Predicting Ambient Air Pollution (PM2.5) Across the Contiguous U.S.

Aileen Li, Nyah Strickland

2023-04-18

Air pollution has consequences for everyone, most especially for those with pre-existing conditions. Air pollution is measured in tiny particles or droplets in the air that are two and one-half microns or less in width called PM2.5. So being able to predict the average PM2.5 concentrations (μg/m^3) will allow us to keep people most vulnerable safe as we look for and implement changes to improve the air quality in the region. We created four models to predict the annual average of ambient air pollution in a location based on the variables in the dataset. We chose our predictors using the Random Forest algorithm. And for our four models, we chose Linear Regression, K-Nearest Neighbors, XG Boost, and Random Forest. Linear Regression models a linear relationship between the PM2.5 value in the atmosphere and the predictors. K-Nearest Neighbors returns predicted PM2.5 values based on their neighboring data points. XG Boost predicts PM2.5 values by combining the estimates of a set of simpler, weaker models. Random forest makes an averaged prediction from a collection of independent decision trees. We hypothesize that the RMSE performance of the XG Boost model will be the best, with a value of around 0.5.

```
library(tidyverse);library(tidymodels);library(broom);library(xgboost);library(caret);library(pl
otROC)
library(randomForest);library(factoextra);library(MASS);library(ranger);library(doParallel)
origin <- read_csv("https://github.com/rdpeng/stat322E_public/raw/main/data/pm25_data.csv.gz")
```

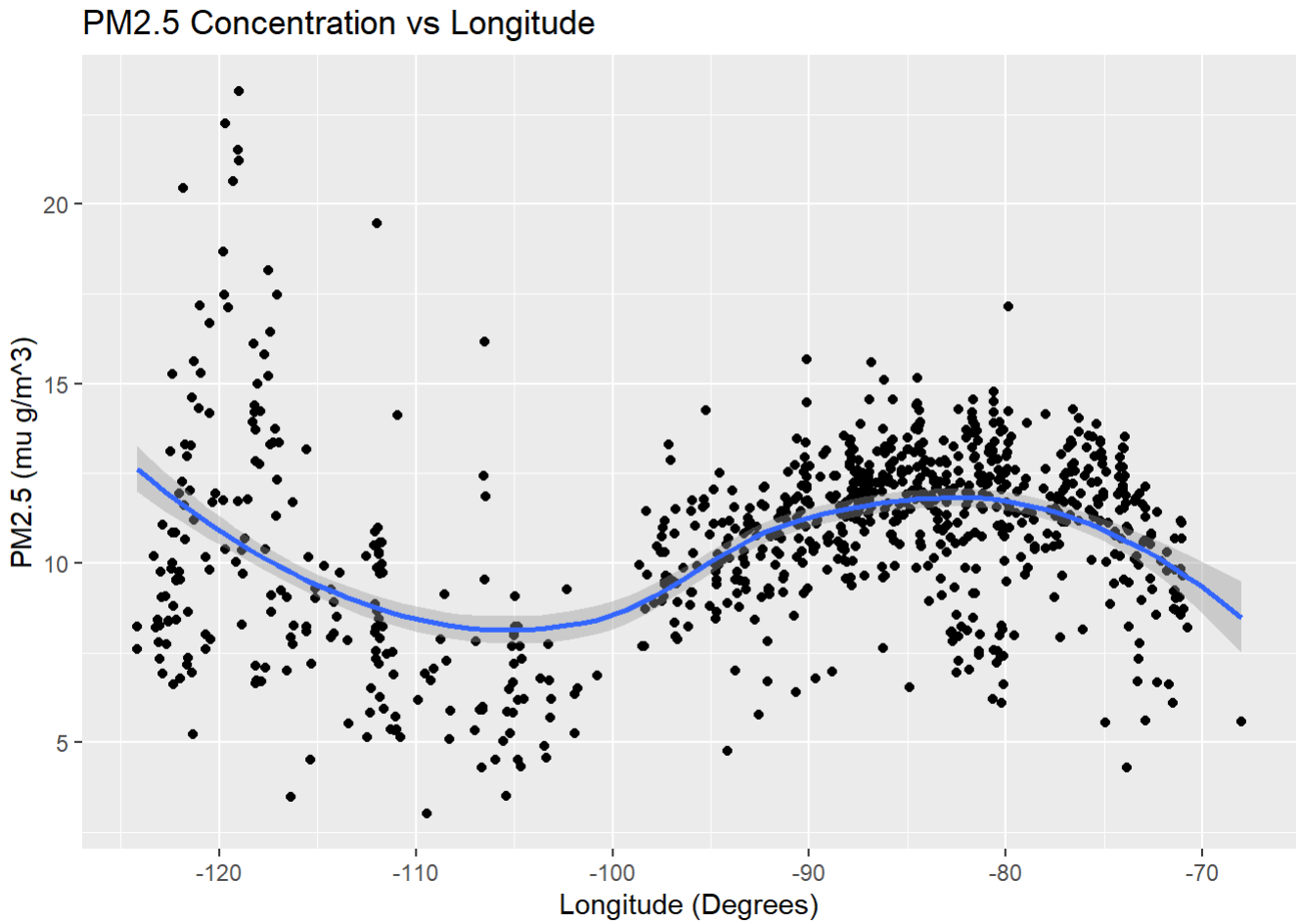
Wrangling

We split 80% of the origin dataset into training and 20% into testing. Then, we scaled the training datasets for PCA and kNN since both of these require little variance among the variables in the dataset. We omitted the ‘id’ and ‘value’ variables from the ‘trained_scaled’ dataset for PCA because the ‘id’ values for the monitors would not provide any useful insight on PM2.5 concentration and ‘value’ is not a predictor. Next, we chose 12 principal components with the highest explained variance since the sum of these components met our threshold of 80% explained variance. Then, we tried to see which variables contributed the most to ‘value’ and reduced the dimension of the ‘trained_scale’ dataset based on these variables. For kNN, we created a scaled training set ‘train_knn’ since extreme values would bias the distance calculation used in kNN modeling. The other models either used ‘train’ dataset since the variables’ values do not affect their prediction performance or used a specially altered version of it. Lastly, we did some exploratory analysis to see the linear relationship between longitude and latitude with PM2.5 concentration.

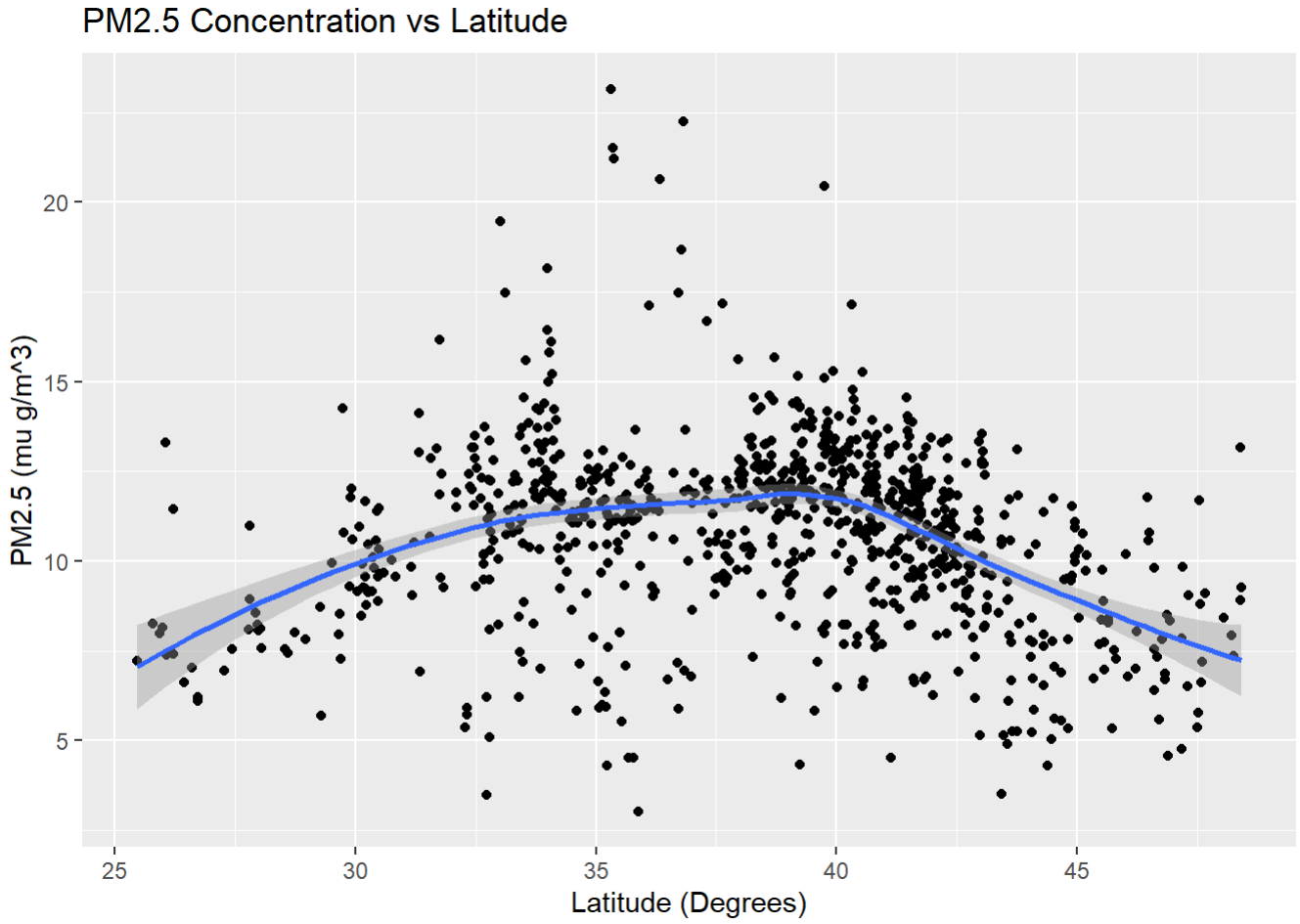
```
set.seed(123)
# Split origin dataset into 80/20
dat_split <- initial_split(origin, prop =.8)
dat_split
```

```
## <Training/Testing/Total>
## <700/176/876>
```

```
# Use 80% of data for training
train <- training(dat_split)
test <- testing(dat_split)

# Create standardize train data for kNN and PCA
# Omit value and id variables since these should not be scaled
# train_scaled for PCA
train_scaled = train[,3:50] %>% dplyr::select(where(is.numeric)) %>% mutate(across(fips:aod, sca
le))
#%>% cbind(state=train$state,county=train$county,city=train$city)

# train_knn for kNN model
train_knn = train[,2:50] %>%
  dplyr::select(where(is.numeric)) %>%
  mutate(across(fips:aod, scale)) %>%
  cbind(state=train$state,county=train$county,city=train$city)

# Visualize linear relationship between predictors (lat, lon, state) and value
origin %>% ggplot(aes(x=lon, y=value)) + geom_point() + geom_smooth() +
  labs(title="PM2.5 Concentration vs Longitude", x="Longitude (Degrees)", y="PM2.5 (mu g/m^3)")
```



```
origin %>% ggplot(aes(x=lat, y=value)) + geom_point() + geom_smooth() +
  labs(title="PM2.5 Concentration vs Latitude", x="Latitude (Degrees)", y="PM2.5 (mu g/m^3)")
```



Predictor/Feature Extraction

We used the random forest algorithm (RF) to determine which variables would be the best predictors for our model since RF is good at using variance to determine which features would be most influential for a model's predictive performance. In other words, the more variability there is in a dataset, the better RF performs at predicting which variables contribute to the outcome by averaging the results of all trees at the end and this averaging reduces the model's variance. Since our dataset has a wide variety of predictors ranging from education attainment levels to poverty levels in a given monitor region to emission data, RF would be good at reducing the variance in our dataset. For a given feature, the lower its impurity levels are, the more important that feature is. As a result, we got an ordered list of important variables and took the top 5 as input for the training set for our models. The top 5 important predictors were 'CMAQ', 'lat', 'county_area', 'lon', and 'aod'. We will use these features to predict 'value' proceeding onward. This website was used: TowardDataScience (https://towardsdatascience.com/feature-selection-using-random-forest-26d7b747597f).

```
# Use Random Forest for variable selection
rfmodel = randomForest(value~., data=train[,2:50])
import=as.data.frame(randomForest::importance(rfmodel)) |> arrange(desc(IncNodePurity))
import=tibble::rownames_to_column(import, 'variables')
# take top 5 important variables
import5 = import[1:5,]
```

Models

1. Linear Regression

For linear regression (LR), the unscaled training set was used since variance does not affect the prediction performance of this model. Step Akaike Information Criteria (AIC) was used to simplify feature amount without impacting the model's performance. Features are dropped as the AIC score decreases and feature dropping stops once the AIC score increases. 'CMAQ'and 'aod' were determined to be the most important features, and these predictors were used in the recipe. Then, the model was fitted. Using the summary function, we found the LR model had a RMSE of 2.27 performance. After 10-fold cross validation, the RMSE score was over 1, indicating that this model's prediction performance is not good.

```
set.seed(123)
# Fit model and get summary
fit_lm = lm(value ~ CMAQ+lat+county_area+lon+aod, data=train)
step <- stepAIC(fit_lm, direction="backward", trace=FALSE)
# Keeps cmaq and aod

# Create the recipe for all models
rec1 <- train %>%
    recipe(value ~ CMAQ+aod)

# Linear regr. model
model1 <- linear_reg() %>%
    set_engine("lm") %>%
    set_mode("regression")

wf1 <- workflow() %>%
    add_recipe(rec1) %>%
    add_model(model1)

res1 <- wf1 |> parsnip::fit(data = train)

# Check performance of linear regression model on the training data
res1 %>% extract_fit_engine() %>% summary()
```

```
##
## Call:
## stats::lm(formula = ..y ~ ., data = data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -6.4165 -1.1913 -0.0098  1.1486 12.8274
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 6.735167   0.284723  23.655  < 2e-16 ***
## CMAQ        0.327107   0.031077  10.526  < 2e-16 ***
## aod         0.030750   0.004663   6.595 8.41e-11 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.288 on 697 degrees of freedom
## Multiple R-squared:  0.2458, Adjusted R-squared:  0.2437
## F-statistic: 113.6 on 2 and 697 DF,  p-value: < 2.2e-16
```

```
# Check linear regression model performance using cross-validation
folds1 <- vfold_cv(train, v = 10)
# folds1
res1 <- fit_resamples(wf1, resamples = folds1)
res1 %>% collect_metrics()
```

```
## # A tibble: 2 × 6
##   .metric .estimator  mean     n std_err .config
##   <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1 rmse    standard   2.27     10  0.118  Preprocessor1_Model1
## 2 rsq     standard   0.249    10  0.0216 Preprocessor1_Model1
```

2. K-Nearest Neighbors

The K-Nearest Neighbors (KNN) model finds the predicted ‘value’ by taking the average ‘value’ of its k neighbors. We first found the ideal number of k neighbors. This was determined by finding the square root of the number of observations, which was rounded to be 26. We fit the model on the training set data and checked its performance using two methods: the first was fitting the model to extracting the MAE and MSE and the second was 10-fold cross validation. RMSE is the square root of MSE and we were able to confirm its value in the cross-validation by comparing it to the square root of MSE attained from the first method. We found that after 10-fold cross validation, the RMSE was 1.84. This indicates that the model does not predict well since the amount of error is greater than 1.

```
# kNN uses scaled data called train_knn
set.seed(123)
# Find ideal k neighbors
sqrt(nrow(train))
```

```
## [1] 26.45751
```

```
# Create recipe
rec2 = train_knn |> recipe(value ~ CMAQ+lat+county_area+lon+aod)

# Create kNN model
model2 <- nearest_neighbor(neighbors = 26) %>%
    set_engine("kknn") %>%
    set_mode("regression")

# Create workflow
  wf2 <- workflow() %>%
    add_recipe(rec2) %>%
    add_model(model2)

# Fit the model on the training dataset using the `fit()` function
res2 <- parsnip::fit(wf2, data = train_knn)

# Check performance on the complete training data
res2 %>% extract_fit_engine() %>% summary()
```

```
##
## Call:
## kknn::train.kknn(formula = ..y ~ ., data = data, ks = min_rows(26,     data, 5))
##
## Type of response variable: continuous
## minimal mean absolute error: 1.257437
## Minimal mean squared error: 3.327239
## Best kernel: optimal
## Best k: 26
```

```
# Check performance using cross-validation
folds2 <- vfold_cv(train_knn, v = 10)
# folds
res2 <- fit_resamples(wf2, resamples = folds2)
res2 %>% collect_metrics()
```

```
## # A tibble: 2 × 6
##   .metric .estimator  mean     n std_err .config
##   <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1 rmse    standard   1.84     10  0.0842 Preprocessor1_Model1
## 2 rsq     standard   0.514    10  0.0381 Preprocessor1_Model1
```

3. Extreme Gradient Boost Regression

The Extreme Gradient Boosting (XGB) model performs by training weak learners, models that have low prediction accuracy, sequentially to become a single strong learner, a model that has strong prediction accuracy. We preprocess the training set to make the predictive modeling process more accurate and smoother by prepping (estimates the quantities for 'train_scaled') and baking (assigns prepped data to an object) the data. The aforementioned top 5 predictors were used in this 'rec3'. Moreover, the categorical variables were factored and features with no variance were removed before this data was baked. Then, this processed data is assigned to

'folds3' and resampled on 5 subsets of the training set. Using 'fold3', we tune the hyperparameters to make the modeling process more efficient and perform grid specification to determine which hyperparameter values have high prediction accuracy (i.e. low prediction error). Next, we isolated the best hyperparameter values, so they can be used for the final boosting model. Lastly, we fitted the model and evaluated its prediction performance. Its RMSE score was 0.23, indicating that XG Boost is an adequate model. This website was used: RBloggers (https://www.r-bloggers.com/2020/05/using-xgboost-with-tidymodels/)

```
set.seed(123)
# Preprocessing recipe
rec3 <-
  recipes::recipe(value ~ CMAQ+lat+county_area+lon+aod, data = training(dat_split)) %>%
  # Convert categorical variables to factors
  recipes::step_string2factor(all_nominal()) %>%
  # Combine low frequency factor levels
  recipes::step_other(all_nominal(), threshold = 0.01) %>%
  # Remove no variance predictors which provide no predictive information
  recipes::step_nzv(all_nominal()) %>%
  prep()

folds3 <- recipes::bake(rec3, new_data = training(dat_split)) %>%
  rsample::vfold_cv(v = 5)

# XGBoost model specification
xgboost_model <- parsnip::boost_tree(mode = "regression", trees = 1000, min_n = tune(),
  tree_depth = tune(), learn_rate = tune(), loss_reduction = tune()) %>%
  set_engine("xgboost", objective = "reg:squarederror")

# Grid specification
xgboost_params <- dials::parameters(min_n(), tree_depth(), learn_rate(), loss_reduction())

xgboost_grid <- dials::grid_max_entropy(xgboost_params, size = 60)

xgboost_wf <- workflows::workflow() %>%
  add_model(xgboost_model) %>%
  add_formula(value ~ CMAQ+lat+county_area+lon+aod)

# Hyperparameter tuning, Takes a long time to run, 5-8 min
xgboost_tuned <- tune::tune_grid(object = xgboost_wf, resamples = folds3,
  grid = xgboost_grid, metrics = yardstick::metric_set(rmse, rsq),
  control = tune::control_grid(verbose = TRUE))

# Find hyper parameter values which performed best at minimizing RMSE
xgboost_tuned %>% tune::show_best(metric = "rmse")
```

```
## # A tibble: 5 × 10
##   min_n tree_depth learn_rate loss_reduction .metric .estimator  mean     n
##   <int>      <int>      <dbl>          <dbl> <chr>   <chr>      <dbl> <int>
## 1     7          9    0.0301        8.93e-10 rmse    standard    1.66     5
## 2    22          8    0.0201        1.69e- 1 rmse    standard    1.67     5
## 3    16         15    0.0229        1.45e+ 0 rmse    standard    1.68     5
## 4     2         11    0.0832        5.03e- 1 rmse    standard    1.68     5
## 5    23          9    0.00597       3.47e-10 rmse    standard    1.71     5
## # ℹ 2 more variables: std_err <dbl>, .config <chr>
```

```
# Isolate the best performing hyperparameter values.
xgboost_best_params <- xgboost_tuned %>% tune::select_best("rmse")
xgboost_best_params
```

```
## # A tibble: 1 × 5
##   min_n tree_depth learn_rate loss_reduction .config
##   <int>      <int>      <dbl>          <dbl> <chr>
## 1     7          9     0.0301       8.93e-10 Preprocessor1_Model44
```

```
# Final boost model
xgboost_model_final <- xgboost_model %>% finalize_model(xgboost_best_params)

# Eval. model performance on train data
train_processed <- bake(rec3,  new_data = training(dat_split))

train_prediction <- xgboost_model_final %>%
  # Fit the model on all the training data
  parsnip::fit(formula = value ~ CMAQ+lat+county_area+lon+aod, data = train_processed) %>%
  # Predict value for the training data
  predict(new_data = train_processed) %>%
  bind_cols(training(dat_split))

res3 <- train_prediction %>%
  yardstick::metrics(value, .pred) %>%
  mutate(.estimate = format(round(.estimate, 2), big.mark = ","))
res3
```

```
## # A tibble: 3 × 3
##   .metric .estimator .estimate
##   <chr>   <chr>      <chr>
## 1 rmse    standard   0.23
## 2 rsq     standard   0.99
## 3 mae     standard   0.15
```

4. Random Forest Regression

The Random Forest (RF) Regression model performs by training multiple different samples simultaneously and averages the outcome at the end for the final prediction. The randomness of these samples contribute to the prediction accuracy of this model. Similar to the XG Boosting Regression model, we preprocess the training set with the same features to improve the predictive modeling process and then create model specifications. This processed data is assigned to 'folds4' and resampled on 5 subsets of the training set to tune the hyperparameters. We then performed grid specification to determine which hyperparameter values have high prediction accuracy (i.e. low prediction error). Next, we isolated the best hyperparameter values, so they can be used for the final boosting model. Lastly, we fitted the model and evaluated its prediction performance. Its RMSE score was 8.81, indicating the RF is not a good predictive model.

```
set.seed(123)
# Preprocessing recipe
rec4 <- recipe(value ~ CMAQ+lat+county_area+lon+aod, data=train) %>%
  step_log(all_outcomes()) |>
  step_normalize(all_numeric_predictors()) |>
  step_dummy(all_nominal_predictors()) |> prep()

#  Create model specification and wf
rf_spec <- rand_forest(mtry = tune(), trees = tune(), min_n = tune()) %>%
  set_engine("ranger", num.threads=7, importance="impurity") |>
  set_mode("regression")

wf4 <- workflow() %>%
  add_recipe(rec4) %>%
  add_model(rf_spec)

folds4 = recipes::bake(rec4, new_data = training(dat_split)) %>%
  rsample::vfold_cv(v = 5)

# Space-filling designs grid
rf_grid <- grid_latin_hypercube(min_n(), mtry(range(4,9)), trees(), size = 60)

# Tuning hyperparameters
all_cores = parallel::detectCores(logical=FALSE)
cl = makePSOCKcluster(all_cores);doParallel::registerDoParallel(cl)

# Takes long time to load 2 min
tune_res <- wf4 |> tune_grid(resamples = folds4,
                             grid = rf_grid,
                             metrics=metric_set(rmse,rsq),
                             control = tune::control_grid(verbose = TRUE))

# Find hyper parameter values which performed best at minimizing RMSE
tune_res %>% tune::show_best(metric = "rmse")
```

```
## # A tibble: 5 × 9
##    mtry trees min_n .metric .estimator   mean     n std_err .config
##   <int> <int> <int> <chr>   <chr>       <dbl> <int>   <dbl> <chr>
## 1     4   793     3 rmse    standard   0.0807     5 0.00434 Preprocessor1_Model…
## 2     4   690    15 rmse    standard   0.0817     5 0.00411 Preprocessor1_Model…
## 3     5  1542     3 rmse    standard   0.0818     5 0.00382 Preprocessor1_Model…
## 4     8  1342     3 rmse    standard   0.0818     5 0.00393 Preprocessor1_Model…
## 5     7   422     5 rmse    standard   0.0819     5 0.00388 Preprocessor1_Model…
```

```
# Isolate the best performing hyperparameter values.
rf_best_params <- tune_res %>% tune::select_best("rmse")
rf_best_params
```

```
## # A tibble: 1 × 4
##    mtry trees min_n .config
##   <int> <int> <int> <chr>
## 1     4   793     3 Preprocessor1_Model28
```

```
# Final Random Forest model
rf_model_final <- rf_spec %>% finalize_model(rf_best_params)

# Check Random Forest model performance
train_processed2 <- bake(rec4,  new_data = training(dat_split))

train_prediction2 = rf_model_final |>
  parsnip::fit(formula = value ~ CMAQ+lat+county_area+lon+aod,
               data=train_processed2) |>
  predict(new_data=train_processed2) |>
  bind_cols(training(dat_split))

res4 = train_prediction2 |>
  yardstick::metrics(value, .pred) |>
  mutate(.estimate=format(round(.estimate, 2), big.mark=","))
res4
```

```
## # A tibble: 3 × 3
##   .metric .estimator .estimate
##   <chr>   <chr>      <chr>
## 1 rmse    standard   8.81
## 2 rsq     standard   0.91
## 3 mae     standard   8.47
```

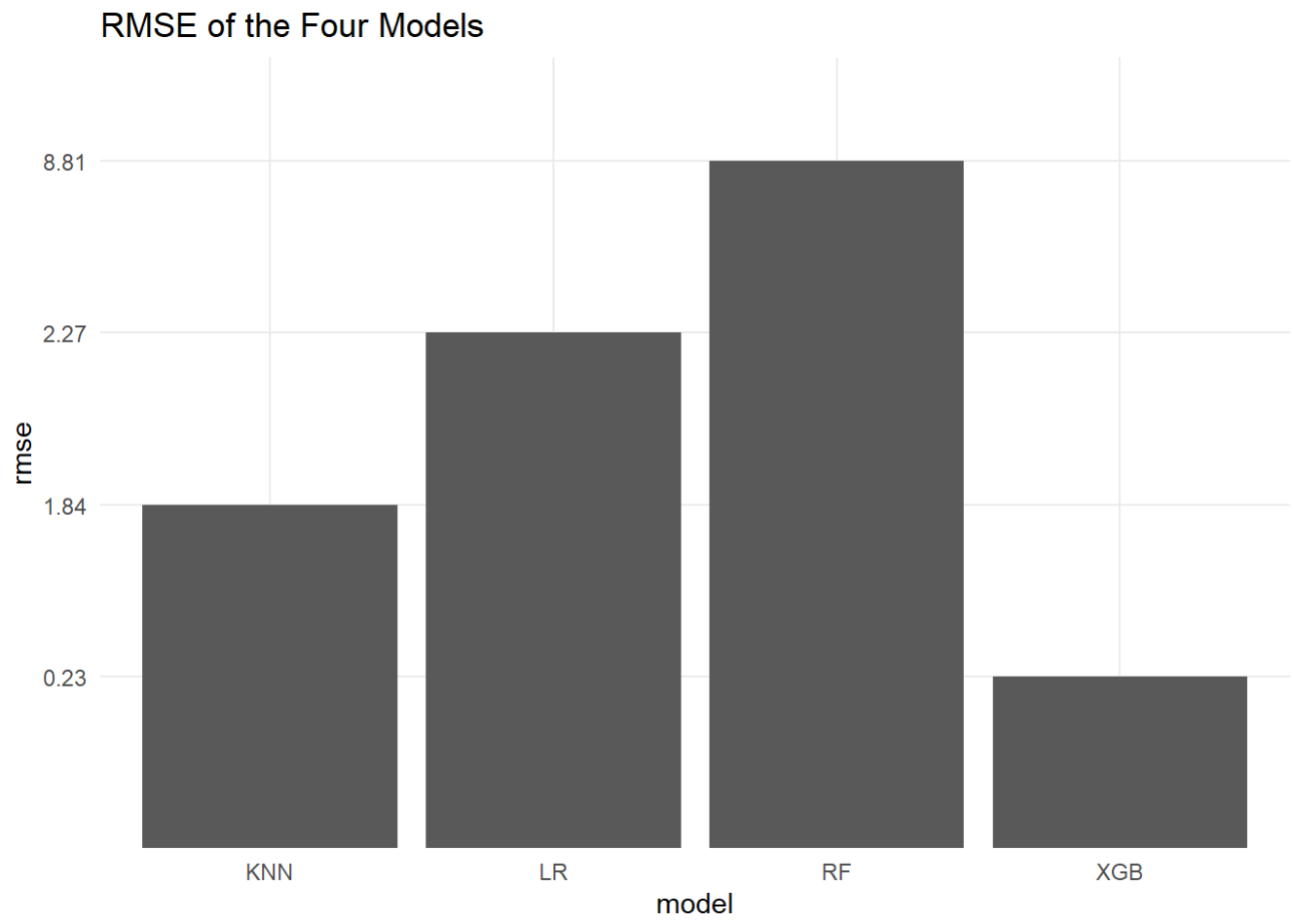
Best Model

The following code creates a table containing the RMSE and RSQ estimates for each of our four models.

```
set.seed(123)
# Create table of prediction metrics from training data
linreg = res1 |> collect_metrics() |> as.data.frame()
linreg = cbind(linreg[,1], round(linreg[,3],2))
knn = res2 %>% collect_metrics() |> as.data.frame()
knn = cbind(knn[,1], round(knn[,3],2))
xgboost = res3 |> filter(.metric!='mae') |> as.data.frame()
xgboost = cbind(xgboost[,1], xgboost[,3])
ranfor = res4 |> filter(.metric!='mae') |> as.data.frame()
ranfor = cbind(ranfor[,1], ranfor[,3])

compare = c('LR','LR','KNN','KNN','XGB','XGB','RF','RF') |> cbind(rbind(linreg,knn,xgboost,ranfo
r))
colnames(compare) = c('model','estimator','estimate')
compare = as.data.frame(compare) |> pivot_wider(names_from=estimator, values_from=estimate)
compare
```

```
## # A tibble: 4 × 3
##   model rmse  rsq
##   <chr> <chr> <chr>
## 1 LR    2.27  0.25
## 2 KNN   1.84  0.51
## 3 XGB   0.23  0.99
## 4 RF    8.81  0.91
```

```
# Visual of rmse for our 4 models
compare |> ggplot(aes(x=model, y=rmse)) + geom_bar(stat="identity") +
  labs(title='RMSE of the Four Models') + theme_minimal()
```




```
# Final model
# Fit XGB model to testing data
test_processed <- bake(rec3, new_data = testing(dat_split))

# Get pred. metrics based on testing data
test_prediction = xgboost_model_final |>
  parsnip::fit(formula = value ~ CMAQ+lat+county_area+lon+aod,
               data=train_processed) |>
  predict(new_data=test_processed) |>
  bind_cols(testing(dat_split))

res_final = test_prediction |>
  yardstick::metrics(value, .pred) |>
  mutate(.estimate=format(round(.estimate, 2), big.mark=","))
res_final
```

```
## # A tibble: 3 × 3
##   .metric .estimator .estimate
##   <chr>   <chr>      <chr>
## 1 rmse    standard   1.38
## 2 rsq     standard   0.68
## 3 mae     standard   1.05
```

As seen on the table and visual, Random Forest has the highest RMSE and lowest R-squared estimate. LR has the second highest RMSE with a moderate R-squared estimate. And XG Boost has the lowest RMSE of 0.23 and the highest R-squared estimate of 0.99. The kNN model has the both the second lowest RMSE and second highest R-squared estimate. Since we are choosing the best model by the best RMSE, which is the lowest one, the XG Boost model is thus the best and final model based on our procedure. After fitting the XG Boost model to the testing data, we find that its RMSE and R-squared is 1.38 and 0.68, respectively. Since we are trying to predict annual average ambient air pollution of a given monitor, then these statistics mean that the amount of error in our model's predictions is about 1.38 μg/m^3, and that the features we used in our model are able to explain 68% of the variance in the data, which means this model does not fit/predict the data well.

1. Based on test set performance, at what locations does your model give predictions that are closest and furthest from the observed values? What do you hypothesize are the reasons for the good or bad performance at these locations?

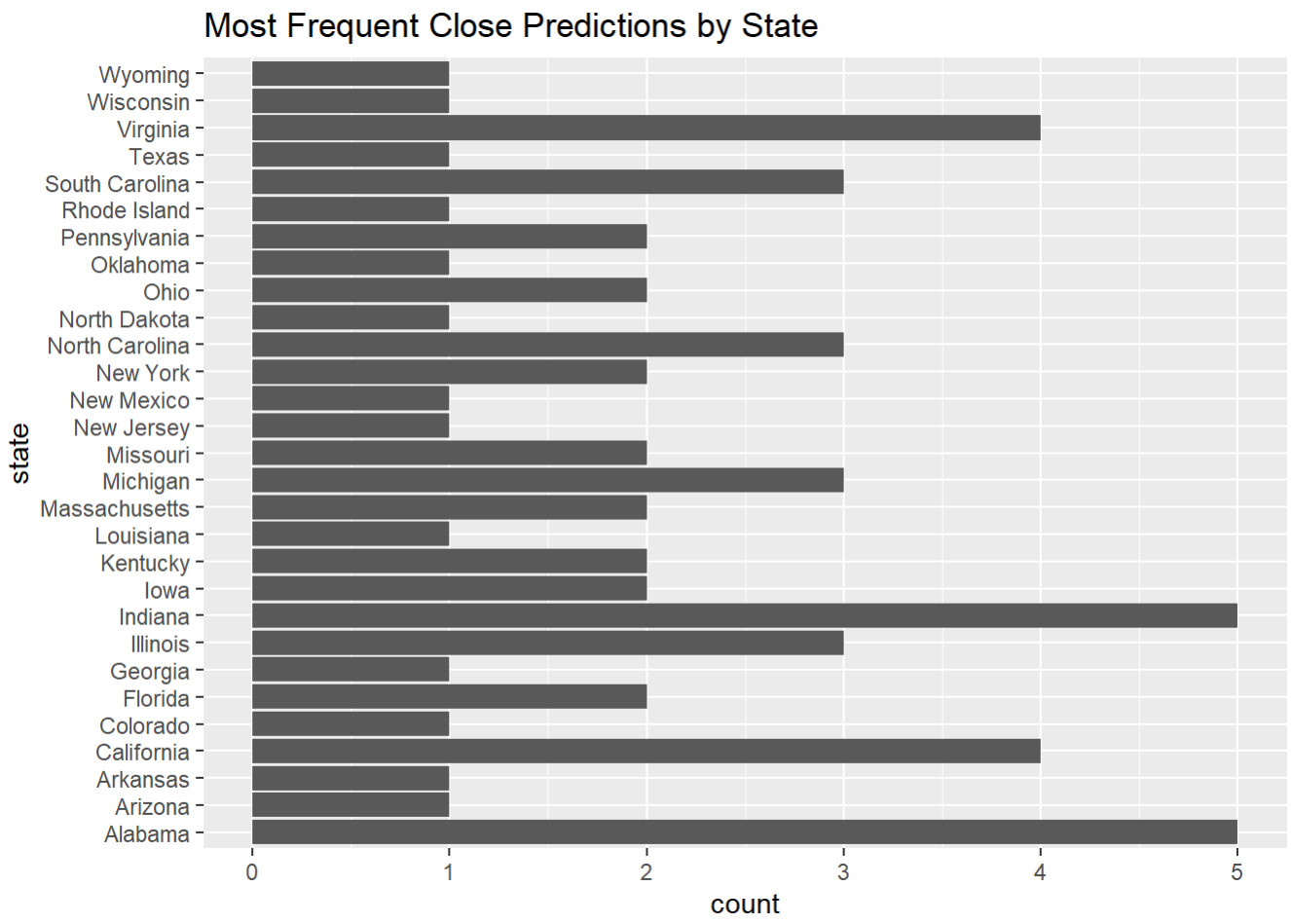
```
# Arrange obs.s by greatest diff
loc = test_prediction |>
  dplyr::select(.pred, value, CMAQ, lat, lon, county_area, aod, state) |>
  mutate(diff=round(abs(value-.pred),2)) |> arrange(desc(diff))

# Find number of obs.s per the 3 states
loc |>
  filter(state=='Oregon' | state=='Iowa' | state=='Virginia') |>
  group_by(state) |> summarise(mon=n())
```

```
## # A tibble: 3 × 2
##   state      mon
##   <chr>    <int>
## 1 Iowa         6
## 2 Oregon       5
## 3 Virginia     5
```

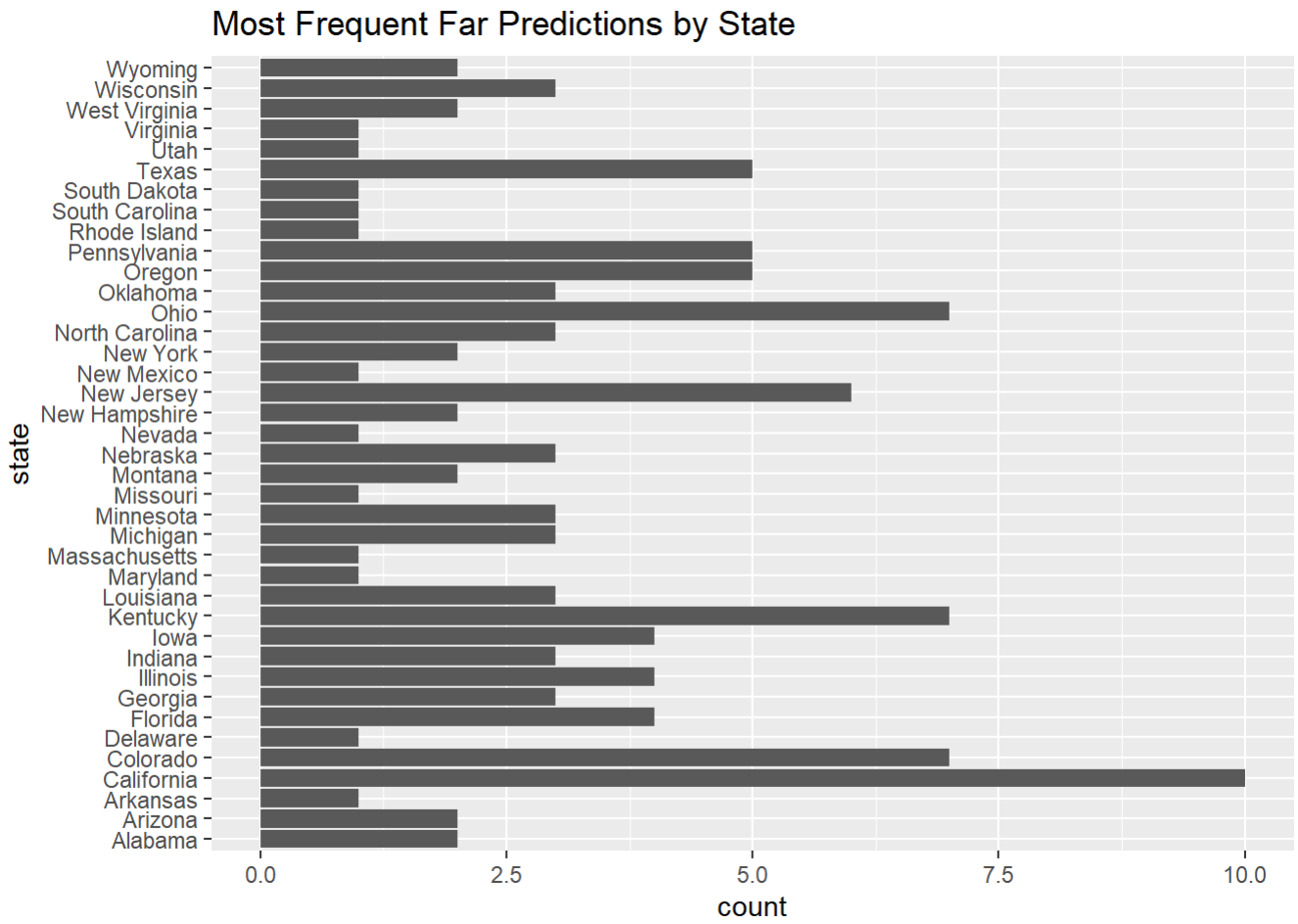
Based on the test set performance, it seems that our model gives the closest prediction for both Iowa and Virginia The model gives the furthest prediction for Oregon. We hypothesize that the model gives bad performance for Oregon because despite being clearly larger than both Iowa and Virginia, it has the same number of monitors as the latter states, making the data less representative of the variation in PM2.5 concentration for the state for the model to make a good prediction based on the features we chose.

2. What variables might predict where your model performs well or not? For example, are their regions of the country where the model does better or worse? Are there variables that are not included in this dataset that you think might improve the model performance if they were included in your model?

```
# Visual for the most frequently close predictions by state
loc |>filter(diff<0.5) |> ggplot(aes(y=state)) +
  geom_bar() +
  labs(title='Most Frequent Close Predictions by State')
```




```
# Visual for the most frequently furthest predictions by state
loc |> filter(diff>=0.5) |> ggplot(aes(y=state)) +
  geom_bar() + labs(title='Most Frequent Far Predictions by State')
```



'CMAQ', 'county_area', and 'aod' might predict where our model performs well or not. We define a good prediction by having an absolute difference of less than 0.5 between 'value' and '.pred'. Indiana, Ohio, Iowa, and Alabama most frequently have good predictions. So the model does fairly well at predicting some of the Midwest. Our model gives the furthest predictions for California, Colorado, Kentucky, Pennsylvania, and Texas most frequently. So our model does poorly at predicting the West and the South. We think that including historical data of ambient air pollution over the years would potentially improve the performance of the model because if we are able to track the air pollution over the years, we could get a better prediction for any given location.

3. There is interest in developing more cost-effect approaches to monitoring air pollution on the ground. Two candidates for replacing the use of ground-based monitors are numerical models like CMAQ and satellite-based observations such as AOD. How well do CMAQ and AOD predict ground-level concentrations of PM2.5? How does the prediction performance of your model change when CMAQ or AOD are included (or not included) in the model?

The model with only 'CMAQ' and 'aod' as features has an RMSE of 1.99. Since it is greater than 1, this means that 'CMAQ' and 'aod' are not good predictors for annual ambient air pollution in a given location. The RMSE of our model on the test data is 1.38. The RMSE of our model without 'CMAQ' and 'aod' is 1.73. This means that the model without 'CMAQ' and 'aod' has the better prediction performance since its RMSE is lower than our original model.

```
set.seed(123)
# XGB model w/only cmaq and aod
# Preprocessing recipe
rec5 <-
  recipes::recipe(value ~ CMAQ + aod, data = training(dat_split)) %>%
  # Convert categorical variables to factors
  recipes::step_string2factor(all_nominal()) %>%
  # Combine low frequency factor levels
  recipes::step_other(all_nominal(), threshold = 0.01) %>%
  # Remove no variance predictors which provide no predictive information
  recipes::step_nzv(all_nominal()) %>% prep()

folds5 <- recipes::bake(rec5, new_data = training(dat_split)) %>%
  rsample::vfold_cv(v = 5)

# XGBoost model specification
xgboost_model <- parsnip::boost_tree(mode = "regression", trees = 1000, min_n = tune(),
  tree_depth = tune(), learn_rate = tune(), loss_reduction = tune()) %>%
  set_engine("xgboost", objective = "reg:squarederror")

# Grid specification
xgboost_params <- dials::parameters(min_n(), tree_depth(), learn_rate(), loss_reduction())
xgboost_grid <- dials::grid_max_entropy(xgboost_params, size = 60)

xgboost_wf <- workflows::workflow() %>%
  add_model(xgboost_model) %>%
  add_formula(value ~ CMAQ+aod)

# Hyperparameter tuning
# Takes a long time to run, 5-8 min
xgboost_tuned <- tune::tune_grid(object = xgboost_wf, resamples = folds5,
  grid = xgboost_grid, metrics = yardstick::metric_set(rmse, rsq),
  control = tune::control_grid(verbose = TRUE))

# Find hyper parameter values which performed best at minimizing RMSE
xgboost_tuned %>% tune::show_best(metric = "rmse")
```

```
## # A tibble: 5 × 10
##   min_n tree_depth learn_rate loss_reduction .metric .estimator  mean     n
##   <int>      <int>      <dbl>          <dbl> <chr>   <chr>      <dbl> <int>
## 1    16         15    0.0229        1.45e+ 0 rmse    standard    2.01     5
## 2    22          8    0.0201        1.69e- 1 rmse    standard    2.07     5
## 3     2         11    0.0832        5.03e- 1 rmse    standard    2.07     5
## 4     7          9    0.0301        8.93e-10 rmse    standard    2.09     5
## 5    23          9    0.00597       3.47e-10 rmse    standard    2.10     5
## # ℹ 2 more variables: std_err <dbl>, .config <chr>
```

```
# Isolate the best performing hyperparameter values.
xgboost_best_params <- xgboost_tuned %>% tune::select_best("rmse")
xgboost_best_params
```

```
## # A tibble: 1 × 5
##   min_n tree_depth learn_rate loss_reduction .config
##   <int>      <int>      <dbl>          <dbl> <chr>
## 1    16         15     0.0229           1.45 Preprocessor1_Model42
```

```
# Final boost model
xgboost_model_final <- xgboost_model %>% finalize_model(xgboost_best_params)

# Eval. model performance on train data
test_processed <- bake(rec5,  new_data = testing(dat_split))
train_processed <- bake(rec5,  new_data = training(dat_split))

test_prediction = xgboost_model_final |>
  parsnip::fit(formula = value ~ CMAQ+aod, data=train_processed) |>
  predict(new_data=test_processed) |>
  bind_cols(testing(dat_split))

res5 = test_prediction |>
  yardstick::metrics(value, .pred) |>
  mutate(.estimate=format(round(.estimate, 2), big.mark=","))
res5
```

```
## # A tibble: 3 × 3
##   .metric .estimator .estimate
##   <chr>   <chr>      <chr>
## 1 rmse    standard   1.99
## 2 rsq     standard   0.36
## 3 mae     standard   1.43
```

```
set.seed(123)
# XGB model w/o cmaq and aod
# Preprocessing recipe
rec8 <-
  recipes::recipe(value ~ lat+county_area+lon, data = training(dat_split)) %>%
  # Convert categorical variables to factors
  recipes::step_string2factor(all_nominal()) %>%
  # Combine low frequency factor levels
  recipes::step_other(all_nominal(), threshold = 0.01) %>%
  # Remove no variance predictors which provide no predictive information
  recipes::step_nzv(all_nominal()) %>%
  prep()

folds8 <- recipes::bake(rec8, new_data = training(dat_split)) %>%
  rsample::vfold_cv(v = 5)

# XGBoost model specification
xgboost_model <- parsnip::boost_tree(mode = "regression", trees = 1000, min_n = tune(),
  tree_depth = tune(), learn_rate = tune(), loss_reduction = tune()) %>%
  set_engine("xgboost", objective = "reg:squarederror")

# Grid specification
xgboost_params <- dials::parameters(min_n(), tree_depth(), learn_rate(), loss_reduction())
xgboost_grid <- dials::grid_max_entropy(xgboost_params, size = 60)

xgboost_wf <- workflows::workflow() %>%
  add_model(xgboost_model) %>%
  add_formula(value ~ lat+county_area+lon)

# Hyperparameter tuning
# Takes a long time to run, 5-8 min
xgboost_tuned <- tune::tune_grid(object = xgboost_wf, resamples = folds8,
  grid = xgboost_grid, metrics = yardstick::metric_set(rmse, rsq),
  control = tune::control_grid(verbose = TRUE))

# Find hyper parameter values which performed best at minimizing RMSE
xgboost_tuned %>% tune::show_best(metric = "rmse")
```

```
## # A tibble: 5 × 10
##   min_n tree_depth learn_rate loss_reduction .metric .estimator  mean     n
##   <int>      <int>      <dbl>          <dbl> <chr>   <chr>      <dbl> <int>
## 1     7          9    0.0301        8.93e-10 rmse    standard    1.68     5
## 2     2         11    0.0832        5.03e- 1 rmse    standard    1.69     5
## 3    16         15    0.0229        1.45e+ 0 rmse    standard    1.70     5
## 4    22          8    0.0201        1.69e- 1 rmse    standard    1.74     5
## 5    23          9    0.00597       3.47e-10 rmse    standard    1.79     5
## # ℹ 2 more variables: std_err <dbl>, .config <chr>
```

```
# Isolate the best performing hyperparameter values.
xgboost_best_params <- xgboost_tuned %>% tune::select_best("rmse")
xgboost_best_params
```

```
## # A tibble: 1 × 5
##   min_n tree_depth learn_rate loss_reduction .config
##   <int>      <int>      <dbl>          <dbl> <chr>
## 1     7          9     0.0301       8.93e-10 Preprocessor1_Model44
```

```
# Final boost model
xgboost_model_final <- xgboost_model %>% finalize_model(xgboost_best_params)

# Eval. model performance on train data
train_processed = bake(rec8, new_data=training(dat_split))
test_processed <- bake(rec8, new_data = testing(dat_split))

test_prediction = xgboost_model_final |>
  parsnip::fit(formula = value ~ lat+county_area+lon, data=train_processed) |>
  predict(new_data=test_processed) |>
  bind_cols(testing(dat_split))

res8 = test_prediction |>
  yardstick::metrics(value, .pred) |>
  mutate(.estimate=format(round(.estimate, 2), big.mark=","))
res8
```

```
## # A tibble: 3 × 3
##   .metric .estimator .estimate
##   <chr>   <chr>      <chr>
## 1 rmse    standard   1.73
## 2 rsq     standard   0.52
## 3 mae     standard   1.13
```

4. The dataset here did not include data from Alaska or Hawaii. Do you think your model will perform well or not in those two states? Explain your reasoning.

Given that the top 5 features were ‘CMAQ’, ‘lat’, ‘county_area’, ‘lon’, and ‘aod’. We think that XG Boost model will not perform well since our model did not perform well on the test dataset. It seems like we overfit our model, so fitting it to data from Hawaii and Alaska, which we expect to be veyr different from our training dataset, will yield similar results to fitting our model to the test dataset.

Discussion

Putting it all together, what did you learn from your data and your model performance? Answer your Question posed above, citing any supporting statistics, visualizations, or results from the data or your models.

We learned that we overfit our Xgboost model to the data, which is why the prediction performance was poor for the testing dataset. This can be seen in our model having an RMSE value of 1.38 despite the RMSE of the model fit to the training data being 0.23. In general, we learned that understanding the disadvantages and advantages of each model is helpful toward understanding why a model fails or succeeds at the training and testing stages. For instance, we learned the Xgboost is helpful for unbalanced data and that Random Forest Regression performs well for datasets with a lot of noise.

Reflect on the process of conducting this project. What was challenging, what have you learned from the process itself?

The most challenging aspect of this project was figuring out the process to determine which predictors were the best to use for our models. We ran into many debugging issues and Googled a lot to resolve our uncertainties. We learned that finding the best predictors for a model can be a lengthy process and that running a new model (e.g. preprocessing and hyperparameter tuning steps for XG Boost and Random Forest Regression) often may require lots of research into what is needed for the model to run before it can actually run.

Reflect on the performance of your final prediction model. Did it perform as well as you originally expected? If not, why do you think it didn't perform as well?

The RMSE of our model was 1.38 on the testing dataset, which means it does not predict well. It did not perform as well as we originally expected, especially since we saw that the RMSE of our model was 0.23 on the training dataset. We believe the model did not perform well because we overfit the model to the training data. This means that the model gives inaccurate predictions and cannot perform well for all types of new data, which is why despite the low RMSE on the training data, the model has poor prediction performance on the testing data.

Include acknowledgments for any help received. If a group project, report the contribution of each member (i.e. who did what?). ## Acknowledgements **Aileen Li & Nyah Strickland:**

- Intro
- Wrangling
- Results
- Discussion
- Formatting