PRICE HOUSING PREDICTION

INTRODUCTION AND MAIN GOALS

For this analysis we use the California Housing dataset found on the Kaggle Data Repository.

This data has features such as the population, median income, median housing price, and so on for each block group in California. Block groups are the smallest geographical unit for which the US Census Bureau publishes sample data.

The objective of the analysis is to predict median house prices in different regions of California as per the 1990 census data.

It is a supervised learning task: a multiple regression problem (univariate, since there's only one value to predict for each district).

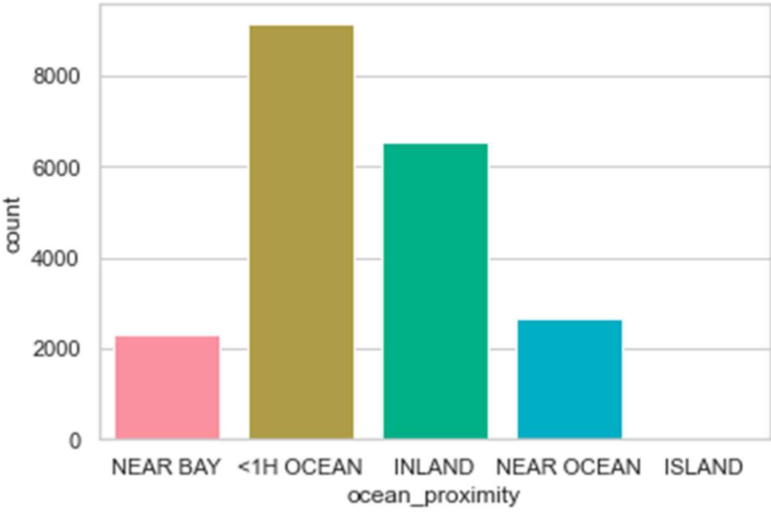
DATASET CONTENT

In total there are 20,640 records and 10 columns. For just one of the features there are 207 missing values, which represents the 1% of the total values. The rest of the fetures have no missing values, and there are no duplicates.

The features are the following:

```
<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
```

There are all floats, except for 'Ocean proximity', which is a categorical feature, and can have 5 possible values.

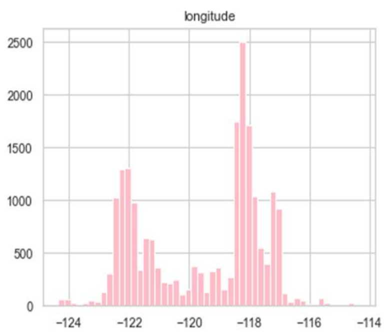


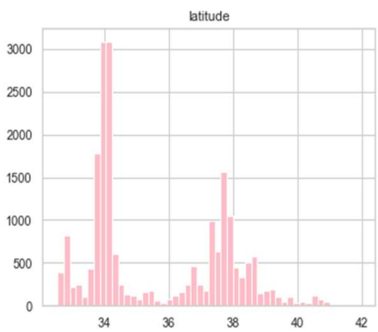
This is the dataset description:

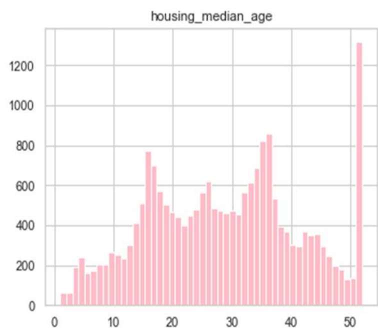
	count	unique	top	freq	mean	std	min	25%	50%	75%	max
longitude	20640.0	NaN	NaN	NaN	-120.0	2.0	-124.0	-122.0	-118.0	-118.0	-114.0
latitude	20640.0	NaN	NaN	NaN	36.0	2.0	33.0	34.0	34.0	38.0	42.0
housing_median_age	20640.0	NaN	NaN	NaN	29.0	13.0	1.0	18.0	29.0	37.0	52.0
total_rooms	20640.0	NaN	NaN	NaN	2636.0	2182.0	2.0	1448.0	2127.0	3148.0	39320.0
total_bedrooms	20433.0	NaN	NaN	NaN	538.0	421.0	1.0	296.0	435.0	647.0	6445.0
population	20640.0	NaN	NaN	NaN	1425.0	1132.0	3.0	787.0	1166.0	1725.0	35682.0
households	20640.0	NaN	NaN	NaN	500.0	382.0	1.0	280.0	409.0	605.0	6082.0
median_income	20640.0	NaN	NaN	NaN	4.0	2.0	0.0	3.0	4.0	5.0	15.0
median_house_value	20640.0	NaN	NaN	NaN	206856.0	115396.0	14999.0	119600.0	179700.0	264725.0	500001.0
ocean_proximity	20640	5	<1H OCEAN	9136	NaN	NaN	NaN	NaN	NaN	NaN	NaN

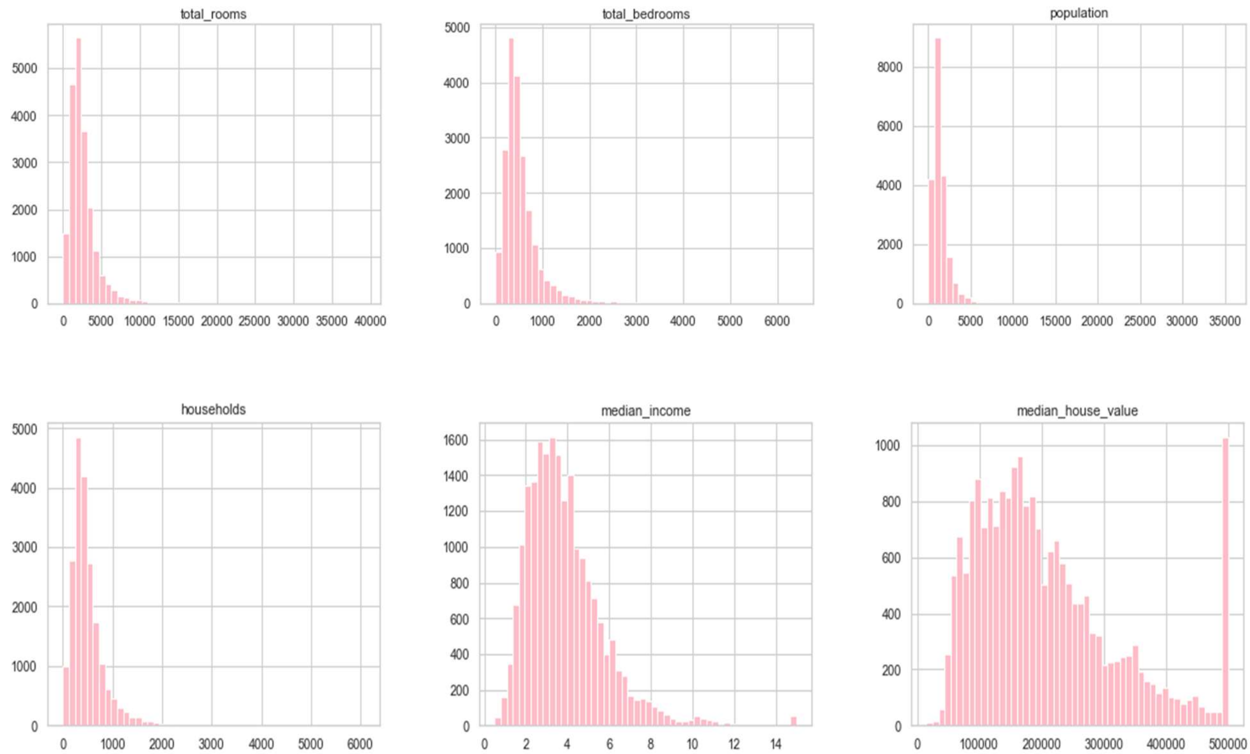
SUMMARY OF EDA

Some graphs





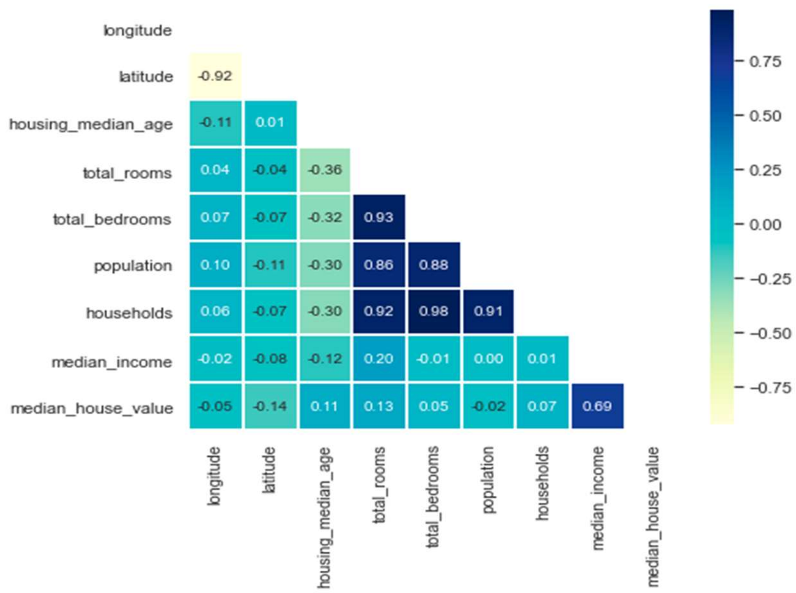




Some insights:

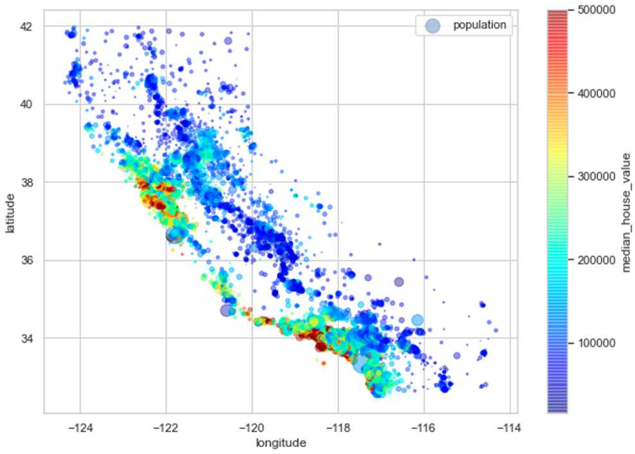
1. Median income: the maximun value is 15 and the minimun, almost 0. It seems that the data has been scaled and cuted at 15 for higher median incomes, and at 0.5 for lower median incomes. The numbers represent tens of thousands of dollars: 5 actually means $50,000
2. The maximum total bedrooms is 6445 and the maximum rooms is 39320... This seems a little bit weird.

Correlation

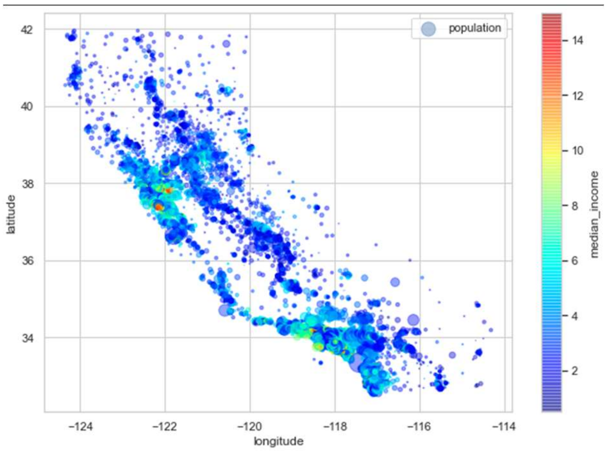


Let’s focus on a few promising attributes that seem most correlated with the median housing value

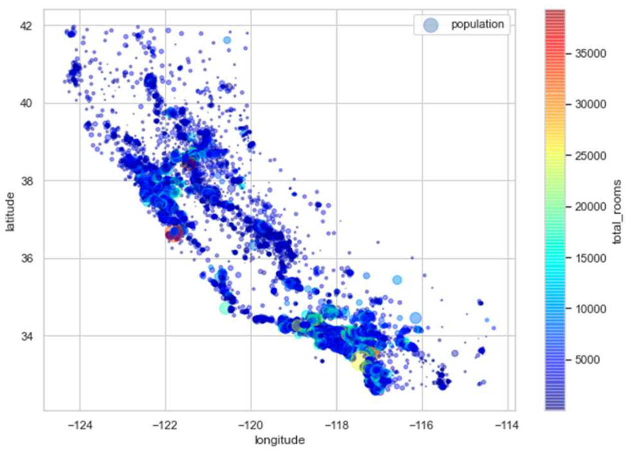
Median house value



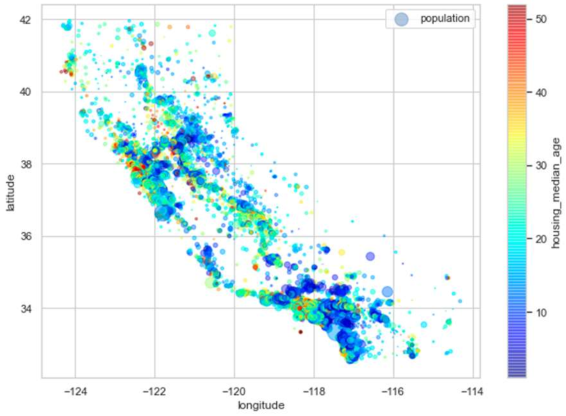
Median income



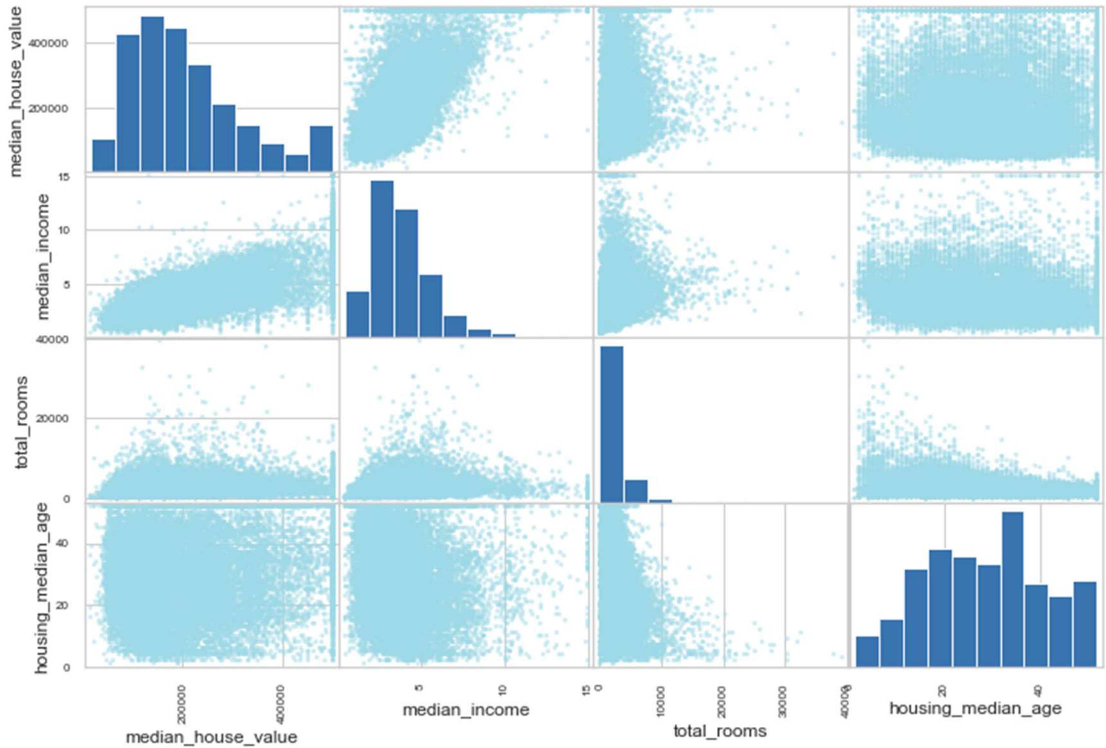
Total rooms



Housing median age



Scatter matrix for those features



Taking a closer look to the correlation between median income and median house value, a price cap is clearly visible as a horizontal line at usd 500,000. But this plot reveals other less obvious straight lines: a horizontal line around usd 450,000, another around usd 350,000, perhaps one around usd 280,000, and a few more below that.

FEATURE ENGINEER

Dealing with missing values:

```
median=df['total_bedrooms'].median()
df['total_bedrooms'].fillna(median, inplace=True)
print("Number of null values in total bedrooms column: {}".format(df['total_bedrooms'].isnull().sum()))
```

```
Number of null values in total bedrooms column: 0
```

Creating new features:

```
#Rooms per household
df["rooms_per_household"] = df["total_rooms"]/df["households"]

#Bedrooms per room
df["bedrooms_per_room"] = df["total_bedrooms"]/df["total_rooms"]

#Population per household
df["population_per_household"]=df["population"]/df["households"]

#Housing median age per median income
df["housing_age_per_income"]=df["housing_median_age"]/df["median_income"]
```

```
#Let's take a look to the correlation with the target, given this new features
correlation=df.corr(method="pearson")
correlation['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              0.049457
population_per_household   -0.023737
population                 -0.024650
longitude                  -0.045967
latitude                   -0.144160
bedrooms_per_room          -0.233303
housing_age_per_income     -0.320028
Name: median_house_value, dtype: float64
```

Converting categorical data into a numeric data

```
ocean_proximity_dumies=pd.get_dummies(df['ocean_proximity'], drop_first=True)
df=df.drop(columns=['ocean_proximity'])
df=pd.concat([df,ocean_proximity_dumies],axis=1)
```

Feature scaling

```
scaler = MinMaxScaler()

df_scaled = pd.DataFrame(scaler.fit_transform(df))
df_scaled
```

Train test split:

1. I keep 20% for test and 80% for train.
2. I define the target, which is median_house_value.
3. I created a new variable, which is only used for the split, and then droped. As the median income is a very important attribute to predict median

housing prices, I want to ensure that the test set is representative of the various categories of incomes in the whole dataset.

```
split = StratifiedShuffleSplit(n_splits=1,
                               test_size=0.2,
                               random_state=42)

for train_index, test_index in split.split(df_scaled, df_scaled["income_categorical"]):
    strat_train_set = df_scaled.loc[train_index]
    strat_test_set = df_scaled.loc[test_index]
```

```
strat_test_set["income_categorical"].value_counts() / len(strat_test_set)
```

```
0.50    0.350533
0.25    0.318798
0.75    0.176357
1.00    0.114583
0.00    0.039729
Name: income_categorical, dtype: float64
```

```
#Check if the proportoins mantein if we analyze the whole dataset
df["income_categorical"].value_counts() / len(df)
```

```
3    0.350581
2    0.318847
4    0.176308
5    0.114438
1    0.039826
Name: income_categorical, dtype: float64
```

Some insights

There are 9 features that are used to predict the median_house_value:

- longitude and latitude: represent California. Houses in San Francisco and Los angeles-San Diego are more expensive. Also, there is negative correlation with latitude: if I go north, the houses get cheaper.
- median_income: the attribute with higher correlation with the house price: 0.687!
- total_rooms: 0.135 of correlation with the median_house_value
- total_bedrooms: has almost not correlation
- population, households: don't help much alone, but we can consider

Finally, creating new features:

- rooms_per_household is more informative than total_rooms or households
- bedrooms_per_room has a good correlation with median_house_value
- population_per_household a bit less, but somehow its inverse household_per_population is more informative
- housing_age_per_income has a strong negative correlation with the target

MODELING

Linear Regression:

```
lin_reg = LinearRegression()
lin_reg.fit(housing_train, housing_train_target)
```

```
LinearRegression()
```

```
housing_train_predictions = lin_reg.predict(housing_train)
housing_test_predictions = lin_reg.predict(housing_test)

error_df=pd.Series({'train': mean_squared_error(housing_train_target, housing_train_predictions),
                    'test' : mean_squared_error(housing_test_target, housing_test_predictions)},)

error_df
```

```
train    0.019996
test     0.019083
dtype: float64
```

```
r2_score(housing_test_target,housing_test_predictions)
```

```
0.6555425196337148
```

It's a quite small r2

Ridge

```
r = Ridge(alpha = 0.1, random_state=42)
```

```
r.fit(housing_train, housing_train_target)
```

```
Ridge(alpha=0.1, random_state=42)
```

```
housing_train_predictions_r = r.predict(housing_train)
housing_test_predictions_r = r.predict(housing_test)

error_df=pd.Series({'train': mean_squared_error(housing_train_target, housing_train_predictions_r),
                    'test' : mean_squared_error(housing_test_target, housing_test_predictions_r)},)

error_df
```

```
train    0.020000
test     0.019089
dtype: float64
```

```
r2_score(housing_test_target,housing_test_predictions_r)
```

```
0.655434922189142
```

It returned a very similar r2

Decision Tree Regressor

```
tree_reg = DecisionTreeRegressor(random_state=42)
tree_reg.fit(housing_train, housing_train_target)
```

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

```
housing_train_predictions_dt = tree_reg.predict(housing_train)
housing_test_predictions_dt = tree_reg.predict(housing_test)

error_df=pd.Series({'train': mean_squared_error(housing_train_target, housing_train_predictions_dt),
                    'test' : mean_squared_error(housing_test_target, housing_test_predictions_dt)},)

error_df
```

```
train    2.239453e-36
test     2.141052e-02
dtype: float64
```

```
r2_score(housing_test_target,housing_test_predictions_dt)
```

```
0.6135305701797974
```

With GridSearch, the r2 score is higher

```
model_basic=DecisionTreeRegressor(random_state=42).fit(housing_train, housing_train_target)

#Creating a dictionary grid for grid search
param_grid = {'max_depth':range(1, model_basic.tree_.max_depth+1, 2),
              'min_samples_leaf':[10,20,30,40,50],
              'max_features':range(1, len(model_basic.feature_importances_)+1)}

#Creating a dictionary grid for scoring search
#scoring = {"r2_score": "r2",
#           "mean_squared_error": make_scorer(mean_squared_error)}


#Fitting grid search to the train data with 5 folds
gridsearch_model_basic = GridSearchCV(estimator=model_basic,
                                      param_grid= param_grid,
                                      #cv=StratifiedKFold(),
                                      n_jobs=-1,
                                      scoring="r2",
                                      #refit="r2",
                                      verbose=2)

gridsearch_dt_basic=gridsearch_model_basic.fit(housing_train, housing_train_target)
```

```
Fitting 5 folds for each of 1520 candidates, totalling 7600 fits
```

```
print("Best parameters: "+str(gridsearch_model_basic.best_params_))
print("Best score: "+str(gridsearch_model_basic.best_score_)+'\n')
```

```
Best parameters: {'max_depth': 17, 'max_features': 13, 'min_samples_leaf': 20}
Best score: 0.7483620603396248
```

Activ

```
tree_reg = DecisionTreeRegressor(random_state=42,
                                 max_depth= 17,
                                 max_features= 13,
                                 min_samples_leaf= 20)

tree_reg.fit(housing_train, housing_train_target)
```

```
DecisionTreeRegressor(max_depth=17, max_features=13, min_samples_leaf=20,
                      random_state=42)
```

```
housing_train_predictions_dtgs = tree_reg.predict(housing_train)
housing_test_predictions_dtgs = tree_reg.predict(housing_test)

error_df=pd.Series({'train': mean_squared_error(housing_train_target, housing_train_predictions_dtgs),
                    'test' : mean_squared_error(housing_test_target, housing_test_predictions_dtgs)},)

error_df
```

```
train    0.009963
test     0.013740
dtype: float64
```

```
r2_score(housing_test_target,housing_test_predictions_dtgs)
```

```
0.7519956112600878
```

Now, the r2 is 75%

Random Forest Regressor

```
random_reg = RandomForestRegressor(random_state=42,
                                   bootstrap= False)
random_reg.fit(housing_train, housing_train_target)
```

```
RandomForestRegressor(bootstrap=False, random_state=42)
```

```
housing_train_predictions_rf = random_reg.predict(housing_train)
housing_test_predictions_rf = random_reg.predict(housing_test)

error_df=pd.Series({'train': mean_squared_error(housing_train_target, housing_train_predictions_rf),
                    'test' : mean_squared_error(housing_test_target, housing_test_predictions_rf)},)

error_df
```

```
train    2.975708e-31
test     2.029180e-02
dtype: float64
```

```
r2_score(housing_test_target,housing_test_predictions_rf)
```

```
0.6337239576147082
```

With GridSearch, the r2 score is higher

```
model_basic=RandomForestRegressor(random_state=42,bootstrap= False).fit(housing_train, housing_train_target)

#Creating a dictionary grid for grid search
param_grid = {'max_depth': [30,40,50],
              'min_samples_leaf':[10,40,50],
              'max_features': ['sqrt', 'log2'],
              'n_estimators':[500,700,800,900]}

#Creating a dictionary grid for scoring search
#scoring = {"r2_score": "r2",
#           "mean_squared_error": make_scorer(mean_squared_error)}


#Fitting grid search to the train data with 5 folds
gridsearch_model_basic = GridSearchCV(estimator=model_basic,
                                      param_grid= param_grid,
                                      #cv=StratifiedKFold(),
                                      n_jobs=-1,
                                      scoring="r2",
                                      #refit="r2",
                                      verbose=2)

gridsearch_rf_basic=gridsearch_model_basic.fit(housing_train, housing_train_target)
```

```
Fitting 5 folds for each of 72 candidates, totalling 360 fits
```

```
print("Best parameters: "+str(gridsearch_rf_basic.best_params_))
print("Best score: "+str(gridsearch_rf_basic.best_score_)+'\n')
```

```
random_reg = RandomForestRegressor(random_state=42,
                                   bootstrap= False,
                                   max_depth= 40,
                                   max_features= 'sqrt',
                                   min_samples_leaf= 10,
                                   n_estimators=900)

random_reg.fit(housing_train, housing_train_target)
```

```
RandomForestRegressor(bootstrap=False, max_depth=40, max_features='sqrt',
                      min_samples_leaf=10, n_estimators=900, random_state=42)
```

```
housing_train_predictions_rfgs = random_reg.predict(housing_train)
housing_test_predictions_rfgs = random_reg.predict(housing_test)

error_df=pd.Series({'train': mean_squared_error(housing_train_target, housing_train_predictions_rfgs),
                    'test' : mean_squared_error(housing_test_target, housing_test_predictions_rfgs)},)

error_df
```

```
train    0.011552
test     0.012515
dtype: float64
```

```
r2_score(housing_test_target,housing_test_predictions_rfgs)
```

```
0.7740983064449678
```


RECOMENDATIONS AND KEY FINDINGS

The highest r2 score is given by the Decision Tree Regressor, so, this is the model which is going to be used to predict the prices for California houses.

SUGGESTIONS FOR NEXT STEPS

1. The first two models can be optimized with GridSearch or RandomizedSearch to find the parameters that minimize the mean error.
2. I could try to normalize the target
3. I could try Voting Regressor