Resampling and Regression Analysis

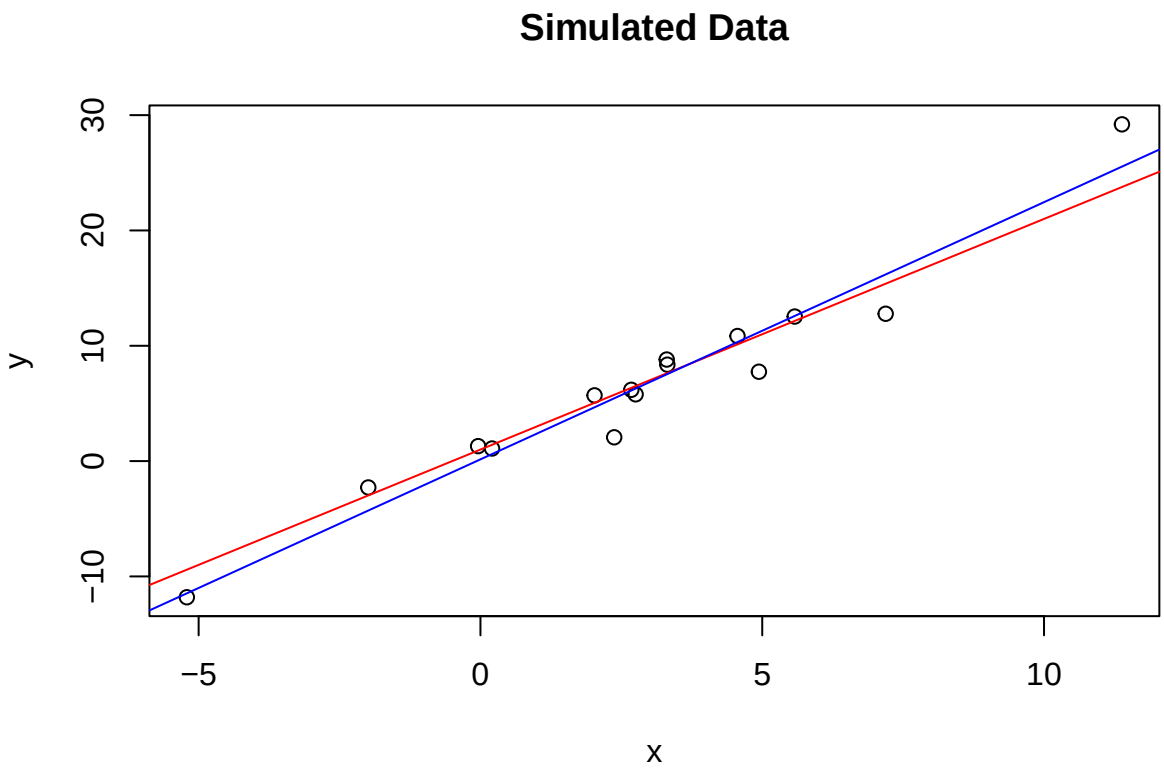
Isabelle Caroline Rose Cretton

Part 1: Resampling Part (a): Simulate Data and Plot

```
set.seed(111) # For reproducibility
n <- 15
beta_0 <- 1
beta_1 <- 2
mu_x <- 4
sigma_x <- 4
sigma <- 2

#sample x and build y
x <- rnorm(n, mean = mu_x, sd = sigma_x)
epsilon <- rnorm(n, mean = 0, sd = sigma) #generate errors (explicit def)
y <- beta_0 + beta_1 * x + epsilon

# generate model + estimate coeff of slope and intercept
model <- lm(y ~ x)
summary(model)
```

```
##
## Call:
## lm(formula = y ~ x)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.4219 -0.4205  0.4828  1.1373  3.6639
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   0.1506     0.6865   0.219     0.83
## x             2.2303     0.1452  15.356 1.03e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.112 on 13 degrees of freedom
## Multiple R-squared:  0.9478, Adjusted R-squared:  0.9437
## F-statistic: 235.8 on 1 and 13 DF,  p-value: 1.033e-09
```

```
#plot the data
plot(x, y, main = "Simulated Data")
abline(a = beta_0, b = beta_1, col = "red")
abline(lm(y ~ x), col = "blue")
```

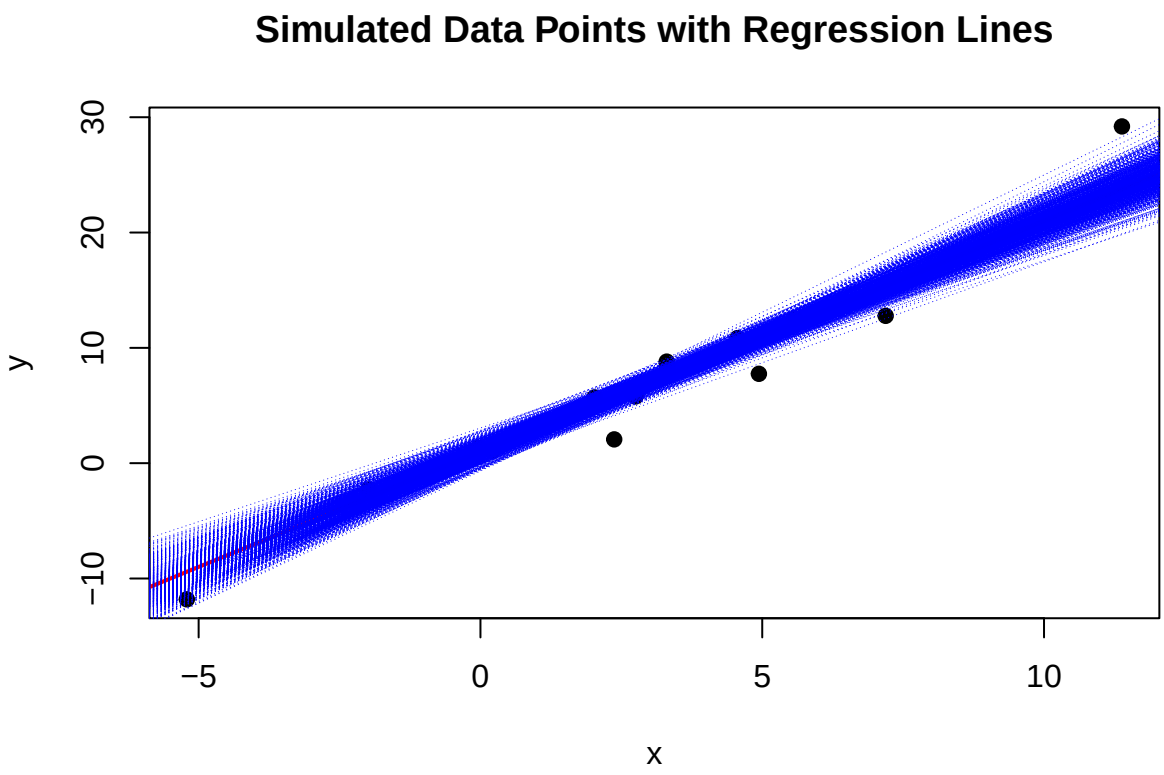


The regression analysis shows a strong positive correlation between x and y, with a slope estimate of 2.2303, indicating that y increases by about 2.23 units for each one-unit increase in x, as evidenced by the tight fit of the regression lines in the plot and a high R^2 value of 0.9478.

Part (b): simulations

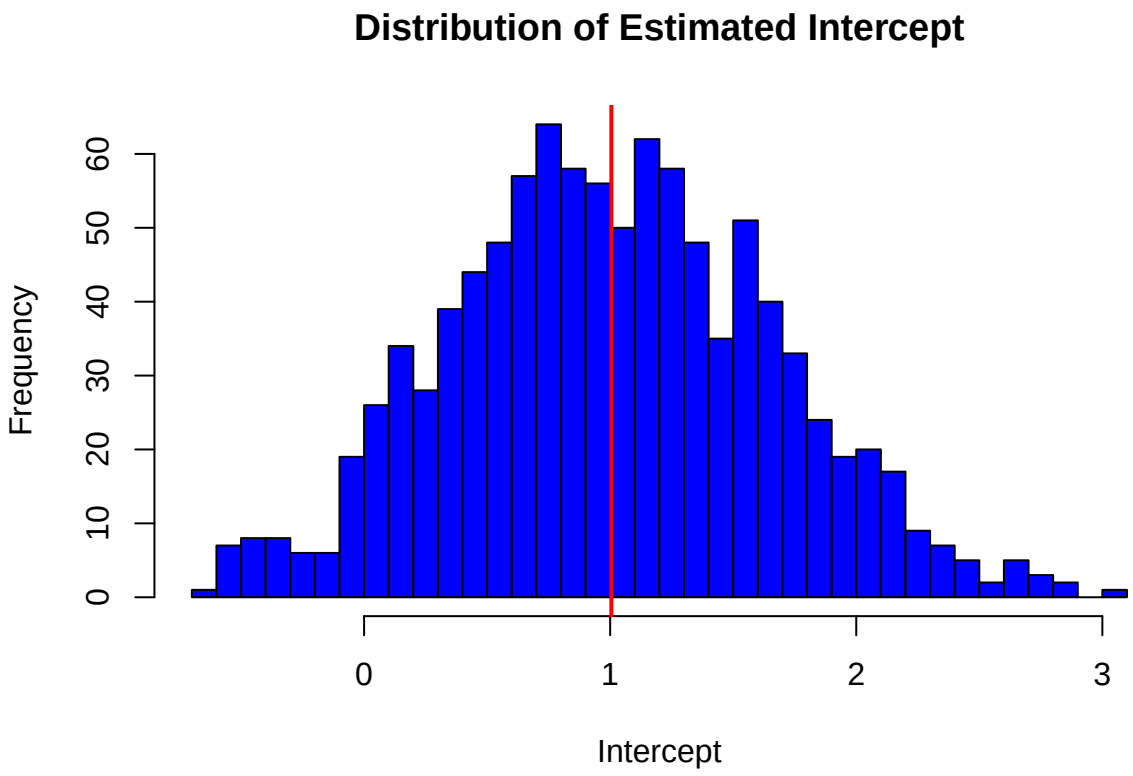
```r
R <- 1000  # n of simulations
intercept_estimates <- numeric(R)
slope_estimates <- numeric(R)

# Creating a data frame to store results
results_df <- data.frame(intercept = numeric(R), slope = numeric(R))

for (i in 1:R) {
  sampled_errors <- rnorm(n, mean = 0, sd = sigma)  # Generate new errors
  y_simulated <- beta_0 + beta_1 * x + sampled_errors
  fit_model <- lm(y_simulated ~ x)
  intercept_estimates[i] <- coef(fit_model)[1]
  slope_estimates[i] <- coef(fit_model)[2]
}

results_df$intercept <- intercept_estimates
results_df$slope <- slope_estimates

# plot the data points and regression lines
plot(x, y, main = "Simulated Data Points with Regression Lines", xlab = "x", ylab = "y", pch = 19)
abline(a = beta_0, b = beta_1, col = "red", lwd = 2)  # True regression line
```

```r
# Add each simulated regression line
apply(results_df, 1, function(row) abline(a = row["intercept"], b = row["slope"], col = "blue", lty = "
```

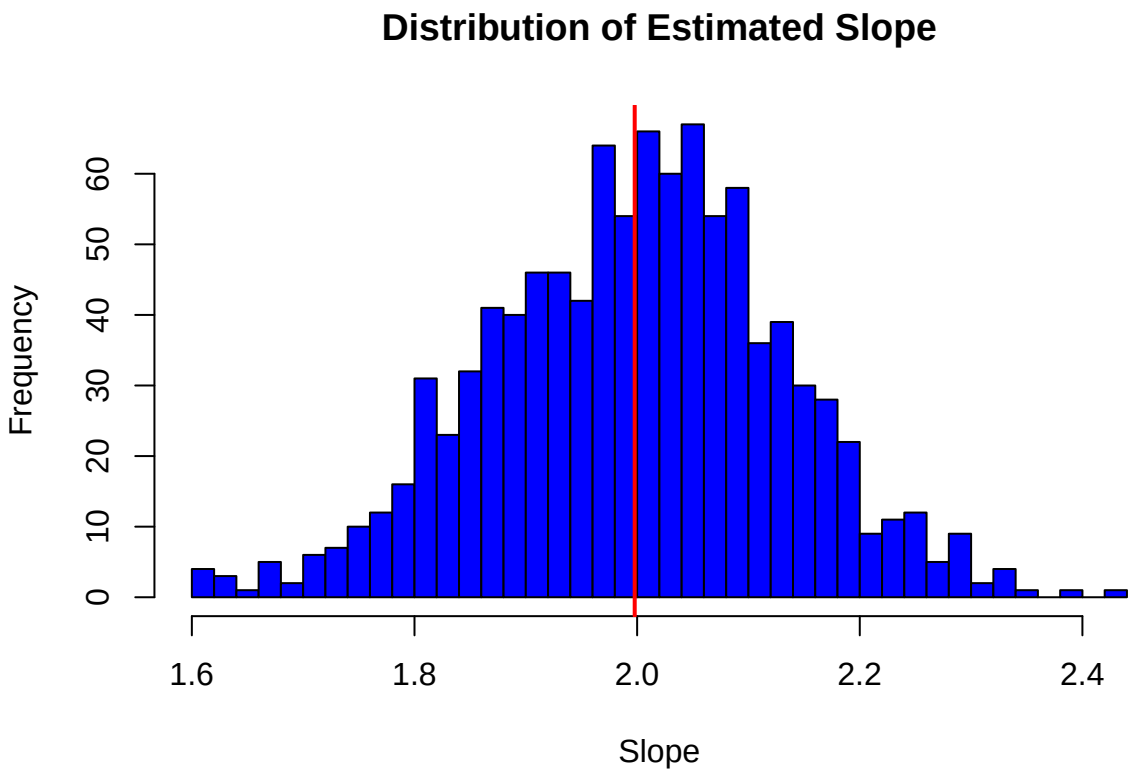



```
## NULL
```

```r
# Histogram of intercept estimates
hist(results_df$intercept, breaks = 30, main = "Distribution of Estimated Intercept", xlab = "Intercept
abline(v = mean(results_df$intercept), col = "red", lwd = 2)
```




```
# Histogram of slope estimates
hist(results_df$slope, breaks = 30, main = "Distribution of Estimated Slope", xlab = "Slope", col = "blu
abline(v = mean(results_df$slope), col = "red", lwd = 2)
```




```
#summaries of beta_0 and beta_1 estimates
cat("Summary of Intercept Estimates:\n")
```

```
## Summary of Intercept Estimates:
```

```
print(summary(results_df$intercept))
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -0.6792  0.5375  0.9861  1.0048  1.4537  3.0488
```

```
cat("Summary of Slope Estimates:\n")
```

```
## Summary of Slope Estimates:
```

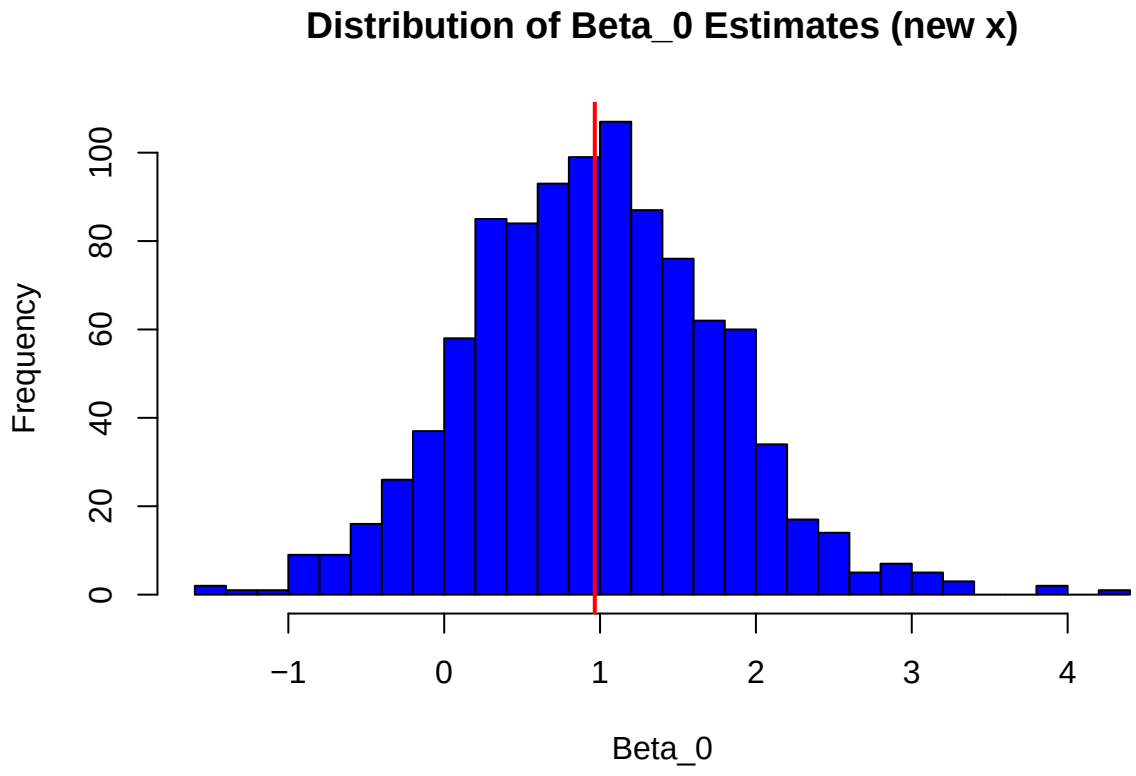
```
print(summary(results_df$slope))
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.605   1.906   2.006   1.998   2.087   2.426
```

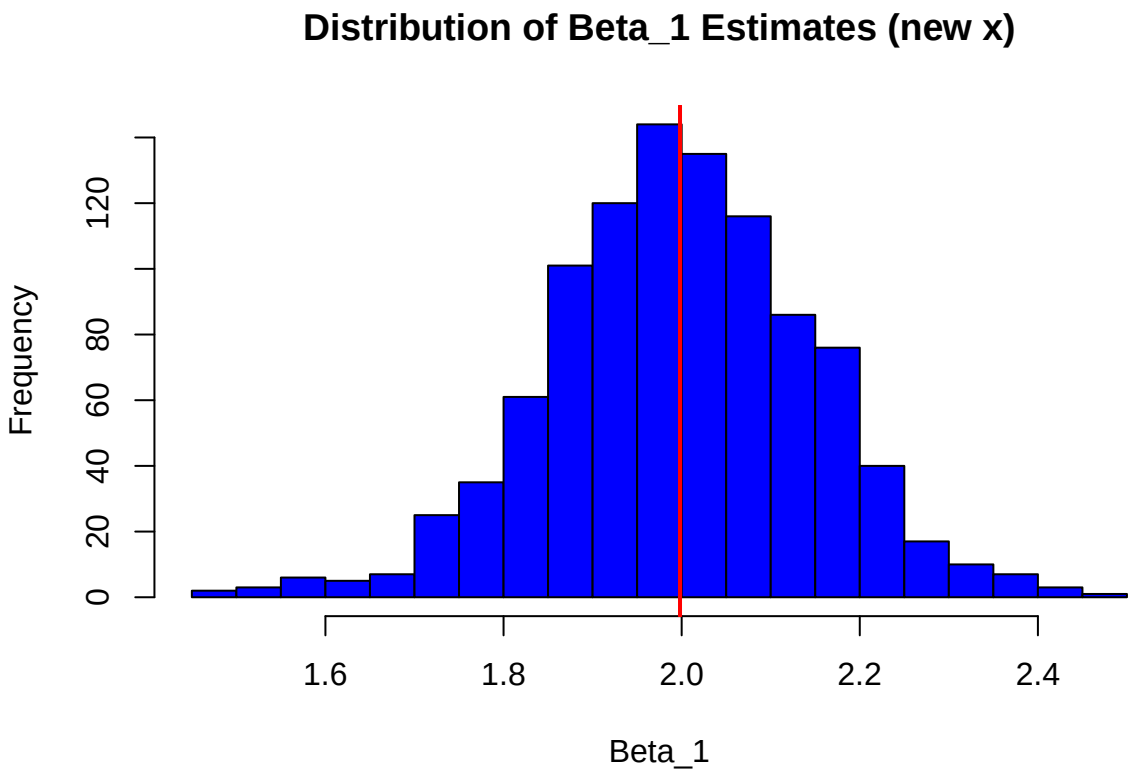
Both beta_0 and beta_1 are good estimates as they cluster around their true values under repeated sampling, confirmed by both the simulation scatter plot and the histogram peaks. The high number of simulations and their distribution confirm the validity of the model under the assumptions made (normally distributed errors w/ mean zero and constant variance)

Part (c): Repeat with a new x for every simulation

```r
beta_0_estimates <- numeric(R) # reset the estimates
beta_1_estimates <- numeric(R)

# Perform the simulations
for (i in 1:R) {
  x_new <- rnorm(n, mean = mu_x, sd = sigma_x)  # Draw new x for each simulation
  epsilon <- rnorm(n, mean = 0, sd = sigma)  # Generate new errors
  y_new <- beta_0 + beta_1 * x_new + epsilon
  model_new <- lm(y_new ~ x_new)
  beta_0_estimates[i] <- coef(model_new)[1]
  beta_1_estimates[i] <- coef(model_new)[2]
}

# Create histograms and print summaries
# Beta_0 Histogram
hist(beta_0_estimates, breaks = 30, main = "Distribution of Beta_0 Estimates (new x)", xlab = "Beta_0",
abline(v = mean(beta_0_estimates), col = "red", lwd = 2)
```




```r
# Beta_1 Histogram
hist(beta_1_estimates, breaks = 30, main = "Distribution of Beta_1 Estimates (new x)", xlab = "Beta_1",
abline(v = mean(beta_1_estimates), col = "red", lwd = 2)
```




```
# Output the summary statistics
cat("Summary of Beta_0 Estimates:\n")
```

```
## Summary of Beta_0 Estimates:
```

```
print(summary(beta_0_estimates))
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -1.4694  0.4116  0.9550  0.9661  1.4673  4.3708
```

```
cat("Summary of Beta_1 Estimates:\n")
```

```
## Summary of Beta_1 Estimates:
```

```
print(summary(beta_1_estimates))
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.470   1.902   1.997   1.998   2.096   2.471
```

There is a wider spread when x is redrawn, this emphasizes the importance of understanding the potential for increased uncertainty in predictions as x varies. However the estimates are still centered around the true values in both (b) and (c), indicating that the model is valid and robust even when x is redrawn.

Problem 2: Splits (cross-validation) Part (a): Split the data into training and testing sets (80/20) and fit polynomial model

```r
# Load required library and data
require(fma)  # Ensure 'fma' is installed
```

```
## Loading required package: fma
```

```
## Loading required package: forecast
```

```
## Warning: package 'forecast' was built under R version 4.3.3
```

```
## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo
```

```r
data("bicoal")
year <- c(time(bicoal))
coal <- as.numeric(bicoal)
coal_df <- data.frame(coal = coal, year = year)

# Set seed for reproducibility
set.seed(111)

# Define the number of iterations and degrees
iterations <- 100
degrees <- 1:8

# Prepare to plot all degrees on a single page
par(mfrow = c(3, 3), mar = c(4, 4, 2, 1))

# Perform repeated subsetting and fitting
for (degree in degrees) {
  rss_list <- numeric(iterations)  # Store RSS values for each iteration

  # Plot initialization for each degree
  plot(NULL, xlim = range(year), ylim = range(coal),
       main = sprintf("Fits for Polynomial Degree %d", degree),
       xlab = "Year", ylab = "Coal Production", type = 'n')

  for (i in 1:iterations) {
    # Randomly sample indices for training (80%) and validation (20%)
    train_indices <- sample(1:nrow(coal_df), size = 0.8 * nrow(coal_df), replace = FALSE)
    train_data <- coal_df[train_indices, ]
    valid_data <- coal_df[-train_indices, ]

    # Fit polynomial model
    model <- lm(coal ~ poly(year, degree), data = train_data)

    # Predict on validation set and calculate RSS
    predictions <- predict(model, newdata = valid_data)
    rss_list[i] <- sum((valid_data$coal - predictions)^2)

    # Add to plot with low opacity to see variability
    lines(year, predict(model, newdata = data.frame(year = year)), col = rgb(0, 0, 1, 0.1))  # Light bl
```

```r
  }

  # Add the actual data points and a bold red line for the overall fit
  points(coal ~ year, data = coal_df, pch = 19)
  abline(lm(coal ~ poly(year, degree), data = coal_df), col = "red", lwd = 2)

  # Output RSS summary for this degree
  cat(sprintf("Degree %d: Mean RSS = %f\n", degree, mean(rss_list)))
}
```

```
## Degree 1: Mean RSS = 64505.677355
```

```
## Warning in abline(lm(coal ~ poly(year, degree), data = coal_df), col = "red", :
## only using the first two of 3 regression coefficients
```

```
## Degree 2: Mean RSS = 66450.343736
```

```
## Warning in abline(lm(coal ~ poly(year, degree), data = coal_df), col = "red", :
## only using the first two of 4 regression coefficients
```

```
## Degree 3: Mean RSS = 72005.232792
```

```
## Warning in abline(lm(coal ~ poly(year, degree), data = coal_df), col = "red", :
## only using the first two of 5 regression coefficients
```

```
## Degree 4: Mean RSS = 62751.715523
```

```
## Warning in abline(lm(coal ~ poly(year, degree), data = coal_df), col = "red", :
## only using the first two of 6 regression coefficients
```

```
## Degree 5: Mean RSS = 60572.690302
```

```
## Warning in abline(lm(coal ~ poly(year, degree), data = coal_df), col = "red", :
## only using the first two of 7 regression coefficients
```

```
## Degree 6: Mean RSS = 57853.775300
```

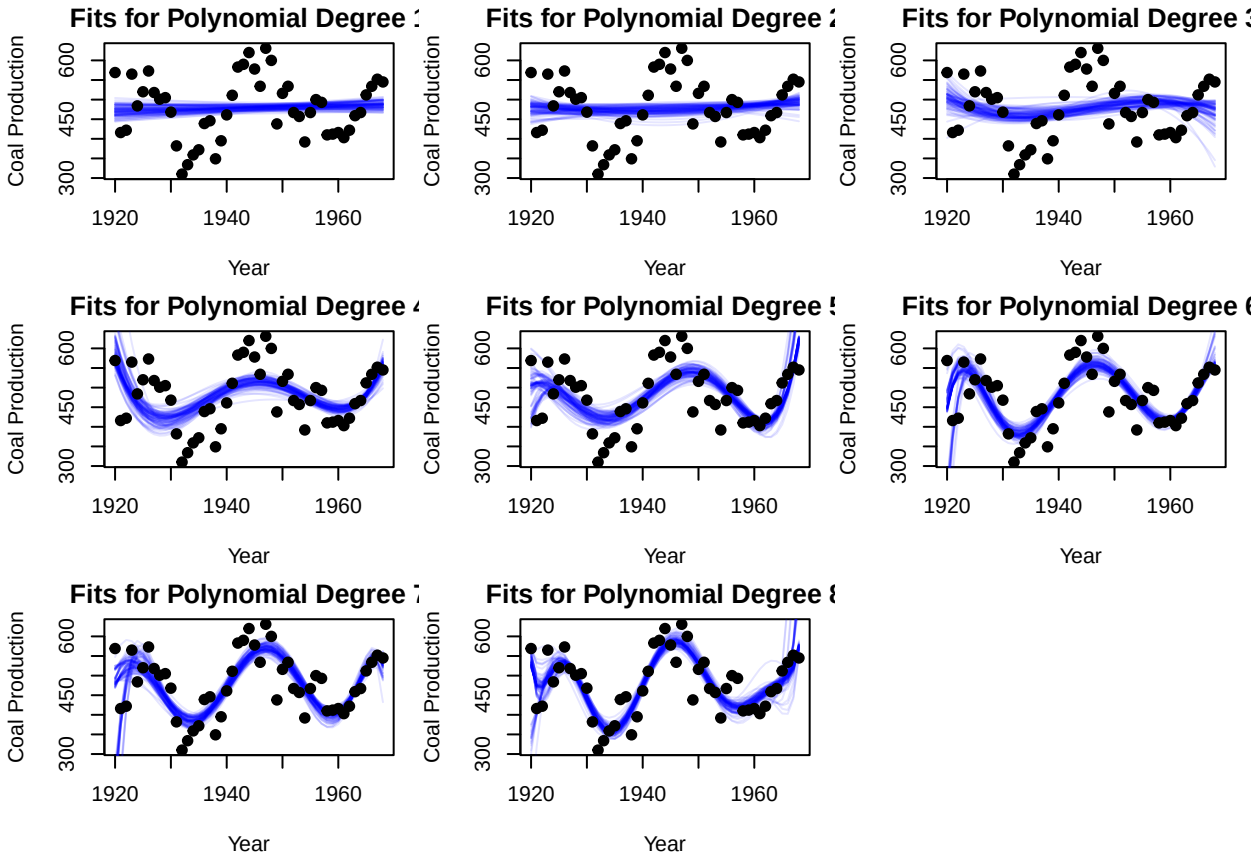
```
## Warning in abline(lm(coal ~ poly(year, degree), data = coal_df), col = "red", :
## only using the first two of 8 regression coefficients
```

```
## Degree 7: Mean RSS = 72858.529084
```

```
## Warning in abline(lm(coal ~ poly(year, degree), data = coal_df), col = "red", :
## only using the first two of 9 regression coefficients
```

```
## Degree 8: Mean RSS = 70144.095593
```

```r
# Reset plotting parameters
par(mfrow = c(1, 1), mar = c(5, 4, 4, 2))
```



The polynomial fits for degrees 1 to 8 indicate varying levels of model complexity. Low-degree models (1-3) are too simplistic, mostly capturing linear trends and failing to reflect the dataset's complexity, thus underfitting. Mid-degree models (4-6) improve fit by capturing more data fluctuations, but might still miss intricate details or slightly overfit. Higher degrees (7-8), especially 8, closely match individual data points, potentially overfitting by modeling noise as much as the underlying trend. Degrees 5 to 7 offer a better balance, with degree 7 visually appearing optimal. However, this should be confirmed through quantitative measures like RSS and cross-validation to ensure accuracy and generalizability beyond the training dataset.

Part (b): Cross-validation w/ best degree selection

```r
# define the test indices (first and last 5 years)
test_indices <- c(1:5, (length(year) - 4):length(year))

# define the remaining indices for training and validation
remaining_indices <- setdiff(1:nrow(coal_df), test_indices)

# prep variables to store RSS results
degrees <- 1:8
rss_list <- matrix(NA, nrow = 100, ncol = length(degrees))

# perform repeated subsetting and fitting for training and validation (cross-validation)
for (degree in degrees) {
  for (i in 1:100) {
    # split remaining indices into training (26) and validation (10)
```

```
    train_indices <- sample(remaining_indices, size = 26)
    valid_indices <- setdiff(remaining_indices, train_indices)

    # fit polynomial model on training set
    model <- lm(coal ~ poly(year, degree), data = coal_df, subset = train_indices)

    # predict on validation set and calculate RSS
    predictions <- predict(model, newdata = coal_df[valid_indices, ])
    rss_list[i, degree] <- sum((coal_df$coal[valid_indices] - predictions)^2)
  }
}

# compute median (!!!) RSS for each degree
rss_median <- apply(rss_list, 2, median)
best_degree <- which.min(rss_median)
cat("Best model degree based on validation RSS:", best_degree, "\n")
```

```
## Best model degree based on validation RSS: 8
```

```
# fit the best model to the entire remaining training data
best_model <- lm(coal ~ poly(year, best_degree), data = coal_df, subset = remaining_indices)

# predict on the test set
test_predictions <- predict(best_model, newdata = coal_df[test_indices, ])

# compute RSS on the test set
test_rss <- sum((coal_df$coal[test_indices] - test_predictions)^2)
cat("Test RSS for the best model (degree", best_degree, "):", test_rss, "\n")
```

```
## Test RSS for the best model (degree 8 ): 11420707
```

```
# compare predicted and actual values for the test set
test_results <- data.frame(Year = coal_df$year[test_indices],
                           Actual = coal_df$coal[test_indices],
                           Predicted = test_predictions)
print(test_results)
```

```
##    Year Actual Predicted
## 1  1920    569  591.6710
## 2  1921    416  465.0290
## 3  1922    422  441.5684
## 4  1923    565  465.8737
## 5  1924    484  502.6823
## 45 1964    467  586.6353
## 46 1965    512  847.0083
## 47 1966    534 1322.6430
## 48 1967    552 2125.3172
## 49 1968    545 3405.6648
```

According to this, the best model degree is 8 based on the median RSS. While the degree 8 polynomial model was optimal based on the training and validation sets, it appears to overfit the data when applied to the test set, especially for the later years. The model captures too much complexity (likely fitting noise), which leads to large prediction errors for the last 5 test years.