Building a Prediction Model for PM2.5 Values across the US

2023-04-19

Names and EID: Solomon Leo (sal4294), Vardhan Koripally (vrk385)

```
# Install required packages
library(tidyverse)
```

```
## ── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──
## ✔ dplyr     1.1.1     ✔ readr     2.1.4
## ✔ forcats   1.0.0     ✔ stringr   1.5.0
## ✔ ggplot2   3.4.1     ✔ tibble    3.2.1
## ✔ lubridate 1.9.2     ✔ tidyr     1.3.0
## ✔ purrr     1.0.1
## ── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
## ✖ dplyr::filter() masks stats::filter()
## ✖ 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 ──
## ✔ broom        1.0.4     ✔ rsample      1.1.1
## ✔ dials        1.2.0     ✔ tune         1.1.1
## ✔ infer        1.0.4     ✔ workflows    1.1.3
## ✔ modeldata    1.1.0     ✔ workflowsets 1.0.1
## ✔ parsnip      1.1.0     ✔ yardstick    1.1.0
## ✔ recipes      1.0.5
## ── Conflicts ────────────────────────────────────── tidymodels_conflicts() ──
## ✖ scales::discard() masks purrr::discard()
## ✖ dplyr::filter()   masks stats::filter()
## ✖ recipes::fixed()  masks stringr::fixed()
## ✖ dplyr::lag()      masks stats::lag()
## ✖ yardstick::spec() masks readr::spec()
## ✖ recipes::step()   masks stats::step()
## • Search for functions across packages at https://www.tidymodels.org/find/
```

```
library(doParallel)
```

```
## Loading required package: foreach
##
## Attaching package: 'foreach'
##
## The following objects are masked from 'package:purrr':
##
##     accumulate, when
##
## Loading required package: iterators
## Loading required package: parallel
```

```
# Download data set
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.
```

```
# Take a look at data
head(dat)
```

```
## # A tibble: 6 × 50
##      id value  fips   lat   lon state   county  city        CMAQ  zcta zcta_area
##   <dbl> <dbl> <dbl> <dbl> <dbl> <chr>   <chr>   <chr>      <dbl> <dbl>     <dbl>
## 1 1003.  9.60  1003  30.5 -87.9 Alabama Baldwin Fairhope    8.10 36532 190980522
## 2 1027. 10.8   1027  33.3 -85.8 Alabama Clay    Ashland     9.77 36251 374132430
## 3 1033. 11.2   1033  34.8 -87.7 Alabama Colbert Muscle Sh…  9.40 35660  16716984
## 4 1049. 11.7   1049  34.3 -86.0 Alabama DeKalb  Crossville  8.53 35962 203836235
## 5 1055. 12.4   1055  34.0 -86.0 Alabama Etowah  Gadsden     9.24 35901 154069359
## 6 1069. 10.5   1069  31.2 -85.4 Alabama Houston Dothan      9.12 36303 162685124
## # ℹ 39 more variables: zcta_pop <dbl>, imp_a500 <dbl>, imp_a1000 <dbl>,
## #   imp_a5000 <dbl>, imp_a10000 <dbl>, imp_a15000 <dbl>, county_area <dbl>,
## #   county_pop <dbl>, log_dist_to_prisec <dbl>, log_pri_length_5000 <dbl>,
## #   log_pri_length_10000 <dbl>, log_pri_length_15000 <dbl>,
## #   log_pri_length_25000 <dbl>, log_prisec_length_500 <dbl>,
## #   log_prisec_length_1000 <dbl>, log_prisec_length_5000 <dbl>,
## #   log_prisec_length_10000 <dbl>, log_prisec_length_15000 <dbl>, …
```

Introduction Because of the impracticality of installing particle monitors in numerous locations of the US, it is desirable to build a prediction model of PM2.5 based on multiple different predictor variables. This report discusses the performance of four different machine learning models based on 9 different predictor variables.

The models that will be used are linear regression, random forest, kNN, and a single-layer neural network. Linear regression is a simple prediction model that attaches coefficients to each predictor variable. Random forest creates a decision tree and averages all of the trees to predict the values. kNN makes predictions based on the values of the neighbors with the closest variable values. A single layer neural network predicts by detecting subtle relationships between the predictor variables and the values.

The 9 predictors being used are CMAQ, zcta_pop, imp_a15000, county_pop, log_prisec_length_25000, popdens_county, popdens_zcta, urc2013, aod.

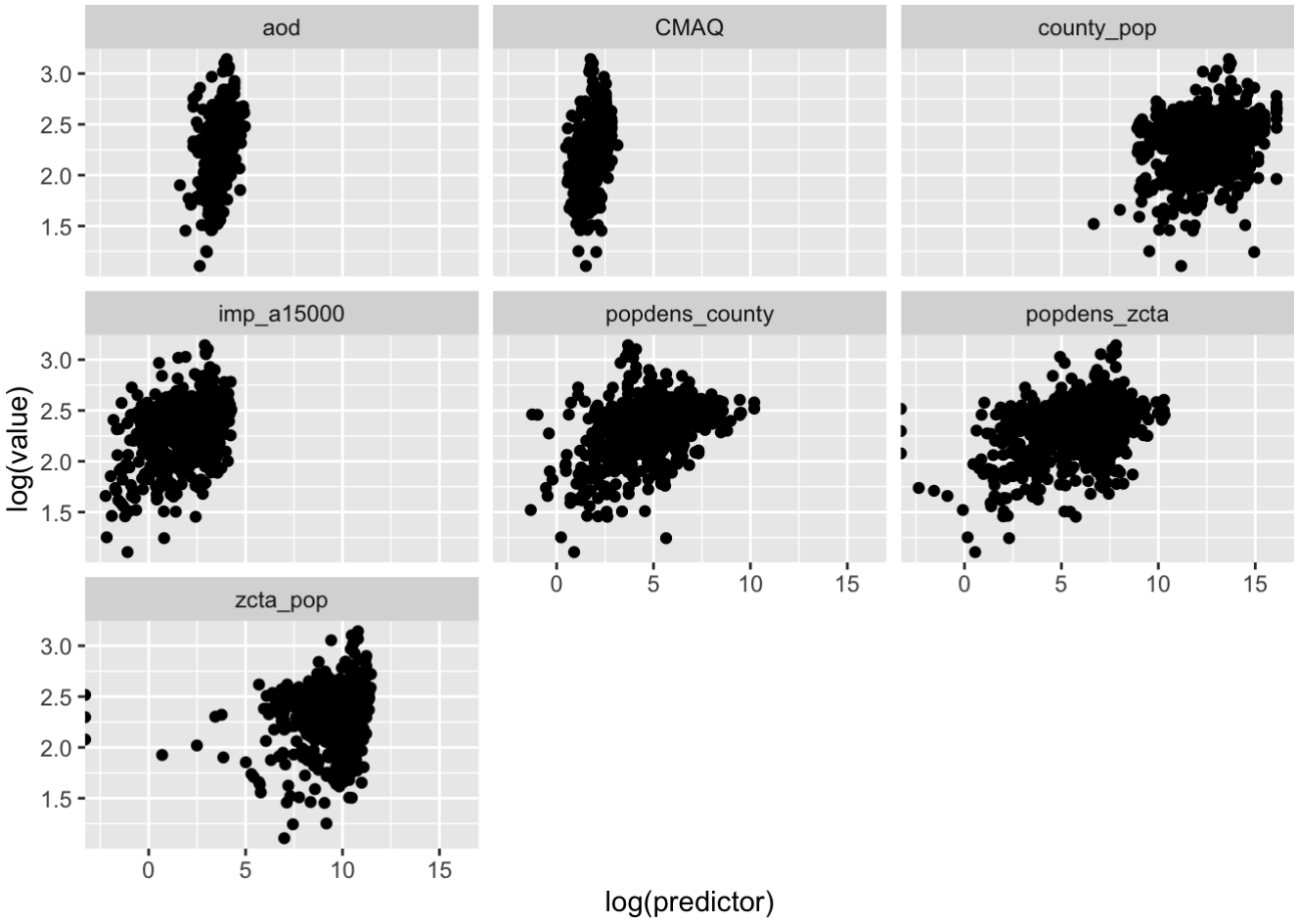
Two of the variables in the data set, CMAQ and aod, compute the particulate pollution in the area using different input data. CMAQ is from the US EPA and uses weather data to predict air pollution, and aod is from NASA and uses satellite data to estimate pollution from the diffraction of a laser. Assuming these computational models are relatively accurate, they will be highly correlated with the actual observed PM2.5, which makes them good predictors.

We predict that location the monitor being close to roads and urban infrastructure will increase the PM2.5 levels, because the cars found on the roads and near the urban infrastructure will pollute the air. This is why the predictors imp_a15000 (measure of impermeable surfaces in a 1,500 meter radius), and log_prisec_length_25000 (log of primary and secondary road length in a 25,000 meter radius).

We also predict that population and population density are highly correlated with PM2.5 because increased population correlates with more cars and construction, which cause pollution. This is why the predictors zcta_pop (zip code population), county_pop (county population), popdens_county (county population density), popdens_zcta (zip code population density), and urc2013 (urban-rural classification) were added as well.

Visualizations of some of the predictors are shown below.

```
# Select only the columns that we expect to be useful and plot log transformed relations
hips
dat3 = dat %>% select(id, value, CMAQ, zcta_pop, imp_a15000, county_pop, popdens_county,
popdens_zcta, aod)
dat3 |> pivot_longer(values_to = "predictor", cols = c(3:9)) |> group_by(name) |> ggplot
(aes(x = log(predictor), y = log(value))) + geom_point() + facet_wrap(~name)
```



These visualizations show a relationship between the predictor variables and PM2.5 values.

We expect random forest to perform the best with a prediction RMSE of around 2.0, kNN following closely at around 2.4, the single layer neural network to be 2.8, and the linear regression to be the worst model at around 3.0.

Wrangling

First we select the predictor variables and save them to a new tibble.

```
# Select predictors for models
dat2 = dat %>% select(id, value, CMAQ, zcta_pop, imp_a15000, county_pop, log_prisec_leng
th_25000, popdens_county, popdens_zcta, urc2013, aod)

# Look at data
dat2 |> head()
```

```
## # A tibble: 6 × 11
##      id value  CMAQ zcta_pop imp_a15000 county_pop log_prisec_length_25000
##   <dbl> <dbl> <dbl>    <dbl>      <dbl>      <dbl>                    <dbl>
## 1 1003.  9.60  8.10    27829      1.44      182265                     13.4
## 2 1027. 10.8   9.77     5103      0.336      13932                     12.8
## 3 1033. 11.2   9.40     9042      5.25       54428                     13.8
## 4 1049. 11.7   8.53     8300      0.973      71109                     13.6
## 5 1055. 12.4   9.24    20045      5.16      104430                     13.7
## 6 1069. 10.5   9.12    30217      4.74      101547                     13.9
## # ℹ 4 more variables: popdens_county <dbl>, popdens_zcta <dbl>, urc2013 <dbl>,
## #   aod <dbl>
```

To perform train, test, split we must split the data into two separate sets. The `train` set will be 80% of the data sampled randomly from the initial data and the `test` set will be the remaining 20%. The models will be tuned and trained on the `train` set and tested by evaluating the RMSE on the `test` set. The `test` set is created by anti joining the `train` set to `dat2`, which selects all elements that do not have a matching key. We also remove the ID variable from the `train` and `test` sets.

```
# Set seed to make data reproducible
set.seed(2003)

# Split data into training and test set
train = sample_frac(dat2, size = 0.8)
test = dat2 %>% anti_join(train, by = "id")

train = train %>% select(-c(id))
test = test %>% select(-c(id))

# Observe final train and test sets
train %>% head()
```

```
## # A tibble: 6 × 10
##   value  CMAQ zcta_pop imp_a15000 county_pop log_prisec_length_25000
##   <dbl> <dbl>    <dbl>      <dbl>      <dbl>                    <dbl>
## 1 16.7   6.81    30805      4.46      255793                     12.7
## 2 11.1  10.5      5532      0.776      21757                     12.6
## 3 10.7   8.89    45099     14.5       147546                     13.8
## 4 11.6  11.4     36867     10.3      1510271                     12.5
## 5 12.5   9.72     7701     27.9       674158                     14.6
## 6  6.71  5.44     2853      1.41      189927                     14.1
## # ℹ 4 more variables: popdens_county <dbl>, popdens_zcta <dbl>, urc2013 <dbl>,
## #   aod <dbl>
```

```
test %>% head()
```

```
## # A tibble: 6 × 10
##   value  CMAQ zcta_pop imp_a15000 county_pop log_prisec_length_25000
##   <dbl> <dbl>    <dbl>      <dbl>      <dbl>                    <dbl>
## 1 12.4  10.2     16140       4.30     658466                     13.8
## 2 10.0   8.65     6106       7.72     412992                     13.9
## 3 12.0  10.2     12997      10.9      229363                     14.1
## 4 11.4   8.82    30545       4.18     119490                     13.7
## 5 11.6  10.2     23020       7.98     195085                     14.1
## 6  8.85 12.2     37644      39.1     3817117                     14.1
## # ℹ 4 more variables: popdens_county <dbl>, popdens_zcta <dbl>, urc2013 <dbl>,
## #   aod <dbl>
```

Results

The four models chosen are linear regression, random forest, kNN, and a single layer neural network. All models except linear regression have hyperparamters that can be tuned. The primary metric for checking the performance of the models is RMSE. The RMSE of the model is computed both by 10 cross fold validation and by using predictions based on the test data set. The RMSE from the cross fold validation is an indicator of the stability of the model and the RMSE from the predictions is a measure of the prediction capability of the model. In this report k-fold cross validation is used to select model parameters and estimate model performance.

Since it was expected that several predictors had a linear relationship with the PM2.5 level, a linear model was chosen first. Since it has no hyperparameters to be tuned it can be set up and run quickly. To do this, the model was trained on the training set and was then validated using RMSE from predictions and 10 cross fold validation.

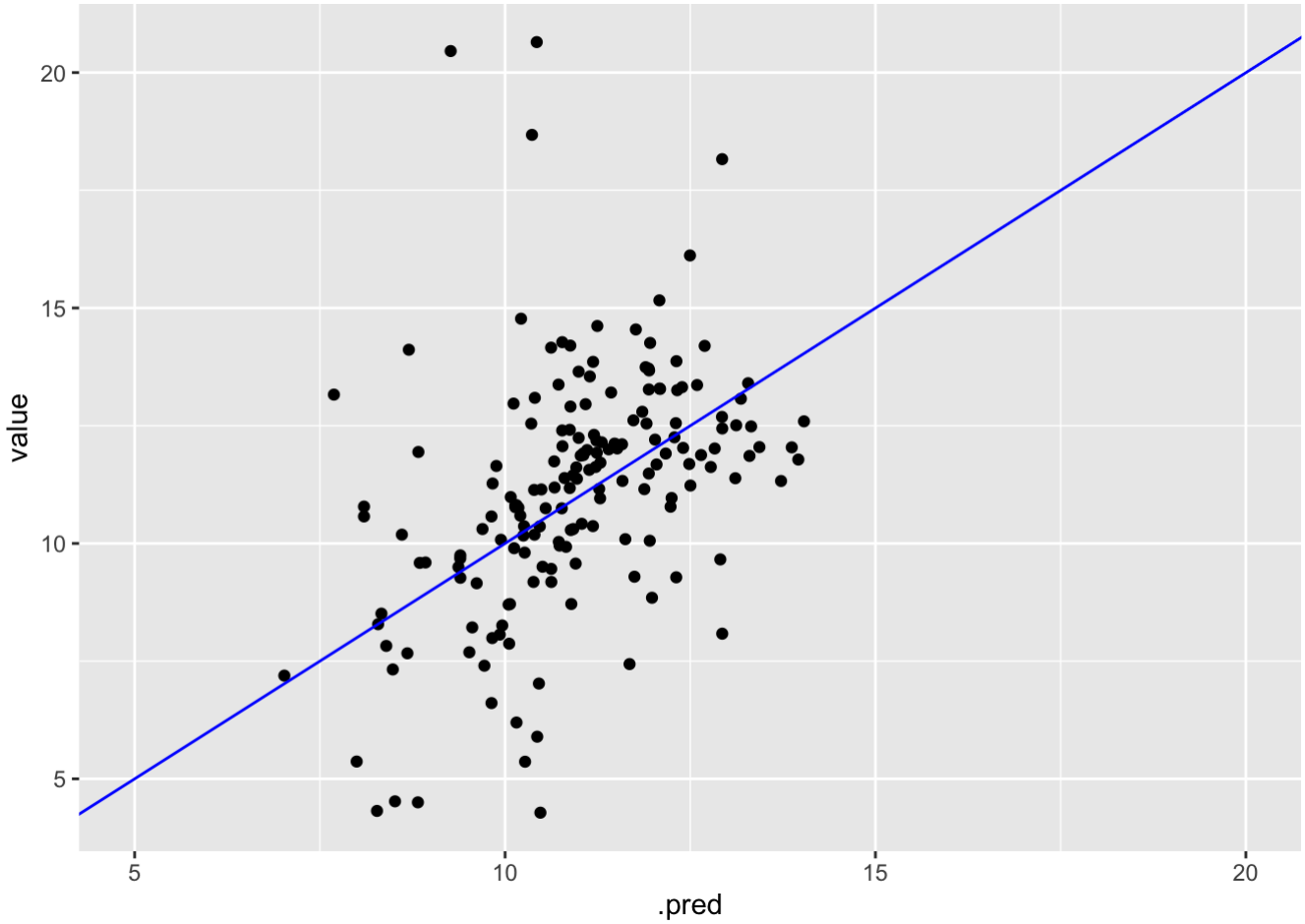
```
# Set seed for reproducible
doParallel::registerDoParallel()
set.seed(2163)

# Create formula, model, and workflow
fom = train %>% formula(value ~ .)
lm_model = linear_reg(mode = "regression")
lm_wf = workflow() %>% add_formula(fom) %>% add_model(lm_model)

# Create 10 folds
lm_folds = vfold_cv(dat2, v = 10)

# Create Linear model based on workflow and training data
lm_fit = fit(lm_wf, train)

# Estimate model performance on training data
lm_validation = fit_resamples(lm_wf, resamples = lm_folds)
lm_validation %>% collect_metrics()
```

```
## # A tibble: 2 × 6
##   .metric .estimator  mean     n std_err .config
##   <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1 rmse    standard   2.19     10  0.106  Preprocessor1_Model1
## 2 rsq     standard   0.280    10  0.0310 Preprocessor1_Model1
```

```
# Compute predictions based on test data
values = test %>% select(value)
lin_pred = cbind(values, predict(lm_fit, new_data = test))
sqrt(mean((lin_pred$value - lin_pred$.pred)^2))
```

```
## [1] 2.353827
```

```
# Plot predictions against values
lin_pred |> ggplot(aes(x = .pred, y = value)) + geom_point() +
  geom_abline(intercept = 0, slope = 1, color = "blue") + xlim(5,20)
```



Since this model has no parameters, the model can be trained immediately. The RMSE from the 10 fold cross validation was found to be 2.191216, which is quite good for a linear model. Since the model is robust and has a relatively low RMSE on the training set we can compute predictions and compute the prediction RMSE, which was found to be 2.353827. This is not as good and suggests that the linear model is not very good at prediction. Finally we can plot the predictions against the known values to estimate model performance. Many points are close to the line, but there are also a large amount of ones very far from the line. This confirms that the prediction capabilities of this model are not very good and another model should be chosen.

The next model used is random forest. This model uses decision trees to predict an outcome based on the selected predictors. As opposed to linear regression, random forest has three hyperparameters that can be tuned: mtry, trees, and min_n. These represent the number of samples when constructing trees, the number of trees, and the minimum tree depth repsectively. In order to tune this model a tuning model and workflow are set up with

each parameter set to tune. 5 folds are then created to be used for model tuning. A grid is also created with all parameter combinations in it. The grid is then used to compute a set of models using the hyperparameters to select the best model.

```
# Set seed for reproducibility
set.seed(2530)
doParallel::registerDoParallel()

# Create the recipe
rf_rec = recipe(value ~ ., data = train)

# Create the model with tuning hyperparameters
tune_spec = rand_forest(mtry = tune(), trees = tune(), min_n = tune()) %>%
  set_mode("regression") %>% set_engine("ranger")

# Create tuning workflow
tune_wf = workflow() %>% add_recipe(rf_rec) %>% add_model(tune_spec)

# Create 5 folds
rf_folds = vfold_cv(train, v = 5)

# Create grid of assortment of values for hyperparameters
rf_grid = grid_regular(mtry(range(c(10,30))), trees(), min_n(), levels = 5)

# Compute relevant statistics for each combination of hyperparameter
tune_res = tune_wf %>%
  tune_grid(resamples = rf_folds, grid = rf_grid)

# Take a look at all models in descending order of RMSE
tune_res %>% collect_metrics() |> filter(.metric == "rmse") |> arrange(mean)
```

```
## # A tibble: 125 × 9
##     mtry trees min_n .metric .estimator  mean     n std_err .config
##    <int> <int> <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
##  1    15   500     2 rmse    standard    1.82     5  0.0654 Preprocessor1_Model…
##  2    20  2000     2 rmse    standard    1.82     5  0.0651 Preprocessor1_Model…
##  3    30   500     2 rmse    standard    1.82     5  0.0714 Preprocessor1_Model…
##  4    15  1500     2 rmse    standard    1.82     5  0.0654 Preprocessor1_Model…
##  5    15  1000     2 rmse    standard    1.82     5  0.0665 Preprocessor1_Model…
##  6    20  1500     2 rmse    standard    1.82     5  0.0683 Preprocessor1_Model…
##  7    30  1000     2 rmse    standard    1.82     5  0.0659 Preprocessor1_Model…
##  8    30  2000     2 rmse    standard    1.83     5  0.0680 Preprocessor1_Model…
##  9    30  1500     2 rmse    standard    1.83     5  0.0676 Preprocessor1_Model…
## 10    25  1000     2 rmse    standard    1.83     5  0.0687 Preprocessor1_Model…
## # ℹ 115 more rows
```

The best model had 500 trees, a minimum tree depth of 2, and minimum sample size of 15. The RMSE for this model is 1.819842. We can now select the best model based on the RMSE and use it to finalize the model and workflow. We can then train this final model on the training data set. This trained model is then used to perform 10 fold cross validation to estimate model stability and performance on the training set.

```
set.seed(2530)
# Select best model and finalize workflow
best_rf = tune_res %>% select_best("rmse")
final_wf_rf = tune_wf %>% finalize_workflow(best_rf)
rf_fit = final_wf_rf %>% fit(data = train)
```

```
## Warning: 15 columns were requested but there were 9 predictors in the data. 9
## will be used.
```

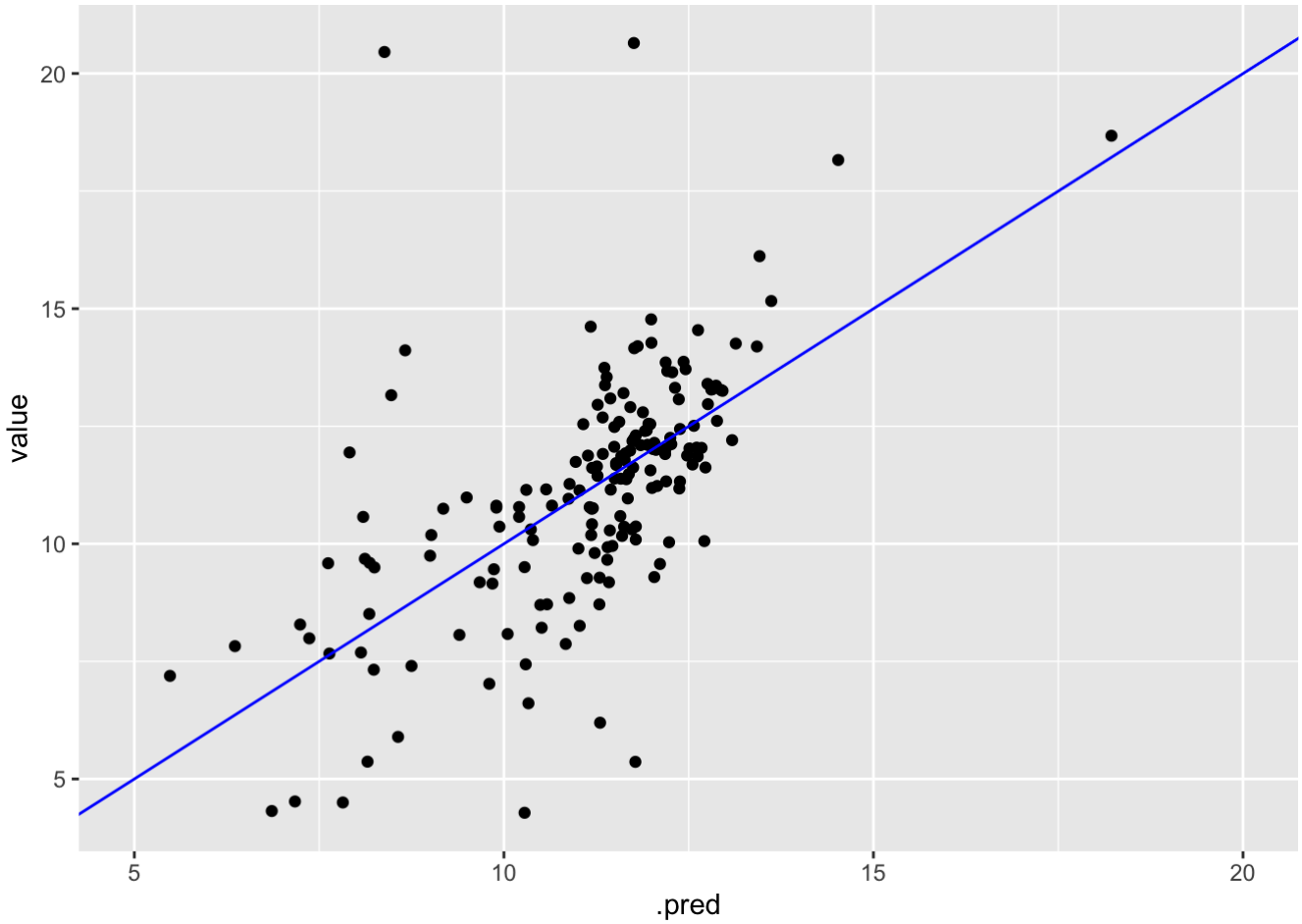
```
# Perform 10 cross fold validation on training set
rf_folds = vfold_cv(dat2, v = 10)
rf_validation = fit_resamples(final_wf_rf, resamples = rf_folds)
rf_validation %>% collect_metrics()
```

```
## # A tibble: 2 × 6
##   .metric .estimator  mean     n std_err .config
##   <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1 rmse    standard   1.86     10  0.0401 Preprocessor1_Model1
## 2 rsq     standard   0.480    10  0.0326 Preprocessor1_Model1
```

```
# Compute predictions for test data
rf_pred = cbind(values, predict(rf_fit, new_data = test))
sqrt(mean((rf_pred$value - rf_pred$.pred)^2))
```

```
## [1] 2.049121
```

```
# Plot predictions against values
rf_pred |> ggplot(aes(x = .pred, y = value)) + geom_point() +
  geom_abline(intercept = 0, slope = 1, color = "blue") + xlim(5,20)
```



The RMSE from 10 fold cross validation was found to be 1.8609770, which is much lower than that of linear regression. This suggests that random forest is a better model than linear regression since it is more robust. It also suggests that this model is more robust. Now that the model has been validated we can compute predictions using the test set. These predictions were used to find the RMSE which was 2.049121. This is much lower than that of the linear regression model which shows that random forest is also a much better prediction model. Finally the predictions can be plotted against the known values. The predictions seem to fall much better on the line which confirms that the random forest model is better than the linear model.

For the kNN model, the only hyperparameter is the number of neighbors. Similar to the random forest model, the number of neighbors is directly related to the strength of the model. The goal of this hyperparameter tuning is to minimze the number of neighbors while maximizing model performace and minimizing computation time. Again a tuning workflow and five folds are created. The model to be tuned is also created along with a grid of tuning parameters. The model is then computed across them and the best model is chosen.

```
# Set seed for reproducibility
set.seed(5926)

# Create recipe
knn_rec = recipe(value ~ ., data = train)

# Create kNN model with tuning parameters
tune_knn = nearest_neighbor(neighbors = tune()) %>% set_mode("regression") %>% set_engin
e("kknn")

# Create tuning workflow
knn_tune_wf = workflow() %>% add_recipe(knn_rec) %>% add_model(tune_knn)

# Create 5 folds
knn_folds = vfold_cv(train, v = 5)

# Create grid of tuning parameters
knn_grid = grid_regular(neighbors(range(7,20)), levels = 10)

# Create model for all tuning parameters in grid
knn_tune_res = knn_tune_wf %>%
  tune_grid(resamples = knn_folds, grid = knn_grid)

# Show relevant statistics for tuned models
knn_tune_res %>% collect_metrics() |> filter(.metric == "rmse") |> arrange(mean)
```

```
## # A tibble: 10 × 7
##    neighbors .metric .estimator  mean     n std_err .config
##        <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
##  1        17 rmse    standard    2.02     5   0.116 Preprocessor1_Model08
##  2        18 rmse    standard    2.02     5   0.117 Preprocessor1_Model09
##  3        20 rmse    standard    2.02     5   0.118 Preprocessor1_Model10
##  4        15 rmse    standard    2.02     5   0.114 Preprocessor1_Model07
##  5        14 rmse    standard    2.02     5   0.113 Preprocessor1_Model06
##  6        12 rmse    standard    2.03     5   0.109 Preprocessor1_Model05
##  7        11 rmse    standard    2.03     5   0.108 Preprocessor1_Model04
##  8         9 rmse    standard    2.04     5   0.104 Preprocessor1_Model03
##  9         8 rmse    standard    2.05     5   0.103 Preprocessor1_Model02
## 10         7 rmse    standard    2.06     5   0.100 Preprocessor1_Model01
```

The best model was found to have 17 neighbors and had an average RMSE of 2.020126. The best model is selected based on RMSE and is used to create the final workflow and model. This model is then trained on the training set.

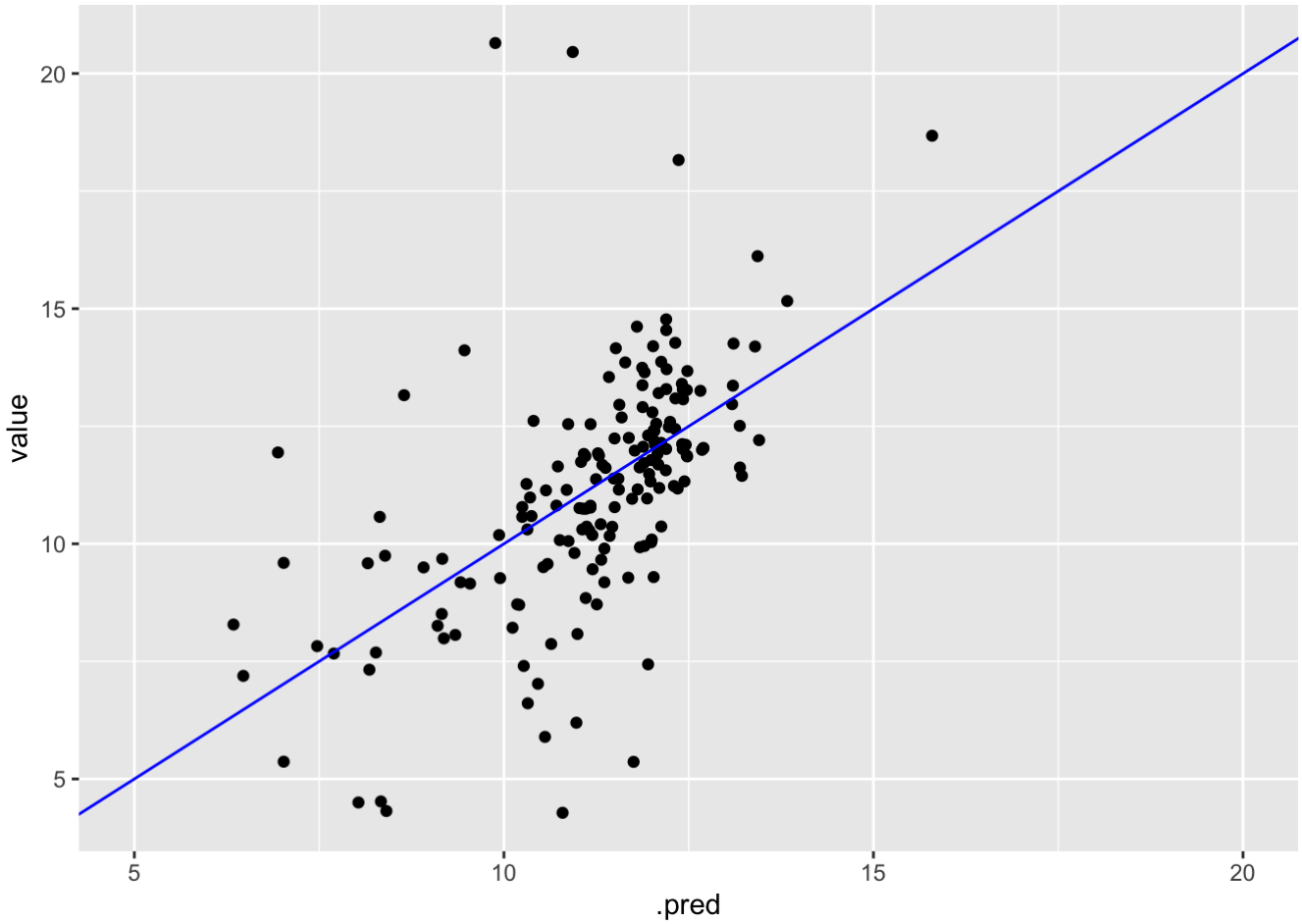
```
set.seed(2024)
# Select best model and create final model
best_knn = knn_tune_res %>% select_best("rmse")
final_wf_knn = knn_tune_wf %>% finalize_workflow(best_knn)
knn_fit = final_wf_knn %>% fit(data = train)

# Compute predictions and RMSE from final model
knn_pred = cbind(values, predict(knn_fit, new_data = test))
sqrt(mean((knn_pred$value - knn_pred$.pred)^2))
```

```
## [1] 2.113405
```

```
# Perform 10 fold cross validation on the model
knn_folds = vfold_cv(train, v = 10)
knn_validation = fit_resamples(final_wf_knn, resamples = knn_folds)
knn_validation %>% collect_metrics()
```

```
## # A tibble: 2 × 6
##   .metric .estimator  mean     n std_err .config
##   <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1 rmse    standard   1.99     10  0.0827 Preprocessor1_Model1
## 2 rsq     standard   0.399    10  0.0328 Preprocessor1_Model1
```

```
# Plot predictions against values
knn_pred |> ggplot(aes(x = .pred, y = value)) + geom_point() +
  geom_abline(intercept = 0, slope = 1, color = "blue") + xlim(5,20)
```



The mean RMSE from 10 fold cross vaildation was found to be 1.9931279, which is higher than that of the random forest model. This suggests that this model is not as robust. The final model is then used to compute predictions based on the testing set. These predictions are used to find the RMSE which was 2.113405. This is also higher than the RMSE of the random forest predictions which suggests that this model is not as good at predicting based on new data. Finally the predictions are plotted against the known values. This plot looks very similar to that of the random forest model but there is less variability in. This is likely since kNN is good at predicting values close to other values, meaning that it will predict values close to known points but will struggle to predict points farther away.

The final model used is a single layer neural network. The hyperparameters for this model are hidden_units, penalty, and epochs. Hidden units is the number of hidden layers, penalty is the amount of weight decay between layers, and epochs is the number of training iterations. Like the other models, a tuning workflow and model are created. A grid and five folds are also created. The tuning parameters are swept over the grid and the best model is chosen.

```
set.seed(2003)

# Create recipe
mlp_rec = recipe(value ~ ., data = train)

# Create model with tuning parameters
tune_mlp = mlp(hidden_units = tune(), penalty = tune(), epochs = tune()) %>% set_mode("r
egression") %>% set_engine("nnet")

# Create workfow
mlp_tune_wf = workflow() %>% add_recipe(mlp_rec) %>% add_model(tune_mlp)

# Create 5 folds
mlp_folds = vfold_cv(train, v = 5)

# Create grid of tuning parameters
mlp_grid = grid_regular(hidden_units(), penalty(), epochs(), levels = 5)

# Compute all models with grid parameters
mlp_tune_res = mlp_tune_wf %>% tune_grid(resamples = mlp_folds, grid = mlp_grid)

# Show relevant statistics for models
mlp_tune_res %>% collect_metrics() |> filter(.metric == "rmse") |> arrange(mean)
```

```
## # A tibble: 125 × 9
##    hidden_units penalty epochs .metric .estimator  mean     n std_err .config
##           <int>   <dbl>  <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
##  1            3       1    505 rmse    standard    2.10     5  0.0578 Preproces…
##  2            5       1    257 rmse    standard    2.11     5  0.0417 Preproces…
##  3            5       1   1000 rmse    standard    2.11     5  0.0640 Preproces…
##  4           10       1    752 rmse    standard    2.12     5  0.0624 Preproces…
##  5            7       1    752 rmse    standard    2.12     5  0.0591 Preproces…
##  6           10       1    257 rmse    standard    2.12     5  0.0617 Preproces…
##  7            7       1    505 rmse    standard    2.12     5  0.0509 Preproces…
##  8            7       1   1000 rmse    standard    2.13     5  0.0576 Preproces…
##  9           10       1   1000 rmse    standard    2.14     5  0.0501 Preproces…
## 10            5       1    752 rmse    standard    2.14     5  0.0507 Preproces…
## # ℹ 115 more rows
```

The best model had 3 hidden units, 1 penalty, and 505 epochs, with an RMSE of 2.103155. Finally this model can be used to finalize the workflow and create a trained model. We then perform 10 fold cross validation on the model.

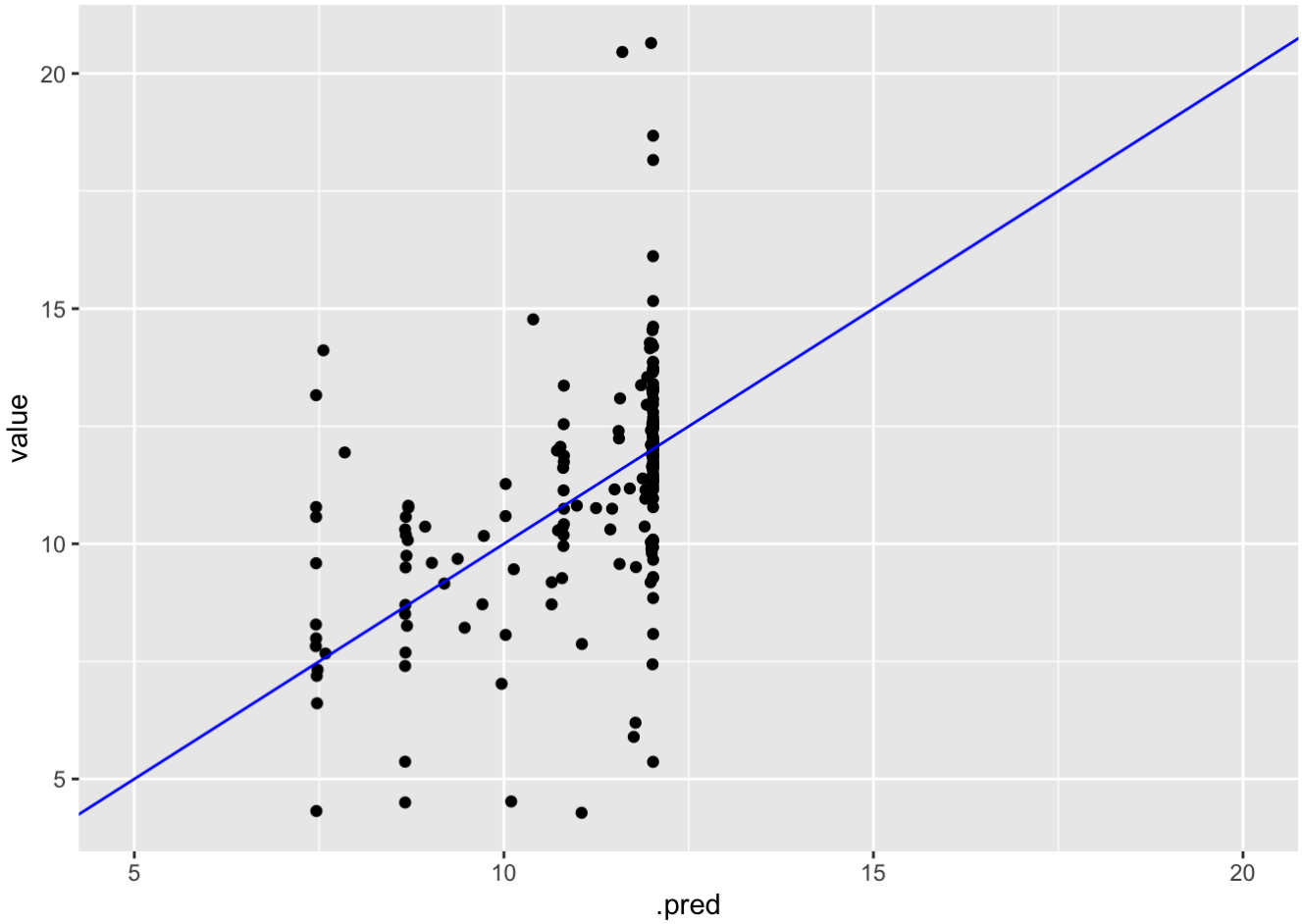
```
# Select best model and finalize model
best_mlp = mlp_tune_res %>% select_best("rmse")
final_wf_mlp = mlp_tune_wf %>% finalize_workflow(best_mlp)
mlp_fit = final_wf_mlp %>% fit(data = train)

# Perform 10 fold cross validation on model
mlp_folds = vfold_cv(dat2, v = 10)
mlp_validation = fit_resamples(final_wf_mlp, resamples = mlp_folds)
mlp_validation %>% collect_metrics()
```

```
## # A tibble: 2 × 6
##   .metric .estimator  mean     n std_err .config
##   <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1 rmse    standard   2.09     10  0.108  Preprocessor1_Model1
## 2 rsq     standard   0.347    10  0.0265 Preprocessor1_Model1
```

```
# Compute predictions and RMSE from final model
mlp_pred = cbind(values, predict(mlp_fit, new_data = test))
sqrt(mean((mlp_pred$value - mlp_pred$.pred)^2))
```

```
## [1] 2.283779
```

```
# Plot predictions against values
mlp_pred |> ggplot(aes(x = .pred, y = value)) + geom_point() +
  geom_abline(intercept = 0, slope = 1, color = "blue") + xlim(5,20)
```



The mean RMSE was 2.0944299, which is good and shows that the model is stable. It is higher than both random forest and kNN which suggests that this model is not as strong as the latter. Finally we can compute predictions and an RMSE using them. This RMSE was found to be 2.283779 which confirms that this model's prediction capabilites are not as strong as random forest or kNN.

Now we can pick the final model. Since we are using RMSE as our metric to evaluate models and the goal is to minimize RMSE, random forest will be the final model. It had the lowest 10 fold cross validation RMSE and prediction RMSE which show that the model is robust and has strong prediction capabilities. We can now construct the final model using the entire data set.

```
set.seed(2023)

# Prepare data set for model
datp = dat2 |> select(-c(id))

# Create final formula, model, and workflow
ff = datp |> formula(value~.)
model = rand_forest(mode = "regression", engine = "ranger", mtry = 15, trees = 500, min_
n = 2)
wf = workflow() |> add_formula(ff) |> add_model(model)

# Create final fit using final workflow
final_fit = fit(wf, datp)
```

```
## Warning: 15 columns were requested but there were 9 predictors in the data. 9
## will be used.
```

Discussion

Now that we have built and trained the final model, we can test it on the entire data set. We use the model to compute predictions for the entire data set and add a new column that is the square error of the prediction. We can use this column to determine locations where the model struggles to predict and locations where the model predicts well.

```
# Use final model to predict based on data and compute RMSE of final model
datp_pred = dat |> cbind(predict(final_fit, new_data = datp))
sqrt(mean((datp_pred$value - datp_pred$.pred)^2))
```

```
## [1] 0.6972941
```

```
# Display the top 10 locations with the greatest square error
datp_pred |> mutate(sqerror = (value-.pred)^2) |> arrange(desc(sqerror)) |> select(stat
e, county, city, sqerror, value, .pred) |> head(10)
```

```
##         state    county          city   sqerror     value    .pred
## 1  California     Butte Not in a city 19.523574 20.457576 16.03903
## 2     Arizona     Pinal      Maricopa 13.178376 19.457895 15.82769
## 3       Texas   El Paso       El Paso 12.915179 16.151786 12.55802
## 4  California San Diego     Escondido 11.618405 17.461806 14.05323
## 5  California    Fresno        Clovis 11.030108 22.259123 18.93796
## 6  California      Kern   Bakersfield 10.314776 23.160784 19.94912
## 7       Texas   El Paso       El Paso  9.104330  9.540678 12.55802
## 8  California    Tulare       Visalia  8.870834 20.648092 17.66970
## 9  California      Kern   Bakersfield  8.136376 21.201299 18.34887
## 10   New York    Monroe     Rochester  6.989500  9.038261 11.68203
```

```
# Display 10 locations with lowest square error
datp_pred |> mutate(sqerror = (value-.pred)^2) |> arrange(sqerror) |> select(state, coun
ty, city, sqerror, value, .pred) |> head(10)
```

```
##                   state               county                      city
## 1        South Carolina         Chesterfield             Not in a city
## 2              Missouri          Saint Louis                    Ladue
## 3  District Of Columbia District of Columbia                Washington
## 4              Arkansas              Pulaski               Little Rock
## 5                 Texas               Travis                    Austin
## 6         Massachusetts              Suffolk                    Boston
## 7                  Ohio               Warren                   Lebanon
## 8              Maryland     Baltimore (City)                 Baltimore
## 9               Indiana               Marion Indianapolis (Remainder)
## 10       North Carolina              Robeson             Not in a city
##         sqerror    value    .pred
## 1  8.001082e-08 11.05091 11.05063
## 2  3.172598e-07 12.16204 12.16261
## 3  4.516106e-07 12.43697 12.43765
## 4  1.370443e-06 12.09854 12.09737
## 5  2.413217e-06 10.45172 10.45328
## 6  3.412620e-06 11.10807 11.10991
## 7  5.255941e-06 11.97373 11.97602
## 8  2.152223e-05 12.53158 12.52694
## 9  2.278650e-05 13.05221 13.05699
## 10 2.509250e-05 11.38707 11.38206
```

The top 5 locations with the closest predicted values are in Chesterfield county in South Carolina, Saint Louis county in Missouri, Washington DC, Pulaski county in Arkansas, and Travis county in Texas. of the bottom 5 locations with the farthest predicted values, 3 are in California, specifically in Butte, San Diego, and Fresno counties. The other two locations are Pinal county in Arizona and El Paso county in Texas. The bottom 5 locations are all in the Southwest United States, whereas the top 5 locations are spread through the Midwest, East coast, and South.

The possible reason for the poor performance of the bottom 5 locations by prediction is that the Southwest US has a dry and dusty climate, and dust could possibly show up as as particulate matter in the PM2.5 readings. These bottom 5 locations also seem to have some of the largest PM2.5 values, which further confirms this hypothesis.

The possible reason for the model accurately predicting the top 5 locations is that these locations have a particularly stable climate, so the only change in air pollution is from human factors, such as population and traffic, which is what the model includes.

Currently there is no variable in the model that accounts for pollution directly from natural factors. To increase the performance of the model, it's hypothesized to add a variable that measures the average humidity of the location. This will help account for any pollution from dust, because a low humidity would mean a dusty location, which will increase pollution. This will help reduce the error in prediction from the locations in the Southwest United States.

The numerical models CMAQ and AOD are very good at predicting the PM2.5 concentrations. These variables had the strongest correlation with PM2.5 as seen in the visualizations. The performance of the model without PM2.5 and CMAQ is given below:

```
set.seed(2023)

# Remove CMAQ, and aod from data
datc = dat2 |> select(-c(id, CMAQ, aod))

# Build new model without CMAQ and aod
ffc = datc |> formula(value~.)
wfc = workflow() |> add_formula(ffc) |> add_model(model)
final_fitc = fit(wfc, datc)
```

```
## Warning: 15 columns were requested but there were 7 predictors in the data. 7
## will be used.
```

```
# Compute predictions and RMSE with this model
datc_pred = dat |> cbind(predict(final_fitc, new_data = datc))
sqrt(mean((datc_pred$value - datc_pred$.pred)^2))
```

```
## [1] 0.7484225
```

The RMSE of the random forest model is 0.748 without including the CMAQ and AOD predictor variables. This is much higher than the RMSE of 0.697 including the variables. Therefore, the numerical models CMAQ and AOD will be a much more cost-effective solution to monitoring air pollution because they are relatively accurate.

When it comes to applying this model to Alaska and Hawaii, we predict that the model won't perform as well as compared to the original data set purely because the model wasn't built with Alaska or Hawaii data. However, the model will still be a good predictor because the predictor variables that are used in the model are common to all locations, such as population density and the primary and secondary road count. We don't expect any bias or other variables that aren't accounted for that are different in Alaska and Hawaii that would drastically decrease the performance of the model. In addition, these states don't have a dry and dusty climate, which we predicted lowered the accuracy of the prediction of the locations in the Southwest US.

The most challenging part of the project was tuning the hyper parameters for the models. Trying to make a grid for each hyper parameter configuration and then testing each model for the RMSE 5 times took a long time to compute and had a lot of syntax. Another challenging aspect of the project was finding a fourth prediction model to use, and understand how it worked in order to apply it. The most important thing that we learned was the workflow that data scientists go through for building and testing a model. First, the model is built on a training data set, then the hyper parameters are tuned to the lowest RMSE configuration, and finally the tuned model is applied to predict the test data set. We learned about the train, test, split model for machine learning and how to apply k-fold cross validation to model tuning and validation. The final model performed much better than we expected, as the RMSE was 0.697, and we predicted it to be 2. This is probably because we tuned all three parameters to make the optimal prediction.

Contributions:
Vardhan - Discussion, Introduction Solomon - Wrangling, Modeling