classic, Clas~
## $ floorplan   <fct> Alpine, Alpine, Alpine, Alpine, Calaveras, Trinity, Tehama~
## $ floor       <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1~
## $ bedrooms    <int> 2, 2, 2, 2, 2, 2, 3, 2, 2, 2, 2, 2, 3, 3, 1, 2, 2, 2, 2, 2~
## $ bathrooms   <int> 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2~
## $ sale_price  <dbl> 420, 508, 600, 475, 395, 590, 500, 460, 465, 490, 550, 685~
```

```
##   year_sold     mo_sold  year_built    home_age     village      street
##           0           0           0           0           0           0
##     address    sqft_lot sqft_living    sqft_mbr sqft_garage   home_type
##           0           0           0           0           0           0
##   floorplan       floor    bedrooms   bathrooms  sale_price
##           0           0           0           0           0
```

All homes were built over a ten-year timeframe and included condos, cottages, and houses. Classic, Premier, and Estate refer to house types.

	1999	2000	2001	2002	2003	2004	2005	2006	2007	2008
Classic	17	41	69	78	69	68	52	17	0	1
Condo	0	0	0	0	0	0	0	0	26	0
Cottage	0	0	0	0	0	0	5	0	0	0
Estate	9	11	29	14	41	52	40	5	1	0
Premier	18	48	53	54	50	42	18	7	0	0

Home floor plans define a home type. While condos and cottages have two floor plans, houses have over forty. The Classic house-type floor plans are the base size in square feet, and Premier house types include more extensive floor plans and higher sales prices. Estate house types have upscale floor plans, the costliest and often the largest.

These are the names of house floor plans.

Classic house floor plans.

```
##  [1] Alpine        Bridgegate    Calaveras     Echo Ridge    Madera
##  [6] Mariposa      Millpond      Orchard Crest Pine Hill     Quail Cove
## [11] Sequoia       Tehama        Trinity       Woodleaf
## 42 Levels: Almanor Alpine Annadel Atherton Baldwin Bridgegate ... Woodleaf
```

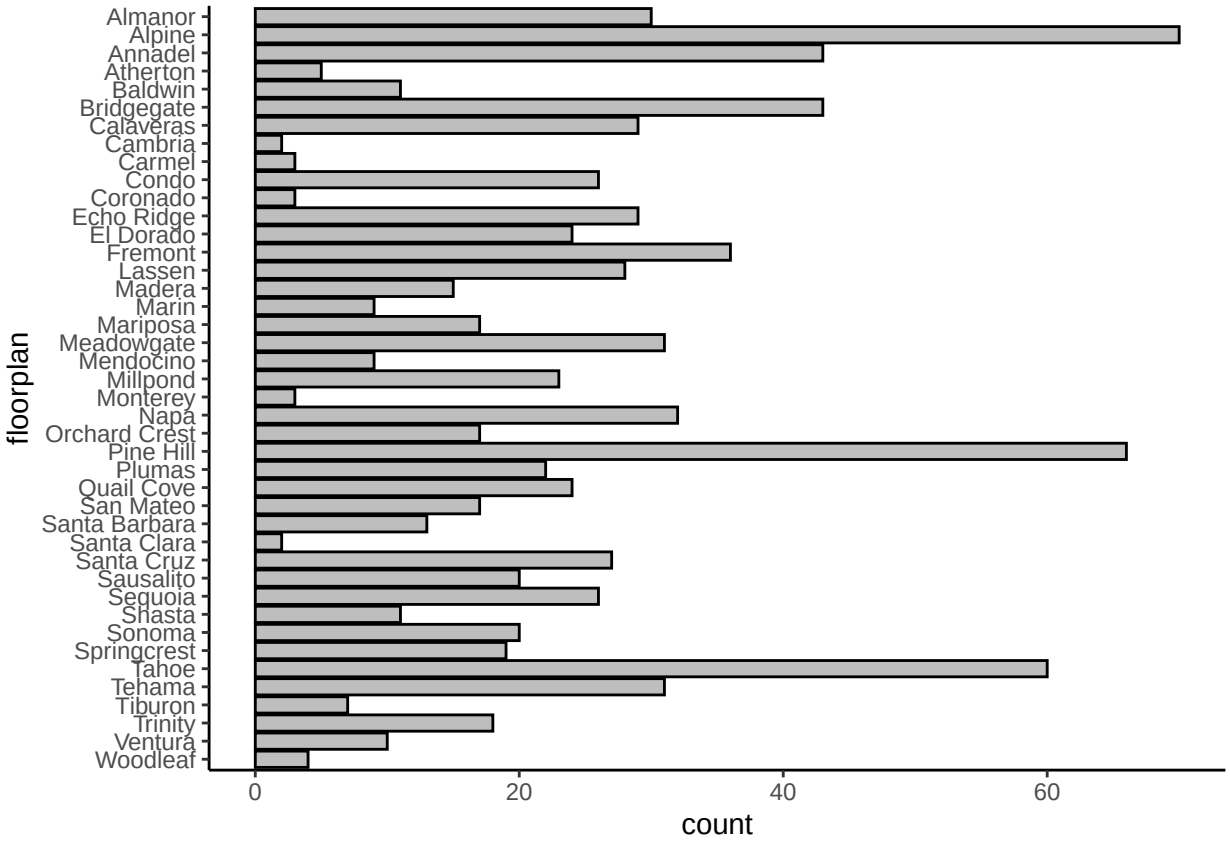
Premier house floor plans.

```
##  [1] Almanor   Annadel   Baldwin   El Dorado Fremont   Lassen    Mendocino
##  [8] Plumas    San Mateo Tahoe     Ventura
## 42 Levels: Almanor Alpine Annadel Atherton Baldwin Bridgegate ... Woodleaf
```

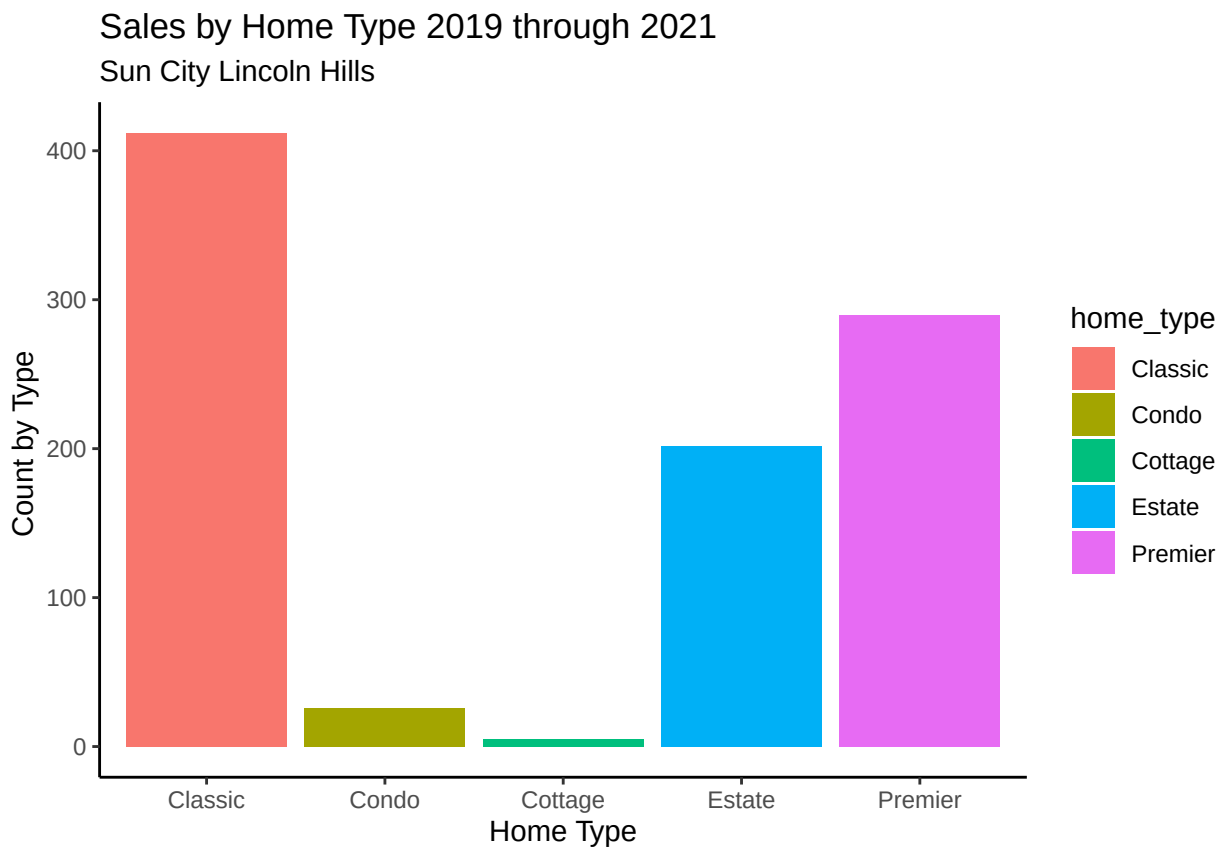
Estate house floor plans.

```
##  [1] Atherton      Carmel        Marin         Meadowgate    Monterey
##  [6] Napa          Santa Barbara Santa Clara   Santa Cruz    Sausalito
## [11] Shasta        Sonoma        Springcrest   Tiburon
## 42 Levels: Almanor Alpine Annadel Atherton Baldwin Bridgegate ... Woodleaf
```

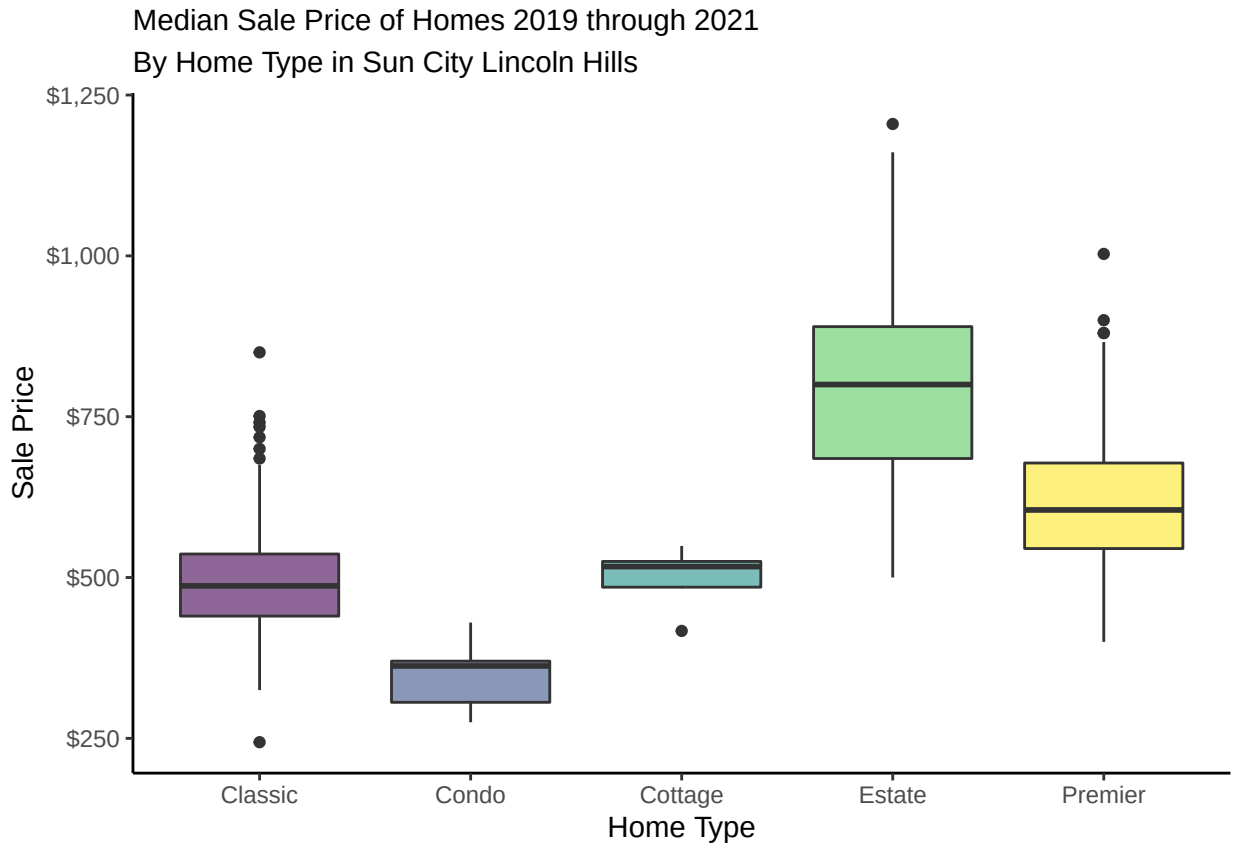
This bar chart shows the distribution of home sales (condo, cottage, and house) in 2019 through 2021 by floor plan. The Classic house accounts for most home sales.



This histogram is of home sales by type.



The median home sale price in 2019 through 2021 is $564,000, with the least expensive at $244,000 and the most expensive at $1,205,000. Note the outliers in the sale price of Estate and Classic houses.



This tabulation is of home sales by home type in 2019 through 2021.

```
##
## Classic   Condo Cottage  Estate Premier
##     412      26       5     202     290
```

These are the proportions of total homes sales by type in 2019 through 2021.

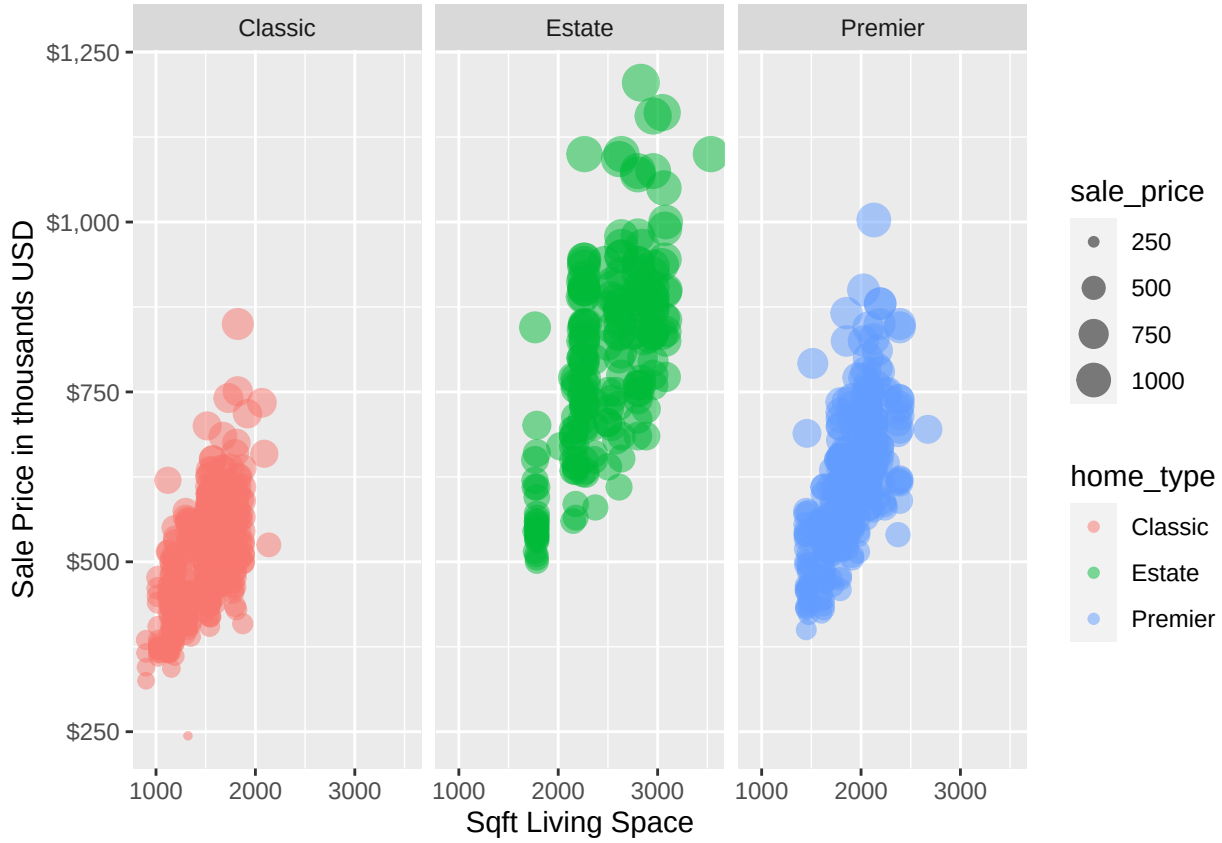
```
##
## Classic   Condo Cottage  Estate Premier
##   44.06    2.78    0.53   21.60   31.02
```

Model refinement: home type(s) selected for modeling.

Homes not selected include condos as they are essentially an apartment and cottages as they number less than a handful. Limiting data dimensions also improves modeling accuracy.

Houses are the home type chosen for modeling and decisions made on which features to include in the process.

```
##
## Classic  Estate Premier
##     412     202     290
```



Features not selected for modeling.

- Review house sales by village.

	Classic	Estate	Premier
Village 28	2	1	3
Village 36	8	2	6
Village 1	7	5	2
Village 10	17	9	5
Village 11	2	1	3
Village 12	0	2	4
Village 13	7	0	0
Village 15	6	0	12
Village 16	8	10	17
Village 17	4	0	0
Village 18	2	0	5
Village 19	37	10	28
Village 2	7	5	5
Village 20	7	1	8
Village 21	8	2	7
Village 23	4	0	1
Village 24	29	14	7
Village 25	2	4	8
Village 26	6	21	14
Village 27	0	0	1
Village 29	6	1	8
Village 3	13	8	6
Village 30	31	5	24
Village 31	18	5	10
Village 32	0	12	0
Village 33	3	4	7
Village 34	18	1	4
Village 35	10	9	4
Village 37	0	3	1
Village 38	43	13	37
Village 39	2	4	3
Village 4	13	15	8
Village 40	12	0	10
Village 41	24	7	9
Village 42	9	1	0
Village 43	5	0	1
Village 5	2	7	3
Village 54	0	2	0
Village 6	1	1	1
Village 7	14	8	2
Village 8	9	1	1
Village 9	2	3	5
Villagre 22	9	2	6
Villge 14	5	3	4


Sun City Lincoln Hills
Village Numbers
KILAGA SPRINGS LODGE
ANGLER'S COVE
SPORTS PAVILION
ORCHARD CREEK LODGE
1
2
3
42
44
43
5
6
4
41
7
10
8
9
11
40
12
15
14
13
17
19
21
18
16
20
22
23
24
26
27
25
34
30
31
32
38
39
36
37
35

- Village data need not be included in modeling. Village numerical assignment is not germane to modeling as it merely refers to construction schedules. Neither relevant to modeling is a comparison of sales volume by the village as villages range in size from a hamlet to a large block of houses. Nor is village milieu relevant because little distinguishes one village from another. Stringent HOA rules dictate uniformity in house exteriors and landscaping.
- Other features not included for modeling: the number of house floor levels, bedrooms, and bathrooms. Virtually all houses are single story and have two bathrooms and two bedrooms. With very few exceptions, Estate models may have more than two bedrooms.

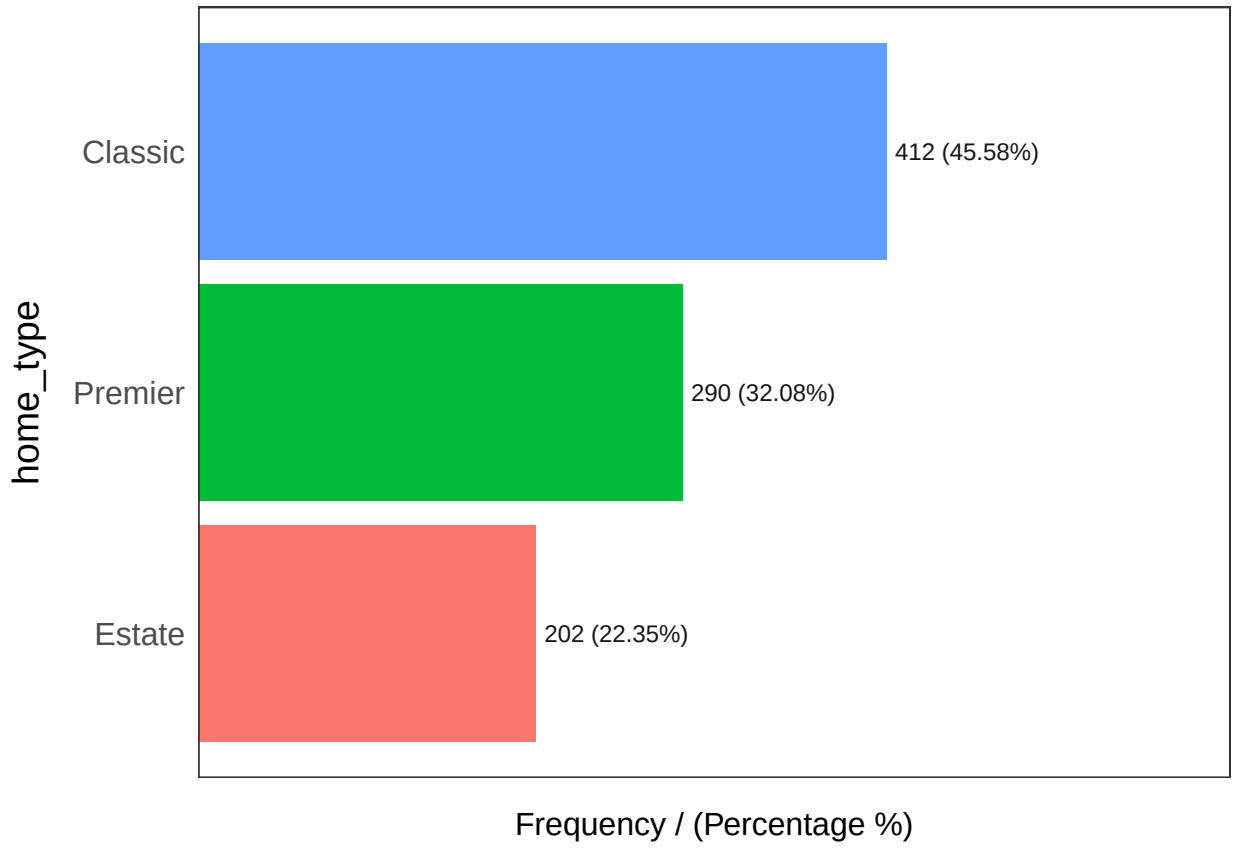
Features selected for modeling.

The dataset features for this project's modeling are limited to the sale price, house model, year built, and the square footage of living, bedroom, and garage spaces. A point is made in modeling literature that smaller datasets are easier to explore, visualize, and analyze.

```
## [1] "sale_price"  "sqft_living" "sqft_lot"    "sqft_mbr"    "sqft_garage"
## [6] "home_type"   "year_built"
```

Features bar chart.

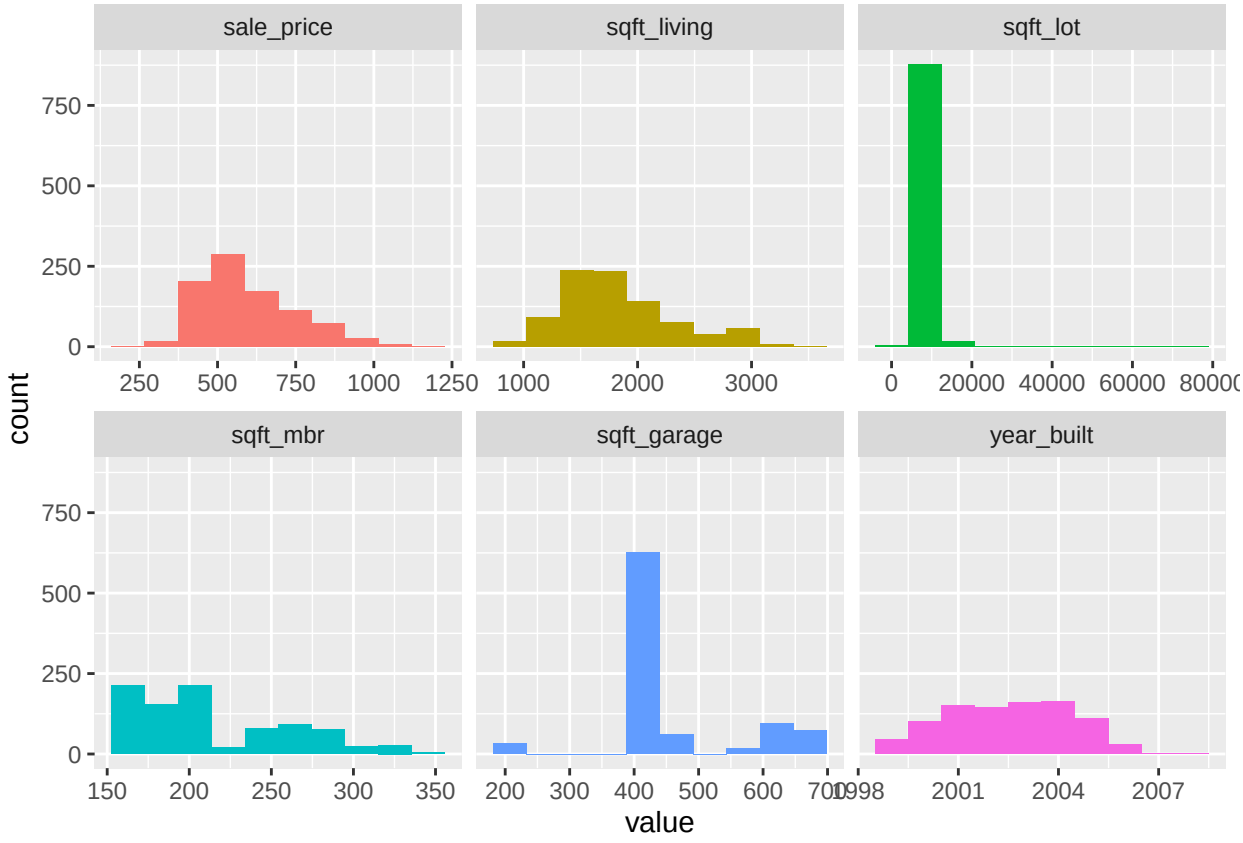
```
## Warning: `guides(<scale> = FALSE)` is deprecated. Please use `guides(<scale> =
## "none")` instead.
```



```
##   home_type frequency percentage cumulative_perc
## 1   Classic       412         46               46
## 2   Premier       290         32               78
## 3    Estate       202         22              100
```

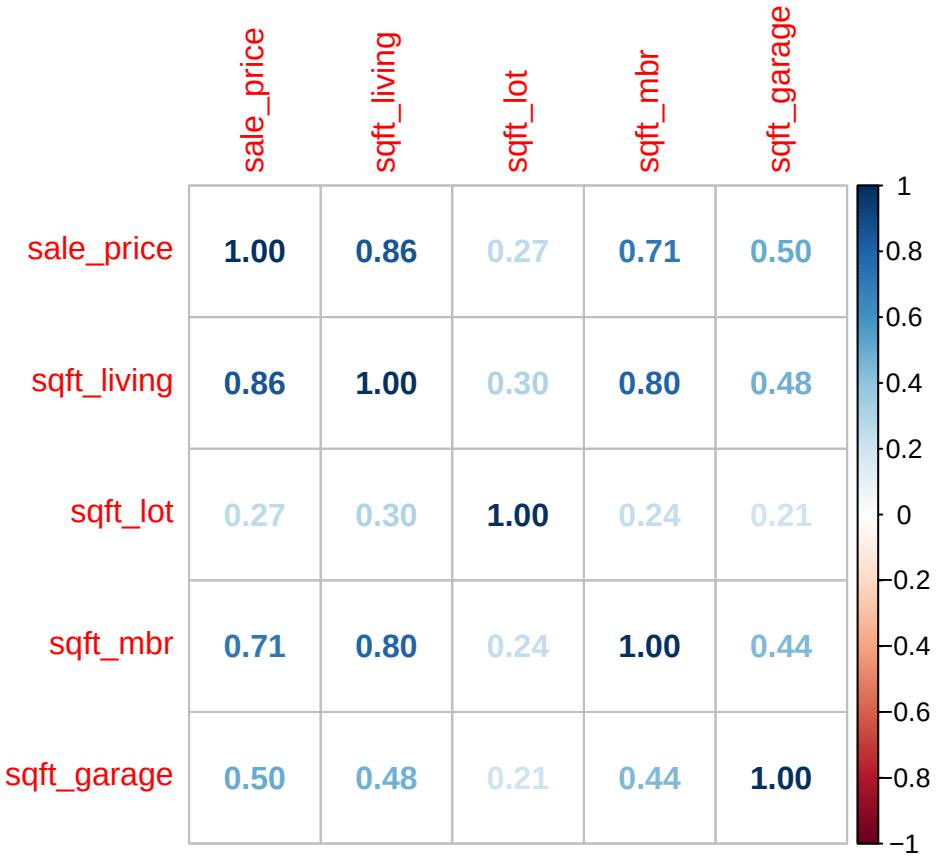
Numeric features histograms.

```
## Warning: `guides(<scale> = FALSE)` is deprecated. Please use `guides(<scale> =
## "none")` instead.
```

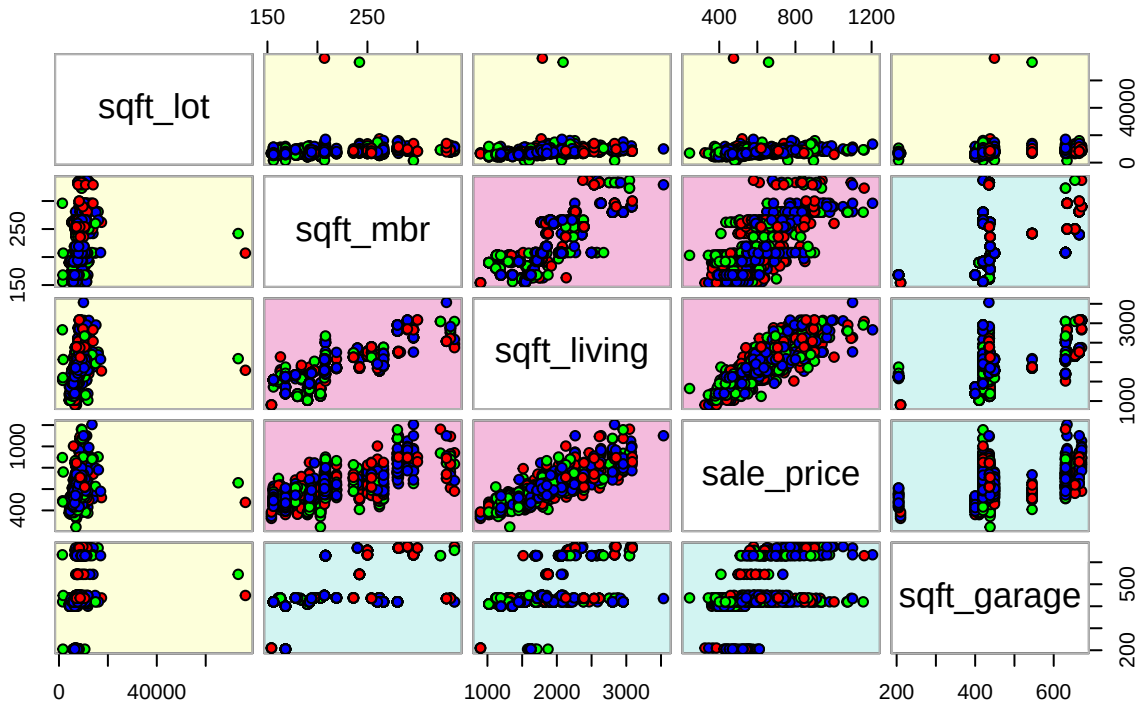


Numeric features correlation.

Feature color corresponds to the strength of the correlation, and the number is the absolute value of the correlation coefficient for the two features. Noteworthy is the relationship between the sale price and square footage of living and bedroom areas.

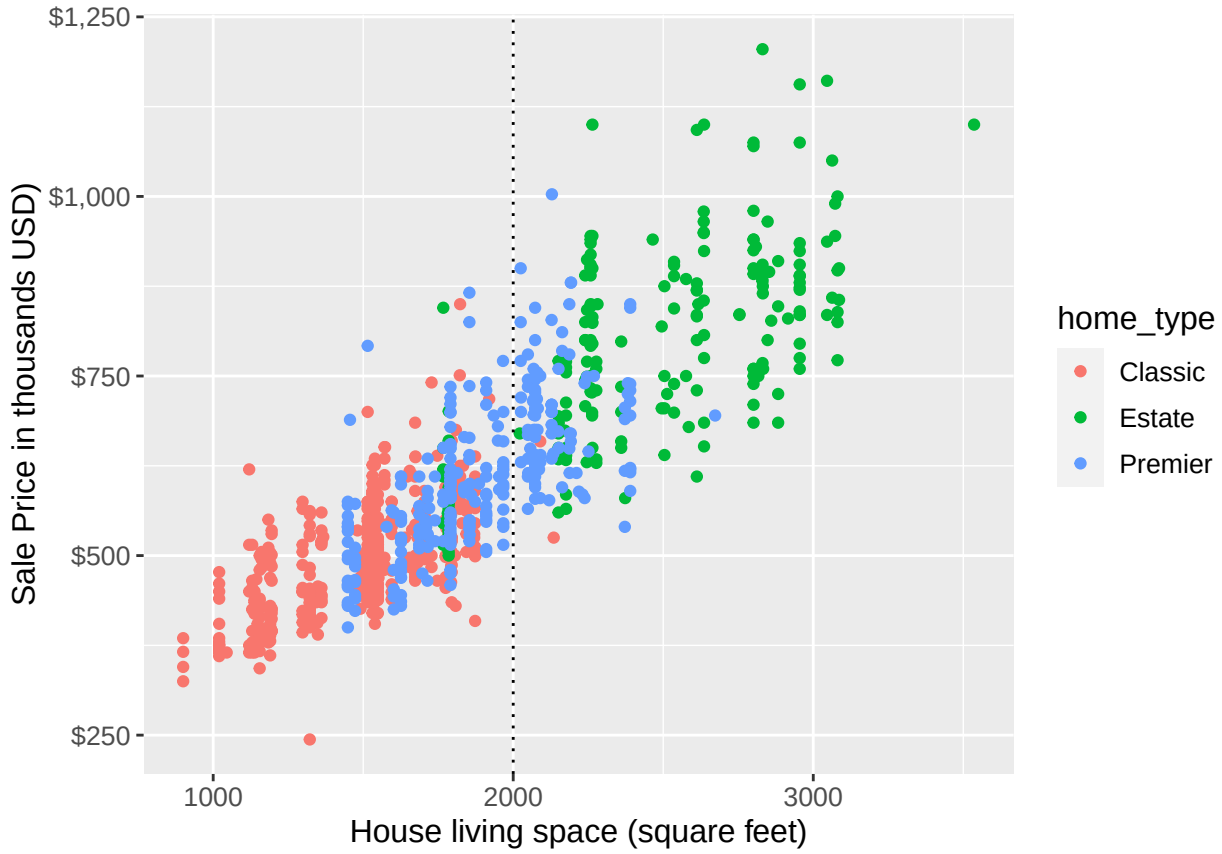


Features matrix with data points colored by group and a pairs correlation plot of numeric variables.



Feature relationships based on EDA.

Scatter plot of the correlation between house type livable space (square feet) and house sale price. The dataset feature, living space (predictor variable), is on the x-axis, and sale price (response variable) is on the y-axis. Machine learning models provide an exciting way of looking at this feature interrelationship.



Given the exploratory analysis of features, continue to modeling with two supervised machine learning models.

- A Decision Tree classification model to select which dataset feature(s) may be most important in predicting house type.
- A KNN regression model to predict home selling price based on dataset features.

The tidymodels workflows provide for combining models, but they appear here independently to facilitate model summaries and viewer feedback.

The Decision Tree classification model.

The model goal is to predict the home type based on the other feature variables in the dataset.

Modeling begins with the rsample package initial-split function to specify instructions for creating training and testing sets and sets the stratification outcome to house_type for a good representation in the testing and training data..

The training set.

```
##   sale_price sqft_living sqft_lot sqft_mbr sqft_garage home_type year_built
## 2        508        1531     8760      156         420   Classic       2004
## 4        475        1549     8934      156         420   Classic       2004
## 5        395        1190     4752      168         400   Classic       2004
## 7        500        1775     6242      262         438   Classic       2005
## 8        460        1531     7169      156         420   Classic       2004
## 9        465        1531     5811      156         420   Classic       2004
```

Training set count:

```
##   home_type   n
## 1   Classic 329
## 2    Estate 161
## 3   Premier 232
```

The testing set estimates how the trained model will perform on new data. It also guards against overfitting, where a model memorizes patterns that exist only in the training data and performs poorly on new data.

```
##    sale_price sqft_living sqft_lot sqft_mbr sqft_garage home_type year_built
## 1         420        1531     7039      156         420   Classic       2004
## 3         600        1549     7318      156         420   Classic       2004
## 6         590        1673     5955      178         438   Classic       2004
## 15        457        1349     5484      190         410   Classic       2005
## 17        443        1531     6163      156         420   Classic       2005
## 18        490        1549     6011      156         420   Classic       2005
```

Testing set count:

```
##   home_type  n
## 1   Classic 83
## 2    Estate 41
## 3   Premier 58
```

Holdout created with the rsample package to deal with hyperparameter aspects about the model that cannot be learned directly during the training process but need to be specified. * rsample package https://rsample.tidymodels.org/reference/vfold_cv.htm

```
## #  4-fold cross-validation
## # A tibble: 4 x 2
##   splits            id
##   <list>            <chr>
## 1 <split [541/181]> Fold1
## 2 <split [541/181]> Fold2
## 3 <split [542/180]> Fold3
## 4 <split [542/180]> Fold4
```

Four data folds are set aside for testing within each fold. The holdout method conducts iterations of the training data, divided into equally sized four folds for training and leaving one fold out for use as a test set.

```
## [[1]]
## <Analysis/Assess/Total>
## <541/181/722>
##
## [[2]]
## <Analysis/Assess/Total>
## <541/181/722>
##
## [[3]]
## <Analysis/Assess/Total>
## <542/180/722>
##
## [[4]]
## <Analysis/Assess/Total>
## <542/180/722>
```

Get better sense of the out-of-sample performance of the model using the training data. Cross-validation samples using v-fold cross-validation. (i.e., k-folds Chap 29, Irizarry text.)

```
##    sale_price sqft_living sqft_lot sqft_mbr sqft_garage home_type year_built
## 2         508        1531     8760      156         420   Classic       2004
## 5         395        1190     4752      168         400   Classic       2004
## 8         460        1531     7169      156         420   Classic       2004
## 9         465        1531     5811      156         420   Classic       2004
## 10        490        1190     6639      168         400   Classic       2004
## 11        550        1595     8354      184         430   Classic       2002
```

```
##    sale_price sqft_living sqft_lot sqft_mbr sqft_garage home_type year_built
## 4         475        1549     8934      156         420   Classic       2004
## 7         500        1775     6242      262         438   Classic       2005
## 22        441        1549     5662      156         420   Classic       2005
## 29        405        1148     6398      180         436   Classic       2003
## 41        450        1530     5662      156         420   Classic       2003
## 47        575        1704     8258      192         420   Classic       2001
```

Create another recipe model and workflow

Specify the model for classification with parsnip and the rpart package as the engine to tune. (parsnip package. https://www.tmwr.org/tuning.html)

```
## Decision Tree Model Specification (classification)
##
## Computational engine: rpart
```

Make workflow

```
## == Workflow ====================================================================
## Preprocessor: Recipe
## Model: decision_tree()
```

```
##
## -- Preprocessor ----------------------------------------------------------------
## 0 Recipe Steps
##
## -- Model -----------------------------------------------------------------------
## Decision Tree Model Specification (classification)
##
## Computational engine: rpart
```

Model fit and tuned with the training data cross-validation subsets.

Model performance is evaluated with cross-validation using tune and the complete training dataset.

```
## == Workflow [trained] ==========================================================
## Preprocessor: Recipe
## Model: decision_tree()
##
## -- Preprocessor ----------------------------------------------------------------
## 0 Recipe Steps
##
## -- Model -----------------------------------------------------------------------
## n= 722
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##   1) root 722 390 Classic (0.4557 0.2230 0.3213)
##     2) sqft_living< 1.6e+03 273  25 Classic (0.9084 0.0000 0.0916)
##       4) sqft_mbr< 1.9e+02 196   1 Classic (0.9949 0.0000 0.0051) *
##       5) sqft_mbr>=1.9e+02 77  24 Classic (0.6883 0.0000 0.3117)
##        10) sqft_living< 1.4e+03 51   0 Classic (1.0000 0.0000 0.0000) *
##        11) sqft_living>=1.4e+03 26   2 Premier (0.0769 0.0000 0.9231) *
##     3) sqft_living>=1.6e+03 449 240 Premier (0.1804 0.3586 0.4610)
##       6) sqft_mbr>=2.7e+02 106   0 Estate (0.0000 1.0000 0.0000) *
##       7) sqft_mbr< 2.7e+02 343 140 Premier (0.2362 0.1603 0.6035)
##        14) sqft_garage>=6.3e+02 29   0 Estate (0.0000 1.0000 0.0000) *
##        15) sqft_garage< 6.3e+02 314 110 Premier (0.2580 0.0828 0.6592)
##          30) sqft_living< 1.9e+03 189  97 Premier (0.4180 0.0952 0.4868)
##            60) sqft_mbr>=2.3e+02 37   1 Classic (0.9730 0.0000 0.0270) *
##            61) sqft_mbr< 2.3e+02 152  61 Premier (0.2829 0.1184 0.5987)
##             122) sqft_mbr< 2e+02 93  50 Classic (0.4624 0.1935 0.3441)
##               244) sqft_mbr< 2e+02 75  32 Classic (0.5733 0.0000 0.4267)
##                 488) sqft_garage>=3.1e+02 52   9 Classic (0.8269 0.0000 0.1731)
##                   976) sqft_mbr< 1.9e+02 44   1 Classic (0.9773 0.0000 0.0227) *
##                   977) sqft_mbr>=1.9e+02 8   0 Premier (0.0000 0.0000 1.0000) *
##                 489) sqft_garage< 3.1e+02 23   0 Premier (0.0000 0.0000 1.0000) *
##               245) sqft_mbr>=2e+02 18   0 Estate (0.0000 1.0000 0.0000) *
##             123) sqft_mbr>=2e+02 59   0 Premier (0.0000 0.0000 1.0000) *
##          31) sqft_living>=1.9e+03 125  10 Premier (0.0160 0.0640 0.9200)
##            62) sqft_living>=2.4e+03 8   1 Estate (0.0000 0.8750 0.1250) *
##            63) sqft_living< 2.4e+03 117   3 Premier (0.0171 0.0085 0.9744) *
```

```
## Warning: ‘pull_workflow_fit()‘ was deprecated in workflows 0.2.3.
## Please use ‘extract_fit_parsnip()‘ instead.
```

Feature importance. wf_fit output describes which features are relevant for classifying the data values. Each feature score shows a decrease in error when splitting each feature relative to others.

```
##    sqft_mbr sqft_living sqft_garage  sale_price    sqft_lot  year_built
##       315.7       306.8       168.3       152.8        70.4         8.7
```

Square footage living space (.988) is the most relevant feature for predicting home type, and it is consistent with the EDA correlation charts.

The predict function returns the predicted values.

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy multiclass     0.988
```

The model correctly predicted the right home type some 99% of the time.

The count function reveals home types correctly predicted.

```
##   home_type   n
## 1   Classic 329
## 2    Estate 161
## 3   Premier 232
```

```
## # A tibble: 3 x 2
##   .pred_class     n
##   <fct>       <int>
## 1 Classic       328
## 2 Estate        161
## 3 Premier       233
```

Predicted is one extra Estate house model and one Classic model less. The bind function binds the predicted house type to the training data, revealing the incorrectly predicted values.

```
##   sale_price sqft_living sqft_lot sqft_mbr sqft_garage home_type year_built
## 1        508        1531     8760      156         420   Classic       2004
## 2        475        1549     8934      156         420   Classic       2004
## 3        395        1190     4752      168         400   Classic       2004
## 4        500        1775     6242      262         438   Classic       2005
## 5        460        1531     7169      156         420   Classic       2004
## 6        465        1531     5811      156         420   Classic       2004
##   predicted_home_type
## 1             Classic
## 2             Classic
## 3             Classic
## 4             Classic
## 5             Classic
## 6             Classic
```

Get performance metrics based on the fit of the cross validation "resamples."

```
## # Resampling results
## # 4-fold cross-validation
## # A tibble: 4 x 4
##   splits            id    .metrics         .notes
##   <list>            <chr> <list>           <list>
## 1 <split [541/181]> Fold1 <tibble [2 x 4]> <tibble [0 x 1]>
## 2 <split [541/181]> Fold2 <tibble [2 x 4]> <tibble [0 x 1]>
## 3 <split [542/180]> Fold3 <tibble [2 x 4]> <tibble [0 x 1]>
## 4 <split [542/180]> Fold4 <tibble [2 x 4]> <tibble [0 x 1]>
```

The accuracy estimate mean of the different cross-validation folds yielded with the tune package collect_metrics function. (https://tune.tidymodels.org/reference/collect_predictions.html)

```
## # A tibble: 2 x 6
##   .metric  .estimator  mean     n std_err .config
##   <chr>    <chr>      <dbl> <int>   <dbl> <chr>
## 1 accuracy multiclass 0.970     4 0.00917 Preprocessor1_Model1
## 2 roc_auc  hand_till  0.989     4 0.00486 Preprocessor1_Model1
```

The takeaway here is that accuracy is now 97%. Reduced performance with cross-validation is not surprising.

The parsnip function specifies hyperparameter tuning with the min_n argument and rpart engine. * parsnip package hyperparameter tuning (https://www.tmwr.org/tuning.html)

```
## Decision Tree Model Specification (classification)
##
## Main Arguments:
##   min_n = tune()
##
## Computational engine: rpart
```

A newly created workflow using the categorical recipe and the tuning model:

fit created with the tune package to fit the vfold_houses cross validation samples of the training data to test different values for the min_n argument. The grid() argument specifies four values to try out. (tune package. https://www.tmwr.org/tuning.html)

The collect_metrics function shows the accuracy and the show_best function shows the min_n values for the top-performing models (those with the highest accuracy).

```
## # A tibble: 8 x 7
##   min_n .metric  .estimator  mean     n std_err .config
##   <int> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
## 1    20 accuracy multiclass 0.970     4 0.00917 Preprocessor1_Model1
## 2    20 roc_auc  hand_till  0.989     4 0.00486 Preprocessor1_Model1
## 3    26 accuracy multiclass 0.949     4 0.0263  Preprocessor1_Model2
## 4    26 roc_auc  hand_till  0.980     4 0.0119  Preprocessor1_Model2
## 5    10 accuracy multiclass 0.982     4 0.00569 Preprocessor1_Model3
## 6    10 roc_auc  hand_till  0.992     4 0.00372 Preprocessor1_Model3
## 7    35 accuracy multiclass 0.932     4 0.0425  Preprocessor1_Model4
## 8    35 roc_auc  hand_till  0.969     4 0.0226  Preprocessor1_Model4
```

```
## # A tibble: 4 x 7
##   min_n .metric  .estimator  mean     n std_err .config
```

```
##   <int> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
## 1    10 accuracy multiclass 0.982     4 0.00569 Preprocessor1_Model3
## 2    20 accuracy multiclass 0.970     4 0.00917 Preprocessor1_Model1
## 3    26 accuracy multiclass 0.949     4 0.0263  Preprocessor1_Model2
## 4    35 accuracy multiclass 0.932     4 0.0425  Preprocessor1_Model4
```

Decision Tree Model Results Discussion

The model was to predict the home type based on the other feature variables. It correctly predicted the right home type with values nearing 1.0, a 100% fit. Predictions were off by only one extra Estate house and one less Classic house. Additionally, Square footage living space (.988) is the most relevant feature for predicting home type, which is consistent with the EDA correlation charts.

Decision Trees are a powerful algorithm and a potent Machine Learning model. Professor Irizarry writes, "Trees are very easy to explain to people, and in fact, they are even easier to explain than linear regression! Some people believe that decision trees more closely mirror human decision-making than do the regression and classification approaches."

KNN Regression Modeling

The goal of the KNN regression model is to predict home selling price based on dataset features.

Like the Decision Tree model, the KNN model uses past information in the dataset to predict future observations. The goal of the KNN model is to predict house sale price as the outcome based on a set of dataset predictors

Also, like the Decision Tree model, it begins with the rsample initial_split function to create training and test sets from a stratified dataset, but in this case, the outcome variable stratified is sale price. Also, training variable names are flipped are flipped to distinguish them from those used in the Decision Tree model.

The training set.

```
##    sale_price sqft_living sqft_lot sqft_mbr sqft_garage home_type year_built
## 1         420        1531     7039      156         420   Classic       2004
## 4         475        1549     8934      156         420   Classic       2004
## 5         395        1190     4752      168         400   Classic       2004
## 8         460        1531     7169      156         420   Classic       2004
## 9         465        1531     5811      156         420   Classic       2004
## 15        457        1349     5484      190         410   Classic       2005
```

Training set count:

```
##   home_type   n
## 1   Classic 331
## 2    Estate 162
## 3   Premier 228
```

The testing set.

```
##    sale_price sqft_living sqft_lot sqft_mbr sqft_garage home_type year_built
## 3         600        1549     7318      156         420   Classic       2004
## 10        490        1190     6639      168         400   Classic       2004
## 13        570        1795     6023      262         438   Classic       2004
## 20        420        1549     5854      156         420   Classic       2005
## 24        499        1190     4799      168         400   Classic       2005
## 26        500        1674     5610      178         438   Classic       2005
```

Testing set count:

```
##   home_type  n
## 1   Classic 81
## 2    Estate 40
## 3   Premier 62
```

Five cross validation folds created from the training data for hyperparameter tuning.

Recipe function specifies sale_price as the response variable and all others as predictor variables.

Recipe prepped on the training data and applied to the test data to confirm transformations.

```
## # A tibble: 183 x 8
##    sqft_living sqft_lot sqft_mbr sqft_garage year_built sale_price
##          <dbl>    <dbl>    <dbl>       <dbl>      <dbl>      <dbl>
##  1     -0.499    -0.143   -1.57       -0.414      0.814        600
##  2     -1.57     -0.488   -1.10       -0.627      0.814        490
##  3      0.0776   -0.839    1.07       -0.226      0.814        570
##  4     -0.499    -0.943   -1.57       -0.414      1.36         420
##  5     -1.57     -1.68    -1.10       -0.627      1.36         499
##  6     -0.194    -1.10    -0.759      -0.226      1.36         500
##  7     -2.21     -1.69    -0.398      -0.520      0.267        365
##  8     -0.587    -0.280   -1.36       -0.247      0.267        510
##  9     -0.587     0.108   -1.36       -0.247      0.267        575
## 10     -0.548    -1.07    -1.57       -0.414      0.267        450
## # ... with 173 more rows, and 2 more variables: home_type_Estate <dbl>,
## #   home_type_Premier <dbl>
```

KNN regression model specified with nearest_neighbor function; neighbors set to tune function for hyperparameter tuning. mode set to regression.

Workflow created with model and recipe combined.

Hyperparameters tuning. Test values of neighbors.

```
## # A tibble: 8 x 1
##   neighbors
##       <dbl>
## 1        10
## 2        20
## 3        30
## 4        50
## 5        75
## 6       100
## 7       125
## 8       150
```

tune_grid function used with values of neighbors to determine hyperparameter optimal value.

Performance metrics.

Show top 5 best models based on rsq metric

```
## # A tibble: 5 x 7
##   neighbors .metric .estimator  mean     n std_err .config
##       <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1        30 rsq     standard   0.730     5  0.0119 Preprocessor1_Model3
## 2        20 rsq     standard   0.729     5  0.0108 Preprocessor1_Model2
## 3        50 rsq     standard   0.727     5  0.0132 Preprocessor1_Model4
## 4        75 rsq     standard   0.718     5  0.0133 Preprocessor1_Model5
## 5       100 rsq     standard   0.712     5  0.0138 Preprocessor1_Model6
```

select_best function used to select the model from tuning results with the best overall performance.

View the best model

```
## # A tibble: 1 x 2
##   neighbors .config
##       <dbl> <chr>
## 1        30 Preprocessor1_Model3
```

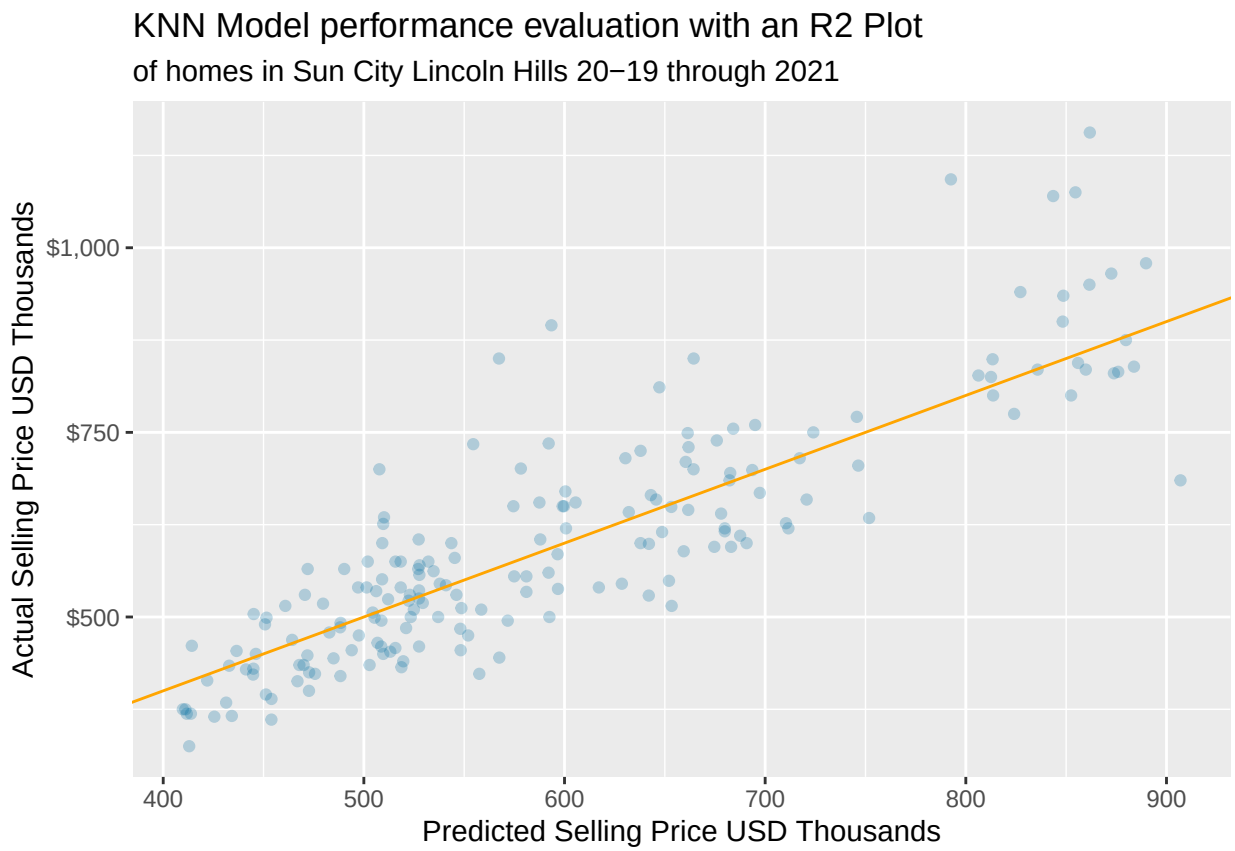
Performance metrics and predictions on the test set – rmse and rsq.

```
## # A tibble: 2 x 4
##   .metric .estimator .estimate .config
##   <chr>   <chr>          <dbl> <chr>
## 1 rmse    standard      81.7   Preprocessor1_Model1
## 2 rsq     standard       0.749 Preprocessor1_Model1
```

The test set predictions saved with the collect_predictions function. It provides a dataframe with the response variables values from the test set and a .pred column with the model predictions.

```
## # A tibble: 183 x 5
##    id               .pred  .row sale_price .config
##    <chr>            <dbl> <int>      <dbl> <chr>
##  1 train/test split  509.     3        600 Preprocessor1_Model1
##  2 train/test split  451.    10        490 Preprocessor1_Model1
##  3 train/test split  528.    13        570 Preprocessor1_Model1
##  4 train/test split  488.    20        420 Preprocessor1_Model1
##  5 train/test split  451.    24        499 Preprocessor1_Model1
##  6 train/test split  523.    26        500 Preprocessor1_Model1
##  7 train/test split  425.    28        365 Preprocessor1_Model1
##  8 train/test split  525.    37        510 Preprocessor1_Model1
##  9 train/test split  516.    38        575 Preprocessor1_Model1
## 10 train/test split  510.    41        450 Preprocessor1_Model1
## # ... with 173 more rows
```

.pred function evaluates accuracy with an R2 plot using the the homes_knn_results data frame to visualize model performance on the test data set.



(.pred, the predicted numeric outcome from the regression model: https://www.tmwr.org/performance.html).

KNN modeling results discussion.

The model goal was to predict home selling price based on dataset features, which it did with an R2 accuracy of R2 of .782 that is generally thought of as good.

Like the Decision Tree model, KNN is easy to interpret and has predictive power. However, its R2 can be a deceiving metric because it relies much on correlation rather than accuracy. General thinking of the model's predictive ability is on how well the observed and predicted values of the R2 plot agree based on the 45-degree line of agreement measure. Arguably, the farther the points are from the line, the worse the model fit.

Project Conclusions.

The machine learning goals pursued two objectives in the project.

1. The Decision Tree classification model sought to predict the home type based on the other feature variables.
2. The KNN regression model sought to predict home selling prices based on dataset features.

Both project models met their goals, and their results aligned with considerations drawn from exploratory data analysis.

As for model processing, Decision Tree preparation requires neither data normalization nor scaling, and KNN modeling requires few assumptions about the data. Additionally, both models are easy for end-users to interpret. The Decision Tree model is intuitive and easy to explain, as is the case for the intuitive algorithm of the KNN model with its straightforward interpretation.

Both models, however, have their disadvantages as well. For a Decision tree, a slight change in the data can cause a significant change in the structure of the decision tree, causing instability. So too for the KNN model. A handful of points can artificially increase R2. The R2 is a measure of correlation, not accuracy, and can be a deceiving metric. A KNN model may have a strong linear relationship between predicted and observed values, yet the predicted values may not conform to the so-called 45-degree line of agreement.

Moreover, the models may not predict well beyond the range of values input in the training data. I look forward to testing their potential with Sun City housing prices in 2022.

References.

1. *An Introduction to Statistical Learning with Applications in R* by Gareth James Data Science, Tiffany Timbers, Trevor Campbell, and Melissa Lee. 3013, https://www.statlearning.com/ Second Edition available with applications in R download: https://hastie.su.domains/ISLR2/ISLRv2_website.pdf
2. *Introduction to Data Science* by Rafael A Irizarry, https://rafalab.github.io/dsbook/
3. *Feature Engineering and Selection: A Practical Approach for Predictive Models* by Max Kuhn and Kjell Johnson, Web version updated 2019-06-21 https://bookdown.org/max/FES/
4. *Tidy Modeling with R* by Max Kuhn and Julia Silge, https://www.tmwr.org/