Housing Price Prediction with **DoubleLink**

Allen Li, Ailing Shang, Kai-Yu Chen, Jiayi (Lily) Hu, Tom Do, Pei-Ying (Daphne) Wu, Winnie Chan

Table of Contents

Who are we?

Our team has a keen eye for identifying undervalued properties with great potential, and we take pride in transforming them into desirable homes that meet modern standards. With years of experience in the real estate industry, we are committed to delivering high-quality housing solutions that not only benefit our investors, but also provide comfortable and stylish living options for our buyers.

Our Business

What about DoubleLink?

Real-estate development company. Building and selling properties.

Our Objective

Get the accurate price of houses to allow for better pricing strategy and financial planning

Overview of **the data**

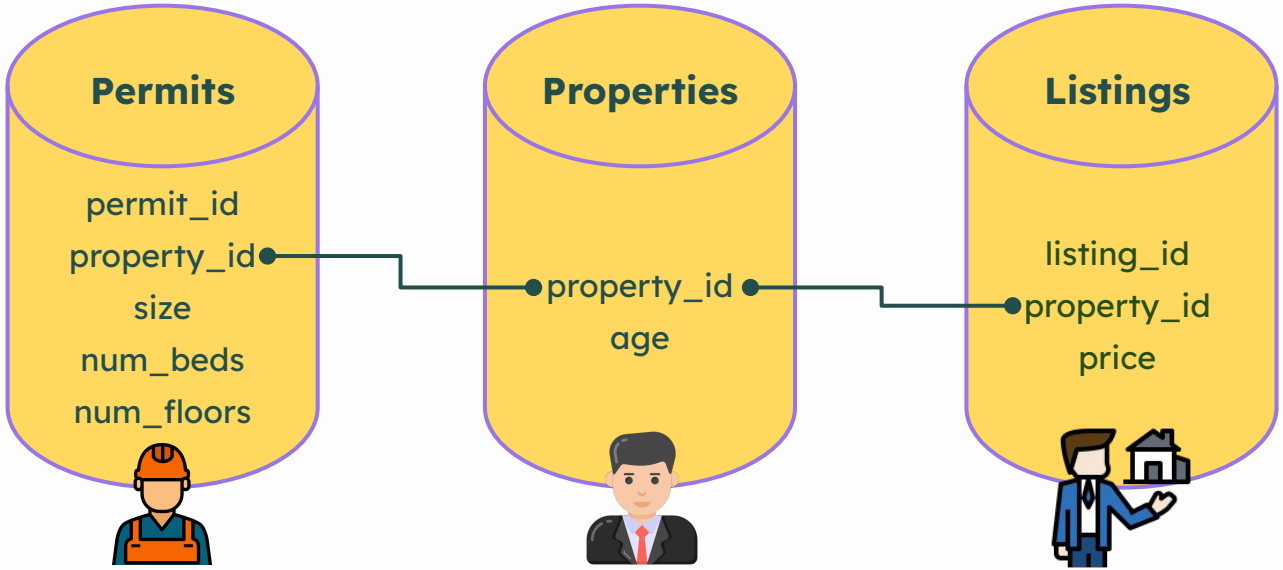
100 properties, 5 features

- Price
- Size
- Number of Beds
- Number of Floors
- House Age

Methodology

Analytics Team

Collections:



Schema for collection “properties”

```
# Create the "properties" collection with validation rules
db.create_collection("properties", validator={
    "$jsonSchema": {
        "bsonType": "object",
        "required": ["property_id", "age"],
        "properties": {
            "property_id": {
                "bsonType": "string",
                "pattern": "^PROP\\d{9}$",
                "description": "must be a string of exactly 10 characters"
            },
            "age": {
                "bsonType": "number",
                "minimum": 0,
                "description": "must be a number and is required"
            }
        }
    }
})
```

13-character long with prefix **“PROP”** and **9 trailing digits**

Must be a **positive integer**

Schema for collection “permits”

```
# Create the "permits" collection with validation rules
db = client["Project"]
db.create_collection("permits", validator={
    "$jsonSchema": {
        "bsonType": "object",
        "required": ["permit_id", "property_id", "size", "num_beds", "num_floors"],
        "properties": {
            "permit_id": {
                "bsonType": "string",
                "pattern": "^PERM\\d{9}$",
                "description": "must be a string and is required"
            },
            "property_id": {
                "bsonType": "string",
                "pattern": "^PROP\\d{9}$",
                "description": "must be a string and is required"
            },
            "size": {
                "bsonType": ["double", "int"],
                "minimum": 0,
                "description": "must be a number and is required"
            },
            "num_beds": {
                "bsonType": ["double", "int"],
                "minimum": 0,
                "description": "must be a number and is required"
            },
            "num_floors": {
                "bsonType": ["double", "int"],
                "minimum": 0,
                "multipleOf": 0.5,
                "description": "must be a number and is required"
            }
        }
    }
})
```

13-character long with prefix **“PERM”** and **9 trailing digits**

13-character long with prefix **“PROP”** and **9 trailing digits**

Must be a **positive number**

Must be a **positive integer**

Must be a **positive number** and can be **0.5 increment**

Schema for collection “listings”

```
# Create the "listings" collection with validation rules
db.create_collection("listings", validator={
    "$jsonSchema": {
        "bsonType": "object",
        "required": ["listing_id", "property_id", "price"],
        "properties": {
            "listing_id": {
                "bsonType": "string",
                "pattern": "^LIST\\d{9}$",
                "description": "must be a string and is required"
            },
            "property_id": {
                "bsonType": "string",
                "pattern": "^PROP\\d{9}$",
                "description": "must be a string and is required"
            },
            "price": {
                "bsonType": "number",
                "minimum": 0,
                "description": "must be a number and is required"
            }
        }
    }
})
```

13-character long with prefix **“LIST”** and **9 trailing digits**

13-character long with prefix **“PROP”** and **9 trailing digits**

Must be a **positive number**

Insert data to **the collections**

```python
def generate_id(prefix):
    """
    Generate a random ID with the beginning prefix.
    """
    prefix = prefix
    characters = string.digits
    while True:
        random_id = prefix + ''.join(random.choices(characters, k=9))
        if random_id not in generated_ids:
            generated_ids.add(random_id)
            return random_id
```

This code generates IDs with prefix either “LIST”, “PERM”, or “PROP” and 9 random trailing digits

```python
generated_ids = set() # Make sure IDs generated are not duplicated
for i in range(len(data)):

    # Initiate IDs:
    listing_id = generate_id("LIST")
    property_id = generate_id("PROP")
    permit_id = generate_id("PERM")

    # Insert data to listings
    db.listings.insert_one({"listing_id": listing_id,
                            "property_id": property_id,
                            "price": data.price[i]})

    # Insert data to permits
    db.permits.insert_one({"permit_id": permit_id,
                           "property_id": property_id,
                           "size": data["size"][i],
                           "num_beds": data["num_beds"][i],
                           "num_floors": data["num_floors"][i]})

    # Insert data to properties
    db.properties.insert_one({"property_id": property_id,
                              "age": data["age"][i]})
```

This code inserts records into the collection

Aggregate data for **regression analysis**

	size	num_beds	num_floors	age	price
0	952.0	2.0	1.0	65.0	271.5
1	1244.0	3.0	1.0	64.0	300.0
2	1947.0	3.0	2.0	17.0	509.8
3	1725.0	3.0	2.0	42.0	394.0
4	1959.0	3.0	2.0	15.0	540.0
...	...	...	...	...	...
95	1224.0	2.0	2.0	12.0	329.0
96	1432.0	2.0	1.0	43.0	388.0
97	1660.0	3.0	2.0	19.0	390.0
98	1212.0	3.0	1.0	20.0	356.0
99	1050.0	2.0	1.0	65.0	257.8

Output

```
# select database and collections
db = client["Project"]
properties = db["properties"]

# define aggregation pipeline
pipeline = [
  {"$lookup": {
   "from": "permits",
   "localField": "property_id",
   "foreignField": "property_id",
   "as": "permits"}},
  {"$unwind": "$permits"},
  {"$lookup": {
   "from": "listings",
   "localField": "property_id",
   "foreignField": "property_id",
   "as": "listings"}},
  {"$unwind": "$listings"},
  {"$project": {
   "_id": 0,
   "size": "$permits.size",
   "num_beds": "$permits.num_beds",
   "num_floors": "$permits.num_floors",
   "age": "$age",
   "price": "$listings.price"}}]

# execute aggregation pipeline
cursor = properties.aggregate(pipeline)

# convert cursor to list of dictionaries
docs = list(cursor)

# convert list of dictionaries to DataFrame
df = pd.DataFrame(docs)
```

Train/test **regression model**

```
#define predictors
X = df[["size", "num_beds", "num_floors", "age"]]
#define target variable
Y = df["price"]
```

```
#split into training set and testing set

from sklearn.model_selection import train_test_split


X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3, random_state=5)
```

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
linear_model = LinearRegression()
model_result = linear_model.fit(X_train, y_train)
y_train_predicted = linear_model.predict(X_train)

print('In-sample R-squared:', linear_model.score(X_train,y_train))
print('Out-sample R-squared:', linear_model.score(X_test,y_test))
```

```
In-sample R-squared: 0.9635410379772535
Out-sample R-squared: 0.931251388378938
```

```
import statsmodels.api as sm

ols = sm.OLS(y_train, sm.add_constant(X_train))
ols_result = ols.fit()
ols_result.summary()
```

Test size = 30%

In-sample R-squared: 0.9635

Out-of-sample R-squared: 0.9313

Key findings from **regression analysis**

- R-squared: 0.964
- Size is the only variable that has positive influence on the housing price

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.964
Model:	OLS	**Adj. R-squared:**	0.961
Method:	Least Squares	**F-statistic:**	429.5
Date:	Mon, 24 Apr 2023	**Prob (F-statistic):**	5.86e-46
Time:	18:04:57	**Log-Likelihood:**	-309.47
No. Observations:	70	**AIC:**	628.9
Df Residuals:	65	**BIC:**	640.2
Df Model:	4		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	195.4756	14.647	13.346	0.000	166.224	224.727
size	0.2677	0.009	29.786	0.000	0.250	0.286
num_beds	-24.0006	5.610	-4.278	0.000	-35.205	-12.796
num_floors	-63.8664	6.901	-9.255	0.000	-77.648	-50.085
age	-1.4382	0.107	-13.392	0.000	-1.653	-1.224

Omnibus:	2.698	**Durbin-Watson:**	2.189
Prob(Omnibus):	0.259	**Jarque-Bera (JB):**	2.047
Skew:	-0.404	**Prob(JB):**	0.359
Kurtosis:	3.221	**Cond. No.**	8.76e+03

Conclusion

Target property: 1,200 sq ft, 1 floor, 3 bedroom, 40 years old

```
input_data = pd.DataFrame({
    'size': [1200],
    'numberBedroom': [3],
    'numberFloor': [1],
    'age': [40]
})

predicted_price = float(linear_model.predict(input_data))
```

Our predicted housing price for the given set of features is

323.35 ~ (aka. $323,350)

$200k

BUY

THANK$!

Questions?

cu$tomer$ervice@doublelink.com
+1 999 999 9999
doublelink.com