Project 2

2023-11-28

Predicting PM2.5 Concentration Across the Continental US

Mindy Li, Mayce Hoang, Carlos Vazquez

Introduction

Fine particulate matter, known as PM2.5, represents a critical environmental concern due to its potential adverse effects on human health and the environment. Understanding the spatial and temporal variations in PM2.5 concentrations is crucial for effective air quality management and informed decision-making. In this report, we aim to model and evaluate annual average concentrations of PM2.5 across the continental U.S. using the U.S. Environmental Protection Agency's monitoring network data. The primary goal is to compare (four) modeling approaches: Linear Regression, k-Nearest Neighbors, Regression Tree and Random Forest and analyze which is the 'best' approach in predicting PM2.5 concentrations.

The Linear Regression model provides a straightforward and interpretable framework, allowing us to understand the linear relationships between predictor variables and PM2.5 concentrations. The kNN model allows us to capture local variations and patterns in the dataset. The Regression Tree model provides a hierarchical structure to understand the influence of various predictor variables on PM2.5 concentrations. The Random Forest model extends the concept of decision trees by aggregating multiple trees.

Each methodology brings its unique strengths to the forefront, allowing us to discern the most suitable framework for capturing the complexities inherent in PM2.5 concentration patterns.

Predictor Variables

In our modeling process, the selection of predictor variables was a strategic decision aimed at capturing key dimensions of air quality while maintaining model simplicity and interpretability. AOD, log distance to primary sector (log_dist_to_prisec), CMAQ, and poverty (pov) constitute the core predictors utilized across our models.
-**CMAQ:** Chosen for its comprehensive, physics-based approach, providing a holistic perspective on air pollution dynamics without direct reliance on PM2.5 gravimetric data.
-**AOD:** Derived from satellite observations, offering a unitless measure for seamless integration with other variables, providing an external perspective on particulate pollution.
-**Log_dist_to_prisec:** Represents proximity to major roads, acknowledging the non-linear relationship between distance and pollution impact through natural log transformation.
-**Pov:** Includes poverty rates to address potential disparities in air quality impact on vulnerable populations, considering socioeconomic factors.
This streamlined set of predictors enhances our model's adaptability to capture nuanced air quality dynamics across diverse geographical and contextual settings.

Wrangling and Exploratory Analysis

Here, the data is read in and subsequently split into two parts: the training set and the testing set. The training set contains 80% of the data and the other 20% is delegated to the test set.

```r
# read data in
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   3.4.3     v tibble    3.2.1
## v lubridate 1.9.2     v tidyr     1.3.0
## v purrr     1.0.2
## -- 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
```

```r
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.
```

```r
# creating the training and testing sets
library(tidymodels)
```

```
## Warning: package 'tidymodels' was built under R version 4.3.2
```

```
## -- Attaching packages --------------------------------------- tidymodels 1.1.1 --
```

```
## v broom        1.0.5     v rsample      1.2.0
## v dials        1.2.0     v tune         1.1.2
## v infer        1.0.5     v workflows    1.1.3
## v modeldata    1.2.0     v workflowsets 1.0.1
## v parsnip      1.1.1     v yardstick    1.2.0
## v recipes      1.0.8
```

```
## Warning: package 'dials' was built under R version 4.3.2
```

```
## Warning: package 'infer' was built under R version 4.3.2
```

```
## Warning: package 'modeldata' was built under R version 4.3.2
```

```
## Warning: package 'parsnip' was built under R version 4.3.2
```

```
## Warning: package 'recipes' was built under R version 4.3.2
```

```
## Warning: package 'rsample' was built under R version 4.3.2
```

```
## Warning: package 'tune' was built under R version 4.3.2

## Warning: package 'workflows' was built under R version 4.3.2

## Warning: package 'workflowsets' was built under R version 4.3.2

## Warning: package 'yardstick' was built under R version 4.3.2

## -- 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.
```

```r
# for reproducibility
set.seed(123)

dat_split = initial_split(dat, prop = 0.8)
train_set = training(dat_split)
test_set = testing(dat_split)
```

In preparation for our predictive modeling of PM2.5 concentrations, we conducted a comprehensive exploratory analysis to understand the relationships within our chosen predictor variables: CMAQ, aod, log_dist_to_prisec, and pov.

We first explored the relationships between our predictor variables through a correlation matrix. The correlation matrix reveals the strength and direction of linear relationships. Notably, we observe a positive correlation of 0.35 between CMAQ and aod, suggesting that higher CMAQ values align with increased aerosol optical depth.

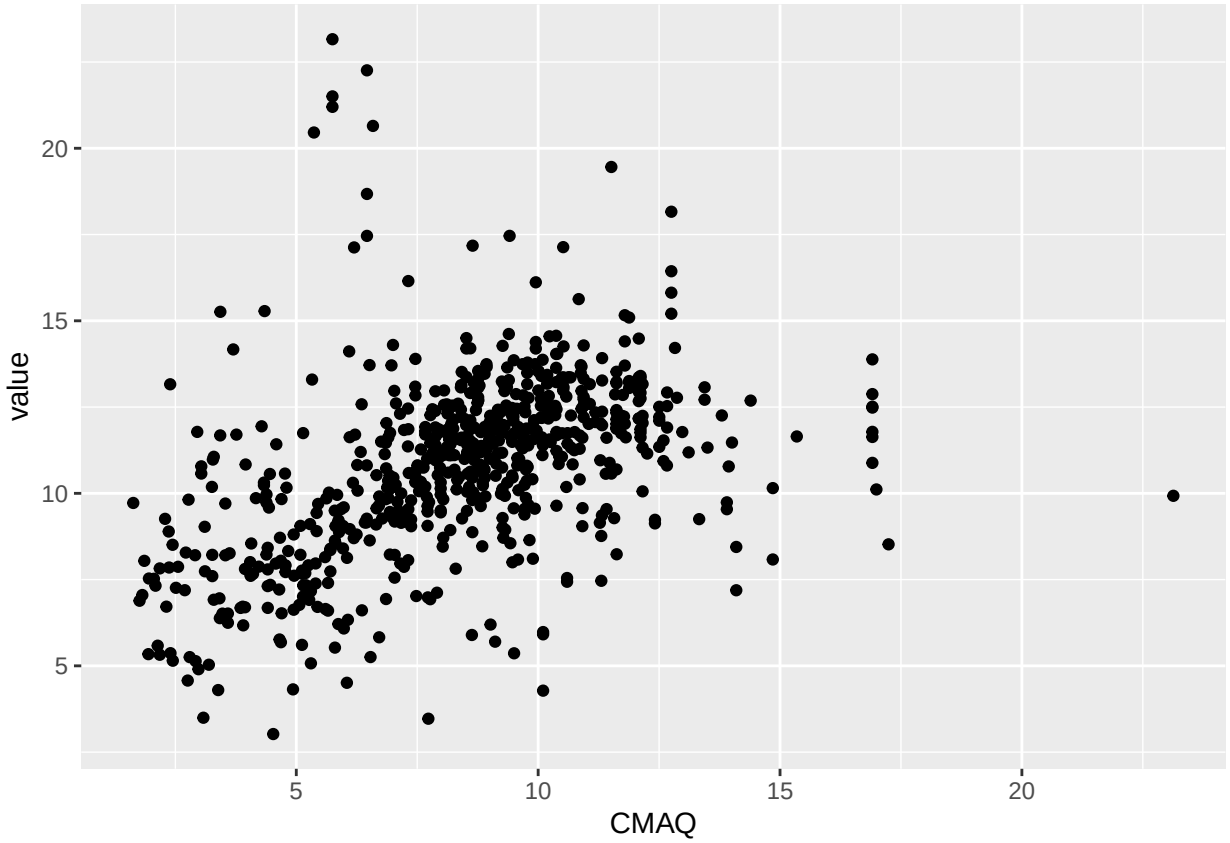
```r
# Create a correlation matrix for the predictor variables
predictors <- dat[, c("CMAQ", "aod", "log_dist_to_prisec", "pov")]

cor_matrix <- cor(predictors)
print(cor_matrix)
```

```
##                          CMAQ         aod log_dist_to_prisec         pov
## CMAQ                1.0000000  0.34772858        -0.13388484  0.14713641
## aod                 0.3477286  1.00000000        -0.15543936  0.05455115
## log_dist_to_prisec -0.1338848 -0.15543936         1.00000000 -0.09314063
## pov                 0.1471364  0.05455115        -0.09314063  1.00000000
```

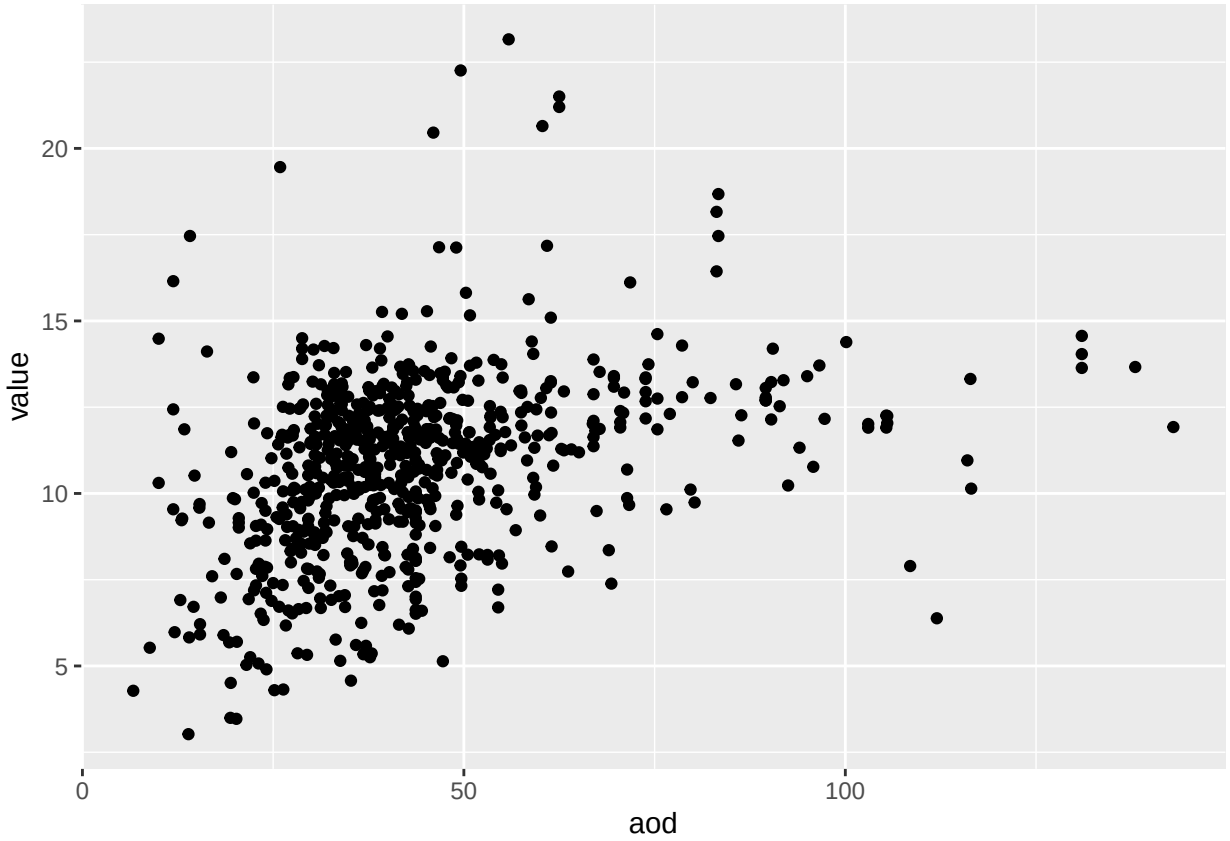
Scatter Plots

To visually explore the relationships, we created scatter plots for each predictor variable and the target variable, 'value'. There seems to be a positive correlation between CMAQ, aod and value. Additionally, there is a negative correlation between log_dist_to_prisec and value. For pov, it is difficult to determine if there is a relationship with value.

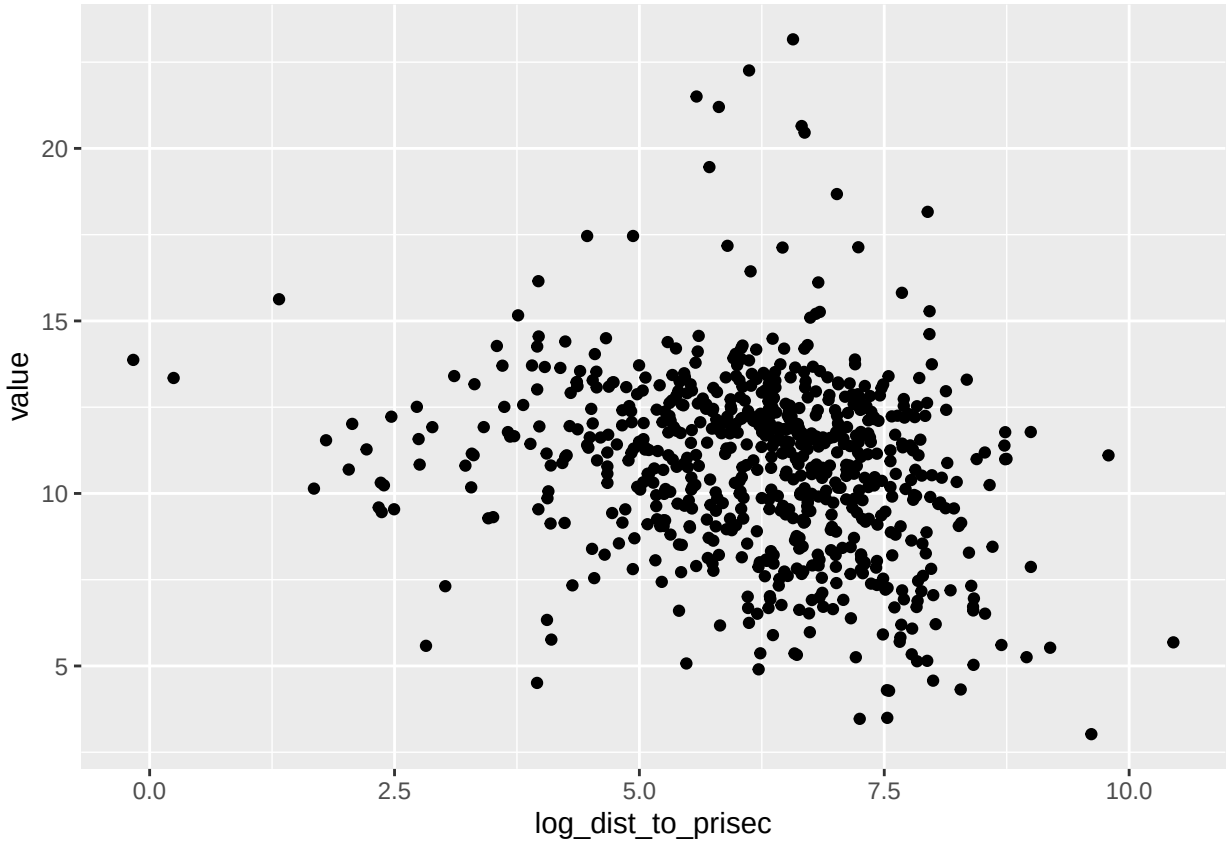
```r
# scatter plot
# CMAQ
train_set |>
  ggplot(aes(x = CMAQ, y = value)) + geom_point()
```



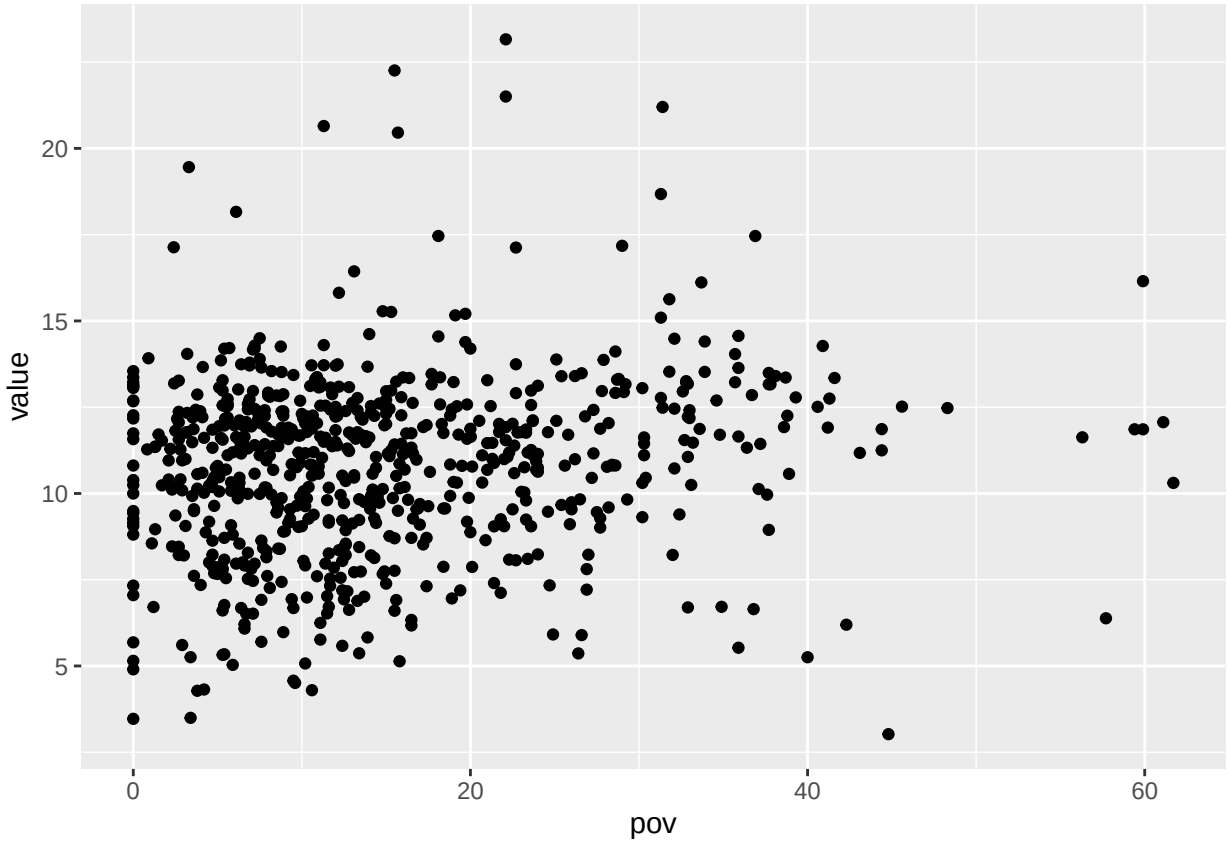

```r
# aod
train_set |>
  ggplot(aes(x = aod, y = value)) + geom_point()
```




```
# log_dist_to_prisec
train_set |>
  ggplot(aes(x = log_dist_to_prisec, y = value)) + geom_point()
```




```
# poverty
train_set |>
  ggplot(aes(x = pov, y = value)) + geom_point()
```



Expectations for RMSE Performance

Before delving into model evaluation, we establish expectations for the Root Mean Squared Error (RMSE) across our selected models. Linear Regression is anticipated to yield a moderate to low RMSE. The k-Nearest Neighbors (kNN) model, leveraging its non-parametric nature, is expected to excel in capturing local variations, resulting in comparatively lower RMSE values. The hierarchical decision-making structure of the Regression Tree is predicted to provide nuanced insights, yielding competitive RMSE values. Lastly, the Random Forest is expected to enhance predictive accuracy, leading to lower RMSE values, positioning it as the top-performing model in our analysis.

Results

A linear regression model was then developed using the training set. The model's summary provides insights into coefficients, R-squared, and other statistics. Predictions were made on the test set using the trained model. The root mean-squared error (RMSE) was calculated as the primary evaluation metric.

Insights:
The linear regression model before cross-validation demonstrates a reasonable fit to the data. An RMSE of 1.889 suggests that, on average, the model's predictions are relatively close to the true values, deviating by approximately 1.889 units from the true values. Then, cross-validation is performed using k-fold cross-validation. Predictions are made on each test fold, and RMSE is computed for model evaluation. Cross-validation results in a slightly higher average RMSE (2.184), suggesting that the model's performance might vary across different subsets of the data. The model appears to be a good starting point, but further refinement or exploration of other models could be considered to improve predictive performance of PM2.5 concentration values.

```
library(tidyverse)
library(kknn)
```

```
## Warning: package 'kknn' was built under R version 4.3.2
```

```
library(tidymodels)
set.seed(123)

## Create the recipe
rec = dat |>
    recipe(value ~ aod + log_dist_to_prisec  + CMAQ + pov)

## Create the model
model = linear_reg() |>
    set_engine("lm") |>
    set_mode("regression")

## Create the workflow
wf <- workflow() %>%
    add_recipe(rec) %>%
    add_model(model)

## Create 10 folds from the dataset
folds = vfold_cv(dat, v = 10)

## Run cross validation with the model
res <- fit_resamples(wf, resamples = folds)

## Show performance metrics
res %>%
    collect_metrics()
```

```
## # A tibble: 2 x 6
##   .metric .estimator  mean     n std_err .config
##   <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1 rmse    standard   2.18     10  0.102  Preprocessor1_Model1
## 2 rsq     standard   0.287    10  0.0370 Preprocessor1_Model1
```

When developing the kNN model, we also began by setting the random seed to ensure reproducibility. A recipe was created to define the variables in the kNN model before configuration. A workflow was then created to combine the recipe and kNN model. Following, a 10-fold cross-validation was initiated. The dataset is divided into 10 subsets, and the model is trained and evaluated 10 times, each time using a different subset as the test set. Hyperparameter tuning is performed to find the optimal number of neighbors. The output shows the results for different model configurations with the best-performing model being "Preprocessor1_Model8" having the lowest mean RMSE of 2.139935. This indicates that, on average, the predictions of this model are close to the actual values. The other configurations (Models 7, 6, 5, and 4) have slightly higher mean RMSE values, indicating slightly less favorable performance. This indicates that the kNN model outperforms the linear regression model in terms of RMSE value on the provided test set.

```
# kNN Model
## Create the recipe
```

```r
set.seed(123)

recipe = dat |>
    recipe(value ~ aod + log_dist_to_prisec  + CMAQ + pov)

## Create the model
knn_model <- nearest_neighbor(neighbors = tune()) %>%
    set_engine("kknn") %>%
    set_mode("regression")

# workflow
workflow = workflow() %>%
    add_recipe(recipe) %>%
    add_model(knn_model)

# cv
cv <- vfold_cv(dat, v = 10)

# results
knn_results =
  workflow %>%
  tune_grid(resamples = cv, grid = 10)

knn_results |>
  collect_metrics()
```

```
## # A tibble: 16 x 7
##    neighbors .metric .estimator  mean     n std_err .config
##        <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
##  1         2 rmse    standard   2.62     10  0.0861 Preprocessor1_Model1
##  2         2 rsq     standard   0.158    10  0.0304 Preprocessor1_Model1
##  3         4 rmse    standard   2.35     10  0.0776 Preprocessor1_Model2
##  4         4 rsq     standard   0.221    10  0.0331 Preprocessor1_Model2
##  5         5 rmse    standard   2.29     10  0.0781 Preprocessor1_Model3
##  6         5 rsq     standard   0.244    10  0.0351 Preprocessor1_Model3
##  7         8 rmse    standard   2.20     10  0.0788 Preprocessor1_Model4
##  8         8 rsq     standard   0.285    10  0.0397 Preprocessor1_Model4
##  9        10 rmse    standard   2.17     10  0.0758 Preprocessor1_Model5
## 10        10 rsq     standard   0.299    10  0.0400 Preprocessor1_Model5
## 11        11 rmse    standard   2.16     10  0.0746 Preprocessor1_Model6
## 12        11 rsq     standard   0.303    10  0.0396 Preprocessor1_Model6
## 13        13 rmse    standard   2.15     10  0.0733 Preprocessor1_Model7
## 14        13 rsq     standard   0.309    10  0.0387 Preprocessor1_Model7
## 15        14 rmse    standard   2.14     10  0.0733 Preprocessor1_Model8
## 16        14 rsq     standard   0.312    10  0.0383 Preprocessor1_Model8
```

```r
knn_results |>
  show_best(metric = 'rmse')
```

```
## # A tibble: 5 x 7
##   neighbors .metric .estimator  mean     n std_err .config
##       <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1        14 rmse    standard    2.14    10  0.0733 Preprocessor1_Model8
```

```
## 2        13 rmse    standard    2.15    10  0.0733 Preprocessor1_Model7
## 3        11 rmse    standard    2.16    10  0.0746 Preprocessor1_Model6
## 4        10 rmse    standard    2.17    10  0.0758 Preprocessor1_Model5
## 5         8 rmse    standard    2.20    10  0.0788 Preprocessor1_Model4
```

The third model we developed was the regression tree model. Likewise, we began by setting the random seed and creating a recipe to define the variables within our function. The tree model is defined using the 'decision_tree' function. The 'tree_depth' parameter is tuned during the model training process. We set the engine and mode before configuring the workflow to use the recursive partitioning algorithm for building the regression tree model. Then, cross-validation is performed using 10-fold cross-validation and the depth of the decision tree is tuned. The output shows the best hyperparameter configuration for different values of 'tree_depth'. The configuration with 'tree_depth' of 4 has the lowest mean RMSE (2.122941), indicating that, on average, this depth provides the best performance across the folds. The other configurations (11, 8, 5, 14) have slightly higher mean RMSE values, indicating slightly less favorable performance. This indicates that the regression tree model outperforms both the linear regression model and kNN model in terms of RMSE value on the provided test set.

```
# Regression Tree
library(rpart)
```

```
##
## Attaching package: 'rpart'

## The following object is masked from 'package:dials':
##
##     prune
```

```
library(caret)
```

```
## Warning: package 'caret' was built under R version 4.3.2

## Loading required package: lattice

##
## Attaching package: 'caret'

## The following object is masked from 'package:kknn':
##
##     contr.dummy

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

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

```
library(tree)
```

```
## Warning: package 'tree' was built under R version 4.3.2
```

```r
set.seed(123)
# recipe
tree_rec = dat |>
    recipe(value ~ aod + log_dist_to_prisec  + CMAQ + pov)

# tree model
tree_model = decision_tree(tree_depth = tune()) |>
  set_engine('rpart') |>
  set_mode('regression')

# tree workflow
tree_workflow =
    workflow() %>%
    add_recipe(tree_rec) %>%
    add_model(tree_model)

cv <- vfold_cv(dat, v = 10)

# tree results
tree_results <- tree_workflow %>%
  tune_grid(resamples = cv, grid = 10)

# metrics
tree_results |>
  show_best(metric = 'rmse')
```

```
## # A tibble: 5 x 7
##   tree_depth .metric .estimator  mean     n std_err .config
##        <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1          4 rmse    standard    2.12    10   0.114 Preprocessor1_Model4
## 2         11 rmse    standard    2.15    10   0.116 Preprocessor1_Model1
## 3          8 rmse    standard    2.15    10   0.116 Preprocessor1_Model2
## 4          5 rmse    standard    2.15    10   0.116 Preprocessor1_Model3
## 5         14 rmse    standard    2.15    10   0.116 Preprocessor1_Model6
```

```r
tree_results |>
  collect_metrics()
```

```
## # A tibble: 16 x 7
##    tree_depth .metric .estimator  mean     n std_err .config
##         <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
##  1         11 rmse    standard   2.15     10  0.116  Preprocessor1_Model1
##  2         11 rsq     standard   0.321    10  0.0583 Preprocessor1_Model1
##  3          8 rmse    standard   2.15     10  0.116  Preprocessor1_Model2
##  4          8 rsq     standard   0.321    10  0.0583 Preprocessor1_Model2
##  5          5 rmse    standard   2.15     10  0.116  Preprocessor1_Model3
##  6          5 rsq     standard   0.321    10  0.0583 Preprocessor1_Model3
##  7          4 rmse    standard   2.12     10  0.114  Preprocessor1_Model4
##  8          4 rsq     standard   0.333    10  0.0607 Preprocessor1_Model4
##  9          2 rmse    standard   2.25     10  0.106  Preprocessor1_Model5
## 10          2 rsq     standard   0.248    10  0.0417 Preprocessor1_Model5
## 11         14 rmse    standard   2.15     10  0.116  Preprocessor1_Model6
## 12         14 rsq     standard   0.321    10  0.0583 Preprocessor1_Model6
```

```
## 13          10 rmse    standard   2.15     10  0.116  Preprocessor1_Model7
## 14          10 rsq     standard   0.321    10  0.0583 Preprocessor1_Model7
## 15          13 rmse    standard   2.15     10  0.116  Preprocessor1_Model8
## 16          13 rsq     standard   0.321    10  0.0583 Preprocessor1_Model8
```

For the fourth model, a random forest model, we once again began by setting a randomization seed. The random forest model is defined with 'rand_forest()' and the number of trees and mtry (the number of available variables to consider at once) were tuned automatically using 'tune()'. The engine was set to 'ranger' and the mode was set to 'regression'. Once the workflow is defined, cross-validation is performed using 10-fold cross-validation and then the forest is tuned manually. The output shows the best hyperparameter configuration for different values of number of trees and mtry. The configuration with an mtry of 2 and 100 trees has the lowest mean RMSE (1.952904), indicating that, on average, this configuration provides the best performance across the folds. The other configurations have slightly higher mean RMSE values, indicating slightly less favorable performance. Out of all four models, this random forest model had the lowest RMSE values, making it our most accurate model and the one we will focus on in the following discussion.

```
# random forest
library(ranger)
```

```
## Warning: package 'ranger' was built under R version 4.3.2
```

```
set.seed(123)

# rec
forest_recipe = dat |>
    recipe(value ~ aod + log_dist_to_prisec  + CMAQ + pov)

# model
forest_model = rand_forest(trees = tune(), mtry = tune()) |>  set_engine('ranger') |>
  set_mode('regression')

# workflow
forest_work =
  workflow() %>%
    add_recipe(forest_recipe) %>%
    add_model(forest_model)

# cv
cv <- vfold_cv(dat, v = 10)

# results
forest_res =
  forest_work |>
  tune_grid(resamples = cv, grid = expand.grid(trees = c(50, 100, 200), mtry = c(1, 2)))

# best tuned parameters metric
forest_res |>
  show_best(metric = 'rmse')
```

```
## # A tibble: 5 x 8
##    mtry trees .metric .estimator  mean     n std_err .config
##   <dbl> <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1     2   100 rmse    standard    1.95    10  0.0894 Preprocessor1_Model5
```

```
## 2     2   200 rmse    standard    1.96    10  0.0901 Preprocessor1_Model6
## 3     2    50 rmse    standard    1.97    10  0.0905 Preprocessor1_Model4
## 4     1   200 rmse    standard    1.98    10  0.0874 Preprocessor1_Model3
## 5     1   100 rmse    standard    1.99    10  0.0821 Preprocessor1_Model2
```

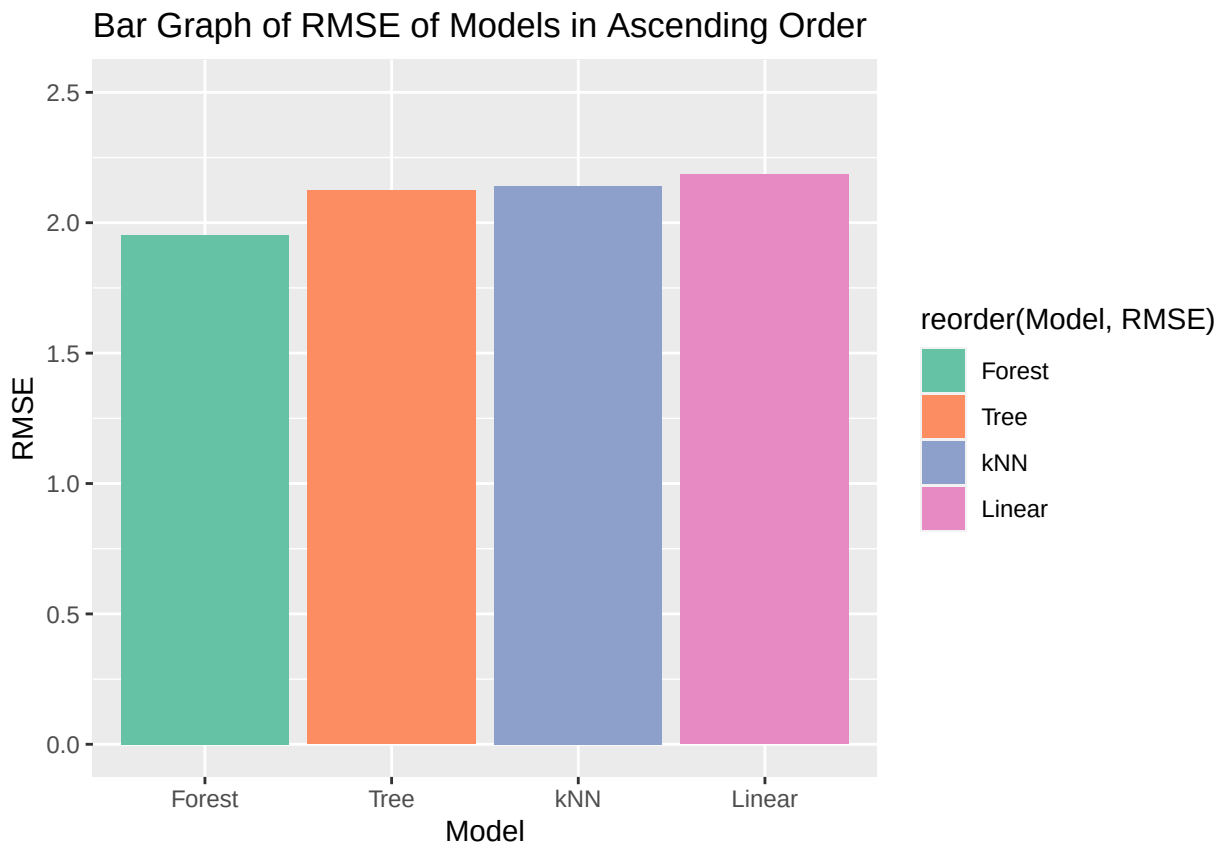
```
forest_res |>
  collect_metrics()
```

```
## # A tibble: 12 x 8
##     mtry trees .metric .estimator  mean     n std_err .config
##    <dbl> <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
##  1     1    50 rmse    standard   2.01     10  0.0935 Preprocessor1_Model1
##  2     1    50 rsq     standard   0.388    10  0.0478 Preprocessor1_Model1
##  3     1   100 rmse    standard   1.99     10  0.0821 Preprocessor1_Model2
##  4     1   100 rsq     standard   0.397    10  0.0426 Preprocessor1_Model2
##  5     1   200 rmse    standard   1.98     10  0.0874 Preprocessor1_Model3
##  6     1   200 rsq     standard   0.401    10  0.0443 Preprocessor1_Model3
##  7     2    50 rmse    standard   1.97     10  0.0905 Preprocessor1_Model4
##  8     2    50 rsq     standard   0.406    10  0.0484 Preprocessor1_Model4
##  9     2   100 rmse    standard   1.95     10  0.0894 Preprocessor1_Model5
## 10     2   100 rsq     standard   0.416    10  0.0507 Preprocessor1_Model5
## 11     2   200 rmse    standard   1.96     10  0.0901 Preprocessor1_Model6
## 12     2   200 rsq     standard   0.412    10  0.0505 Preprocessor1_Model6
```

Now that our RMSE values have been determined, we are able to compare them in the following bar graph and table. Although the values differ only slightly, our forest tree model is clearly the superior model in terms of prediction accuracy.

```
# mean RMSE evaluation after cv for each model
# creates df
model_data =
  data.frame(Model = c('Linear', 'kNN', 'Tree', 'Forest'),
             RMSE = c(2.1848615, 2.139935, 2.122941, 1.952904),
              Rsq = c(0.2874238, 0.3116427, 0.3326779,  0.4157813))

# Visualization of performance of models
model_data |>
  ggplot(aes(x = reorder(Model, RMSE), y = RMSE, fill = reorder(Model, RMSE))) +
  geom_bar(stat = 'summary') +
  scale_fill_brewer(palette = 'Set2') +
  labs(title = 'Bar Graph of RMSE of Models in Ascending Order', x = 'Model') +
  scale_y_continuous(limits = c(0, 2.5))
```

```
## No summary function supplied, defaulting to `mean_se()`
```




```
# table summarizing metrics
model_data |>
  select(Model, RMSE) |>
  arrange(RMSE)
```

```
##    Model     RMSE
## 1 Forest 1.952904
## 2   Tree 2.122941
## 3    kNN 2.139935
## 4 Linear 2.184861
```

Now that our best model has been determined, we are able to further analyze its performance and reflect on other factors that impact its accuracy.

```
# random forest best model
set.seed(123)
library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 4.3.2
```

```
## randomForest 4.7-1.1
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'

## The following object is masked from 'package:ranger':
##
##     importance

## The following object is masked from 'package:dplyr':
##
##     combine

## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
formula = value ~ aod + log_dist_to_prisec  + CMAQ + pov

# forest model on train data
forest_model = randomForest(formula, mtry = 2, ntree = 100, data = train_set)

# predictions on test data
predictions = predict(forest_model, newdata = test_set)

# rmse
rmse = RMSE(pred = predictions, obs = test_set$value)

# resid
residuals = test_set$value - predictions

# add resid to df
test_set2 =
  test_set|>
  mutate(residuals = residuals) |>
  mutate(predictions = predictions)

# largest residuals
test_set2 |>
  arrange(desc(abs(residuals))) |>
  select(residuals, predictions, value, county, city, state)
```

```
## # A tibble: 176 x 6
##    residuals predictions value county       city          state
##        <dbl>       <dbl> <dbl> <chr>        <chr>         <chr>
##  1      6.26       10.4  16.7  Merced       Merced        California
##  2      6.19        7.11 13.3  Klamath      Altamont      Oregon
##  3     -5.86       14.9   9.04 Monroe       Rochester     New York
##  4     -4.66       11.9   7.26 Alachua      Gainesville   Florida
##  5     -4.57        9.07  4.5  Laramie      Cheyenne      Wyoming
##  6      4.43       11.2  15.6  Jefferson    Birmingham    Alabama
##  7     -4.13        8.88  4.74 Cass         Not in a city Minnesota
##  8      4.11        9.01 13.1  Lane         Oakridge      Oregon
##  9     -3.83       12.1   8.23 Hillsborough Valrico       Florida
## 10     -3.70       12.1   8.40 Washington   Bayport       Minnesota
## # i 166 more rows
```

```r
# largest residuals by state
test_set2 %>%
  group_by(state) %>%
  summarize(avg_residuals = mean(residuals),
            standard_dev = sd(residuals)) %>%
  arrange(desc(abs(avg_residuals)))
```

```
## # A tibble: 40 x 3
##    state         avg_residuals standard_dev
##    <chr>                 <dbl>        <dbl>
##  1 Florida               -2.97        1.16
##  2 New York              -2.89        2.30
##  3 West Virginia          2.77        0.590
##  4 Oregon                 2.61        2.98
##  5 Nevada                -2.47       NA
##  6 Minnesota             -2.36        2.71
##  7 New Mexico            -2.25        1.02
##  8 Wyoming               -2.18        2.27
##  9 Pennsylvania           2.05        0.864
## 10 Arizona               -1.81        0.743
## # i 30 more rows
```

```r
# smallest residuals
test_set2 |>
  arrange(abs(residuals)) |>
  select(residuals, predictions, value, county, city, state)
```

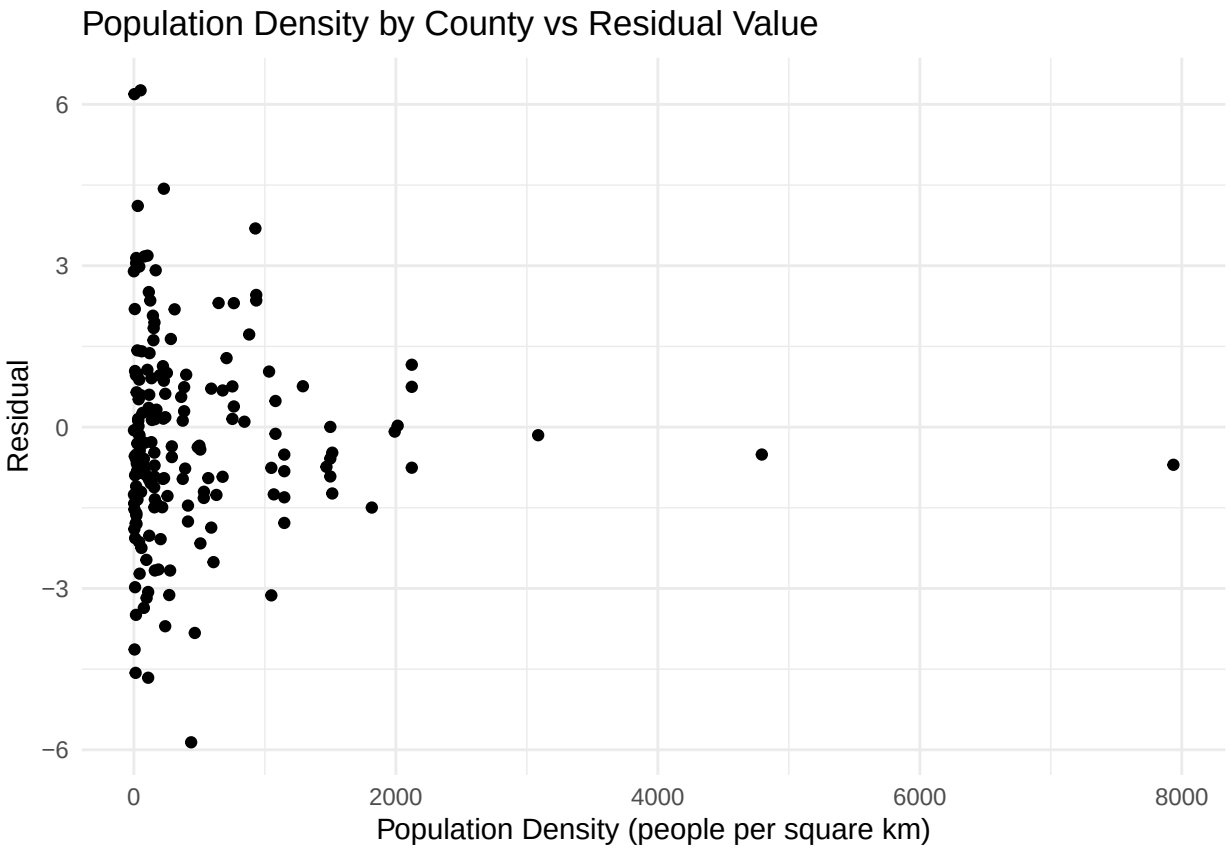
```
## # A tibble: 176 x 6
##    residuals predictions value county           city          state
##        <dbl>       <dbl> <dbl> <chr>            <chr>         <chr>
##  1   0.00330        12.7  12.7  Bergen           Carlstadt     New Jersey
##  2   0.0190         11.2  11.2  Colbert          Muscle Shoals Alabama
##  3   0.0254         12.7  12.7  Union            Elizabeth     New Jersey
##  4  -0.0595          6.23  6.17 Sublette         Pinedale      Wyoming
##  5  -0.0842         12.6  12.5  St. Louis City   St. Louis     Missouri
##  6   0.100          12.3  12.4  Franklin         Columbus      Ohio
##  7   0.115          11.4  11.6  Muskogee         Muskogee      Oklahoma
##  8   0.119          11.3  11.4  East Baton Rouge Baton Rouge   Louisiana
##  9  -0.125          12.0  11.9  Cuyahoga         Brook Park    Ohio
## 10   0.132          11.9  12.0  Alamance         Burlington    North Carolina
## # i 166 more rows
```

```r
# smallest residuals by state
test_set2 %>%
  group_by(state) %>%
  summarize(avg_residuals = mean(residuals),
            standard_dev = sd(residuals)) %>%
  arrange(abs(avg_residuals))
```

```
## # A tibble: 40 x 3
##    state          avg_residuals standard_dev
##    <chr>                  <dbl>        <dbl>
```

```
##  1 Kentucky               0.145        0.882
##  2 Virginia               0.187        1.19
##  3 Iowa                  -0.203        0.865
##  4 Illinois               0.248        1.14
##  5 California             0.288        2.50
##  6 Oklahoma               0.341        0.853
##  7 South Carolina         0.341        0.768
##  8 Arkansas              -0.372        0.740
##  9 Delaware              -0.374       NA
## 10 Indiana                0.379        1.08
## # i 30 more rows
```

```r
# Find the mean value of pop. density, ignoring missing values
mean_dens = mean(test_set2$popdens_county, na.rm = TRUE)
# Find the standard deviation of pop. density, ignoring missing values
sd_dens = sd(test_set2$popdens_county, na.rm = TRUE)

# Create a column that represents the z-score values of pop. density
test_set3 <- test_set2 %>%
  mutate(dens_zscore = (popdens_county - mean_dens) / sd_dens)

# Plot the Density vs residuals
test_set3 %>%
  select(residuals, state, city, dens_zscore, popdens_county) %>%
  ggplot(aes(x = popdens_county, y = residuals)) +
  geom_point() +
  labs(title = "Population Density by County vs Residual Value",
       x = "Population Density (people per square km)",
       y = 'Residual') +
  theme_minimal()
```



In this figure, the x-axis represents population density and the y-axis represents the residual value. As density increases, the spread vertical spread decreases, meaning the residual values move closer to zero. This indicates that our random forest model is better at predicting PM2.5 levels in areas with a higher population density.

Discussion and Reflection

1. The locations where the model gives the smallest residuals and largest residuals are summarized in the code. The locations with the smallest average residuals are Kentucky, Virginia, Iowa. Locations with the largest average residuals were Florida, New York, and West Virginia. According to the previous graph, our model provided predictions that were closest to the observed values in locations where population density was high. The residual standard deviation was very high in areas with low population density. One possible cause of this is our predictor variable pov. Out of our four predictors, poverty seemed to be the weakest. Poverty tends to be higher in rural areas, which are less dense, which potentially lowered the accuracy of our model in these places.

2. As stated before, the most notable variable that predicts model performance is population density. Factors such as longitude, latitude, region, and proximity to coast seemed to be poor predictors of model accuracy. One variable not included in the data set that may improve the accuracy of our model is biome types. Certain types of environments may be more effective at dispersing or retaining air pollution, For example, forested areas may be able to capture some amounts of pollution particles. Another variable that might help is wind patterns. Windier areas experience more air flow, which may lessen the amount of air pollution that sticks around. Areas with more stagnant air could accumulate more particles of pollution.

3. When CMAQ and aod are excluded as predictors from the model, the RMSE increases drastically. While both are included, the RMSE value ranges from 1.95-1.99. After removing only aod, that value

rises to 2.10-2.13. When only CMAQ is excluded the RMSE increases to 2.32-2.34. When both are removed, the RMSE ranges from 2.64-2.70.These two predictor variables seem to be far more important than our other two: Log_dist_to_prisec and pov. When these are excluded from the model, the RMSE hardly changes at all. In fact, it even lowers in some cases. Based on these observations, it seems that CMAQ and aod are excellent predictors of ground-level concentrations of PM2.5.

4. Finding patterns on where exactly our model did best was difficult, it doesn't seem to be impacted by geographic region, poverty, or similar variables. One pattern we did find, though, was that as population density increased, the model's accuracy increased as well. Residuals were smaller in counties with high population density. For that reason, our model likely would not perform well in Alaska due to the incredibly low population density. Hawaii, on the other hand, is a bit more densely populated, so our model is more likely to do well there.

What was challenging for our group was figuring out which models to develop and making sure we tuned parameters for the more intricate models like the kNN. It was also difficult to analyze our results in terms of finding patterns.

Although our model performance was satisfactory, we were anticipating a slightly higher accuracy in its predictions. The forest tree model was certainly better than the regression tree model, but not to the degree we originally expected. This may be due to the fairly small amount of predictor variables we chose. There were nearly fifty to choose from, but we incorporated only four in order to refrain from over complicating the models. In addition to this, the variables pov and log_dist_to_prisec seemed to be far less useful than the other two we utilized: CMAQ and aod. In fact, when pov was removed, the average RMSE over our cross validation folds went down, meaning the model became more accurate. If we were to attempt to improve our model in the future, we would take a more in depth look at the predictor variables available to us when deciding what to incorporate. We could also use a more trial and error based system to find which combination of predictors yields the best results.

Our group contributions were as following:
Mindy: Responsible for creating the models
Mayce: Analyzed models and results
Carlos: Analyzed models and results