Building Prediction Models for PM2.5 Concentrations in the Continental U.S.

Jason Kim, Sina Saberi, Sarah Nguyen

Introduction: The goal of this research is to evaluate the performance of many predictors of ambient air pollution concentrations in the continental United States. This will be accomplished with a dataset containing annual average concentrations of fine particulate matter (PM2.5). Four regression prediction models will be built to determine how well the predictors explain the data and any variation that might be observed, primarily focusing on the outcome 'value'. Then, the performance of each of the models will be compared to one another to deduce which is the best and most fitting model, using the RMSE.

To first determine which predictor variables to use, the data was split into a training and testing dataset and the correlation values for the predictors were calculated. PCA was also used to find other relevant predictors. At the end of this analysis, we found that CMAQ, imp_a5000, and log_pri_length_15000 were the three predictor variables with the greatest correlations and will thus be used in the development of the models. CMAQ represents the concentration predictions from a numerical computer model developed by the EPA that simulates pollution in the atmosphere, imp_a5000 is the impervious surface measure within a circular radius of 5000 meters around the monitor, and log_pri_length_15000 is the count of primary road length in meters in a circular radius of 15000 meters around the monitor.

To investigate these predictor variables, the first predictor model we will build is the linear regression, with the second being the Poisson regression model. The third that we will build to analyze is the random Forest regression, and the fourth will be the multivariate adaptive regression splines (MARS) regression model.

The data will be split into a training dataset (contains a random 70% of the datapoints) and a testing dataset (contains the remaining random 30% of the datapoints). The main prediction metric that will be used to compare each of the models is the root mean-squared error (RMSE).

```
# Load tidyverse + tidymodels + randomForest + plotROC + earth packages and PM2.5 dataset
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.2     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.0
## v ggplot2   3.4.2     v tibble    3.2.1
## v lubridate 1.9.2     v tidyr     1.3.0
## v purrr     1.0.1
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
library(tidymodels)
```

```
## -- Attaching packages -------------------------------------- tidymodels 1.0.0 --
## v broom        1.0.4     v rsample      1.1.1
## v dials        1.2.0     v tune         1.1.1
```

```
## v infer        1.0.4     v workflows    1.1.3
## v modeldata    1.1.0     v workflowsets 1.0.1
## v parsnip      1.1.0     v yardstick    1.2.0
## v recipes      1.0.6
## -- Conflicts ----------------------------------------- tidymodels_conflicts() --
## x scales::discard() masks purrr::discard()
## x dplyr::filter()   masks stats::filter()
## x recipes::fixed()  masks stringr::fixed()
## x dplyr::lag()      masks stats::lag()
## x yardstick::spec() masks readr::spec()
## x recipes::step()   masks stats::step()
## * Use tidymodels_prefer() to resolve common conflicts.
```

```
library(randomForest)
```

```
## randomForest 4.7-1.1
## Type rfNews() to see new features/changes/bug fixes.
##
## Attaching package: 'randomForest'
##
## The following object is masked from 'package:dplyr':
##
##     combine
##
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
library(plotROC)
library(earth)
```

```
## Loading required package: Formula
## Loading required package: plotmo
## Loading required package: plotrix
##
## Attaching package: 'plotrix'
##
## The following object is masked from 'package:scales':
##
##     rescale
##
## Loading required package: TeachingDemos
```

```
dat <- read_csv("https://github.com/rdpeng/stat322E_public/raw/main/data/pm25_data.csv.gz")
```

```
## Rows: 876 Columns: 50
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (3): state, county, city
## dbl (47): id, value, fips, lat, lon, CMAQ, zcta, zcta_area, zcta_pop, imp_a5...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

Wrangling: Splitting Dataset into Training and Testing & Finding First Predictor

```
library(caret)
```

```
## Loading required package: lattice

##
## Attaching package: 'caret'

## The following objects are masked from 'package:yardstick':
##
##     precision, recall, sensitivity, specificity

## The following object is masked from 'package:purrr':
##
##     lift
```

```
# split the data into training and testing sets
set.seed(123)
trainIndex <- createDataPartition(dat$value, p = 0.7, list = FALSE, times = 1)
training <- dat[trainIndex, ]
testing <- dat[-trainIndex, ]

# compute correlations between predictors and outcome
correlations <- cor(training[, sapply(training, is.numeric)], training$value)

# find max correlation
max_correlation <- max(abs(correlations))

# check the correlation between CMAQ and the outcome value in the training set is high
cor(training$CMAQ, training$value)
```

```
## [1] 0.4425671
```

```
correlations
```

```
##                                    [,1]
## id                          -0.08075540
## value                        1.00000000
## fips                        -0.08075485
## lat                         -0.11324025
## lon                          0.17151802
## CMAQ                         0.44256711
## zcta                        -0.14796040
## zcta_area                   -0.27979728
## zcta_pop                     0.15083096
## imp_a500                     0.29119912
## imp_a1000                    0.29840534
## imp_a5000                    0.30203376
## imp_a10000                   0.26767481
## imp_a15000                   0.23871379
```

```
## county_area                  -0.15580328
## county_pop                    0.14580082
## log_dist_to_prisec           -0.23688814
## log_pri_length_5000           0.16628854
## log_pri_length_10000          0.19629390
## log_pri_length_15000          0.20965725
## log_pri_length_25000          0.24484357
## log_prisec_length_500         0.20938794
## log_prisec_length_1000        0.23004364
## log_prisec_length_5000        0.31576834
## log_prisec_length_10000       0.33487925
## log_prisec_length_15000       0.34788706
## log_prisec_length_25000       0.35703623
## log_nei_2008_pm25_sum_10000  0.35404781
## log_nei_2008_pm25_sum_15000  0.35096742
## log_nei_2008_pm25_sum_25000  0.37159084
## log_nei_2008_pm10_sum_10000  0.34526756
## log_nei_2008_pm10_sum_15000  0.34299883
## log_nei_2008_pm10_sum_25000  0.35408977
## popdens_county                0.13204661
## popdens_zcta                  0.13826389
## nohs                          0.14428584
## somehs                        0.16955860
## hs                            0.03846887
## somecollege                  -0.03839904
## associate                    -0.09844641
## bachelor                     -0.10840113
## grad                         -0.08689183
## pov                           0.13241658
## hs_orless                     0.14871915
## urc2013                      -0.23059117
## urc2006                      -0.24768355
## aod                           0.31959552
```

To wrangle with the dataset, we first split it into training and testing sub-datasets in order to properly evaluate the prediction metrics in an unbiased manner by using the createDataPartition() function from the caret library. We gave the training dataset random datapoints from 70% of the dataset, and the remaining 30% to the testing dataset.

Once we obtained our training dataset, we found the predictor with the highest correlation to the outcome value by using the cor() function, which was found to be *CMAQ*.

Finding Other Relevant Predictor(s)

```r
# select numeric columns for PCA
data_for_pca <- training[, sapply(training, is.numeric)]

# standardize data for PCA
scaled_data <- scale(data_for_pca)

# perform PCA
pca_result <- prcomp(scaled_data)

# extract values in PC1 column
pc1 <- pca_result$rotation[, 1]
```

```r
pc1
```

```
##                          id                      value
##                 0.009672336               -0.111102538
##                        fips                        lat
##                 0.009672324                0.017094260
##                         lon                       CMAQ
##                -0.077486779               -0.140375892
##                        zcta                  zcta_area
##                 0.065630879                0.106399461
##                    zcta_pop                   imp_a500
##                -0.081240205               -0.195684119
##                   imp_a1000                  imp_a5000
##                -0.202369108               -0.214806411
##                  imp_a10000                 imp_a15000
##                -0.208737856               -0.193194406
##                 county_area                 county_pop
##                 0.061018572               -0.103408664
##          log_dist_to_prisec        log_pri_length_5000
##                 0.115071549               -0.188043806
##        log_pri_length_10000       log_pri_length_15000
##                -0.207730866               -0.212322162
##        log_pri_length_25000      log_prisec_length_500
##                -0.211486125               -0.101124084
##      log_prisec_length_1000     log_prisec_length_5000
##                -0.126241490               -0.188016570
##     log_prisec_length_10000    log_prisec_length_15000
##                -0.211592960               -0.215469056
##     log_prisec_length_25000 log_nei_2008_pm25_sum_10000
##                -0.206769206               -0.184585449
## log_nei_2008_pm25_sum_15000 log_nei_2008_pm25_sum_25000
##                -0.188028050               -0.181632672
## log_nei_2008_pm10_sum_10000 log_nei_2008_pm10_sum_15000
##                -0.181117575               -0.185128110
## log_nei_2008_pm10_sum_25000             popdens_county
##                -0.178718905               -0.101232217
##                popdens_zcta                       nohs
##                -0.118882390               -0.051579145
##                      somehs                         hs
##                -0.053390074                0.044179807
##                 somecollege                  associate
##                 0.044861789                0.074752428
##                    bachelor                       grad
##                -0.030133493               -0.022038368
##                         pov                  hs_orless
##                -0.062474751               -0.010214317
##                     urc2013                    urc2006
##                 0.198796394                0.198994181
##                         aod
##                -0.126948518
```

In this section, we perform PCA to find other relevant predictors. From the PCA results, we decided to choose *imp_a5000* and *log_pri_length_15000*, which had the highest correlation values to PC1.

First Predictor Model: Linear Regression

```r
# Set up 10-fold cross-validation
trainControl <- trainControl(method = "cv", number = 10)

# Fit linear regression model using 10-fold cross-validation
linearRegression_model <- train(value ~ CMAQ + imp_a5000 + log_pri_length_15000,
                                data = training,
                                method = "lm",
                                trControl = trainControl)

# Print summary of the model
summary(linearRegression_model)
```

```
##
## Call:
## lm(formula = .outcome ~ ., data = dat)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -6.1034 -1.3425  0.0667  1.1583 12.3893
##
## Coefficients:
##                       Estimate Std. Error t value Pr(>|t|)
## (Intercept)           9.433568   1.162005   8.118 2.61e-15 ***
## CMAQ                  0.341328   0.032892  10.377  < 2e-16 ***
## imp_a5000             0.039676   0.008386   4.731 2.77e-06 ***
## log_pri_length_15000 -0.199119   0.111673  -1.783   0.0751 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.247 on 612 degrees of freedom
## Multiple R-squared:  0.2273, Adjusted R-squared:  0.2235
## F-statistic: 59.99 on 3 and 612 DF,  p-value: < 2.2e-16
```

```r
# Predict outcome value using linear regression model with testing dataset
predict(linearRegression_model, testing)
```

```
##         1         2         3         4         5         6         7         8
## 10.412730 11.321924 11.282548 10.443190 10.488639 10.571558 11.373348 10.162309
##         9        10        11        12        13        14        15        16
##  9.352310 11.184940 11.560522  9.888189 11.274928 11.271020 11.089903 11.752129
##        17        18        19        20        21        22        23        24
## 10.025941 10.805182 13.312437 12.279387  9.187413 11.065172  9.135692 13.024771
##        25        26        27        28        29        30        31        32
## 11.141596  9.720222  9.892181  8.686027 12.655701 11.342037  9.871614 12.279789
##        33        34        35        36        37        38        39        40
## 12.012487 12.126592 12.183796 11.743136 11.762212 11.947722 11.940352 11.623298
##        41        42        43        44        45        46        47        48
##  9.836574  9.279344  9.653093  9.593827  8.868138 11.992994 12.083683 10.492100
##        49        50        51        52        53        54        55        56
## 10.039636  9.525727  9.359565  9.266153  9.969252 10.140738 10.973029 11.204693
##        57        58        59        60        61        62        63        64
```

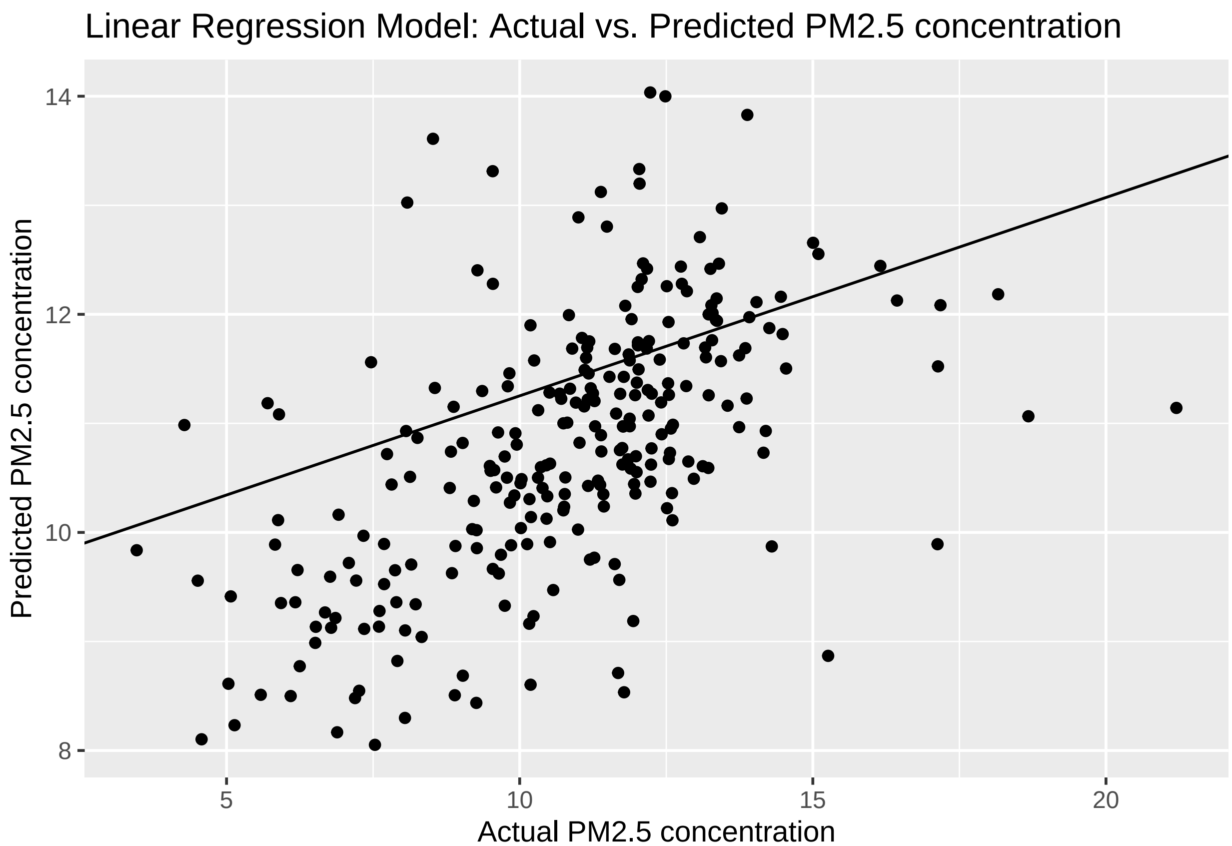
```
## 10.987420 10.770740 12.466800 10.509598 10.439282 11.324646 10.866989  9.558269
##        65        66        67        68        69        70        71        72
## 10.929714 11.272012 11.689504 10.965381 12.707844 12.212029 11.873583 11.261328
##        73        74        75        76        77        78        79        80
## 11.695882 10.585189 11.043113  8.232131  8.773158 13.332083 12.971479 13.198364
##        81        82        83        84        85        86        87        88
## 11.296683 11.818732 11.272299 12.417906 10.598571 11.225387 11.503088 11.071887
##        89        90        91        92        93        94        95        96
## 13.828860 14.034290 13.999151 11.695289 11.305921 12.553444 12.417332 10.406892
##        97        98        99       100       101       102       103       104
## 10.504135 10.238121 10.673026 10.696036  9.855622  9.882266 10.916848 10.338818
##       105       106       107       108       109       110       111       112
## 11.576767 11.577496 10.698647 11.715430 10.465183 11.605633 11.570436 10.973413
##       113       114       115       116       117       118       119       120
## 13.122091 12.403582 10.501960 13.609669  8.510906 10.754070 11.257802 12.257920
##       121       122       123       124       125       126       127       128
## 11.754494 11.367267 10.020532 10.351757 11.458842 10.501951  9.622761  9.471419
##       129       130       131       132       133       134       135       136
##  8.534228 11.218461 10.909368  9.134539 11.685547 11.955840 11.190359 12.464544
##       137       138       139       140       141       142       143       144
## 12.078222 11.120387 11.155555 10.564944 11.426480 10.607175 10.900001 10.970817
##       145       146       147       148       149       150       151       152
##  9.892664 11.006521 11.929435  8.506936  8.436707  8.167074  8.052110  8.298652
##       153       154       155       156       157       158       159       160
##  8.480802  8.547779 10.740580  9.893993 10.029961  9.125524  8.499654 10.288918
##       161       162       163       164       165       166       167       168
## 12.322393 12.437770 11.683054 11.489810 10.450854 11.317719  9.626393 11.899512
##       169       170       171       172       173       174       175       176
## 11.585094 10.426465 11.082546  9.413396  9.655067 10.984769 10.112552  9.557433
##       177       178       179       180       181       182       183       184
##  9.705769  9.664960 12.889221 12.251143  8.821338 10.553053 10.222035  9.564826
##       185       186       187       188       189       190       191       192
##  9.911486 10.615217  9.709590 10.973429 10.891585 10.650227  9.750841 10.742282
##       193       194       195       196       197       198       199       200
##  8.102969  9.215310  9.041402 11.973904 11.733859 11.999478 12.111532 11.783627
##       201       202       203       204       205       206       207       208
## 12.083520 12.161035 10.435760 10.773814 12.145598 10.591132  9.795108 11.000629
##       209       210       211       212       213       214       215       216
##  8.603448  8.711108 10.718368  9.327589 10.349688 11.522583 11.192183 11.161975
##       217       218       219       220       221       222       223       224
## 11.227863 11.258605 10.822470 10.360309 11.340260 10.820518 10.609278 10.730510
##       225       226       227       228       229       230       231       232
## 10.621930 11.686192 10.670219 10.356189  9.875512 10.474798 10.952544 12.804316
##       233       234       235       236       237       238       239       240
## 11.632131 12.444119 11.457537 11.151962  9.231951  9.101365  8.987034  9.359593
##       241       242       243       244       245       246       247       248
##  9.341117 11.494948 10.630605 11.601101 10.125748 11.426925 10.331373 10.305362
##       249       250       251       252       253       254       255       256
## 10.407673 10.272527  9.115162 10.730637 10.930408 10.111288 10.202292 10.622801
##       257       258       259       260
##  9.767958 10.234994  9.162362  8.612263
```

```
# Calculate RMSE using the 10-fold cross-validation results
linear_RMSE1 <- sqrt(linearRegression_model$results$RMSE)
```

```r
# Alternatively, you can calculate RMSE using the testing dataset
linear_RMSE <- testing %>%
  mutate(pred = predict(linearRegression_model, testing)) %>%
  summarize(rmse = sqrt(mean(value - pred)^2))

linear_RMSE
```

```
## # A tibble: 1 x 1
##      rmse
##     <dbl>
## 1 0.00606
```

```r
# Create scatterplot of predicted vs. actual values
testing %>%
  mutate(pred = predict(linearRegression_model, testing)) %>%
  ggplot(aes(x = value, y = pred)) +
  geom_point() +
  geom_abline(intercept = linearRegression_model$finalModel$coefficients[1], slope = linearRegression_mo
  labs(x = "Actual PM2.5 concentration", y = "Predicted PM2.5 concentration",
       title = "Linear Regression Model: Actual vs. Predicted PM2.5 concentration")
```



In this section, we created our first predictor model, using linear regression, to predict our outcome value using the testing dataset with our chosen predictors. We calculated a residual standard error of 2.247. a multiple r-squared value of 0.2273, and an adjusted r-squared value of 0.2235. Because the residual standard error is rather large and the r-squared values are close to 0, this would indicate that the model does not perform very well. However, once we predicted the outcome values for the PM2.5 concentrations using the testing dataset, we calculated the root mean-squared error and found a value of 0.00606. This value is less than 2 micrograms per meters cubed, suggesting that it is good for the model or that the linear regression is a good fit for the testing dataset.

Second Predictor Model: Poisson Regression

```
# Create a new column called rounded_value by rounding the value column
training$rounded_value <- round(training$value)

# Set up 10-fold cross-validation
trainControl <- trainControl(method = "cv", number = 10)

# Fit Poisson regression model using 10-fold cross-validation
poisson_model_cv <- train(rounded_value ~ imp_a5000 + log_pri_length_15000 + CMAQ,
                          data = training,
                          method = "glm",
                          family = poisson(),
                          trControl = trainControl)

# Print summary of the model
summary(poisson_model_cv)
```

```
##
## Call:
## NULL
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.3772  -0.4118   0.0293   0.3881   3.2668
##
## Coefficients:
##                       Estimate Std. Error z value Pr(>|z|)
## (Intercept)           2.241575   0.159107  14.088  < 2e-16 ***
## imp_a5000             0.003511   0.001121   3.131  0.00174 **
## log_pri_length_15000 -0.017505   0.015238  -1.149  0.25064
## CMAQ                  0.031300   0.004392   7.126 1.03e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 383.97  on 615  degrees of freedom
## Residual deviance: 301.51  on 612  degrees of freedom
## AIC: 2899.1
##
## Number of Fisher Scoring iterations: 4
```

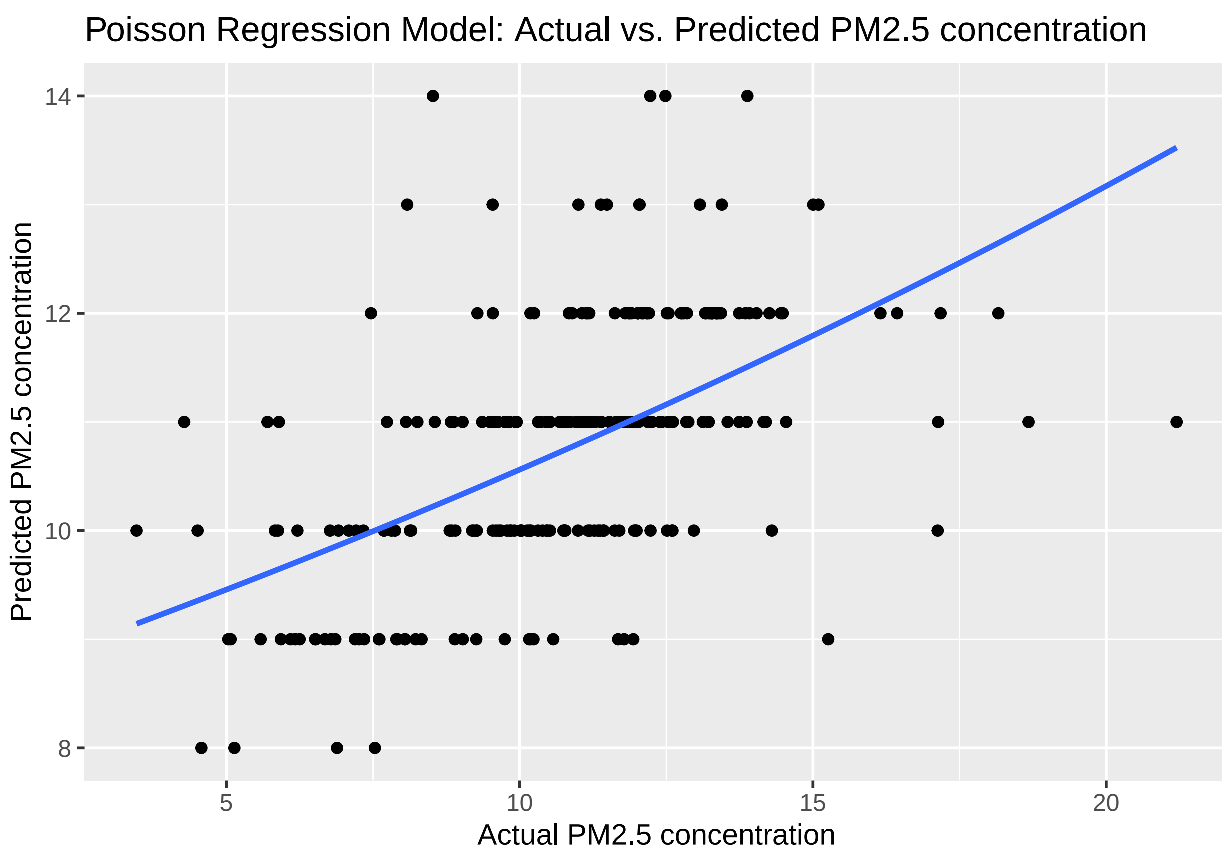
```
# Predict outcome value using Poisson regression model with testing dataset
poisson_pred <- predict.train(poisson_model_cv, newdata = testing)

# Convert predicted values to integer
poisson_pred <- as.integer(round(poisson_pred))
```

```r
# Calculate RMSE using the testing dataset
poisson_RMSE <- testing %>%
  mutate(pred = poisson_pred) %>%
  summarize(rmse = sqrt(mean((value - pred)^2)))

poisson_RMSE
```

```
## # A tibble: 1 x 1
##    rmse
##   <dbl>
## 1  2.26
```

```r
# Create scatterplot of predicted vs. actual values
poisson_df <- data.frame(actual = testing$value, pred = poisson_pred)
ggplot(poisson_df, aes(x = actual, y = pred)) +
  geom_point() +
  stat_smooth(method = "glm", method.args = list(family = "poisson"), se = FALSE) +
  labs(x = "Actual PM2.5 concentration", y = "Predicted PM2.5 concentration",
       title = "Poisson Regression Model: Actual vs. Predicted PM2.5 concentration")
```

```
## `geom_smooth()` using formula = 'y ~ x'
```



Here, we create a Poisson model to predict the outcome value (annual average PM2.5 concentration). Once we calculated the RMSE for the testing dataset, we found a value of 2.26, which is moderately high; thus, we observe that the model may be improved by taking into account more predictors.

Third Predictor Model: Random Forest

```r
# Load necessary packages
library(caret)
library(randomForest)

# Set up 10-fold cross-validation
train_control <- trainControl(method = "cv", number = 10)

# Fit random forest model using 10-fold cross-validation
randomForest_model <- train(value ~ CMAQ + imp_a5000 + log_pri_length_15000, data = training,
                             method = "rf", trControl = train_control)
```

```
## note: only 2 unique complexity parameters in default grid. Truncating the grid to 2 .
```

```r
# Print summary of random forest model
print(randomForest_model)
```

```
## Random Forest
##
## 616 samples
##   3 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 555, 554, 555, 553, 554, 554, ...
## Resampling results across tuning parameters:
##
##   mtry  RMSE      Rsquared   MAE
##   2     1.975385  0.3925156  1.437767
##   3     1.984527  0.3896584  1.439503
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was mtry = 2.
```

```r
# Predict outcome value using random forest model with testing dataset
predict(randomForest_model, testing)
```

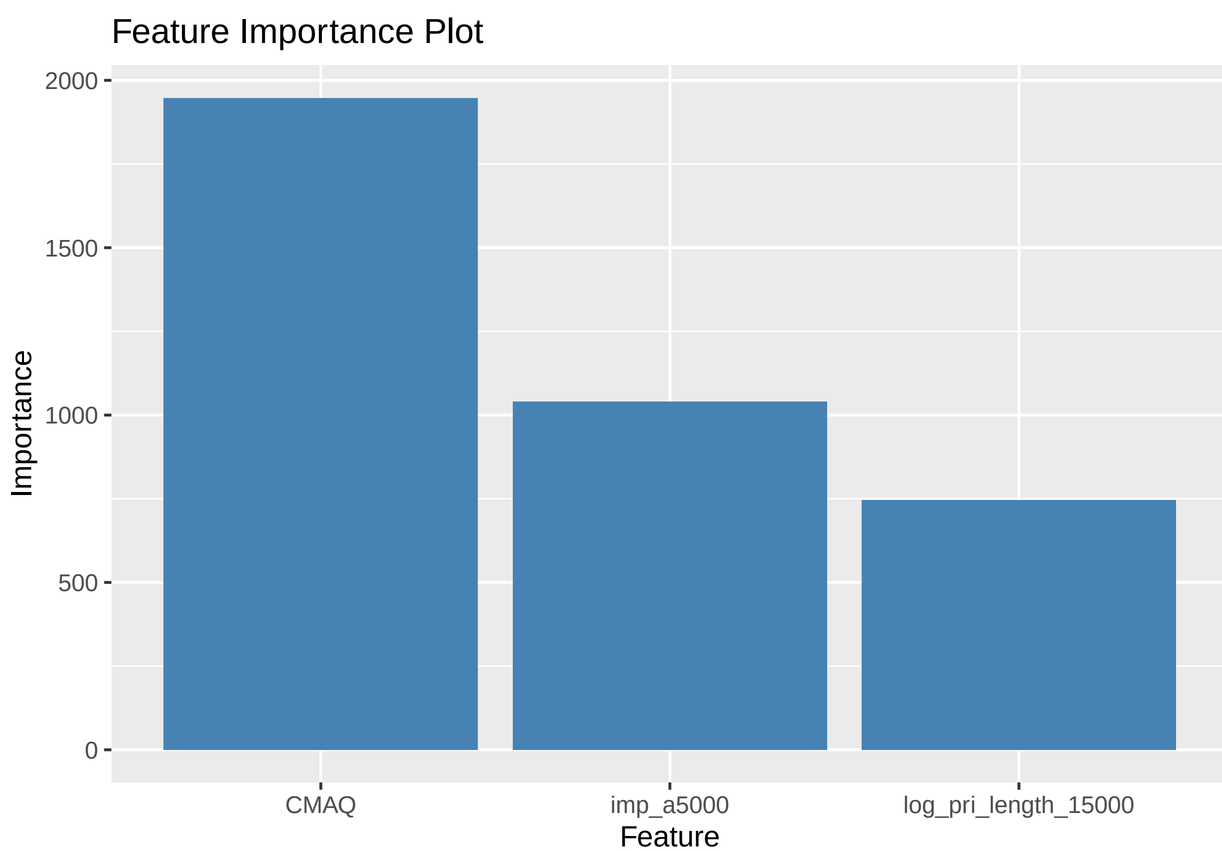
```
##         1         2         3         4         5         6         7         8
## 11.037312 11.407724 11.569972 12.037241 11.455891 11.594714 11.469387  8.233955
##         9        10        11        12        13        14        15        16
##  7.907959 11.583830 11.508724 11.262646 11.766175 11.008298 10.964873 10.785285
##        17        18        19        20        21        22        23        24
##  9.932623 12.326347 13.211992 12.386726 10.112927 15.967361  8.154861 10.269694
##        25        26        27        28        29        30        31        32
## 19.055724  8.074664  9.914251 10.068622 13.007956 11.805546 10.436524 11.503144
##        33        34        35        36        37        38        39        40
## 12.095714 12.042673 12.190475 12.273402 12.253169 12.771796 12.263288 11.989623
##        41        42        43        44        45        46        47        48
## 10.134075  8.867953  8.647543  7.613675  7.529423 12.295935 14.402533  9.624923
##        49        50        51        52        53        54        55        56
##  9.059674  8.704082  9.200931  8.536317  9.155288 10.734628 12.128988 11.232180
```

```
##        57        58        59        60        61        62        63        64 
## 11.708230  9.159290 11.934911  9.639579 11.029214 12.438737 15.443575  8.852579 
##        65        66        67        68        69        70        71        72 
## 11.675607 11.975596 12.530529 12.649309 11.607083 11.385813 13.019994 11.828385 
##        73        74        75        76        77        78        79        80 
## 10.714076 11.045945 11.130062  6.920998  5.429914 12.646526 11.872945 12.075608 
##        81        82        83        84        85        86        87        88 
## 11.035068 13.354003 11.417949 12.612677 10.956178 10.744139 12.218250 12.242380 
##        89        90        91        92        93        94        95        96 
## 12.427413 12.452418 11.857758  9.893731 11.150584 12.848303 12.908429 10.489789 
##        97        98        99       100       101       102       103       104 
## 11.550341 11.110305 11.687184 10.485141 10.546633  8.518216 10.106868 10.257384 
##       105       106       107       108       109       110       111       112 
## 11.476237 12.151426 10.891194 10.764624 11.464663 12.050683 12.086930 11.199262 
##       113       114       115       116       117       118       119       120 
## 10.878432 12.168146 12.034052 10.126065  9.070789 10.954969 11.735524 11.259301 
##       121       122       123       124       125       126       127       128 
## 12.265827 12.681731 10.308670 11.243831 11.343559 10.578800 10.421578 11.942339 
##       129       130       131       132       133       134       135       136 
##  6.062612 11.255793 11.618093  5.422996 11.019271 10.801532 10.755876 12.497530 
##       137       138       139       140       141       142       143       144 
## 10.920890 10.601431 10.750269 11.425153 11.855181 11.101664 11.910353 11.630571 
##       145       146       147       148       149       150       151       152 
##  9.503695  9.946013 13.263467  9.012426 10.541170  7.705783  8.184589  9.795459 
##       153       154       155       156       157       158       159       160 
##  6.694045  7.735336 10.793480  8.721942 10.531690  8.439056  9.002647  9.676677 
##       161       162       163       164       165       166       167       168 
## 12.624106 12.566007 12.618262  9.814031 11.711555 11.253463 10.169846 12.042119 
##       169       170       171       172       173       174       175       176 
## 12.238528 11.162961 11.094490  9.763400  8.495815 10.432034  9.465354  8.331798 
##       177       178       179       180       181       182       183       184 
##  7.492129  8.267018 12.857059 11.903364  7.742305 11.442336 11.277237  9.915178 
##       185       186       187       188       189       190       191       192 
## 10.146355 10.793796 10.442695 12.324711 11.752701 11.565979  9.524192 10.367747 
##       193       194       195       196       197       198       199       200 
##  7.451047  7.896478  7.734870 11.367330 10.942326 13.183744 13.074753 12.578106 
##       201       202       203       204       205       206       207       208 
## 12.120232 12.908555 12.143551 11.162771 11.052799 12.125604  8.792690 12.096883 
##       209       210       211       212       213       214       215       216 
##  8.159747  5.665878 10.308095  8.661825 10.884076 11.876404 11.459884 12.377638 
##       217       218       219       220       221       222       223       224 
## 10.198047 11.310116 11.865284 11.615644 10.552592  9.880198 11.856158 11.081627 
##       225       226       227       228       229       230       231       232 
## 11.342897 12.569528 11.735490 11.333377  7.925169 10.356298 10.059785 11.777543 
##       233       234       235       236       237       238       239       240 
## 10.592498 10.408705 10.803055 12.033662  8.485362  7.947266  8.551216  7.890754 
##       241       242       243       244       245       246       247       248 
##  8.552914 11.982568 11.814487 13.033647 11.024498 11.933659 10.981774 11.505685 
##       249       250       251       252       253       254       255       256 
##  8.262160  9.497967 10.464474 11.537357 13.196951  9.933371 11.256600  9.760184 
##       257       258       259       260 
## 10.008541 10.609806  4.913215  4.976280 
```

```r
# Calculate RMSE
randomForest_RMSE <- sqrt(mean((predict(randomForest_model, testing) - testing$value)^2))

randomForest_RMSE
```

```
## [1] 2.078261
```

```r
# Plot feature importance
importance <- varImp(randomForest_model$finalModel)
ggplot(importance, aes(x = row.names(importance), y = Overall)) +
  geom_bar(stat = "identity", fill = "steelblue") +
  ggtitle("Feature Importance Plot") +
  xlab("Feature") +
  ylab("Importance")
```



In this section, we created a randomForest model to predict the outcome value (annual average PM2.5 concentration). Once we calculated the RMSE for the testing dataset, we found a value of 2.07925, which is mildly highly and could possibly be improved by having larger training datasets and taking into account more predictors.

Fourth Predictor Model: Multivariate Adaptive Regression Splines (MARS)

```r
# Create a grid of tuning parameters to search over
mars_grid <- expand.grid(degree = 1:2, nprune = 2:5)

# Fit MARS model to predict annual average PM2.5 concentration using 10-fold cross-validation
```

```r
set.seed(123)
mars_model_cv <- train(value ~ CMAQ + imp_a5000 + log_pri_length_15000,
                       data = training,
                       method = "earth",
                       trControl = trainControl(method = "cv", number = 10, verboseIter = TRUE),
                       tuneGrid = mars_grid)
```

```
## + Fold01: degree=1, nprune=5
## - Fold01: degree=1, nprune=5
## + Fold01: degree=2, nprune=5
## - Fold01: degree=2, nprune=5
## + Fold02: degree=1, nprune=5
## - Fold02: degree=1, nprune=5
## + Fold02: degree=2, nprune=5
## - Fold02: degree=2, nprune=5
## + Fold03: degree=1, nprune=5
## - Fold03: degree=1, nprune=5
## + Fold03: degree=2, nprune=5
## - Fold03: degree=2, nprune=5
## + Fold04: degree=1, nprune=5
## - Fold04: degree=1, nprune=5
## + Fold04: degree=2, nprune=5
## - Fold04: degree=2, nprune=5
## + Fold05: degree=1, nprune=5
## - Fold05: degree=1, nprune=5
## + Fold05: degree=2, nprune=5
## - Fold05: degree=2, nprune=5
## + Fold06: degree=1, nprune=5
## - Fold06: degree=1, nprune=5
## + Fold06: degree=2, nprune=5
## - Fold06: degree=2, nprune=5
## + Fold07: degree=1, nprune=5
## - Fold07: degree=1, nprune=5
## + Fold07: degree=2, nprune=5
## - Fold07: degree=2, nprune=5
## + Fold08: degree=1, nprune=5
## - Fold08: degree=1, nprune=5
## + Fold08: degree=2, nprune=5
## - Fold08: degree=2, nprune=5
## + Fold09: degree=1, nprune=5
## - Fold09: degree=1, nprune=5
## + Fold09: degree=2, nprune=5
## - Fold09: degree=2, nprune=5
## + Fold10: degree=1, nprune=5
## - Fold10: degree=1, nprune=5
## + Fold10: degree=2, nprune=5
## - Fold10: degree=2, nprune=5
## Aggregating results
## Selecting tuning parameters
## Fitting nprune = 4, degree = 2 on full training set
```

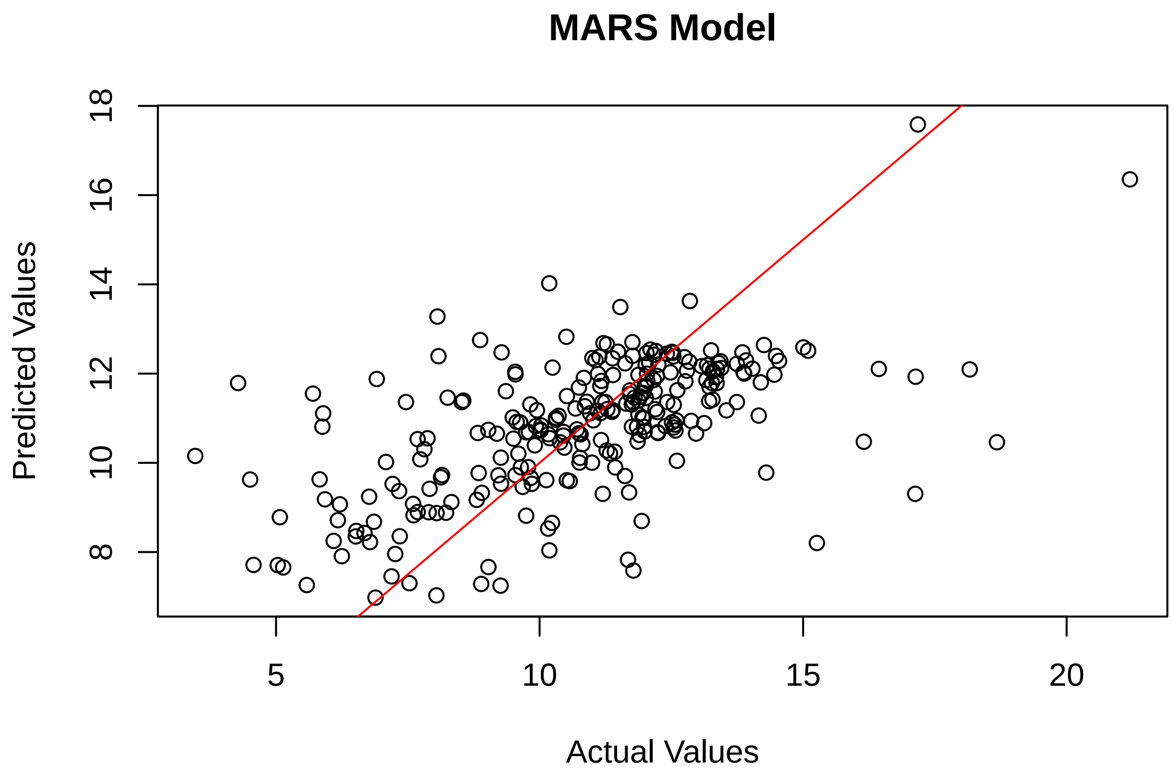
```r
# Print summary of MARS model
summary(mars_model_cv)
```

```
## Call: earth(x=matrix[616,3], y=c(10.8,11.66,12...), keepxy=TRUE, degree=2,
##             nprune=4)
##
##                                                         coefficients
## (Intercept)                                               11.3618259
## h(10.3725-CMAQ)                                           -0.5089035
## h(log_pri_length_15000-11.1009)                            0.5632170
## h(imp_a5000-6.18832) * h(11.727-log_pri_length_15000)     0.0975428
##
## Selected 4 of 19 terms, and 3 of 3 predictors (nprune=4)
## Termination condition: Reached nk 21
## Importance: CMAQ, imp_a5000, log_pri_length_15000
## Number of terms at each degree of interaction: 1 2 1
## GCV 4.560121    RSS 2732.047    GRSq 0.2995645    RSq 0.3165442
```

```r
# Predict outcome value using MARS model with testing dataset
mars_pred <- predict(mars_model_cv, testing)

# Calculate RMSE using the testing dataset
mars_RMSE <- testing %>%
  mutate(pred = mars_pred) %>%
  summarize(rmse = sqrt(mean((value - pred)^2)))

mars_RMSE
```

```
## # A tibble: 1 x 1
##    rmse
##   <dbl>
## 1  2.15
```

```r
# Plot predicted values against actual values
plot(testing$value, mars_pred, xlab = "Actual Values", ylab = "Predicted Values", main = "MARS Model")

# Add a diagonal reference line
abline(a = 0, b = 1, col = "red")
```



In this section, we create our fourth predictor model, using Multivariate Adaptive Regression Splines (MARS). The RMSE value comes out to be 2.152469, which is moderately high; thus, we observe that this model may be improved by taking into account more predictors, or using larger training datasets.

Results:

```
# create a data frame of RMSE values
RMSE_data <- data.frame(Model = c("Linear Regression", "Poisson Regression", "MARS", "Random Forest"),
                        RMSE = c(linear_RMSE$rmse, poisson_RMSE$rmse, mars_RMSE$rmse, randomForest_RMSE

# sort the data frame by RMSE values in ascending order
RMSE_data_sorted <- RMSE_data %>% arrange(RMSE)

# print the sorted data frame
RMSE_data_sorted
```

```
##                Model        RMSE
## 1  Linear Regression 0.006058944
## 2      Random Forest 2.078260975
## 3               MARS 2.152468709
## 4 Poisson Regression 2.257295388
```

The development of the four prediction models involved creating a training and testing dataset, fitting each model to the training dataset, and evaluating their performance using the testing dataset.

In splitting the data into training and testing sets, we used the createDataPartition function, with a ratio of 70:30, which randomly splits the dataset into two sets based on the specified ratio. The seed was set to 123 to ensure reproducibility of the results.

The first model developed was a linear regression model. To evaluate the performance of this model, we used 10-fold cross-validation. The *trainControl()* function helped to set up the cross-validation, with the method set to "cv" and the number of folds set to 10. Then we used the *train()* function to fit the model to the training dataset, setting the method = "lm". The predict function was used to predict the outcome value using the testing dataset. Using the root mean squared error (RMSE), which was calculated using the cross-validation results or the testing dataset, the performance of the model was evaluated. Finally, we created a scatterplot of predicted vs. actual values to visualize the predictor model.

The second model developed was a Poisson regression model. Before fitting the model, a new column called "rounded_value" was created by rounding the value column. This was necessary because the Poisson regression model assumes that the response variable is a count variable. To evaluate the performance of this model, 10-fold cross-validation was used. The *trainControl()* function was then used to set up the cross-validation, with the method set to "cv" and the number of folds set to 10. We then fit the model to the training dataset using the *train()* function, with the method set to "glm" and the family set to "poisson". Furthermore, we used the *predict.train()* function to predict the outcome value using the testing dataset. Finally, the performance of the model was evaluated using RMSE, which was calculated using the testing dataset, and a scatterplot of predicted vs. actual values was also created to help visualize the model.

The third model developed was a MARS (Multivariate Adaptive Regression Splines) model. A grid of tuning parameters was created using the *expand.grid()* function, with the degree set to 1 or 2 and the nprune set to 2, 3, 4, or 5. To evaluate the performance of this model, 10-fold cross-validation was used. The trainControl function was used to set up the cross-validation, with the method set to "cv" and the number of folds set to 10. The train function was then used to fit the model to the training dataset, with the method set to "earth" and the tuneGrid argument set to the created grid of tuning parameters. The predict function was used to predict the outcome value using the testing dataset. Finally, we evaluated the performance of our model using RMSE, which was calculated using the testing dataset, and created a scatterplot of predicted vs. actual values.

The fourth model developed was a random forest model. To evaluate the performance of this model, 10-fold cross-validation was used. The trainControl function was used to set up the cross-validation, with the method set to "cv" and the number of folds set to 10. The train function was then used to fit the model to the training dataset, with the method set to "rf". The predict function was used to predict the outcome value using the testing dataset. The performance of the model was evaluated using RMSE, which was calculated using the cross-validation results. A scatterplot of predicted vs. actual values was also created.

After developing our four models, we created a table displaying each of their respective RMSE values for comparison. As a result, we observed that our Linear Regression Model had the lowest RMSE value of 0.0061, thus proving to be the best fit model.

Discussion & Policy Questions: Policy Question 1: For a similar set of actual values, their predicted concentrations are also similar and clustered most prominently at x-values (actual concentrations) 10-15 micrograms per cubic meter and y-values (predicted concentrations) 10-12 micrograms per cubic meter. We suspect that the performance is good at these locations because if the model's predictions are closest to the observed values in areas with high levels of CMAQ and imp_a5000, it could suggest that those variables are relevant predictors of PM2.5 concentrations at those locations. On the contrary, if the model's predictions are farthest from the observed values in areas with low levels of log_pri_length_15000, it could suggest that this variable is not as relevant for predicting PM2.5 concentrations at those locations. It is also possible that other variables or factors that are not included in the model could be contributing to the good or bad performance at certain locations. For example, if there are sources of PM2.5 pollution in a particular area that are not captured by the variables in the model, this could lead to poorer performance of the model in predicting PM2.5 concentration in that location.

Policy Question 2: The weather for different regions could vary and we hypothesize that during rainfall and precipitation, air pollution can be removed from the atmostphere , while during drier weather, air pollution can accumulate. Another factor to consider is measurement error. The model may perform worse in areas where there is more measurement error or where monitoring data is sparse. When less data is accumulated in a specific region, the variation for the data could be greater. Finally, weather and time of day, variables that are not included in the dataset, are possible confounding factors that may improve the model performance if they were included.

Policy Question 3:

```
# Add AOD predictor to the model
linearRegression_model_AOD <- train(value ~ CMAQ + imp_a5000 + log_pri_length_15000 + aod,
                                    data = training,
                                    method = "lm",
                                    trControl = trainControl)

# Remove AOD and CMAQ predictors from model
linearRegression_model_without <- train(value ~ imp_a5000 + log_pri_length_15000,
                                    data = training,
                                    method = "lm",
                                    trControl = trainControl)

# Calculate respectifve RMSE values for modified models
linear_RMSE_AOD <- sqrt(linearRegression_model_AOD$results$RMSE)
linear_RMSE_without <- sqrt(linearRegression_model_without$results$RMSE)

# Compare the RMSE values between original, with, and without AOD/CMAQ
linear_RMSE1
```

```
## [1] 1.496037
```

```
linear_RMSE_AOD
```

```
## [1] 1.487763
```

```
linear_RMSE_without
```

```
## [1] 1.559086
```

With AOD as an added variable, the output value decreases very slightly in comparison to the original. Without AOD and CMAQ as added variables, the output value increases slightly. Thus, we can assume that using aod and CMAQ as predictor variables helps to make the linear regression model a slightly better predictor model.

Policy Question 4: Alaska and Hawaii are more far removed from the industrialization and globalization that has taken place in the continental United States. As a result, they likely experience less pollution and less contaminants in their air. Additionally, Alaska and Hawaii have different geographic features than the contiguous United States, such as mountains, volcanoes, and ocean currents, that can affect air pollution patterns. Therefore, our model might not be able to capture the unique air pollution patterns in these states. Furthermore, Hawaii's economy is based less on industry and more on tourism, thus having its PM2.5 value being possibly lower than expected based on models developed in contiguous United States. As for Alaska, however, the state has a significant oil and gas industry that may also impact air quality.

We each found the most challenging part of this project to be finding which regression models to use. Each model has its own unique presentation, and at times, it appeared that our data and the functions we used were not compatible with the model and we had to backtrack and see if another model might be a better choice. It was also rather difficult trying to determine what kind of visualizations to create and how to get the code to work to produce the visualizations. We especially struggled with creating an ROC plot at some point in the project. We learned from this process as a whole that several models have been developed in the data science field, and they each have their own individualized purposes when it comes to predicting data. Each model may work better or worse with certain datasets, depending on what kind of data they contain. Overall, this research assignment was very insightful.

Group Acknowledgements: Jason was responsible for creating outlines for much of the coding, while Sina and Sarah assisted with proof-reading the code, interpreting the results, visualizations, and any computed values such as the RMSE. We each spent time together on every model ensuring that the model-building and its accompanying RMSE computation was accurate. Sarah drafted a good portion of the introduction and Sina contributed largely to the discussions. We each answered the policy questions together and successfully divided the work evenly!