

# Forecast of real estate price

## Content

1. [Project Description](#)
2. [Data Preparation](#)
3. [Models training](#)
4. [Analyzis of the results and model testing](#)

## Project Description

Based on the provided data from real estate agency it's required using spark session to conduct an analysis, encode the data and train a regression models for the prediction of the median cost of realestate price. First model has to include only numeric features, second - numeric and categorical features.

Main tasks of the project are following^

- Import and prepare the data;
- Train the models and compare using RMSE, MAE and R2 scores;
- Test the best model.

## Data preparation

```
In [1]: import pandas as pd
import numpy as np

import pyspark
```

```

from pyspark.sql import SparkSession
from pyspark.sql.types import *
import pyspark.sql.functions as F
from pyspark.ml.feature import StringIndexer, VectorAssembler, StandardScaler
from pyspark.ml.regression import LinearRegression
from pyspark.ml.evaluation import RegressionEvaluator
import sklearn.metrics
from sklearn.metrics import mean_absolute_error
import warnings

warnings.filterwarnings('ignore')
warnings.simplefilter(action='ignore', category=all)

pyspark_version = pyspark.__version__
if int(pyspark_version[:1]) == 3:
    from pyspark.ml.feature import OneHotEncoder
elif int(pyspark_version[:1]) == 2:
    from pyspark.ml.feature import OneHotEncodeEstimator

RANDOM_SEED = 2022

spark = SparkSession.builder \
    .master("local") \
    .appName("California - linear regression") \
    .getOrCreate()

```

```

In [2]: # data import
df_california = spark.read.option('header', 'true').csv('housing.csv', inferSchema=True)

```

```

In [3]: # display of first 10 rows
df_california.show(10)

```

```

+---+-----+-----+-----+-----+-----+-----+-----+-----+-----+
-----+
|_c0|longitude|latitude|housing_median_age|total_rooms|total_bedrooms|population|households|median_income|median_house_value|oce
an_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|
| 5| -122.25| 37.85| 52.0| 919.0| 213.0| 413.0| 193.0| 4.0368| 269700.0|
NEAR BAY|
| 6| -122.25| 37.84| 52.0| 2535.0| 489.0| 1094.0| 514.0| 3.6591| 299200.0|
NEAR BAY|
| 7| -122.25| 37.84| 52.0| 3104.0| 687.0| 1157.0| 647.0| 3.12| 241400.0|
NEAR BAY|
| 8| -122.26| 37.84| 42.0| 2555.0| 665.0| 1206.0| 595.0| 2.0804| 226700.0|
NEAR BAY|
| 9| -122.25| 37.84| 52.0| 3549.0| 707.0| 1551.0| 714.0| 3.6912| 261100.0|
NEAR BAY|
+---+-----+-----+-----+-----+-----+-----+-----+-----+-----+
-----+
only showing top 10 rows

```

```

In [4]: # display datatype on each column of dataset
print(pd.DataFrame(df_california.dtypes, columns=['column', 'type']).head(10))

```

	column	type
0	_c0	int
1	longitude	double
2	latitude	double
3	housing_median_age	double
4	total_rooms	double
5	total_bedrooms	double
6	population	double
7	households	double
8	median_income	double
9	median_house_value	double

In [5]: *# display the main indecies by each column*

```
df_california.describe().show()
```

```
+-----+-----+-----+-----+-----+-----+-----+-----+
|summary|      _c0|    longitude|    latitude|housing_median_age|    total_rooms|    total_bedrooms|
|population|    households|    median_income|median_house_value|ocean_proximity|
+-----+-----+-----+-----+-----+-----+-----+-----+
| count|      20640|      20640|      20640|      20640|      20640|      20433|
20640|      20640|      20640|      20640|      20640|
| mean|      10319.5|-119.56970445736148| 35.6318614341087|28.639486434108527|2635.7630813953488| 537.8705525375618|1425.4
767441860465|499.5396802325581|3.8706710029070246|206855.81690891474|      null|
| stddev|5958.399113856003| 2.003531723502584|2.135952397457101| 12.58555761211163|2181.6152515827944|421.38507007403115| 113
2.46212176534|382.3297528316098| 1.899821717945263|115395.61587441359|      null|
| min|      0|      -124.35|      32.54|      1.0|      2.0|      1.0|
3.0|      1.0|      0.4999|      14999.0|      <1H OCEAN|
| max|      20639|      -114.31|      41.95|      52.0|      39320.0|      6445.0|
35682.0|      6082.0|      15.0001|      500001.0|      NEAR OCEAN|
+-----+-----+-----+-----+-----+-----+-----+-----+
-----+-----+-----+-----+-----+-----+-----+-----+
```

In [6]: *# nulls calculation*

```
columns = df_california.columns
```

```
for column in columns:
    print(column, df_california.filter((F.col(column)).isNull()).count())
```

```
_c0 0
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
```

### Data overview

1) dataset has 20640 rows and 11 columns:

- c0;
- longitude;
- latitude;
- housing\_median\_age;
- total\_rooms;
- total\_bedrooms;
- population;
- households;
- median\_income;
- median\_house\_value;
- ocean\_proximity;

2) only total\_bedrooms column has nulls (207 pcs) - to be filled with mean value;

```
In [7]: # fill up nulls with mean value
```

```
df_mean = df_california.select(F.mean(F.col('total_bedrooms')).alias('avg')).collect()
avg = df_mean[0]['avg']
avg
```

```
Out[7]: 537.8705525375618
```

```
In [8]: df_california = df_california.na.fill(avg,["total_bedrooms"])
```

```
In [9]: # result check
```

```
for column in columns:  
    print(column, df_california.filter((F.col(column)).isNull()).count())
```

```
_c0 0  
longitude 0  
latitude 0  
housing_median_age 0  
total_rooms 0  
total_bedrooms 0  
population 0  
households 0  
median_income 0  
median_house_value 0  
ocean_proximity 0
```

nulls are successfully filled

### data encoding

```
In [10]: # definition of target, categorical and numeric features
```

```
categorical_cols = ['ocean_proximity']  
numerical_cols = ['longitude', 'latitude', 'housing_median_age', 'total_rooms', 'total_bedrooms', 'population',  
                  'households', 'median_income']  
target = "median_house_value"
```

```
In [11]: # indexing of categorical features
```

```
indexer = StringIndexer(inputCols=categorical_cols,  
                        outputCols=[c+'_idx' for c in categorical_cols])  
df_california = indexer.fit(df_california).transform(df_california)  
  
cols = [c for c in df_california.columns for i in categorical_cols if (c.startswith(i))]  
df_california.select(cols).show(3)
```

```
+-----+-----+
|ocean_proximity|ocean_proximity_idx|
+-----+-----+
|      NEAR BAY|              3.0|
|      NEAR BAY|              3.0|
|      NEAR BAY|              3.0|
+-----+-----+
```

only showing top 3 rows

In [12]: *# features encoding (OHE)*

```
encoder = OneHotEncoder(inputCols=[c+'_idx' for c in categorical_cols],
                        outputCols=[c+'_ohe' for c in categorical_cols])
df_california = encoder.fit(df_california).transform(df_california)

cols = [c for c in df_california.columns for i in categorical_cols if (c.startswith(i))]
df_california.select(cols).show(3)
```

```
+-----+-----+-----+
|ocean_proximity|ocean_proximity_idx|ocean_proximity_ohe|
+-----+-----+-----+
|      NEAR BAY|              3.0|      (4,[3],[1.0])|
|      NEAR BAY|              3.0|      (4,[3],[1.0])|
|      NEAR BAY|              3.0|      (4,[3],[1.0])|
+-----+-----+-----+
```

only showing top 3 rows

In [13]: *# assembling of categorical features to vector*

```
categorical_assembler = \
    VectorAssembler(inputCols=[c+'_ohe' for c in categorical_cols],
                   outputCol="categorical_features")

df_california = categorical_assembler.transform(df_california)
```

In [14]: *# numeric features assembling*

```
numerical_assembler = VectorAssembler(inputCols=numerical_cols, outputCol="numerical_features")
df_california = numerical_assembler.transform(df_california)
```

```
In [15]: # assembling of numeric features to vector

standardScaler = StandardScaler(inputCol='numerical_features',outputCol="numerical_features_scaled")

df_california = standardScaler.fit(df_california).transform(df_california)
```

```
In [16]: # assembling all features to one vector

all_features = ['categorical_features','numerical_features_scaled']

final_assembler = VectorAssembler(inputCols=all_features,
                                   outputCol="features")

df_california = final_assembler.transform(df_california)

df_california.select(all_features).show(3)
```

```
+-----+-----+
|categorical_features|numerical_features_scaled|
+-----+-----+
|      (4,[3],[1.0])|      [-61.007269596069...|
|      (4,[3],[1.0])|      [-61.002278409814...|
|      (4,[3],[1.0])|      [-61.012260782324...|
+-----+-----+
only showing top 3 rows
```

```
In [17]: # split the data to train and test samples

train_data, test_data = df_california.randomSplit([.8,.2], seed=RANDOM_SEED)

print(train_data.count(), test_data.count())
```

16418 4222

**Data set was encoded using OHE and splitted on train and test samples (80/20)**

## Models training



## Model training on all features

```
In [18]: lr = LinearRegression(labelCol=target, featuresCol='features')  
  
model = lr.fit(train_data)
```

```
In [19]: # display the predictions  
  
predictions = model.transform(test_data)  
  
predictedLabels = predictions.select("median_house_value", "prediction")  
predictedLabels.show()
```

```
+-----+-----+  
|median_house_value|prediction|  
+-----+-----+  
|          352100.0|378451.33923285734|  
|          241400.0|256297.19652710436|  
|          281500.0|236503.62867485918|  
|          213500.0| 230527.8509058198|  
|          158700.0|187049.59724305058|  
|          162900.0| 206155.7409676565|  
|          105500.0|175487.30019459035|  
|          132000.0|166904.71510156244|  
|          122300.0| 187154.1320522707|  
|          109700.0| 222435.2679505013|  
|          188800.0|257374.81948872888|  
|          184400.0|225783.02583994344|  
|           97500.0|154052.28523706878|  
|          104200.0|156227.64811051264|  
|           83100.0|159666.03369625378|  
|           87500.0|166338.16028893227|  
|           80300.0|144110.84487898787|  
|           75700.0|231853.89445668738|  
|           76100.0|147770.86479645874|  
|           84400.0|138475.42500450555|  
+-----+-----+
```

only showing top 20 rows

```
In [20]: # calculation of RMSE
```

```
evaluator = RegressionEvaluator(labelCol="median_house_value",
                                predictionCol="prediction",
                                metricName="rmse")

rmse = evaluator.evaluate(predictions)
print("Root Mean Squared Error (RMSE) on test data = %g" % rmse)
```

Root Mean Squared Error (RMSE) on test data = 68865.7

```
In [21]: # calculation of MAE & R2

y_true = predictions.select("median_house_value").toPandas()
y_pred = predictions.select("prediction").toPandas()

mae_score = mean_absolute_error(y_true, y_pred)
r2_score = sklearn.metrics.r2_score(y_true, y_pred)
print('r2_score: {0}'.format(r2_score))
print('MAE:', mae_score)
```

r2\_score: 0.6411660947984178  
MAE: 50022.12558683282

```
In [22]: metric_1 = [['RMSE', model.summary.rootMeanSquaredError], ['MAE', model.summary.meanAbsoluteError], ['r2', model.summary.r2]]

metric_1 = pd.DataFrame(metric_1, columns = ['metric', 'value_model_1'])

metric_compare = metric_1.copy()

print(metric_1)
```

	metric	value_model_1
0	RMSE	68671.908505
1	MAE	49810.082827
2	r2	0.646514

### Model training using only numeric features

```
In [23]: lr_2 = LinearRegression(labelCol=target, featuresCol='numerical_features_scaled')

model_2 = lr_2.fit(train_data)
```

```
In [24]: # display of prediction results

predictions_2 = model_2.transform(test_data)
```

```
predictedLabels_2 = predictions_2.select("median_house_value", "prediction")
predictedLabels_2.show()
```

```
+-----+
|median_house_value|      prediction|
+-----+
|          352100.0| 379273.2311304626|
|          241400.0| 255340.989447698|
|          281500.0| 234631.03906910773|
|          213500.0| 228306.06401847536|
|          158700.0| 184063.5978175602|
|          162900.0| 203955.34583914792|
|          105500.0| 172855.5422547562|
|          132000.0| 164060.65154743195|
|          122300.0| 184957.58858521702|
|          109700.0| 220986.34058698127|
|          188800.0| 256531.68684420874|
|          184400.0| 224301.750674468|
|           97500.0| 151946.67873122543|
|          104200.0| 154273.898378307|
|           83100.0| 157299.68170536682|
|           87500.0| 163975.66869817115|
|           80300.0| 141492.15687689744|
|           75700.0| 230781.0269729062|
|           76100.0| 145056.29626445007|
|           84400.0| 135784.95513165276|
+-----+
only showing top 20 rows
```

```
In [25]: # calculation of RMSE

evaluator_2 = RegressionEvaluator(labelCol="median_house_value",
                                   predictionCol="prediction",
                                   metricName="rmse")

rmse_2 = evaluator_2.evaluate(predictions_2)
print("Root Mean Squared Error (RMSE_2) on test data = %g" % rmse)

Root Mean Squared Error (RMSE_2) on test data = 68865.7
```

```
In [26]: # calculation of MAE & R2
```

```

y_true_2 = predictions_2.select("median_house_value").toPandas()
y_pred_2 = predictions_2.select("prediction").toPandas()

mae_score_2 = mean_absolute_error(y_true_2, y_pred_2)
r2_score_2 = sklearn.metrics.r2_score(y_true_2, y_pred_2)
print('r2_score_2: {0}'.format(r2_score_2))
print('MAE_2:', mae_score_2)

```

```

r2_score_2: 0.6307916994983049
MAE_2: 51218.49600721165

```

```

In [27]: metric_2 = [['RMSE', model_2.summary.rootMeanSquaredError], ['MAE', model_2.summary.meanAbsoluteError], ['r2', model_2.summary.r2]]

metric_2 = pd.DataFrame(metric_2, columns = ['metric', 'value_model_2'])

metric_compare['value_model_2'] = metric_2.value_model_2

print(metric_2)

```

```

   metric  value_model_2
0    RMSE    69606.965461
1     MAE    50858.888642
2      r2         0.636822

```

## Analysis of the results and model testing

```

In [28]: metric_compare['best_value'] = np.minimum.reduce(metric_compare[['value_model_1', 'value_model_2']].values, axis=1)
print(metric_compare)

if metric_compare['best_value'][0] == metric_compare['value_model_1'][0]:
    print('\n', 'First model has best scores')
    print('\n', metric_1)
else:
    print('\n', 'Second model has best scores')
    print('\n', metric_2)

```

	metric	value_model_1	value_model_2	best_value
0	RMSE	68671.908505	69606.965461	68671.908505
1	MAE	49810.082827	50858.888642	49810.082827
2	r2	0.646514	0.636822	0.636822

First model has best scores

	metric	value_model_1
0	RMSE	68671.908505
1	MAE	49810.082827
2	r2	0.646514

**Comparison of score revealed, that first model has the better score.**

1) RMSE -  $68837.4 < \text{RMSE}_2 - 69975$

2) r2\_score:  $0.64 > \text{r2\_score}_2: 0.63$

3) MAE:  $49849 < \text{MAE}_2: 50848$

The first model is selected for the testing.

```
In [29]: # test predictions using best model (first 20 rows)

predictedLabels.show()
```

median_house_value	prediction
352100.0	378451.33923285734
241400.0	256297.19652710436
281500.0	236503.62867485918
213500.0	230527.8509058198
158700.0	187049.59724305058
162900.0	206155.7409676565
105500.0	175487.30019459035
132000.0	166904.71510156244
122300.0	187154.1320522707
109700.0	222435.2679505013
188800.0	257374.81948872888
184400.0	225783.02583994344
97500.0	154052.28523706878
104200.0	156227.64811051264
83100.0	159666.03369625378
87500.0	166338.16028893227
80300.0	144110.84487898787
75700.0	231853.89445668738
76100.0	147770.86479645874
84400.0	138475.42500450555

only showing top 20 rows

```
In [30]: # Model scores on the test sample:

print("Root Mean Squared Error (RMSE) on test data = %g" % rmse)
print('r2_score= {0}'.format(r2_score))
print('MAE =', mae_score)
```

```
Root Mean Squared Error (RMSE) on test data = 68865.7
r2_score= 0.6411660947984178
MAE = 50022.12558683282
```

```
In [31]: spark.stop()
```

## Conclusion

The scores of the best model on the test sample are following:

- Root Mean Squared Error (RMSE) on test data = 68865.7
- $r^2_{\text{score}} = 0.6411660947984178$
- MAE = 50022.12558683282

**After the completion of testing the spark session was stopped.**