exercise-module-52

October 30, 2024

1 Module 52: Hypothesis Testing

Author: Juliho Castillo Colmenares

[15]:
```
# Import necessary libraries
import pandas as pd
import numpy as np
import statsmodels.api as sm
from sklearn.model_selection import train_test_split
```

[16]:
```
# Load the dataset
data = pd.read_csv("kc_house_data.csv")
data.head()
```

[16]:
```
           id             date     price  bedrooms  bathrooms  sqft_living  \
0  7129300520  20141013T000000  221900.0         3       1.00         1180
1  6414100192  20141209T000000  538000.0         3       2.25         2570
2  5631500400  20150225T000000  180000.0         2       1.00          770
3  2487200875  20141209T000000  604000.0         4       3.00         1960
4  1954400510  20150218T000000  510000.0         3       2.00         1680

   sqft_lot  floors  waterfront  view  …  grade  sqft_above  sqft_basement  \
0      5650     1.0           0     0  …      7        1180              0
1      7242     2.0           0     0  …      7        2170            400
2     10000     1.0           0     0  …      6         770              0
3      5000     1.0           0     0  …      7        1050            910
4      8080     1.0           0     0  …      8        1680              0

   yr_built  yr_renovated  zipcode      lat     long  sqft_living15  \
0      1955             0    98178  47.5112 -122.257           1340
1      1951          1991    98125  47.7210 -122.319           1690
2      1933             0    98028  47.7379 -122.233           2720
3      1965             0    98136  47.5208 -122.393           1360
4      1987             0    98074  47.6168 -122.045           1800

   sqft_lot15
0        5650
1        7639
```

```
2       8062
3       5000
4       7503

[5 rows x 21 columns]
```

[17]:
```python
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   id             21613 non-null  int64
 1   date           21613 non-null  object
 2   price          21613 non-null  float64
 3   bedrooms       21613 non-null  int64
 4   bathrooms      21613 non-null  float64
 5   sqft_living    21613 non-null  int64
 6   sqft_lot       21613 non-null  int64
 7   floors         21613 non-null  float64
 8   waterfront     21613 non-null  int64
 9   view           21613 non-null  int64
 10  condition      21613 non-null  int64
 11  grade          21613 non-null  int64
 12  sqft_above     21613 non-null  int64
 13  sqft_basement  21613 non-null  int64
 14  yr_built       21613 non-null  int64
 15  yr_renovated   21613 non-null  int64
 16  zipcode        21613 non-null  int64
 17  lat            21613 non-null  float64
 18  long           21613 non-null  float64
 19  sqft_living15  21613 non-null  int64
 20  sqft_lot15     21613 non-null  int64
dtypes: float64(5), int64(15), object(1)
memory usage: 3.5+ MB
```

[18]:
```python
data["date"] = pd.to_datetime(data["date"])
data.head()
```

[18]:
```
           id       date     price  bedrooms  bathrooms  sqft_living  \
0  7129300520 2014-10-13  221900.0         3       1.00         1180
1  6414100192 2014-12-09  538000.0         3       2.25         2570
2  5631500400 2015-02-25  180000.0         2       1.00          770
3  2487200875 2014-12-09  604000.0         4       3.00         1960
4  1954400510 2015-02-18  510000.0         3       2.00         1680

   sqft_lot  floors  waterfront  view  …  grade  sqft_above  sqft_basement  \
```

```
0     5650    1.0           0      0  ...      7        1180              0
1     7242    2.0           0      0  ...      7        2170            400
2    10000    1.0           0      0  ...      6         770              0
3     5000    1.0           0      0  ...      7        1050            910
4     8080    1.0           0      0  ...      8        1680              0

   yr_built  yr_renovated  zipcode      lat     long  sqft_living15  \
0      1955             0    98178  47.5112 -122.257           1340
1      1951          1991    98125  47.7210 -122.319           1690
2      1933             0    98028  47.7379 -122.233           2720
3      1965             0    98136  47.5208 -122.393           1360
4      1987             0    98074  47.6168 -122.045           1800

   sqft_lot15
0        5650
1        7639
2        8062
3        5000
4        7503

[5 rows x 21 columns]
```

[19]:
```python
# Step 1: Correlation analysis to preselect relevant features
correlation_threshold = 0.1
correlations = data.corr()["price"].abs()
selected_features = (
    correlations[correlations > correlation_threshold].index.drop("price").
 ↪tolist()
)
```

[20]:
```python
# Prepare variables for the model
X = data[selected_features]
y = data["price"]
```

[21]:
```python
# Add a constant to X for the intercept term
X = sm.add_constant(X)
```

[22]:
```python
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=0)
```

[23]:
```python
# Approach 1: Initial model fit and hypothesis testing
model = sm.OLS(y_train, X_train).fit()
print("Initial Model:\n", model.summary())
```

```
Initial Model:
                             OLS Regression Results
==============================================================================
```

```
Dep. Variable:                  price   R-squared:                       0.663
Model:                            OLS   Adj. R-squared:                  0.663
Method:                 Least Squares   F-statistic:                     3088.
Date:                Wed, 30 Oct 2024   Prob (F-statistic):               0.00
Time:                        12:31:33   Log-Likelihood:            -2.3693e+05
No. Observations:               17290   AIC:                         4.739e+05
Df Residuals:                   17278   BIC:                         4.740e+05
Df Model:                          11
Covariance Type:            nonrobust
================================================================================
=
                    coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------------------
-
const          -3.23e+07   5.76e+05    -56.038      0.000   -3.34e+07
-3.12e+07
bedrooms      -2.475e+04   2223.890    -11.131      0.000   -2.91e+04
-2.04e+04
bathrooms     -6735.6975   3649.897     -1.845      0.065   -1.39e+04
418.470
sqft_living     130.0585      2.652     49.034      0.000     124.859
135.258
floors        -2.756e+04   4115.392     -6.696      0.000   -3.56e+04
-1.95e+04
waterfront     6.121e+05   2.02e+04     30.294      0.000    5.73e+05
6.52e+05
view           6.464e+04   2525.197     25.597      0.000    5.97e+04
6.96e+04
grade          8.049e+04   2499.260     32.204      0.000    7.56e+04
8.54e+04
sqft_above       66.0563      2.597     25.440      0.000      60.967
71.146
sqft_basement    64.0022      3.088     20.729      0.000      57.950
70.054
yr_renovated     58.9956      4.077     14.470      0.000      51.004
66.987
lat             6.71e+05   1.22e+04     55.188      0.000    6.47e+05
6.95e+05
sqft_living15     9.1956      3.983      2.309      0.021       1.388
17.003
==============================================================================
Omnibus:                    14475.158   Durbin-Watson:                   2.007
Prob(Omnibus):                  0.000   Jarque-Bera (JB):          1291257.946
Skew:                           3.514   Prob(JB):                         0.00
Kurtosis:                      44.749   Cond. No.                     4.62e+15
==============================================================================
```

```
Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The smallest eigenvalue is 1.08e-20. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

[24]:
```python
# Approach 2: Stepwise Selection Process
def stepwise_selection(X, y, significance_level=0.05):
    initial_features = X.columns.tolist()
    best_features = []

    while len(initial_features) > 0:
        # Test each combination of features
        remaining_features = list(set(initial_features) - set(best_features))
        new_pval = pd.Series(index=remaining_features)

        for new_column in remaining_features:
            model = sm.OLS(y, sm.add_constant(X[best_features + [new_column]])).
 ↪fit()
            new_pval[new_column] = model.pvalues[new_column]

        # Identify feature with the lowest p-value
        min_p_value = new_pval.min()
        if min_p_value < significance_level:
            best_features.append(new_pval.idxmin())
        else:
            break

    return best_features
```

[25]:
```python
# Execute Stepwise Selection on training data
selected_features_stepwise = stepwise_selection(X_train, y_train)
X_train_stepwise = X_train[selected_features_stepwise]
X_test_stepwise = X_test[selected_features_stepwise]
```

[26]:
```python
# Final model fit using selected features
final_model = sm.OLS(y_train, sm.add_constant(X_train_stepwise)).fit()
print("Final Model with Stepwise Selection:\n", final_model.summary())
```

```
Final Model with Stepwise Selection:
                             OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.663
Model:                            OLS   Adj. R-squared:                  0.663
Method:                 Least Squares   F-statistic:                     3774.
Date:                Wed, 30 Oct 2024   Prob (F-statistic):               0.00
Time:                        12:31:34   Log-Likelihood:            -2.3693e+05
No. Observations:               17290   AIC:                         4.739e+05
```

```
Df Residuals:                   17280   BIC:                         4.740e+05
Df Model:                           9
Covariance Type:            nonrobust
================================================================================
=
                  coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------------------
-
grade            7.99e+04   2450.537     32.604      0.000    7.51e+04
8.47e+04
sqft_living      192.5027      3.584     53.715      0.000     185.478
199.527
const          -3.231e+07   5.69e+05    -56.826      0.000   -3.34e+07
-3.12e+07
waterfront      6.126e+05   2.02e+04     30.340      0.000    5.73e+05
6.52e+05
lat             6.712e+05    1.2e+04     56.021      0.000    6.48e+05
6.95e+05
view            6.443e+04   2473.284     26.052      0.000    5.96e+04
6.93e+04
yr_renovated      58.7711      4.076     14.420      0.000      50.782
66.760
bedrooms       -2.561e+04   2180.724    -11.745      0.000   -2.99e+04
-2.13e+04
floors         -2.917e+04   3444.219     -8.468      0.000   -3.59e+04
-2.24e+04
sqft_living15      9.8914      3.896      2.539      0.011       2.255
17.528
==============================================================================
Omnibus:                    14479.294   Durbin-Watson:                   2.007
Prob(Omnibus):                  0.000   Jarque-Bera (JB):          1290208.958
Skew:                           3.516   Prob(JB):                         0.00
Kurtosis:                      44.731   Cond. No.                     1.06e+06
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 1.06e+06. This might indicate that there are
strong multicollinearity or other numerical problems.
```

[27]:

```
# Approach 3: Model validation on test data
y_pred = final_model.predict(sm.add_constant(X_test_stepwise))
r2 = final_model.rsquared
adjusted_r2 = final_model.rsquared_adj
mse = np.mean((y_test - y_pred) ** 2)
```

```python
mae = np.mean(abs(y_test - y_pred))

print(f"R-squared: {r2}")
print(f"Adjusted R-squared: {adjusted_r2}")
print(f"Mean Squared Error (MSE): {mse}")
print(f"Mean Absolute Error (MAE): {mae}")
```

```
R-squared: 0.6627890581265967
Adjusted R-squared: 0.6626134274277043
Mean Squared Error (MSE): 40982815045.67618
Mean Absolute Error (MAE): 130492.11260494834
```

1.0.1 Initial Model

- **Purpose**: The initial model includes all preselected features that show some correlation with price. This is a broad model meant to capture all potential predictors for price without filtering out less significant ones yet.
- **Output**: The summary provides coefficients and p-values for each feature. Coefficients show the estimated impact of each feature on the price (positive or negative influence), while p-values indicate the significance level, helping identify which features may be statistically impactful.

1.0.2 Stepwise Selection

- **Method**: Stepwise selection is applied to iteratively add or remove features based on their p-values. Features with a p-value below 0.05 are retained, as they meet the conventional threshold for statistical significance.
- **Goal**: By narrowing down to only the most significant predictors, the model becomes more robust and focused, improving predictive accuracy. This step eliminates noise from variables that have a weak relationship with price.

1.0.3 Validation Metrics

These metrics evaluate model performance on test data to gauge how well the model generalizes:

- **R-squared (R^2)**: Represents the proportion of variability in the price that the model explains. Higher values indicate better fit.
- **Adjusted R-squared**: Adjusts R^2 to account for the number of predictors, which prevents artificially inflated R^2 values when adding more variables.
- **Mean Squared Error (MSE) and Mean Absolute Error (MAE)**: Both are measures of prediction error:
  - **MSE** penalizes larger errors more heavily (squared errors).
  - **MAE** provides the average magnitude of prediction errors, making it easier to interpret as it's in the same units as the target variable, house price.

These metrics collectively help interpret the model's accuracy, highlighting how well it predicts house prices and the precision of its predictions on unseen data.