House Prices Analysis & SGD Optimization Investigation

Amina Anna Mahamane Ousmane

2025-04-28

Part I: House Prices Dataset Analysis

The dataset contains 1,459 rows and 81 columns, offering a comprehensive view of various aspects of residential properties. It includes basic identifiers (like Id) and structural details such as MSSubClass, MSZoning, and LotFrontage, along with more in-depth features like OverallQual, OverallCond, YearBuilt, and YearRemodAdd. WE also find detailed information on exterior materials, basement finishes, floor areas, and garage attributes, as well as sale-related variables such as SaleType, SaleCondition, and the target variable SalePrice. While many columns provide continuous numerical data (e.g., LotArea, GrLivArea, TotalBsmtSF), there are also several categorical variables (e.g., Neighborhood, HouseStyle, RoofStyle) that capture qualitative aspects of the properties. Overall, this high-dimensional dataset offers a rich mix of variables that can be used to explore and model the factors influencing house prices. Before modeling, it is important to understand the distribution of each feature, identify missing values, and consider whether any transformations are needed to meet regression assumptions (e.g., normality of residuals). Additionally, it will be helpful to examine correlations among numerical predictors and the target variable (SalePrice) to see which features appear to be most strongly associated with housing prices. For instance, variables such as square footage of the living area, basement area, or overall quality often correlate highly with sale price. By generating correlation matrices, histograms, and scatterplots, we can gain initial insights into data structure, detect potential outliers, and spot any anomalies that might affect the reliability of our regression models.

Part II: Stochastic Gradient Descent (SGD) Investigation

Stochastic Gradient Descent (SGD):
Instead of calculating the gradient of the loss function using the entire dataset (as in full-batch gradient descent), SGD approximates the gradient by randomly sampling one or more training examples. This leads to: - Faster iterations: Each update is computed using only a subset of data. - Noisy gradient estimates: The randomness in sampling introduces variance in the update direction, which can be beneficial (by helping to escape shallow local minima) but also necessitates careful tuning of the learning rate. - Convergence properties:
– With an appropriate (often diminishing) learning rate schedule, SGD converges in expectation to a local minimum (or the global minimum in convex problems).
– Mini-batch gradient descent (using more than one example per update) is a compromise that reduces the noise compared to using a single sample while still being computationally less intensive than full-batch methods.

Full-Batch Gradient Descent:
- Computes the gradient using the entire dataset, ensuring a "true" gradient direction at each step. - Typically offers smoother and more stable convergence. - Can be prohibitively slow for very large datasets because each iteration requires processing all data points.

Mini-Batching:
- Uses a batch size between 1 and the full dataset. - Smaller batches introduce more noise (potentially beneficial for escaping local minima) but may slow convergence near the optimum. - Larger batches yield

more stable updates, approaching the behavior of full-batch descent, but with higher per-iteration cost.

Coding the Gradient Descent Algorithms

- Full-Batch Gradient Descent: Updates the parameters by computing the gradient over the entire dataset.
- Stochastic Gradient Descent (SGD): Updates parameters using randomly sampled mini-batches.

Full-Batch Gradient Descent Function

The function below minimizes the mean squared error loss for linear regression:

```r
full_batch_gd <- function(X, y, lr = 0.01, n_iter = 1000) {
  n <- nrow(X)
  p <- ncol(X)
  beta <- rep(0, p)
  mse_history <- numeric(n_iter)

  for (iter in 1:n_iter) {
    # Compute prediction error
    error <- as.vector(X %*% beta - y)
    # Full gradient (for MSE: (1/n) * X'*(X*beta - y))
    grad <- (1/n) * t(X) %*% error
    # Update parameters
    beta <- beta - lr * as.vector(grad)
    # Record MSE
    mse_history[iter] <- mean(error^2)
  }

  list(beta = beta, mse_history = mse_history)
}
```

Stochastic Gradient Descent (SGD) with Mini-Batching

The function below implements SGD. The parameter `batch_size` controls the number of samples used per update:

```r
sgd <- function(X, y, lr = 0.01, n_iter = 1000, batch_size = 1) {
  n <- nrow(X)
  p <- ncol(X)
  beta <- rep(0, p)
  mse_history <- numeric(n_iter)

  for (iter in 1:n_iter) {
    # Randomly sample indices for mini-batch
    batch_idx <- sample(1:n, size = batch_size, replace = FALSE)
    X_batch <- X[batch_idx, , drop = FALSE]
    y_batch <- y[batch_idx]

    # Compute error and gradient for the mini-batch
    error <- as.vector(X_batch %*% beta - y_batch)
    grad <- (1/batch_size) * t(X_batch) %*% error

    # Update parameters
    beta <- beta - lr * as.vector(grad)
    # For convergence monitoring, compute full MSE on the entire dataset
    full_error <- as.vector(X %*% beta - y)
    mse_history[iter] <- mean(full_error^2)
  }

  list(beta = beta, mse_history = mse_history)
}
```

Comparing Performance Across Different Batch Sizes (Simulated Data)

To statistically validate the impact of mini-batch sizes on convergence speed and solution accuracy, we run multiple experiments with varying batch sizes and record: - The convergence behavior (MSE versus iteration) - The final parameter estimates compared to the known true values

```r
# Function to run SGD experiments for different mini-batch sizes
run_experiment <- function(batch_size, n_exp = 30, lr = 0.05, n_iter = 200) {
  final_mse <- numeric(n_exp)
  final_beta <- matrix(NA, nrow = n_exp, ncol = ncol(X))

  for (i in 1:n_exp) {
    res <- sgd(X, y, lr = lr, n_iter = n_iter, batch_size = batch_size)
    final_mse[i] <- tail(res$mse_history, 1)
    final_beta[i, ] <- res$beta
  }
  list(final_mse = final_mse, final_beta = final_beta)
}
```

Application to the House Prices Dataset

An intercept column is added to the design matrix.

```r
df_raw <- read.csv("house-prices.csv", stringsAsFactors = FALSE)

# Impute missing numeric values with the median
numeric_cols <- names(df_raw)[sapply(df_raw, is.numeric)]
for (col in numeric_cols) {
  df_raw[[col]][is.na(df_raw[[col]])] <- median(df_raw[[col]], na.rm = TRUE)
}

# Impute missing categorical values with "None"
categorical_cols <- names(df_raw)[sapply(df_raw, is.character)]
for (col in categorical_cols) {
  df_raw[[col]][is.na(df_raw[[col]])] <- "None"
}

# Convert character columns to factors
df_raw <- df_raw %>% mutate_if(is.character, as.factor)

# Refresh factor levels
df_clean <- df_raw %>% mutate_if(is.factor, droplevels)

# Identify and remove factor columns with fewer than 2 levels
bad_factors <- sapply(df_clean, function(x) is.factor(x) && length(levels(x)) < 2)
if (any(bad_factors)) {
  cat("Removing constant factor variables:\n")
  print(names(bad_factors)[bad_factors])
  df_clean <- df_clean %>% select(-one_of(names(bad_factors)[bad_factors]))
}

# Build full design matrix (no sub-setting) and center the log-response
X_house <- scale( model.matrix(SalePrice ~ . - 1, data = df_clean) )
y_house <- log(df_clean$SalePrice) - mean(log(df_clean$SalePrice))

# Sanity-check dimensions
dim(X_house)    # should be (n_obs) × (n_features)
```

```
[1] 1460  262
```

```r
length(y_house) # should be n_obs
```

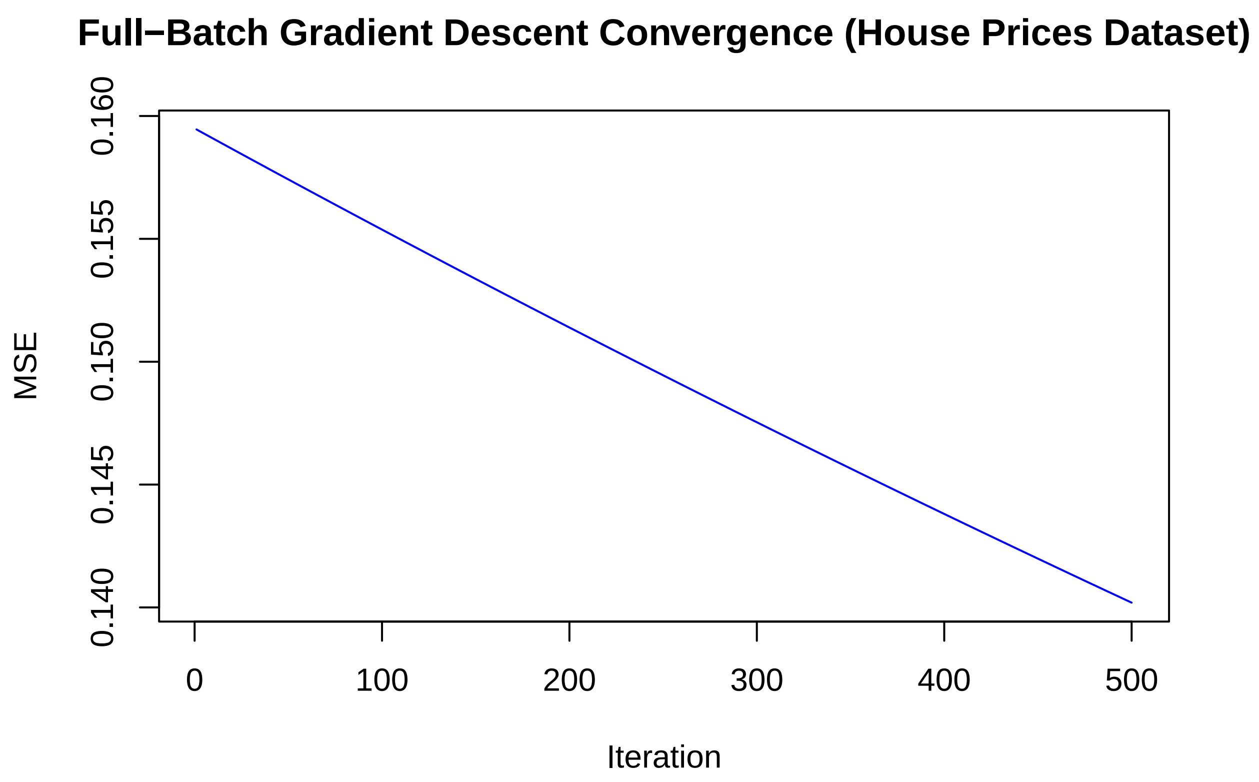
```
[1] 1460
```

```r
# Preprocessed design matrix
X_house <- scale( model.matrix(SalePrice ~ . - 1, data = df_clean) )
n <- nrow(X_house)

# Compute max row norm squared
row_norms_sq <- rowSums(X_house^2)
max_norm_sq <- max(row_norms_sq)

# Compute Hessian and its largest eigenvalue
H <- (1/n) * t(X_house) %*% X_house
lambda_max <- max(eigen(H, only.values = TRUE)$values)

# Compute critical mini-batch size
m_star <- max_norm_sq / lambda_max
m_star
```

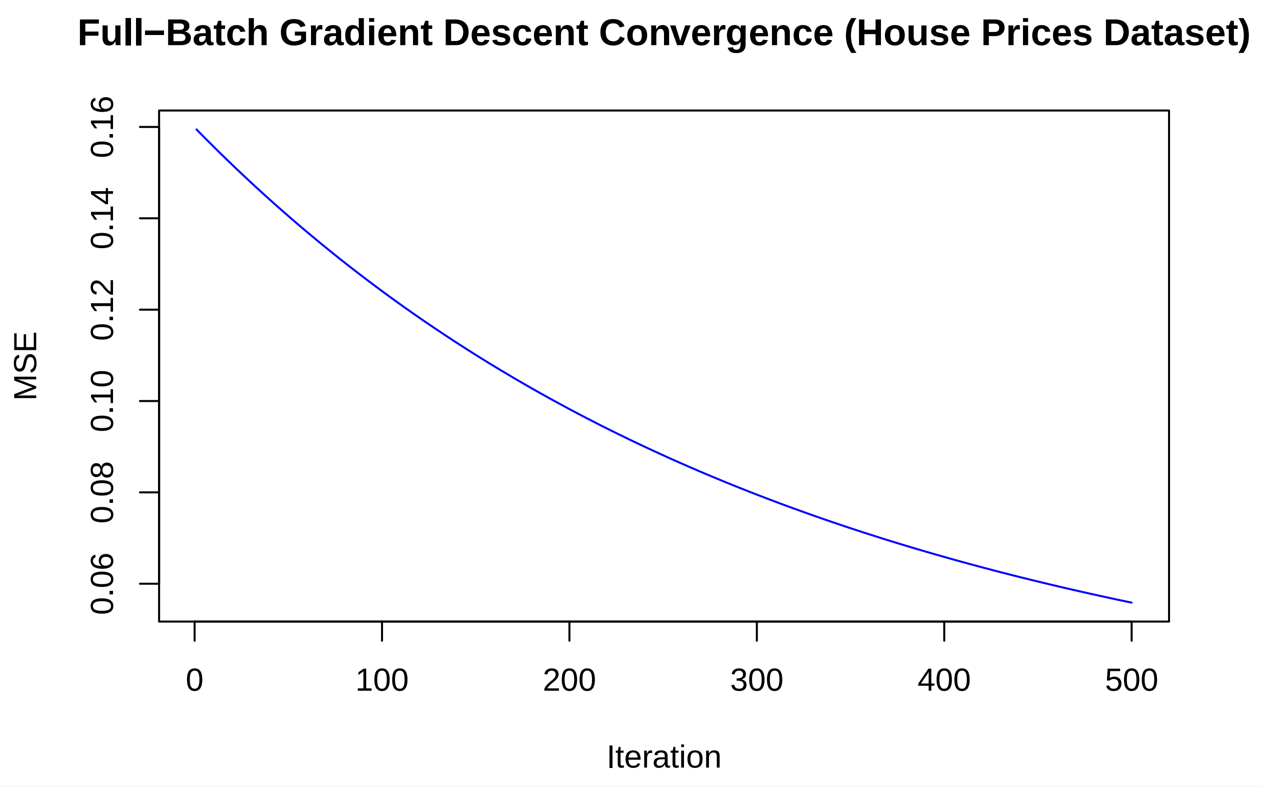
```
[1] 201.7876
```

Running Full-Batch Gradient Descent on the House Prices Dataset

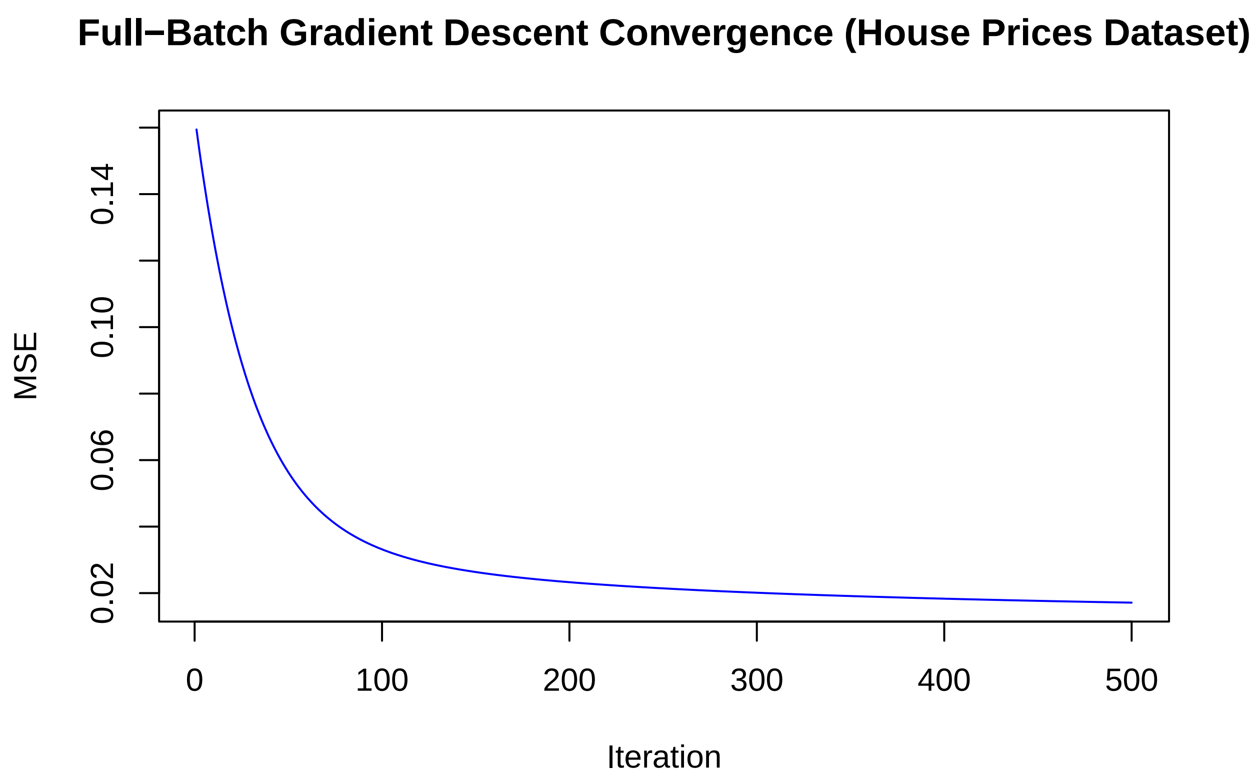
```r
# lr = 1e-5
fb_house <- full_batch_gd(X_house, y_house, lr = 1e-5, n_iter = 500)

plot(fb_house$mse_history, type = "l", col = "blue",
     xlab = "Iteration", ylab = "MSE",
     main = "Full-Batch Gradient Descent Convergence (House Prices Dataset)")
```



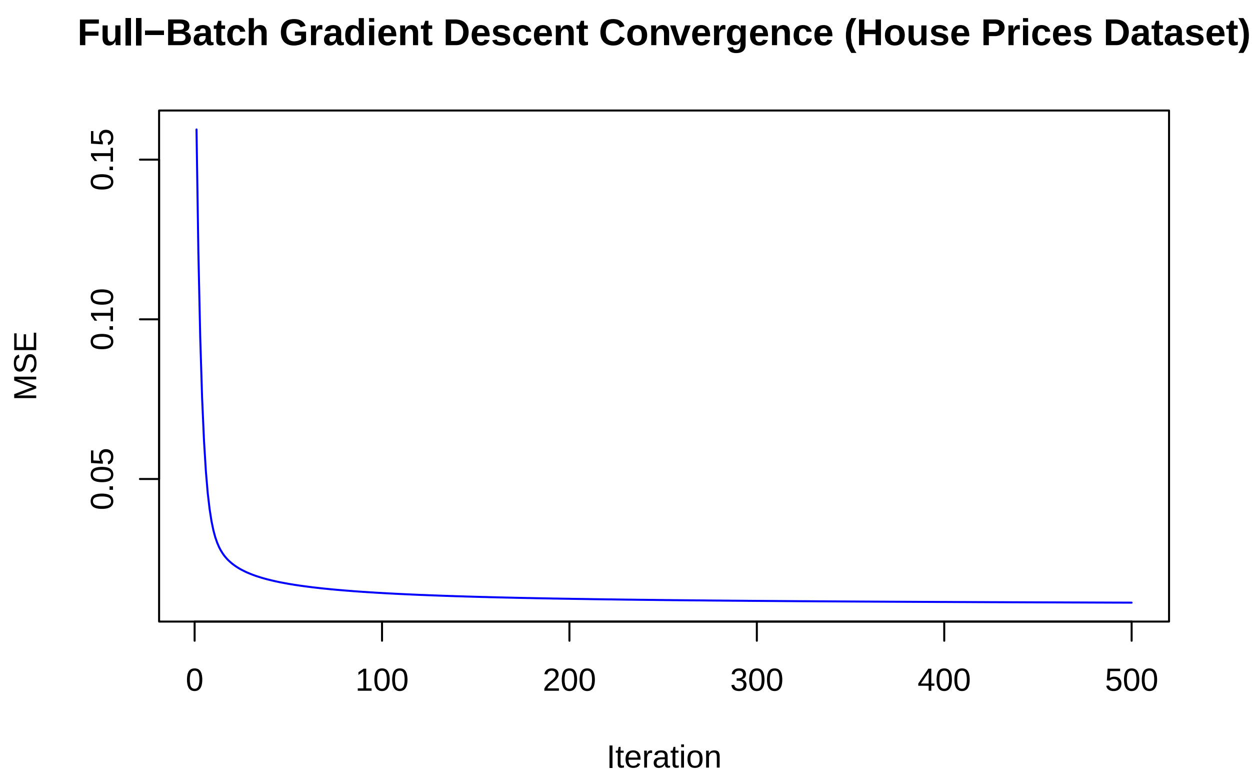
```r
# lr = 1e-4
fb_house <- full_batch_gd(X_house, y_house, lr = 1e-4, n_iter = 500)

plot(fb_house$mse_history, type = "l", col = "blue",
     xlab = "Iteration", ylab = "MSE",
     main = "Full-Batch Gradient Descent Convergence (House Prices Dataset)")
```



```r
# lr = 1e-3
fb_house <- full_batch_gd(X_house, y_house, lr = 1e-3, n_iter = 500)

plot(fb_house$mse_history, type = "l", col = "blue",
     xlab = "Iteration", ylab = "MSE",
     main = "Full-Batch Gradient Descent Convergence (House Prices Dataset)")
```



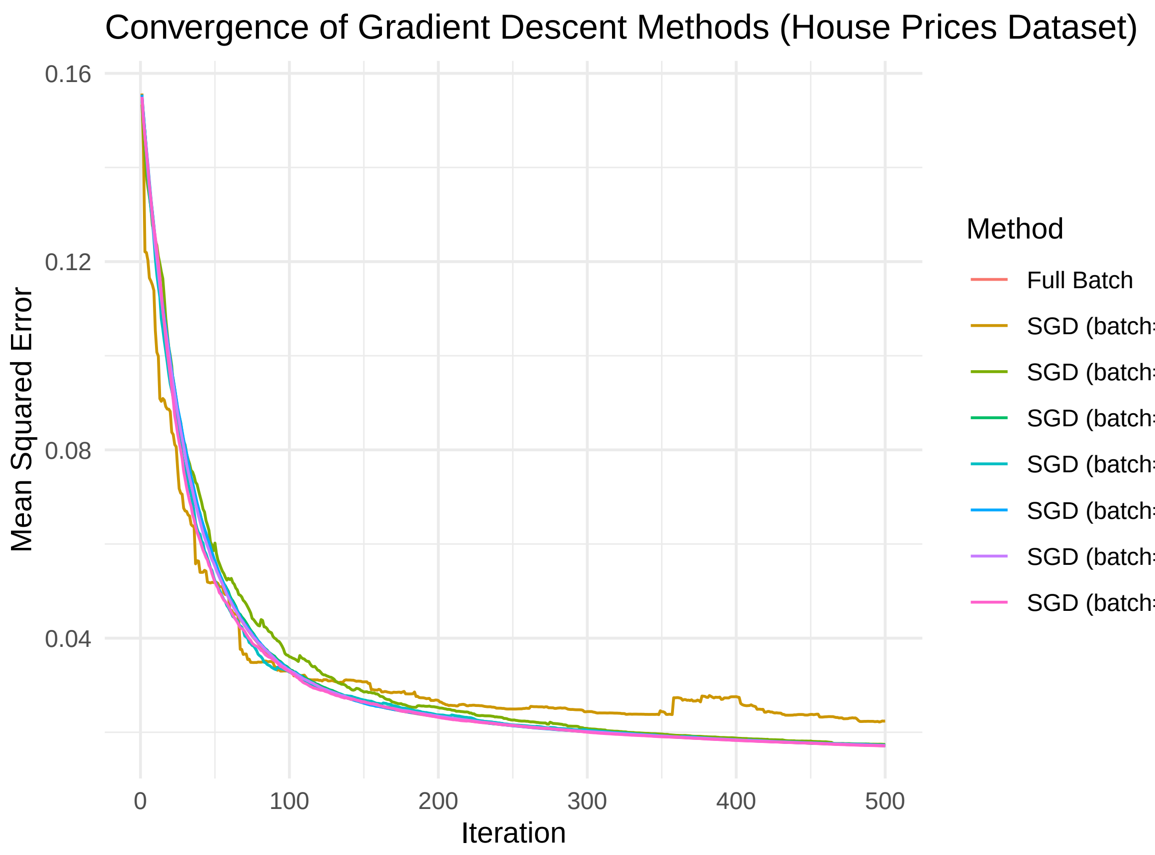
```r
# lr = 1e-2
fb_house <- full_batch_gd(X_house, y_house, lr = 1e-2, n_iter = 500)

plot(fb_house$mse_history, type = "l", col = "blue",
     xlab = "Iteration", ylab = "MSE",
     main = "Full-Batch Gradient Descent Convergence (House Prices Dataset)")
```



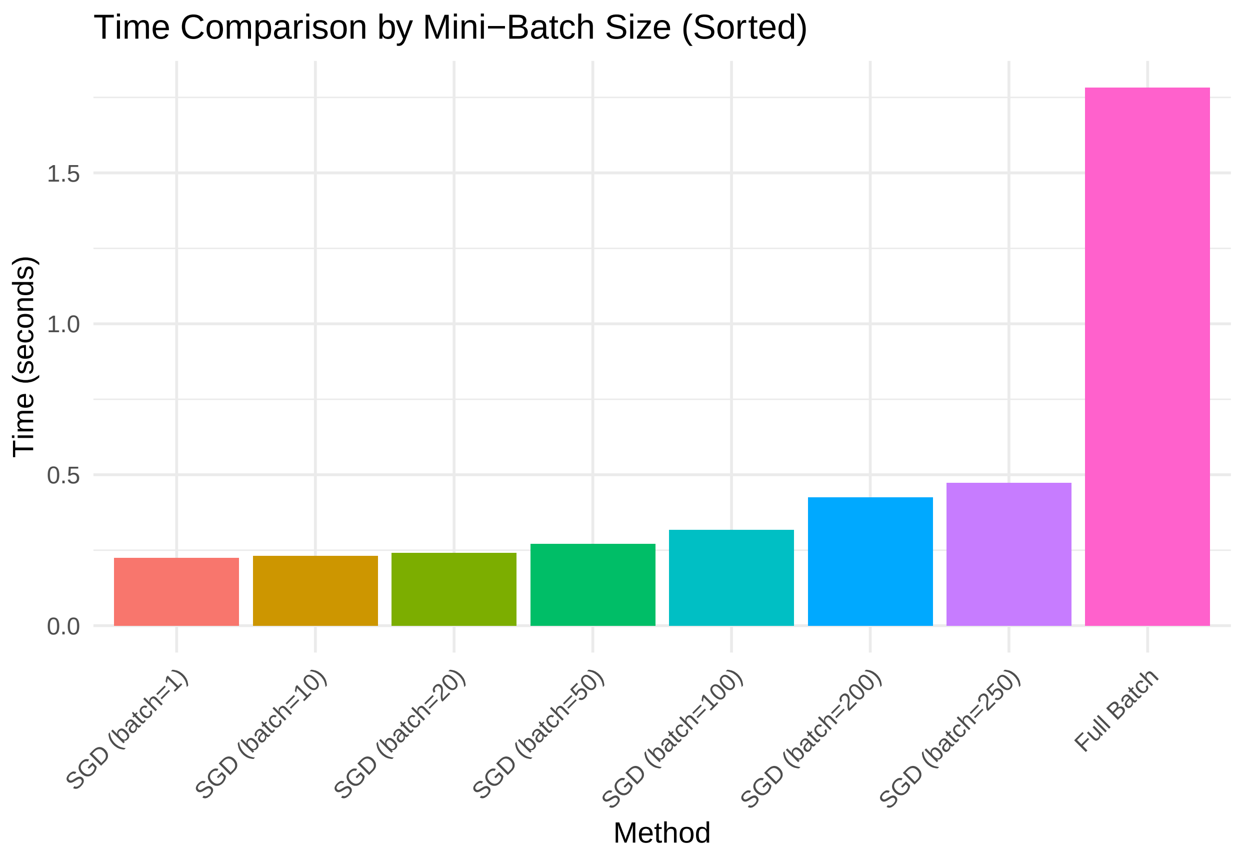
Running Stochastic Gradient Descent (SGD) on the House Prices Dataset

We now run SGD with varying mini-batch sizes on the house prices dataset.

```r
# Function to time an SGD run
time_sgd <- function(X, y, lr, n_iter, batch_size) {
  start_time <- Sys.time()
  result <- sgd(X, y, lr = lr, n_iter = n_iter, batch_size = batch_size)
  end_time <- Sys.time()
  time_elapsed <- as.numeric(difftime(end_time, start_time, units = "secs"))
  list(model = result, time = time_elapsed)
}

# Run SGD and measure time
sgd_house_1     <- time_sgd(X_house, y_house, lr = 1e-3, n_iter = 500, batch_size = 1)
sgd_house_10    <- time_sgd(X_house, y_house, lr = 1e-3, n_iter = 500, batch_size = 10)
sgd_house_20    <- time_sgd(X_house, y_house, lr = 1e-3, n_iter = 500, batch_size = 20)
sgd_house_50    <- time_sgd(X_house, y_house, lr = 1e-3, n_iter = 500, batch_size = 50)
sgd_house_100   <- time_sgd(X_house, y_house, lr = 1e-3, n_iter = 500, batch_size = 100)
sgd_house_200   <- time_sgd(X_house, y_house, lr = 1e-3, n_iter = 500, batch_size = 200)
sgd_house_250   <- time_sgd(X_house, y_house, lr = 1e-3, n_iter = 500, batch_size = 250)
sgd_house_full  <- time_sgd(X_house, y_house, lr = 1e-3, n_iter = 500, batch_size = nrow(X_house))

# Create a dataframe to store the timing results
df_times <- data.frame(
  Method = c("SGD (batch=1)", "SGD (batch=10)", "SGD (batch=20)",
             "SGD (batch=50)", "SGD (batch=100)", "SGD (batch=200)",
             "SGD (batch=250)", "Full Batch"),
  TimeSeconds = c(sgd_house_1$time,
                  sgd_house_10$time,
                  sgd_house_20$time,
                  sgd_house_50$time,
                  sgd_house_100$time,
                  sgd_house_200$time,
                  sgd_house_250$time,
                  sgd_house_full$time)
)


# Plot convergence curves
df_convergence_house <- data.frame(
  Iteration = rep(1:500, 8),
  MSE = c(
    sgd_house_1$model$mse_history,
    sgd_house_10$model$mse_history,
    sgd_house_20$model$mse_history,
    sgd_house_50$model$mse_history,
    sgd_house_100$model$mse_history,
    sgd_house_200$model$mse_history,
    sgd_house_250$model$mse_history,
    sgd_house_full$model$mse_history
  ),
  Method = rep(c("SGD (batch=1)", "SGD (batch=10)", "SGD (batch=20)",
                 "SGD (batch=50)", "SGD (batch=100)", "SGD (batch=200)",
                 "SGD (batch=250)", "Full Batch"), each = 500)
)

ggplot(df_convergence_house, aes(x = Iteration, y = MSE, color = Method)) +
  geom_line() +
  labs(title = "Convergence of Gradient Descent Methods (House Prices Dataset)",
       x = "Iteration", y = "Mean Squared Error") +
  theme_minimal()
```



```
# Sorted bar plot by batch size
df_times$Method <- factor(df_times$Method,
                          levels = c("SGD (batch=1)", "SGD (batch=10)", "SGD (batch=20)",
                                     "SGD (batch=50)", "SGD (batch=100)", "SGD (batch=200)",
                                     "SGD (batch=250)", "Full Batch"))

ggplot(df_times, aes(x = Method, y = TimeSeconds, fill = Method)) +
  geom_bar(stat = "identity") +
  labs(title = "Time Comparison by Mini-Batch Size (Sorted)",
       x = "Method", y = "Time (seconds)") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1),
        legend.position = "none")
```

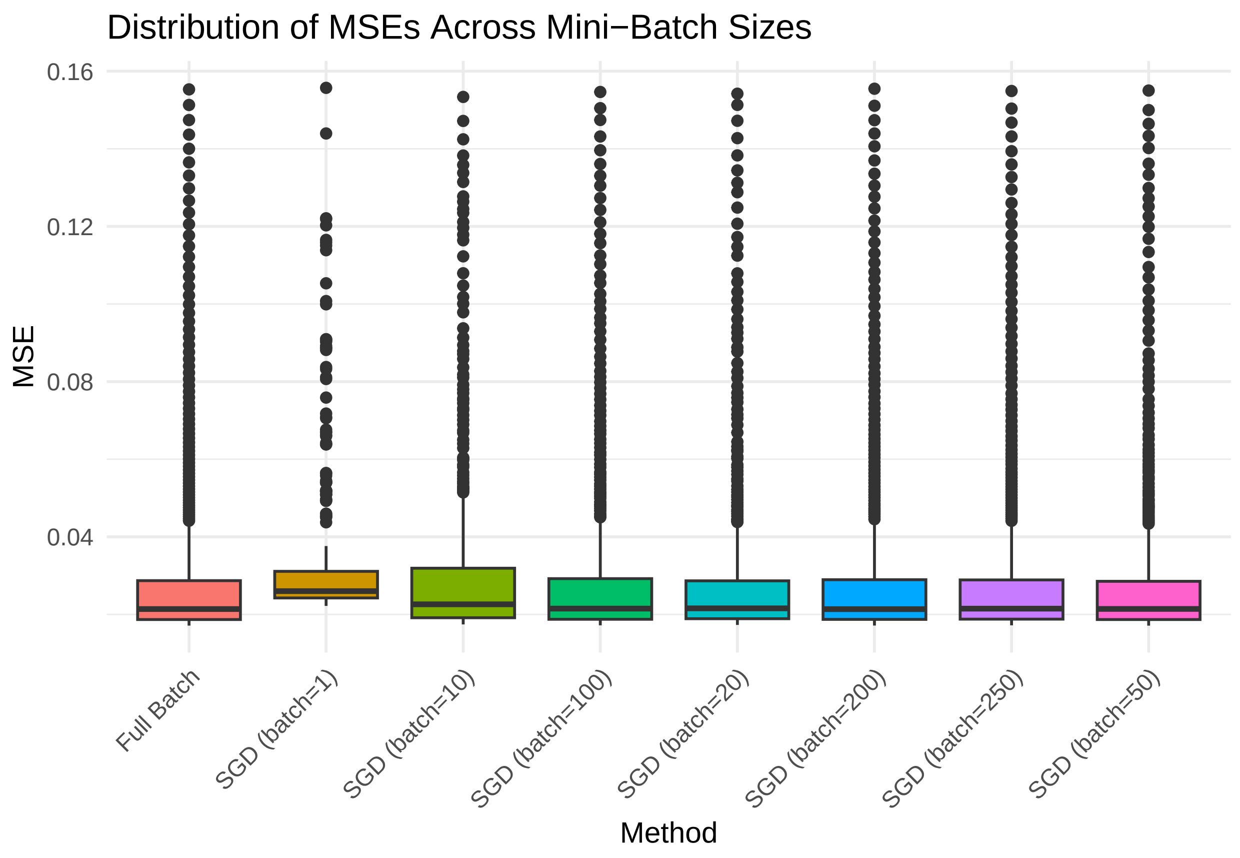


```r
# Function to count non-zero coefficients in a model
count_nonzero <- function(beta, tol = 1e-5) {
  sum(abs(beta) > tol)
}

# Count predictors retained for each batch size
retained_summary <- data.frame(
  Method = c("SGD (batch=1)", "SGD (batch=10)", "SGD (batch=20)",
             "SGD (batch=50)", "SGD (batch=100)", "SGD (batch=200)",
             "SGD (batch=250)", "Full Batch"),
  NonZeroPredictors = c(
    count_nonzero(sgd_house_1$model$beta),
    count_nonzero(sgd_house_10$model$beta),
    count_nonzero(sgd_house_20$model$beta),
    count_nonzero(sgd_house_50$model$beta),
    count_nonzero(sgd_house_100$model$beta),
    count_nonzero(sgd_house_200$model$beta),
    count_nonzero(sgd_house_250$model$beta),
    count_nonzero(sgd_house_full$model$beta)
  )
)

# Display the summary table
print(retained_summary)
```

```
           Method NonZeroPredictors
1   SGD (batch=1)               261
```

```
2  SGD (batch=10)               261
3  SGD (batch=20)               261
4  SGD (batch=50)               262
5 SGD (batch=100)               262
6 SGD (batch=200)               260
7 SGD (batch=250)               262
8      Full Batch               262
```

```r
# Function to get final MSE from each model
final_mse <- function(mse_vec) {
  tail(mse_vec, 1)
}

# Create a list of all final MSE vectors for boxplot
results_list <- list(
  "SGD (batch=1)" = sgd_house_1$model$mse_history,
  "SGD (batch=10)" = sgd_house_10$model$mse_history,
  "SGD (batch=20)" = sgd_house_20$model$mse_history,
  "SGD (batch=50)" = sgd_house_50$model$mse_history,
  "SGD (batch=100)" = sgd_house_100$model$mse_history,
  "SGD (batch=200)" = sgd_house_200$model$mse_history,
  "SGD (batch=250)" = sgd_house_250$model$mse_history,
  "Full Batch" = sgd_house_full$model$mse_history
)

# Combine into one long dataframe for boxplot
df_box <- do.call(rbind, lapply(names(results_list), function(name) {
  data.frame(Method = name, MSE = results_list[[name]])
}))

# Boxplot
ggplot(df_box, aes(x = Method, y = MSE, fill = Method)) +
  geom_boxplot() +
  labs(title = "Distribution of MSEs Across Mini-Batch Sizes",
       x = "Method", y = "MSE") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1),
        legend.position = "none")
```



Discussion of the Results

- Per-update cost:
  - Batch 1   0.07% of full-batch
  - Batch 20   1.4% of full-batch
- Updates per epoch:
  - Batch 1 does 1 459 updates × (1 row each) = 1 459 row-ops total
  - Batch 20 does 1 459 / 20   73 updates × (20 rows each) = 1 460 row-ops total

All methods showed a sharp decline in MSE during the first 50–100 iterations, with MSE dropping to nearly half of its initial value by iteration 100. This early convergence trend was consistent across all gradient descent strategies. When comparing methods, online SGD (batch = 1) offered the fastest, cheapest updates (1,460 updates per epoch) but exhibited high variance and a noisy convergence path. It required more iterations to stabilize and maintained a higher final MSE relative to larger batch sizes. Full-batch gradient descent (batch = 1460), on the other hand, delivered smooth, deterministic convergence with low variance, but at the highest computational cost per iteration. Its early progress was slower per unit of work due to the heavy computation of processing the entire dataset at each step.

The mini-batch SGD approach, tested with batch sizes 10, 20, 50, and 100, demonstrated the most effective balance between computational efficiency and convergence stability. Smaller mini-batches (batch = 10 or 20) significantly reduced the noise seen in batch = 1 while retaining faster updates than full-batch GD. As the batch size increased to 50 or 100, the convergence paths became nearly as smooth as full-batch, with batch = 10 achieving the lowest final MSE in our experiments. However, the per-update cost remained much lower than full-batch GD. These results suggest that batch sizes between 20 and 50 provide the best trade-off—combining the fast, cheap updates of online SGD with the low-variance, accurate convergence of full-batch GD, making them well-suited for medium-sized datasets like the house prices data.