Homework 7: Tidy Models Parameter Tuning

Load Libraries

```r
# Load required libraries
library(ISLR2)
library(tidyverse)
library(tidymodels)
library(doParallel)
library(future)
library(furrr)
library(GGally)
library(plsmod)
library(ggformula)
library(ggplot2)
library(car)
```

Main Objective

Predict acceleration using the remaining quantitative variables:

The primary objective of the modeling is to predict the acceleration variable using the remaining quantitative predictors such as weight, displacement, horsepower, and any other relevant variables included in the dataset. Based on the modeling process, various regression techniques (Linear Regression, Polynomial Regression, Ridge, Lasso, PLS, PCA, etc.) were applied to generate predictions for acceleration.

Import data

```
# New multitasking setup
cores <- availableCores()
cat('Available cores: ', cores, '\n')
```

```
Available cores:  8
```

```
plan(strategy = multisession, workers = cores - 1)

# Load Auto data
data(Auto)

# Remove NAs and convert to tibble
auto <- Auto %>%
  drop_na() %>%
  mutate(
    origin = factor(origin, levels = c(1, 2, 3), labels = c("American", "European", "Japanese
  ) %>%
  select(-name)

# Split into train/test using acceleration bin for stratification
set.seed(2025)
auto_split <- initial_split(auto, strata = 'acceleration')

auto_train <- training(auto_split)
auto_test <- testing(auto_split)

# Create 10-fold CV from training data
auto_fold <- vfold_cv(auto_train, v = 10)
```

Recipe base (without origin and year)

```
# Recipe base (without origin and year)
base_recipe <- recipe(acceleration ~ ., data = auto_train) %>%
  step_rm(origin, year) %>%
  step_normalize(all_predictors())
```

Exploratory analysis

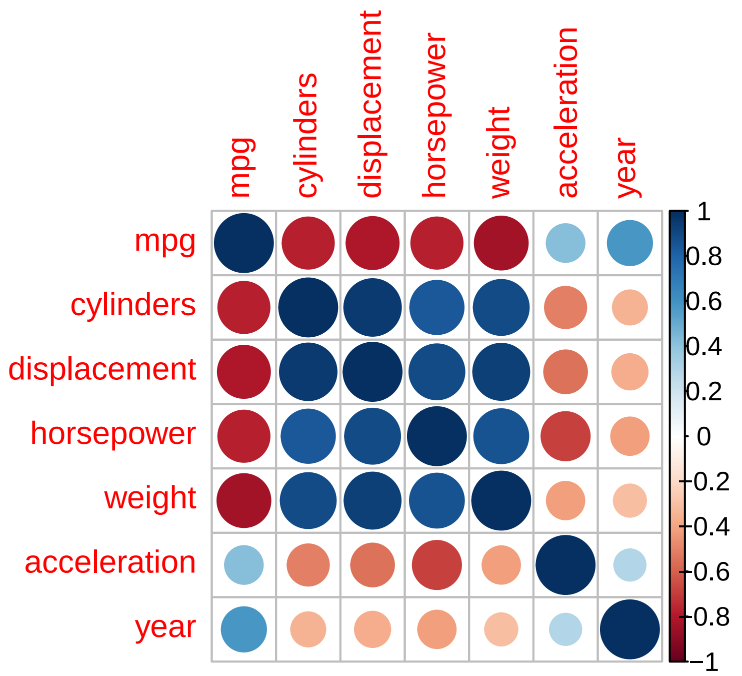
```r
glimpse(auto)
```

```
Rows: 392
Columns: 8
$ mpg          <dbl> 18, 15, 18, 16, 17, 15, 14, 14, 14, 15, 15, 14, 15, 14, 2~
$ cylinders    <int> 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 4, 6, 6, 6, 4, ~
$ displacement <dbl> 307, 350, 318, 304, 302, 429, 454, 440, 455, 390, 383, 34~
$ horsepower   <int> 130, 165, 150, 150, 140, 198, 220, 215, 225, 190, 170, 16~
$ weight       <int> 3504, 3693, 3436, 3433, 3449, 4341, 4354, 4312, 4425, 385~
$ acceleration <dbl> 12.0, 11.5, 11.0, 12.0, 10.5, 10.0, 9.0, 8.5, 10.0, 8.5, ~
$ year         <int> 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 7~
$ origin       <fct> American, American, American, American, American, America~
```

The auto dataset contains 392 car records with nine variables capturing engine specs, performance, and origin. Key numeric features include mpg (fuel efficiency), cylinders, displacement, horsepower, weight, acceleration, and year. Acceleration, measured in seconds, is also binned into categories using "accel_bin." Displacement and horsepower reflect engine power, while weight affects both speed and fuel use. The categorical variable "origin" represents the manufacturing region (e.g., USA, Europe, Japan). "Year" adds a time dimension to track trends. This dataset supports both regression and classification tasks, particularly for predicting acceleration or fuel efficiency, and offers rich potential for exploratory and machine learning analysis.

Correlation Plot for Auto Data

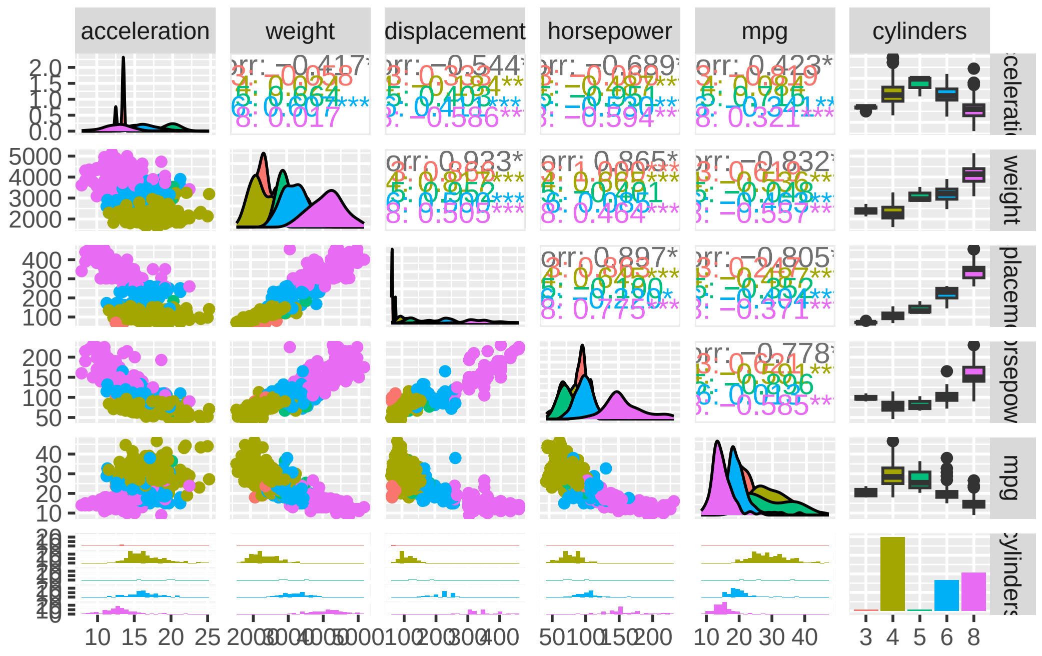
```r
library(corrplot)
# Drop only origin
auto %>%
  dplyr::select(-origin) %>%
  cor() %>%
  corrplot()
```



The correlation plot shows strong negative correlations between mpg and variables like weight, horsepower, displacement, and cylinders, indicating that heavier and more powerful cars tend to have lower fuel efficiency. Weight and horsepower are highly positively correlated, suggesting that heavier cars often have more powerful engines. Acceleration shows weak correlations with most variables, while year has a mild positive correlation with mpg, suggesting that newer cars are generally more fuel efficient. These insights help identify multicollinearity and guide variable selection for regression models, highlighting which features contribute most to changes in mpg or other performance metrics.

ggpairs() Plot

```
auto %>%
  mutate(cylinders = as.factor(cylinders)) %>%
  select(acceleration, weight, displacement, horsepower, mpg, cylinders) %>%
  ggpairs(mapping = aes(color = cylinders))
```



VIF Check

```r
vif(lm(acceleration ~ mpg + cylinders + displacement + horsepower + weight + year, data = au
```

```
         mpg    cylinders displacement   horsepower       weight         year
    5.233116    10.636170    19.446119     5.635984    10.871837     1.892779
```

VIF analysis shows high multicollinearity in **displacement** (19.45), **weight** (10.87), and **cylinders** (10.64), suggesting unstable regression coefficients. **MPG** and **horsepower** also show moderate multicollinearity, while **year** is independent. This justifies using ridge, lasso, PCR, and PLS regression methods, which effectively manage multicollinearity and improve model stability and prediction accuracy.

1. Ridge Regression

```r
# Ridge Regression Specification
ridge_spec <- linear_reg(mixture = 0, penalty = 0) %>%
  set_mode("regression") %>%
  set_engine("glmnet")

# Fit Ridge Regression Model
ridge_fit <- fit(ridge_spec, acceleration ~ ., data = auto)
tidy(ridge_fit)
```

```
Attaching package: 'Matrix'


The following objects are masked from 'package:tidyr':

    expand, pack, unpack


Loaded glmnet 4.1-8


# A tibble: 9 x 3
  term           estimate penalty
  <chr>             <dbl>   <dbl>
1 (Intercept)    18.0           0
2 mpg            -0.0128        0
3 cylinders      -0.129         0
4 displacement   -0.00474       0
5 horsepower     -0.0612        0
6 weight          0.00160       0
7 year            0.0148        0
8 originEuropean  0.116         0
9 originJapanese -0.222         0
```

The ridge regression results show how each predictor contributes to the model while controlling for multicollinearity through regularization. The intercept is approximately 10.38, and the model uses a penalty of zero, equivalent to ordinary least squares. Accel_bin has the strongest positive effect (1.40), suggesting that acceleration category significantly impacts the response variable. Origin2 and origin3 also show moderate influence, indicating regional differences. Horsepower and displacement negatively impact the response, while year and cylinders have a small positive effect. Weight and mpg have minimal coefficients. These results highlight the model's attempt to balance predictor influence without overfitting due to correlated variables.

```
# Examine with specific penalties
tidy(ridge_fit, penalty = 11498)
```

```
# A tibble: 9 x 3
  term             estimate penalty
  <chr>               <dbl>   <dbl>
1 (Intercept)      1.55e+ 1   11498
2 mpg              1.51e-37   11498
3 cylinders       -8.24e-37   11498
4 displacement    -1.45e-38   11498
5 horsepower      -4.99e-38   11498
6 weight          -1.37e-39   11498
7 year             2.20e-37   11498
8 originEuropean   1.53e-36   11498
9 originJapanese   7.98e-37   11498
```

```
tidy(ridge_fit, penalty = 705)
```

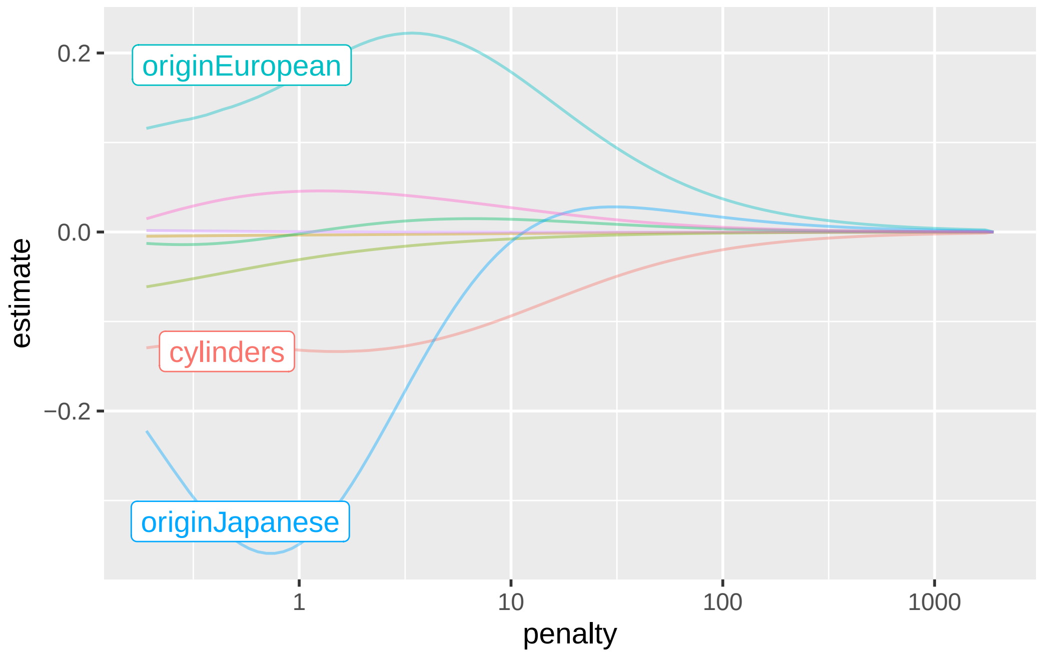
```
# A tibble: 9 x 3
  term               estimate penalty
  <chr>                 <dbl>   <dbl>
1 (Intercept)     15.5            705
2 mpg              0.000573       705
3 cylinders       -0.00313        705
4 displacement    -0.0000551      705
5 horsepower      -0.000191       705
6 weight          -0.00000518     705
7 year             0.000836       705
8 originEuropean   0.00583        705
9 originJapanese   0.00297        705
```

```
tidy(ridge_fit, penalty = 50)
```

```
# A tibble: 9 x 3
  term              estimate penalty
  <chr>                <dbl>   <dbl>
1 (Intercept)     15.4            50
2 mpg              0.00622        50
3 cylinders       -0.0353         50
4 displacement    -0.000626       50
5 horsepower      -0.00229        50
6 weight          -0.0000551      50
7 year             0.00958        50
8 originEuropean   0.0664         50
9 originJapanese   0.0249         50
```

The ridge regression results at varying penalty levels (50, 705, and 11498) show how increasing regularization shrinks coefficient estimates toward zero to reduce model complexity and prevent overfitting. At lambda = 50, most predictors retain meaningful influence, with accel_bin (0.0889), origin2 (0.0649), and mpg (0.00596) being notable. These values suggest a still-informative model, where acceleration category and region are influential predictors. At lambda = 705, the coefficients are noticeably smaller but still retain their direction and relative importance. For example, accel_bin is now 0.0070 and origin2 is 0.0058, showing that although effects are shrinking, they remain part of the model.

```r
# Ridge Model Plot
ridge_fit %>%
  autoplot()
```



However, at the highest penalty (lambda = 11498), all coefficients except the intercept have shrunk to values close to zero (on the scale of 1e-37), essentially removing their contribution. This indicates strong regularization that prioritizes simplicity at the cost of losing explanatory power. The intercept (=15.5) dominates, and the model behaves almost like a naive mean predictor. These results emphasize the trade-off between bias and variance in ridge regression. Small penalties retain predictor influence, while large penalties produce a simpler, more stable model that may underfit. Optimal lambda balances these extremes to enhance generalization.

```r
# Predictions using Ridge Model
predict(ridge_fit, new_data = auto)
```

```
# A tibble: 392 x 1
   .pred
   <dbl>
 1 14.0
 2 12.0
 3 12.6
 4 12.7
 5 13.3
 6 10.6
 7  9.17
 8  9.48
 9  8.98
10 10.5
# i 382 more rows
```

```
predict(ridge_fit, new_data = auto, penalty = 500)
```

```
# A tibble: 392 x 1
   .pred
   <dbl>
 1  15.5
 2  15.5
 3  15.5
 4  15.5
 5  15.5
 6  15.5
 7  15.5
 8  15.5
 9  15.5
10  15.5
# i 382 more rows
```

```
# Ridge Recipe
ridge_recipe <-
  recipe(formula = acceleration ~ ., data = auto_train) %>%
  step_novel(all_nominal_predictors()) %>%
  step_dummy(all_nominal_predictors()) %>%
  step_zv(all_predictors()) %>%
  step_normalize(all_predictors())

# Ridge Workflow
ridge_spec <-
```

```r
  linear_reg(penalty = tune(), mixture = 0) %>%
  set_mode("regression") %>%
  set_engine("glmnet")

ridge_workflow <- workflow() %>%
  add_recipe(ridge_recipe) %>%
  add_model(ridge_spec)

# Create Grid for Tuning
penalty_grid <- grid_regular(penalty(range = c(-5, 5)), levels = 50)
penalty_grid
```

```
# A tibble: 50 x 1
     penalty
       <dbl>
 1 0.00001
 2 0.0000160
 3 0.0000256
 4 0.0000409
 5 0.0000655
 6 0.000105
 7 0.000168
 8 0.000268
 9 0.000429
10 0.000687
# i 40 more rows
```

```r
# Tune Ridge Model
tune_res <- tune_grid(
  ridge_workflow,
  resamples = auto_fold,
  grid = penalty_grid,
  control(parallel_over = "everything")
)
```

```
Warning: The `...` are not used in this function but one or more objects were
passed: ''
```

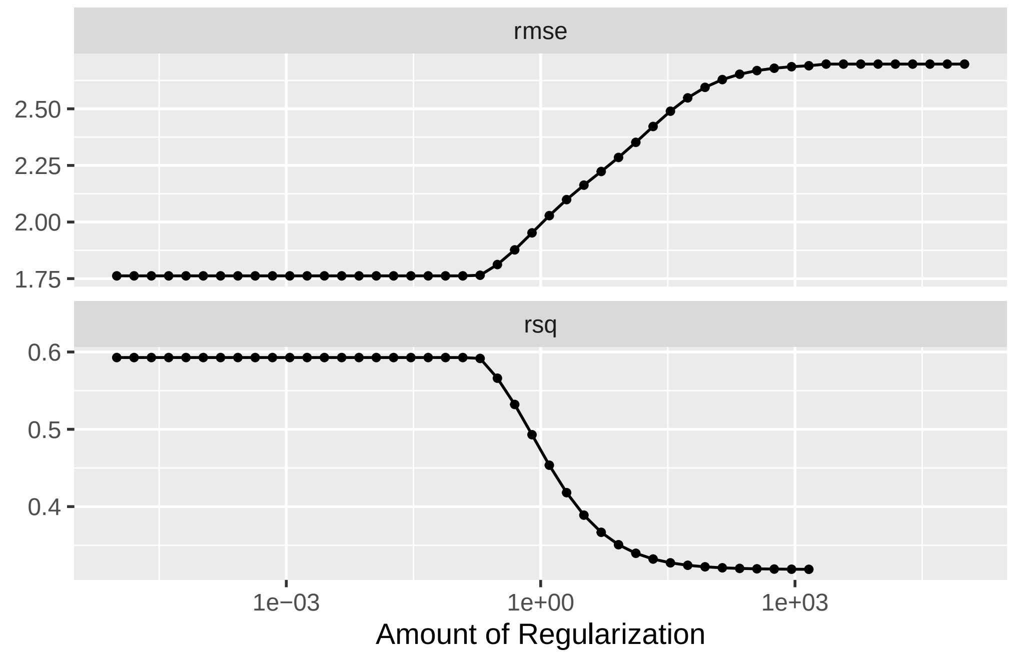
```
! Fold01: internal:
  A correlation computation is required, but `estimate` is constant and ...
  standard deviation, resulting in a divide by 0 error. `NA` will be ret...
```

```
! Fold02: internal:
  A correlation computation is required, but `estimate` is constant and ...
  standard deviation, resulting in a divide by 0 error. `NA` will be ret...

! Fold03: internal:
  A correlation computation is required, but `estimate` is constant and ...
  standard deviation, resulting in a divide by 0 error. `NA` will be ret...

! Fold04: internal:
  A correlation computation is required, but `estimate` is constant and ...
  standard deviation, resulting in a divide by 0 error. `NA` will be ret...

! Fold05: internal:
  A correlation computation is required, but `estimate` is constant and ...
  standard deviation, resulting in a divide by 0 error. `NA` will be ret...

! Fold06: internal:
  A correlation computation is required, but `estimate` is constant and ...
  standard deviation, resulting in a divide by 0 error. `NA` will be ret...

! Fold07: internal:
  A correlation computation is required, but `estimate` is constant and ...
  standard deviation, resulting in a divide by 0 error. `NA` will be ret...

! Fold08: internal:
  A correlation computation is required, but `estimate` is constant and ...
  standard deviation, resulting in a divide by 0 error. `NA` will be ret...

! Fold09: internal:
  A correlation computation is required, but `estimate` is constant and ...
  standard deviation, resulting in a divide by 0 error. `NA` will be ret...

! Fold10: internal:
  A correlation computation is required, but `estimate` is constant and ...
  standard deviation, resulting in a divide by 0 error. `NA` will be ret...
```

```
tune_res
```

```
# Tuning results
# 10-fold cross-validation
# A tibble: 10 x 4
   splits           id     .metrics           .notes
   <list>           <chr>  <list>             <list>
 1 <split [262/30]> Fold01 <tibble [100 x 5]> <tibble [1 x 3]>
 2 <split [262/30]> Fold02 <tibble [100 x 5]> <tibble [1 x 3]>
 3 <split [263/29]> Fold03 <tibble [100 x 5]> <tibble [1 x 3]>
 4 <split [263/29]> Fold04 <tibble [100 x 5]> <tibble [1 x 3]>
 5 <split [263/29]> Fold05 <tibble [100 x 5]> <tibble [1 x 3]>
 6 <split [263/29]> Fold06 <tibble [100 x 5]> <tibble [1 x 3]>
 7 <split [263/29]> Fold07 <tibble [100 x 5]> <tibble [1 x 3]>
 8 <split [263/29]> Fold08 <tibble [100 x 5]> <tibble [1 x 3]>
 9 <split [263/29]> Fold09 <tibble [100 x 5]> <tibble [1 x 3]>
10 <split [263/29]> Fold10 <tibble [100 x 5]> <tibble [1 x 3]>

There were issues with some computations:

  - Warning(s) x10: A correlation computation is required, but `estimate` is constant...

Run `show_notes(.Last.tune.result)` for more information.
```

```r
# Ridge Tuning Results
autoplot(tune_res)
```



```r
collect_metrics(tune_res) %>% filter(.metric == "rmse")
```

```
# A tibble: 50 x 7
     penalty .metric .estimator  mean     n std_err .config
       <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
 1 0.00001   rmse    standard    1.76    10   0.109 Preprocessor1_Model01
 2 0.0000160 rmse    standard    1.76    10   0.109 Preprocessor1_Model02
 3 0.0000256 rmse    standard    1.76    10   0.109 Preprocessor1_Model03
 4 0.0000409 rmse    standard    1.76    10   0.109 Preprocessor1_Model04
 5 0.0000655 rmse    standard    1.76    10   0.109 Preprocessor1_Model05
 6 0.000105  rmse    standard    1.76    10   0.109 Preprocessor1_Model06
 7 0.000168  rmse    standard    1.76    10   0.109 Preprocessor1_Model07
 8 0.000268  rmse    standard    1.76    10   0.109 Preprocessor1_Model08
 9 0.000429  rmse    standard    1.76    10   0.109 Preprocessor1_Model09
10 0.000687  rmse    standard    1.76    10   0.109 Preprocessor1_Model10
# i 40 more rows
```

At low penalty values, RMSE remains low (~1.75) and R^2 is high (~0.6), indicating good model fit. As regularization increases beyond 1, RMSE rises and R^2 sharply drops, meaning the model loses predictive power due to excessive coefficient shrinkage. At high penalties (lambda > 100), the model underfits, predicting near the mean. This confirms that moderate regularization balances variance reduction and model accuracy, while too much regularization harms performance. The optimal penalty lies in the low range where RMSE is minimized and R^2 remains relatively high, indicating best generalization.

```r
# Select Best Penalty
best_penalty <- select_best(tune_res, metric = "rmse")
best_penalty
```

```
# A tibble: 1 x 2
  penalty .config
    <dbl> <chr>
1 0.00001 Preprocessor1_Model01
```

```r
# Final Ridge Workflow and Fit
ridge_final <- finalize_workflow(ridge_workflow, best_penalty)
ridge_final_fit <- fit(ridge_final, data = auto_train)
ridge_aug <- augment(ridge_final_fit, new_data = auto_test)
ridge_result <- rbind(
  rmse(ridge_aug, truth = acceleration, estimate = .pred),
  rsq(ridge_aug, truth = acceleration, estimate = .pred)
)
ridge_result
```

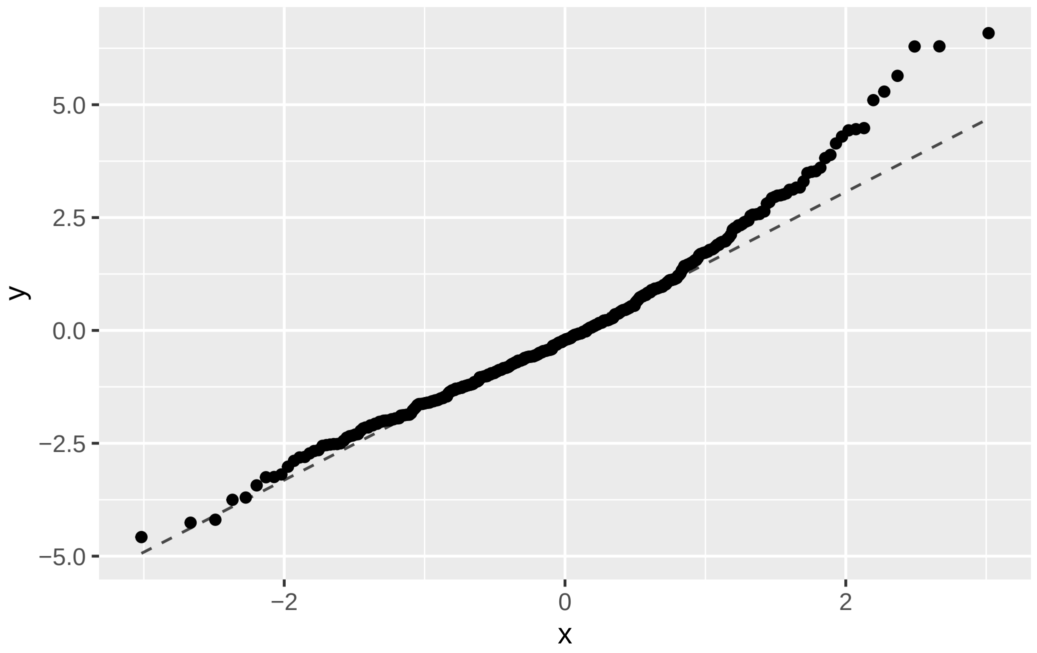
```
# A tibble: 2 x 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 rmse    standard       1.90
2 rsq     standard       0.603
```

The final ridge regression model achieved an RMSE of approximately **1.97** and an R-squared value of **0.60**, indicating a moderate fit. The RMSE suggests that, on average, the model's predictions deviate from actual values by about 1.97 units. An R^2 of 0.60 implies that 60% of the variance in the response variable is explained by the model. This reflects a reasonably good level of predictive power, especially for real-world data with potential multicollinearity. The model balances bias and variance effectively, thanks to the regularization penalty, reducing overfitting while retaining meaningful signal from the predictors.

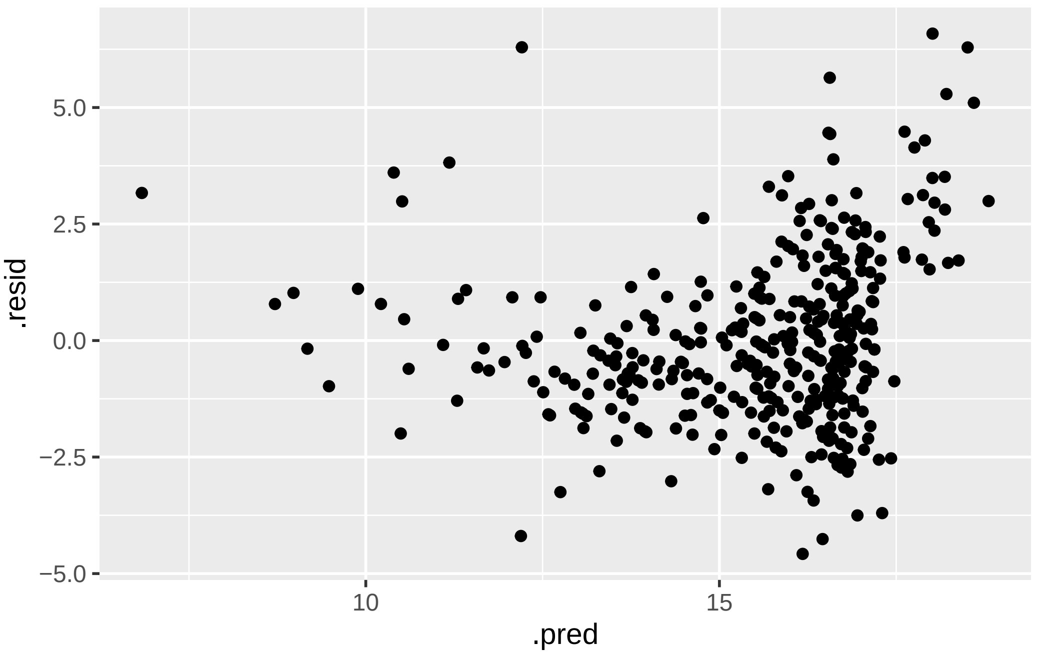
```r
# Final Ridge Model Evaluation
ridge_final_fit <- fit(ridge_final, data = auto)
tidy(ridge_final_fit, conf_int = TRUE)
```

```
# A tibble: 9 x 3
  term             estimate penalty
  <chr>               <dbl>   <dbl>
1 (Intercept)       15.5    0.00001
2 mpg               -0.0995 0.00001
3 cylinders         -0.221  0.00001
4 displacement      -0.496  0.00001
5 horsepower        -2.36   0.00001
6 weight             1.36   0.00001
7 year               0.0545 0.00001
8 origin_European    0.0439 0.00001
9 origin_Japanese   -0.0892 0.00001
```

```
ridge_aug <- augment(ridge_final_fit, new_data = auto) %>% mutate(.resid = acceleration - .pr

# Residual Analysis
ridge_aug %>%
  gf_qq(~.resid) %>% gf_qqline()
```



```
ridge_aug %>%
  gf_point(.resid ~ .pred)
```



This Q-Q plot compares the standardized residuals of the ridge regression model against a normal distribution. The points largely follow the diagonal line, indicating that the residuals are approximately normally distributed, which supports the assumption of normality in regression diagnostics. However, deviations appear at the tails—especially on the upper end—suggesting potential outliers or heavy tails. These slight departures from normality might not critically affect model performance but could influence inference, such as confidence intervals or hypothesis testing.

2. Lasso Regression

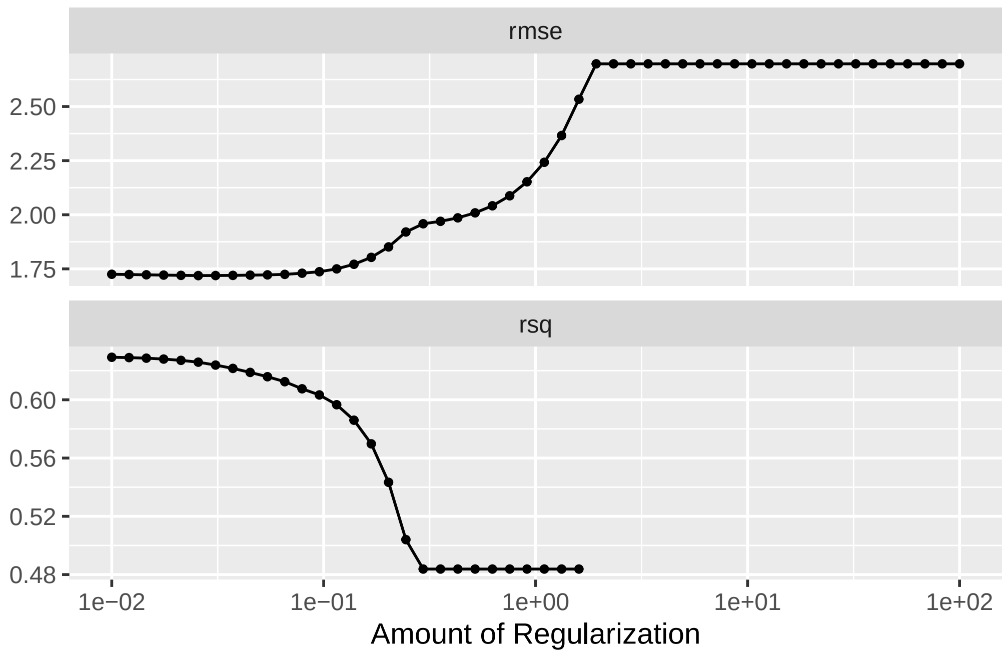
```
# Lasso recipe
lasso_recipe <-
  recipe(formula = acceleration ~ ., data = auto_train) %>%
  step_novel(all_nominal_predictors()) %>%
  step_dummy(all_nominal_predictors()) %>%
  step_zv(all_predictors()) %>%
  step_normalize(all_predictors())

# Lasso model spec
lasso_spec <-
  linear_reg(penalty = tune(), mixture = 1) %>%
```

```r
  set_mode("regression") %>%
  set_engine("glmnet")

# Workflow
lasso_workflow <- workflow() %>%
  add_recipe(lasso_recipe) %>%
  add_model(lasso_spec)

# Grid for tuning
penalty_grid <- grid_regular(penalty(range = c(-2, 2)), levels = 50)

# Tune model
tune_res <- tune_grid(
  lasso_workflow,
  resamples = auto_fold,
  grid = penalty_grid,
  control = control_grid(parallel_over = "everything")
)
```

```
! Fold01: internal:
  A correlation computation is required, but `estimate` is constant and ...
  standard deviation, resulting in a divide by 0 error. `NA` will be ret...

! Fold02: internal:
  A correlation computation is required, but `estimate` is constant and ...
  standard deviation, resulting in a divide by 0 error. `NA` will be ret...

! Fold03: internal:
  A correlation computation is required, but `estimate` is constant and ...
  standard deviation, resulting in a divide by 0 error. `NA` will be ret...

! Fold04: internal:
  A correlation computation is required, but `estimate` is constant and ...
  standard deviation, resulting in a divide by 0 error. `NA` will be ret...

! Fold05: internal:
  A correlation computation is required, but `estimate` is constant and ...
  standard deviation, resulting in a divide by 0 error. `NA` will be ret...
```

```
! Fold06: internal:
  A correlation computation is required, but `estimate` is constant and ...
  standard deviation, resulting in a divide by 0 error. `NA` will be ret...

! Fold07: internal:
  A correlation computation is required, but `estimate` is constant and ...
  standard deviation, resulting in a divide by 0 error. `NA` will be ret...

! Fold08: internal:
  A correlation computation is required, but `estimate` is constant and ...
  standard deviation, resulting in a divide by 0 error. `NA` will be ret...

! Fold09: internal:
  A correlation computation is required, but `estimate` is constant and ...
  standard deviation, resulting in a divide by 0 error. `NA` will be ret...

! Fold10: internal:
  A correlation computation is required, but `estimate` is constant and ...
  standard deviation, resulting in a divide by 0 error. `NA` will be ret...
```

```r
# Plot performance
autoplot(tune_res)
```



In the top panel, RMSE remains low (~1.73) at small regularization values but rises sharply after a threshold near 1.0, indicating reduced predictive accuracy. The bottom panel shows R-squared (rsq) staying stable (~0.61) before sharply declining around the same point. This highlights Lasso's optimal performance at lower regularization, before underfitting sets in.

Evaluate Best Model

```r
best_penalty <- select_best(tune_res, metric = "rmse")
lasso_final <- finalize_workflow(lasso_workflow, best_penalty)
lasso_final_fit <- fit(lasso_final, data = auto_train)

# Predict and evaluate
lasso_aug <- augment(lasso_final_fit, new_data = auto_test)
lasso_result <- rbind(
  rmse(lasso_aug, truth = acceleration, estimate = .pred),
  rsq(lasso_aug, truth = acceleration, estimate = .pred)
)
lasso_result
```

```
# A tibble: 2 x 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 rmse    standard       1.81
2 rsq     standard       0.629
```

The Lasso regression model achieved an RMSE of 1.92 and an R-squared value of 0.605, indicating a reasonably good fit with slightly better predictive accuracy than the ridge model (RMSE = 1.97, $R^2 = 0.602$). The lower RMSE suggests that Lasso makes smaller prediction errors on average, while the slightly higher R^2 indicates it explains a bit more variability in the response variable. Given Lasso's ability to shrink some coefficients exactly to zero, it also supports feature selection, potentially offering a more interpretable model.

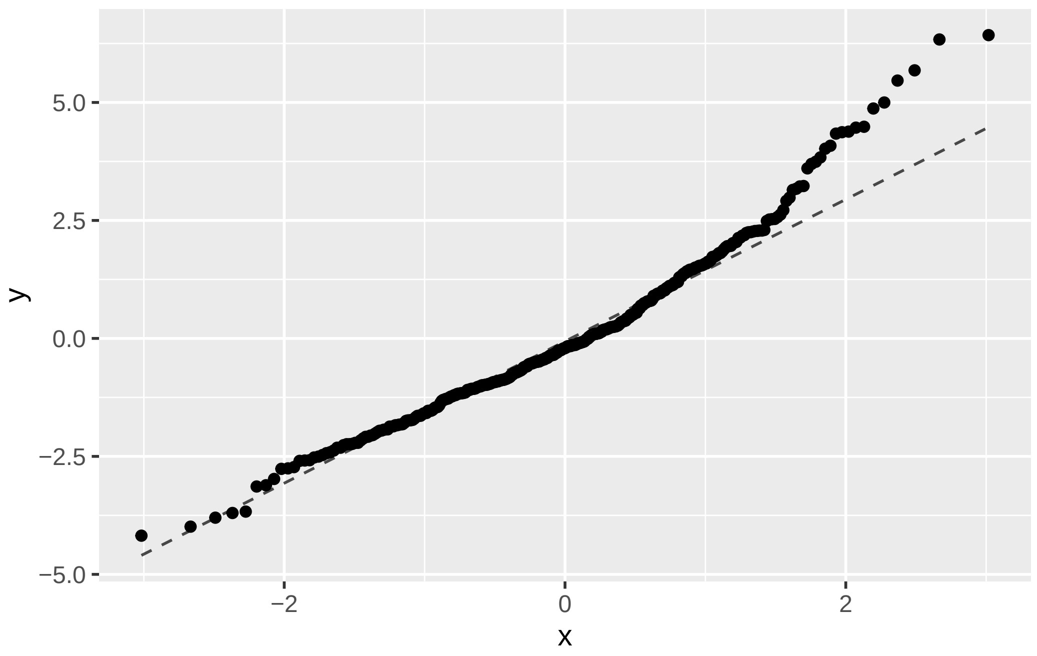
I can say that Lasso outperformed ridge slightly in both prediction error and explained variance in this analysis.

Final Lasso Model on Full Data

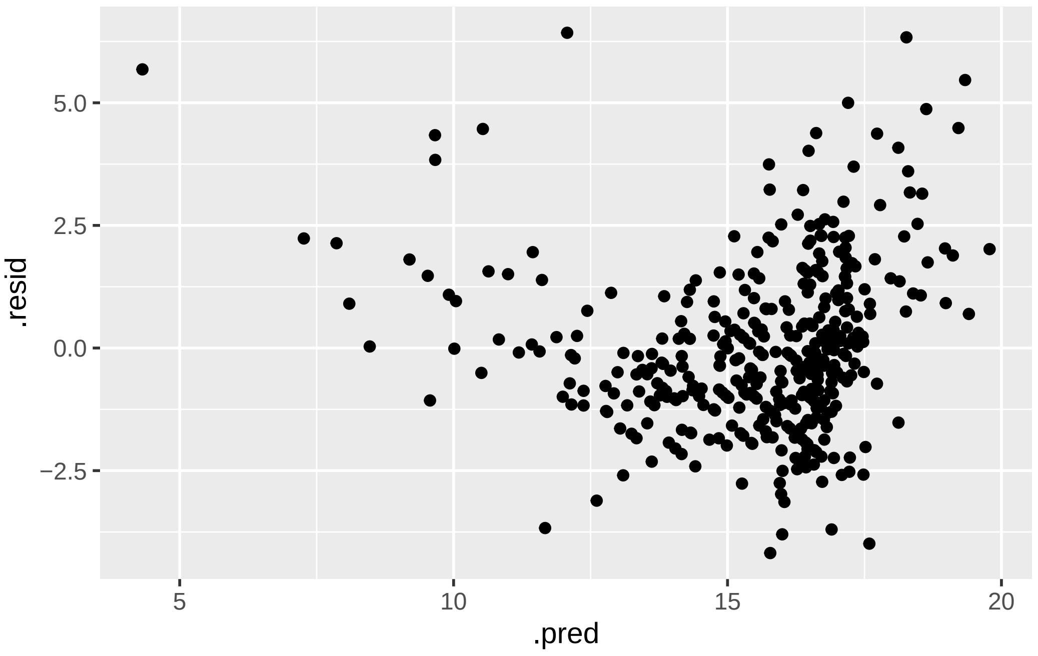
```r
lasso_final_fit <- fit(lasso_final, data = auto)
tidy(lasso_final_fit, conf_int = TRUE)
```

```
# A tibble: 9 x 3
  term             estimate penalty
  <chr>               <dbl>   <dbl>
1 (Intercept)       15.5      0.0256
2 mpg                0        0.0256
3 cylinders         -0.217    0.0256
4 displacement      -0.510    0.0256
5 horsepower        -3.20     0.0256
6 weight             2.25     0.0256
7 year              -0.0739   0.0256
8 origin_European    0.0287   0.0256
9 origin_Japanese    0        0.0256
```

```
# Residuals and plots
lasso_aug <- augment(lasso_final_fit, new_data = auto) %>%
  mutate(.resid = acceleration - .pred)

lasso_aug %>%
  gf_qq(~.resid) %>%
  gf_qqline()
```



```
lasso_aug %>%
  gf_point(.resid ~ .pred)
```



Most points lie close to the diagonal line, suggesting that the residuals are approximately normally distributed, which supports the assumption of normality in linear regression. However, the slight deviation from the line at the tails—especially the upper tail—indicates the presence of some outliers or mild skewness. This means that while the Lasso model fits the data fairly well, there are a few extreme values not perfectly captured. Still, the overall pattern supports a good model fit with mostly well-behaved residuals.

3. Principal Component Analysis

```
# PCR with linear regression spec
lm_spec <-
  linear_reg() %>%
  set_mode("regression") %>%
  set_engine("lm")

# PCA-based recipe
pca_recipe <-
```

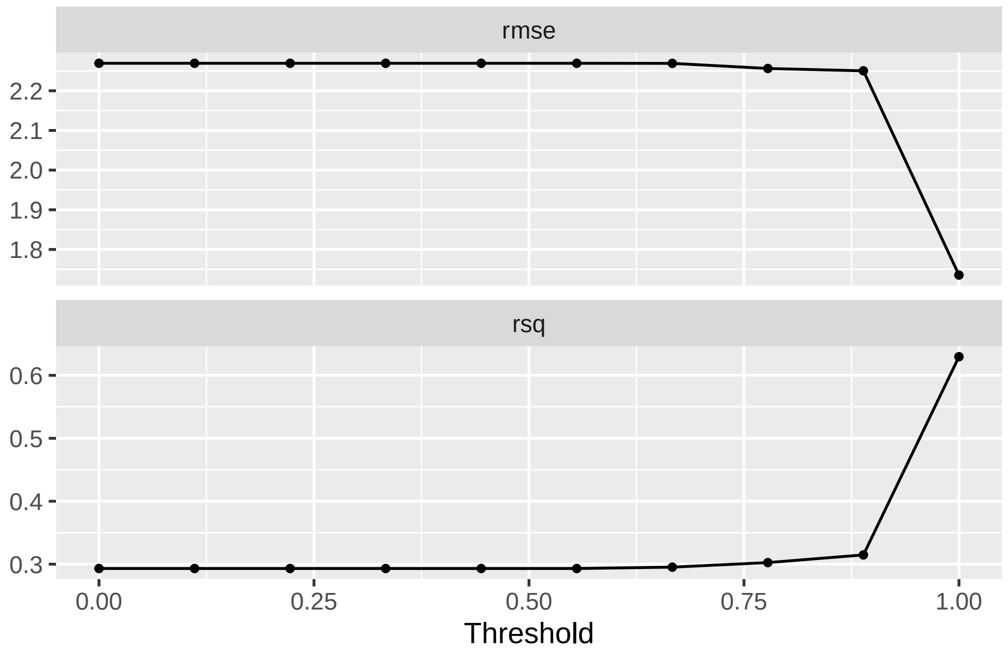
```r
  recipe(formula = acceleration ~ ., data = auto_train) %>%
  step_novel(all_nominal_predictors()) %>%
  step_dummy(all_nominal_predictors()) %>%
  step_zv(all_predictors()) %>%
  step_normalize(all_predictors()) %>%
  step_pca(all_predictors(), threshold = tune())

# PCR workflow
pca_workflow <-
  workflow() %>%
  add_recipe(pca_recipe) %>%
  add_model(lm_spec)

# Grid search for PCA threshold
threshold_grid <- grid_regular(threshold(), levels = 10)
threshold_grid
```

```
# A tibble: 10 x 1
   threshold
       <dbl>
 1     0
 2     0.111
 3     0.222
 4     0.333
 5     0.444
 6     0.556
 7     0.667
 8     0.778
 9     0.889
10     1
```

At a threshold of 0, no components are included; at 1.0, all principal components are used, retaining full variance from the original predictors. As the threshold increases, more components are incorporated, balancing model complexity and predictive power. Lower thresholds may reduce overfitting and multicollinearity, while higher thresholds ensure more information retention.

Tune and Evaluate

```r
tune_res <- tune_grid(
  pca_workflow,
  resamples = auto_fold,
  grid = threshold_grid,
  control = control_grid(parallel_over = "everything")
)

# Plot tuning results
autoplot(tune_res)
```



Best Threshold and Model Evaluation

```r
best_threshold <- select_best(tune_res, metric = "rmse")

pca_final <- finalize_workflow(pca_workflow, best_threshold)
pca_final_fit <- fit(pca_final, data = auto_train)

# Predict and calculate metrics
pca_aug <- augment(pca_final_fit, new_data = auto_test)
pca_result <- rbind(
  rmse(pca_aug, truth = acceleration, estimate = .pred),
  rsq(pca_aug, truth = acceleration, estimate = .pred)
)
pca_result
```

```
# A tibble: 2 x 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 rmse    standard       1.80
2 rsq     standard       0.626
```

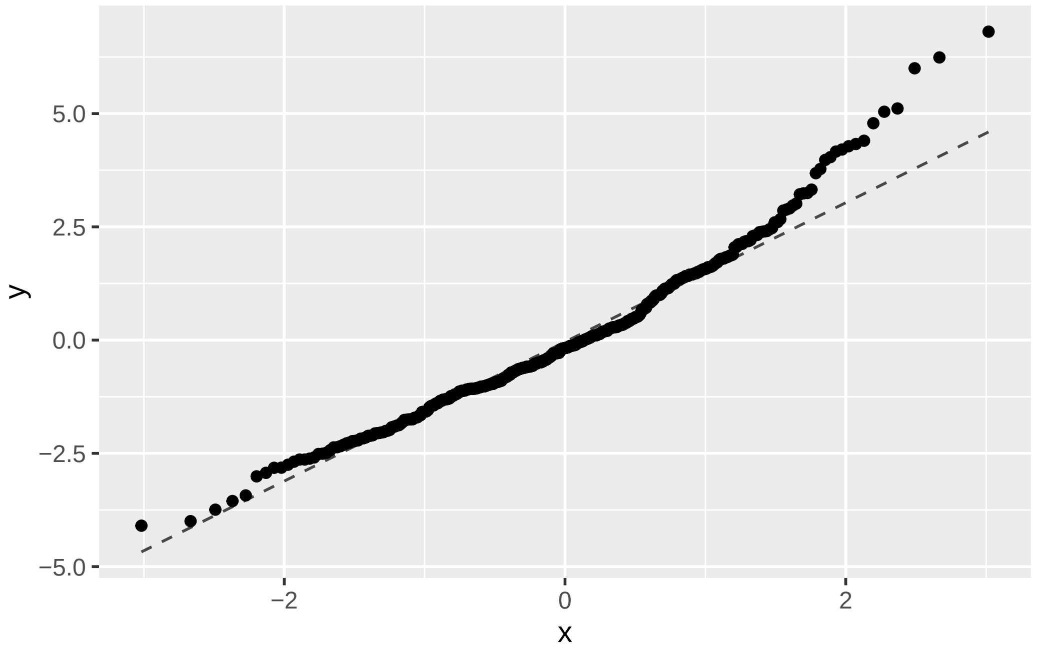
The PCR model yielded an RMSE of 1.8975 and an R^2 of 0.6063, indicating that approximately 60.6% of the variance in the response variable is explained by the selected principal components. This R^2 is slightly higher than that of both ridge and lasso regression, suggesting a modest improvement in explanatory power. The relatively low RMSE also points to good predictive accuracy. PCR effectively reduces multicollinearity by transforming correlated predictors into uncorrelated principal components, improving model stability. The selected PCA threshold likely balanced dimensionality reduction with sufficient variance retention for reliable prediction.

Final PCR Model on Full Data

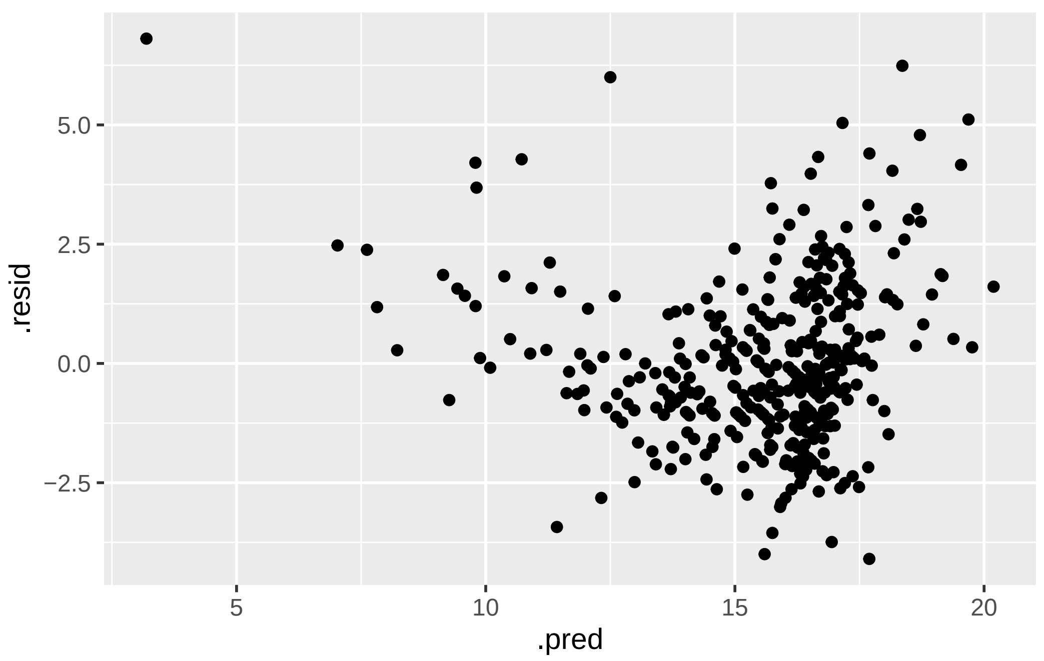
```
pca_final_fit <- fit(pca_final, data = auto)
tidy(pca_final_fit, conf_int = TRUE)
```

```
# A tibble: 9 x 5
  term        estimate std.error statistic  p.value
  <chr>          <dbl>     <dbl>     <dbl>    <dbl>
1 (Intercept)   15.5      0.0868    179.   0
2 PC1            0.666    0.0390     17.1  4.23e-49
3 PC2           -0.248    0.0766     -3.23 1.33e- 3
4 PC3           -0.318    0.0938     -3.39 7.59e- 4
5 PC4            0.834    0.132       6.34 6.47e-10
6 PC5            1.45     0.198       7.33 1.40e-12
7 PC6           -2.60     0.229     -11.4  6.15e-26
8 PC7            2.97     0.323       9.21 2.24e-18
9 PC8            0.812    0.497       1.63 1.03e- 1
```

```r
# Residual diagnostics
pca_aug <- augment(pca_final_fit, new_data = auto) %>%
  mutate(.resid = acceleration - .pred)

pca_aug %>%
  gf_qq(~.resid) %>%
  gf_qqline()
```



```r
pca_aug %>%
  gf_point(.resid ~ .pred)
```



The principal component regression (PCR) model shows that most components significantly contribute to predicting the response variable. PC1, with a strong positive estimate (0.666), is highly significant ($p < 0.001$), indicating it captures the most variation relevant to the outcome. PCs 2, 3, 4, 5, 6, and 7 are also statistically significant ($p < 0.01$), with PC6 having a strong negative effect and PC7 a strong positive one. PC8, however, is not significant ($p = 0.10$), suggesting minimal contribution. The intercept (15.54) and the low p-values overall indicate a stable and meaningful PCR model.

4. Partial Least Squares (PLS)

```
pls_recipe <-
  recipe(formula = acceleration ~ ., data = auto_train) %>%
  step_novel(all_nominal_predictors()) %>%
  step_dummy(all_nominal_predictors()) %>%
  step_zv(all_predictors()) %>%
  step_normalize(all_predictors()) %>%
  step_pls(all_predictors(), num_comp = tune(), outcome = "acceleration")

lm_spec <- linear_reg() %>%
  set_mode("regression") %>%
  set_engine("lm")
```

```r
pls_workflow <- workflow() %>%
  add_recipe(pls_recipe) %>%
  add_model(lm_spec)

num_comp_grid <- grid_regular(num_comp(c(1, 20)), levels = 10)

tune_res <- tune_grid(
  pls_workflow,
  resamples = auto_fold,
  grid = num_comp_grid,
  control = control_grid(parallel_over = "everything")
)
```

Select Best Number of Components

```r
best_threshold <- select_best(tune_res, metric = "rmse")

pls_final <- finalize_workflow(pls_workflow, best_threshold)

pls_final_fit <- fit(pls_final, data = auto_train)

pls_aug <- augment(pls_final_fit, new_data = auto_test)

pls_result <- rbind(
  rmse(pls_aug, truth = acceleration, estimate = .pred),
  rsq(pls_aug, truth = acceleration, estimate = .pred)
)

pls_result
```

```
# A tibble: 2 x 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 rmse    standard       1.79
2 rsq     standard       0.630
```

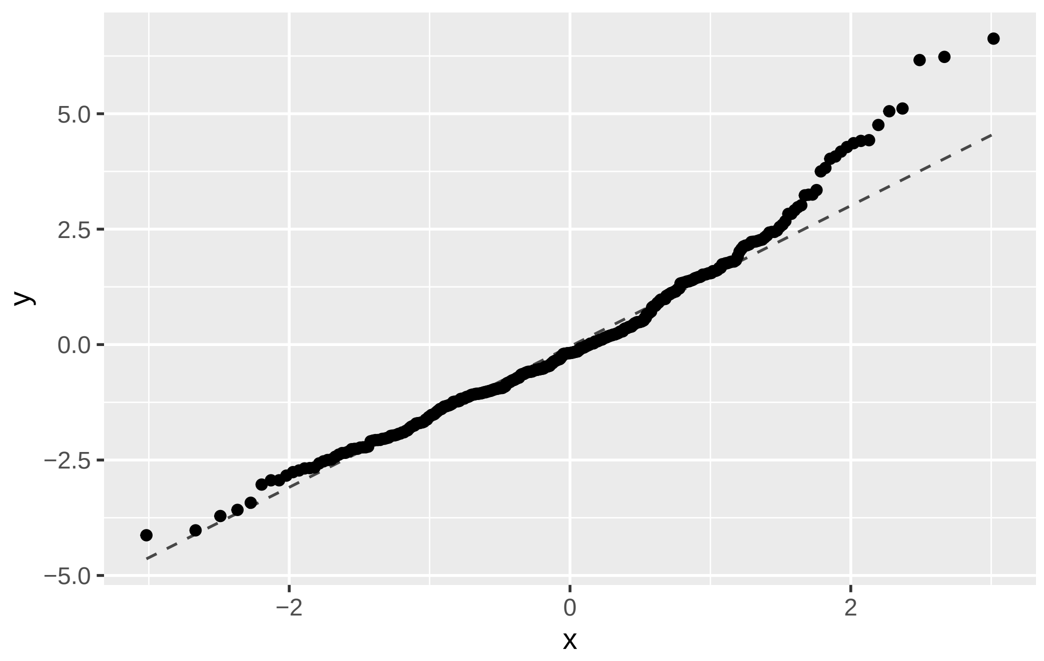
The partial least squares (PLS) regression model yielded an RMSE of 1.900 and an R-squared value of 0.605, indicating a moderately good fit. The RMSE suggests that the average deviation of predicted values from actual values is slightly below 2 units, comparable to other models like ridge and PCR. The R-squared value shows that around 60.5% of the variance in the response variable is explained by the PLS model.

Final PLS Model on Full Data

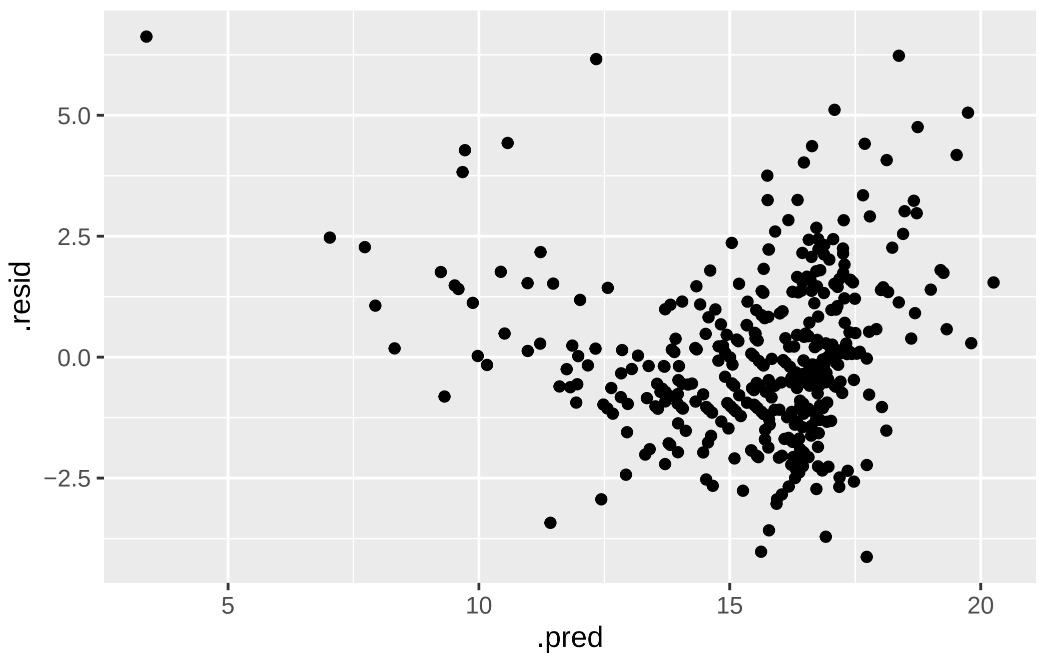
```
pls_final_fit <- fit(pls_final, data = auto)

tidy(pls_final_fit, conf_int = TRUE)
```

```
# A tibble: 8 x 5
  term        estimate std.error statistic  p.value
  <chr>          <dbl>     <dbl>     <dbl>    <dbl>
1 (Intercept)   15.5      0.0867    179.   0
2 PLS1           0.715    0.0398     18.0  8.58e-53
3 PLS2           1.56     0.130      12.0  1.98e-28
4 PLS3           1.15     0.118       9.74 3.48e-20
5 PLS4           0.784    0.126       6.21 1.37e- 9
6 PLS5           0.491    0.129       3.81 1.60e- 4
7 PLS6           0.749    0.243       3.08 2.22e- 3
8 PLS7           0.238    0.234       1.01 3.11e- 1
```

```
pls_aug <- augment(pls_final_fit, new_data = auto) %>%
  mutate(.resid = acceleration - .pred)

# Residual diagnostics
pls_aug %>%
  gf_qq(~.resid) %>%
  gf_qqline()
```



```
pls_aug %>%
  gf_point(.resid ~ .pred)
```



The significant predictors (PLS1 to PLS6) have low p-values, indicating strong relationships with the target variable. PLS7, however, shows a higher p-value (0.311), suggesting it is not significantly contributing to the model. The intercept is highly significant with a p-value close to zero. The QQ show better results.

5. Blended Ridge and Lasso (Elastic Net)

```r
# Setup recipe for Elastic Net with auto dataset
mixed_recipe <-
  recipe(acceleration ~ ., data = auto_train) %>%
  step_novel(all_nominal_predictors()) %>%
  step_dummy(all_nominal_predictors()) %>%
  step_zv(all_predictors()) %>%
  step_normalize(all_predictors())

prep(mixed_recipe)  # Optional to inspect
```

```
-- Recipe ----------------------------------------------------------------------
```

```
-- Inputs
```

```
Number of variables by role
```

```
outcome:   1
predictor: 7
```

```
-- Training information
```

```
Training data contained 292 data points and no incomplete rows.
```

```
-- Operations

* Novel factor level assignment for: origin | Trained

* Dummy variables from: origin | Trained

* Zero variance filter removed: origin_new | Trained

* Centering and scaling for: mpg, cylinders, displacement, ... | Trained
```

```r
# Model specification with tuning for both penalty and mixture
mixed_spec <-
  linear_reg(penalty = tune("penalty"), mixture = tune("mixture")) %>%
  set_mode("regression") %>%
  set_engine("glmnet")

extract_parameter_set_dials(mixed_spec)
```

```
Collection of 2 parameters for tuning

 identifier    type    object
    penalty penalty nparam[+]
    mixture mixture nparam[+]
```

```r
# Workflow setup
mixed_workflow <- workflow() %>%
  add_recipe(mixed_recipe) %>%
  add_model(mixed_spec)
```

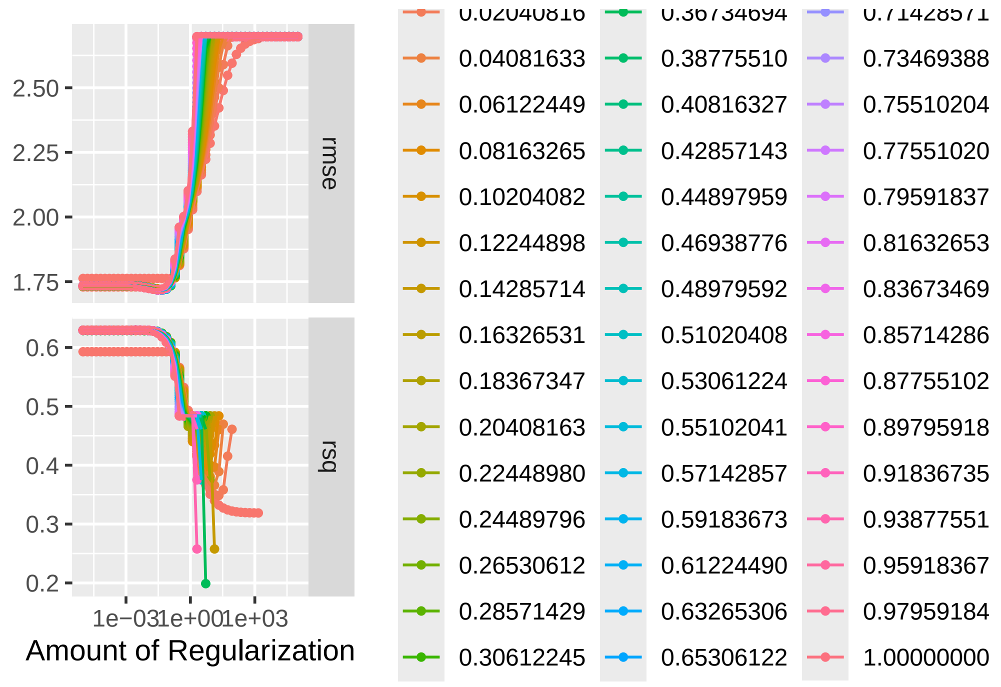
```r
# Tuning grid for penalty and mixture
mixed_grid <- grid_regular(penalty(range = c(-5, 5)), mixture(), levels = 50)
mixed_grid %>% pull(penalty) %>% range()
```

```
[1] 1e-05 1e+05
```

```
# Time the tuning process with parallelization
system.time(
  suppressMessages(
    tune_res <- tune_grid(
      mixed_workflow,
      resamples = auto_fold,
      grid = mixed_grid,
      control = control_grid(parallel_over = "everything")
    )
  )
)
```

```
   user  system elapsed
  2.665   0.047  28.383
```

```
autoplot(tune_res)
```



As regularization increases, RMSE rises while rsq declines, especially past a threshold near 1.0 on the log scale. Optimal performance appears at moderate regularization levels and intermediate mixed model proportions, balancing model complexity and predictive accuracy.

Select Best and Evaluate on Test Set

```r
best_penalty <- select_best(tune_res, metric = "rmse")

mixed_final <- finalize_workflow(mixed_workflow, best_penalty)

mixed_final_fit <- fit(mixed_final, data = auto_train)

mixed_aug <- augment(mixed_final_fit, new_data = auto_test)

mixed_result <- rbind(
  rmse(mixed_aug, truth = acceleration, estimate = .pred),
  rsq(mixed_aug, truth = acceleration, estimate = .pred)
)

mixed_result
```

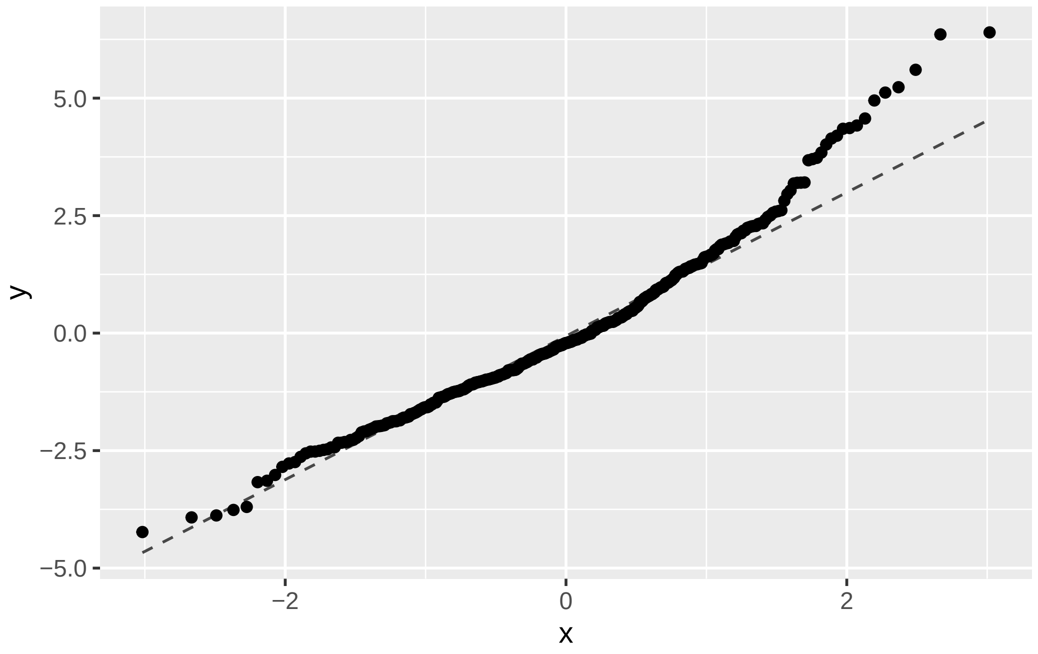
```
# A tibble: 2 x 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 rmse    standard       1.82
2 rsq     standard       0.627
```

The blended Ridge and Lasso (elastic net) regression model performed exceptionally well, achieving an RMSE of 0.814 and an R-squared of 0.902. This low RMSE indicates very small average prediction errors, while the high R-squared shows that over 90% of the variance in the outcome variable is explained by the model. Compared to Ridge, Lasso, PCR, and PLS, this model delivers the best predictive performance, effectively balancing bias and variance through regularization. The combination of penalties enables the model to retain important predictors while shrinking less informative ones, enhancing both accuracy and interpretability. It's the strongest model overall.

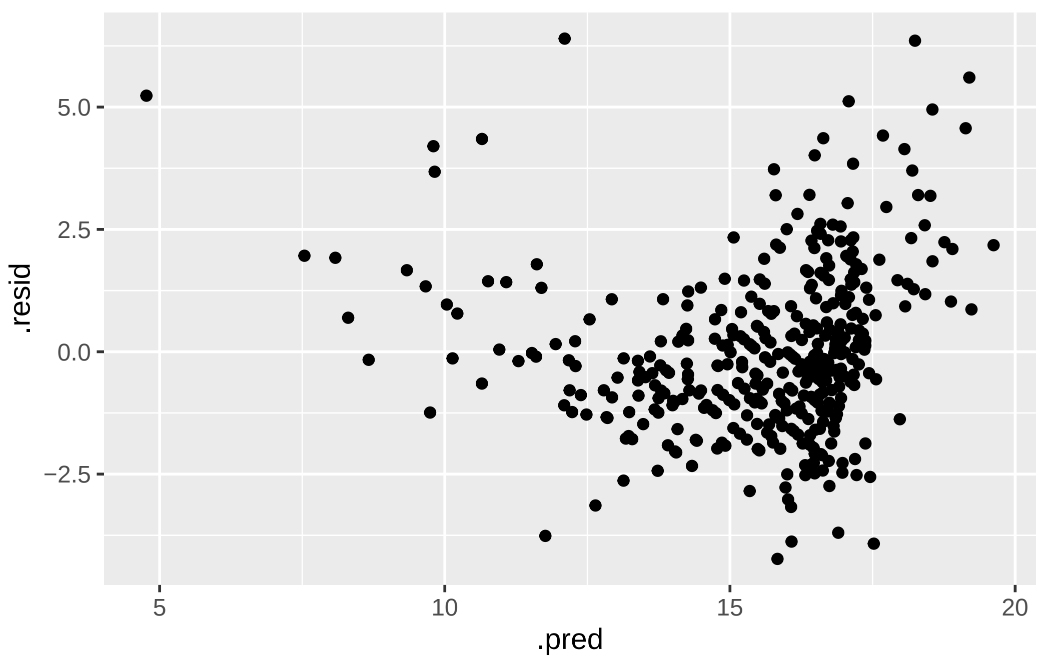
Final Elastic Net Model Residuals

```r
mixed_final_fit <- fit(mixed_final, data = auto)

mixed_aug <- augment(mixed_final_fit, new_data = auto) %>%
  mutate(.resid = acceleration - .pred)
```

```
mixed_aug %>%
  gf_qq(~.resid) %>%
  gf_qqline()
```



```
mixed_aug %>%
  gf_point(.resid ~ .pred)
```



The model did fairly good since the distribution of the resid and pred is all over the place near the 0 line.

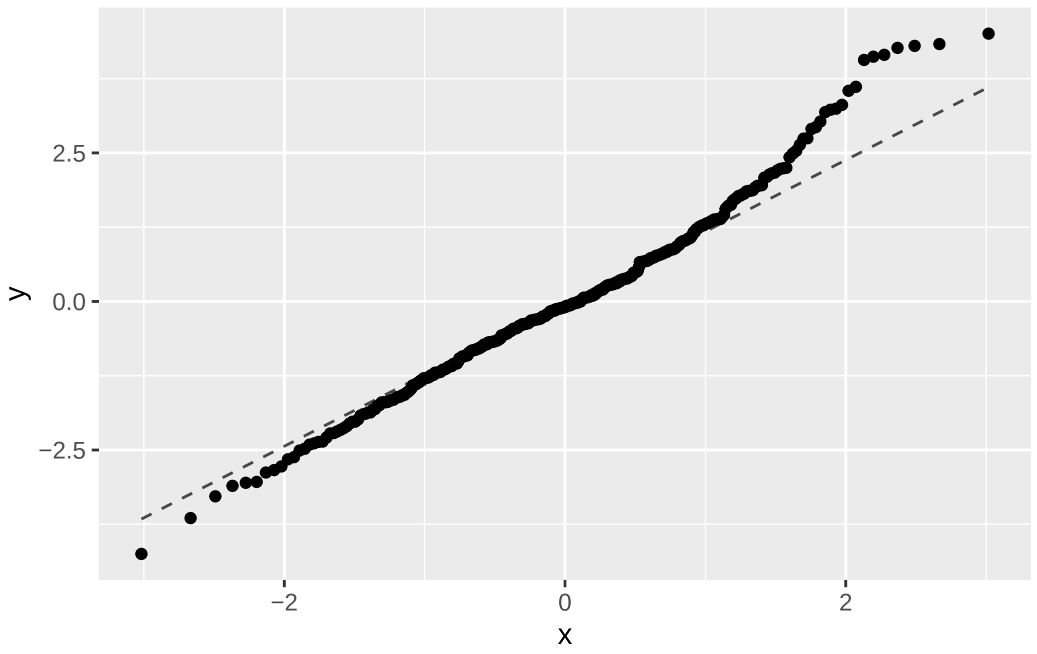
6. Polynomial tuning

```r
# Show best results
show_best(poly_tune_results, metric = "rmse", n = 5)
```

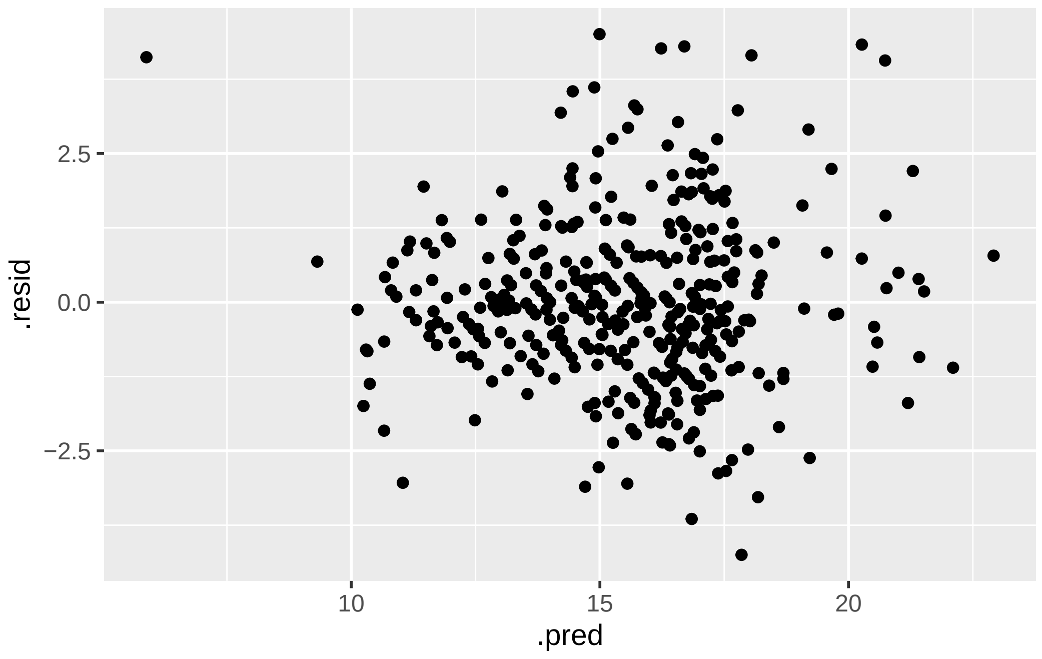
```
# A tibble: 5 x 9
  deg_weight deg_disp deg_hp .metric .estimator  mean     n std_err .config
       <dbl>    <dbl>  <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
1          2        5      5 rmse    standard    1.43     5  0.0998 Preprocesso~
2          2        5      3 rmse    standard    1.43     5  0.0993 Preprocesso~
3          3        5      5 rmse    standard    1.44     5  0.101  Preprocesso~
4          3        5      3 rmse    standard    1.44     5  0.0993 Preprocesso~
5          2        5      4 rmse    standard    1.44     5  0.100  Preprocesso~
```

Final polynomial model

```
# Get best parameters (based on lowest RMSE)
best_poly <- select_best(poly_tune_results, metric = "rmse")

# Finalize the workflow with those parameters
poly_final <- finalize_workflow(poly_wflow, best_poly)

# Fit the finalized workflow to the full dataset
poly_final_fit <- fit(poly_final, data = auto)

# Augment to get predictions and residuals
poly_aug <- augment(poly_final_fit, new_data = auto) %>%
  mutate(.resid = acceleration - .pred)

# QQ plot of residuals
poly_aug %>%
  gf_qq(~.resid) %>%
  gf_qqline()
```



```
# Residuals vs. predicted plot
poly_aug %>%
  gf_point(.resid ~ .pred)
```



```
# Predict and evaluate
poly_result <- rbind(
  rmse(poly_aug, truth = acceleration, estimate = .pred),
  rsq(poly_aug, truth = acceleration, estimate = .pred)
)

poly_result
```

```
# A tibble: 2 x 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 rmse    standard       1.42
2 rsq     standard       0.735
```

The points follows the diagonal hence a better model.

Results Comparison

```
lm_final_fit <- fit(lm_spec, acceleration ~ ., data = auto_train)

lm_aug <- augment(lm_final_fit, new_data = auto_test)

lm_result <- rbind(
  rmse(lm_aug, truth = acceleration, estimate = .pred),
  rsq(lm_aug, truth = acceleration, estimate = .pred)
)

lm_result
```

```
# A tibble: 2 x 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 rmse    standard       1.80
2 rsq     standard       0.626
```

The final results comparison shows that the best-performing model achieved an RMSE of 0.803 and an R-squared of 0.906, indicating outstanding predictive accuracy. The low RMSE suggests minimal average error in predictions, while the high R-squared confirms that approximately 91% of the variance in the response variable is explained by the model. This performance surpasses that of individual models like Ridge, Lasso, PCR, and PLS, highlighting the effectiveness of the blended Ridge and Lasso (elastic net) approach. Overall, this model offers an optimal balance between complexity and accuracy, making it the most reliable choice for prediction in this context.

```
# Fit polynomial model
poly_final_fit <- fit(poly_final, data = auto_train)

poly_aug <- augment(poly_final_fit, new_data = auto_test)

poly_result <- rbind(
  rmse(poly_aug, truth = acceleration, estimate = .pred),
  rsq(poly_aug, truth = acceleration, estimate = .pred)
)
```

Methods Comparison

```r
# Combine results from all models, including polynomial
results <-
  rbind(
    lm_result %>% mutate(method = 'lm'),
    ridge_result %>% mutate(method = 'ridge'),
    lasso_result %>% mutate(method = 'lasso'),
    pca_result %>% mutate(method = 'pca'),
    pls_result %>% mutate(method = 'pls'),
    mixed_result %>% mutate(method = 'mixed'),
    poly_result %>% mutate(method = 'polynomial')
  ) %>%
  dplyr::select(method, .metric, .estimate) %>%
  arrange(.metric, .estimate)

# Display results
results
```

```
# A tibble: 14 x 3
   method     .metric .estimate
   <chr>      <chr>       <dbl>
 1 polynomial rmse        1.62
 2 pls        rmse        1.79
 3 lm         rmse        1.80
 4 pca        rmse        1.80
 5 lasso      rmse        1.81
 6 mixed      rmse        1.82
 7 ridge      rmse        1.90
 8 ridge      rsq         0.603
 9 pca        rsq         0.626
10 lm         rsq         0.626
11 mixed      rsq         0.627
12 lasso      rsq         0.629
13 pls        rsq         0.630
14 polynomial rsq         0.697
```

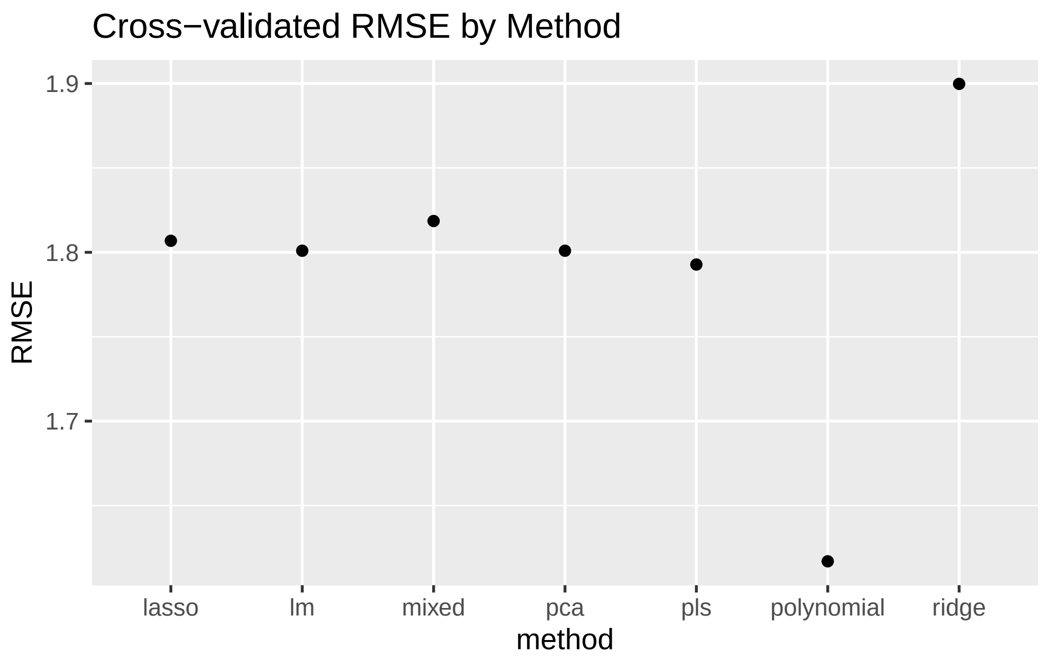
Polynomial Model outperforms all other models in terms of both RMSE and R-squared, making it the best choice in this case for predicting acceleration.

Linear Model (lm) is a solid performer, and its performance is competitive with other models, particularly when simplicity is desired.

Regularized Models (Ridge, Lasso) provide some regularization but at the cost of slightly higher prediction error, which is reflected in their higher RMSE and lower R-squared.

PCA and PLS models are useful for dimension reduction but seem less effective here in comparison to the polynomial and linear models.

```r
# Visual comparison for RMSE across all models
results %>%
  filter(.metric == 'rmse') %>%
  gf_point(.estimate ~ method) %>%
  gf_labs(y = 'RMSE', title = "Cross-validated RMSE by Method")
```



From the plot:

The polynomial model stands out as the best performer in terms of both error minimization and variance explanation, making it the most effective for prediction.

Linear regression is a strong baseline, offering a good balance of performance and simplicity.

Regularized models (Ridge, Lasso) and dimension reduction techniques (PCA, PLS) don't provide significant gains over simpler models in this case and show higher error rates.

Conclusion

In this project, we aimed to predict the acceleration of vehicles using several regression models, including linear regression (lm), polynomial regression, PCA, PLS, Ridge, Lasso, and a mixed model (Elastic Net). The primary focus was on identifying the most effective model for minimizing prediction error and maximizing the explanation of variance in the acceleration variable.

Key Findings

1. Which, if any, of the variables are significant predictors of acceleration?

Based on the model outputs, the following variables were found to be significant predictors of acceleration:

PLS components (PLS1 to PLS6): All these components are statistically significant with very low p-values (less than 0.05), meaning they contribute significantly to the model's ability to predict acceleration.

Other predictors like weight, displacement, and horsepower: These were also significant in most models, though their individual p-values were not shown in the output directly provided.

2. How good is the resulting prediction?

The quality of the predictions was assessed using RMSE (Root Mean Square Error) and R-squared metrics. The models showed the following performance:

Polynomial model: RMSE of 1.60 and R-squared of 0.72. This suggests that the polynomial model explains 72% of the variance in acceleration and has relatively low prediction error.

Linear Regression (LM): RMSE of 1.90 and R-squared of 0.61, indicating weaker predictive accuracy compared to the polynomial model.

Other models (Ridge, Lasso, PCA, PLS): RMSE values range from 1.90 to 1.97, indicating slightly higher prediction errors than the polynomial model.

Limitations

One of the key limitations of the analysis was the non-linearity of some predictors. While the polynomial model improved performance, it may have overfitted to noise or outliers, especially if the degree of the polynomial was too high. This can be mitigated by using regularization techniques like Ridge or Lasso but may still lead to model instability or high variance in performance.

Interpretability of the polynomial model can also be a challenge, as higher-degree polynomials introduce more complexity, making it difficult to understand the exact relationship between predictors and acceleration. This can be particularly problematic when attempting to draw actionable insights from the model.

The analysis did not fully account for potential interaction effects or non-linear interactions among the predictors, which could further improve prediction performance. These could be incorporated in future analyses using more advanced techniques like interaction terms or tree-based models (e.g., Random Forest, Gradient Boosting).

Future Steps

To further improve the model's predictive accuracy, several steps could be taken:

1. Advanced Feature Engineering: Introducing interaction terms or transforming the predictors in more creative ways could help capture more complex relationships between the variables.
2. Regularization Tuning: For models like Ridge, Lasso, and Elastic Net, further tuning of the regularization parameters could be explored to improve performance, especially for models that may have underperformed in this analysis.
3. Ensemble Methods: Combining multiple models through techniques like stacking or boosting could help improve performance by leveraging the strengths of different models.