Machine Learning with R and Python

Don Hale

2025-09-22

Table of contents

About

This book is a companion to the Machine Learning Class. It will be a repository of R and Python Code. The chapters will be updated as we progress through the class.

1 Introduction

1.1 Introduction to Machine Learning

Machine Learning (ML) is a branch of artificial intelligence (AI) that focuses on building systems that **learn patterns from data** and make predictions or decisions without being explicitly programmed to perform specific tasks. Instead of writing rules by hand, a machine learning algorithm "learns" from examples in the data.

1.2 What is Machine Learning?

Formally, machine learning is the process of using data to **train a model** to make accurate predictions or decisions. The model identifies patterns in the training data and generalizes these patterns to new, unseen data.

ML can be thought of as:

- **Predictive modeling** – learning a function that maps inputs (features) to outputs (targets).
- **Automated decision-making** – systems that improve their performance over time without explicit reprogramming.

1.3 Major Types of Machine Learning

Machine learning is commonly categorized into several types based on the **nature of the task** and **availability of labeled data**.

1.3.1 1. Supervised Learning

- Uses **labeled data**, meaning that each training example has an input and a known output.
- The goal is to **learn a function** that maps inputs to outputs accurately.
- Typical tasks:
  - **Regression**: Predict a continuous variable (e.g., house prices).
  - **Classification**: Predict a categorical variable (e.g., spam vs. non-spam email).

1.3.2 2. Unsupervised Learning

- Works with **unlabeled data**, where the output is unknown.
- The goal is to **discover patterns or structure** in the data.
- Typical tasks:
  - **Clustering**: Group similar observations together (e.g., customer segmentation).
  - **Dimensionality reduction**: Reduce the number of features while retaining important information (e.g., PCA).

1.3.3 3. Semi-Supervised Learning

- Combines a small amount of labeled data with a large amount of unlabeled data.
- Useful when labeling data is expensive or time-consuming.

1.3.4 4. Reinforcement Learning

- Focuses on **learning through trial and error**.
- An agent learns to take actions in an environment to **maximize cumulative reward**.
- Examples: Robotics, game AI, recommendation systems.

1.4 Objectives of Machine Learning

The main objective of machine learning is to **build models that generalize well** from historical data to make accurate predictions or decisions on **new, unseen data**. Key considerations include:

1. **Accuracy** – How well does the model predict outcomes?

2. **Generalization** – Does the model perform well on unseen data, or is it overfitting the training data?
3. **Interpretability** – Can humans understand how the model makes predictions?

1.5 Bias-Variance Tradeoff

A fundamental concept in machine learning is the **bias-variance tradeoff**, which explains the balance between:

- **Bias**: Error due to overly simplistic models that **underfit** the data.
- **Variance**: Error due to overly complex models that **overfit** the training data.

The goal is to find a model that **minimizes the total prediction error**:

[Total Error = Bias^2 + Variance + Irreducible Error]

- **Underfitting** → High bias, low variance

- **Overfitting** → Low bias, high variance

The optimal model balances bias and variance for the best predictive performance.

1.6 The Machine Learning Process

A typical workflow in machine learning consists of the following steps:

1. **Data Collection**
    - Gather data from experiments, databases, or external sources.
2. **Data Preprocessing**
    - Handle missing values, outliers, and feature engineering.
3. **Model Selection**
    - Choose an appropriate algorithm (e.g., linear regression, random forest, neural networks).
4. **Model Training**

- Fit the model to the training data.

5. **Model Evaluation**
    - Assess performance using metrics such as RMSE, accuracy, precision, recall, or AUC.
6. **Model Tuning**
    - Adjust hyperparameters to improve performance.
7. **Model Deployment**
    - Use the trained model to make predictions on new data.
8. **Monitoring and Maintenance**
    - Continuously track model performance and retrain if necessary.

1.7 Summary

Machine learning allows us to build predictive models that **learn from data**. Understanding the different types of ML, their objectives, and the tradeoffs between bias and variance is critical to applying ML effectively. The remainder of this book explores practical algorithms, their implementation in **R and Python**, and strategies for model selection and interpretation.

2 Model Selection and Regularization - R Version

2.1 Overview

In this section, we'll use the **Ames Housing** dataset to demonstrate model selection and regularization.

We'll cover:

1. Splitting data into training and testing sets
2. Performing **stepwise regression** with cross-validation
3. Running **Ridge** and **Lasso** regression, and visualizing how affects RMSE and model complexity
4. Exploring **Elastic Net**, varying to balance Ridge and Lasso
5. Comparing all models on the test set

2.2 Step 0: Setup and Data Preparation

We'll use a reduced set of variables for speed and clarity.

```r
# Suppress messages and warnings globally
knitr::opts_chunk$set(echo = TRUE, message = FALSE, warning = FALSE)

library(tidyverse)
library(caret)
library(glmnet)
library(AmesHousing)
```

```r
library(GGally)

set.seed(123)

# Load data and split into 70% training, 30% testing
ames <- make_ordinal_ames() %>% mutate(id = row_number())
train <- ames %>% sample_frac(0.7)
test  <- anti_join(ames, train, by = "id")

# Select a manageable subset of predictors
keep <- c("Sale_Price", "Bedroom_AbvGr", "Year_Built", "Mo_Sold", "Lot_Area",
          "Street", "Central_Air", "First_Flr_SF", "Second_Flr_SF", "Full_Bath",
          "Half_Bath", "Fireplaces", "Garage_Area", "Gr_Liv_Area", "TotRms_AbvGrd")

train <- train %>% select(all_of(keep))
test  <- test  %>% select(all_of(keep))

# Convert categorical variables to factors
train <- train %>% mutate(across(c(Street, Central_Air), as.factor))
test  <- test %>% mutate(across(c(Street, Central_Air), as.factor))
```

2.3 Step 1: Stepwise Regression with 10-Fold Cross Validation

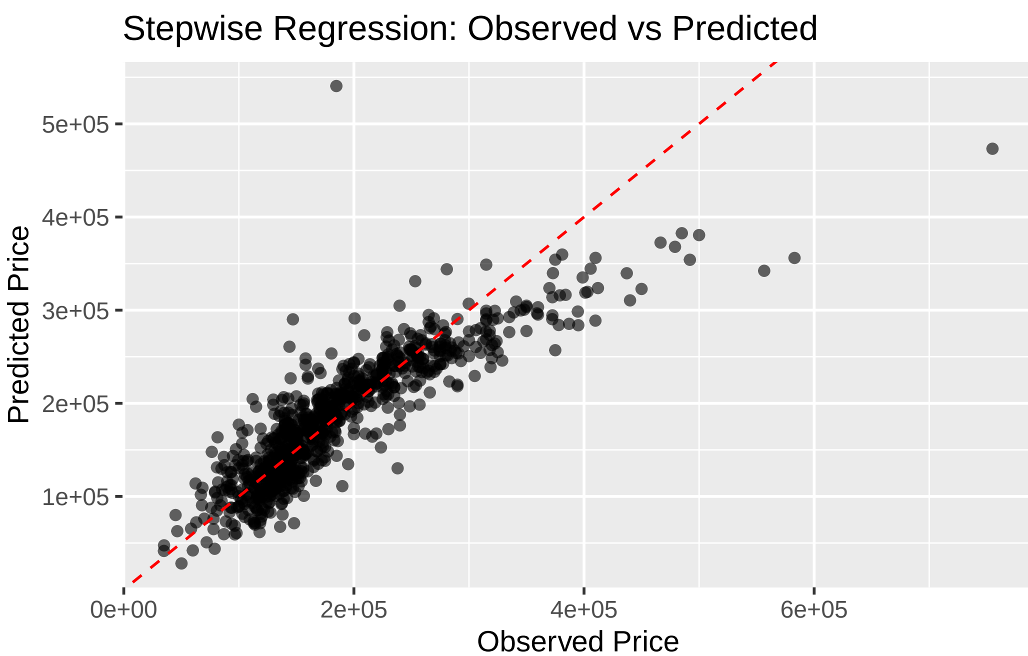
We'll use backward stepwise regression as a traditional benchmark.

```r
ctrl <- trainControl(method = "cv", number = 10)

step_model <- train(
  Sale_Price ~ ., data = train,
  method = "lmStepAIC",
  trControl = ctrl,
  trace = FALSE,
  direction = "backward"
)

# Evaluate on the test set
step_pred <- predict(step_model, newdata = test)
```

```r
step_perf <- postResample(step_pred, test$Sale_Price)
step_perf
```

```
        RMSE     Rsquared          MAE
3.820776e+04 7.670269e-01 2.613098e+04
```

```r
# Visualize predicted vs observed
ggplot(data.frame(obs = test$Sale_Price, pred = step_pred), aes(obs, pred)) +
  geom_point(alpha = 0.6) +
  geom_abline(linetype = "dashed", color = "red") +
  labs(title = "Stepwise Regression: Observed vs Predicted",
       x = "Observed Price", y = "Predicted Price")
```



Note:
Stepwise regression is intuitive and fast, but it can be unstable.

Stepwise regression (forward, backward, or both) selects predictors one at a time based on how much they improve a criterion (like AIC or adjusted R^2).

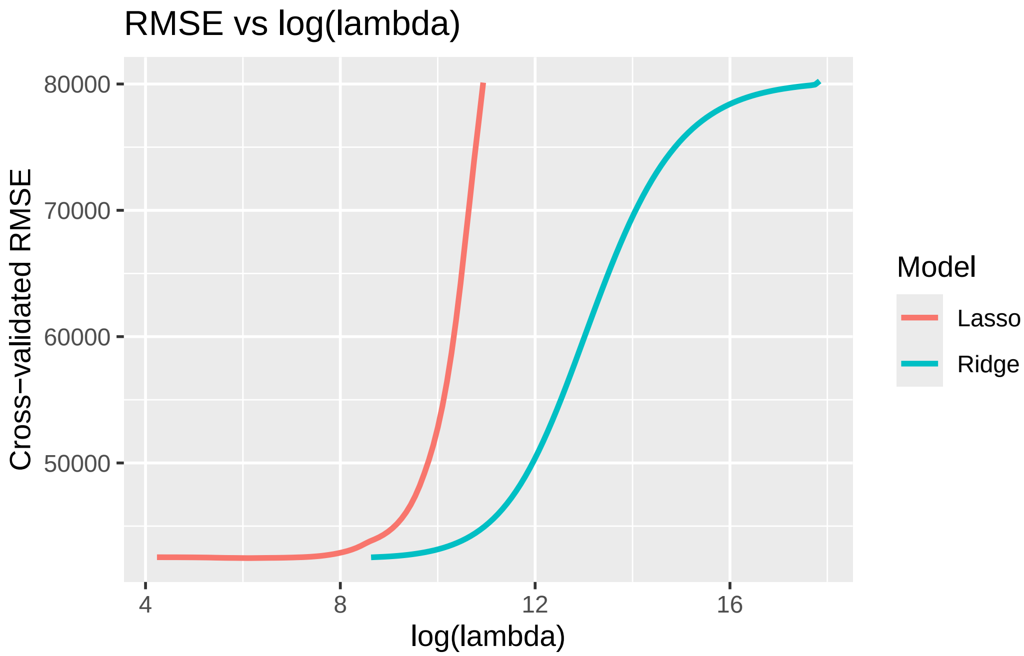
2.4 Step 2: Ridge and Lasso Regression

Ridge and Lasso both shrink coefficients, but in different ways: - Ridge shrinks all coefficients toward zero. - Lasso can set some coefficients *exactly* to zero, performing variable selection.

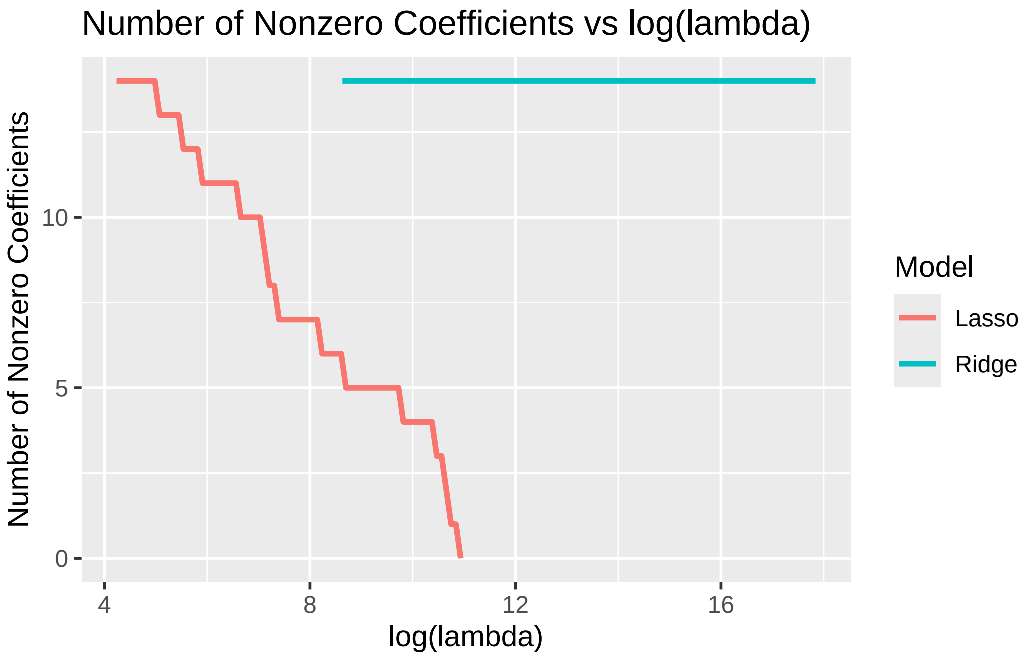
```
x_train <- model.matrix(Sale_Price ~ ., train)[, -1]
y_train <- train$Sale_Price
x_test  <- model.matrix(Sale_Price ~ ., test)[, -1]
y_test  <- test$Sale_Price

set.seed(123)
cv_ridge <- cv.glmnet(x_train, y_train, alpha = 0, nfolds = 10)
cv_lasso <- cv.glmnet(x_train, y_train, alpha = 1, nfolds = 10)
```

2.4.1 Visualizing RMSE and Model Complexity

```
build_path_df <- function(cvfit, label) {
  fit <- cvfit$glmnet.fit
  tibble(
    lambda = fit$lambda,
    log_lambda = log(fit$lambda),
    RMSE = sqrt(cvfit$cvm),
    nonzero = colSums(abs(fit$beta) > 0),
    Model = label
  )
}

ridge_df <- build_path_df(cv_ridge, "Ridge")
lasso_df <- build_path_df(cv_lasso, "Lasso")
df <- bind_rows(ridge_df, lasso_df)

ggplot(df, aes(log_lambda, RMSE, color = Model)) +
  geom_line(size = 1) +
  labs(title = "RMSE vs log(lambda)", y = "Cross-validated RMSE", x = "log(lambda)")
```




```
ggplot(df, aes(log_lambda, nonzero, color = Model)) +
  geom_line(size = 1) +
  labs(title = "Number of Nonzero Coefficients vs log(lambda)",
       y = "Number of Nonzero Coefficients", x = "log(lambda)")
```



Note:
- Increasing increases regularization.
- Ridge never eliminates variables, Lasso can.
- There's a sweet spot where RMSE is minimized.

2.5 Step 3: Elastic Net (Balancing Ridge and Lasso)

Elastic Net introduces to control the mix between Ridge (= 0) and Lasso (= 1).

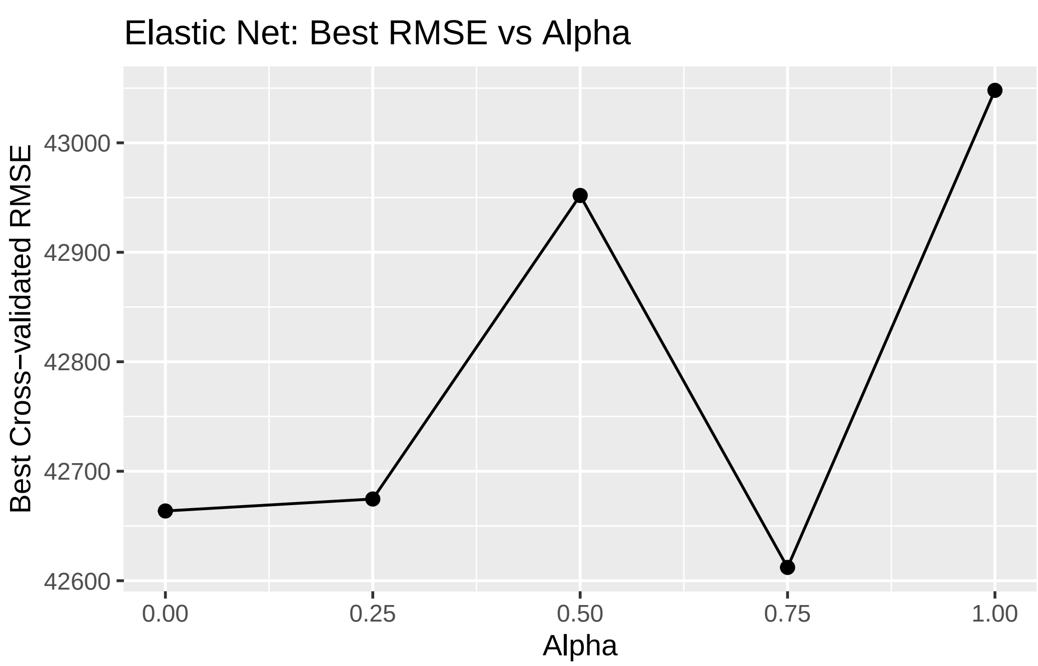
```r
alpha_grid <- seq(0, 1, by = 0.25)

elastic_results <- map_df(alpha_grid, function(a) {
  cv_fit <- cv.glmnet(x_train, y_train, alpha = a, nfolds = 10)
  tibble(
    alpha = a,
    best_lambda = cv_fit$lambda.min,
    best_RMSE = sqrt(min(cv_fit$cvm)),
    nonzero = sum(abs(coef(cv_fit, s = "lambda.min")[-1]) > 0)
  )
```

```
})

elastic_results
```

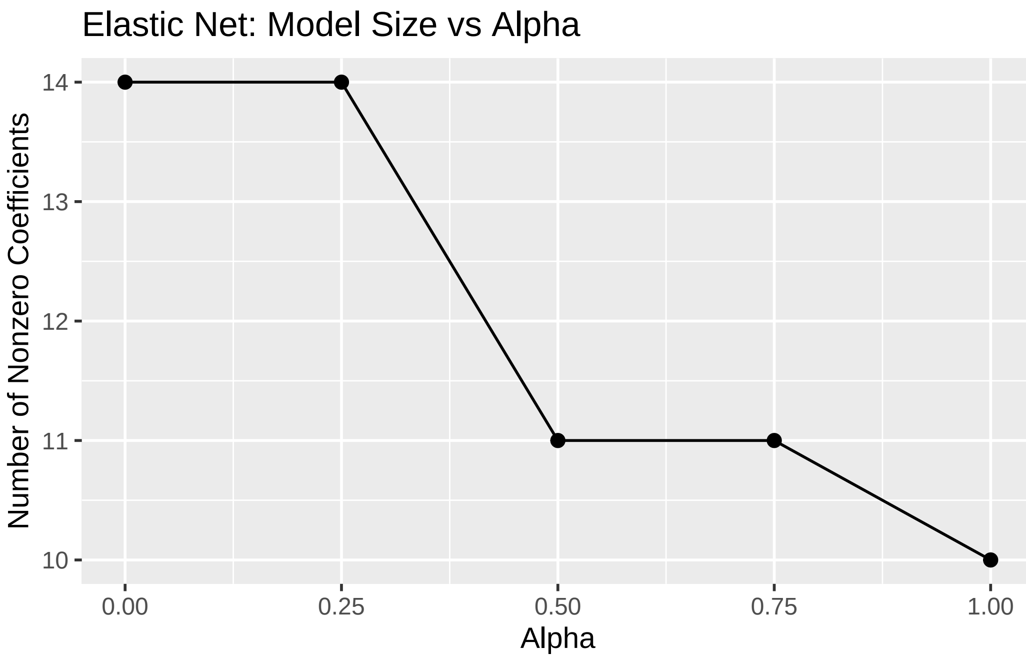
```
# A tibble: 5 x 4
  alpha best_lambda best_RMSE nonzero
  <dbl>       <dbl>     <dbl>   <int>
1  0          5603.    42664.      14
2  0.25       1224.    42675.      14
3  0.5        1552.    42952.      11
4  0.75        650.    42612.      11
5  1          1126.    43048.      10
```

2.5.1 Visualizing   Effects

```
ggplot(elastic_results, aes(alpha, best_RMSE)) +
  geom_line() + geom_point(size = 2) +
  labs(title = "Elastic Net: Best RMSE vs Alpha",
       y = "Best Cross-validated RMSE", x = "Alpha")
```



```r
ggplot(elastic_results, aes(alpha, nonzero)) +
  geom_line() + geom_point(size = 2) +
  labs(title = "Elastic Net: Model Size vs Alpha",
       y = "Number of Nonzero Coefficients", x = "Alpha")
```



Note:
Elastic Net combines the strengths of Ridge and Lasso: - Ridge for stability. - Lasso for sparsity. - Often performs best at intermediate values.

2.6 Step 4: Comparing All Models on the Test Set

We'll compare Stepwise, Ridge, Lasso, and the best Elastic Net.

```r
ridge_pred <- predict(cv_ridge, newx = x_test, s = "lambda.min")
lasso_pred <- predict(cv_lasso, newx = x_test, s = "lambda.min")

best_alpha <- elastic_results %>% arrange(best_RMSE) %>% slice(1) %>% pull(alpha)
cv_en_best <- cv.glmnet(x_train, y_train, alpha = best_alpha, nfolds = 10)
elastic_pred <- predict(cv_en_best, newx = x_test, s = "lambda.min")
```

```
compare_tbl <- bind_rows(
  Stepwise = as_tibble_row(postResample(step_pred, test$Sale_Price)),
  Ridge = as_tibble_row(postResample(ridge_pred, y_test)),
  Lasso = as_tibble_row(postResample(lasso_pred, y_test)),
  Elastic = as_tibble_row(postResample(elastic_pred, y_test)),
  .id = "Model"
)

colnames(compare_tbl) <- c("Model", "RMSE", "Rsquared", "MAE")
compare_tbl <- compare_tbl %>% arrange(RMSE)
compare_tbl
```

```
# A tibble: 4 x 4
  Model       RMSE Rsquared    MAE
  <chr>      <dbl>    <dbl>  <dbl>
1 Stepwise 38208.    0.767 26131.
2 Lasso    38297.    0.767 26045.
3 Elastic  38301.    0.767 26039.
4 Ridge    38412.    0.766 26022.
```

2.7 Key Takeaways

- Stepwise selection is simple but can lead to unstable models.
- Ridge adds stability via coefficient shrinkage.
- Lasso enforces sparsity by removing weak predictors.
- Elastic Net balances both effects.
- Regularization often produces models that generalize better than purely stepwise ones.

3 Model Selection and Regularization - Python Version

3.1 Overview

This section parallels the R version of Model Selection and Regularization.
We will:

1. Load the Ames Housing data
2. Split into training and test sets
3. Perform Ridge, Lasso, and Elastic Net regressions with 10-fold CV
4. Visualize RMSE and coefficient shrinkage patterns
5. Compare test set performance

3.2 Step 0: Setup and Data

We'll use scikit-learn for modeling.
If the CSVs `ames_training.csv` and `ames_testing.csv` are not present, you can load Ames Housing via the `fetch_openml` function.

```
import warnings
warnings.filterwarnings("ignore")

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_openml
from sklearn.model_selection import train_test_split, KFold, cross_val_score
```

```python
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import RidgeCV, LassoCV, ElasticNetCV
from sklearn.metrics import mean_squared_error

# Load Ames dataset directly
ames = fetch_openml(name="house_prices", as_frame=True)
df = ames.frame

# Subset variables to match the R version
keep = [
    "SalePrice","BedroomAbvGr","YearBuilt","MoSold","LotArea","Street","CentralAir",
    "1stFlrSF","2ndFlrSF","FullBath","HalfBath","Fireplaces","GarageArea",
    "GrLivArea","TotRmsAbvGrd"
]
df = df[keep].copy()

# Clean up names to match Python variable rules
df.columns = ["Sale_Price","Bedroom_AbvGr","Year_Built","Mo_Sold","Lot_Area","Street",
              "Central_Air","First_Flr_SF","Second_Flr_SF","Full_Bath","Half_Bath",
              "Fireplaces","Garage_Area","Gr_Liv_Area","TotRms_AbvGrd"]

# Drop missing rows
df = df.dropna()

# Train/test split (70/30)
train, test = train_test_split(df, test_size=0.3, random_state=123)

# Identify predictors and target
y_train = train["Sale_Price"]
y_test  = test["Sale_Price"]
X_train = train.drop(columns="Sale_Price")
X_test  = test.drop(columns="Sale_Price")

cat_vars = ["Street","Central_Air"]
num_vars = [c for c in X_train.columns if c not in cat_vars]

# Preprocess: scale numeric, one-hot encode categorical
preprocessor = ColumnTransformer([
    ("num", StandardScaler(), num_vars),
    ("cat", OneHotEncoder(drop="first"), cat_vars)
```

```
])
```

3.3 Step 1: Ridge Regression

We'll use cross-validation to tune (alpha).
Then we visualize RMSE vs log(lambda) and the number of nonzero coefficients.

```
alphas = np.logspace(4, -4, 80)
ridge = Pipeline([
    ("prep", preprocessor),
    ("model", RidgeCV(alphas=alphas, scoring="neg_mean_squared_error", cv=10))
])
ridge.fit(X_train, y_train)

ridge_best = ridge.named_steps["model"].alpha_
print("Best Ridge alpha:", ridge_best)

# Evaluate RMSE on test set
ridge_pred = ridge.predict(X_test)
ridge_rmse = np.sqrt(mean_squared_error(y_test, ridge_pred))
print("Test RMSE (Ridge):", ridge_rmse)
```

```
Best Ridge alpha: 94.33732216299774
Test RMSE (Ridge): 37055.84511170759
```

3.4 Step 2: Lasso Regression

Lasso adds variable selection — some coefficients go exactly to zero.

```
lasso = Pipeline([
    ("prep", preprocessor),
    ("model", LassoCV(alphas=alphas, cv=10, max_iter=20000))
])
lasso.fit(X_train, y_train)
```

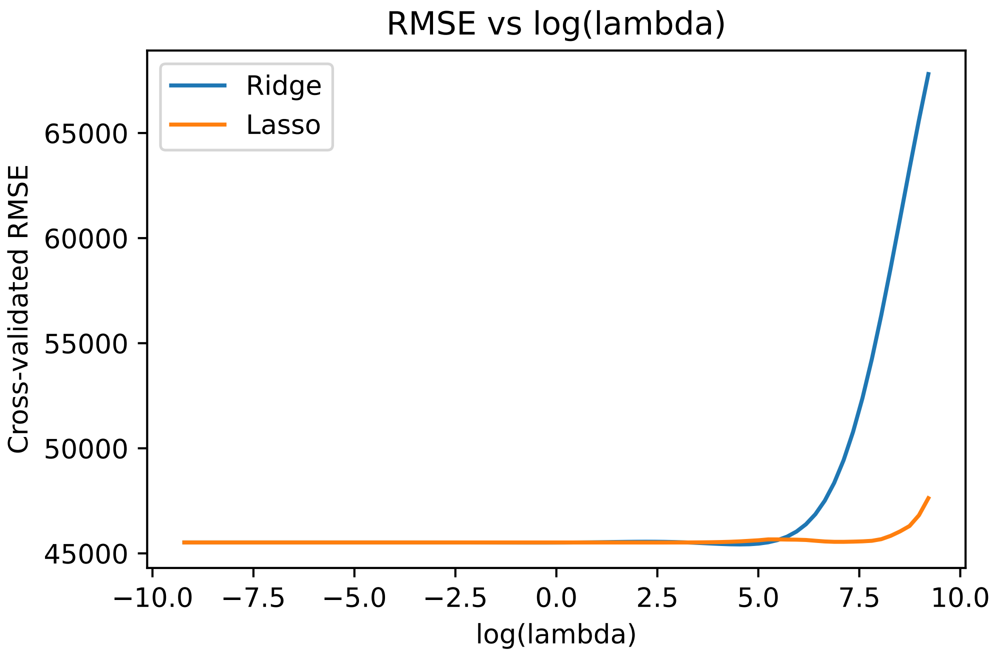
```
lasso_best = lasso.named_steps["model"].alpha_
print("Best Lasso alpha:", lasso_best)

lasso_pred = lasso.predict(X_test)
lasso_rmse = np.sqrt(mean_squared_error(y_test, lasso_pred))
print("Test RMSE (Lasso):", lasso_rmse)
```

```
Best Lasso alpha: 11.568875283162821
Test RMSE (Lasso): 37074.74978063246
```

3.5 Step 3: Visualize Coefficient Shrinkage and RMSE

Let's visualize how RMSE and model sparsity change with .

```
# Compute RMSE path manually
def cv_rmse(model_class, X, y, alphas):
    rmse = []
    for a in alphas:
        model = Pipeline([
            ("prep", preprocessor),
            ("model", model_class(alpha=a, max_iter=20000))
        ])
        scores = cross_val_score(model, X, y, scoring="neg_mean_squared_error", cv=KFold(10,
        rmse.append(np.sqrt(-scores.mean()))
    return np.array(rmse)

from sklearn.linear_model import Ridge, Lasso
ridge_rmse_path = cv_rmse(Ridge, X_train, y_train, alphas)
lasso_rmse_path = cv_rmse(Lasso, X_train, y_train, alphas)

plt.figure()
plt.plot(np.log(alphas), ridge_rmse_path, label="Ridge")
plt.plot(np.log(alphas), lasso_rmse_path, label="Lasso")
plt.xlabel("log(lambda)")
plt.ylabel("Cross-validated RMSE")
plt.title("RMSE vs log(lambda)")
plt.legend()
plt.show()
```



3.6 Step 4: Elastic Net

Elastic Net blends Ridge and Lasso.
We'll vary (the mix) and visualize the tradeoff.

```
l1_ratios = np.linspace(0, 1, 6)
en_cv = Pipeline([
    ("prep", preprocessor),
    ("model", ElasticNetCV(l1_ratio=l1_ratios, alphas=alphas, cv=10, max_iter=50000))
])
en_cv.fit(X_train, y_train)

print("Best Elastic Net alpha:", en_cv.named_steps["model"].alpha_)
print("Best Elastic Net l1_ratio:", en_cv.named_steps["model"].l1_ratio_)

en_pred = en_cv.predict(X_test)
en_rmse = np.sqrt(mean_squared_error(y_test, en_pred))
print("Test RMSE (Elastic Net):", en_rmse)
```

```
Best Elastic Net alpha: 0.55825862688627
Best Elastic Net l1_ratio: 0.8
Test RMSE (Elastic Net): 37134.497858434275
```

3.7 Step 5: Compare Models

```
results = pd.DataFrame({
    "Model": ["Ridge", "Lasso", "Elastic Net"],
    "Test_RMSE": [ridge_rmse, lasso_rmse, en_rmse]
})
print(results.sort_values("Test_RMSE"))
```

```
         Model     Test_RMSE
0        Ridge  37055.845112
1        Lasso  37074.749781
2  Elastic Net  37134.497858
```

3.8 Summary

- Ridge stabilizes coefficients by shrinking them.
- Lasso enforces sparsity by zeroing weak predictors.
- Elastic Net blends both for balance.
- Regularization often beats pure stepwise regression on unseen data.

4 generalized-additive-models

5 tree-based-models

6 neural-network-models

7 naive-bayes-models

8 model-agnostic-interpretability

9 support-vector-machines