Predicting Housing Prices - California Housing Price

1. Objective of the analysis

This is a Machine Learning project trained on California Housing Prices dataset based on the 1990 census data. We will work on supervised learning and our aim is to do predictive analysis on the median house value.

2. Description of the data set

The following table provides descriptions, data ranges, and data types for each feature in the data set:

Column title	Description	Range*	Datatype
`longitude`	A measure of how far west a house is; a higher value is farther west	• Longitude values range from -180 to +180 • Data set min: -124.3 • Data set max: -114.3	float64
`latitude`	A measure of how far north a house is; a higher value is farther north	• Latitude values range from -90 to +90 • Data set min: 32.5 • Data set max: 42.5	float64
`housingMedianAge`	Median age of a house within a block; a lower number is a newer building	• Data set min: 1.0 • Data set max: 52.0	float64
`totalRooms`	Total number of rooms within a block	• Data set min: 2.0 • Data set max: 37937.0	float64
`totalBedrooms`	Total number of bedrooms within a block	• Data set min: 1.0 • Data set max: 6445.0	float64
`population`	Total number of people residing within a block	• Data set min: 3.0 • Data set max: 35682.0	float64
`households`	Total number of households, a group of people residing within a home unit, for a block	• Data set min: 1.0 • Data set max: 6082.0	float64
`medianIncome`	Median income for households within a block of houses (measured in tens of thousands of US Dollars)	• Data set min: 0.5 • Data set max: 15.0	float64
`medianHouseValue`	Median house value for households within a block (measured in US Dollars)	• Data set min: 14999.0 • Data set max: 500001.0	float64

The data set contains 20,640 observations and 10 features:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0	NEAR BAY
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0	NEAR BAY
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	352100.0	NEAR BAY
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	341300.0	NEAR BAY
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	342200.0	NEAR BAY
...	...	...	...	...	...	...	...	...	...	...
20635	-121.09	39.48	25.0	1665.0	374.0	845.0	330.0	1.5603	78100.0	INLAND
20636	-121.21	39.49	18.0	697.0	150.0	356.0	114.0	2.5568	77100.0	INLAND
20637	-121.22	39.43	17.0	2254.0	485.0	1007.0	433.0	1.7000	92300.0	INLAND
20638	-121.32	39.43	18.0	1860.0	409.0	741.0	349.0	1.8672	84700.0	INLAND
20639	-121.24	39.37	16.0	2785.0	616.0	1387.0	530.0	2.3886	89400.0	INLAND

20640 rows × 10 columns

3. Feature engineering

In order to prepare our data for the analysis, we followed these steps:

Missing values

It is important to see if the data set contains missing values. If so, we have to evaluate if it would be necessary to remove or replace those values.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   longitude           20640 non-null  float64
 1   latitude            20640 non-null  float64
 2   housing_median_age  20640 non-null  float64
 3   total_rooms         20640 non-null  float64
 4   total_bedrooms      20433 non-null  float64
 5   population          20640 non-null  float64
 6   households          20640 non-null  float64
 7   median_income       20640 non-null  float64
 8   median_house_value  20640 non-null  float64
 9   ocean_proximity     20640 non-null  object 
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
```

The only feature missing values is *'total_bedrooms'* and we have 3 different option we can choose from:

- Get rid of the corresponding districts.
- Get rid of the whole attribute.
- Set the values to some value (zero, the mean, the median, etc.).

I have opted to replace the missing values with the median and use '*SimpleImputer*' in order to apply the code to all numerical values ensuring that this will eventually apply to new data also:

```
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(strategy="median")

housing_num = housing.drop("ocean_proximity", axis=1)

imputer.fit(housing_num)

X = imputer.transform(housing_num)
housing_tr = pd.DataFrame(X, columns=housing_num.columns,
                          index=housing.index)
```

We had to drop *'ocean_proximity' since the median can be only computed on numerical values.*

One-Hot-Encoding

I will convert *'ocean_proximity' from categorical attribute to numerical attribute, since ML algorithms work better with numbers:*

	ocean_proximity
14196	NEAR OCEAN
8267	NEAR OCEAN
17445	NEAR OCEAN
14265	NEAR OCEAN
2271	INLAND
17848	<1H OCEAN
6252	<1H OCEAN
9389	NEAR BAY
6113	<1H OCEAN
6061	<1H OCEAN

```
# One Hot Encoding

from sklearn.preprocessing import OneHotEncoder

cat_encoder = OneHotEncoder()

housing_cat_1hot = cat_encoder.fit_transform(housing_cat)
housing_cat_1hot
```

```
<16512x5 sparse matrix of type '<class 'numpy.float64'>'
        with 16512 stored elements in Compressed Sparse Row format>
```

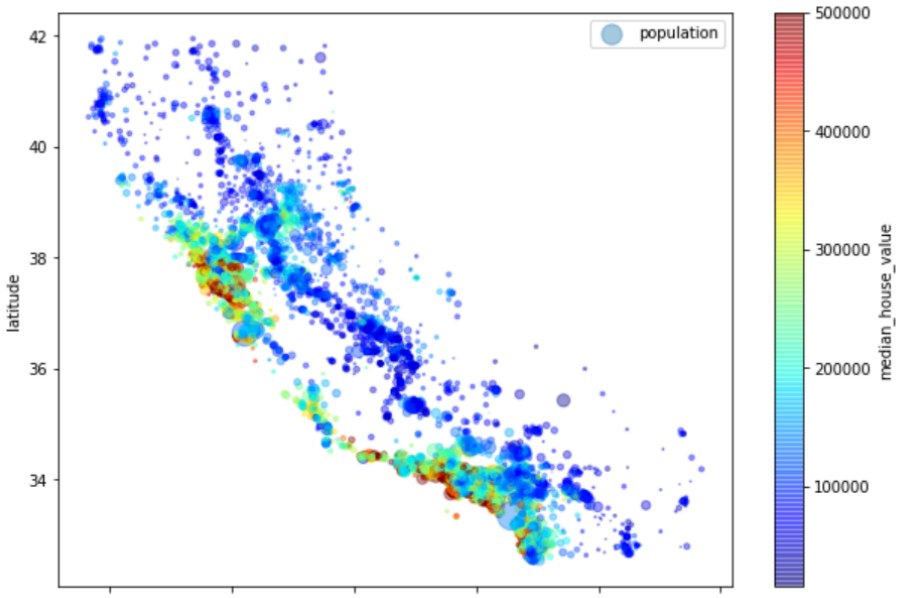
```
# Matrix of the categories in an Numpy arrey - toarrey

housing_cat_1hot.toarray()
```

```
array([[0., 0., 0., 0., 1.],
       [0., 0., 0., 0., 1.],
       [0., 0., 0., 0., 1.],
       ...,
       [1., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0.],
       [0., 0., 0., 1., 0.]])
```

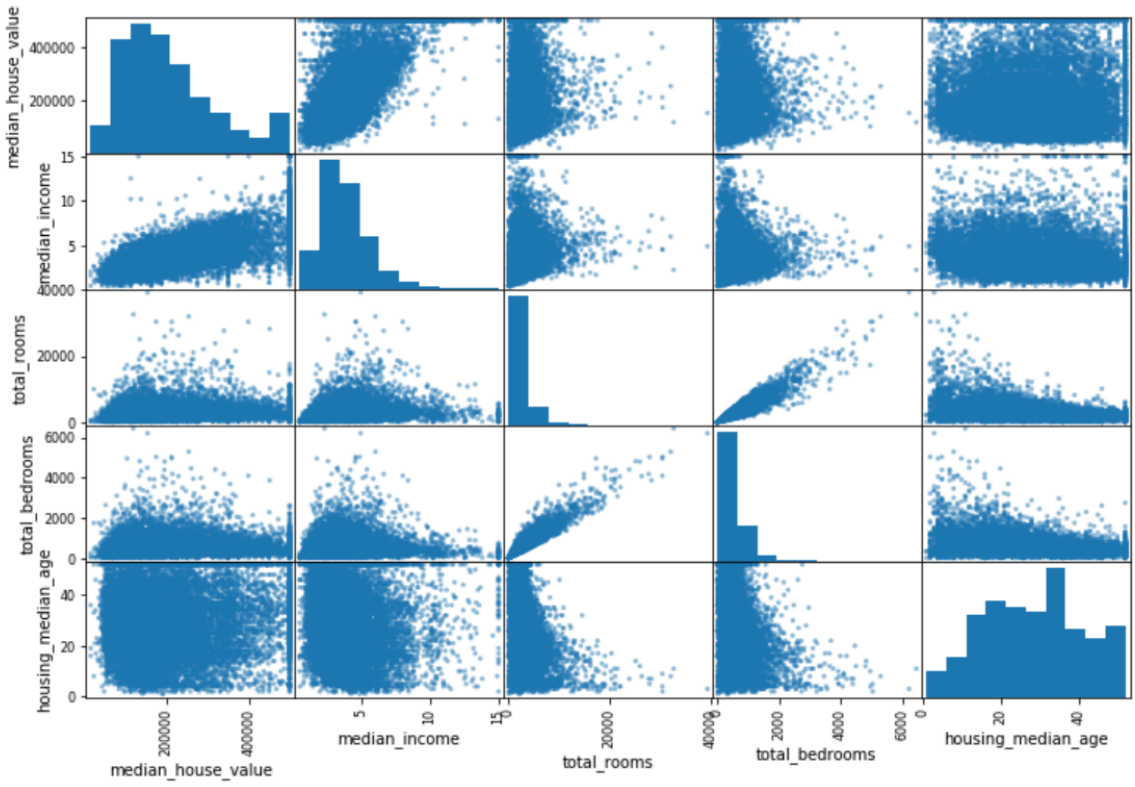
Data Visualization

First of all I have created a scatterplot with the geographical information to see the distribution of the districts. By using *cmap*, I have also added *'population'* and *'median_house_value'* to have a clear understanding of how the data are distributed in each district:



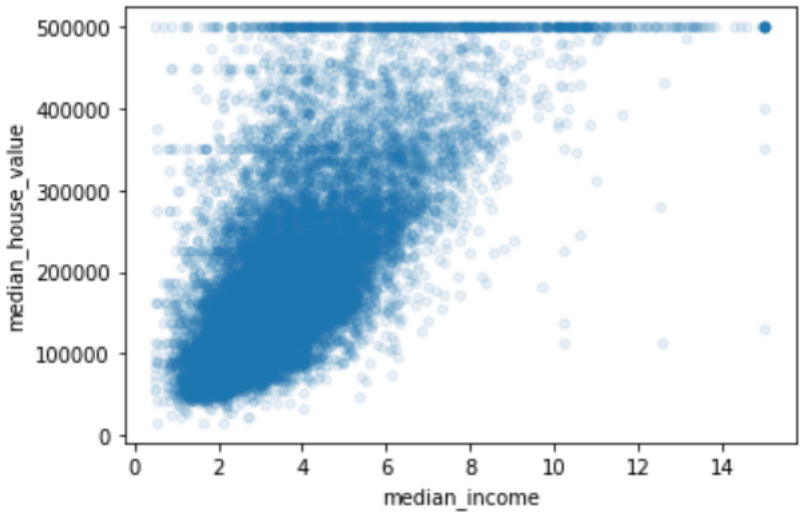
We can say that the house price is related to the location and the population density. Also *'ocean_proximity'* seems a useful attribute to take into consideration.

This is how correlation between the attributes using 'scatter_matrix':



The attributes that correlate the most are 'median_income' and 'median_house_value'.

If we take a closer look to the correlation, we will notice that there is a price cap at $50,000:



I have also created additional attribute to see in order to explore and get more data insight.

```
housing['rooms_per_household'] = housing['total_rooms']/housing['households']
housing['bedrooms_per_household'] = housing['total_bedrooms']/housing['total_rooms']
housing['population_per_household'] = housing['population']/housing['households']
```

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

```
median_house_value          1.000000
median_income               0.690647
rooms_per_household         0.158485
total_rooms                 0.133989
housing_median_age          0.103706
households                  0.063714
total_bedrooms              0.047980
population_per_household   -0.022030
population                 -0.026032
longitude                  -0.046349
latitude                   -0.142983
bedrooms_per_household     -0.257419
Name: median_house_value, dtype: float64
```

We can see that 'rooms_per_household' has a better correlation than *'total_rooms'*, but still *'median_income'* correlate the best.

4. Train the model

I have performed different model in order to find the one that fitted the most:

- Linear Regression
- Decision Tree
- Random Forest

Linear Regression

```
from sklearn.linear_model import LinearRegression

lin_reg = LinearRegression()
lin_reg.fit(housing_prepared, housing_labels)
```

```
LinearRegression()
```

```
# let's try the full preprocessing pipeline on a few training instances
some_data = housing.iloc[:5]
some_labels = housing_labels.iloc[:5]
some_data_prepared = full_pipeline.transform(some_data)

print("Predictions:", lin_reg.predict(some_data_prepared))
```

```
Predictions: [181746.54359616 290558.74973505 244957.50017771 146498.51061398
 163230.42393939]
```

```
print("Labels:", list(some_labels))
```

```
Labels: [103000.0, 382100.0, 172600.0, 93400.0, 96500.0]
```

The linear regression model does not give a very accurate prediction given the fact that is off by almost 40%, also if we measure de Mean Squared Error (RMSE) will give us a prediction error of $68,628 which is not very satisfying given the fact that the median housing values range between $120,000 and $265,000.

This is the result of applying cross-validation to linear regression model:

```
lin_scores = cross_val_score(lin_reg, housing_prepared, housing_labels,
                             scoring="neg_mean_squared_error", cv=10)
lin_rmse_scores = np.sqrt(-lin_scores)
display_scores(lin_rmse_scores)
```

```
Scores: [65000.67382615 70960.56056304 67122.63935124 66089.63153865
 68402.54686442 65266.34735288 65218.78174481 68525.46981754
 72739.87555996 68957.34111906]
Mean: 67828.38677377408
Standard deviation: 2468.0913950652284
```

Decision Tree

```
from sklearn.tree import DecisionTreeRegressor

tree_reg = DecisionTreeRegressor(random_state=42)
tree_reg.fit(housing_prepared, housing_labels)
```

```
DecisionTreeRegressor(random_state=42)
```

```
housing_predictions = tree_reg.predict(housing_prepared)
tree_mse = mean_squared_error(housing_labels, housing_predictions)
tree_rmse = np.sqrt(tree_mse)
tree_rmse
```

```
0.0
```

Decision Trees gives a no error at all, but I wanted to be sure that this was correct and not just an overfitting. This is the result by applying cross-validation:

```
from sklearn.model_selection import cross_val_score

scores = cross_val_score(tree_reg, housing_prepared, housing_labels,
                         scoring="neg_mean_squared_error", cv=10)
tree_rmse_scores = np.sqrt(-scores)
```

```
def display_scores(scores):
    print("Scores:", scores)
    print("Mean:", scores.mean())
    print("Standard deviation:", scores.std())

display_scores(tree_rmse_scores)
```

```
Scores: [65312.86044031 70581.69865676 67849.75809965 71460.33789358
 74035.29744574 65562.42978503 67964.10942543 69102.89388457
 66876.66473025 69735.84760006]
Mean: 68848.18979613911
Standard deviation: 2579.6785558576307
```

We can actually see that the Decision Tree is actually overfitting and performing worse than the Linear Regression

Random Forest

```
from sklearn.ensemble import RandomForestRegressor

forest_reg = RandomForestRegressor(n_estimators=100, random_state=42)
forest_reg.fit(housing_prepared, housing_labels)
```

```
RandomForestRegressor(random_state=42)
```

```
housing_predictions = forest_reg.predict(housing_prepared)
forest_mse = mean_squared_error(housing_labels, housing_predictions)
forest_rmse = np.sqrt(forest_mse)
forest_rmse
```

```
18527.322990316152
```

Random Forest result looks better than the other model I previously used, but it is still overfitting the training set.

I have fine-tuned the random Forest model in order to achieve a better result.

```
from sklearn.model_selection import cross_val_score

forest_scores = cross_val_score(forest_reg, housing_prepared, housing_labels,
                                scoring="neg_mean_squared_error", cv=10)
forest_rmse_scores = np.sqrt(-forest_scores)
display_scores(forest_rmse_scores)
```

```
Scores: [47341.96931397 51653.53070248 49360.29148883 51625.62777032
 52771.91063892 46989.97118038 47333.72603398 50636.24303693
 48951.73251683 50183.60590465]
Mean: 49684.86085873057
Standard deviation: 1929.9797084102233
```

The grid will search for 18 combinations of RandomForestRegression hyperparameter values and will train each model 10 times (cv = 10). This is the result of the code:

```
The best hyperparameters combination is: {'max_features': 8, 'n_estimators': 30}
The best hyperparameters estimator is: RandomForestRegressor(max_features=8, n_estimators=30, random_state=42)
```

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

```
64878.27480854276 {'max_features': 2, 'n_estimators': 3}
55391.003575336406 {'max_features': 2, 'n_estimators': 10}
52721.66494842234 {'max_features': 2, 'n_estimators': 30}
58541.12715494087 {'max_features': 4, 'n_estimators': 3}
51623.59366665994 {'max_features': 4, 'n_estimators': 10}
49787.65951361993 {'max_features': 4, 'n_estimators': 30}
58620.88234614251 {'max_features': 6, 'n_estimators': 3}
51645.862673140065 {'max_features': 6, 'n_estimators': 10}
49917.66994061786 {'max_features': 6, 'n_estimators': 30}
58640.96129790229 {'max_features': 8, 'n_estimators': 3}
51650.365501630005 {'max_features': 8, 'n_estimators': 10}
49672.50940389753 {'max_features': 8, 'n_estimators': 30}
61560.24110015614 {'bootstrap': False, 'max_features': 2, 'n_estimators': 3}
53889.80996032937 {'bootstrap': False, 'max_features': 2, 'n_estimators': 10}
58667.89389226964 {'bootstrap': False, 'max_features': 3, 'n_estimators': 3}
52764.2630869393 {'bootstrap': False, 'max_features': 3, 'n_estimators': 10}
58496.54315778315 {'bootstrap': False, 'max_features': 4, 'n_estimators': 3}
51440.83147410301 {'bootstrap': False, 'max_features': 4, 'n_estimators': 10}
```

5. Recommended Regression Model

The combination of our hyperparameters gives a result of $49,682 more or less the same as the one we had before $49,684.

We can consider Random Forest Regression our best model.

6. Results on the Test set

```
final_model = grid_search.best_estimator_

X_test = test_set.drop("median_house_value", axis=1)
y_test = test_set["median_house_value"].copy()

X_test_prepared = full_pipeline.transform(X_test)
final_predictions = final_model.predict(X_test_prepared)

final_mse = mean_squared_error(y_test, final_predictions)
final_rmse = np.sqrt(final_mse)

final_rmse
```

```
49198.020631676336
```

In order to have a better idea of how prices this estimation is, I compute a 95% confidence interval for the general error:

```
# We can compute a 95% confidence interval for the test RMSE:

from scipy import stats

confidence = 0.95
squared_errors = (final_predictions - y_test) ** 2
np.sqrt(stats.t.interval(confidence, len(squared_errors) - 1,
                         loc=squared_errors.mean(),
                         scale=stats.sem(squared_errors)))
```

```
array([46948.10215126, 51349.4515311 ])
```

7. Summary Key Findings and Insights

The final performance of the system is not better than the experts' price estimates, which are often off by about 20%. Nevertheless, is still worth it to use the model to predict the median house value.

8. Suggestions for next steps

I concentrated my analysis mainly on the median income as this was the most correlated attribute with the median house value.

A more detailed analysis and considering more attribute for fitting the model, will definitely improve our chances to create a better model.