# Housing Prices Prediction

**Sai Sindhu Thondapu (002785016)**

**Vijaya Bhati (002754378)**

COE, Northeastern University - Toronto

**Abstract**

Housing Prices Prediction Analysis is a project that predicts California housing prices using machine learning. This study is an expansion of the midterm project, in which we just employed one model and projected accuracy using a small dataset. In this project we used a large dataset and compared different models like linear regression, decision tree and random forest to select the best among them. During the analysis phase we have learnt to draw conclusions from histograms, partition data into training and testing subsets, and explain what tail heaviness is and how to overcome it. In terms of data preparation, we talked about missing values and filled some. We processed categorical attributes and text and reshaped arrays along the way. We did some feature scaling and learned that most machine learning algorithms perform bad, when the input numerical attributes have widely varying scales. We also used the sklearn pipeline class, combined two pipelines into one, used RMSE to evaluate our models, and learnt a lot about over- and underfitting. On top of that, we employed K-fold cross validation and grid search to fine-tune our hyperparameters.

## 1. Dataset:

The dataset we will use is the "California Housing Prices" dataset from the statlib repository, which is based on 1990 census data. This dataset contains numerous learning opportunities. The dataset's prediction task will be to estimate housing prices based on numerous features. This data includes variables such as population, median income, and median dwelling price for each block group in California. Block groups are the smallest geographical units for which the US Census Bureau releases statistics. We will just call them "districts" for now.

| | Dataset Characteristics | Associated Tasks | Number of Instances | Number of Attributes |
|---|---|---|---|---|
| Dataset | Multivariate | Regression | 20640 | 10 |

Table 1: Basic feature of the dataset

| longitude | latitude | housing_median_age | total_rooms | total_bedrooms | population | households | median_income | ocean_proximity | median_house_value |
|---|---|---|---|---|---|---|---|---|---|

Table 2: Variables of the dataset

## 2. Data Analysis:

The dataset has the shape (20640, 10), with 9 columns of input variables and 1 column of targeted variables.This data is analyzed by using visual techniques. According to the dataset information below, there are 20,640 instances (entries) in the whole dataset. The collection has 20,640 instances (entries). The total_bedrooms property contains only 20,433

non-null values, indicating that this feature is unavailable in 207 districts. We can also observe that ocean_proximity is not numerical and is most likely a categorical attribute.

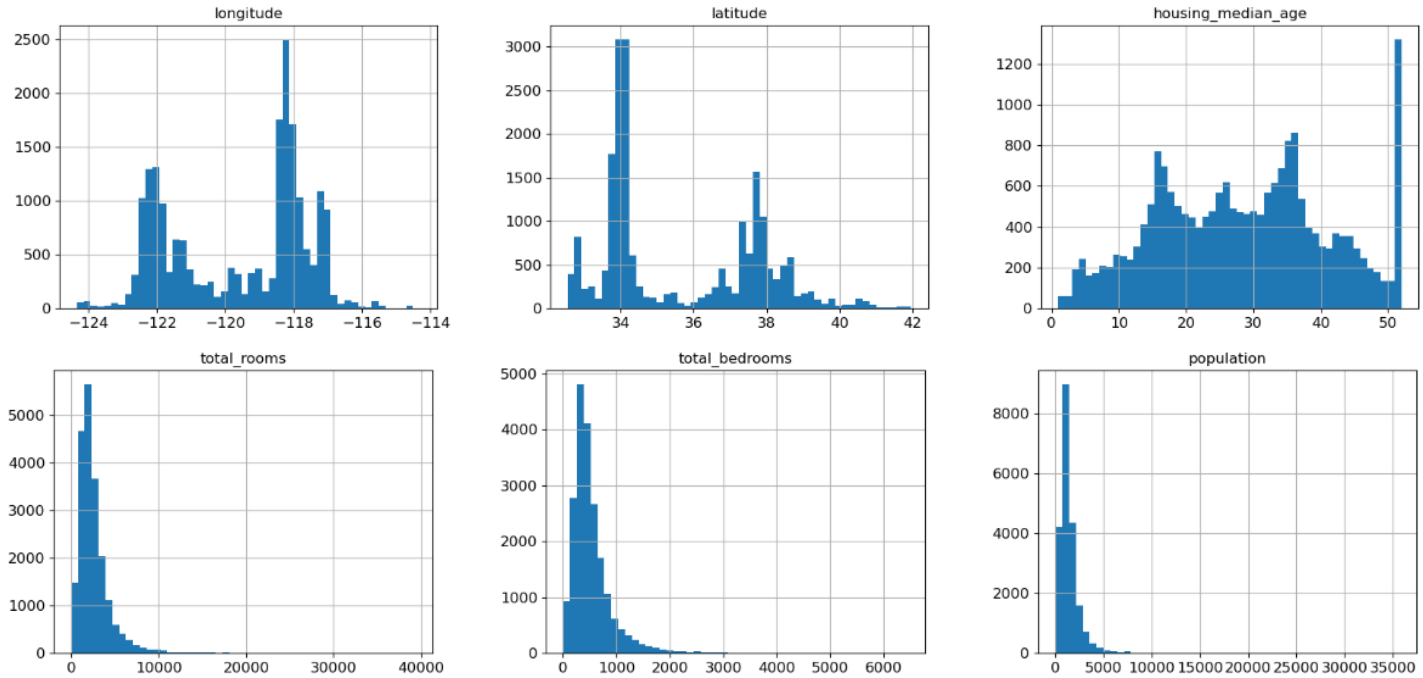
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
longitude             20640 non-null float64
latitude              20640 non-null float64
housing_median_age    20640 non-null float64
total_rooms           20640 non-null float64
total_bedrooms        20433 non-null float64
population            20640 non-null float64
households            20640 non-null float64
median_income         20640 non-null float64
median_house_value    20640 non-null float64
ocean_proximity       20640 non-null object
dtypes: float64(9), object(1)
```

Figure 1: Datatype Information

```
longitude             0
latitude              0
housing_median_age    0
total_rooms           0
total_bedrooms      207
population            0
households            0
median_income         0
median_house_value    0
ocean_proximity       0
dtype: int64
```

Figure 2: Missing data information

The presented dataset can be explored by plotting a histogram for each numerical attribute. From the below plots it is evident that many of the histograms are "tail heavy," which means they stretch more to the right of the median than to the left, making it more difficult for an algorithm to find patterns. The median_income attribute is not in US dollars because the data has been scaled and capped at 15.0001 and 0.4999. This is known as a "preprocessed attribute" and is common in machine learning. The housing_median_age and the median_house_value attributes are also capped. The fact that the median_house_value is capped may be a big issue, because this is our label (what we want to forecast), and our model may learn that the price never exceeds that limit. In this scenario, we only have the option of removing the capped one or collecting the appropriate labels. The attributes in the plots have varying scales as well. We then trained and tested the data and dropped the unwanted attributes in the dataset for further analysis.



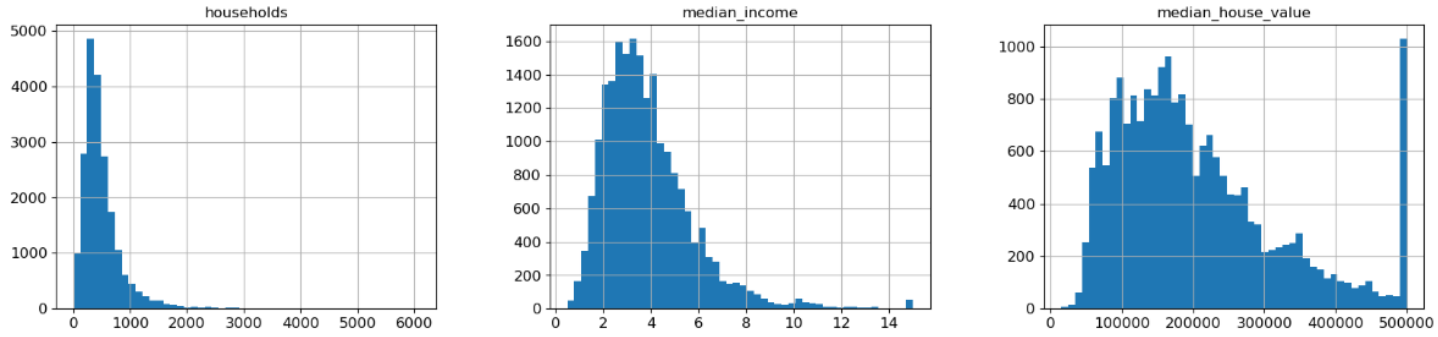



Figure 3: Histogram plots to analyze the numeric independent variable

***2.1. Extensive Data Exploration:*** Firstly, we want to visualize the geographical data using latitude and longitude. A good way to do this is to create a scatterplot of all the districts. We can clearly observe from the figure 4 that the density is high in the Bay Area, Los Angeles, San Diego, and the Central Valley surrounding Sacramento and Fresno if we are familiar with California.

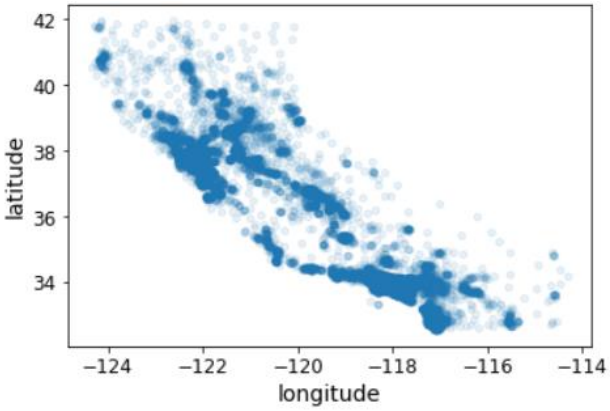



Figure 4: Scatterplot of all the districts

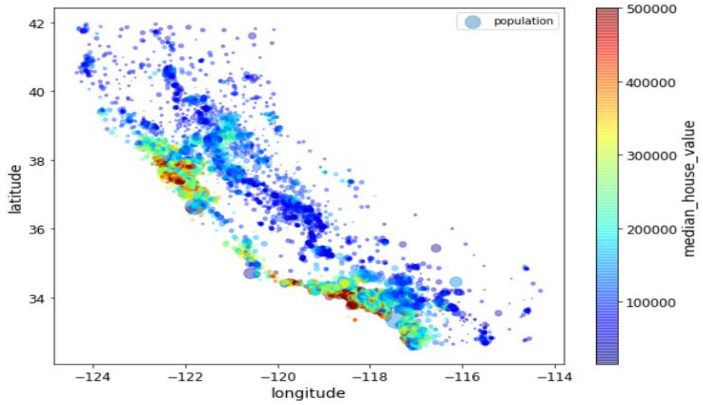
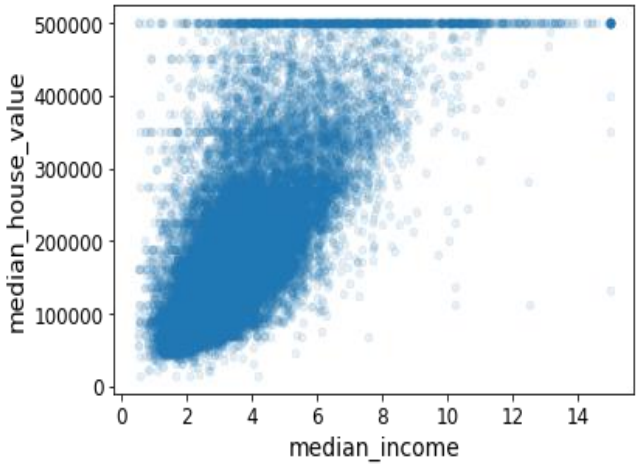



Figure 5: Scatterplot of house prices

Now let's have a look at the scatterplot of house prices in figure 5. The scatterplot shows that home costs are connected to location (near to the ocean) and population density.

***2.2. Searching for correlations:*** We have established a correlation between every attribute and examined how much each attribute is related to the median house value.




```
corr_matrix = housing.corr()
corr_matrix["median_house_value"].sort_values(ascending=False)
```

```
median_house_value          1.000000
median_income               0.688075
rooms_per_household         0.151948
total_rooms                 0.134153
housing_median_age          0.105623
households                  0.065843
total_bedrooms              0.049686
population_per_household   -0.023737
population                 -0.024650
longitude                  -0.045967
latitude                   -0.144160
bedrooms_per_room          -0.255880
Name: median_house_value, dtype: float64
```

Figure 6 & 7: Detailed & scatterplot diagrams to show median_income is correlated the most with the median house value

We can see that the median_income has the strongest correlation with the median house value. Correlation coefficients range from -1 to 1, with values close to -1 indicating a strong negative correlation, values close to 1 indicating a strong positive correlation, and values close to 0 indicating no correlation. As a result, we've created a more thorough scatterplot which can be seen in figure 6 above.

## Data Processing

Before training the model, ensure that the data is prepared. We noted before that the total_bedrooms field has some missing values. So we computed the median on the training set. As median can be computed only on numeric attributes we have dropped ocean_proximity attribute as it contains text and no numbers. And then applied the imputer to all numerical attributes. The ocean_proximity attribute is still a categorical feature, which we transformed to an integer categorical feature.

***3.1. Feature scaling:*** In histogram plots we have identified that the numerical attributes have widely varying scales in this dataset, so we required feature scaling technique. This can be achived by min-max scaling or standardization. Here we have used standardization as it is less affected by outliers and gives a more accurate outcome.

***3.2. Pipeline for transformation:*** When working with huge datasets and extensive preparation stages, pipeline can be extremely valuable because it ensures that the transformation steps are executed consistently and reproducibly. We transformed our data in this step by picking the relevant attributes, removing the rest, and transforming the resulting dataframe into a NumPy array. We have implemented two pipelines one for numeric attributes and other for categorical attributes. Using FeatureUnion we combined both the pipelines into single that handles both numerical and categorical attributes.

## Model Training

The data is preprocessed and is ready to train the model. In this section we have trained different models and tested which one model is the best. First, we normally applied the data without making any changes, later we tuned some features and tested for the best result. This is explained in a detail manner below:

First, we have tested with **Linear Regression** to check whether it gives us satisfying results. We used RSME to judge the quality of our predictions and achieved a score of 68376.5125 which can be seen in table 3. It clear from the results that the score is not a good one. Because the median_housing_values in most districts range between 120,000 and 265,000 dollars, a predicted error of $68,376 is unsatisfactory and also an example of a model underfitting the data. This indicates that either the features do not provide enough information to produce accurate predictions, or that the model is simply insufficiently powerful.

Underfitting can be fixed by feeding the model more features, picking a more powerful model, or decreasing the model's restrictions. We attempted a more powerful model because we only tested one. Now, we utilized a **DecisionTreeRegressor**, which can detect complex nonlinear correlations in data. The RSME value for this model is 0.0 which means that we strongly overfitted our data. To deal with overfitting, we employed K-fold cross validation, which used part of the training set for training and part for validation. We divided the training set into ten folds at random. The model was then trained and evaluated ten times, with each fold used once for either training or validation. This cross validation is done on both the models and the outputs can be viewed in below table 3.

From the table 3 we can say that the Decision Tree performs worse than Linear Regression. So we have tried one last model **RandomForestRegressor**. Random Forest works by training a large number of Decision Trees on random selections of the features and then averaging their predictions. The scores of random forest can be viewed in the table 3 which is lot better

than other two models. It should be noted that the score on the training set is still significantly lower than the score on the validation set, indicating that the model is overfitting the training set and that we need improve the model to address this issue. This can be done with fine tuning.

| Model Name | Accuracy |
|---|---|
| Linear Regression | RMSE: 68376.51254853733 |
| Decision Tree Regressor | RMSE: 0.0 |
| Decision Tree Regressor with K-Fold Cross-Validation | Mean: 71085.1898997<br>Standard deviation: 2431.96102969 |
| Linear regression with K-Fold Cross-Validation | Mean: 68828.9994845<br>Standard deviation: 2662.76157061 |
| Random Forest Regressor | RMSE: 22112.540875989125<br>Mean: 52710.9285312<br>Standard deviation: 2414.72717912 |

Table 3: Results of the models without tuning the features

Below are the pictorial representations of how the various algorithms perform on our model:


Figure 8: Decision Tree Regressor – K Fold Score

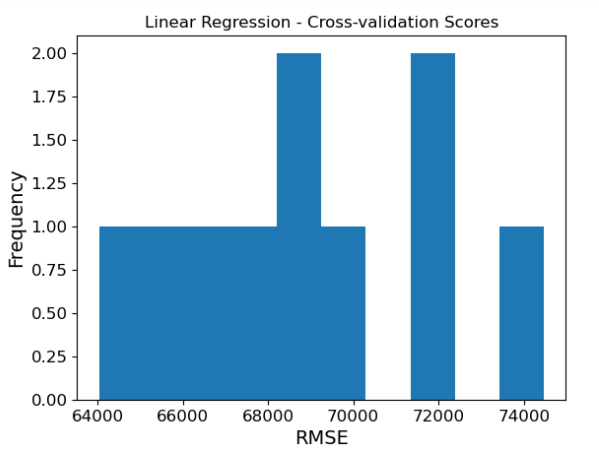



Figure 9: Linear Regression – K Fold Score

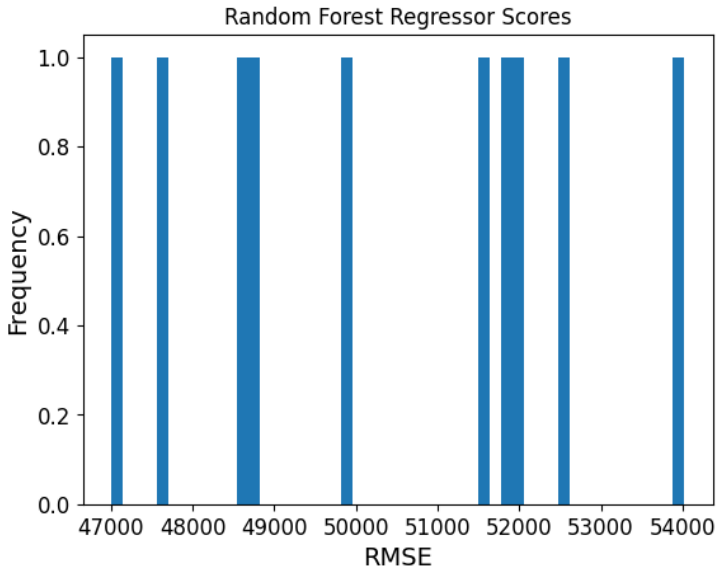



Figure 10: Random forest Regressor Score

We used Grid Search which searches for the best combination of hyperparameter values for Random Forest Regressor. Here the RMSE score for this combination obtained using Grid Search is 50,804. For best solution we have kept the max_features to 6 & n_estimators to 30 and evaluated our final model again and obtained the final prediction error of - **$48557.3361**.

```
cvres = grid_search.cv_results_
for mean_score, params in zip(cvres["mean_test_score"], cvres["params"]):
    print(np.sqrt(-mean_score), params)
```

```
64794.30269596007 {'max_features': 2, 'n_estimators': 3}
56170.628706376316 {'max_features': 2, 'n_estimators': 10}
53112.07892952066 {'max_features': 2, 'n_estimators': 30}
61836.14128922096 {'max_features': 4, 'n_estimators': 3}
54015.050600771814 {'max_features': 4, 'n_estimators': 10}
51501.312925751794 {'max_features': 4, 'n_estimators': 30}
59654.781331729675 {'max_features': 6, 'n_estimators': 3}
52654.921131020776 {'max_features': 6, 'n_estimators': 10}
50743.74265388978 {'max_features': 6, 'n_estimators': 30}
60051.26348940765 {'max_features': 8, 'n_estimators': 3}
52909.31100796091 {'max_features': 8, 'n_estimators': 10}
50800.78516244517 {'max_features': 8, 'n_estimators': 30}
63176.30025923348 {'bootstrap': False, 'max_features': 2, 'n_estimators': 3}
54596.210174483705 {'bootstrap': False, 'max_features': 2, 'n_estimators': 10}
60999.682580275425 {'bootstrap': False, 'max_features': 3, 'n_estimators': 3}
53236.2602917408 {'bootstrap': False, 'max_features': 3, 'n_estimators': 10}
60376.15876878502 {'bootstrap': False, 'max_features': 4, 'n_estimators': 3}
53060.99451893582 {'bootstrap': False, 'max_features': 4, 'n_estimators': 10}
```

Figure 11: Results from Grid Search

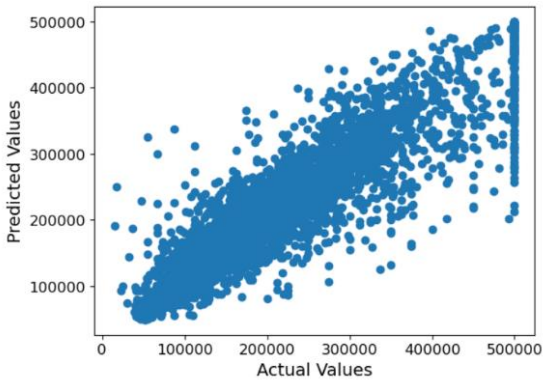



Figure 12: Predicted values vs actual values of Random Forest Regressor

## 5. Conclusion

Comparing the predicted vs. actual values, we can see that the points generally lie close to the 45-degree line, indicating that the model's predictions are relatively accurate. However, there are some points that are far from the line, which suggests that the model is not capturing all of the patterns in the data. Additionally, we can see that the model is underestimating the prices for some of the more expensive houses. This suggests that the model may not be capturing all of the factors that drive higher housing prices, such as location or unique features of the property. Overall, the model appears to be making reasonably accurate predictions, but there is still room for improvement.

## 6. Reference Links:

[1] *House Prices - Advanced Regression Techniques. (*2016). Kaggle
House Prices - Advanced Regression Techniques | Kaggle

[2] *seaborn.countplot — seaborn 0.12.2 documentation*. (n.d.).
https://seaborn.pydata.org/generated/seaborn.countplot.html

[3] *Choosing Colormaps in Matplotlib — Matplotlib 3.7.1 documentation*.(n.d.).
https://matplotlib.org/stable/tutorials/colors/colormaps.html

[4] *seaborn.countplot — seaborn 0.12.2 documentation*. (n.d.-b).
https://seaborn.pydata.org/generated/seaborn.countplot.html