House Price Prediction Model in Python

Ömer Talha BAYSAN
18011103
talha.baysan@std.yildiz.edu.tr
BLM4800 Introduction to Data Mining

***Abstract*— This document has been prepared to report a project for which a house price estimation model was created using the California Housing Prices dataset available on the Kaggle site.**

After loading the dataset, I will perform the Data Exploration, Data Preprocessing and Feature Engineering steps. I will try Linear Regression and Random Forest methods to build the model and evaluate them according to their performance.

Keywords*— *Data Exploration, Data Preprocessing, and Feature Engineering

I. Introduction

After the California Housing Prices dataset was selected, data exploration and data analysis was performed. Linear Regression and Random Forest methods were developed from data mining models to fit the data set in my analysis. Visualizations and exploratory graphs were created to help me express the findings of my analysis.

II. Loading Data Set

The Python Jupyter Notebook environment is an ideal environment for our project, as the project includes data science topics such as Data Exploration, Data Preprocessing, and Feature Engineering. We need basic data science libraries like Numpy, Pandas, Matplotlib Seaborn, Scikit-learn.

California housing prices dataset from Kaggle will be used for the project. The data pertains to the houses found in a given California district and some summary stats about them based on the 1990 census data. Be warned the data aren't cleaned so there are some preprocessing steps required! The columns are as follows, their names are pretty self explanitory: Longitude, latitude, housing_median_age, total_rooms, total_bedrooms, population, households, median_income, median_house_value, ocean_proximity.

A. About this file

1. **longitude**: A measure of how far west a house is; a higher value is farther west

   2. latitude: A measure of how far north a house is; a higher value is farther north

   3. housingMedianAge: Median age of a house within a block; a lower number is a newer building

   4. totalRooms: Total number of rooms within a block

   5. totalBedrooms: 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. medianIncome: Median income for households within a block of houses (measured in tens of thousands of US Dollars)

   9. medianHouseValue: Median house value for households within a block (measured in US Dollars)

   10. oceanProximity: Location of the house w.r.t ocean/sea

The value of the house, so this would be a regression task. An attempt was made to estimate the value of the house based on all values. The file downloaded from the site is called housing.csv[1].

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
data = pd.read_csv("housing.csv")
```

Figure 1

Libraries are added and the dataset is loaded.

III. Data Exploration

```
data.dropna(inplace = True)
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 20433 entries, 0 to 20639
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   longitude           20433 non-null  float64
 1   latitude            20433 non-null  float64
 2   housing_median_age  20433 non-null  float64
 3   total_rooms         20433 non-null  float64
 4   total_bedrooms      20433 non-null  float64
 5   population          20433 non-null  float64
 6   households          20433 non-null  float64
 7   median_income       20433 non-null  float64
 8   median_house_value  20433 non-null  float64
 9   ocean_proximity     20433 non-null  object 
dtypes: float64(9), object(1)
memory usage: 1.7+ MB
```

Figure 2

Empty values are cleared from the dataframe. We have a total of 20433 values.

The data is split into training and test data and they are split into X and Y data. This is very important because what you want to do here is to train and evaluate them all on a single dataset.

```
from sklearn.model_selection import train_test_split

x = data.drop(['median_house_value'], axis = 1)
y = data['median_house_value']
```

Figure 3

What we do is that x is the dataframe without the target variables. So, there will be data dropout and the target variable which is the median is left. It has been observed that the axis is equal to one, since we dropped the house value and one column, and Y was just the median house, on the contrary.

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2)
```

Figure 4

It is specified what percentage of the data we want to use to test the "go-to" value. 0.2, that is, 20 percent of the data is reserved for evaluation and will not touch this data for anything that will not go away. Its model is initialized so that the test set is something that can only be looked at when done. At the end of my training, the model was started with hyper parameter setting. The test data evaluated on it were used to test the data for this. That's why we won't look at this table anymore until we're done.

```
train_data
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	ocean_proximity	median_house_value
6026	-117.74	34.07	42.0	2504.0	553.0	1550.0	509.0	3.0294	INLAND	135700.0
1877	-119.96	38.94	27.0	1492.0	393.0	717.0	254.0	1.8906	INLAND	104200.0
1254	-122.01	39.21	52.0	1989.0	392.0	985.0	396.0	2.5556	INLAND	75800.0
1461	-121.99	37.97	30.0	3320.0	589.0	1470.0	543.0	4.6071	INLAND	184100.0
1183	-121.52	39.50	33.0	1462.0	241.0	569.0	231.0	3.2833	INLAND	82600.0
...	...	...	...	...	...	...	...	...	...	...
8476	-118.32	33.90	35.0	3189.0	680.0	1882.0	651.0	3.6625	<1H OCEAN	188000.0
18630	-121.93	37.04	36.0	1522.0	230.0	677.0	206.0	5.8642	<1H OCEAN	363500.0
11951	-117.44	33.94	32.0	1814.0	320.0	903.0	306.0	4.1776	INLAND	118700.0
12984	-121.30	38.67	20.0	1234.0	208.0	649.0	211.0	4.8523	INLAND	143000.0
669	-122.15	37.70	36.0	1468.0	252.0	733.0	229.0	3.4583	NEAR BAY	192600.0

16346 rows × 10 columns

Figure 5

Some basic operations have been made with the training data here. Exploration of numerical features would thus not involve ocean proximity. For example, a histogram is obtained for train data point hist distributions.

```
train_data.hist(figsize = (15,8))
```

```
array([[<Axes: title={'center': 'longitude'}>,
        <Axes: title={'center': 'latitude'}>,
        <Axes: title={'center': 'housing_median_age'}>],
       [<Axes: title={'center': 'total_rooms'}>,
        <Axes: title={'center': 'total_bedrooms'}>,
        <Axes: title={'center': 'population'}>],
       [<Axes: title={'center': 'households'}>,
        <Axes: title={'center': 'median_income'}>,
        <Axes: title={'center': 'median_house_value'}>]], dtype=object)
```

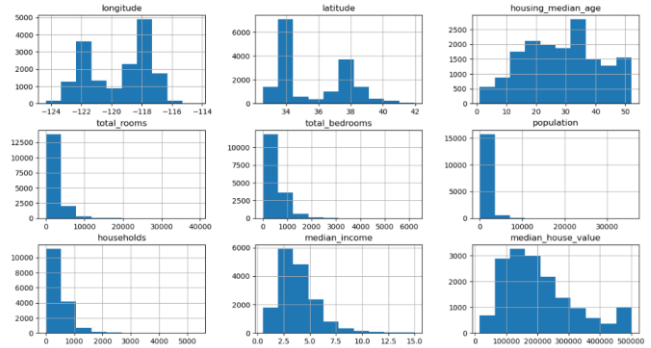

Figure 6

"corr()" is a function that generates a correlation heatmap. Before drawing it is shown what it looks like. The data core is trained. This gave us a Correlation Matrix.

```
train_data.corr()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
longitude	1.000000	-0.925535	-0.106343	0.039736	0.061541	0.094168	0.049575	-0.013468	-0.044851
latitude	-0.925535	1.000000	0.011860	-0.032106	-0.060282	-0.102893	-0.065062	-0.079467	-0.142786
housing_median_age	-0.106343	0.011860	1.000000	-0.357587	-0.316140	-0.290670	-0.299147	-0.129172	0.101443
total_rooms	0.039736	-0.032106	-0.357587	1.000000	0.932185	0.861490	0.923441	0.196890	0.133327
total_bedrooms	0.061541	-0.060282	-0.316140	0.932185	1.000000	0.880343	0.982179	-0.005677	0.051732
population	0.094168	-0.102893	-0.290670	0.861490	0.880343	1.000000	0.907002	0.007636	-0.024369
households	0.049575	-0.065062	-0.299147	0.923441	0.982179	0.907002	1.000000	0.015650	0.066385
median_income	-0.013468	-0.079467	-0.129172	0.196890	-0.005677	0.007636	0.015650	1.000000	0.683967
median_house_value	-0.044851	-0.142786	0.101443	0.133327	0.051732	-0.024369	0.066385	0.683967	1.000000

Figure 7

It is seen in the heatmap that each feature has a correlation with itself. So, every attribute is the same value. Time and then other correlations for feature combinations are seen.

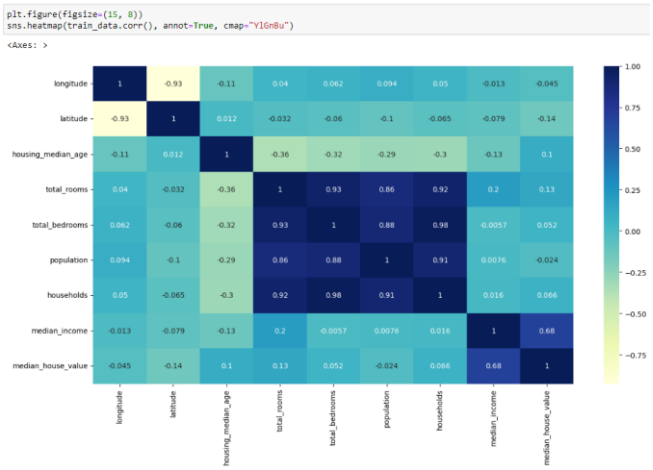

Figure 8

By looking at the actual correlation numbers one can say that the color map would be yellow green blue. According to the table, the correlation with the median home value shows that the median income of a block is highly correlated. The median house value appears to be high or quite strong. So this is a pretty important variable to look at. It makes a great estimator of the home's value. For example, it is seen that latitude is negatively correlated. The house is part of these coordinates with its value.

It will now go into preprocessing and as you can see there are a lot of crooked features here. Total rooms are on the property. Households with total bedrooms and population highly skewed appear to be skewed to the right in statistics but appear to be skewed to the left visually. The data is skewed. This is not a nice gaussian. The bell curve, so what is done for these properties is to take the logarithm of these properties.

IV. Data Preprocessing

```
train_data['total_rooms'] = np.log(train_data['total_rooms']+1)
train_data['total_bedrooms'] = np.log(train_data['total_bedrooms']+1)
train_data['population'] = np.log(train_data['population']+1)
train_data['households'] = np.log(train_data['households']+1)
```

Figure 9

The values for total rooms, total bedrooms, population, and households are increased by one to avoid zero values.

Figure 10

When the code here is run, new values are seen. Now it seems that we have something more like a gaussian bell.

```
train_data.ocean_proximity.value_counts()
```

```
<1H OCEAN     7202
INLAND        5234
NEAR OCEAN    2097
NEAR BAY      1811
ISLAND           2
Name: ocean_proximity, dtype: int64
```

Figure 11

Home prices will be higher when the oceanfront property is closer to the shore. Home prices will have lower prices when inland. If this is true, this property should be converted to a numeric value. It is preferred not to encode them sequentially, as categorical properties do not only give numbers. To take individual category values and set them in binary to turn on one hot by category and overlay them by converting them to binary properties that can be zero or one, called training. The data points are counted as the value that we can say as the proximity point to the ocean. These categories are shown here, and numbers are assigned to them, rather than just taking and assigning them. A new feature has been created for each of these. It's a value of one or zero, so it can be said that this is a feature it can do. It can be yes or no, this is a feature that represents whether the location is less than an hour from the ocean.

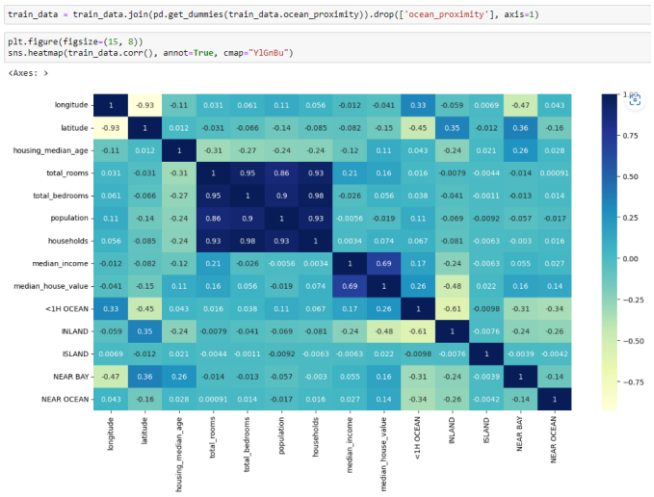

Figure 12

In the code in the cell below, categories will be left followed by proximity to the ocean. Because it has been transformed into multiple features. It was recorded in the training data. We then looked at the correlation between these values so the heatmap was taken from there and the values were redone to see how these new features relate to a value. The target variable is seen here. For example, what is the median home value we have? A negative correlation with Inland, meaning you are paid less if you are Inland. The median price on the house or block is less than an hour from the ocean if not Inland and much lower than if it were the same or vice versa. This is usually a higher price. Now that we have a lot of useful features, the thing to do is take the features we already have and combine them into new features. It introduces the feature engineering part or maybe the coordinates were visualized before the feature engineering was done.

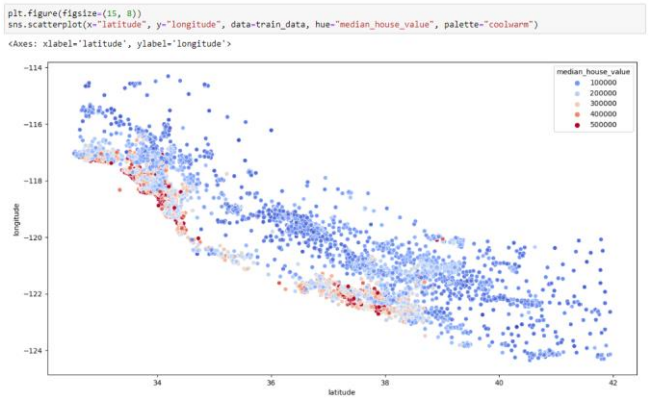

Figure 13

The graph in the figure shows how proximity and the location of individual blocks affect pricing. It is arranged so that the x coordinate is latitude and the y coordinate is longitude. Then the data itself is the training data and the color will be set to our target variable which is the median house value. The palette will be cold and warm so what is seen here is the median house value. So the more red, the more expensive the houses, and the more blue, the less expensive. Expensive houses are shown in the chart. On the coast, near the ocean, houses near the water seem to be more expensive.

V. Feature Engineering

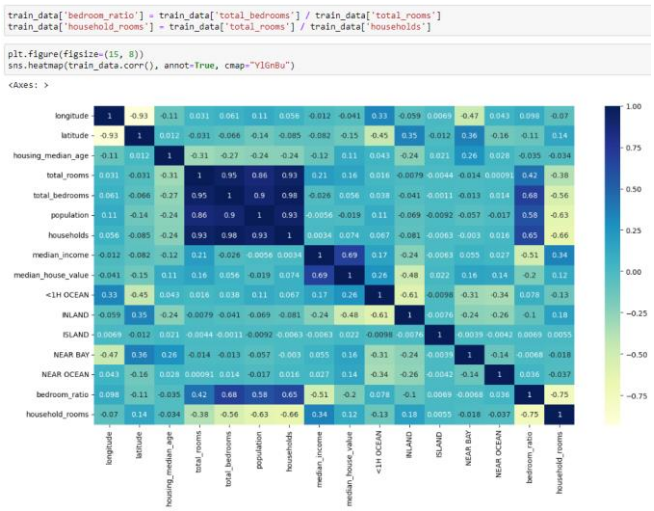

Figure 14

In feature engineering, we have a number of features that are interesting on their own, but can combine them with more interesting features. For example here we have total rooms per block and we also have total rooms. The interesting thing is the number of bedrooms per room. So how many rooms of those rooms are bedrooms, so this can be a separate feature. It can also be said that the proportion of bedrooms.

The ratio of bedrooms was found by dividing the total bedroom with the trained data by the sum of the training data. Another feature can be added as well. For example, total rooms, but if there is a block with more households, there will likely be more rooms for the household. Therefore, the rooms by themselves do not give us the full picture. What we want to do is also tell the training data. It is desired to know how many rooms there are per household. On this basis, only the training data is the total number of rooms divided by the training data. The target variable can be looked at with a correlation heatmap.

When new features are added, they seem to have a significant correlation. Therefore, the bedroom ratio is negative. The correlation between the median and the house value is the house value per block and it is even more so. Thus, it is seen that the households are not in and of themselves. It appears to have a correlation of approximately 0.07 as an input variable.

Rooms seem to have interesting values relative to the target value. The bedroom values owned this year do not seem to have interesting values. However, the bedroom ratio seems to have quite interesting values. That's why two features that seem interesting were designed.

VI. Linear Regression Model

```
from sklearn.linear_model import LinearRegression

x_train, y_train = train_data.drop(['median_house_value'], axis=1), train_data['median_house_value']

reg = LinearRegression()

reg.fit(x_train, y_train)
```

Figure 15

What I want to do in this project is to train multiple models. What we want to do in this section is to create a simple model that can be trained.

Linear regression was used for the model. The training data is divided into x_train and y_train. The data here is split into the original data. The target variable is dropped again. So the median house value X is one and the Y value is where the Y feature is located. Then the model was fit. The next step is to scale the data.

```
test_data = x_test.join(y_test)

test_data['total_rooms'] = np.log(test_data['total_rooms']+1)
test_data['total_bedrooms'] = np.log(test_data['total_bedrooms']+1)
test_data['population'] = np.log(test_data['population']+1)
test_data['households'] = np.log(test_data['households']+1)

test_data = test_data.join(pd.get_dummies(test_data.ocean_proximity)).drop(['ocean_proximity'], axis=1)

test_data['bedroom_ratio'] = test_data['total_bedrooms'] / test_data['total_rooms']
test_data['household_rooms'] = test_data['total_rooms'] / test_data['households']
```

```
x_test, y_test = test_data.drop(['median_house_value'], axis=1), test_data['median_house_value']
```

Figure 16

No hyper-parameter adjustment was required for linear regression. We'll see linear regression towards production and how that performs for the other model. Basically everything that is done with training data has been done.

The next step is to modify the training data to test the data, so the data is tested. There are 16 columns in the training and test data. When we apply the x_test and y_test rec.score() function, the result is: 0.668, which isn't too bad but isn't too good either.

```
reg.score(x_test, y_test)
```

```
0.6706771689902601
```

Figure 17

VII. Random Forest Model

In this section, the Random Forest algorithm will be continued. It has been observed whether this model is a stronger model. Hyper parameter setting was made and tried to find the most suitable model.

```
from sklearn.ensemble import RandomForestRegressor

forest = RandomForestRegressor()

forest.fit(x_train, y_train)
```

Figure 18

```
forest.score(x_test, y_test)
```

```
0.8111460606114163
```

Figure 19

In Figure 19, we import the RandomForestRegressor class from the sklearn.ensemble module. This class allows us to do regression with random forest algorithm. We create an object named forest, which is an example of the RandomForestRegressor class. This object represents our random forest regression model. We call the fit() method on the forest object we created. This is used to train the random forest regression model.

x_train and y_train represent the training dataset. x_train is a dataset containing training examples of independent variables (features) and y_train is a dataset containing training examples of dependent variables (target values).

The training of the model is performed using the x_train and y_train datasets. The model applies the learning process of the random forest algorithm and creates a regression model suitable for the data.

When we apply the x_test and y_test rec.score() function, the result is: 0.811, which is much better than linear regression.

```
from sklearn.model_selection import GridSearchCV

forest = RandomForestRegressor()

param_grid = {
    "n_estimators": [100, 200, 300],
    "min_samples_split": [2, 4],
    "max_depth": [None, 4, 8]
}

grid_search = GridSearchCV(forest, param_grid, cv=5,
                           scoring="neg_mean_squared_error",
                           return_train_score=True)

grid_search.fit(x_train, y_train)
```

Figure 20

In Figure 20, a parameter grid (param_grid) is defined for hyperparameter settings. This parameter grid allows us to experiment with the n_estimators, min_samples_split and max_depth hyperparameters with different values.

n_estimators determines the number of trees, min_samples_split determines the minimum number of samples in node splits, and max_depth determines the maximum depth of the tree.

```
best_forest = grid_search.best_estimator_
```

```
best_forest.score(x_test, y_test)
```

```
0.8092151612782265
```

Figure 21

After many parameter trials, the optimum values are as in the figure above.

When we apply the x_test and y_test rec.score() function, the result is: 0.809, which is much better than linear regression.

VIII. Conclusion

As a result of the project, a housing price dataset was obtained from Kaggle to perform exploratory analysis using the necessary preprocessing and engineering techniques. The data set was scaled and then evaluated by creating different models. First, classification was made using logistic regression. In addition, an experiment was carried out with a simple decision tree. In this process, designing different features has been tried and some features have been removed when necessary. In these experiments, it was predicted that I could improve the performance of each method. This project can be considered as an example of a real data science and machine learning project or analysis and these studies will make me a better data scientist and have definitely allowed me to learn new things. Experience has been gained in this process and skills in data science have been developed.

References

[1] https://www.kaggle.com/datasets/camnugent/california-housing-prices