

Regression & Prediction

Theory and Practice with House Prices

Your Name

February 23, 2025

xAI

Our Housing Journey

- Starting Simple:
Foundations
- Expanding the Scope:
Complexity
- Refining Precision:
Optimization
- Facing Challenges: Pitfalls
- Insights & Next Steps:
Applications

Focus

Unpacking King County house prices with rich theory and dual R/Python implementations

The Quest to Understand Relationships

- Imagine you are exploring data to understand how different factors relate to each other.
- That's regression: a tool to ask, "How does Y change with X , and can we predict it?"
- In very basically It's the bridge between stats, where we explain the past, and data science, where we predict the future—our journey begins here.

Starting Simple

The Housing Puzzle

- **Objective:** Decode drivers of house prices in King County—size, location, features
- **Regression:** A statistical lens linking price (Y) to predictors like size (X)
- **Purpose:** Explain historical sales patterns and forecast future values for buyers and assessors

Simple Linear Regression: Theory & R

- **Theory:** Models a straight-line relationship: $Y = b_0 + b_1X + e$
- b_0 : Base price when size is zero;
 b_1 : Price increase per sq ft; e : Random error
- Assumes linearity and independence—foundation of regression

```
1 # R
2 simple_lm <- lm(AdjSalePrice ~ SqFtTotLiving, data
3               = house_df)
4 # Create the plot
5 ggplot(house_df, aes(x = sqft_living, y = price)) +
6   geom_point(alpha = 0.5) + # Scatter plot
7   geom_smooth(method = "lm", color = "blue", se =
8               FALSE) + # Regression line
9   labs(title = "Price vs. Size Fit",
10        x = "Size (sqft_living)",
11        y = "Price") +
12   theme_minimal()
```



Figure 1: Price vs. Size Fit

- Size as a core driver
- $b_0 = -43580.7$,
 $b_1 = 280.6$

Simple Linear Regression: Python

- **Practice:** Fits price to living space, revealing size's impact
- **Key Insight:** Positive slope shows larger homes fetch higher prices

```
1 # Python (p. 152 adapted)
2 from sklearn.linear_model import LinearRegression
3 predictors = ['SqFtTotLiving']
4 outcome = 'AdjSalePrice'
5 simple_lm = LinearRegression()
6 simple_lm.fit(house[predictors], house[outcome])
7 # Example: Intercept ~ base, Coef ~ $ per sq ft
```

Finding the Best Fit: Least Squares

- **How do we draw that line?** We minimize the mess—sum of squared errors
- **Theory:** Finds the line minimizing residual sum of squares:
$$\sum_{i=1}^n (Y_i - \hat{Y}_i)^2$$
- **How:** Adjusts b_0 and b_1 to reduce prediction errors—optimal for linear fits
- **History:** Legendre (1805) and Gauss; computationally efficient but outlier-sensitive in small datasets

Expanding the Scope

More Clues: Multiple Linear Regression

- **Theory:** Extends to multiple predictors:
$$Y = b_0 + b_1X_1 + b_2X_2 + \cdots + e$$
- **Power:** Captures combined effects—size, lot, bedrooms—assuming linearity
- **Use:** Explains complex housing dynamics

- Size adds \$229/sq ft

```
1 # R (p. 152)
2 house_lm <- lm(AdjSalePrice ~ SqFtTotLiving +
3   SqFtLot + Bathrooms +
4   Bedrooms + BldgGrade, data = house)
# Coefs: SqFtTotLiving = 228.831, Bedrooms =
-47769.955
```

Multiple Linear Regression: Key Findings

```
1  lm(formula = price ~ sqft_living + sqft_lot + bathrooms + bathrooms +  
2  grade, data = house_df, na.action = na.omit)  
3  
4  Coefficients:  
5  (Intercept)  sqft_living    sqft_lot    bathrooms    grade  
6  -5.957e+05    2.065e+02    -2.664e-01    -3.944e+04    1.037e+05
```

- **sqft_living:** +\$206.5 per sq ft → Larger houses increase price significantly.
- **grade:** +\$103,700 per unit → Higher quality homes boost price.
- **sqft_lot:** -\$0.266 per sq ft → Lot size has a tiny negative effect.
- **bathrooms:** -\$39,440 per extra bathroom → Unexpected negative impact.

Multiple Linear Regression: Model Evaluation

```
1 # R
2 summary(house_lm)# Model Summary (Extracts Coefficients, p-values, R²)
3 predictions <- predict(house_lm, newdata = house_df)# Extract RMSE
4 residuals <- house_df$price - predictions
5 RMSE <- sqrt(mean(residuals^2))
6 R_squared <- summary(house_lm)$r.squared # Extract R-squared
7 p_values <- summary(house_lm)$coefficients[, 4] # Extract P-values
8 # Print Results
9 cat("RMSE:", RMSE, "\n")
10 cat("R²:", R_squared, "\n")
11 cat("P-values:\n")
12 print(p_values)
```

```
1 > cat("RMSE:", RMSE, "\n") | RMSE: 249532.2
2 > cat("R²:", R_squared, "\n") | R²: 0.5380018
3 > cat("P-values:\n") > print(p_values)
4 (Intercept)  sqft_living  sqft_lot  bathrooms  grade
5 0.000000e+00 0.000000e+00 1.711092e-10 2.689854e-30 0.000000e+00
```

- **RMSE:** 261,300 Predictions are off by 261K on average.
- **R²:** 0.5406 (54.06
- **P-values:** SqFtLot is **not statistically significant (p = 0.323)'

Multiple Linear Regression: Python

- **Practice:** Models housing with multiple factors
- **Key Insight:** Negative bedroom coef suggests smaller rooms hurt value

- Bedrooms vs. size tension

```
1 # Define predictors and outcome
2 predictors = ['sqft_living', 'sqft_lot', 'bathrooms',
3              ', 'grade']
4 outcome = 'price'
5
6 # Fit Multiple Linear Regression Model
7 house_lm = LinearRegression()
8 house_lm.fit(house_df[predictors], house_df[outcome])
```

Encoding Categorical Variables in Regression Models

- **Categorical variables** must be converted into numerical values for regression.
- **Encoding methods:**
 - **Dummy (One-Hot) Encoding** – Creates binary columns for each category.
 - **Reference (Treatment) Coding** – Uses one category as a reference, keeping $P - 1$ columns.
 - **Deviation (Sum) Coding** – Compares each category to the overall mean.
 - **Ordered Factor Encoding** – Converts ordered categories into numeric values.
- **How it's used in regression:**
 - Each encoded category appears as a separate coefficient.
 - The model estimates how each category affects the outcome relative to the reference level.
 - For ordered factors, treating them as numeric assumes a linear relationship.

Multiple Linear Regression: Key Findings

```
1 # Ensure PropertyType has a valid reference level
2 house$PropertyType <- relevel(house$PropertyType, ref = "Multiplex")
3
4 # Fit the regression model with the refined dataset
5 house_lm <- lm(AdjSalePrice ~ SqFtTotLiving + SqFtLot + Bathrooms +
6   BldgGrade + PropertyType, data = house)
7
8 Coefficients:
9 Estimate Std. Error t value Pr(>|t|)
10 (Intercept)      173429.87    31512.77    5.503  0.00271 **
11 ...
12 BldgGrade          -1442.60     6734.35   -0.214  0.83884
13 PropertyTypeSingle Family    1456.54     8063.28    0.181  0.86374
14 PropertyTypeTownhouse     9677.07    11444.71    0.846  0.43639
```

- Sets "Multiplex" as the reference category, ensuring comparisons against it.
- "Single Family" and "Townhouse" to have separate coefficients in the regression output.

Factor Variables: Python

- **Practice:** Integrates property type into price model
- **Key Insight:** Townhouses may differ from single-family homes

- Baseline comparison

```
1 # Python (p. 166 adapted)
2 import pandas as pd
3 X = pd.get_dummies(house['PropertyType'], drop_
    first=True)
4 # Drops first level (e.g., Multiplex) as reference
```


Nonlinear Fit

- Nonlinear via polynomial segments joined at knots
- **Why:** Captures diminishing returns—e.g., small homes gain more per sq ft
- **Advantage:** Flexible fit.

```
1 # Load data (replace 'house_98105' with
  your dataset)
2 model_poly <- lm(price ~ poly(sqft_living,
  2) + sqft_lot +
3 grade + bathrooms + bedrooms, data=house_
  df)
4
5 # Generate partial residual plot
6 visreg(model_poly, "sqft_living", gg=TRUE)
  +
7 geom_point(color = "black", shape = 1) +
8 geom_smooth(method="loess", color="blue",
  size=1.2, linetype="dashed") +
9 theme_minimal() +
10 labs(title="Partial Residual Plot", x="
  SqFtTotLiving", y="Partial Residuals")
```

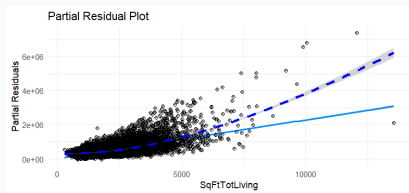
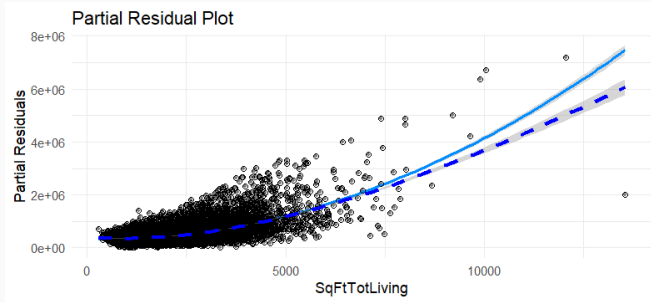


Figure 2: Without poly.

Comparison of Result

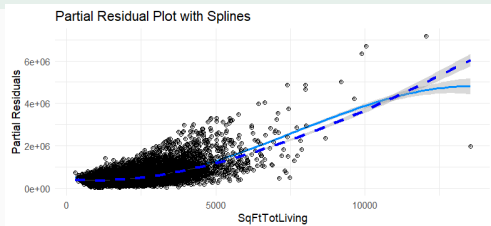


With Poly

Observation: With polynomial regression (2nd degree) aligns correctly with the trend, improving model accuracy.

Spline Regression: Partial Residual Plot

```
Generate partial residual plot for  
spline regression model  $spline <- lm(price$   
 $bs(sqft_{living}, degree = 3, df =$   
 $6) + sqft_{tot} + grade + bathrooms +$   
 $bedrooms, data = house_{df})$ 
```



Spline Fit Visualization

Explanation:

Cubic Splines: The function is made of piecewise cubic polynomials (degree = 3).

Degrees of Freedom (df = 6): Controls flexibility—higher df allows more variation

Splines fit the data smoothly without unnecessary fluctuations.

Nonlinear Fit: Splines in Python

- **Practice:** Fits nonlinear price trends in zip 98105
- **Key Insight:** Better matches small vs. large home value shifts

- Curves reflect reality

```
1 # Python (p. 190)
2 import statsmodels.formula.api as smf
3 formula = 'AdjSalePrice~\u03bcs(SqFtTotLiving,\u03bdfs=6,\u03bcs
           degree=3)\u03bcs+\u03bcsSqFtLot+\u03bcsBathrooms+\u03bcsBedrooms+\u03bcs
           BldgGrade'
4 model_spline = smf.ols(formula=formula, data=house_
                        98105).fit()
```

Refining Precision

Model Assessment: Theory

- **Theory (p. 153):** Measures prediction quality and fit
- **RMSE:** $\sqrt{\frac{\sum (y_i - \hat{y}_i)^2}{n}}$ —average error magnitude
- R^2 : Proportion of variance explained (0-1); higher means better fit
- **Use:** Guides housing prediction accuracy

Cross-Validation: Theory

- **Theory (p. 155):** Validates model on unseen data via k -fold splits
- **Process:** Divide data, train on $k - 1$, test on 1, repeat, average RMSE
- **Why:** Ensures predictions generalize beyond training sales—crucial for real estate

Picking the Best Story: Model Selection

- Too many clues clutter the tale. Stepwise selection trims variables, AIC balances fit and simplicity, and penalties shrink extras.
- Think of it as editing: keep the essentials, cut the fluff—Occam's razor guides us to a lean, powerful narrative.
- Our goal? A story that's clear and predicts well, not a sprawling epic.

Model Selection: Theory & R

- **Theory (p. 156):** Balances fit vs. complexity—Occam's razor
- **AIC:**
 $2P + n \log(\text{RSS}/n)$ —penalizes extra predictors
- **Goal:** Optimal housing model without overkill
- Streamlined predictors

```
1 # R (p. 157)
2 library(MASS)
3 house_full <- lm(AdjSalePrice ~ SqFtTotLiving +
4   SqFtLot + Bathrooms +
5   Bedrooms + BldgGrade + PropertyType, data = house)
6 step <- stepAIC(house_full, direction = "both")
# Drops less impactful vars
```

Multiple Linear Regression: Key Findings

```
1 house_full <- lm(price ~ sqft_living + sqft_lot15 + bathrooms +  
2 bedrooms + grade+yr_renovated , data = house_df)  
3 step <- stepAIC(house_full, direction = "both")  
4 summary(step)  
5  
6 Coefficients:  
7 Estimate Std. Error t value Pr(>|t|)  
8 (Intercept) -4.841e+05 1.477e+04 -32.777 <2e-16 ***  
9 sqft_living 2.300e+02 3.597e+00 63.956 <2e-16 ***  
10 sqft_lot15 -7.053e-01 6.252e-02 -11.282 <2e-16 ***  
11 bathrooms -3.096e+04 3.442e+03 -8.994 <2e-16 ***  
12 bedrooms -4.026e+04 2.271e+03 -17.726 <2e-16 ***  
13 grade 9.776e+04 2.290e+03 42.693 <2e-16 ***  
14 yr_renovated 8.746e+01 4.160e+00 21.022 <2e-16 ***
```

- sqft lot15 (-5.135e-01) Small , impact on price can be removal.
- grade (1.315e+05): Highest positive impact, should be kept.

Model Selection: Python

- **Practice:** Automates predictor choice for housing
- **Key Insight:** Reduces noise, enhances prediction

```
1 # Python (p. 158 adapted)
2 from dmbs import stepwise_selection
3 predictors = ['SqFtTotLiving', 'SqFtTot', '
4             Bathrooms', 'Bedrooms', 'BldgGrade']
5 def train_model(vars):
6     model = LinearRegression()
7     model.fit(house[vars], house[outcome])
8     return model
best_model, _ = stepwise_selection(house[predictors
    ].columns, train_model)
```

- Focused fit

Weighted Regression: Theory & R

- **Theory (p. 159):** Weights adjust influence by reliability
- **Why:** Older sales less relevant—recent data gets priority
- **Impact:** Refines coefficients for current market
- Recent sales emphasized

```
1 # R (p. 159)
2 house$Weight = year(house$DocumentDate) - 2005
3 house_wt <- lm(AdjSalePrice ~ SqFtTotLiving +
4   SqFtLot + Bathrooms +
5   Bedrooms + BldgGrade, data = house, weight = Weight
6   )
7 # Shifts coeffs slightly
```

Weighted Regression: Python

- **Practice:** Weights tune housing model
- **Key Insight:** Aligns predictions with market trends

- Fresher focus

```
1 # Python (p. 160)
2 house['Weight'] = [int(date.split('-')[0]) for date
3                     in house.DocumentDate] - 2005
4 house_wt = LinearRegression()
5 house_wt.fit(house[predictors], house[outcome],
6              sample_weight=house.Weight)
```

Facing Challenges

Prediction Limits: Theory

- **Theory (p. 161):** Extrapolation beyond data fails—e.g., empty lots
- **Intervals:** Confidence for b_i , wider prediction for \hat{Y}_i
- **Why:** Uncertainty spikes outside training range—limits housing forecasts

Interpreting Coefficients: Theory

- **Theory (p. 171):** Coefficients mislead if predictors correlate
- **Multicollinearity:** Size and bedrooms overlap—unstable fits
- **Confounding:** Missing location skews results
- **Interactions:** Size's effect varies by zip—needs modeling

Diagnostics: Theory & R

- **Theory (p. 176):** Residuals reveal model flaws
- **Outliers:** Extreme sales (e.g., \$119,748); **Influence:** Sway points
- **Heteroskedasticity:** Uneven errors signal gaps

```
1 # R (p. 177)
2 house_98105 <- house[house$ZipCode == 98105, ]
3 lm_98105 <- lm(AdjSalePrice ~ SqFtTotLiving +
4   SqFtLot + Bathrooms +
5   Bedrooms + BldgGrade, data = house_98105)
6 sresid <- rstandard(lm_98105) # -4.326732 outlier
```

figure4-6-placeholder.pd

Figure 3: Influence Plot

Diagnostics: Python

- **Practice:** Identifies \$119,748 as partial sale anomaly
- **Key Insight:** Diagnostics ensure robust housing predictions

```
1 # Python (p. 178)
2 from statsmodels.stats.outliers_influence import
   OLSInfluence
3 house_98105 = house[house['ZipCode'] == 98105]
4 model = smf.ols('AdjSalePrice~_SqFtTotLiving+_
   SqFtLot+_Bathrooms+_Bedrooms+_BldgGrade',
   data=house_98105).fit()
5 influence = OLSInfluence(model)
6 sresiduals = influence.resid_studentized
```

- Spots critical flaws

Influence Plot (Bubble Plot) – Identify Influential Values

- **Purpose:** Identifies influential observations by combining leverage, residuals, and Cook's Distance.
- **Key Insights:**
 - Large bubbles = high Cook's Distance → Removing these changes regression results significantly.
 - Possible reasons:
 1. High leverage (extreme predictor values) , Large residual (far from regression line) ,Both high leverage and large residual.
- **Results:** Four large influential points found (Cook's $D > 0.08$), impacting coefficients.
- Residuals beyond ± 2.5

Influence Plot (Bubble Plot) – Identify Influential Values

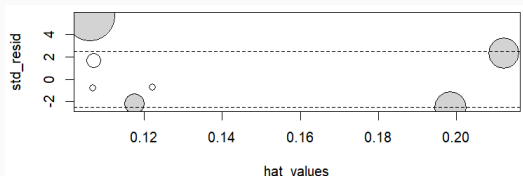


Figure 4: Bubble plot showing influential points.

Influence Plot (Bubble Plot) – Identify Influential Values

Listing 1: Influence Plot in R

```
1 library(car)
2 lm_model <- lm(AdjSalePrice ~ SqFtTotLiving + SqFtLot + Bathrooms + Bedrooms +
  BldgGrade, data=house_98105)
3 influencePlot(lm_model)
```

Listing 2: Influence Plot in Python

```
1 house_98105 = house[house['ZipCode'] == 98105]
2 X = house_98105[['SqFtTotLiving', 'SqFtLot', 'Bathrooms', 'Bedrooms', 'BldgGrade']].
  assign(const=1)
3 y = house_98105['AdjSalePrice']
4 model = sm.OLS(y, X).fit()
5 influence = sm.stats.outliers_influence.OLSInfluence(model)
6 fig, ax = plt.subplots(figsize=(5, 5))
7 ax.axhline(-2.5, ls='--', color='C1')
8 ax.axhline(2.5, ls='--', color='C1')
9 ax.scatter(influence.hat_matrix_diag, influence.resid_studentized_internal,
10 s=1000 * np.sqrt(influence.cooks_distance[0]), alpha=0.5)
11 ax.set_xlabel('hat_values')
12 ax.set_ylabel('studentized_residuals')
13 plt.show()
```

Residual Plot (Heteroskedasticity Check)

- **Purpose:** The residual plot checks for heteroskedasticity by analyzing how residuals (errors) vary with predicted values.
- **Key Insights:**
 - X-axis (Predicted Values): Represents the fitted values from the regression model.
 - Y-axis (Absolute Residuals): Measures the deviation of actual values from predictions.
 - Scatter Points: Each dot represents an observation's residual.
 - LOESS Smoother (Blue Line): Shows the trend in residuals.
 - Shaded Region: Indicates confidence around the trend.
 1. Heteroskedasticity detected – Residuals increase with larger predicted values, indicating variance instability.
 2. Curved Trend – Suggests missing variables or non-linearity in the data.
 3. Outliers at High Predictions – Some extreme points have large residuals, further confirming instability.

Residual Plot (Heteroskedasticity Check)

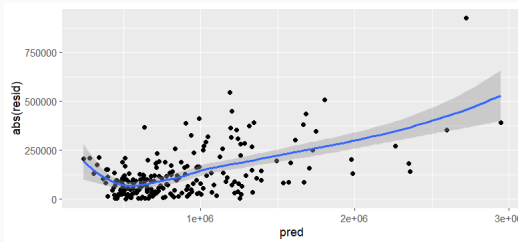


Figure 5: Bubble plot showing influential points.

Influence Plot (Bubble Plot) – Identify Influential Values

Listing 3: Heteroskedasticity Plot in R

```
1 df <- data.frame(resid = residuals(lm_98105), pred = predict(lm_98105))
2 ggplot(df, aes(pred, abs(resid))) + geom_point() + geom_smooth()
```

Listing 4: Heteroskedasticity Plot in Python

```
1 import seaborn as sns
2 fig, ax = plt.subplots(figsize=(5, 5))
3 sns.regplot(model.fittedvalues, np.abs(model.resid), scatter_kws={'alpha': 0.25},
4 line_kws={'color': 'C1'}, lowess=True, ax=ax)
5 ax.set_xlabel('predicted')
6 ax.set_ylabel('abs(residual)')
7 plt.show()
```


Histogram of Standardized Residuals: Normality Check

- **Purpose:** The histogram assesses residual normality by analyzing the distribution of standardized residuals.
- **Key Insights:**
 - **Centering Around Zero:** The residuals are centered around 0, indicating no strong systematic bias in the model.
 - **Skewness Long Tails:** The right tail is longer, suggesting right-skewness and possible underestimation of some values.
 - **Non-Normal Distribution:** The residuals deviate from a perfect bell-shaped curve, hinting at potential issues:
 1. Missing predictors affecting the model.
 2. Heteroskedasticity, as observed in the residual plot.
 3. Outliers influencing the regression fit.

Histogram of Standardized Residuals: Normality Check

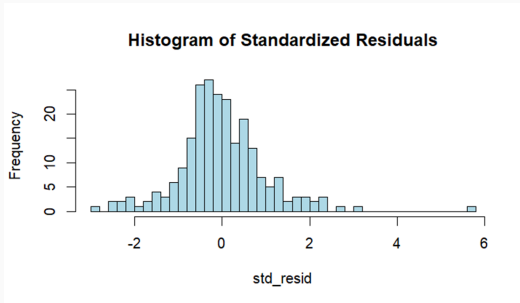


Figure 6: Bubble plot showing influential points.

Histogram of Standardized Residuals: Normality Check

Listing 5: Histogram of Standardized Residuals in R

```
1 df <- data.frame(resid = residuals(lm_98105), pred = predict(lm_98105))
2 ggplot(df, aes(pred, abs(resid))) + geom_point() + geom_smooth()
```

Listing 6: Histogram of Standardized Residuals in Python

```
1 plt.hist(influence.resid_studentized_internal, bins=50, color='lightblue')
2 plt.xlabel('Standardized Residuals')
3 plt.title('Histogram of Standardized Residuals')
4 plt.show()
```

Insights & Next Steps

- **Findings:** Linear ties price to size, splines capture nonlinear trends
- **Diagnostics:** Reveal quirks like partial sales—critical for accuracy
- **Application:** Real-world tool for buyers, sellers, and assessors

Key Takeaways

- **Flexible:** Evolves from simple lines to complex curves for housing
- **Precise:** RMSE and cross-validation ensure reliable price predictions
- **Powerful:** R and Python implementations unlock data-driven insights

Looking Ahead

- **Resources:** *Statistical Learning* (Hastie et al.), *Time Series Forecasting* (Shmueli)
- **Next Steps:** Dive into splines, time series for dynamic housing models
- **Call:** Blend theory and practice for smarter predictions