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: $Y = b_0 + b_1 X + e$
- 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

```r
# R
simple_lm <- lm(AdjSalePrice ~ SqFtTotLiving, data
    = house_df)
# Create the plot
ggplot(house_df, aes(x = sqft_living, y = price)) +
geom_point(alpha = 0.5) +  # Scatter plot
geom_smooth(method = "lm", color = "blue", se =
    FALSE) +  # Regression line
labs(title = "Price vs. Size Fit",
x = "Size (sqft_living)",
y = "Price") +
theme_minimal()
```

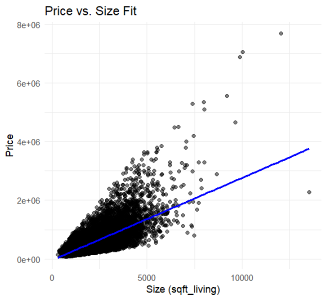

Figure 1: Price vs. Size Fit

- Size as a core driver
- b0=−43580.7 , b1=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

```python
# Python (p. 152 adapted)
from sklearn.linear_model import LinearRegression
predictors = ['SqFtTotLiving']
outcome = 'AdjSalePrice'
simple_lm = LinearRegression()
simple_lm.fit(house[predictors], house[outcome])
# 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

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

- Size adds $229/sq ft

Multiple Linear Regression: Key Findings

```
lm(formula = price ~ sqft_living + sqft_lot + bathrooms + bathrooms +
grade, data = house_df, na.action = na.omit)

Coefficients:
(Intercept)  sqft_living     sqft_lot    bathrooms        grade
-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

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

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

- **RMSE**: $261,300 *Predictionsareoffby* 261K on average.
- R^2: 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

```
# Define predictors and outcome
predictors = ['sqft_living', 'sqft_lot', 'bathrooms
    ', 'grade']
outcome = 'price'

# Fit Multiple Linear Regression Model
house_lm = LinearRegression()
house_lm.fit(house_df[predictors], house_df[outcome
    ])
```

- Bedrooms vs. size tension

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

```
# Ensure PropertyType has a valid reference level
house$PropertyType <- relevel(house$PropertyType, ref = "Multiplex")

# Fit the regression model with the refined dataset
house_lm <- lm(AdjSalePrice ~ SqFtTotLiving + SqFtLot + Bathrooms +
      BldgGrade + PropertyType, data = house)
Coefficients:
Estimate Std. Error t value Pr(>|t|)
(Intercept)                 173429.87   31512.77   5.503  0.00271 **
...
BldgGrade                    -1442.60    6734.35  -0.214  0.83884
PropertyTypeSingle Family     1456.54    8063.28   0.181  0.86374
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

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

- Baseline comparison

Nonlinear Fit: Theory & Splines in R

- **Theory (p. 187)**: Nonlinear via polynomial segments joined at knots
- **Why**: Captures diminishing returns—e.g., small homes gain more per sq ft
- **Advantage**: Flexible fit without overfitting like high-order polynomials

```r
# R (p. 190)
library(splines)
knots <- quantile(house_98105$SqFtTotLiving, p = c
      (.25, .5, .75))
lm_spline <- lm(AdjSalePrice ~ bs(SqFtTotLiving,
      knots = knots, degree = 3) +
SqFtLot + Bathrooms + Bedrooms + BldgGrade, data =
      house_98105)
```

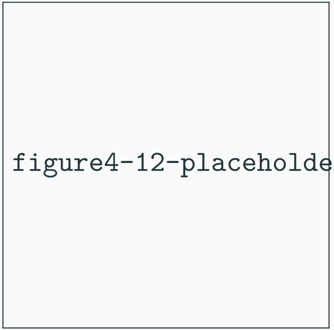

Figure 2: Spline Fit

Nonlinear Fit: Splines in Python

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

```
# Python (p. 190)
import statsmodels.formula.api as smf
formula = 'AdjSalePrice ~ bs(SqFtTotLiving, df=6, degree=3) + SqFtLot + Bathrooms + Bedrooms + BldgGrade'
model_spline = smf.ols(formula=formula, data=house_98105).fit()
```

- Curves reflect reality

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

Model Selection: Theory & R

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

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

- Streamlined predictors

Model Selection: Python

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

```
# Python (p. 158 adapted)
from dmba import stepwise_selection
predictors = ['SqFtTotLiving', 'SqFtLot', 'Bathrooms', 'Bedrooms', 'BldgGrade']
def train_model(vars):
model = LinearRegression()
model.fit(house[vars], house[outcome])
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

```
# R (p. 159)
house$Weight = year(house$DocumentDate) - 2005
house_wt <- lm(AdjSalePrice ~ SqFtTotLiving + SqFtLot + Bathrooms +
Bedrooms + BldgGrade, data = house, weight = Weight)
# Shifts coefs slightly
```

- Recent sales emphasized

Weighted Regression: Python

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

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

- Fresher focus

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

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

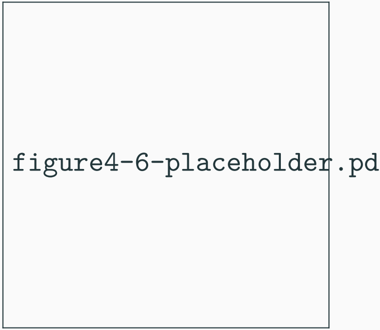

Figure 3: Influence Plot

Diagnostics: Python

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

```
# Python (p. 178)
from statsmodels.stats.outliers_influence import
    OLSInfluence
house_98105 = house[house['ZipCode'] == 98105]
model = smf.ols('AdjSalePrice ~ SqFtTotLiving +
    SqFtLot + Bathrooms + Bedrooms + BldgGrade',
    data=house_98105).fit()
influence = OLSInfluence(model)
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

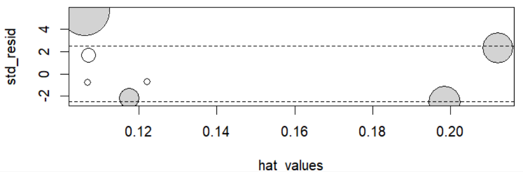

Figure 4: Bubble plot showing influential points.

Influence Plot (Bubble Plot) – Identify Influential Values

Listing 1: Influence Plot in R

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

Listing 2: Influence Plot in Python

```
house_98105 = house[house['ZipCode'] == 98105]
X = house_98105[['SqFtTotLiving', 'SqFtLot', 'Bathrooms', 'Bedrooms', 'BldgGrade']].assign(const=1)
y = house_98105['AdjSalePrice']
model = sm.OLS(y, X).fit()
influence = sm.stats.outliers_influence.OLSInfluence(model)
fig, ax = plt.subplots(figsize=(5, 5))
ax.axhline(-2.5, ls='--', color='C1')
ax.axhline(2.5, ls='--', color='C1')
ax.scatter(influence.hat_matrix_diag, influence.resid_studentized_internal,
s=1000 * np.sqrt(influence.cooks_distance[0]), alpha=0.5)
ax.set_xlabel('hat␣values')
ax.set_ylabel('studentized␣residuals')
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)

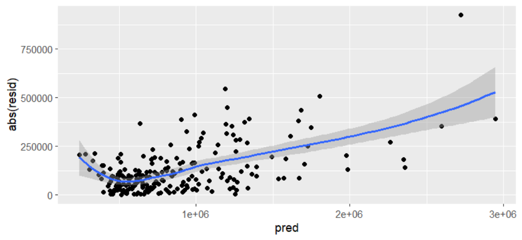

Figure 5: Bubble plot showing influential points.

Influence Plot (Bubble Plot) – Identify Influential Values

Listing 3: Heteroskedasticity Plot in R

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

Listing 4: Heteroskedasticity Plot in Python

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

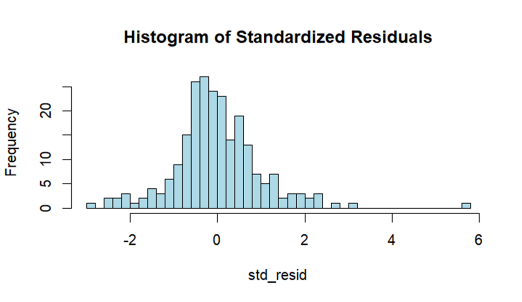

Figure 6: Bubble plot showing influential points.

Histogram of Standardized Residuals: Normality Check

Listing 5: Histogram of Standardized Residuals in R

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

Listing 6: Histogram of Standardized Residuals in Python

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

Insights & Next Steps

Housing Insights

- **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