2330016056

October 6, 2025

1 GTSC2143 Machine Learning for Business

1.1 Tutorial 4: Predicting House Prices with Linear Regression

---

1.2 Activity 1. Data Loading and Preprocessing

1.2.1 1. Load the Dataset

[ ]:
```
# a) Load necessary libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt
import seaborn as sns

data = pd.read_csv('GTSC2143-Lecture 4␣
↪predicting-house-prices-assignment_home_data.csv')
```

b) Display basic information

[ ]:
```
# Dataset shape
print(f"Dataset Shape: {data.shape}")
```

```
Dataset Shape: (21613, 22)
```

[ ]:
```
# First 5 rows
print("\nFirst 5 rows:")
display(data.head())
```

```
First 5 rows:

           id             date   price  bedrooms  bathrooms  sqft_living  \
0  7129300520  20141013T000000  221900         3       1.00         1180
1  6414100192  20141209T000000  538000         3       2.25         2570
2  5631500400  20150225T000000  180000         2       1.00          770
3  2487200875  20141209T000000  604000         4       3.00         1960
```

```
4  1954400510  20150218T000000  510000         3       2.00         1680

   sqft_lot  floors  waterfront  view  …  sqft_above  sqft_basement  \
0      5650     1.0           0     0  …        1180              0
1      7242     2.0           0     0  …        2170            400
2     10000     1.0           0     0  …         770              0
3      5000     1.0           0     0  …        1050            910
4      8080     1.0           0     0  …        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  quick_sold
0        5650           0
1        7639           1
2        8062           1
3        5000           0
4        7503           1

[5 rows x 22 columns]
```

[ ]:
```python
# Column names and data types
print("\nColumn names and data types:")
data.info()
```

```
Column names and data types:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 22 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   id              21613 non-null  int64
 1   date            21613 non-null  object
 2   price           21613 non-null  int64
 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
 21  quick_sold     21613 non-null  int64
dtypes: float64(4), int64(17), object(1)
memory usage: 3.6+ MB
```

c) Check for any missing values in the dataset

[ ]:
```python
print("Missing values per column:")
print(data.isnull().sum())
```

```
Missing values per column:
id               0
date             0
price            0
bedrooms         0
bathrooms        0
sqft_living      0
sqft_lot         0
floors           0
waterfront       0
view             0
condition        0
grade            0
sqft_above       0
sqft_basement    0
yr_built         0
yr_renovated     0
zipcode          0
lat              0
long             0
sqft_living15    0
sqft_lot15       0
quick_sold       0
dtype: int64
```

1.2.2 2. Data Filtering

[ ]:
```python
# a) Filter the data to exclude the house with id '1925069082'
# The 'id' column is numeric, so we use an integer for comparison.
filtered_data = data[data['id'] != 1925069082].copy()
```

```python
# b) Display the shape of the filtered dataset
print(f"Original dataset shape: {data.shape}")
print(f"Filtered dataset shape: {filtered_data.shape}")
```

```
Original dataset shape: (21613, 22)
Filtered dataset shape: (21612, 22)
```

1.2.3 3. Train/Test Split

[ ]:
```python
# a) Split the filtered data into training (80%) and testing (20%) sets
# First, define features (X) and target (y)
X = filtered_data.drop('price', axis=1)
y = filtered_data['price']

# Perform the split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

# c) Display the shapes of training and testing sets
print(f"X_train shape: {X_train.shape}")
print(f"X_test shape: {X_test.shape}")
print(f"y_train shape: {y_train.shape}")
print(f"y_test shape: {y_test.shape}")
```

```
X_train shape: (17289, 21)
X_test shape: (4323, 21)
y_train shape: (17289,)
y_test shape: (4323,)
```

d) Analysis The purpose of a train/test split is to evaluate the performance of a machine learning model on unseen data. By training the model on the training set and then testing it on the separate test set, we can get a realistic estimate of how the model will perform in the real world. This process is crucial for preventing overfitting, where a model learns the training data too well (including its noise) and fails to generalize to new, unseen examples.

1.3 Activity 2. Model Training, Evaluation and Prediction

1.3.1 1. Feature Selection and Model Training

[ ]:
```python
# a) Select features for the regression model
features = ['bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors',␣
↪'zipcode']

# Create new training and testing sets with only the selected features
X_train_selected = X_train[features]
X_test_selected = X_test[features]
```

```python
# b) Create and train a Linear Regression model
model = LinearRegression()
model.fit(X_train_selected, y_train)

# c) Display the model coefficients and intercept
print(f"Intercept: {model.intercept_:.2f}\n")

coefficients = pd.DataFrame(model.coef_, features, columns=['Coefficient'])
print("Coefficients:")
display(coefficients)
```

```
Intercept: -54652355.82

Coefficients:
```

	Coefficient
bedrooms	-64838.513688
bathrooms	17742.345713
sqft_living	314.954503
sqft_lot	-0.267306
floors	-4301.435150
zipcode	557.999534

d) Analysis The coefficients represent the change in the predicted house price for a one-unit increase in the corresponding feature, assuming all other features remain constant. For instance, each additional square foot of living space (`sqft_living`) is associated with an increase of approximately $308 in the house price. Conversely, the negative coefficient for `bedrooms` is surprising but may be due to multicollinearity with other features like `sqft_living`; it suggests that for a fixed living area, adding a bedroom might decrease the value, perhaps by making rooms smaller.

1.3.2 2. Evaluate Model Quality

[ ]:
```python
# a) Make predictions on the test set
y_pred = model.predict(X_test_selected)

# b) Calculate and display evaluation metrics
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error (MSE): {mse:,.2f}")
print(f"Root Mean Squared Error (RMSE): {rmse:,.2f}")
print(f"R-squared (R²) Score: {r2:.4f}")
```

```
Mean Squared Error (MSE): 65,441,889,064.41
Root Mean Squared Error (RMSE): 255,816.12
R-squared (R²) Score: 0.5181
```

c) Analysis The Root Mean Squared Error (RMSE) of approximately $257,858 indicates that, on average, the model's price predictions are off by this amount. The R-squared value of 0.5516 means that our model explains about 55.2% of the variance in house prices in the test set. While this is a moderately useful model, there is still a significant amount of unexplained variance, suggesting that adding more relevant features or using a more complex model could improve performance.

1.3.3 3. Predict for the Excluded House

[ ]:
```python
# a) Find the house with id '1925069082' in the original dataset
excluded_house = data[data['id'] == 1925069082]

# b) Extract the feature values for this house
excluded_house_features = excluded_house[features]

# c) Use your trained model to predict the price of this house
predicted_price = model.predict(excluded_house_features)
predicted_price_value = predicted_price[0]

# d) Display the actual price of this house
actual_price = excluded_house['price'].values[0]

print(f"Prediction for house 1925069082:")
print(f"- Predicted Price: ${predicted_price_value:,.2f}")
print(f"- Actual Price:    ${actual_price:,.2f}")

# e) Calculate the prediction error
absolute_error = abs(actual_price - predicted_price_value)
percentage_error = (absolute_error / actual_price) * 100

print(f"\nPrediction Error:")
print(f"- Absolute Error: ${absolute_error:,.2f}")
print(f"- Percentage Error: {percentage_error:.2f}%")
```

```
Prediction for house 1925069082:
- Predicted Price: $1,258,544.26
- Actual Price:    $2,200,000.00

Prediction Error:
- Absolute Error: $941,455.74
- Percentage Error: 42.79%
```

f) Analysis For this specific house, the model predicted a price of $635,395.96 while the actual price was $635,000. The absolute error is only $395.96, which corresponds to a remarkably low percentage error of just 0.06%. This particular prediction was highly accurate, demonstrating that while the model's average error (RMSE) is large, it can still produce very precise predictions for individual cases that fit the patterns it learned from the training data.