King County Real Estate Data Analysis

Alan Lin

August 28 2025

Motivation

Housing prices are a central concern in modern society, with prospective buyers seeking to balance affordability and desirable home features. This project analyzes King County housing data using linear regression, bootstrap confidence intervals, and Jackknife+ conformal prediction to model housing prices and assess predictive accuracy based on key features.

Data Setup

The King County housing dataset is used, containing home sale prices and associated property features. For this analysis, the raw dataset is imported **without variable transformations** to preserve interpretability of the original features. The `ggplot2` library is also loaded for creating visualizations later in the project.

```r
# Import ggplot2 for data visualizations
library(ggplot2)

# Load training and test datasets
train_data = read.table("data/king_county_train.dat", header = TRUE)
test_data = read.table("data/king_county_test_1.dat", header = TRUE)

head(train_data, 5)
```

```
##      price bedrooms bathrooms sqft_living  sqft_lot floors waterfront view
## 1 689999.9        3      1.75    1599.686  4399.749      1          0    0
## 2 941499.3        5      3.50    3489.082  9679.206      2          0    4
## 3 583498.8        4      1.75    2860.841 48347.974      1          0    0
## 4 534949.8        5      1.50    2240.365  6337.596      1          0    0
## 5 288000.1        3      1.75    1660.068 10440.545      1          0    0
##   condition grade sqft_above sqft_basement yr_built yr_renovated zipcode
## 1         3     7   1031.022      569.5599     1941            0   98112
## 2         3     9   2457.885     1029.7130     1980            0   98177
## 3         3     8   1710.299     1150.7620     1978            0   98077
## 4         3     8   1280.911      961.5429     1956            0   98125
## 5         3     7   1038.876      620.1758     1978            0   98045
##   sqft_living15 sqft_lot15 log10price
## 1      2150.323   4002.387   5.838849
## 2      3079.059  13488.615   5.973820
## 3      2460.783  43558.850   5.766040
## 4      1869.462   6377.969   5.728313
## 5      1238.731  10377.387   5.459393
```

The dataset contains housing sale prices and features such as number of bedrooms, square footage (of living spaces and lots), year built, location (zipcode), and property attributes (condition, grade, waterfront, view).

Linear Regression.

Variable Selection.

The original dataset contained 18 variables (17 excluding the response variable `log10price`). The `price` feature was dropped, as it directly overlaps with the response. This left 16 variables, from which eight were selected to simplify the model and improve interpretability.

The first set of features relates to the core living space of a house, which naturally influences buyer interest and price:

- `bedrooms`
- `bathrooms`
- `floors`
- `sqft_living` (interior living space)
- `sqft_lot` (total lot size, including yard, driveway, etc.)

More bedrooms, bathrooms, floors, or larger living and lot space generally correlate with higher prices, making these strong predictors.

The `grade` feature was included because it represents construction quality. Higher grades indicate better materials and workmanship, which typically increase the building cost and market value.

Finally, two neighborhood-related features were considered:

- `sqft_living15` (average living space of the 15 nearest houses)
- `sqft_lot15` (average lot size of the 15 nearest houses)

These provide context about surrounding properties and local standards, which can significantly impact the value of an individual house.

The resulting regression model is:

$$\begin{aligned}\log_{10}(\text{price}) = \beta_0 &+ \beta_1 \cdot \text{bedrooms} + \beta_2 \cdot \text{bathrooms} + \beta_3 \cdot \text{sqft\_living} + \beta_4 \cdot \text{sqft\_lot} \\ &+ \beta_5 \cdot \text{floors} + \beta_6 \cdot \text{grade} + \beta_7 \cdot \text{sqft\_living15} + \beta_8 \cdot \text{sqft\_lot15}\end{aligned}$$

Test Data Predictions

The provided test dataset did not include house prices. To demonstrate the model's application to unseen data, predictions for the test set were generated and written to `regression_test_predictions.dat`. This file contains the predicted `log10price` for each house in the test dataset, allowing for future evaluation once true prices become available.

```r
# Make vector with chosen subset of features
best_vars = c("bedrooms", "bathrooms", "sqft_living", "sqft_lot", "floors",
              "grade", "sqft_living15", "sqft_lot15")
formula_str = paste("log10price ~", paste(best_vars, collapse = " + "))
model_formula = as.formula(formula_str)

# Fit OLS model with chosen subset of features
```

```r
ols_model = lm(model_formula, data = train_data)

# Select subset of test data with chosen features
test_X_subset = test_data[, best_vars]

# Predict housing prices of test data
predictions = predict(ols_model, newdata = test_X_subset)

# Write predictions to data file
write.table(predictions, file = "results/regression_test_predictions.dat",
            row.names = FALSE, col.names = FALSE)

# Print coefficients of OLS model
coef(ols_model)
```

```
##   (Intercept)      bedrooms     bathrooms   sqft_living      sqft_lot
##  4.834969e+00 -1.828300e-03  1.282518e-02  7.206469e-05  1.295956e-06
##        floors         grade sqft_living15    sqft_lot15
## -5.071991e-02  9.004629e-02  3.343642e-05 -2.126941e-06
```

Interpretation of OLS Coefficients

The following interprets the coefficients from the linear regression model fitted on the selected features.

If all the selected features were zero in value, the model would predict a $\log_{10}(\text{price})$ of approximately 4.834969. However, since this scenario is unrealistic in practice, the intercept should be interpreted with caution.

- Every additional **bedroom decreases** the $\log_{10}(\text{price})$ by approximately 1.83×10^{-3}. The negative coefficient for bedrooms may reflect correlations with other features such as square footage or number of bathrooms.
- An additional **bathroom increases** the $\log_{10}(\text{price})$ by approximately 1.28×10^{-2}.
- For every additional square foot of **living space**, the $\log_{10}(\text{price})$ **increases** by about 7.21×10^{-5}.
- Every additional square foot of **lot size increases** the $\log_{10}(\text{price})$ by about 1.30×10^{-6}.
- An additional **floor decreases** the $\log_{10}(\text{price})$ by approximately 5.07×10^{-2}.
- For each additional point in the **grade** rating, the $\log_{10}(\text{price})$ **increases** by about 9.00×10^{-2}.
- Every additional square foot of average **neighbor living space** (`sqft_living15`) **increases** the $\log_{10}(\text{price})$ by roughly 3.34×10^{-5}.
- For every additional square foot of average **neighbor lot size** (`sqft_lot15`), the $\log_{10}(\text{price})$ **decreases** by approximately 2.13×10^{-6}.

These interpretations represent the **expected change in** $\log_{10}(\text{price})$ per unit increase in each variable, holding all other variables constant.

If all features were zero, the model predicts a $\log_{10}(\text{price})$ of ~4.835. Since this is unrealistic, the intercept should be interpreted cautiously.

Note: Coefficients are rounded for interpretability. Full precision is available above in the output.

Computing 95% Confidence Intervals

Next, 95% confidence intervals for the regression coefficients were computed using the standard (Gaussian) method. These intervals provide a range of plausible values for each coefficient, reflecting the uncertainty in the estimates. The results were then printed for review.

```r
# Generate Gaussian confidence intervals
gaussian_cis = confint(ols_model, level = 0.95)

# Print results
t(gaussian_cis)
```

```
##        (Intercept)    bedrooms   bathrooms  sqft_living      sqft_lot
## 2.5 %     4.714840 -0.02077076 -0.01760346 3.857516e-05 -1.610614e-07
## 97.5 %    4.955097  0.01711416  0.04325382 1.055542e-04  2.752973e-06
##              floors      grade sqft_living15    sqft_lot15
## 2.5 %  -0.08115960 0.07047082  1.116960e-08 -3.439612e-06
## 97.5 % -0.02028021 0.10962176  6.686168e-05 -8.142704e-07
```

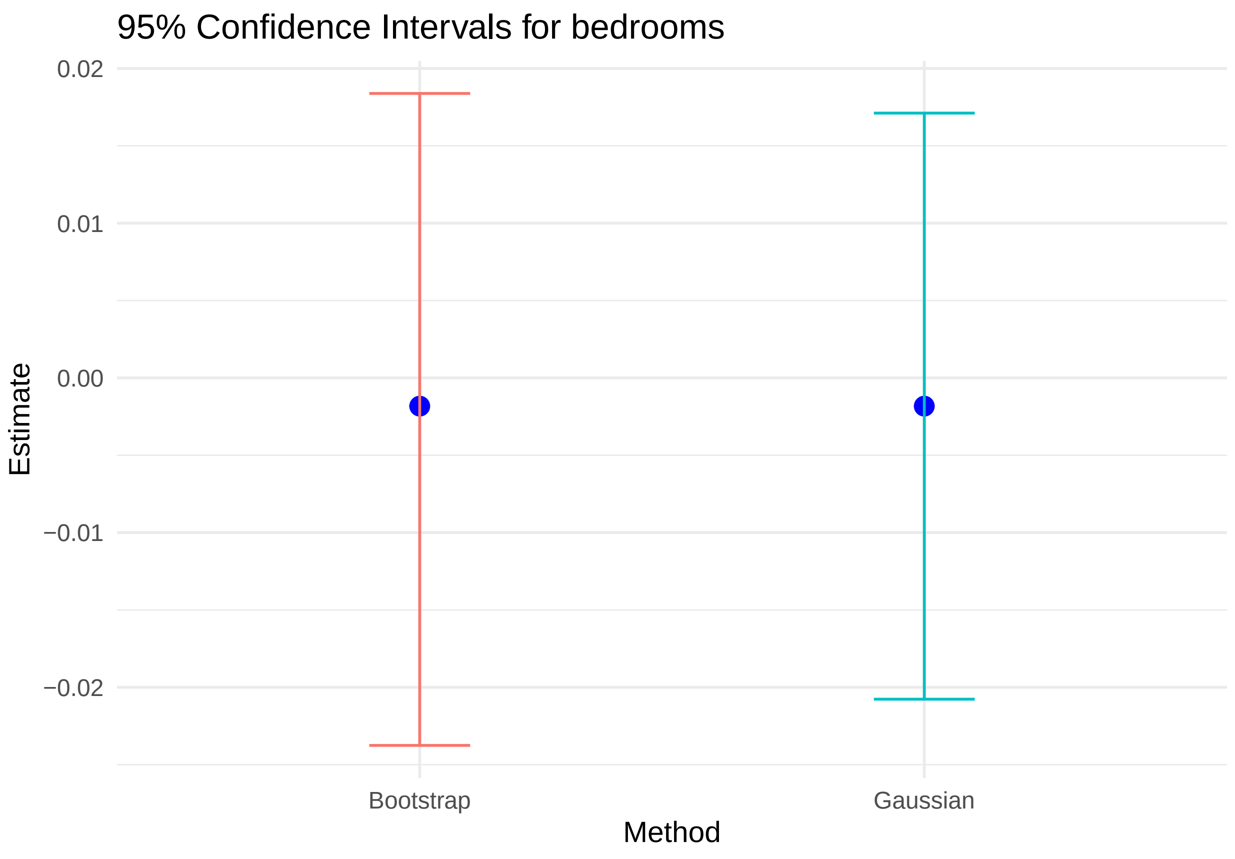
In addition to the standard method, 95% confidence intervals were computed using the bootstrap approach. This method resamples the data with replacement to better capture variability in the coefficient estimates. The resulting intervals were printed for review.

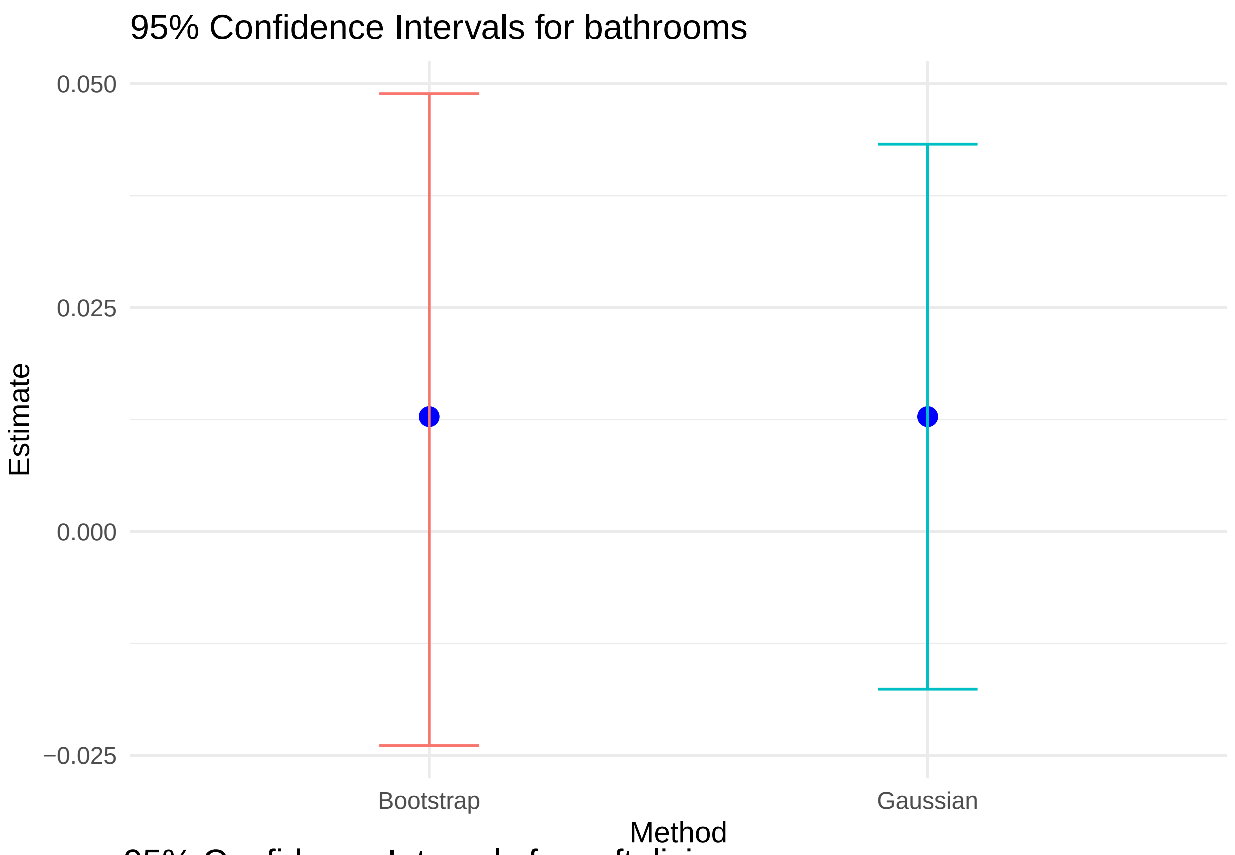
```r
set.seed(125)  # for consistent reproducibility
B = 1000       # number of bootstrap samples
coef_mat = matrix(NA, nrow = B, ncol = length(coef(ols_model)))
colnames(coef_mat) = names(coef(ols_model))

# generate 1000 bootstrapped samples
for (b in 1:B) {
  boot_indices = sample(1:nrow(train_data), replace = TRUE)
  boot_sample = train_data[boot_indices, ]
  boot_model = lm(model_formula, data = boot_sample)
  coef_mat[b, ] = coef(boot_model)
}

# Create 95% confidence intervals with bootstrapped samples
boot_cis = apply(coef_mat, 2, quantile, probs = c(0.025, 0.975))

# Print resulting confidence intervals
boot_cis
```

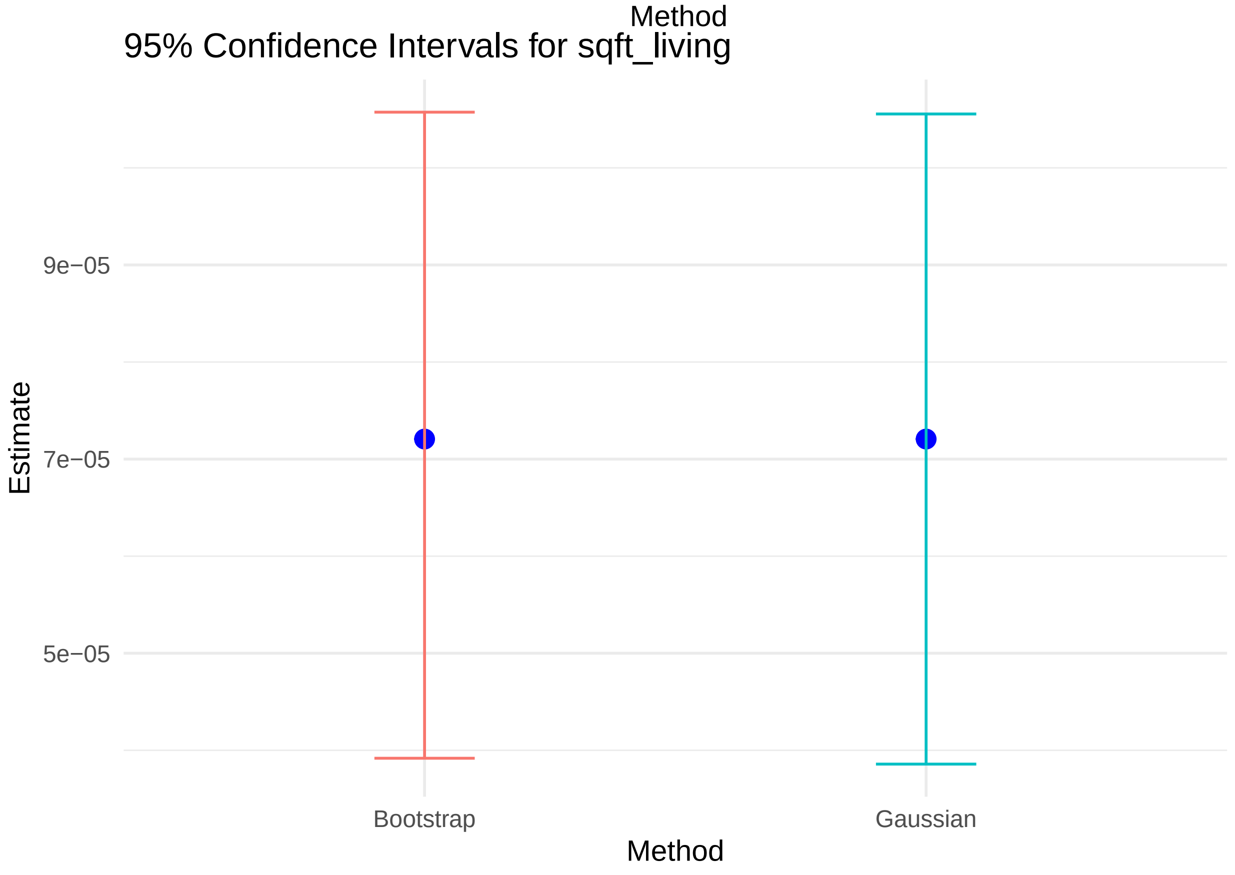
```
##       (Intercept)    bedrooms   bathrooms  sqft_living      sqft_lot
## 2.5%     4.710813 -0.02375865 -0.02392506 3.918174e-05 -4.571425e-07
## 97.5%    4.952198  0.01838465  0.04888134 1.057409e-04  2.313737e-06
##             floors      grade sqft_living15    sqft_lot15
## 2.5%  -0.08129104 0.07112538 -2.719308e-06 -3.095438e-06
## 97.5% -0.02399404 0.10986494  7.012835e-05 -4.978172e-07
```

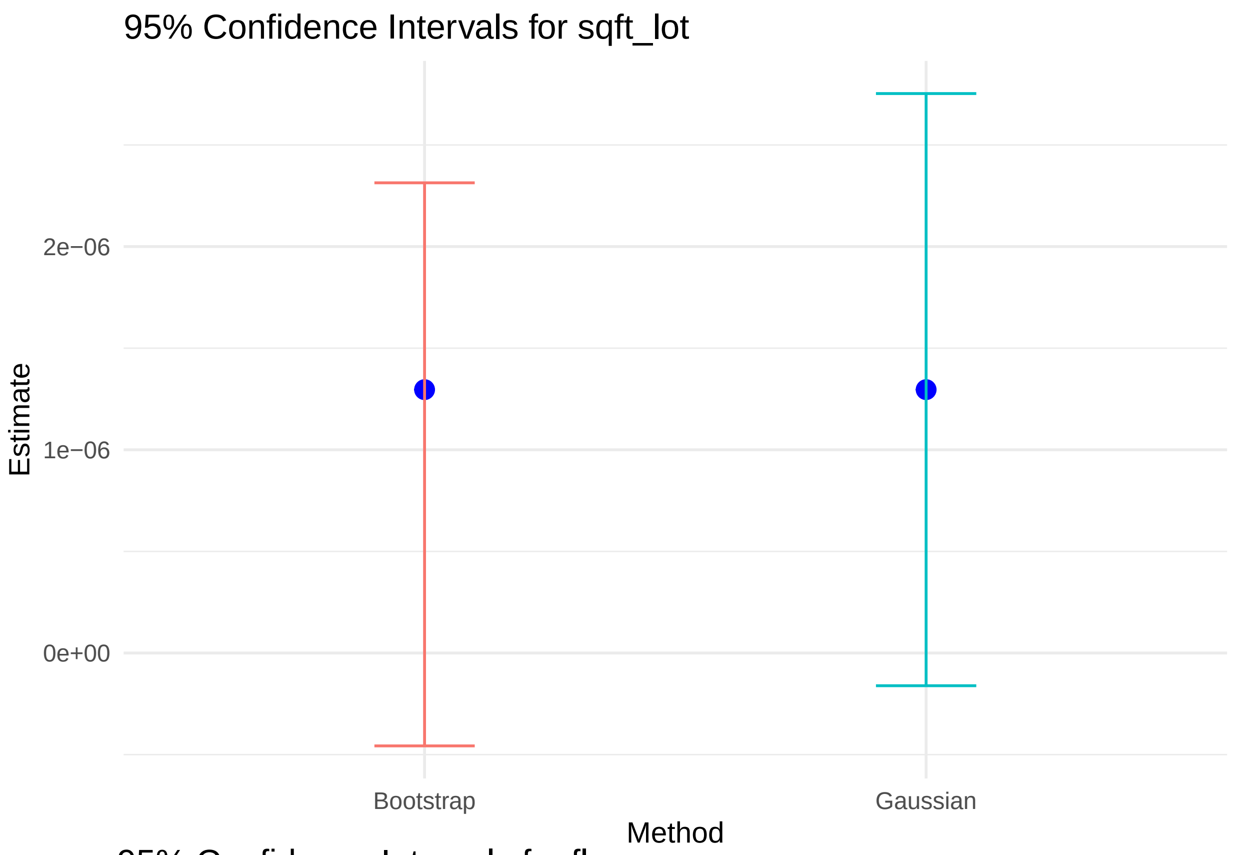
The 95% confidence intervals for each coefficient (excluding the intercept) were plotted using both the standard Gaussian method and the bootstrap approach. These plots were then compared alongside the original OLS coefficient estimates to visualize differences in interval width and assess the stability of each estimate.

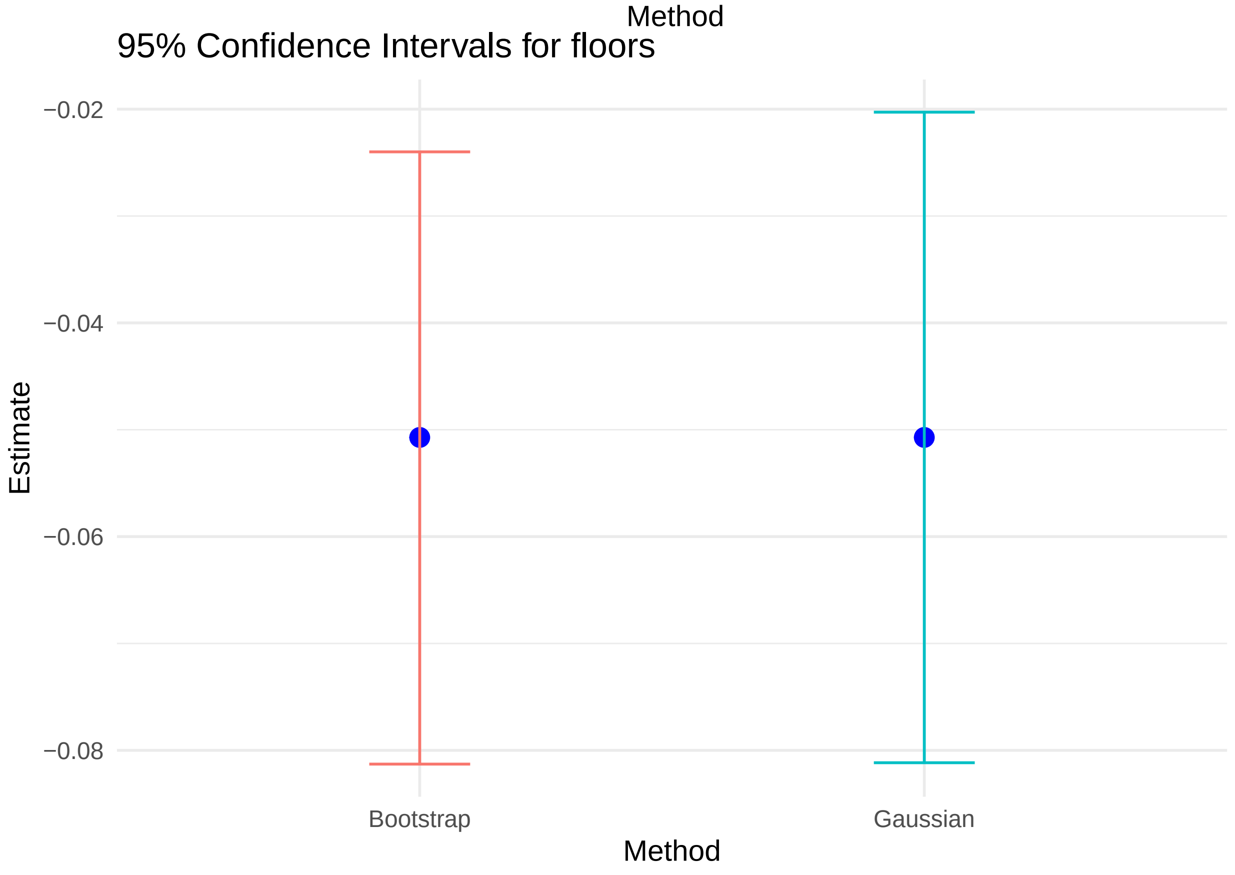
```r
# Generate confidence interval charts for each feature selected
for (i in best_vars) {
  coef_name = i
  ols_estimate <- coef(ols_model)[coef_name]
  gauss_lower <- gaussian_cis[coef_name, 1]
  gauss_upper <- gaussian_cis[coef_name, 2]
  boot_lower <- boot_cis[1, coef_name]
  boot_upper <- boot_cis[2, coef_name]

  plot_df <- data.frame(
    Method = c("Gaussian", "Bootstrap"),
    Lower = as.numeric(c(gauss_lower, boot_lower)),
    Upper = as.numeric(c(gauss_upper, boot_upper)),
    Estimate = as.numeric(c(ols_estimate, ols_estimate))
  )

  p = ggplot(plot_df, aes(x = Method, y = Estimate)) +
    geom_point(size = 3, color = "blue") +
    geom_errorbar(aes(ymin = Lower, ymax = Upper, color = Method), width = 0.2) +
    labs(title = paste("95% Confidence Intervals for", i),
         y = "Estimate",
         x = "Method"
         ) +
    theme_minimal() + theme(legend.position = "none")

  print(p)
}
```

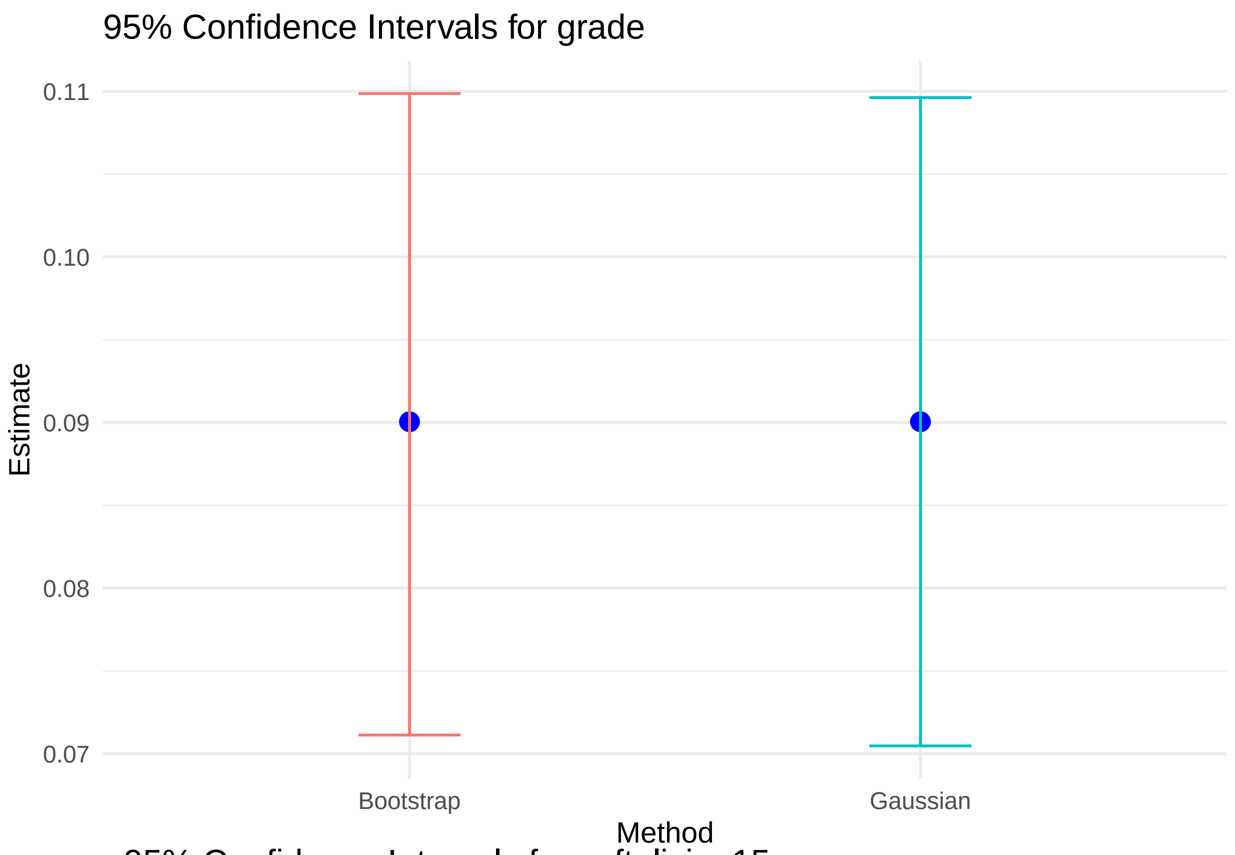


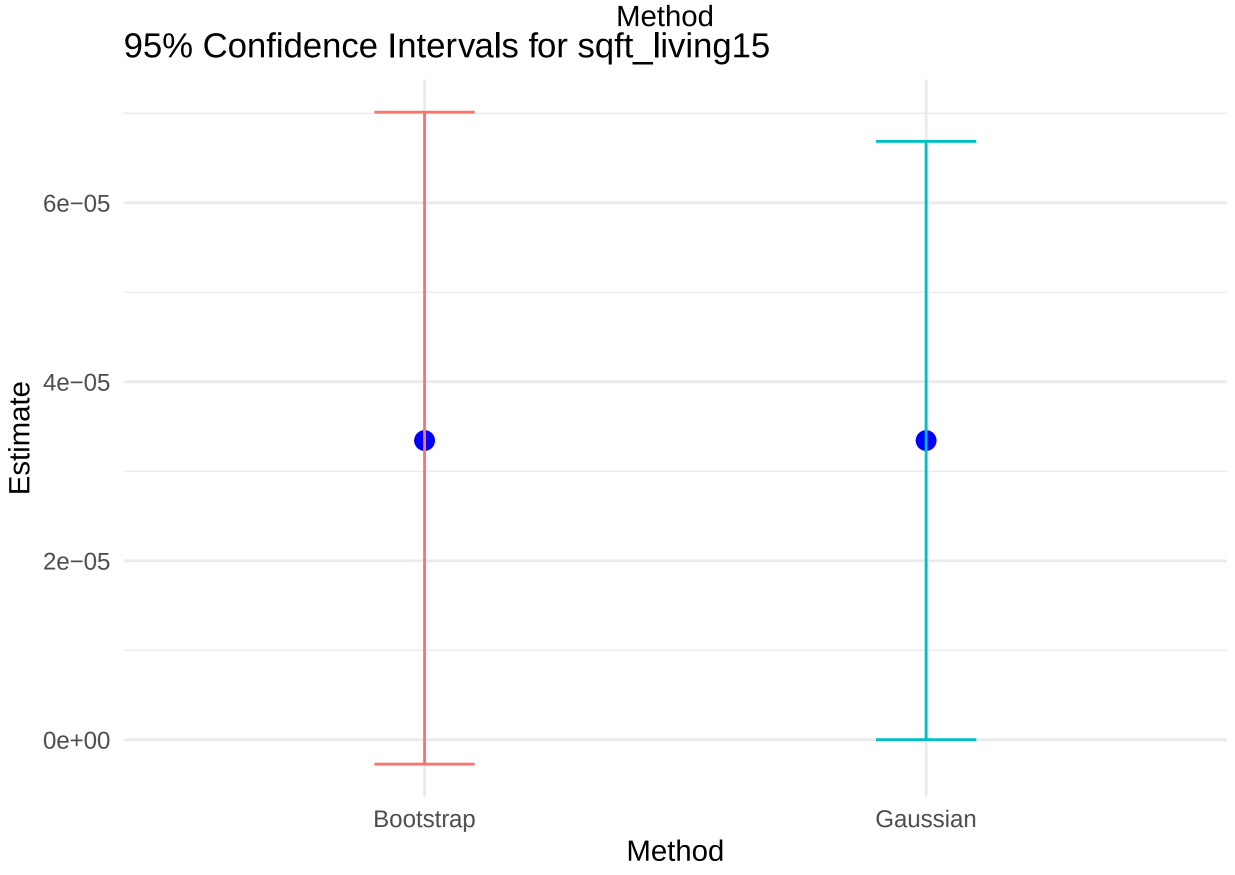

95% Confidence Intervals for bathrooms
0.050
0.025
0.000
−0.025
Estimate
Bootstrap
Gaussian
Method

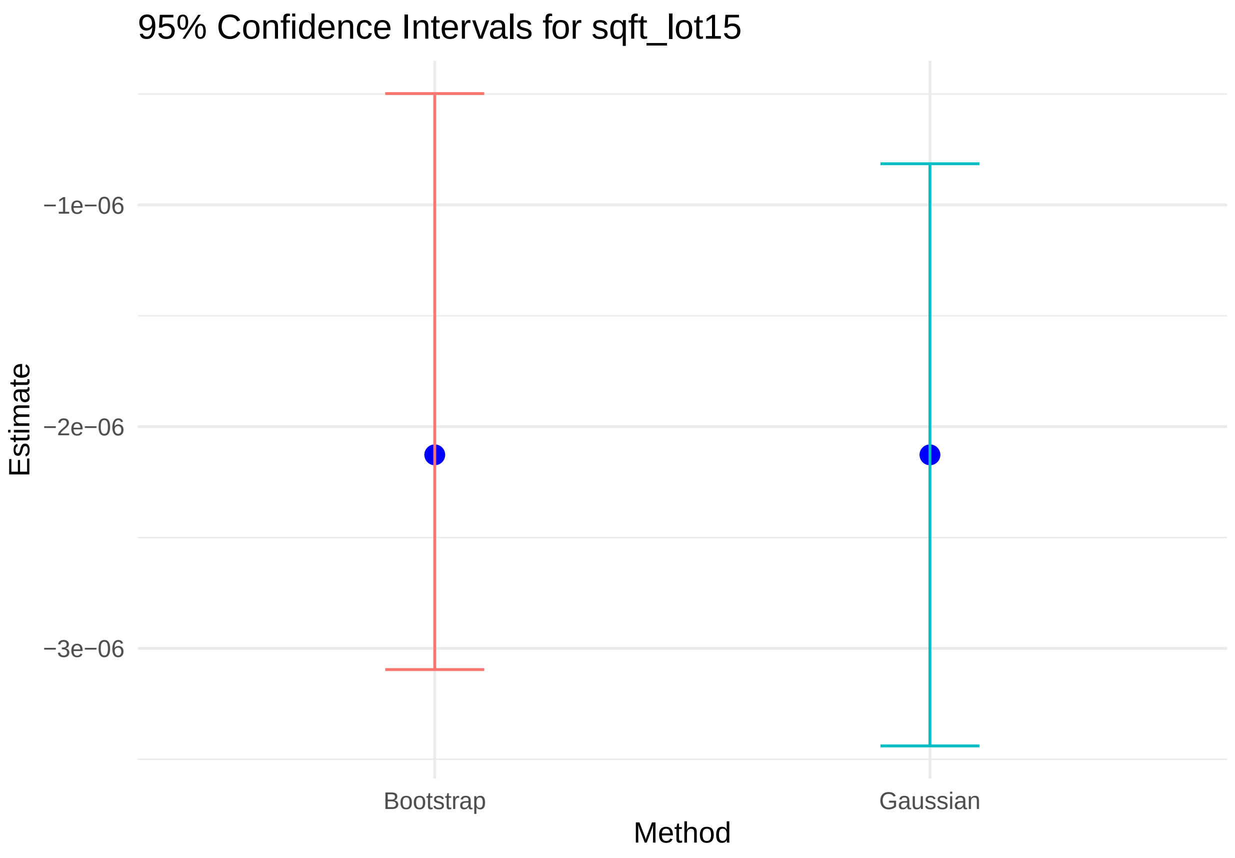

95% Confidence Intervals for sqft_living
9e−05
7e−05
5e−05
Estimate
Bootstrap
Gaussian
Method


95% Confidence Intervals for sqft_lot
Estimate
2e−06
1e−06
0e+00
Bootstrap
Gaussian
Method


95% Confidence Intervals for floors
Estimate
−0.02
−0.04
−0.06
−0.08
Bootstrap
Gaussian
Method


95% Confidence Intervals for grade
Estimate
0.11
0.10
0.09
0.08
0.07
Bootstrap
Gaussian
Method


95% Confidence Intervals for sqft_living15
Estimate
6e−05
4e−05
2e−05
0e+00
Bootstrap
Gaussian
Method



Commentary on Confidence Intervals

The 95% confidence intervals show notable variability in their widths across features. For some features, the bootstrap intervals are wider than the Gaussian intervals, while for others, the Gaussian intervals are wider or roughly the same.

Another key difference is that Gaussian intervals are always centered on the original OLS coefficient estimate, reflecting the symmetric assumption of the normal distribution. In contrast, bootstrap intervals may be asymmetric and are not necessarily centered on the original estimate, since they are based on the empirical distribution of resampled coefficients.

Conformal Prediction (Jackknife+).

Conformal prediction (CP) is a method for creating **valid prediction intervals** for new observations without assuming a specific distribution for the errors. Jackknife+ is a variant of CP that provides finite-sample coverage guarantees for these intervals. It works by repeatedly leaving out one observation from the training set, fitting a model on the remaining data, and predicting the left-out point. The residuals from these leave-one-out predictions are then used to adjust predictions on the test set, producing intervals that account for uncertainty in both the model and the data. This approach is especially useful when we want reliable prediction intervals without making strict distributional assumptions.

Using Jackknife+ allows us to quantify **uncertainty in predictions** on the test dataset, complementing the standard and bootstrap confidence intervals computed for the regression coefficients.

Variable Selection.

For the conformal prediction analysis, the following features were selected:

- Total square footage (`sqft_living`)
- Lot size (`sqft_lot`)
- Grade (`grade`)
- Basement square footage (`sqft_basement`)

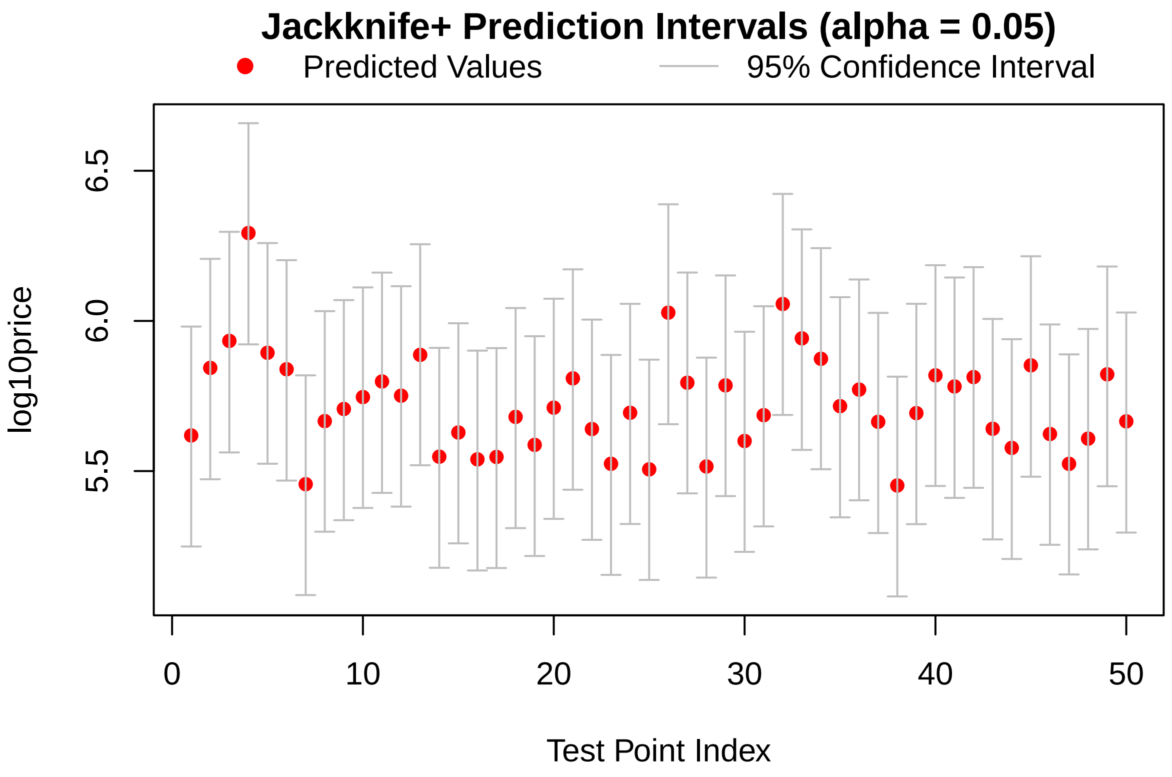
No variable transformations were applied. Since the provided test dataset did not include house prices, distribution-free 95% confidence intervals for the test set were generated and written to `jackknife_plus_intervals.dat`. This file contains the predicted confidence intervals for `price` for each house, allowing for future evaluation once the true prices become available.

```r
# Load test data and select features
test_data = read.table("data/king_county_test_2.dat", header = TRUE)
selected_vars = c("sqft_living", "sqft_lot", "grade", "sqft_basement")
test_data = test_data[, selected_vars]

# Store number of training and test samples
n = nrow(train_data)
num_test = nrow(test_data)

# Prepare storage for leave-one-out models and predictions
loo_models = vector("list", n)
loo_preds_train = numeric(n)

# Fit leave-one-out linear models and predict left-out points
for (i in 1:n) {
  model_i = lm(log10price ~ sqft_living + sqft_lot + grade + sqft_basement,
               data = train_data[-i, ])
  loo_models[[i]] = model_i
  loo_preds_train[i] = predict(model_i, newdata = train_data[i, ])
}

# Compute absolute residuals for Jackknife+
residuals = abs(train_data$log10price - loo_preds_train)

# Prepare matrix for test prediction intervals
intervals = matrix(0, nrow = num_test, ncol = 2)
```

```r
# Compute interval quantile indices for 95% coverage
a = floor(0.5 * n * 0.05)
b = ceiling(n * (1 - 0.5*0.05))

# Generate Jackknife+ prediction intervals for each test point
for (j in 1:num_test) {
  x_test = test_data[j, ]
  preds = sapply(loo_models, function(model) predict(model, newdata = x_test))

  lower_bounds = preds - residuals
  upper_bounds = preds + residuals

  sorted_lowers = sort(lower_bounds)
  sorted_uppers = sort(upper_bounds)

  a_index = max(1, a)
  b_index = min(n, b)

  intervals[j, 1] = sorted_lowers[a_index]
  intervals[j, 2] = sorted_uppers[b_index]
}

# Convert from log10 back to price and save intervals into file
results = 10^(intervals)
write.table(results, file = "results/jackknife_plus_intervals.dat",
            row.names = FALSE, col.names = FALSE)

# Fit OLS on full training data for reference predictions
test_model = lm(log10price ~ sqft_living + sqft_lot + grade + sqft_basement,
                data = train_data)
test_preds = predict(test_model, newdata = test_data)
```

After performing conformal prediction with Jackknife+, we created a plot showing the first fifty predicted `log10price` values for the test dataset along with their associated prediction intervals.

```r
# Plot the first 50 predicted log10 prices for the test set
plot(
  1:50, test_preds[1:50],
  pch = 16,
  col = "red",
  ylim = c(min(intervals[1:50,1]),max(intervals[1:50,2])),
  xlab = "Test Point Index",
  ylab = "log10price",
  main = "Jackknife+ Prediction Intervals (alpha = 0.05)"
)

# Add vertical lines representing the 95% Jackknife+ prediction intervals
arrows(
  x0 = 1:50,
  y0 = intervals[1:50, 1],
  x1 = 1:50,
  y1 = intervals[1:50, 2],
  angle = 90,
  code = 3,
  length = 0.05,
  col = "gray"
)

# Add a legend for points and prediction intervals
legend("top",
       legend = c("Predicted Values", "95% Confidence Interval"),
       pch = c(19, NA),
       lty = c(NA, 1),
       col = c("red", "gray"),
       horiz = TRUE,
       bty = "n",
       inset = c(0, -0.15),
       xpd = TRUE)
```



The predicted values were originally computed on the $\log_{10}$ scale. Here, we transform them back to the original scale for interpretability.

```
# Convert predicted log10 prices back to the original price scale
test_preds = 10^(test_preds)
```

Discussion of Jackknife+ Confidence Intervals

The 95% confidence intervals for the test data points exhibited a fairly large spread. This is partly due to performing the predictions on the $\log_{10}$(price) scale, which expands the range of predicted values and therefore the resulting intervals. Additionally, choosing 95% as the coverage level naturally produces wider intervals, since higher confidence requires a broader range to ensure the true value falls within the interval. Overall, these intervals provide a conservative estimate of uncertainty for unseen homes in the test dataset.

Key Takeaways

- There is substantial **uncertainty in predicting housing prices**, even for logical features like square footage or number of bedrooms.
- Some features that might seem important (e.g., bedrooms, floors) had **smaller effects** on the model than expected.
- **Jackknife+ conformal prediction** provided distribution-free prediction intervals, theoretically guaranteeing ~95% coverage, demonstrating its power for generating reliable uncertainty estimates.
- Using **log-transformed prices** affects the scale of predictions and intervals, which should be considered when interpreting results.

Overall, the project highlights the **challenges of predicting housing prices** and the value of combining classical regression, bootstrap intervals, and conformal prediction to assess model uncertainty.