# Linear-regression-prediction-by-retzam-ai.ipynb

June 16, 2024

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from imblearn.over_sampling import RandomOverSampler
from joblib import dump
```

[2]:
```
df = pd.read_csv('housing_price_dataset.csv')
df.head()
```

[2]:

| | SquareFeet | Bedrooms | Bathrooms | Neighborhood | YearBuilt | Price |
|---|---|---|---|---|---|---|
| 0 | 2126 | 4 | 1 | Rural | 1969 | 215355.283618 |
| 1 | 2459 | 3 | 2 | Rural | 1980 | 195014.221626 |
| 2 | 1860 | 2 | 1 | Suburb | 1970 | 306891.012076 |
| 3 | 2294 | 2 | 1 | Urban | 1996 | 206786.787153 |
| 4 | 2130 | 5 | 2 | Suburb | 2001 | 272436.239065 |

[3]:
```
unique_values = df['Neighborhood'].unique()
unique_values
```

[3]:
```
array(['Rural', 'Suburb', 'Urban'], dtype=object)
```

[4]:
```
# Create a map using the unique values array above.
mapping = {
   'Rural': 0,
   'Suburb': 1,
   'Urban': 2,
}

# Replace the values, so we have nominal data that our model understands.
df['Neighborhood'] = df['Neighborhood'].replace(mapping)
```

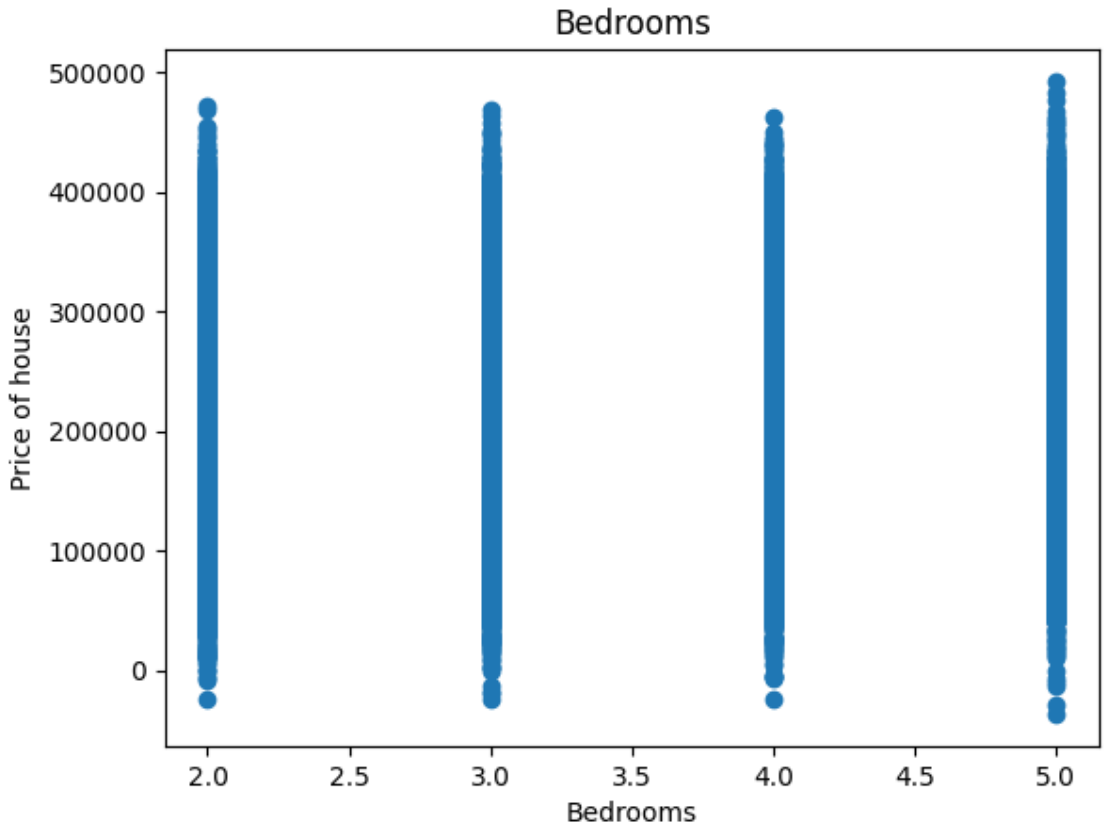
[5]:
```
df.head()
```

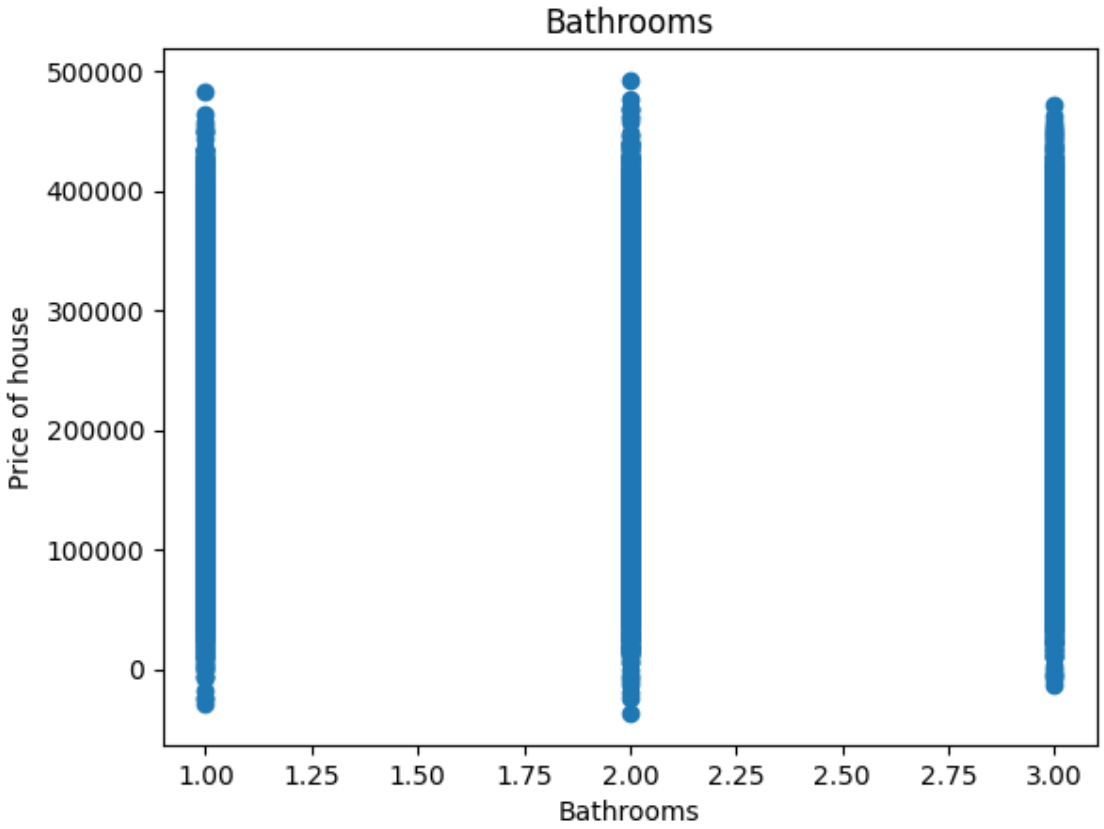
[5]:

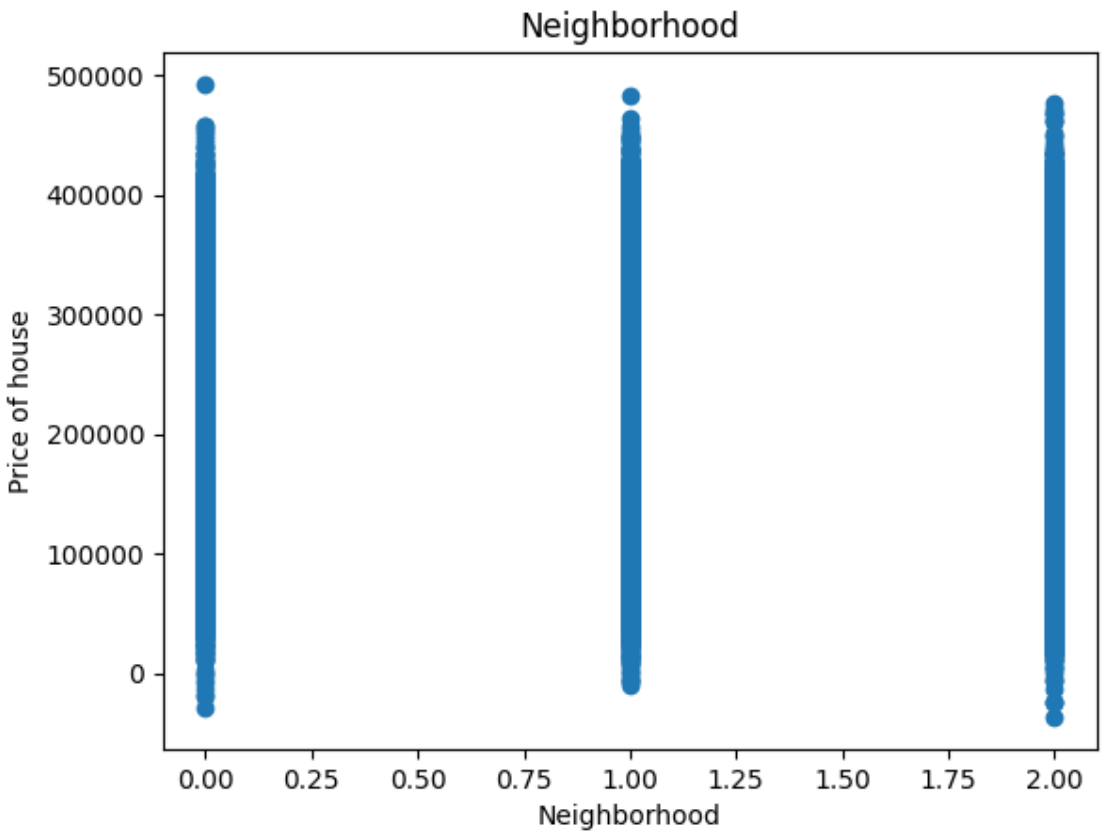
| | SquareFeet | Bedrooms | Bathrooms | Neighborhood | YearBuilt | Price |
|---|---|---|---|---|---|---|
| 0 | 2126 | 4 | 1 | 0 | 1969 | 215355.283618 |
| 1 | 2459 | 3 | 2 | 0 | 1980 | 195014.221626 |
| 2 | 1860 | 2 | 1 | 1 | 1970 | 306891.012076 |

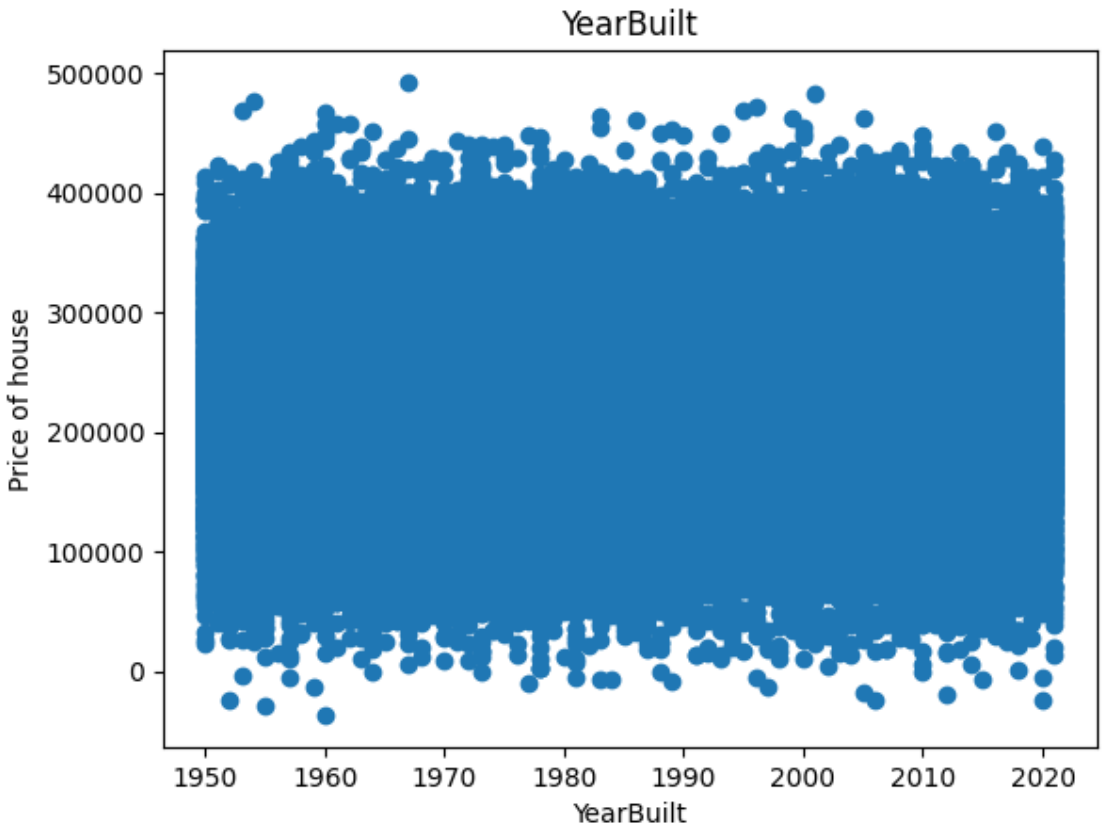
| | | | | | | |
|---|---|---|---|---|---|---|
| 3 | 2294 | 2 | 1 | 2 | 1996 | 206786.787153 |
| 4 | 2130 | 5 | 2 | 1 | 2001 | 272436.239065 |

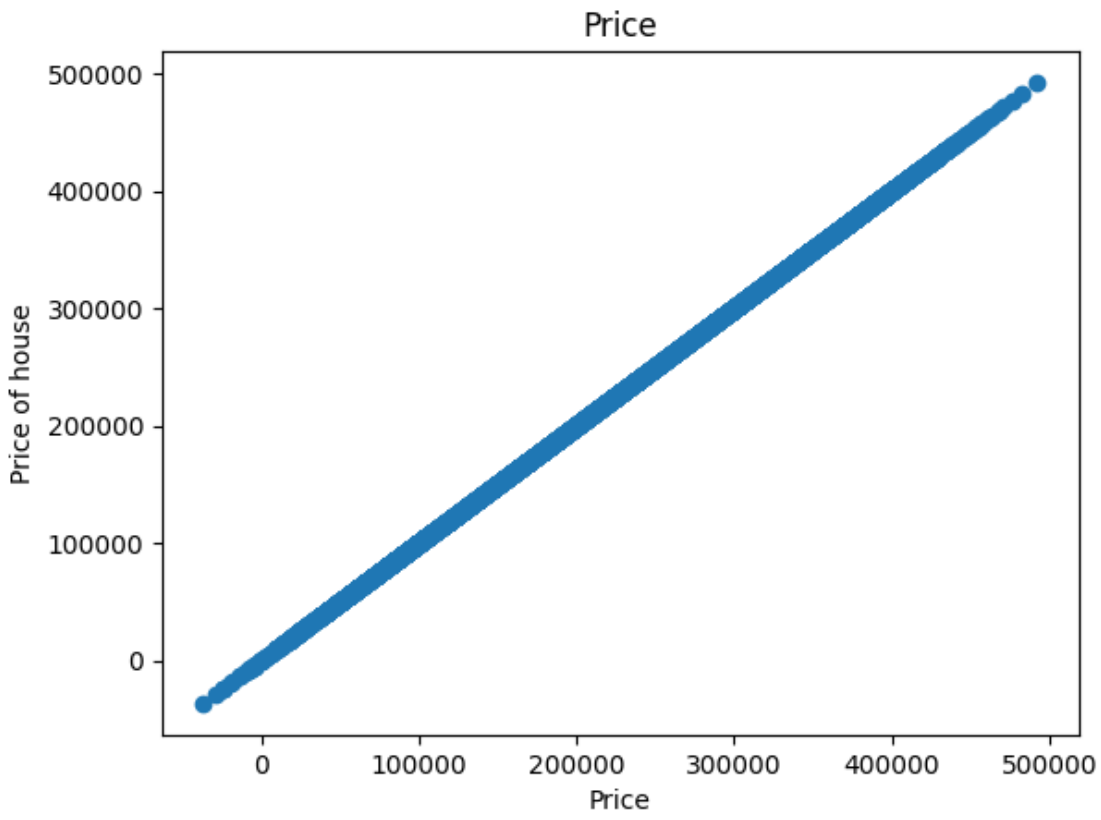
[6]:
```
# We plot a histogram to check which features affect the outcome the most or␣
 ↪the least
# This helps us determine, which features to use in training our model and the␣
 ↪ones to discard

for label in df.columns[1:]:
  plt.scatter(df[label], df["Price"])
  plt.title(label)
  plt.ylabel("Price of house")
  plt.xlabel(label)
  plt.show()
```




Bathrooms
Price of house
500000
400000
300000
200000
100000
0
1.00
1.25
1.50
1.75
2.00
2.25
2.50
2.75
3.00
Bathrooms


Neighborhood
500000
400000
300000
200000
100000
0
Price of house
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
Neighborhood


YearBuilt
500000
400000
300000
200000
100000
0
Price of house
1950
1960
1970
1980
1990
2000
2010
2020
YearBuilt




[7]:
```
# Split our train, val and test sets in 60, 20, 20 percents respectively
train, val, test = np.split(df.sample(frac=1), [int(0.6*len(df)), int(0.
 ↪8*len(df))])
```

[8]:
```
import copy
```

[9]:
```
""" Scale dataset so better prediction can be made.
    Specifically to separate the features (X) and the target variable (y) from␣
 ↪the dataset.
"""
def get_xy(dataframe, y_label, x_labels=None):

  # Copy dataset
  dataframe = copy.deepcopy(dataframe)

  # If x_labes is not provided, use all columns except the y_label
  if x_labels is None:
    X = dataframe[[c for c in dataframe.columns if c != y_label]].values
  else:
    # if x_labels is 1 reshape to a 2D array, else get the values
```

```
    if len(x_labels) == 1:
      X = dataframe[x_labels[0]].values.reshape(-1, 1)
    else:
      X = dataframe[x_labels].values

    y = dataframe[y_label].values.reshape(-1, 1)

    # Horizontally stack X and y in each column.
    # Where the last item is the target variable (y)
    data = np.hstack((X, y))

    return data, X, y
```

[10]:
```
# Simple Linear Regression
```

[11]:
```
# Process data to train, val, and test datasets for a simple linear regression␣
 ↪model
# That's why we are using only 1 feature (x_labels)
_, X_train_temp, y_train_temp = get_xy(train, "Price", x_labels=["SquareFeet"])
_, X_val_temp, y_val_temp = get_xy(val, "Price", x_labels=["SquareFeet"])
_, X_test_temp, y_test_temp = get_xy(test, "Price", x_labels=["SquareFeet"])
```

[12]:
```
# We'll import and use a regression model from sklearn
from sklearn.linear_model import LinearRegression
```

[13]:
```
simple_reg = LinearRegression()
# Train simple linear regression model
simple_reg.fit(X_train_temp, y_train_temp)
```

[13]:
```
LinearRegression()
```

[14]:
```
# Save the model using joblib
dump(simple_reg, "simple-linear-regression")
```

[14]:
```
['simple-linear-regression']
```

[15]:
```
# Score the model to get its accuracy based on variance, bias etc.
# Using R Squared(R^2) score.
simple_reg.score(X_test_temp, y_test_temp)
```

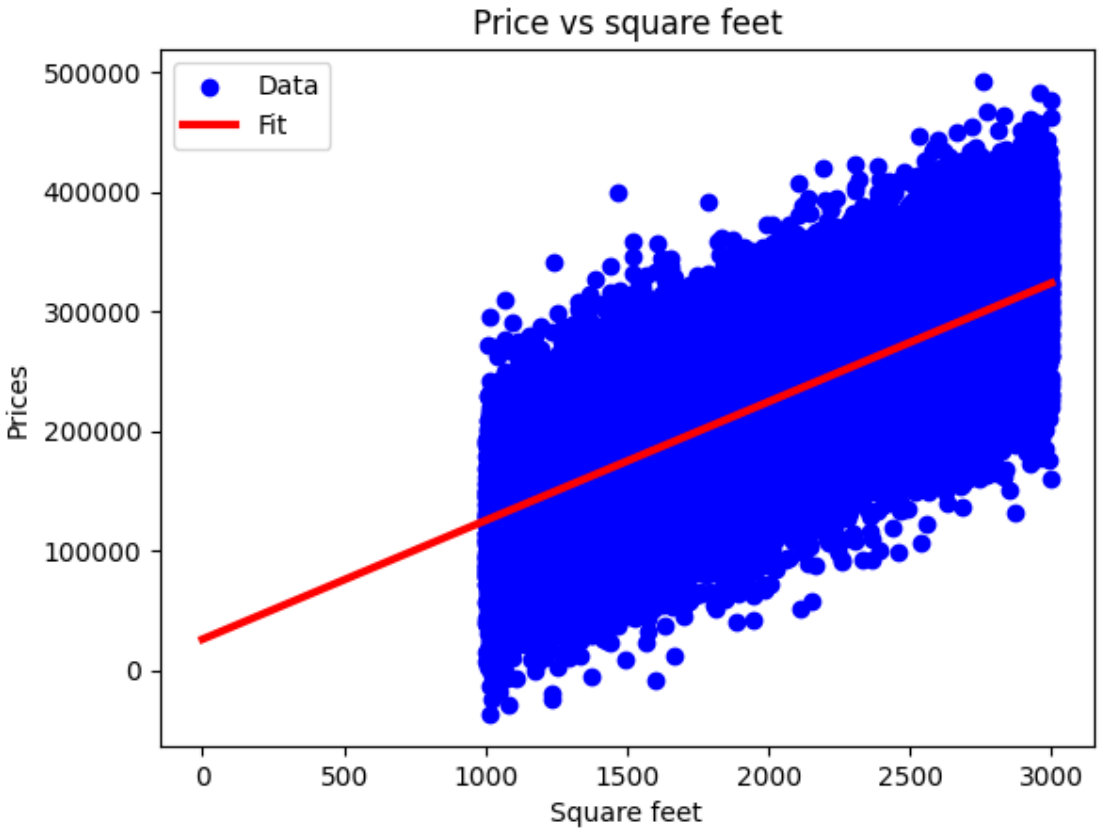
[15]:
```
0.5691031790851901
```

[16]:
```
import tensorflow as tf
```

```
2024-06-16 08:31:32.442422: I tensorflow/core/platform/cpu_feature_guard.cc:182]
This TensorFlow binary is optimized to use available CPU instructions in
performance-critical operations.
```

```
To enable the following instructions: AVX2 FMA, in other operations, rebuild
TensorFlow with the appropriate compiler flags.
```

[17]:
```
# Lets graphically show our simple linear regression model using a scatter graph
# Showing the line of best fit, using tensorflow
plt.scatter(X_train_temp, y_train_temp, label="Data", color="blue")
x = tf.linspace(0, 3000, 5000)
plt.plot(x, simple_reg.predict(np.array(x).reshape(-1, 1)), label="Fit",␣
 ↪color="red", linewidth=3)
plt.legend()
plt.title("Price vs square feet")
plt.ylabel("Prices")
plt.xlabel("Square feet")
plt.show()
```




[18]:
```
# Multiple Linear Regression
```

[19]:
```
# Process data to train, val, and test datasets for a multiple linear␣
 ↪regression model
# That's why we are passing all features except the target variable. (x_labels)
```

```
_, X_train_all, y_train_all = get_xy(train, "Price", x_labels=df.columns[1:])
_, X_val_all, y_val_all = get_xy(val, "Price", x_labels=df.columns[1:])
_, X_test_all, y_test_all = get_xy(test, "Price", x_labels=df.columns[1:])
```

[20]:
```
multiple_reg = LinearRegression()
# Train the multiple linear regression model
multiple_reg.fit(X_train_all, y_train_all)
```

[20]:
```
LinearRegression()
```

[21]:
```
# Save the model using joblib
dump(multiple_reg, "multiple-linear-regression")
```

[21]:
```
['multiple-linear-regression']
```

[22]:
```
# Score the model to get its accuracy based on variance, bias etc.
# Using R Squared(R^2) score.
multiple_reg.score(X_test_all, y_test_all)
```

[22]:
```
1.0
```

[ ]: