Code

PSTAT 131 Project Predicting California Housing Price

Christina Hefan Cui

2024-02-15

Introduction

This project aims to build multiple models to predict housing price in California based on variables from the California Housing Data set, which obtained information from the 1990 California Census.

Here is the link of the data

https://www.kaggle.com/datasets/fedesoriano/california-housing-prices-data-extra-features/data (https://www.kaggle.com/datasets/fedesoriano/california-housing-prices-data-extra-features/data)

The data pertains to the houses found in a given California district and some summary stats about them based on the 1990 census data. The columns are as follows, their names are pretty self-explanatory:

1. Median House Value: Median house value for households within a block (measured in US Dollars) [$]
2. Median Income: Median income for households within a block of houses (measured in tens of thousands of US Dollars) [10k$]
3. Median Age: Median age of a house within a block; a lower number is a newer building [years]

4. Total Rooms: Total number of rooms within a block
5. Total Bedrooms: Total number of bedrooms within a block
6. Population: Total number of people residing within a block
7. Households: Total number of households, a group of people residing within a home unit, for a block
8. Latitude: A measure of how far north a house is; a higher value is farther north [°]
9. Longitude: A measure of how far west a house is; a higher value is farther west [°]
10. Distance to coast: Distance to the nearest coast point [m]
11. Distance to Los Angeles: Distance to the centre of Los Angeles [m]
12. Distance to San Diego: Distance to the centre of San Diego [m]
13. Distance to San Jose: Distance to the centre of San Jose [m]
14. Distance to San Francisco: Distance to the centre of San Francisco [m]

For this project, I want to know what above factors affect California housing price and how we can predict it.

Loading Packages

Hide

```
# Loading all necessary packages
library(tidyverse)
library(dplyr)
library(tidymodels)
library(readr)
library(kknn)
library(glmnet)
library(corrr)
library(corrplot)
library(randomForest)
library(xgboost)
library(rpart.plot)
library(vip)
library(tidytext)
library(ggplot2)
library(visdat)
tidymodels_prefer()
```

Loading and Exploring the Data

Hide

```
set.seed(3000)
# Assigning the data to a variable
housing <- read_csv("California_Houses.csv")
# Calling head() to see the first few rows
head(housing)
```

```
## # A tibble: 6 × 14
##   Median_House_Value Median_Income Median_Age Tot_Rooms Tot_Bedrooms Population
##                <dbl>         <dbl>      <dbl>     <dbl>        <dbl>      <dbl>
## 1             452600          8.33         41       880          129        322
## 2             358500          8.30         21      7099         1106       2401
## 3             352100          7.26         52      1467          190        496
## 4             341300          5.64         52      1274          235        558
## 5             342200          3.85         52      1627          280        565
## 6             269700          4.04         52       919          213        413
## # ℹ 8 more variables: Households <dbl>, Latitude <dbl>, Longitude <dbl>,
## #   Distance_to_coast <dbl>, Distance_to_LA <dbl>, Distance_to_SanDiego <dbl>,
## #   Distance_to_SanJose <dbl>, Distance_to_SanFrancisco <dbl>
```

Hide

```
# see how many rows and columns
dim(housing)
```

```
## [1] 20640    14
```

Hide

```
summary(housing)
```

```
##  Median_House_Value Median_Income      Median_Age     Tot_Rooms
##  Min.   : 14999     Min.   : 0.4999   Min.   : 1.00   Min.   :    2
##  1st Qu.:119600     1st Qu.: 2.5634   1st Qu.:18.00   1st Qu.: 1448
##  Median :179700     Median : 3.5348   Median :29.00   Median : 2127
##  Mean   :206856     Mean   : 3.8707   Mean   :28.64   Mean   : 2636
##  3rd Qu.:264725     3rd Qu.: 4.7432   3rd Qu.:37.00   3rd Qu.: 3148
##  Max.   :500001     Max.   :15.0001   Max.   :52.00   Max.   :39320
##   Tot_Bedrooms      Population      Households       Latitude
##  Min.   :   1.0   Min.   :    3   Min.   :   1.0   Min.   :32.54
##  1st Qu.: 295.0   1st Qu.:  787   1st Qu.: 280.0   1st Qu.:33.93
##  Median : 435.0   Median : 1166   Median : 409.0   Median :34.26
##  Mean   : 537.9   Mean   : 1425   Mean   : 499.5   Mean   :35.63
##  3rd Qu.: 647.0   3rd Qu.: 1725   3rd Qu.: 605.0   3rd Qu.:37.71
##  Max.   :6445.0   Max.   :35682   Max.   :6082.0   Max.   :41.95
##    Longitude      Distance_to_coast  Distance_to_LA      Distance_to_SanDiego
##  Min.   :-124.3   Min.   :   120.7   Min.   :    420.6   Min.   :    484.9
##  1st Qu.:-121.8   1st Qu.:  9079.8   1st Qu.:  32111.3   1st Qu.: 159426.4
##  Median :-118.5   Median : 20522.0   Median : 173667.5   Median : 214739.8
##  Mean   :-119.6   Mean   : 40509.3   Mean   : 269422.0   Mean   : 398164.9
##  3rd Qu.:-118.0   3rd Qu.: 49830.4   3rd Qu.: 527156.2   3rd Qu.: 705795.4
##  Max.   :-114.3   Max.   :333804.7   Max.   :1018260.1   Max.   :1196919.3
##  Distance_to_SanJose Distance_to_SanFrancisco
##  Min.   :   569.4    Min.   :   456.1
##  1st Qu.:113119.9    1st Qu.:117395.5
##  Median :459758.9    Median :526546.7
##  Mean   :349187.6    Mean   :386688.4
##  3rd Qu.:516946.5    3rd Qu.:584552.0
##  Max.   :836762.7    Max.   :903627.7
```

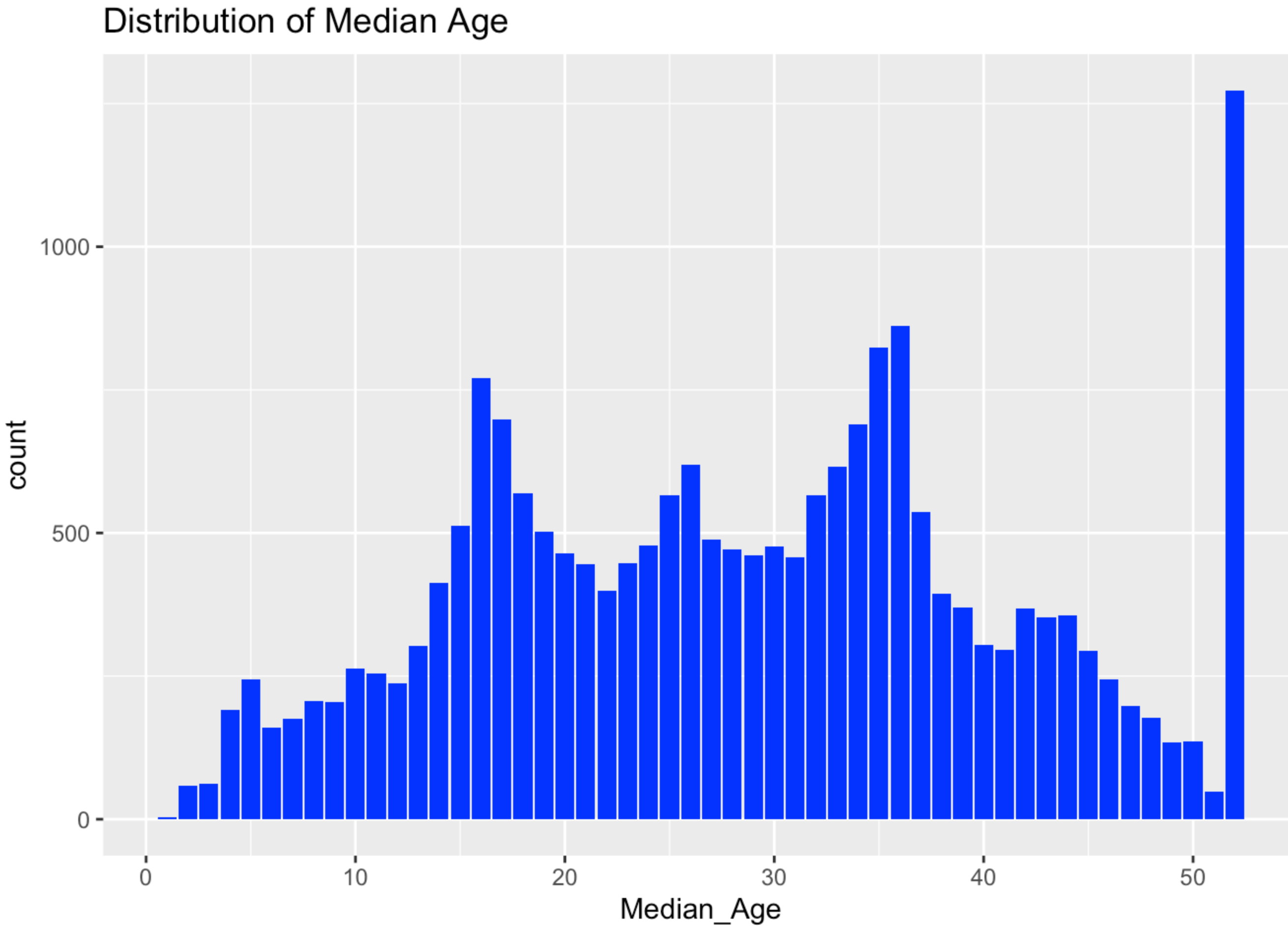
There are 20640 rows and 14 columns in the data set, meaning there are 20640 observations and 14 variables. One of those variables Median_House_Value is our response, and the rest are predictor variables.

Summary gives an overview of the data set.

Exploratory Data Analysis with Visualizations

Hide

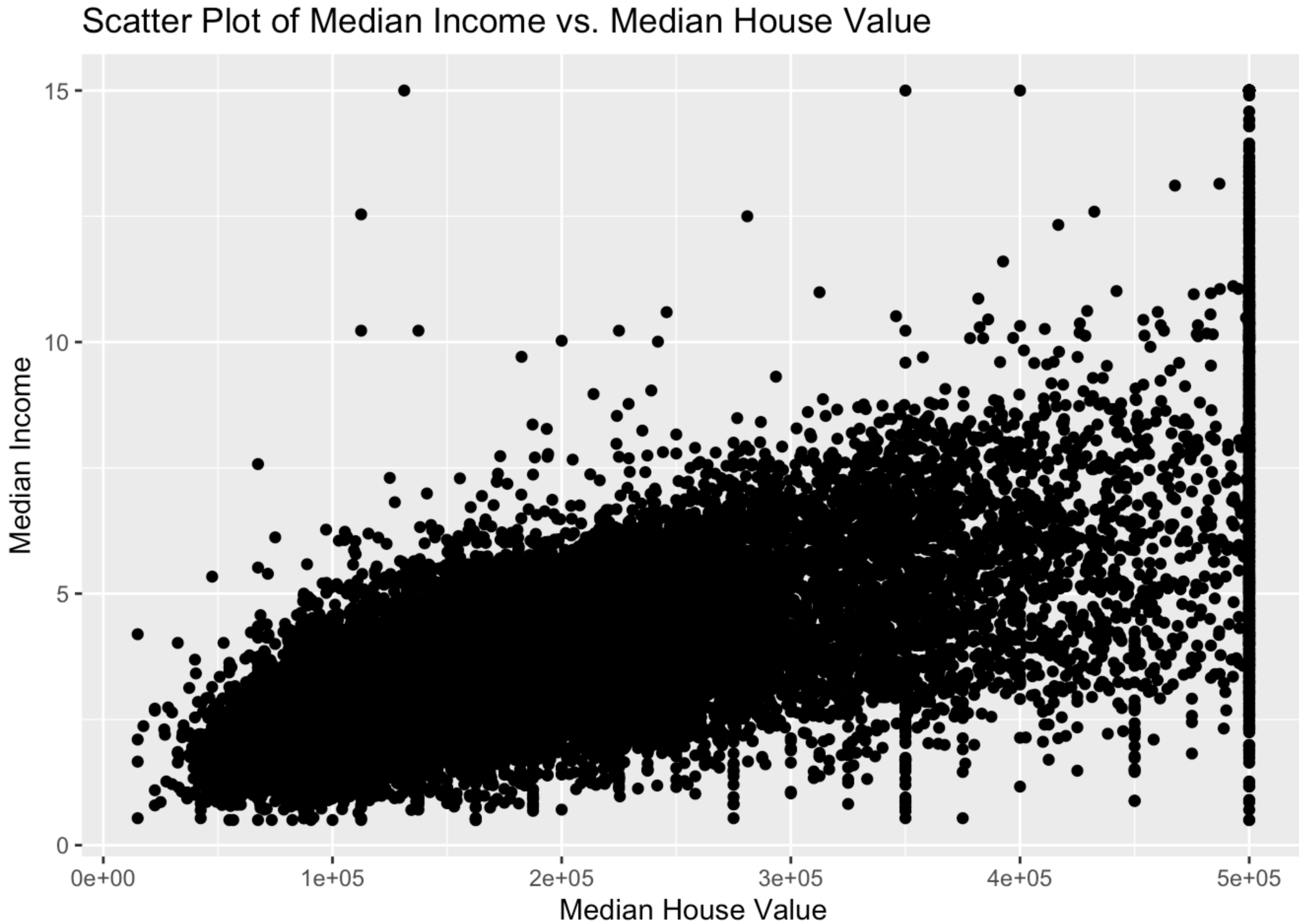
```
ggplot(housing, aes(Median_Age)) +
  geom_bar(fill = 'blue') +
  labs(title = "Distribution of Median Age")
```



Here is a histogram that shows distribution of median age. We can see there are many people older than 50 years old. Overall, the histogram is multimodal. Most appeared medium age range are 17~18, 25~26, 35~36. And potential factors can be people going to college, graduating from college, and starting their own/new family.

Hide

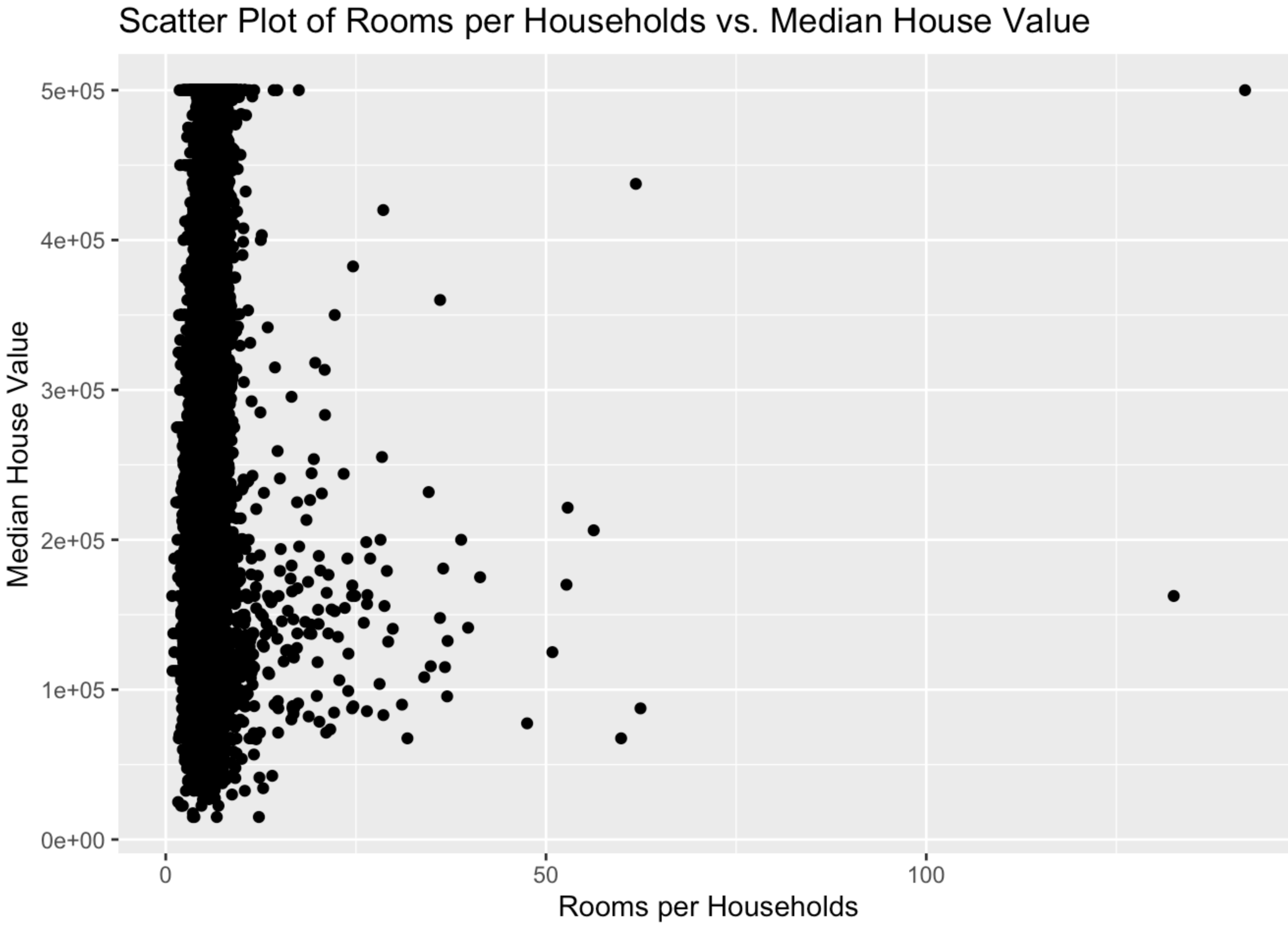
```
housing %>%
  ggplot(aes(x = Median_House_Value, y = Median_Income)) +
  geom_point() +
  labs(title = "Scatter Plot of Median Income vs. Median House Value",
       x = "Median House Value", y = "Median Income")
```



This is a scatter plot of median income vs. median house value. We can see that people have higher income tend to live at more expansive houses. That is to say, there is a positive correlation between median income and median house value.

Hide

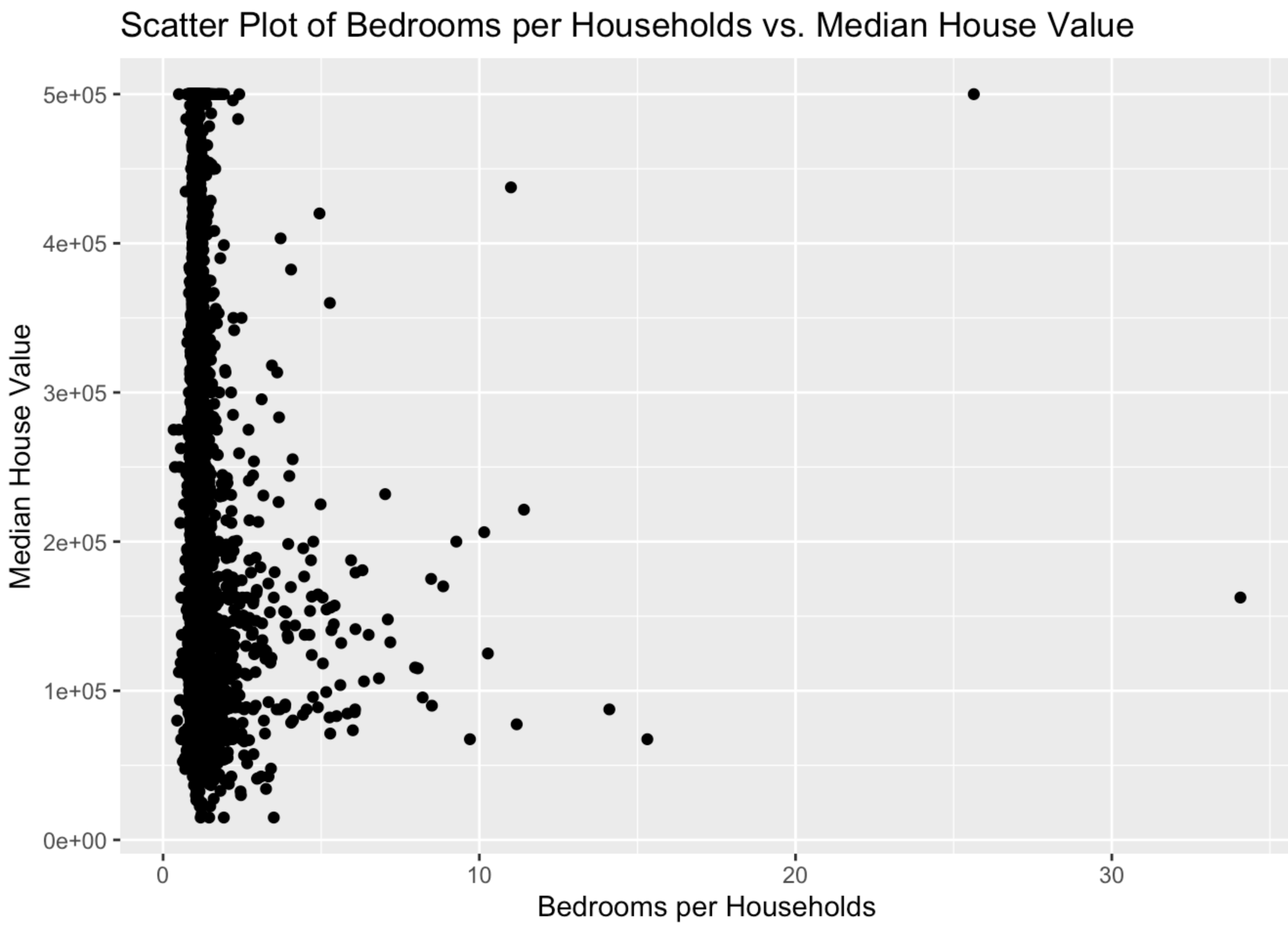
```
housing %>%
  ggplot(aes(x = Tot_Rooms/Households, y = Median_House_Value)) +
  geom_point() +
  labs(title = "Scatter Plot of Rooms per Households vs. Median House Value",
       x = "Rooms per Households", y = "Median House Value")
```



This is a scatter plot of rooms per households vs. median house value. There is no clear correlation between rooms per households and median house value. I am surprised because I think bigger houses with more rooms should be more expansive. But in the graph, there are some houses that have many rooms but not very expensive.

Hide

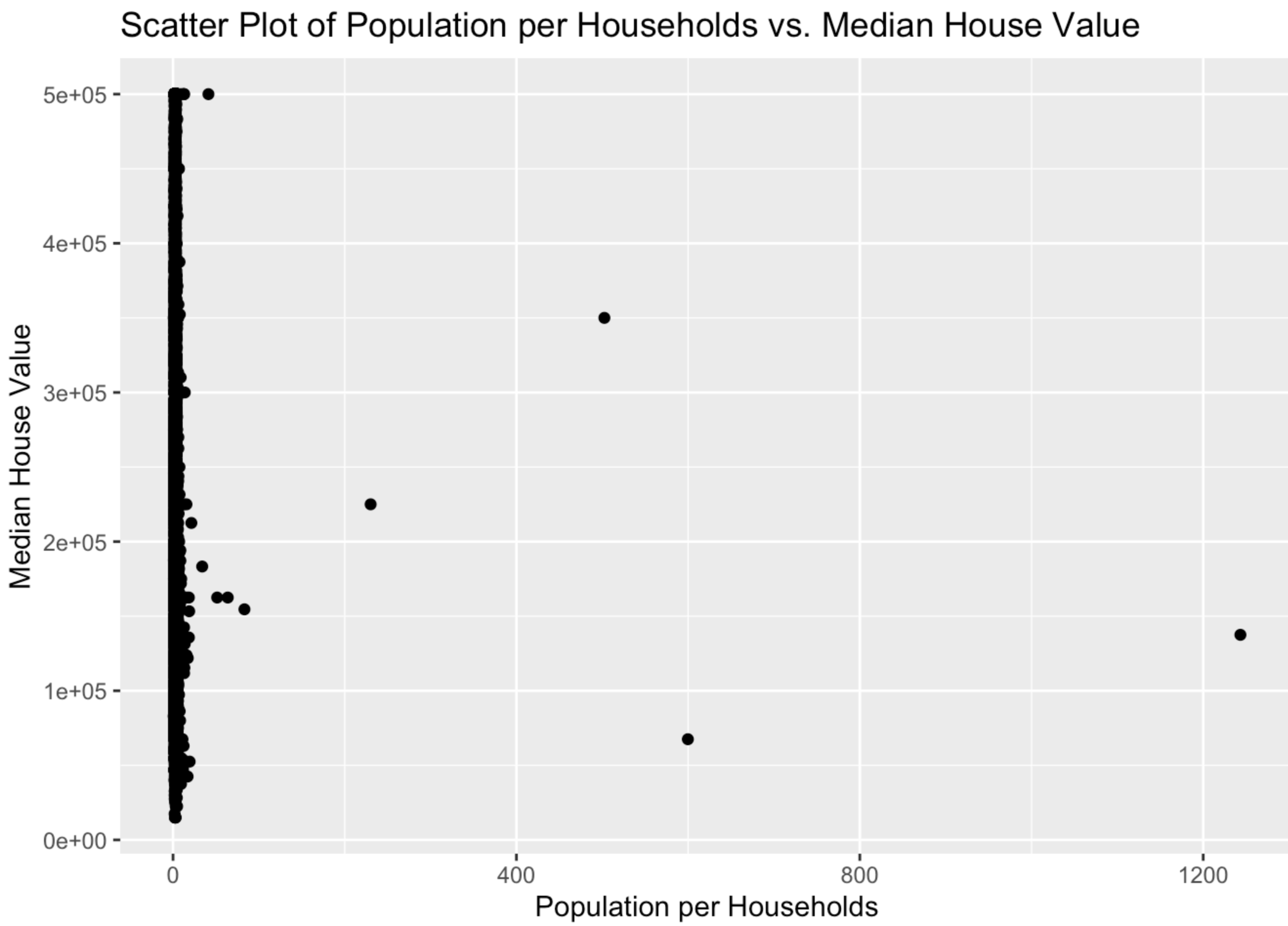
```
housing %>%
  ggplot(aes(x = Tot_Bedrooms/Households, y = Median_House_Value)) +
  geom_point() +
  labs(title = "Scatter Plot of Bedrooms per Households vs. Median House Value",
       x = "Bedrooms per Households", y = "Median House Value")
```



This is a scatter plot of bedrooms per households vs. median house value. There is no clear correlation between bedrooms per households and median house value. Similar to previous finding, I am also surprised because I think bigger houses with more bedrooms should be more expansive. But in the graph, there are some houses that have many rooms but not very expensive.

Hide

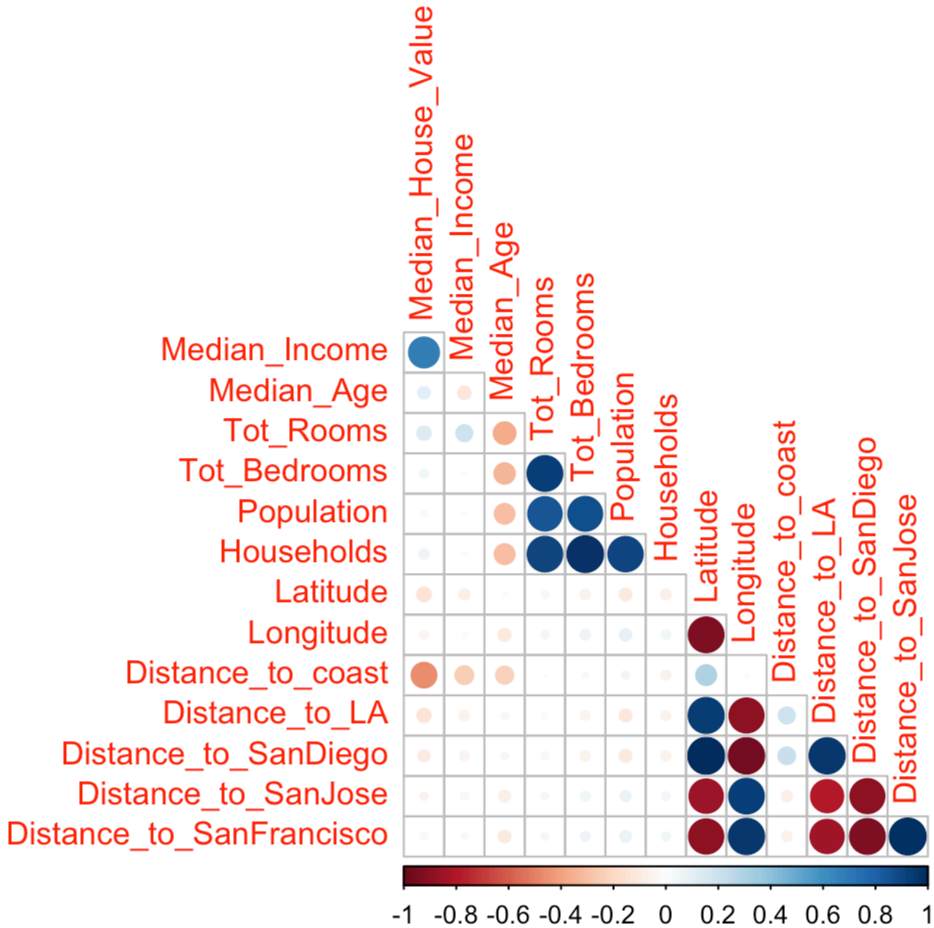
```
housing %>%
  ggplot(aes(x = Population/Households, y = Median_House_Value)) +
  geom_point() +
  labs(title = "Scatter Plot of Population per Households vs. Median House Value",
       x = "Population per Households", y = "Median House Value")
```



This is a scatter plot of population per households vs. median house value. There is no clear correlation between population per households and median house value. I am surprised about extreme values of population per households greater than 400.

Hide

```
housing %>% select(Median_House_Value, Median_Income, Median_Age, Tot_Rooms, Tot_Be
drooms, Population, Households, Latitude, Longitude, Distance_to_coast, Distance_to
_LA, Distance_to_SanDiego, Distance_to_SanJose, Distance_to_SanFrancisco) %>%
  cor(use = "pairwise.complete.obs") %>%
  corrplot(method = "circle", type = "lower", diag = FALSE)
```



This is a correlation plot. There are strong positive correlation between median income and median house value, between total rooms and total bedrooms, between population and total rooms, between households and total rooms, between total bedrooms and population, between total bedrooms and households, between households and population. And for distance to big cities, we all know that southern California cities are close to each other, so they have strong positive correlation. And southern California cities are far from northern California cities, so they have strong negative correlation. Same idea for northern California cities.

Data Splitting

The first step we have to take before fitting any models is splitting our data into a training and testing set. The training set will be used to train our models. The testing set acts as a test in the sense that our models will not be able to train on that data. So, once we fit whichever model we decide is the "best" (usually based on lowest RMSE or root mean squared error for regression) to our testing set, we will see how it truly performs on new data. By splitting our data into testing and training sets, we avoid over-fitting because the model is not using all of the available data to learn. The split I have chosen is a 80/20 split, which means 80% of the data will go towards the training set and 20% of the data will go towards the testing set. This way, most of our data is being used to train the model; however, we still have an adequate amount of data to test the model on.

Hide

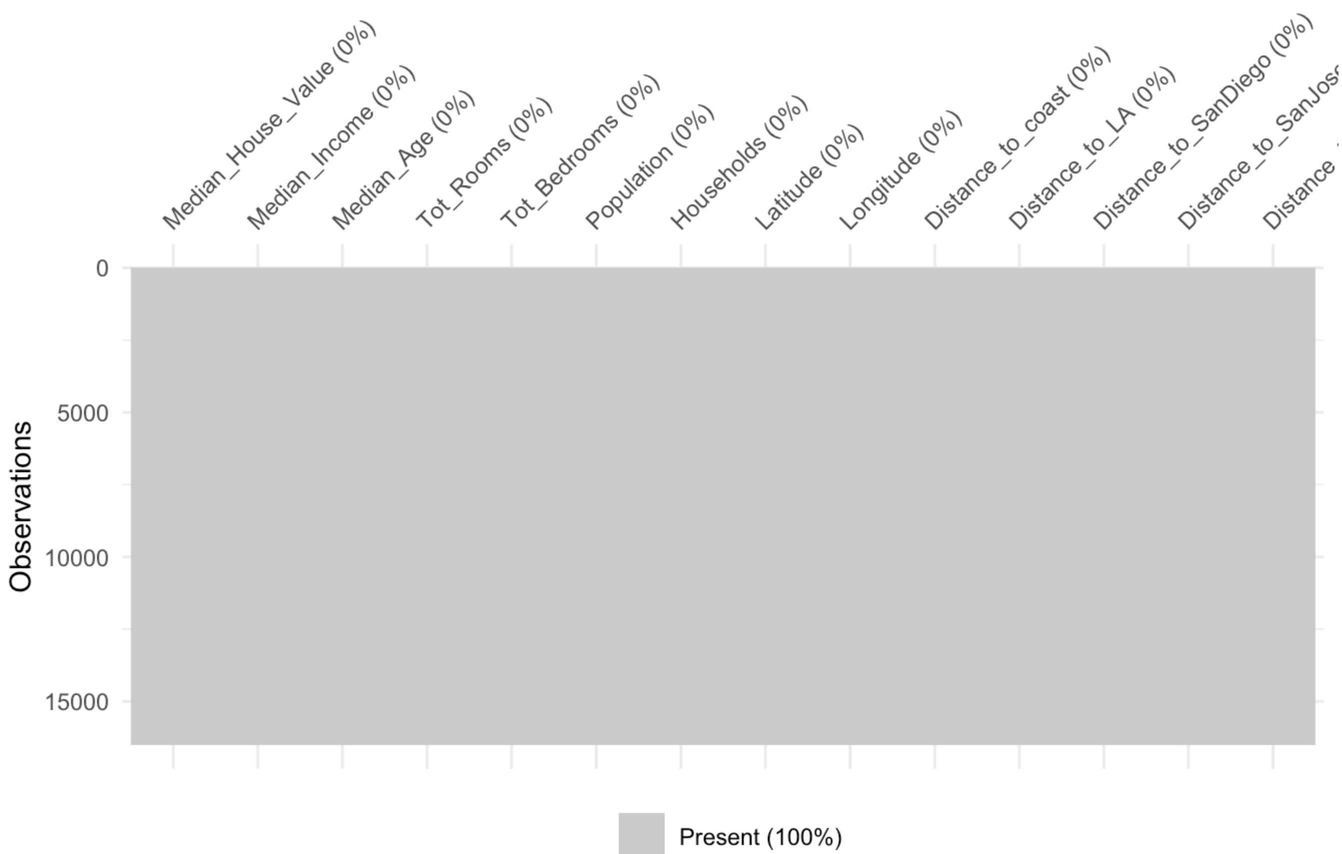
```
housing_split <- initial_split(housing, prop = 0.80,
                                strata = Median_House_Value)
housing_train <- training(housing_split)
housing_test <- testing(housing_split)
```

Missing Data Check

Before we move on to creating recipe, we must check for any missing data because that could potentially cause issues.

Hide

```
vis_miss(housing_train)
```



Hide

```
# no missing value
```

Luckily, there is no missing value, so we are good to move forward.

Creating Recipe

From our EDA, we have not find many variables that have clear effects on our response variable median house value, so we will use all 13 other variables as predictors, including Median Income, Median Age, Total Rooms, Total Bedrooms, Population, Households, Latitude, Longitude, Distance to coast, Distance to

Los Angeles, Distance to San Diego, Distance to San Jose, and Distance to San Francisco. Since there is no categorical variable, we do not need the process of step_dummy().

Hide

```
housing_recipe <- recipe(Median_House_Value ~ ., data = housing_train) %>%
   # no step() interact
  step_normalize(all_predictors())
```

Bake our recipe to make sure it is ready to use.

Hide

```
prep(housing_recipe) %>%
  bake(new_data=housing_train) %>%
  head()
```

```
## # A tibble: 6 × 14
##   Median_Income Median_Age Tot_Rooms Tot_Bedrooms Population Households Latitude
##           <dbl>      <dbl>     <dbl>        <dbl>      <dbl>      <dbl>    <dbl>
## 1        -1.14       1.86    -0.0902      0.00939     -0.362    -0.0549     1.03
## 2        -0.894      1.86    -0.439      -0.482       -0.505    -0.461      1.03
## 3        -0.776      0.982   -0.976      -0.996       -0.979    -1.01       1.04
## 4        -0.746      1.62    -0.699      -0.705       -0.722    -0.690      1.04
## 5        -1.09       1.86    -0.341      -0.279       -0.285    -0.270      1.04
## 6        -1.01       1.86    -0.222      -0.135       -0.226    -0.153      1.03
## # ℹ 7 more variables: Longitude <dbl>, Distance_to_coast <dbl>,
## #   Distance_to_LA <dbl>, Distance_to_SanDiego <dbl>,
## #   Distance_to_SanJose <dbl>, Distance_to_SanFrancisco <dbl>,
## #   Median_House_Value <dbl>
```

K-Fold Cross-Validation

We will create 5 folds to conduct k-fold (10-fold in our case) stratified cross validation. K-fold cross validation is done by splitting the data into k folds as described above with each fold being a testing set with the other k-1 folds being the training set for that fold. Then, whichever model we are fitting is fit to each training set and tested on the corresponding testing set.

We use k-fold cross validation rather than simply fitting and testing models on the entire training set because cross validation provides a better estimate of the testing accuracy. It is better to take the mean accuracy from several samples instead of just one accuracy from one sample because, as n increases, we reduce variation.

We stratify on the outcome variable, median house value.

Hide

```
# Create 5 folds for cross-validation
housing_folds <- vfold_cv(housing_train, v = 5, strata = Median_House_Value)
```

Building Models

Set up workflows for four models:

1. k-nearest neighbors with the kknn engine, tuning neighbors;
2. linear regression;
3. elastic net linear regression, tuning penalty and mixture;
4. random forest.

The steps of building these models are similar. We first built the model by specifying what type of model it is and using the corresponding function. Then we set the corresponding engine and set its mode to “regression”. After that, we set up the workflow for the model, add our baked housing recipe and the corresponding model. Linear regression model is simpler model that does not require hyperparameters to be tuned. However, we need to set up the tuning grid with the parameters for k-nearest neighbors, elastic net linear regression, and random forest models. We set the ranges for how many different levels of tuning we want for each parameter. Since fitting random forest model would take longer time, I had to use small parameters for its grid. I tried to use trees(range = c(50, 500)), but it took more than 10 hours to run due to large dataset with over 20000 observations. Unfortunately I had to reduce the number of trees to range from 1 to 10 even though random forest model with large number of trees performs better.

Hide

```
# knn
knn_model <- nearest_neighbor(neighbors = tune()) %>%
  set_mode("regression") %>%
  set_engine("kknn")

knn_workflow <- workflow() %>%
  add_recipe(housing_recipe) %>%
  add_model(knn_model)

neighbors_grid <- grid_regular(neighbors(range = c(1, 10)), levels = 10)


# linear regression
lm_model <- linear_reg() %>%
  set_engine("lm") %>%
  set_mode("regression")

lm_workflow <- workflow() %>%
  add_recipe(housing_recipe) %>%
  add_model(lm_model)



# elastic net
enet_model <- linear_reg(penalty = tune(), mixture = tune()) %>%
  set_engine("glmnet") %>%
  set_mode("regression")

enet_workflow <- workflow() %>%
  add_recipe(housing_recipe) %>%
  add_model(enet_model)

elastic_grid <- grid_regular(penalty(), mixture(range = c(0, 1)), levels = 10)


# random forest
rf_model <- rand_forest(mtry = tune(),
                           trees = tune(),
                           min_n = tune()) %>%
  set_engine("ranger", importance = "impurity") %>%
  set_mode("regression")

rf_workflow <- workflow() %>%
  add_model(rf_model) %>%
  add_recipe(housing_recipe)

rf_grid <- grid_regular(mtry(range = c(1, 13)), trees(range = c(1, 10)) , min_n(ran
ge = c(1, 10)), levels = 8)
```

Fitting all the models

Next, We fit all four models with corresponding workflow to the housing training dataset. We need add grid and cross validation folds for tuning models.

Hide

```
# knn
knn_tune_res <- tune_grid(
  object = knn_workflow,
  resamples = housing_folds,
  grid = neighbors_grid
)
```

Hide

```
# linear regression
lm_fit <- fit(lm_workflow, housing_train)
```

Hide

```
# elastic net
enet_tune_res <- tune_grid(
  object = enet_workflow,
  resamples = housing_folds,
  grid = elastic_grid
)
```

Hide

```
# random forest
rf_tune_res <- tune_grid(
  object = rf_workflow,
  resamples = housing_folds,
  grid = rf_grid
)
```

We save our results to RDA files and load them back for next step.

Hide

```
save(knn_tune_res, file = "knn_tune_res.rda")
save(lm_fit, file = "lm_fit.rda")
save(enet_tune_res, file = "enet_tune_res.rda")
save(rf_tune_res, file = "rf_tune_res.rda")
```

Hide

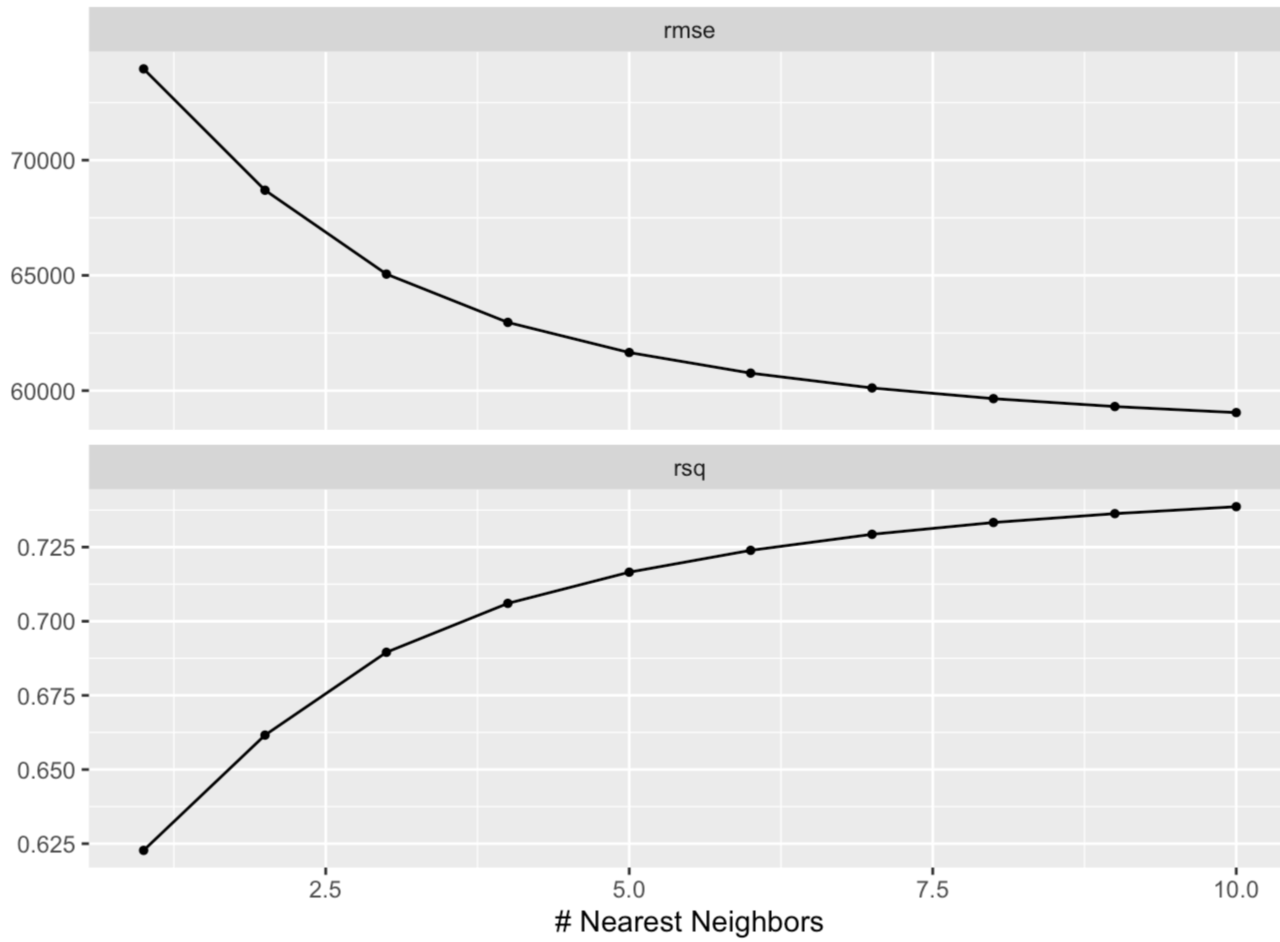
```
load("knn_tune_res.rda")
load("lm_fit.rda")
load("enet_tune_res.rda")
load("rf_tune_res.rda")
```

Model Results

We want to decide which of the models has performed the best. We use autoplot function to visualize our model tuning results. The performance of the models is measured by the RMSE (the lower the RMSE, the better the model has performed).

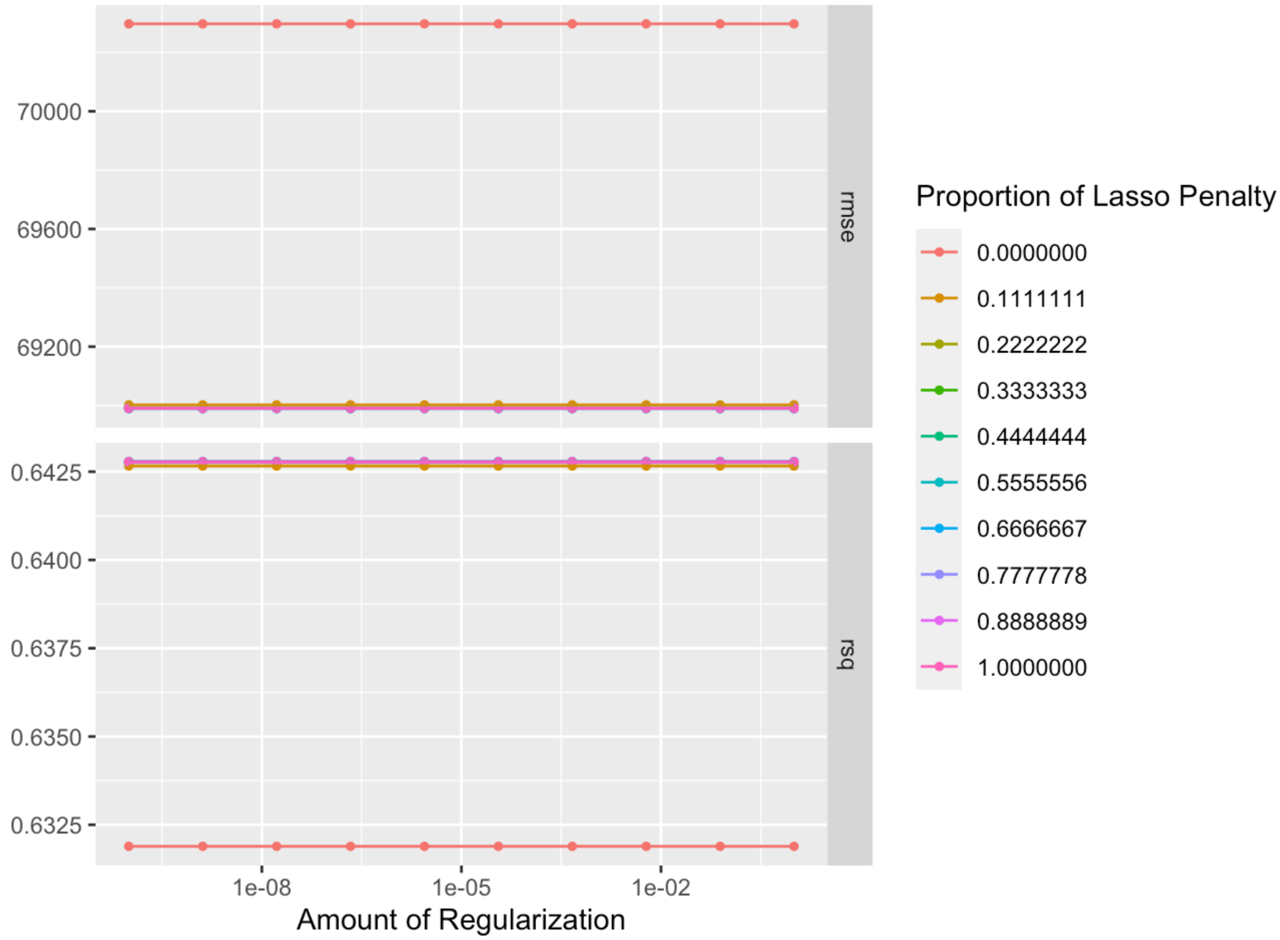
Hide

```
# add autoplot
autoplot(knn_tune_res)
```



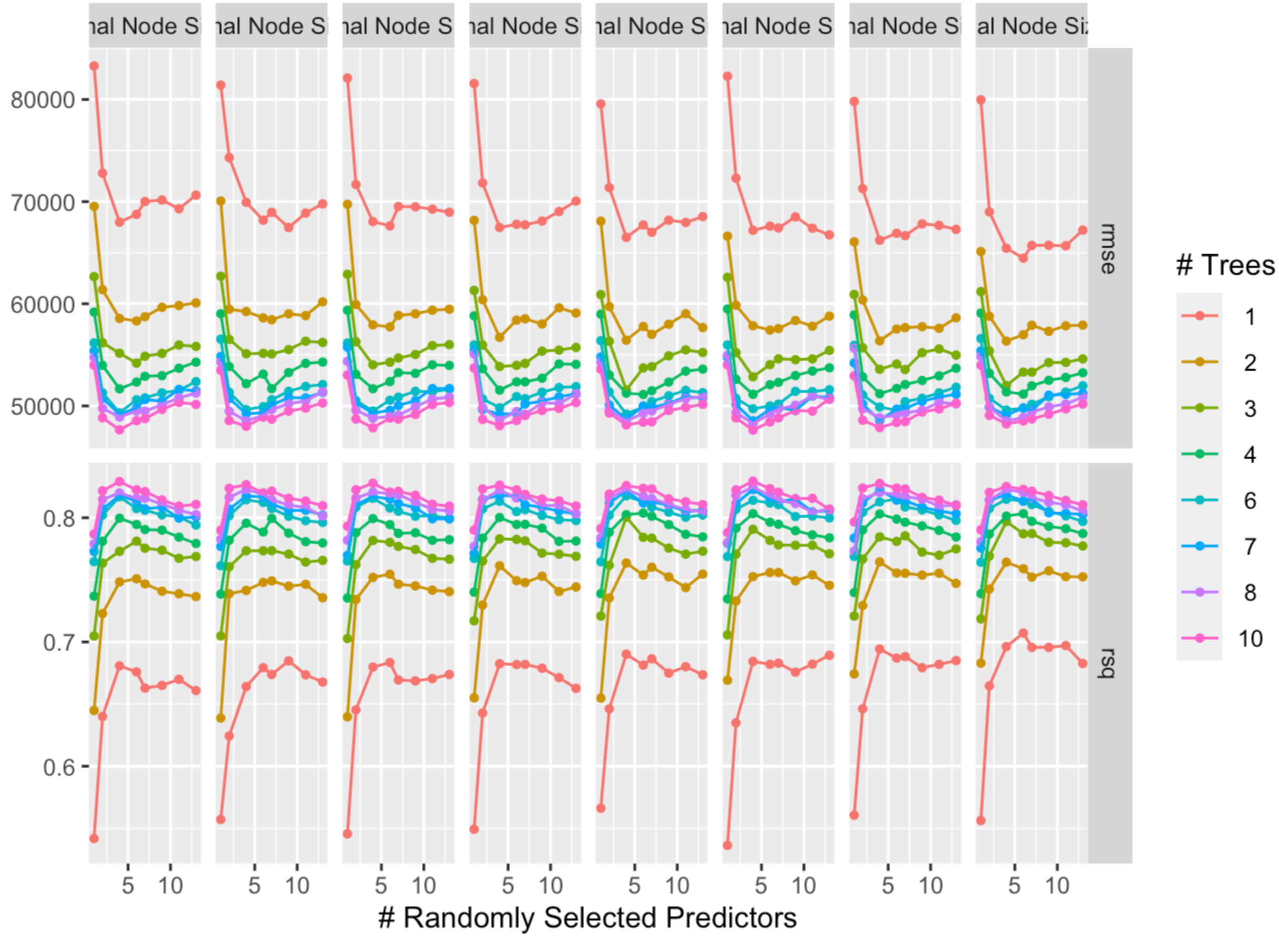
Hide

```
autoplot(enet_tune_res)
```



Hide

```
autoplot(rf_tune_res)
```



I have chosen Root Mean Squared Error (RMSE) as my metric because it works as an overall metric for all models. The RMSE is one of the most commonly used measures for evaluating the performance of regression models by showing how far the model's predictions are from the true values using Euclidian distance. So, a lower RMSE is better since that means the predicted values have a smaller distance from the actual values. Since RMSE measures distance, it is important that we normalize our data, which we did in the recipe.

Hide

```
collect_metrics(knn_tune_res) %>%
  filter(.metric == "rmse") %>%
  arrange(mean)
```

```
## # A tibble: 10 × 7
##    neighbors .metric .estimator   mean     n std_err .config
##        <int> <chr>   <chr>       <dbl> <int>   <dbl> <chr>
##  1        10 rmse    standard   59047.     5    574. Preprocessor1_Model10
##  2         9 rmse    standard   59310.     5    574. Preprocessor1_Model09
##  3         8 rmse    standard   59652.     5    571. Preprocessor1_Model08
##  4         7 rmse    standard   60116.     5    561. Preprocessor1_Model07
##  5         6 rmse    standard   60760.     5    542. Preprocessor1_Model06
##  6         5 rmse    standard   61654.     5    517. Preprocessor1_Model05
##  7         4 rmse    standard   62961.     5    478. Preprocessor1_Model04
##  8         3 rmse    standard   65057.     5    437. Preprocessor1_Model03
##  9         2 rmse    standard   68696.     5    427. Preprocessor1_Model02
## 10         1 rmse    standard   73960.     5    465. Preprocessor1_Model01
```

Hide

```
# or r-squared

collect_metrics(enet_tune_res) %>%
  filter(.metric == "rmse") %>%
  arrange(mean)
```

```
## # A tibble: 100 × 8
##          penalty mixture .metric .estimator   mean     n std_err .config
##            <dbl>   <dbl> <chr>   <chr>       <dbl> <int>   <dbl> <chr>
##  1 0.0000000001    0.667 rmse    standard   68990.     5    641. Preprocessor1_…
##  2 0.00000000129   0.667 rmse    standard   68990.     5    641. Preprocessor1_…
##  3 0.0000000167    0.667 rmse    standard   68990.     5    641. Preprocessor1_…
##  4 0.000000215     0.667 rmse    standard   68990.     5    641. Preprocessor1_…
##  5 0.00000278      0.667 rmse    standard   68990.     5    641. Preprocessor1_…
##  6 0.0000359       0.667 rmse    standard   68990.     5    641. Preprocessor1_…
##  7 0.000464        0.667 rmse    standard   68990.     5    641. Preprocessor1_…
##  8 0.00599         0.667 rmse    standard   68990.     5    641. Preprocessor1_…
##  9 0.0774          0.667 rmse    standard   68990.     5    641. Preprocessor1_…
## 10 1               0.667 rmse    standard   68990.     5    641. Preprocessor1_…
## # ℹ 90 more rows
```

Hide

```
collect_metrics(rf_tune_res) %>%
  filter(.metric == "rmse") %>%
  arrange(mean)
```

```
## # A tibble: 512 × 9
##     mtry trees min_n .metric .estimator   mean     n std_err .config
##    <int> <int> <int> <chr>   <chr>       <dbl> <int>   <dbl> <chr>
##  1     4    10     7 rmse    standard   47630.     5    749. Preprocessor1_Mode…
##  2     4    10     1 rmse    standard   47661.     5    641. Preprocessor1_Mode…
##  3     4    10     3 rmse    standard   47861.     5    591. Preprocessor1_Mode…
##  4     4    10     8 rmse    standard   47879.     5    785. Preprocessor1_Mode…
##  5     4    10     2 rmse    standard   48004.     5    671. Preprocessor1_Mode…
##  6     4    10     4 rmse    standard   48069.     5    708. Preprocessor1_Mode…
##  7     4    10     6 rmse    standard   48143.     5    414. Preprocessor1_Mode…
##  8     4    10    10 rmse    standard   48246.     5    515. Preprocessor1_Mode…
##  9     4     8     7 rmse    standard   48277.     5    644. Preprocessor1_Mode…
## 10     6    10     8 rmse    standard   48372.     5    539. Preprocessor1_Mode…
## # ℹ 502 more rows
```

Hide

```
show_best(knn_tune_res, n = 1)
```

```
## # A tibble: 1 × 7
##   neighbors .metric .estimator   mean     n std_err .config
##       <int> <chr>   <chr>       <dbl> <int>   <dbl> <chr>
## 1        10 rmse    standard   59047.     5    574. Preprocessor1_Model10
```

Hide

```
show_best(enet_tune_res, n = 1)
```

```
## # A tibble: 1 × 8
##        penalty mixture .metric .estimator   mean     n std_err .config
##          <dbl>   <dbl> <chr>   <chr>       <dbl> <int>   <dbl> <chr>
## 1 0.0000000001   0.667 rmse    standard   68990.     5    641. Preprocessor1_Mo…
```

Hide

```
show_best(rf_tune_res, n = 1)
```

```
## # A tibble: 1 × 9
##    mtry trees min_n .metric .estimator   mean     n std_err .config
##   <int> <int> <int> <chr>   <chr>       <dbl> <int>   <dbl> <chr>
## 1     4    10     7 rmse    standard   47630.     5    749. Preprocessor1_Model…
```

We compare the best results of each model and see which one performs the best! The k-nearest neighbors model has lowest mean of 59046.93 with 10 tuning neighbors. The elastic net linear regression model has lowest mean of 68989.59 with penalty = 1e-10 and mixture = 0.6667. The random forest model has lowest mean of 47630.36 with mtry = 4, trees = 10, min_n = 7. Overall, the random forest model performs the best with the smallest rmse value. And I believe using larger number of trees would perform even better.

Using the Best Model to Train and Test

Now, we take the best model from the tuned random forest and fit it to the training data. This will train that random forest one more time on the entire training data set. Once we have fit and trained the random forest on the training data, we will see how it performs on our testing data.

Hide

```
best_rf <- rf_workflow %>%
  finalize_workflow(select_best(rf_tune_res)) %>%
  fit(housing_train)

augment(best_rf, housing_test) %>%
  rmse(Median_House_Value, .pred)
```

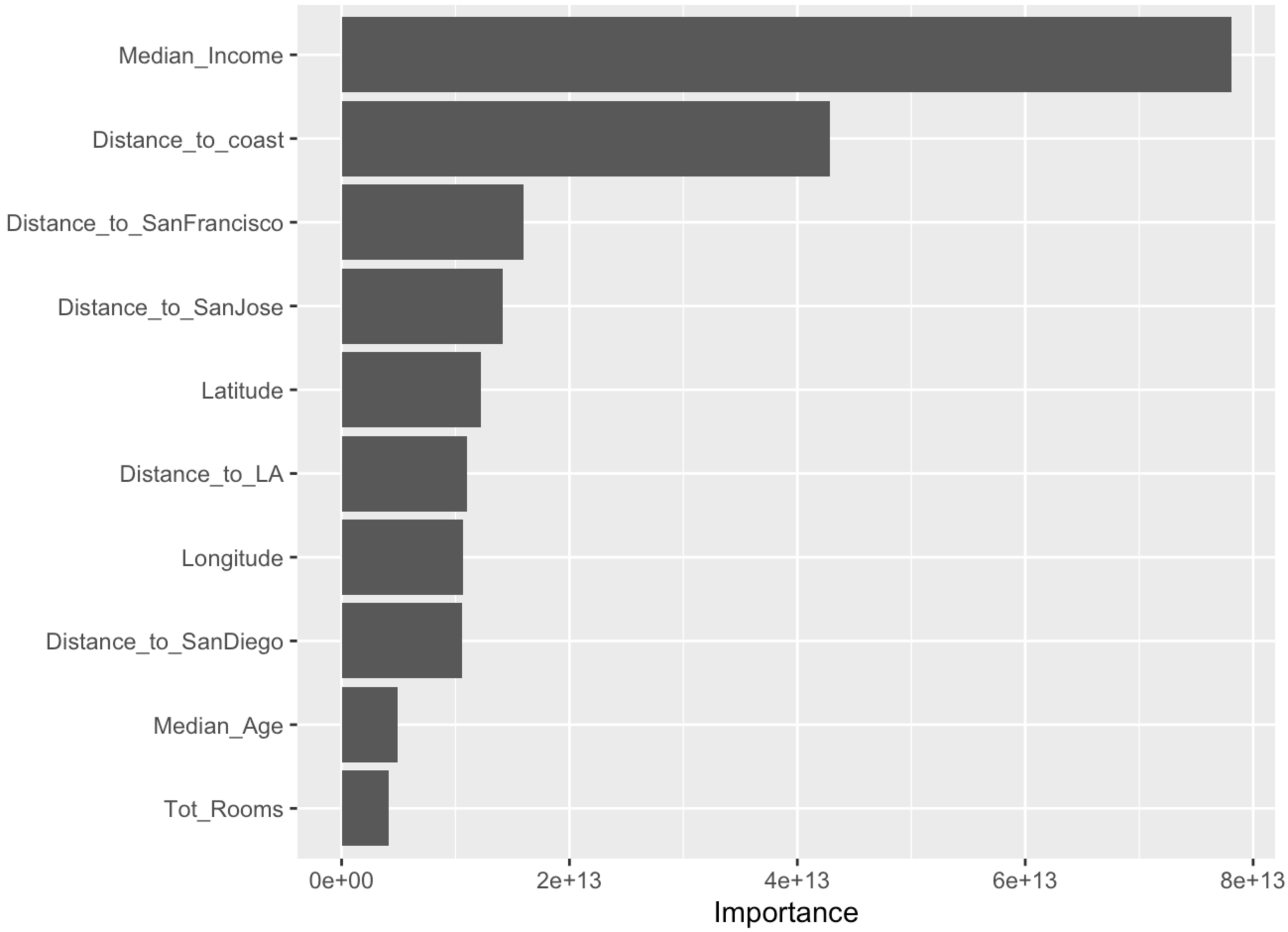
```
## # A tibble: 1 × 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard      45323.
```

The best random forest model gives us a rmse of 45369.72 for testing data, which is lower than 47630.36 for training data. This means our model performs well.

Variable Importance

Hide

```
vip(best_rf)
```



The variable importance plot shows us median income is the factor that affects California housing price the most. Secondly, distance to coast also plays an important role. Other than that, distance to major cities has effects on California housing price. These results are expected and reasonable since people have higher income can afford more expansive housing. Beside, distance to coast and big cities may affect people life styles. The consumption level of high-income people is high. We can predict California housing price depending on these factors.

Conclusion

Throughout this project, we have explored and analyzed our housing data and its variables in order to build and test a model that could know what factors make California housing so expansive. After comparing different models' performance, we conclude that the random forest model performs the best at predicting the California housing price. Unfortunately, this model was not ideal and good enough due to low trees levels for the tuning parameter. If the dataset has less observations, we could possibly use higher trees values with range from 50 to 500. However, this is a census data so it has so many observations.

My biggest takeaway from this project is to avoid the simple, traditional stereotype that houses with more rooms and more bedrooms is more expensive. After seeing the variable importance plot, I realized the locations of the houses play bigger roles than the number of total rooms or bedrooms. We may probably say median income is a subjective factor, and the location is an objective factor since median income depends on the owners of the houses and the location depends on where the houses are built. From my perspective, the location of the houses might play more important role than median income. I believe the housing price is not determined by who lives there with what kind of income level since houses are priced before they are sold. The price is determined by houses' quality and location is one of the factor. The price determines who has the ability to buy the houses. Expensive housing districts would attract people with higher income levels. Moreover, it doesn’t mean that a house with more rooms and more bedrooms is more expensive. This is related to the interior decoration and structural layout of the house. Many high income people prefer open space, meaning there are not many rooms but each space can be multi-functional and spacious. Overall, I find this topic interesting. There may be other data about houses that can be used to predict California housing prices, such as such as whether it has a designer, the quality of the furniture, the green environment, the school district, etc. That could be a classification analysis.