Predicting Housing Prices Using Machine Learning

SpringBoard DSC
Capstone Project 1
Matthew Wong

Overview

- This project focuses on predicting the selling price of the house depending on various parameters like square feet size, number of bed and bathrooms, condition, and other features.

- The data is taken from https://www.kaggle.com/harlfoxem/housesalesprediction

- Objective: **Create a machine learning model to predict housing prices**

- Analysis is done using Python and Juypter Notebook

Features Used for Analysis

List of dependent and independent variables

- We have 19 independent variables and 1 dependent variable.

Dependent	Independent
Price	sqft_living
	grade
	Lat
	Long
	yr_built
	sqft_above
	sqft_lot
	Zipcode
	View
	sqft_living15
	sqft_lot15
	Bathrooms
	Bedrooms
	Condition
	sqft_basement
	Waterfront
	yr_renovated
	Floors
	Renovated

Data Wrangling and Acquisition

This process consisted of:

- Exporting the data into the analytical environment
  - pd.read_csv()
- Checking the data for missing values
  - df.info()
- Correcting data types
  - .astype()
- Exploring the initial dataset
  - df.head()
  - df.describe()

```
df.info()
```

click to scroll output; double click to hide

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 20 columns):
date             21613 non-null object
price            21613 non-null float64
bedrooms         21613 non-null int64
bathrooms        21613 non-null float64
sqft_living      21613 non-null int64
sqft_lot         21613 non-null int64
floors           21613 non-null float64
waterfront       21613 non-null category
view             21613 non-null category
condition        21613 non-null category
grade            21613 non-null category
sqft_above       21613 non-null int64
sqft_basement    21613 non-null int64
yr_built         21613 non-null int64
yr_renovated     21613 non-null int64
zipcode          21613 non-null int64
lat              21613 non-null float64
long             21613 non-null float64
sqft_living15    21613 non-null int64
sqft_lot15       21613 non-null int64
dtypes: category(4), float64(5), int64(10), object(1)
memory usage: 2.7+ MB
```

Descriptive Statistics

```
df.describe()
```

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	sqft_above	sqft_basement
count	2.161300e+04	21613.000000	21613.000000	21613.000000	2.161300e+04	21613.000000	21613.000000	21613.000000
mean	5.400881e+05	3.370842	2.114757	2079.899736	1.510697e+04	1.494309	1788.390691	291.509045
std	3.671272e+05	0.930062	0.770163	918.440897	4.142051e+04	0.539989	828.090978	442.575043
min	7.500000e+04	0.000000	0.000000	290.000000	5.200000e+02	1.000000	290.000000	0.000000
25%	3.219500e+05	3.000000	1.750000	1427.000000	5.040000e+03	1.000000	1190.000000	0.000000
50%	4.500000e+05	3.000000	2.250000	1910.000000	7.618000e+03	1.500000	1560.000000	0.000000
75%	6.450000e+05	4.000000	2.500000	2550.000000	1.068800e+04	2.000000	2210.000000	560.000000
max	7.700000e+06	33.000000	8.000000	13540.000000	1.651359e+06	3.500000	9410.000000	4820.000000

Z-Score Analysis on Outliers

- I created a function that detected outliers. It takes in a dates and computes the z-score for each data point and returns the datapoint that is an outlier.

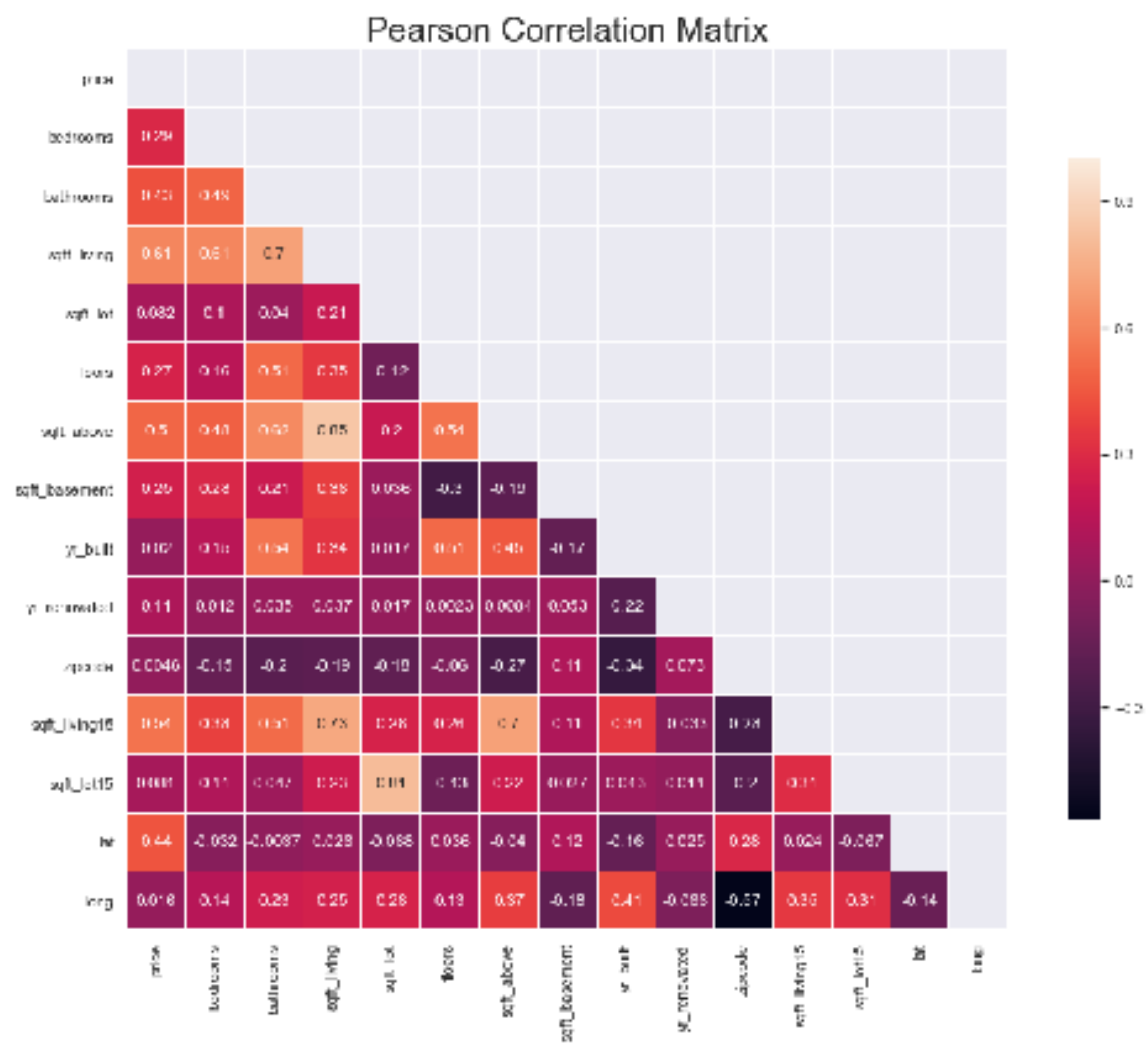
- I then filtered the dataset without the outliers using the functions minimal return value as a parameter.

Finding outliers using Z-score ¶

```
#function for detecting outliers of a dataset
def detect_outlier(data_1):
    '''Takes in a dataset and computes its z_score for each data point and returns
    if a datapoint is an outlier'''

    outliers = []

    threshold = 3 #if the z-score is greater than 3 than we can classify it as an outlier
    mean_1 = np.mean(data_1)
    std_1 = np.std(data_1)

    for y in data_1:
        z_score = (y - mean_1)/std_1
        if np.abs(z_score) > threshold:
            outliers.append(y)
    return outliers
```

Correlation Matrix

- I created a correlation matrix to quantify and summarize the relationships between the features. This matrix contains the pearson's r correlation coefficient. Which determines the relationship of the features. This gives us a better understanding how each feature relates to one another.



Performance Metrics

In order to quantify the performance of the model in the training and testing sets we need performance metrics. This is usually done by calculating some type of error, goodness of fit, or other measurement.

- **R-Squared:** the coefficient of determination for a model is a useful statistic in regression analysis, describes how ‘good’ the model is at making predictions.

- **Mean Absolute Error**: Measures the average of magnitude of the errors in a set of predictions, without considering their direction.

- **Root Mean Squared Error:** Is a quadratic scoring rule that also measures average magnitude of the error.

Training and Testing sets

A machine learning algorithm needs to be trained on a set of data to learn the relationships between different features on how these features affect the target variable. In order to do this I needed to divide the data into two sets.

- **Training set**: used to train and build the model
- **Test set:** used to compute the accuracy of our predictions

The goal is to determine if our model has learned properly from the training split.

- **Under-fit:** the model didn't learn well on the data, and can't predict outcomes, high bias
- **Overfit:** the model learned too well from the data, which prevents from being able to generalize with new data, high variance.
- The model has the right balance between variance and bias.

```
X = new_df.drop('price',axis=1)
y = new_df['price'].copy()

X_train,X_test,y_train,y_test =
        train_test_split(X,y,test_size=0.2,random_state =42)
```

Linear Regression

Our Y (a vector) is our target variable, the data that we are trying to predict. For this project our Y is the price of the house. Our X (a matrix) will be all other features that will influence Y.

- Y= Housing Prices (dependent variable, response variable)

- X = all other features (independent variables, predictors, explanatory variables)

Linear Regression

Steps:

1. Import necessary packages

2. Created a pipeline

```
Scaled_Reg = Pipeline(
    [('Scaler', StandardScaler()),
    ('Reg', LinearRegression())])
```

```
Scaled_Reg.fit(X_train,y_train)
```

   - Normalized data

   - Called linear regression classifier

3. Trained the training data and computed the result

4. Computed the performance metrics of the training and testing data

Linear Regression

Analytics Results:

Linear Regression Performance Metrics:

Accuracy training set: 0.6895

R-squared: 0.6891

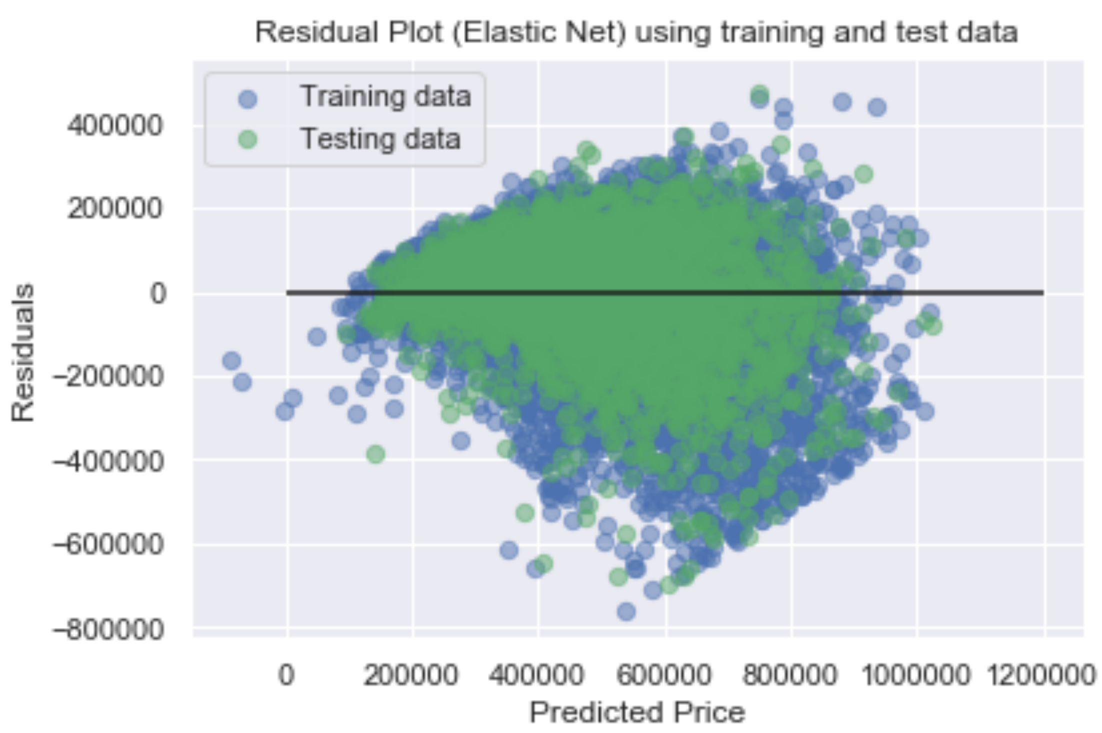
MAE: $88,839

RMSE: $120,251

Linear Regression

Residual Analysis

- Randomly distributed
- Normally distributed around zero, indicates a well fitted model



Random Forest

Features of Random Forest:

- Used for both classification and regression
- Flexible
- Ensemble method
- Highly accurate
- Does not suffer from overfitting
- Can handle missing values
- Can compute feature importance

Random Forest

Following the same pipeline as linear regression we get:

Analytics Results:

Random Forest Performance Metrics:

Accuracy training set: 0.970

R-squared: 0.8515

MAE: $56,879

RMSE: $83,091

Random Forest

Hyperparameter Tuning

- Parameters in a random forest model
    - n_estimators = number of trees in the forest
    - max_features = max number of features considered for splitting
    - max_depth = max number of levels in each decision tree
    - min_samples_split = min number of data points placed in a node before the node is split
    - n_sample_leaf = min number of data points allowed in a leaf node
    - Bootstrap = method for sampling data points (with or without replacement)
- RandomizedSearchCV
    - Sets up a grid of hyper-parameter values and select random combinations to train the model and score. The number of search iterations is set based on time/resources.

Random Forest - best parameters

Analytics Results:

Random Forest (best parameters) Performance Metrics:

Accuracy training set: 0.970

R-squared: 0.869

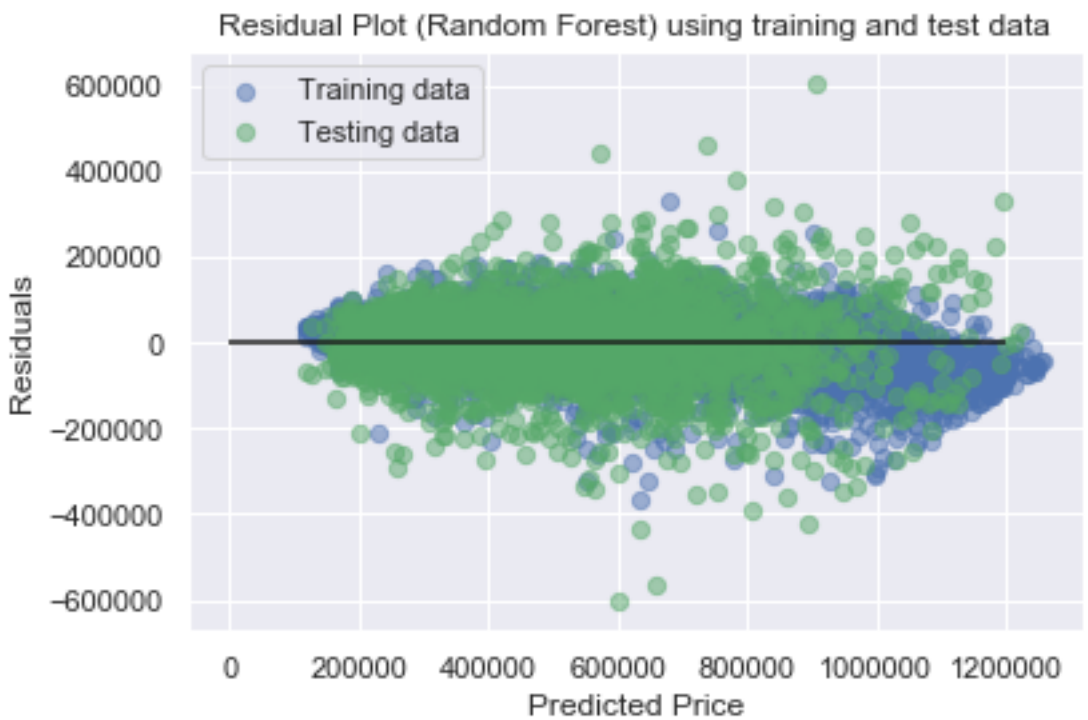
MAE: $52,810

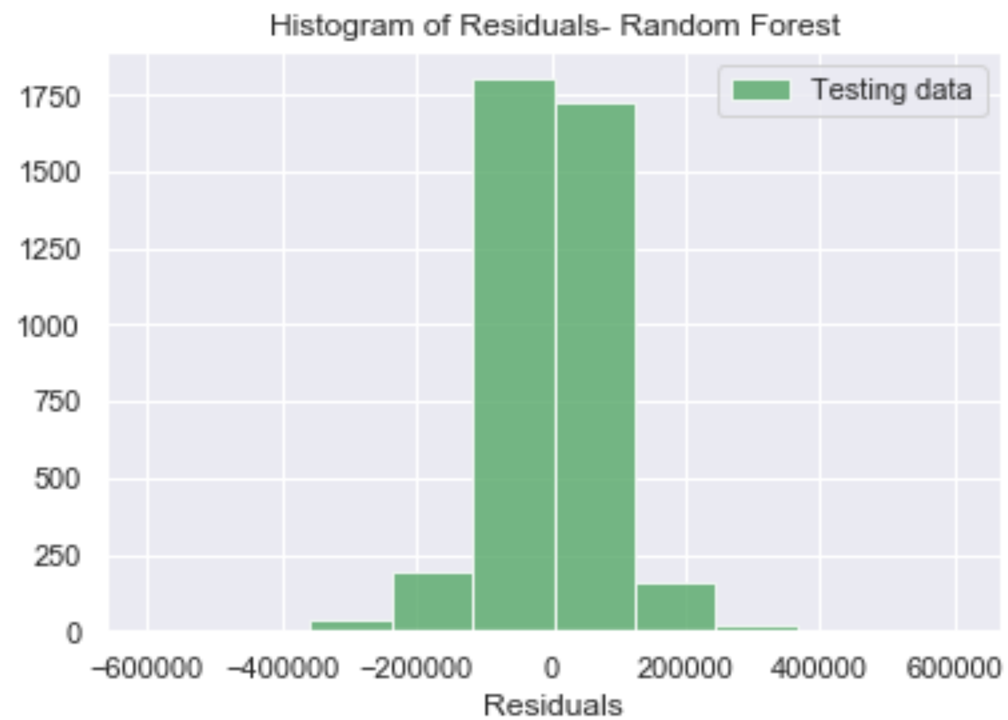
RMSE: $78,053

Random Forest

Residual Analysis

- Randomly distributed
- Normally distributed around zero, indicates a well fitted model





Gradient Boosting

Gradient Boosting

- Used for regression and classification problems
- Ensemble method
- Method of converting weak learners into strong learners

Gradient Boosting

Following the same pipeline we get:

Analytics Results:

Random Forest Performance Metrics:

Accuracy training set: 0.8567

R-squared: 0.8494

MAE: $58,694

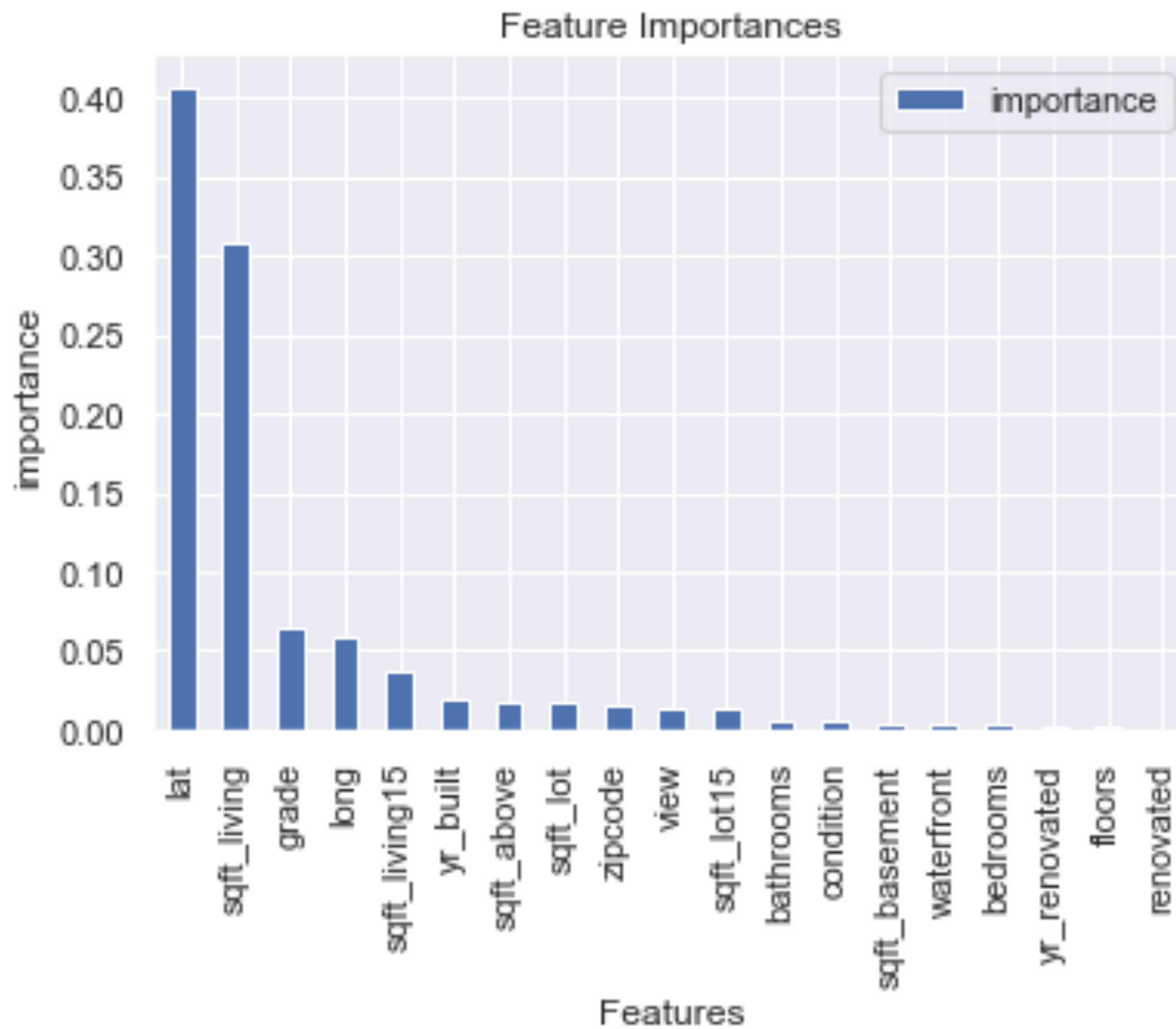
RMSE: $83,688

Results

Model	R-Squared	RMSE	MAE
Linear Regression	0.689065	120251.569707	88839.314143
Random Forest (Best Params)	0.869000	78053.481167	52810.844499
Gradient boosting	0.849402	83688.548215	58694.255076

- Random Forest with tuning is the best model for predicting price. It has the highest R-squared, and lowest RMSE and MAE.

Feature Importance



- The most valuable features are ‘lat’ and ‘sqft_living’. Both strongly impact model performance: permuting them decreases model performance by ~40% and ~31% respectively, you should take this into consideration.

Recommendations

- Use Random Forest Regressor as the model to predict price you are able to compute price with an accuracy score of ~87%.
- RMSE is 78,053. Indicating our model was able to predict the values of every house in the set within $78,053 of the mean price.
- The most valuable features are ‘lat’ and ‘sqft_living’. Both strongly impact model performance.