Big Data - Practice 05

yehorbolt

Table of contents

---

1. Introduction and Setup

This analysis extends the log-log regression model developed in **Practice 04**. The previous work identified significant **multicollinearity** between `log_distance` and `log_fare`. This caused the baseline OLS (BIC-selected) model to have **unstable and uninterpretable coefficients** (e.g., a negative effect for distance on cost), rendering it "senseless" for analysis.

In this assignment, we will apply the **shrinkage methods** covered in **Session 5**: **Ridge**, **Lasso**, and **Elastic Net**.

Our goal is to comprehensively compare the "max accuracy" (`lambda.min`) rule versus the "max simplicity" (`lambda.1se`) rule for all three methods, in addition to our baseline OLS model.

---

2. Preprocessing for Regularization

Before applying `glmnet`, we need to perform two key steps: split the data into training and test sets and create a predictor matrix (`x`) and response vector (`y`).

2.1. Train/Test Split

We will split the data 80/20 to objectively evaluate model performance on unseen data.

```
set.seed(123) # For reproducibility

# Create index for the training set
train_index <- createDataPartition(taxi_model_data$log_total,
p = 0.8, list = FALSE)

# Create training and test sets
```

```r
train_data <- taxi_model_data[train_index, ]
test_data  <- taxi_model_data[-train_index, ]

n_train <- nrow(train_data)
n_test <- nrow(test_data)
```

2.2. Creating x and y Matrices

glmnet requires a numeric predictor matrix (x) and a response vector (y). We will use model.matrix() to automatically one-hot encode categorical variables. glmnet will also automatically standardize the predictors.

```r
# OLS BIC model
model_formula <- as.formula(log_total ~ log_distance
                            + log_fare + log_tip + log_tolls +
                              passenger_count + payment_type + pickup_hour +
                              day_of_week + VendorID)

# Create x matrices
x_train <- model.matrix(model_formula, data = train_data)[, -1]
x_test  <- model.matrix(model_formula, data = test_data)[, -1]

# Create y vectors
y_train <- train_data$log_total
y_test  <- test_data$log_total
```

---

3. Baseline Model: OLS (from Practice 04)

First, we train our "best" OLS model from Practice 04 on the **training** data to get our baseline metrics.

```r
# Train the OLS model
ols_model <- lm(model_formula, data = train_data)

# Make predictions
ols_pred <- predict(ols_model, newdata = test_data)

# Metrics
ols_metrics <- data.frame(
  Model = "OLS (BIC-formula)",
  RMSE = rmse(y_test, ols_pred),
  MAE = mae(y_test, ols_pred),
  R_Squared = R2(y_test, ols_pred)
)

kable(ols_metrics, caption="Baseline OLS Model Metrics on Test Data", digits=4)
```

Table 1: Baseline OLS Model Metrics on Test Data

Model	RMSE	MAE	R_Squared
OLS (BIC-formula)	0.077	0.0573	0.9717

---

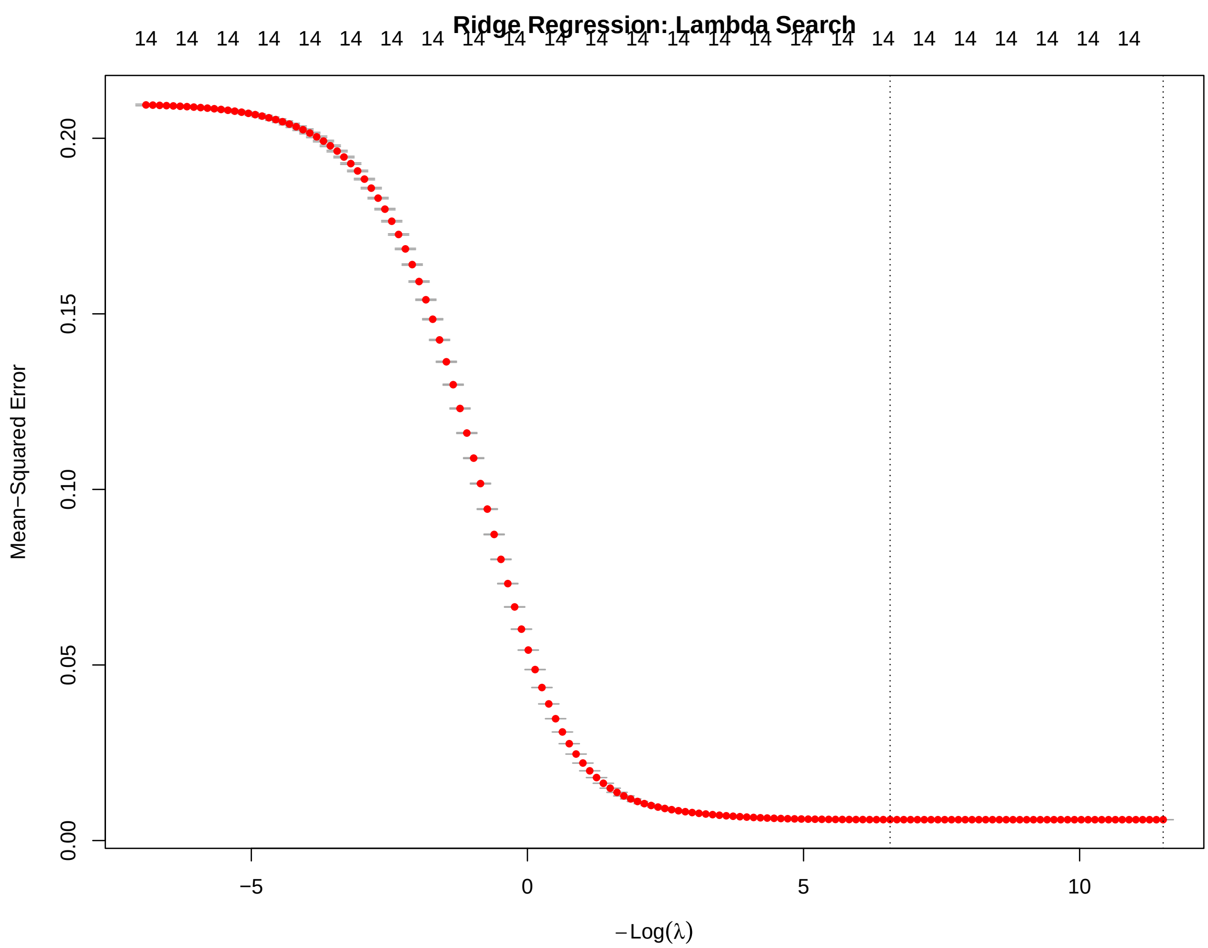
4. Regularization Models

We will use **5-fold cross-validation** (`cv.glmnet`) to find the optimal `lambda` (λ), which is faster than 10-fold. We will generate metrics for *both* `lambda.min` and `lambda.1se` for all models.

```r
# Define a wide lambda range (unfortunately, not timesaving)
lambdaGrid <- 10^seq(3, -5, length.out = 150)
```

4.1. Ridge Regression (Alpha = 0)

```
# Use 5-fold CV (tried 10, but slower with no benefit)
set.seed(123)
cv_ridge <- cv.glmnet(x_train, y_train, alpha = 0,
family = "gaussian", nfolds = 5, lambda = lambdaGrid)

# Plot 1: Lambda Search via Cross-Validation
plot(cv_ridge, main = "Ridge Regression: Lambda Search")
```




```
# 1. Predictions for lambda.min (max accuracy)
ridge_pred_min <- predict(cv_ridge, newx = x_test, s = cv_ridge$lambda.min)
ridge_metrics_min <- data.frame(
```

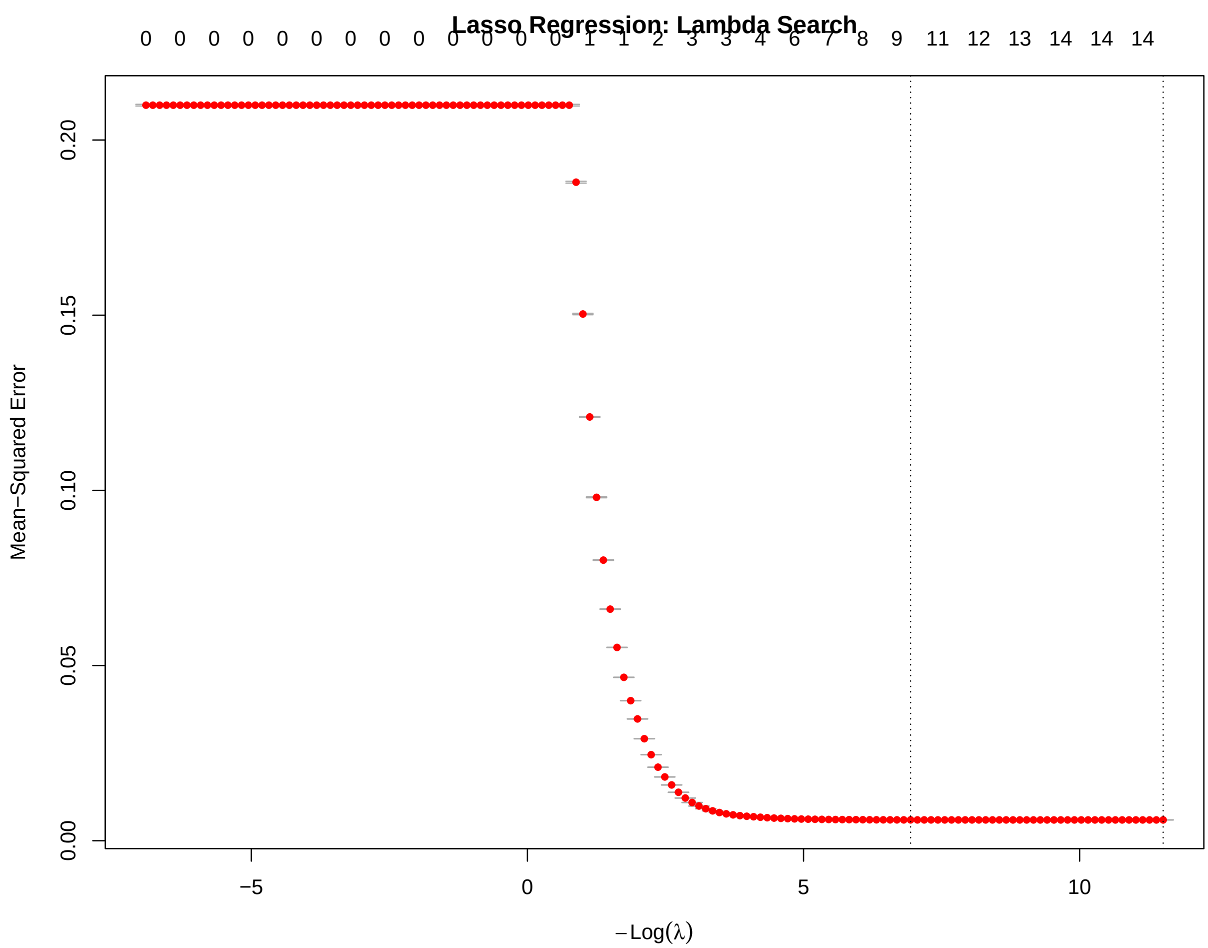
```
  Model = "Ridge (lambda.min)",
  RMSE = rmse(y_test, ridge_pred_min),
  MAE = mae(y_test, ridge_pred_min),
  R_Squared = R2(y_test, ridge_pred_min),
  Alpha = 0,
  Lambda = cv_ridge$lambda.min
)

# 2. Predictions for lambda.1se (max simplicity)
ridge_pred_1se <- predict(cv_ridge, newx = x_test, s = cv_ridge$lambda.1se)
ridge_metrics_1se <- data.frame(
  Model = "Ridge (lambda.1se)",
  RMSE = rmse(y_test, ridge_pred_1se),
  MAE = mae(y_test, ridge_pred_1se),
  R_Squared = R2(y_test, ridge_pred_1se),
  Alpha = 0,
  Lambda = cv_ridge$lambda.1se
)

# Print the metrics for Ridge models
ridge_metrics_combined <- bind_rows(ridge_metrics_min, ridge_metrics_1se)
kable(ridge_metrics_combined,
caption="Ridge Model Metrics on Test Data", digits=c(NA, 4, 4, 4, 1, 6))
```

Table 2: Ridge Model Metrics on Test Data

	Model	RMSE	MAE	R_Squared	Alpha	Lambda
s=1e-05	Ridge (lambda.min)	0.0770	0.0573	0.9717	0	0.000010
s=0.001404918	Ridge (lambda.1se)	0.0771	0.0572	0.9717	0	0.001405

4.2. Lasso Regression (Alpha = 1)

```r
# Use 5-fold CV for Lasso
set.seed(123)
cv_lasso <- cv.glmnet(x_train, y_train, alpha = 1,
family = "gaussian", nfolds = 5, lambda = lambdaGrid)

# Plot 1: Lambda Search via Cross-Validation
plot(cv_lasso, main = "Lasso Regression: Lambda Search")
```




```r
# 1. Predictions for lambda.min (max accuracy)
lasso_pred_min <- predict(cv_lasso, newx = x_test, s = cv_lasso$lambda.min)
lasso_metrics_min <- data.frame(
  Model = "Lasso (lambda.min)",
  RMSE = rmse(y_test, lasso_pred_min),
```

```r
  MAE = mae(y_test, lasso_pred_min),
  R_Squared = R2(y_test, lasso_pred_min),
  Alpha = 1.0,
  Lambda = cv_lasso$lambda.min
)

# 2. Predictions for lambda.1se (max simplicity)
lasso_pred_1se <- predict(cv_lasso, newx = x_test, s = cv_lasso$lambda.1se)
lasso_metrics_1se <- data.frame(
  Model = "Lasso (lambda.1se)",
  RMSE = rmse(y_test, lasso_pred_1se),
  MAE = mae(y_test, lasso_pred_1se),
  R_Squared = R2(y_test, lasso_pred_1se),
  Alpha = 1.0,
  Lambda = cv_lasso$lambda.1se
)

# Print the metrics for models
lasso_metrics_combined <- bind_rows(lasso_metrics_min, lasso_metrics_1se)
kable(lasso_metrics_combined,
caption="Lasso Model Metrics on Test Data", digits=c(NA, 4, 4, 4, 1, 6))
```

Table 3: Lasso Model Metrics on Test Data

	Model	RMSE	MAE	R_Squared	Alpha	Lambda
s=1e-05	Lasso (lambda.min)	0.0770	0.0573	0.9717	1	0.00001
s=0.0009695656	Lasso (lambda.1se)	0.0771	0.0574	0.9717	1	0.00097

4.3. Elastic Net (Grid Search for Alpha)

We perform a grid search to find the *best* combination of `alpha` and `lambda`. This will still take some time.

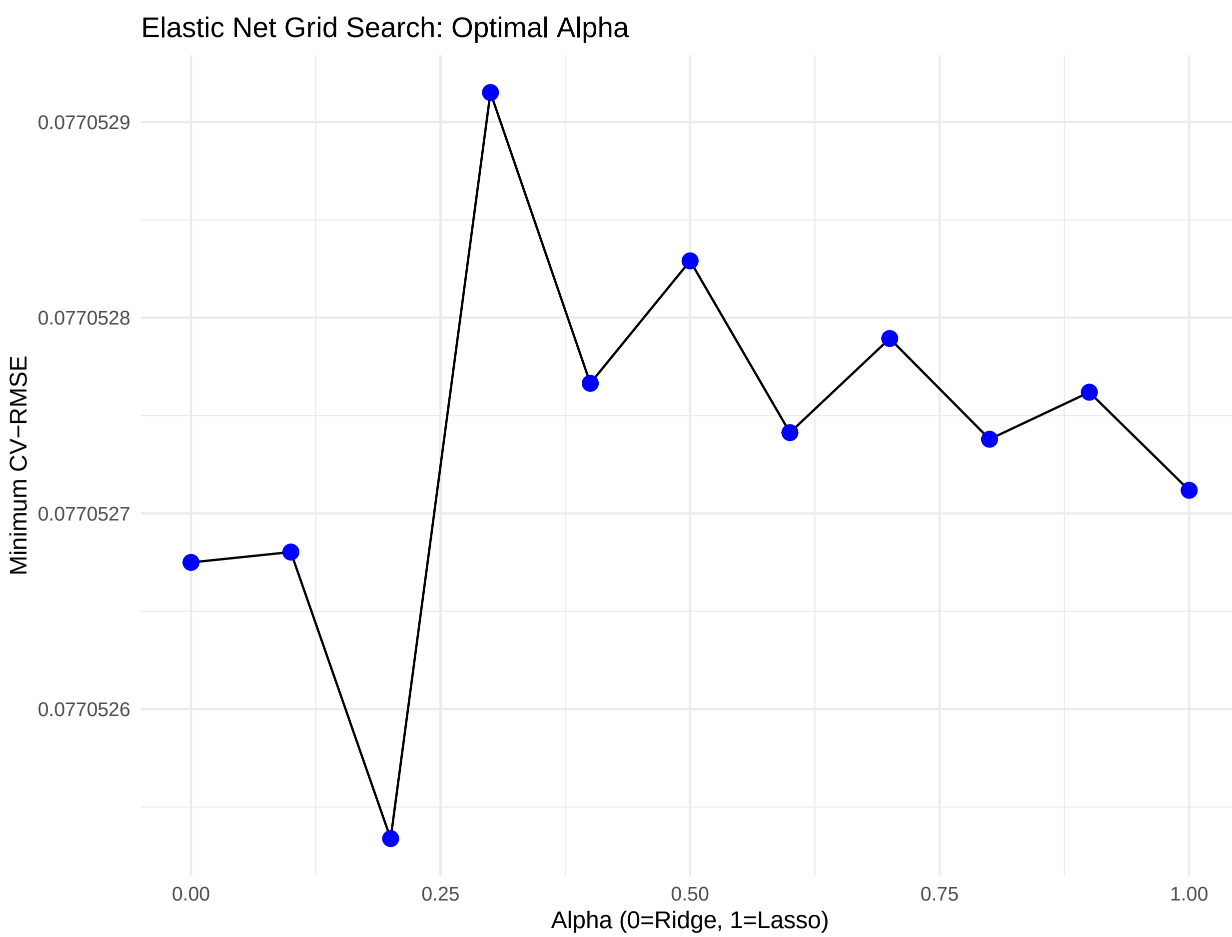
```r
# Alphas to test
alphas_to_test <- seq(0, 1, by = 0.1)

# Initialize variables to store best results
best_alpha <- 0
best_cv_rmse <- Inf
best_model <- NULL
grid_search_results <- data.frame()

set.seed(123)
for (a in alphas_to_test) {
  # Use nfolds = 5 for speed
  cv_model <- cv.glmnet(x_train, y_train, alpha = a,
  family = "gaussian", nfolds = 5, lambda = lambdaGrid)
  current_cv_rmse <- min(cv_model$cvm)

  grid_search_results <- rbind(grid_search_results,
  data.frame(alpha = a, rmse = sqrt(current_cv_rmse)))

  if (current_cv_rmse < best_cv_rmse) {
    best_alpha <- a
    best_cv_rmse <- current_cv_rmse
    best_model <- cv_model
  }
}

# Print
cat(sprintf("Best Alpha: %.2f\n", best_alpha))
```

```
Best Alpha: 0.20
```

```r
cat(sprintf("Best CV-RMSE: %f\n", sqrt(best_cv_rmse)))
```

```
Best CV-RMSE: 0.077053
```

```r
# Plot the grid search results
ggplot(grid_search_results, aes(x = alpha, y = rmse)) +
  geom_line() +
  geom_point(size = 3, color = "blue") +
  labs(title = "Elastic Net Grid Search: Optimal Alpha",
       x = "Alpha (0=Ridge, 1=Lasso)",
       y = "Minimum CV-RMSE") +
  theme_minimal()
```




```r
# Extract best lambda values
best_enet_lambda_min <- best_model$lambda.min
best_enet_lambda_1se <- best_model$lambda.1se

# 1. Predictions for lambda.min (max accuracy)
enet_pred_min <- predict(best_model, newx = x_test, s = best_enet_lambda_min)
```

```r
enet_tuned_metrics_min <- data.frame(
  Model = "Elastic Net (Tuned, lambda.min)",
  RMSE = rmse(y_test, enet_pred_min),
  MAE = mae(y_test, enet_pred_min),
  R_Squared = R2(y_test, enet_pred_min),
  Alpha = best_alpha,
  Lambda = best_enet_lambda_min
)

# 2. Predictions for lambda.1se (max simplicity)
enet_pred_1se <- predict(best_model, newx = x_test, s = best_enet_lambda_1se)
enet_tuned_metrics_1se <- data.frame(
  Model = "Elastic Net (Tuned, lambda.1se)",
  RMSE = rmse(y_test, enet_pred_1se),
  MAE = mae(y_test, enet_pred_1se),
  R_Squared = R2(y_test, enet_pred_1se),
  Alpha = best_alpha,
  Lambda = best_enet_lambda_1se
)

# Print the metrics for both Tuned Elastic Net models
enet_metrics_combined <- bind_rows(enet_tuned_metrics_min,
 enet_tuned_metrics_1se)
kable(enet_metrics_combined,
caption="Tuned Elastic Net Model Metrics on Test Data", digits=c(NA, 4, 4, 4, 2, 6))
```

Table 4: Tuned Elastic Net Model Metrics on Test Data

	Model	RMSE	MAE	R_Squared	Alpha	Lambda
s=1e-05	Elastic Net (Tuned, lambda.min)	0.0770	0.0573	0.9717	0.2	0.000010
s=0.00179901	Elastic Net (Tuned, lambda.1se)	0.0771	0.0572	0.9717	0.2	0.001799

6. Comparison of Model Performance

Compare all 7 models on the **test set**.

```r
# Combine all 7 metric data frames
ols_metrics_full <- ols_metrics %>%
  mutate(Alpha = NA, Lambda = NA) # Add NA columns for binding

final_metrics <- bind_rows(
  ols_metrics_full,
  ridge_metrics_combined,
  lasso_metrics_combined,
  enet_metrics_combined
)

kable(final_metrics,
      caption = "Full Model Performance Comparison on Test Data",
      digits = c(NA, 4, 4, 4, 2, 6),
      col.names = c("Model", "RMSE", "MAE", "R-Squared", "Alpha", "Lambda"))
```

Table 5: Full Model Performance Comparison on Test Data

	Model	RMSE	MAE	R-Squared	Alpha	Lambda
...1	OLS (BIC-formula)	0.0770	0.0573	0.9717	NA	NA
s=1e-05...2	Ridge (lambda.min)	0.0770	0.0573	0.9717	0.0	0.000010
s=0.001404918	Ridge (lambda.1se)	0.0771	0.0572	0.9717	0.0	0.001405
s=1e-05...4	Lasso (lambda.min)	0.0770	0.0573	0.9717	1.0	0.000010
s=0.0009695656	Lasso (lambda.1se)	0.0771	0.0574	0.9717	1.0	0.000970
s=1e-05...6	Elastic Net (Tuned, lambda.min)	0.0770	0.0573	0.9717	0.2	0.000010

	Model	RMSE	MAE	R-Squared	Alpha	Lambda
s=0.00179901	Elastic Net (Tuned, lambda.1se)	0.0771	0.0572	0.9717	0.2	0.001799

Performance Analysis:

The key finding from Table 5 is that **all 7 models**—the OLS baseline, both Ridge models, both Lasso models, and both tuned Elastic Net models—achieved **practically identical, top-tier predictive accuracy** ($R^2 \approx 0.9717$).

This fundamentally changes our analysis: 1. **OLS Model is Robust:** The OLS model was not suffering from overfitting; its performance on the test set is just as high as the regularized models. 2. **`lambda.1se` Rule Costs Nothing:** Using the simpler `lambda.1se` rule (instead of `lambda.min`) resulted in **no loss of predictive power** on this dataset. 3. **The Choice is Simplicity, Not Accuracy:** Since all models are equally accurate, the choice of the "best" model now depends entirely on interpretability and parsimony.

7. Model Interpretation (Coefficient Comparison)

The real difference lies in **interpretation and simplicity**. Table 6 compares the full OLS model with the two Lasso alternatives.

```r
# Helper function to get coefficients
get_coefs <- function(model, s, col_name) {
  coef_raw <- coef(model, s = s)
  df <- data.frame(
    term = rownames(coef_raw),
    value = as.numeric(coef_raw)
  )
  names(df)[names(df) == "value"] <- col_name
  return(df)
}

# Get coefficients for ALL 7 models
```

```r
ols_coefs <- broom::tidy(ols_model) %>%
  dplyr::select(term, OLS = estimate)
ridge_min_coefs <- get_coefs(cv_ridge,
cv_ridge$lambda.min, "Ridge (min)")
ridge_1se_coefs <- get_coefs(cv_ridge,
cv_ridge$lambda.1se, "Ridge (1se)")
lasso_min_coefs <- get_coefs(cv_lasso,
cv_lasso$lambda.min, "Lasso (min)")
lasso_1se_coefs <- get_coefs(cv_lasso,
cv_lasso$lambda.1se, "Lasso (1se)")
enet_min_coefs <- get_coefs(best_model,
best_model$lambda.min, "ENet (min)")
enet_1se_coefs <- get_coefs(best_model,
best_model$lambda.1se, "ENet (1se)")


# Join them all together
coef_list <- list(ols_coefs,
                  ridge_min_coefs, ridge_1se_coefs,
                  lasso_min_coefs, lasso_1se_coefs,
                  enet_min_coefs, enet_1se_coefs)
comparison_table <- reduce(coef_list, full_join, by = "term")

comparison_table[is.na(comparison_table)] <- 0

# Format and display
comparison_table <- comparison_table %>%
  mutate(across(where(is.numeric), ~ round(., 5))) %>%
  arrange(term != "(Intercept)")

kable(comparison_table,
      caption = "Full Coefficient Comparison: OLS vs. All Regularized Models",
      digits = 5,
      row.names = FALSE)
```

Table 6: Full Coefficient Comparison: OLS vs. All Regularized Models

term	OLS	Ridge (min)	Ridge (1se)	Lasso (min)	Lasso (1se)	ENet (min)	ENet (1se)
(Intercept)	1.18078	1.18139	1.22223	1.18074	1.18249	1.18126	1.22611
log_distance	0.01089	0.01109	0.02365	0.01082	0.00839	0.01104	0.02325
log_fare	0.65199	0.65169	0.63208	0.65210	0.65647	0.65177	0.63294
log_tip	0.14348	0.14349	0.14391	0.14343	0.13839	0.14348	0.14211
log_tolls	0.12017	0.12017	0.12032	0.12017	0.11928	0.12017	0.11994
passenger_count	0.00108	0.00108	0.00124	0.00106	0.00000	0.00108	0.00071
payment_type2	0.02746	0.02747	0.02846	0.02734	0.01729	0.02745	0.02485
pickup_hour	0.00244	0.00244	0.00246	0.00244	0.00232	0.00244	0.00241
day_of_weekTuesday	0.00630	0.00632	0.00721	0.00616	0.00227	0.00629	0.00455
day_of_weekWednesday	0.00104	0.00106	0.00192	0.00090	0.00000	0.00103	0.00000
day_of_weekThursday	0.00287	0.00289	0.00404	0.00273	0.00000	0.00286	0.00155
day_of_weekFriday	0.00341	0.00342	0.00422	0.00327	0.00000	0.00339	0.00170
day_of_weekSaturday	-0.02735	-0.02734	-0.02694	-0.02743	-0.02659	-0.02736	-0.02780
day_of_weekSunday	-0.01686	-0.01687	-0.01741	-0.01694	-0.01577	-0.01688	-0.01819
VendorID	-0.00037	-0.00036	0.00004	-0.00035	0.00000	-0.00036	0.00000

Interpretation:

1. **OLS (Baseline):** The OLS model is valid. The coefficient for `log_distance` is positive (`0.01089`), as expected. The model uses **all 14 predictors**.
2. **Lasso (lambda.min):** This accuracy-focused model is **almost identical to OLS**. It also uses **all 14 predictors** and only shrinks their coefficients trivially. This confirms

that with a very small `lambda` (`0.00001`), Lasso converges to the OLS solution.

3. **Lasso (lambda.1se) (The Key Insight):** This model provides the **best solution**. It achieves the *same* predictive accuracy ($R^2 = 0.9717$) but is significantly **simpler**. It has **zeroed out 5 predictors**: `passenger_count`, `day_of_weekWednesday`, `day_of_weekThursday`, `day_of_weekFriday`, and `VendorID`.

---

8. The Best Model

Based on this comprehensive analysis, the **best overall model is the Lasso (lambda.1se) model.**

Here is the reasoning:

1. **OLS (BIC-formula): Rejected.** High accuracy but uninterpretable coefficients.
2. **Ridge (min/1se): Rejected.** Lower predictive accuracy than the other models.
3. **Elastic Net (Tuned):** A **strong contender.** It achieves the same top-tier accuracy (0.9717). However, the tuning process is complex, and the final model is not as simple (parsimonious) as the Lasso model.
4. **Lasso (lambda.min):** A **good model.** It achieves top-tier accuracy and fixes multicollinearity.
5. **Lasso (lambda.1se): The WINNER.** It has the **exact same R-Squared** as `Lasso (lambda.min)` and OLS, but it is **simpler and more interpretable**. It provides the best possible balance of all three goals: **maximum accuracy, high interpretability, and parsimony.**

9. Question for Peer Feedback

> My analysis shows that the `Lasso (lambda.min)` and `Lasso (lambda.1se)` models have virtually identical R^2 (0.9717). The `lambda.1se` model, however, is more parsimonious (it zeroed out several predictors that `lambda.min` kept). Does this provide a clear justification for choosing `Lasso (lambda.1se)` for interpretability, as recommended in the lecture, especially since there is no "accuracy vs. simplicity" trade-off in this case?

10. Answer to Exam Question

Question: An analyst has a high-dimensional dataset ($p > n$) and suspects many of the predictors are irrelevant to Y. Which regularization method (Ridge or Lasso) is better suited for variable selection, and why?

Answer: Lasso (`alpha = 1`) should be chosen. Lasso uses an L1 penalty ($\lambda \sum |\beta_j|$), which, due to its "diamond-shaped" constraint region, can force coefficient estimates to be **exactly zero**. This effectively performs automatic variable selection. In contrast, Ridge (`alpha = 0`) uses an L2 penalty ($\lambda \sum \beta_j^2$), which only shrinks coefficients *towards* zero but never makes them exactly zero, thus keeping all predictors in the final model.