Predicting Particulate Matter Concentrations for Different Regions in California During Wildfires

Christopher Wagner

Introduction

In this project, I aim to predict air quality using weather data from different locations in California in 2020. This project looks at particulate matter with a diameter of fewer than 2.5 microns (PM2.5) measured in micrograms per cubic meter of air ($\mu g/m^3$) which is mainly caused by smoke from fires. 2020 was the worst year for wildfires in all of California's history, thus there is a large amount of data available. Many meteorological factors can lead to different degrees of air quality, for example wind speed and wind direction. With this intuition, I will use data from different regions during 2020 in order to predict the PM2.5 level at a given location.

Figure 1: Orange hue in the San Francisco sky on Orange Skies Day (September 9th, 2020) as a result of smoke from more than 20 wildfires.

In 2020, California experienced its worst wildfire season in recorded history, with more than 4.2 million acres burned across the state (California Department of Forestry and Fire Protection, 2021). The fires released enormous amounts of smoke into the atmosphere which caused record levels of PM2.5 in many regions. Some cities, including San Francisco, saw air quality index (AQI) values reach hazardous levels, turning skies orange and prompting widespread health warnings (Martichoux and Jue, 2020). Figure 1 shows an example of the orange sky due in San Francisco in 2020. This unprecedented fire season highlighted the need for developing

air quality forecasting tools, particularly for PM2.5 in order to protect the public from the harmful effects of wildfire smoke.

Figure 2: Large amounts of smoke from the wildfires in California in 2020.

Predicting PM2.5 levels is an important task because air quality has a direct impact on public health. PM2.5 particles are small enough to enter the lungs and bloodstream, which poses serious risks such as respiratory and cardiovascular issues (World Health Organization, 2021). Due to California's frequent wildfires, over 39 million people are at risk of the dangerously high levels of air pollution, especially during fire season. Using machine learning to predict PM2.5 has several benefits. First, these models can process large amounts of complex data to create accurate, fast, and dynamic forecasting compared to traditional models. Second, machine learning can adapt to new observations being collected. Overall, machine learning has a huge potential to speed up air quality forecasting and make PM2.5 measurements more reliable, especially as environmental conditions change due to climate variability.

Cleaning the Raw Data

Load Packages

```
library(tidymodels)
library(tidyverse)
library(ISLR)
library(ISLR2)
library(tidyverse)
library(glmnet)
library(modeldata)
```

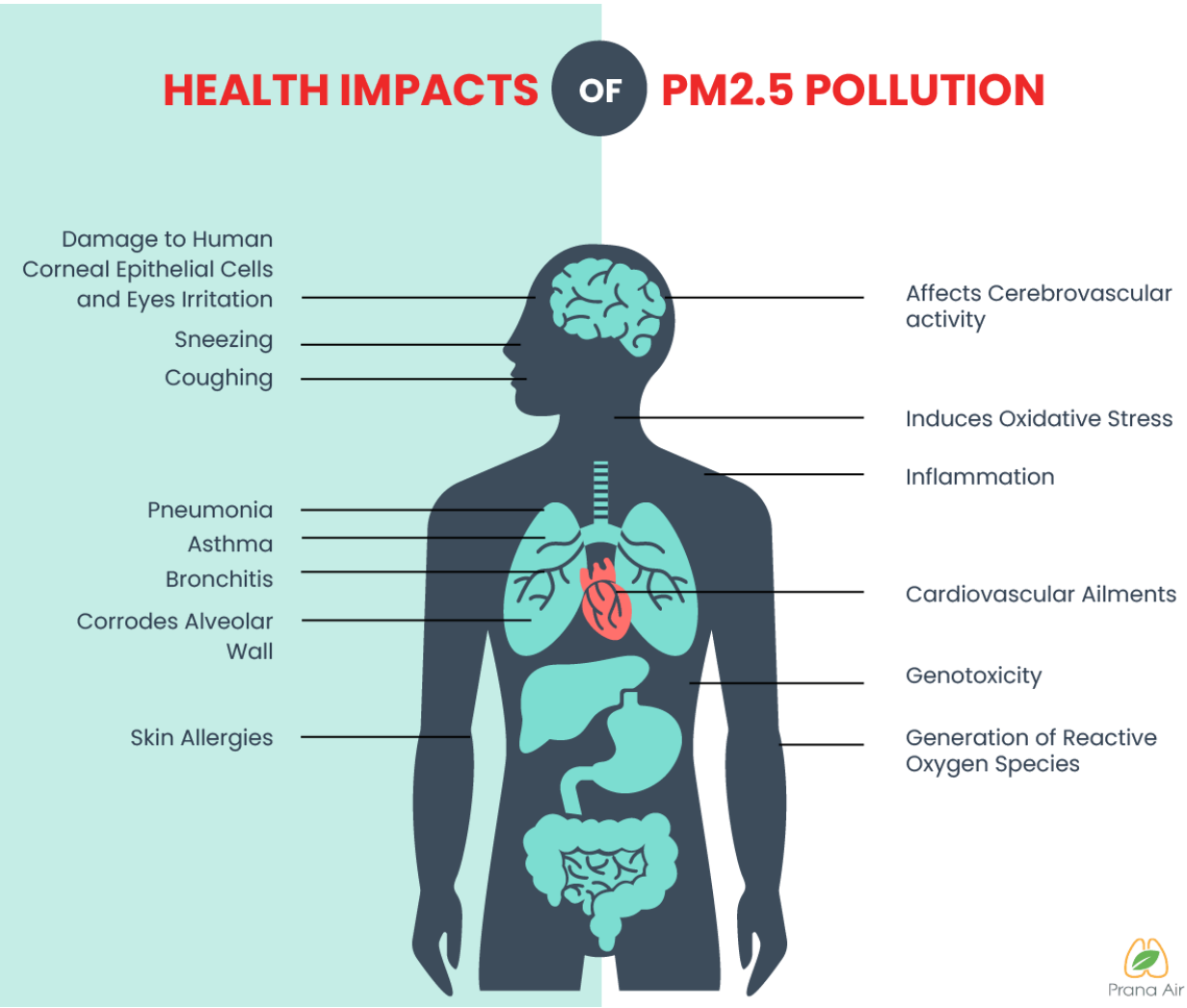


Figure 3: The effects of PM2.5 on human health

```
library(ggthemes)
library(janitor)
library(xgboost)
library(ranger)
library(vip)
library(corrplot)
library(kableExtra)
library(rsample)
library(modeldata)
library(janitor)
library(naniar) # to assess missing data patterns
library(themis) # for upsampling
library(maps)
library(knitr)
library(parsnip)
tidymodels_prefer()
```

First we need to load all the necessary packages and import the raw data.

Import Data

The dataset includes PM2.5 ($\mu g/m^3$) levels, meteorological data (average temperature, maximum temperature, minimum temperature, humidity, wind speed, wind direction) and latitude and longitude coordinates from different locations in California. I collected the data by scraping the website OpenAQ (https://openaq.org/) and aggregating it alongside weather data from meteostat (https://meteostat.net/en/) using a Python script (https://colab.research.google.com/drive/1QTTk-XguZlQkmPgedGjOIkATuayJKOYX?usp=sharing). Let's import the data and take a look at the first few observations:

```
pm25_data <- read.csv("PM25_data.csv")
head(pm25_data)
```

```
##     date   SiteID MeanPM25Concentration      SITE      STATE COUNTY_CODE
## 1 1/1/20 60010007                   8.6 Livermore California           1
## 2 1/2/20 60010007                   4.5 Livermore California           1
## 3 1/3/20 60010007                  14.2 Livermore California           1
## 4 1/4/20 60010007                  10.9 Livermore California           1
## 5 1/5/20 60010007                   7.8 Livermore California           1
## 6 1/6/20 60010007                   6.2 Livermore California           1
##    COUNTY SITE_LATITUDE SITE_LONGITUDE tavg tmin tmax prcp snow wdir wspd wpgt
## 1 Alameda      37.68753      -121.7842  9.5  4.4 16.1    0    0  307  6.5   NA
## 2 Alameda      37.68753      -121.7842  8.9  5.6 16.1    0    0   17  2.0   NA
## 3 Alameda      37.68753      -121.7842  8.0  2.8 15.6    0    0   18  3.7   NA
## 4 Alameda      37.68753      -121.7842  9.1  3.3 15.0    0    0  350  6.5   NA
## 5 Alameda      37.68753      -121.7842  7.4  1.1 15.6    0    0   10  6.8   NA
## 6 Alameda      37.68753      -121.7842  7.2  1.7 14.4    0    0   41  7.1   NA
##     pres tsun
## 1 1020.0   NA
## 2 1021.4   NA
## 3 1022.8   NA
## 4 1028.1   NA
## 5 1033.3   NA
## 6 1031.2   NA
```

We can also view the dimensions of the dataset and see that it has 30515 rows and 19 columns. Additionally, we will print out the names of the columns and rename them to make it easier to work with.

```r
dim(pm25_data)
```

```
## [1] 30515    19
```

```r
names(pm25_data)
```

```
##  [1] "date"                  "SiteID"                "MeanPM25Concentration"
##  [4] "SITE"                  "STATE"                 "COUNTY_CODE"
##  [7] "COUNTY"                "SITE_LATITUDE"         "SITE_LONGITUDE"
## [10] "tavg"                  "tmin"                  "tmax"
## [13] "prcp"                  "snow"                  "wdir"
## [16] "wspd"                  "wpgt"                  "pres"
## [19] "tsun"
```

```r
# rename some columns to make them easier to work with
pm25_data <- pm25_data %>%
  rename(
    site_id = SiteID,
    pm25 = MeanPM25Concentration,
    site_name = SITE,
    latitude = SITE_LATITUDE,
    longitude = SITE_LONGITUDE,
    state = STATE,
    county_id = COUNTY_CODE,
    county_name = COUNTY
  )
```

The

Variable	Description
pm25	Daily PM2.5 measurement
date	Date of the PM2.5 measurement
site_id	ID number of the site
site_name	Name of the site
state	State of the site (all CA)
county_id	ID of the county of the site
county_name	Name of the county of the site
latitude	Latitude of the site location
longitude	Longitude of the site location
tavg	Average daily temperature (°C)
tmin	Minimum daily temperature (°C)
tmax	Maximum daily temperature (°C)
prcp	Daily precipitation (m/s)
wspd	Daily wind speed
wdir	Direction of wind (°)
pres	Daily atmospheric pressure (hPa)

Missing Data

We will take a look at a summary of our dataset to obtain the number of missing values per column, and print out the names of columns that contain any missing values.

```r
pm25_data %>%
  summary()
```

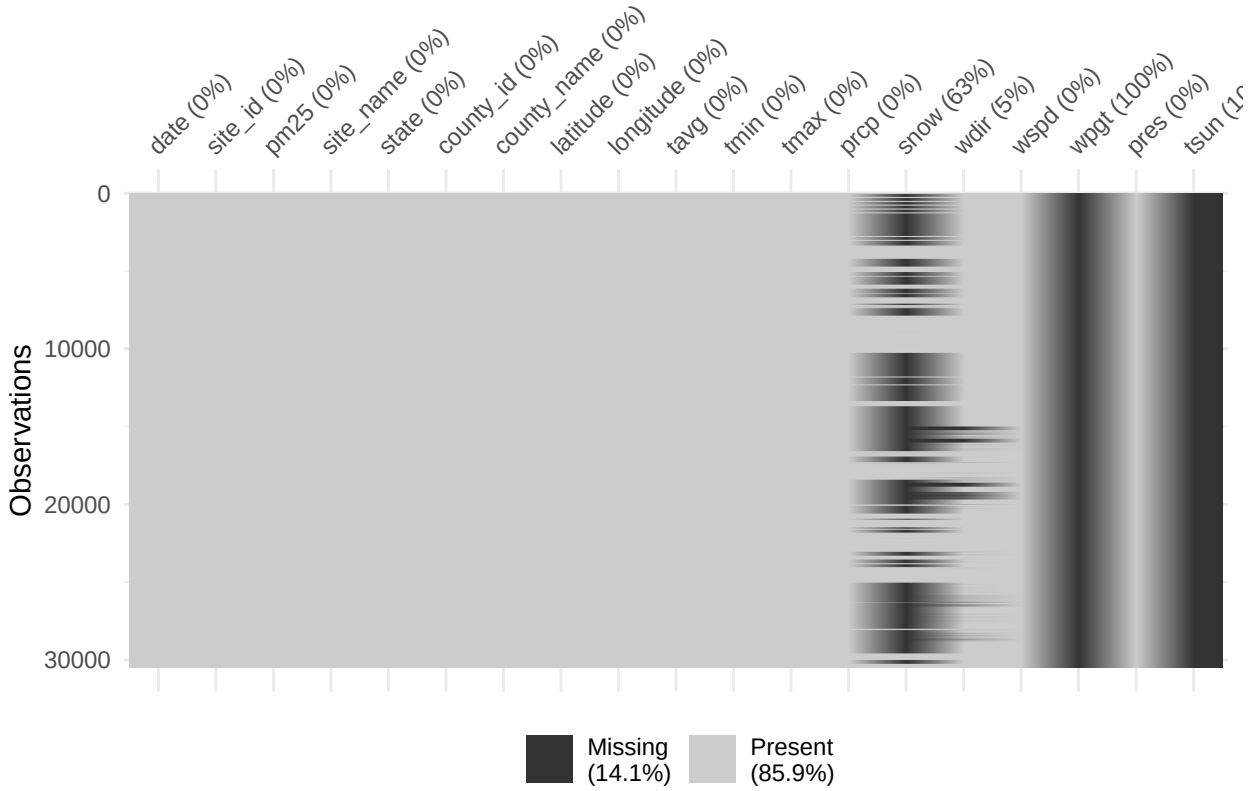
```
##      date              site_id             pm25          site_name
##  Length:30515       Min.   :60010007   Min.   : -3.000   Length:30515
##  Class :character   1st Qu.:60370016   1st Qu.:  4.100   Class :character
##  Mode  :character   Median :60650009   Median :  6.700   Mode  :character
##                     Mean   :60575369   Mean   :  9.356
##                     3rd Qu.:60792004   3rd Qu.: 10.800
##                     Max.   :61131003   Max.   :378.500
##
##     state              county_id      county_name           latitude
##  Length:30515       Min.   :  1.00   Length:30515       Min.   :32.58
##  Class :character   1st Qu.: 37.00   Class :character   1st Qu.:34.24
##  Mode  :character   Median : 65.00   Mode  :character   Median :36.71
##                     Mean   : 57.41                      Mean   :36.35
##                     3rd Qu.: 79.00                      3rd Qu.:37.95
##                     Max.   :113.00                      Max.   :41.76
##
##    longitude           tavg             tmin             tmax
##  Min.   :-124.2   Min.   :-9.10   Min.   :-33.70   Min.   :-21.60
##  1st Qu.:-121.6   1st Qu.:11.70   1st Qu.:  6.00   1st Qu.: 17.20
##  Median :-120.1   Median :15.60   Median : 10.00   Median : 22.00
##  Mean   :-119.9   Mean   :16.48   Mean   : 10.17   Mean   : 23.06
##  3rd Qu.:-118.2   3rd Qu.:20.50   3rd Qu.: 14.40   3rd Qu.: 28.30
##  Max.   :-115.5   Max.   :41.80   Max.   : 35.60   Max.   : 53.30
##
##       prcp              snow            wdir           wspd
##  Min.   : 0.000   Min.   :0       Min.   :  0    Min.   : 0.000
##  1st Qu.: 0.000   1st Qu.:0       1st Qu.:164    1st Qu.: 6.233
##  Median : 0.000   Median :0       Median :298    Median : 8.833
##  Mean   : 1.199   Mean   :0       Mean   :243    Mean   : 9.747
##  3rd Qu.: 0.000   3rd Qu.:0       3rd Qu.:331    3rd Qu.:12.200
##  Max.   :82.300   Max.   :0       Max.   :360    Max.   :51.000
##                   NA's   :19197   NA's   :1622
##    wpgt              pres          tsun
##  Mode:logical   Min.   : 996.9   Mode:logical
##  NA's:30515     1st Qu.:1012.8   NA's:30515
##                 Median :1015.7
##                 Mean   :1016.0
##                 3rd Qu.:1018.9
##                 Max.   :1035.4
##
```

```
missing_columns <- pm25_data %>%
  summarise_all(~ any(is.na(.))) %>%
  pivot_longer(everything(), names_to = "column", values_to = "has_missing") %>%
  filter(has_missing) %>%
  pull(column)

missing_columns
```

```
## [1] "snow" "wdir" "wpgt" "tsun"
```

We can also generate an easier-to-read visualization of the missing data:

```
vis_miss(pm25_data)
```

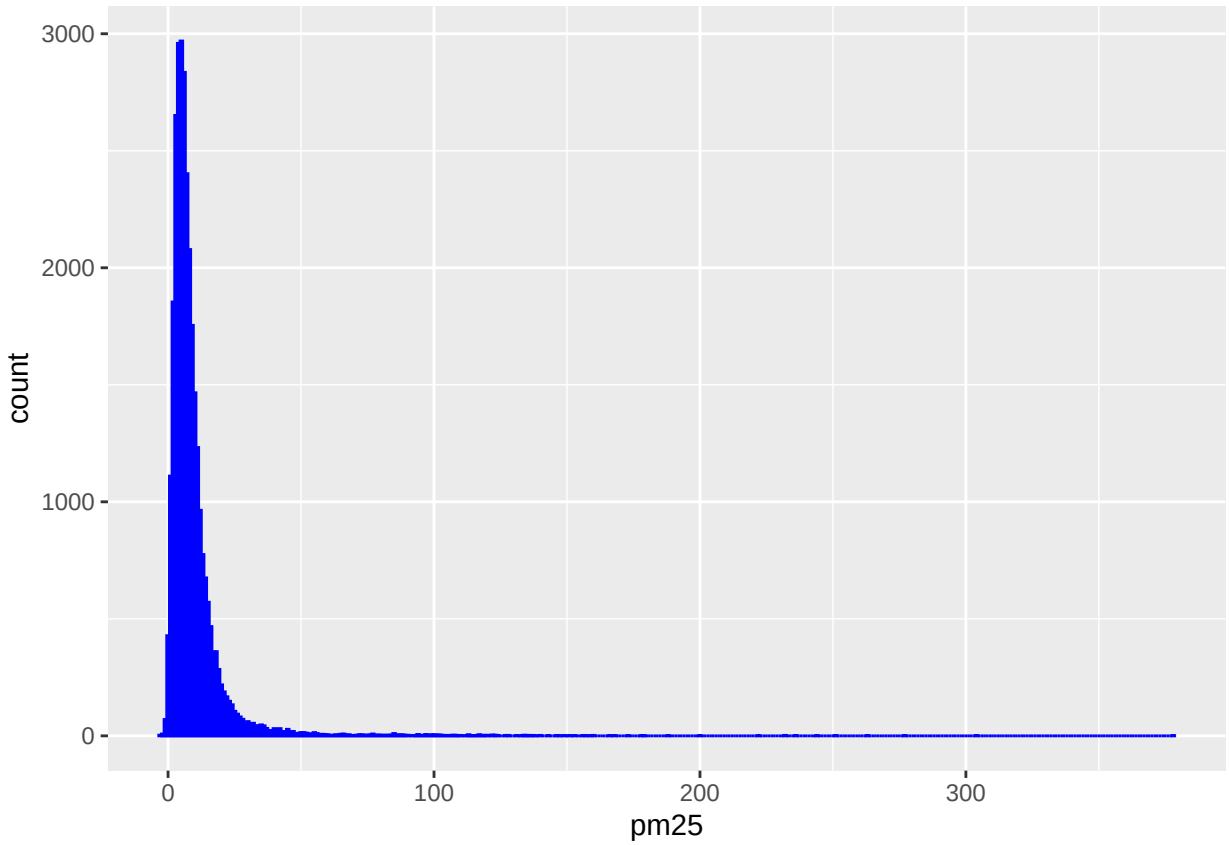


From the output, several columns contain missing data: CBSA_CODE (core based statistical area code), snow (maximum snow depth), wdir (wind direction in degrees), wpgt (peak wind gust), tsun (minutes of sunlight). Most of these are not relevant for predicting PM2.5 levels or are already encoded by some other variable. For example, it does not snow at most of these site locations, so the variable 'snow' likely won't affect PM2.5 levels. Thus we will choose to drop the CBSA_CODE, snow, and tsun columns as well as other columns that aren't relevant. This will be handled in the recipe creation. The variables wpgt and wdir may affect PM2.5 levels, since wind carries smoke from wildfires to different areas. Unfortunately, wpgt has no values, so we will drop the column. For wdir, only 5% of the values are missing, so we will want to impute them. This might be difficult given that degrees is a circular metric (ie, linear models will not work to impute the data), so we will handle this using bagged trees in the recipe creation.

Exploratory Data Analysis

Now we are ready to perform EDA in order to obtain a better understanding of our data and the relationships between variables. First, let's visualize the distribution of our outcome variable, which is the air quality PM2.5 level.

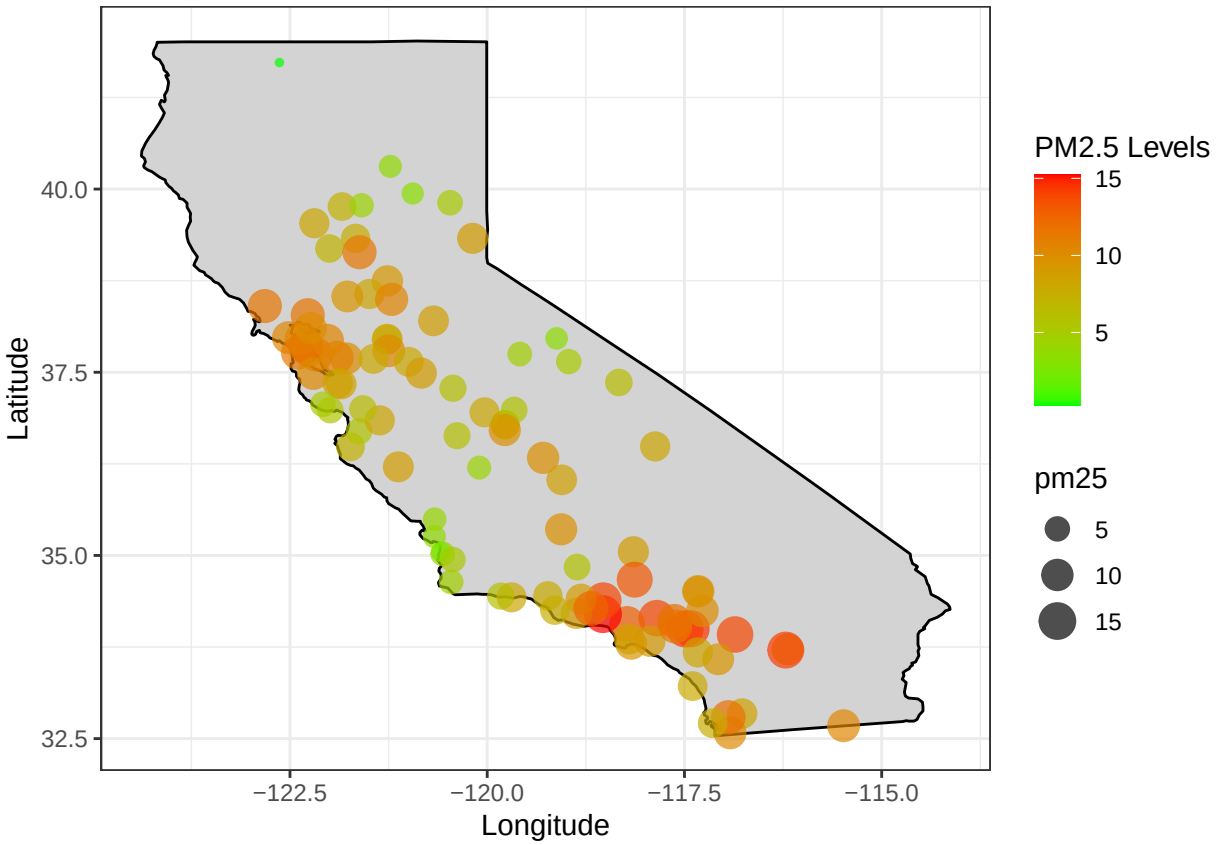
```
pm25_data %>% ggplot(aes(x = pm25)) +
  geom_histogram(binwidth = 1, color = "blue")
```



The majority of the outcome variable are small values around 0.

Next, we can visualize the spatial distribution of PM2.5 on a map of California. This will allow us to see if there are any patterns in the concentration of high PM2.5. There are many time points, so we will choose the date 7/1/2020 to visualize.

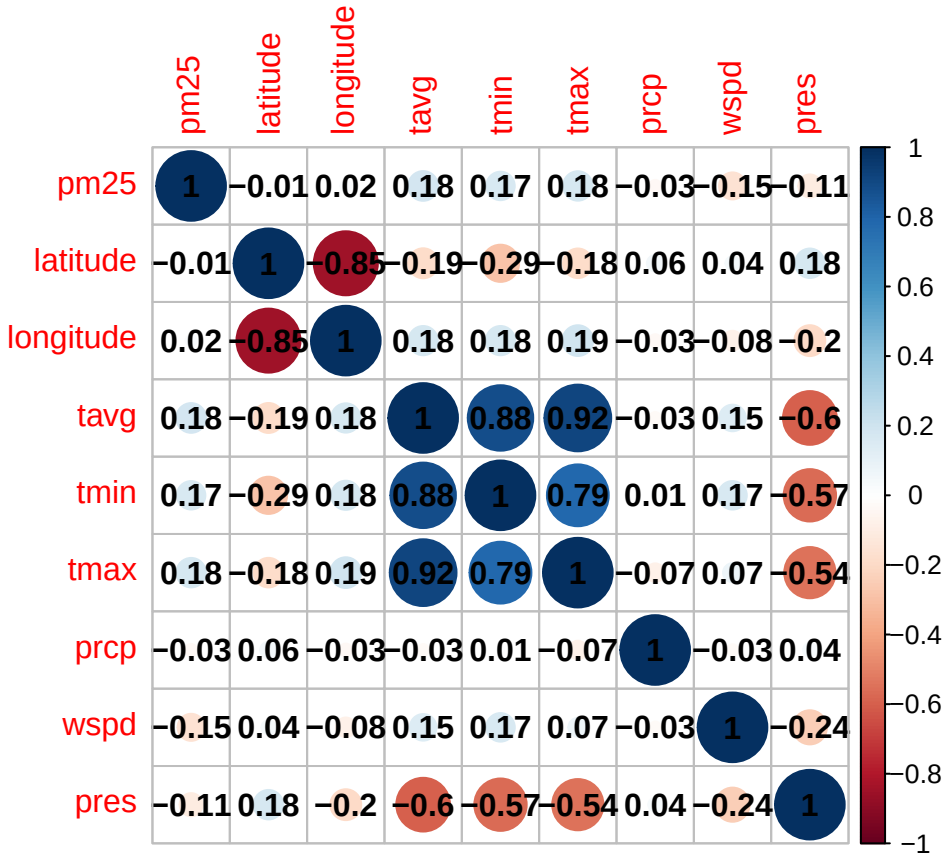
```
pm25_july2020 <- pm25_data %>%
  filter(date == "7/1/20")

california_map <- map_data("state") %>%
  filter(region == "california")

ggplot() +
  geom_polygon(data = california_map, aes(x = long, y = lat, group = group), fill = "lightgray", color =
  geom_point(data = pm25_july2020, aes(x = longitude, y = latitude, color = pm25, size = pm25), alpha =
  scale_color_gradient(low = "green", high = "red", name = "PM2.5 Levels") +
  labs(x = "Longitude", y = "Latitude") + theme_bw()
```



This provides us some insight into the spatial correlations exhibited by the data, which might indicate the usefulness of the location variables for predicting PM2.5. For example, we can see that the highest concentrations were in the coast of Southern California.

Most of the predictors in the dataset are continuous, so it would be useful to visualize the correlations between these variables.

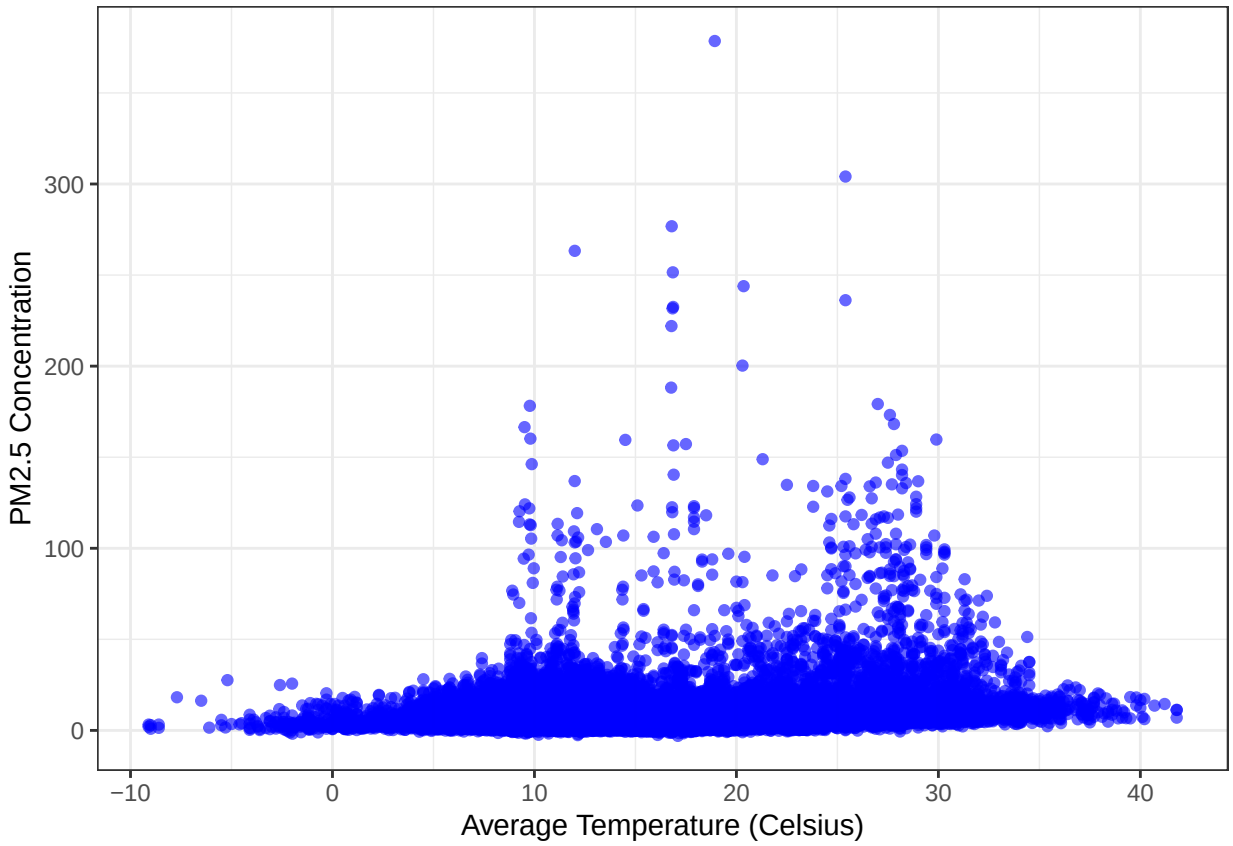
```
numeric_data <- pm25_data %>%
  select_if(is.numeric) %>%
  select(-site_id, -county_id, -wdir, -snow)

pm25_cor <- cor(numeric_data)

pm25_corrplot <- corrplot(pm25_cor, method = "circle", addCoef.col = 1)
```



From this, we can see that most predictors are weakly correlated or completely uncorrelated with our outcome variable. The predictors tavg, tmin, tmax are all strongly correlated with each other, which makes sense because they are all derived from measures of temperature at a given location. Additionally, tavg, tmin, and tmax are each negatively correlated with pressure. This may affect some models that have strong assumptions about the correlation between variables, ie linear regression.

Next, we can generate scatterplots between our outcome variable and the predictors, starting with the average temperature.

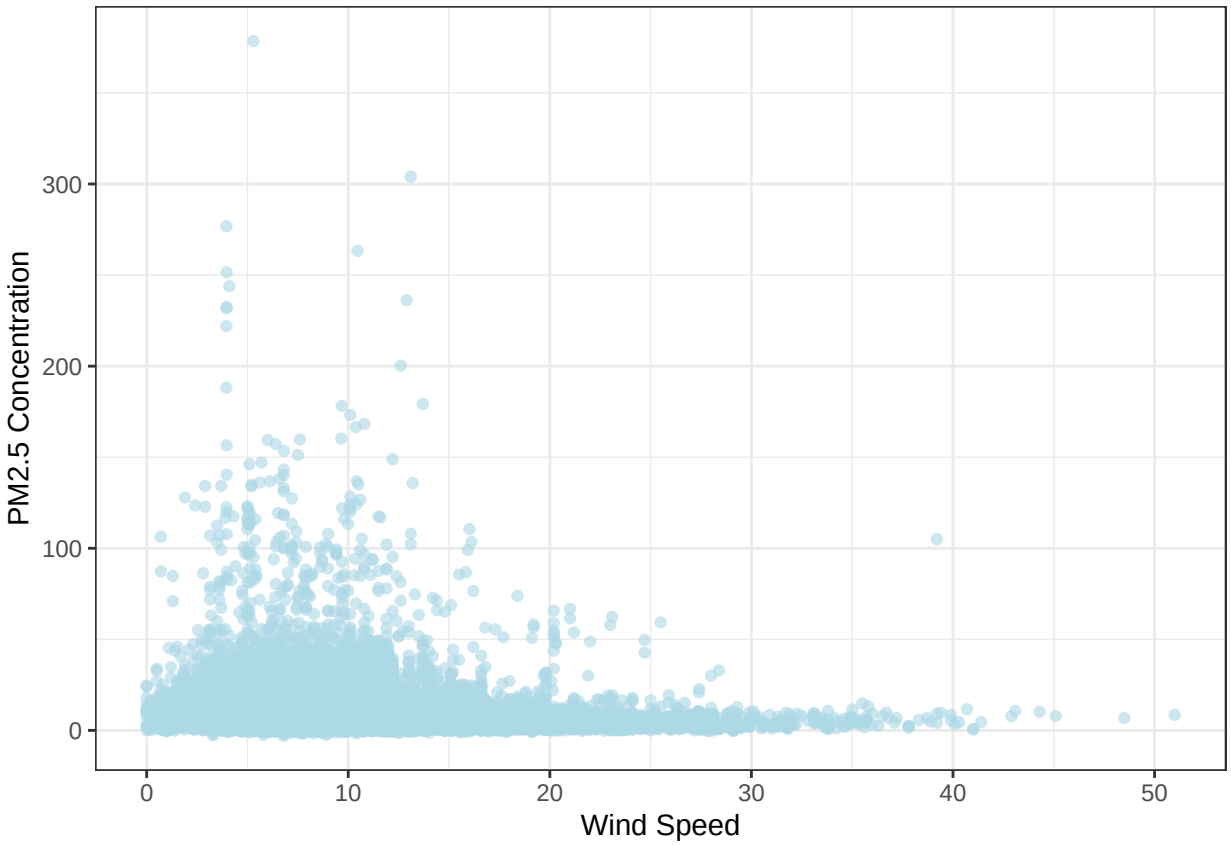
```
ggplot(pm25_data, aes(x = tavg, y = pm25)) +
  geom_point(alpha = 0.6, color = "blue") +
  labs(x = "Average Temperature (Celsius)",
       y = "PM2.5 Concentration") +
  theme_bw()
```



From this we can see that while most of the values are 0, there seems to be a slight positive linear relationship between temperature and the PM2.5 concentration.

Similarly, for wind speed:

```
ggplot(pm25_data, aes(x = wspd, y = pm25)) +
  geom_point(alpha = 0.6, color = "lightblue") +
  labs(x = "Wind Speed",
       y = "PM2.5 Concentration") +
  theme_bw()
```



We can see that there is a slightly negative linear relationship between wind speed and PM2.5.

Data Preparation

Now we can select our predictors and start preparing our models. We will first need to split our data into training and testing sets in order to create our model's recipe. From there we can create folds through resampling in order to cross validate our model.

In order to create an accurate recipe, we must split our data into training and testing sets. For this data set we will split it into 70 percent training and 30 percent testing. This is done in order to prevent over fitting, which can occur when training a model on a full data set. Since our data set is large enough this allows us to get an accurate representation of training and testing data. We will stratify our data on the outcome variable to ensure we have an equal distribution of the variable in our data sets.

Select Predictors

From the plots in the EDA section, we can see that all the variables may be useful for prediction. Hence, we will select all the predictors (latitude, longitude, tavg, tmin, tmax, prcp, wspd, pres) to use in our recipe. The predictors are described in further detail in the table in the *Import Data* section.

Split Data

In order to train and evaluate the models, we need to split the data into training and testing sets. For this, we will use a 70/30 split, meaning 70% of the data will be used for training and 30% will be used for testing. Since we have a lot of data (over 30,000 observations), the split size is sufficient both for allowing a model to see enough samples to learn an optimal function and for obtaining a good representation of how a model will perform on unseen data.

However, stratifying on the outcome variable might not be the best approach for our dataset since it has spatial dependencies. Splitting the data this way will not allow us to obtain a good representation of the model's ability to generalize to unseen locations, since it will likely include training samples from each location. Thus, we will split the data by location (site ID).

```
set.seed(111) # for reproducibility

pm25_split <- initial_split(pm25_data, prop=0.7, strata = "site_id")
pm25_train <- training(pm25_split)
pm25_test <- testing(pm25_split)

# verify
nrow(pm25_train)/nrow(pm25_data)
```

```
## [1] 0.6999181
```

```
nrow(pm25_test)/nrow(pm25_data)
```

```
## [1] 0.3000819
```

Recipe Creation

The next step is to create a recipe of the preprocessing steps to ensurethat the transformations are consistently applied during both training and testing. As previously mentioned, we will be dropping the irrelevant columns (BSA_CODE, snow, X, POC, PERCENT_COMPLETE, STATE, CBSA_NAME, DAILY_OBS_COUNT, UNITS) along with the columns representing variables which are completely missing (wpgt, tsun). This leaves us with only one column with missing values (wdir), which is handled by imputation via bagged trees. The reason we use bagged trees is because wind direction measured in degrees is a circular metric, thus linear methods do not apply. For example, 350° and 10° are close in direction despite being far away on a number line. Lastly, we will center and scale all our predictors in order to normalize them.

Additionally, we will log transform our outcome variable to avoid numerical errors and reduce skewness in order to improve the training process. A couple of the pm25 values are negative due to measurement error, and since pm25 cannot be negative we set these equal to 0 before transforming the column.

```
pm25_recipe <- recipe(pm25 ~ county_id + latitude + longitude
                      + tavg + tmin + tmax + prcp + wdir + wspd +
                        pres + date, data=pm25_data) %>%
  step_mutate(date = as.Date(date, format = "%m/%d/%Y")) %>%
  step_impute_bag(wdir) %>%
  step_date(date, features = "doy") %>%
  step_rm(date) %>%
  step_center(all_predictors()) %>%
  step_scale(all_predictors())

# prep(pm25_recipe) %>% bake(pm25_train)
```

Additionally, the variable 'date' must be considered carefully. Since there are over 250 days of PM2.5 measurements, there are 250 separate dates, which means each of these would be encoded into separate dummy variables. This might introduce too much complexity in the model (maybe too much for my computer to handle) and might lead to overfitting due to the large number of parameters that need to be estimated. In order to get around this, we can convert the Date column into the day of the year, which can preserve the temporal information without creating too many predictors. Luckily, we only have data collected from 1/1/2020 to 9/9/2020, so we don't need to worry about the cyclical nature of this.

K-Fold Cross Validation

Next, we will create the folds to perform K-Fold cross validation. This is when we split the training dataset into K equal sized subsets and train the model on K-1 folds while leaving the remaining fold for validation. This is repeated K times with each fold used once as the validation set, which allows us to validate our models without losing any training data. For this, we will choose 10 folds and stratify on our outcome variable.

Since there is a timestamp associated with each observation, this may raise a concern about using K-Fold cross validation since it might mix up the time order. However, the nature of the problem is not a pure time series forecasting one, where future values are predicted based on past values in a strict sequential order.

Additionally, the folds are usually created by stratifying on the outcome variable, but this will not work for our dataset since it has spatial dependencies. As mentioned before, splitting this way will not allow us to obtain a good representation of the model's ability to generalize to unseen locations, since it will likely be trained from each location. Thus, since are predicting PM2.5 levels based on a combination of geospatial and environmental variables, using the location to group the folds for K-Fold cross validation makes the most sense.

```
pm25_folds <- group_vfold_cv(pm25_train, v = 10, group = "site_id")
```

Model Training

Now we are ready to begin building the models. The first step is to instantiate the models and specify which mode they are in (regression) and which parameters we will want to tune. In this project, we'll fit 8 models.

Fit Models

```
# model 1: Linear Regression
linear_model = linear_reg() %>%
  set_engine("lm")

# model 2: K-Nearest Neighbors
knn_model = nearest_neighbor(neighbors = tune()) %>%
  set_mode("regression") %>%
  set_engine("kknn")

# model 3: Elastic Net
elastic_model = linear_reg(penalty = tune(),
                           mixture = tune()) %>%
  set_mode("regression") %>%
  set_engine("glmnet")

# model 4: Random Forest
rf_model= rand_forest(mtry = tune(),
                      trees = tune(),
                      min_n = tune()) %>%
  set_mode("regression") %>%
  set_engine("ranger", importance = "impurity")

# model 5: Gradient Boosted Trees (XGBoost)
xgb_model = boost_tree(mtry = tune(),
                       trees = tune(),
                       learn_rate = tune()) %>%
  set_engine("xgboost") %>%
  set_mode("regression")
```

```r
# model 6: Neural Network
nn_model = mlp(hidden_units = tune(),
               penalty = tune()) %>%
  set_mode("regression") %>%
  set_engine("nnet")
```

Next, we create the workflows with the recipe:

```r
# Linear Regression
lm_workflow <- workflow() %>%
  add_model(linear_model) %>%
  add_recipe(pm25_recipe)

# K-Nearest Neighbors
knn_workflow <- workflow() %>%
  add_model(knn_model) %>%
  add_recipe(pm25_recipe)

# Elastic Net
elastic_workflow <- workflow() %>%
  add_model(elastic_model) %>%
  add_recipe(pm25_recipe)

# Random Forest
rf_workflow <- workflow() %>%
  add_recipe(pm25_recipe) %>%
  add_model(rf_model)

# Gradient Boosted Trees (XGBoost)
xgb_workflow <- workflow() %>%
  add_recipe(pm25_recipe) %>%
  add_model(xgb_model)

# Neural Network
nn_workflow <- workflow() %>%
  add_recipe(pm25_recipe) %>%
  add_model(nn_model)
```

There are many parameters to tune, so we will set up tuning grids for each model (excluding linear regression):

```r
# K-Nearest Neighbors
knn_grid <- grid_regular(
  neighbors(range = c(1, 20)),
  levels = 5
)

# Elastic Net
elastic_grid <- grid_regular(
  penalty(range = c(-5, 5)),
  mixture(range = c(0, 1)),
  levels = 10
)

# Random Forest
rf_grid <- grid_regular(
```

```
  mtry(range = c(1, 11)),  # number of predictors at each split
  trees(range = c(200, 1000)),  # trees
  min_n(range = c(2, 10)),  # min data points in a node
  levels = 5  #  levels for each parameter
)

# Gradient Boosted Trees (XGBoost)
xgb_grid <- grid_regular(
  mtry(range = c(1, 11)),  # number of predictors at each split
  trees(range = c(200, 1000)),  # trees
  learn_rate(range = c(0.01, 0.3)),  # learning rate
  levels = 5
)


# Neural Network
nn_grid <- grid_regular(
  hidden_units(range = c(32, 64)),  # neurons in the hidden layer
  penalty(range = c(0.001, 0.1)),  # regularization penalty
  levels = 10
)
```

Now we are ready to tune the models:

```
# K-Nearest Neighbors
knn_results <- tune_grid(
  knn_workflow,
  resamples = pm25_folds,
  grid = knn_grid
)

# Elastic Net
elastic_results <- tune_grid(
  elastic_workflow,
  resamples = pm25_folds,
  grid = elastic_grid
)

# Random Forest
rf_results <- tune_grid(
  rf_workflow,
  resamples = pm25_folds,
  grid = rf_grid
)

# Gradient Boosted Trees (XGBoost)
xgb_results <- tune_grid(
  xgb_workflow,
  resamples = pm25_folds,
  grid = xgb_grid
)


# Neural Network
```

```
nn_results <- tune_grid(
  nn_workflow,
  resamples = pm25_folds,
  grid = nn_grid
)
```

Now we can save the models in order to avoid rerunning (it took over 12 hours for me):

```
# KNN
write_rds(knn_results, file = "Models/KNN.rds")

# Elastic Net
write_rds(elastic_results, file = "Models/ElasticNet.rds")

# Random Forest
write_rds(rf_results, file = "Models/RF.rds")

# Gradient Boosted Trees (XGBoost)
write_rds(xgb_results, file = "Models/XGB.rds")

# Neural Network
write_rds(nn_results, file = "Models/NeuralNet.rds")
```

Then we can read in the tuned models:

```
# Linear Regression
lm_fit <- fit_resamples(lm_workflow, resamples = pm25_folds)

# KNN
knn_tuned <- read_rds(file = "Models/KNN.rds")

# Elastic Net
elastic_tuned <- read_rds(file = "Models/ElasticNet.rds")

# Random Forest
rf_tuned <- read_rds(file = "Models/RF.rds")

# Gradient Boosted Trees (XGBoost)
xgb_tuned <- read_rds(file = "Models/XGB.rds")

# Neural Network
nn_tuned <- read_rds(file = "Models/NeuralNet.rds")
```

Training Results

Next, we can display the results of training and tuning the models. We will display a table that shows the best performance for each model type in terms of RMSE:

```
# Linear Regression
lm_rmse <- lm_fit %>%
  collect_metrics() %>%
  filter(.metric == "rmse") %>%
  select(mean)

# best KNN
best_knn <- knn_tuned %>%
```

```
  select_best(metric = "rmse")

best_knn_rmse <- knn_tuned %>%
  collect_metrics() %>%
  filter(.config == best_knn$.config, .metric == "rmse") %>%
  select(mean)

# best Elastic Net
best_elastic <- elastic_tuned %>%
  select_best(metric = "rmse")

best_elastic_rmse <- elastic_tuned %>%
  collect_metrics() %>%
  filter(.config == best_elastic$.config, .metric == "rmse") %>%
  select(mean)

# best Random Forest
best_rf <- rf_tuned %>%
  select_best(metric = "rmse")

best_rf_rmse <- rf_tuned %>%
  collect_metrics() %>%
  filter(.config == best_rf$.config, .metric == "rmse") %>%
  select(mean)

# best XGBoost
best_xgb <- xgb_tuned %>%
  select_best(metric = "rmse")

best_xgb_rmse <- xgb_tuned %>%
  collect_metrics() %>%
  filter(.config == best_xgb$.config, .metric == "rmse") %>%
  select(mean)

# best Neural Network
best_nn <- nn_tuned %>%
  select_best(metric = "rmse")

best_nn_rmse <- nn_tuned %>%
  collect_metrics() %>%
  filter(.config == best_nn$.config, .metric == "rmse") %>%
  select(mean)
```

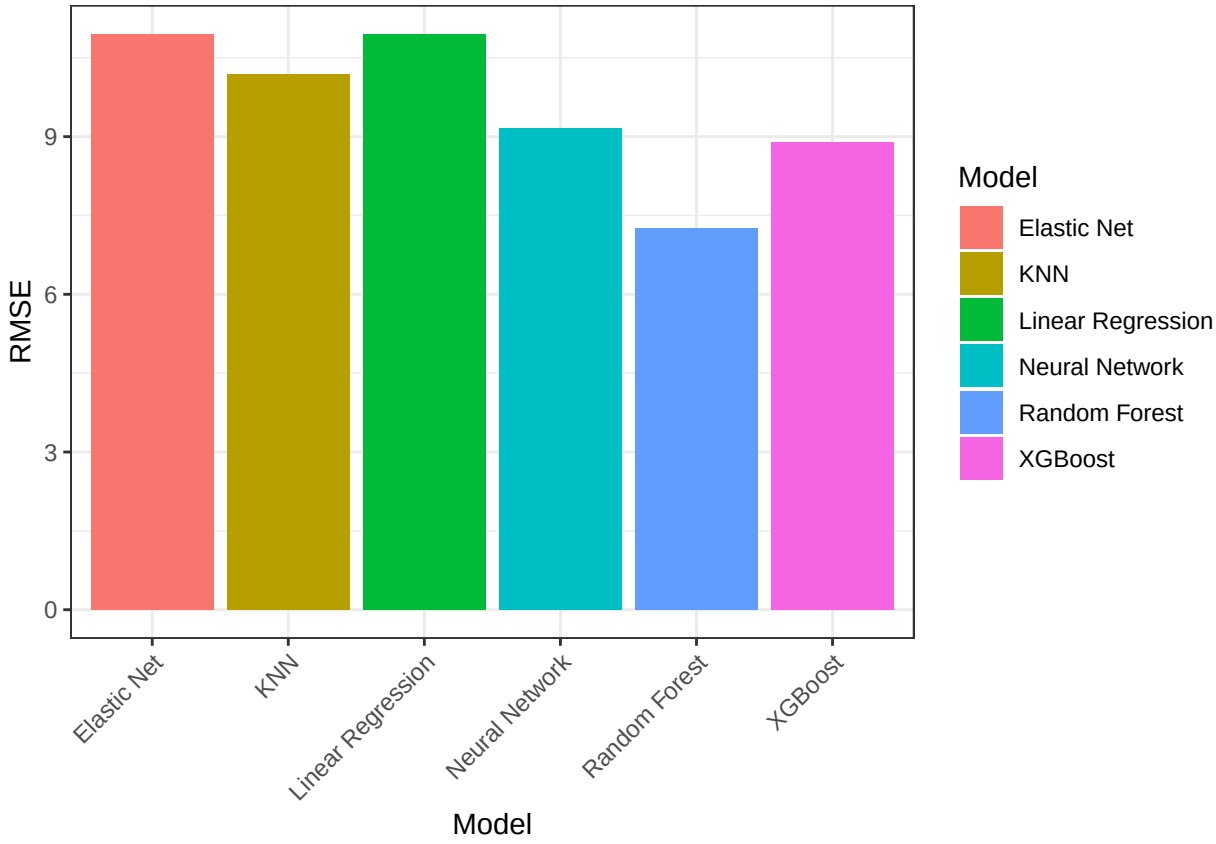
```
training_results <- tibble(
  Model = c("Linear Regression","KNN", "Elastic Net", "Random Forest",
            "XGBoost","Neural Network"),
  RMSE = c(lm_rmse$mean, best_knn_rmse$mean, best_elastic_rmse$mean,
           best_rf_rmse$mean, best_xgb_rmse$mean, best_nn_rmse$mean))
training_results
```

```
## # A tibble: 6 x 2
##   Model              RMSE
##   <chr>             <dbl>
## 1 Linear Regression 11.0
```

```
## 2 KNN               10.2
## 3 Elastic Net       10.9
## 4 Random Forest      7.25
## 5 XGBoost            8.89
## 6 Neural Network     9.16
```

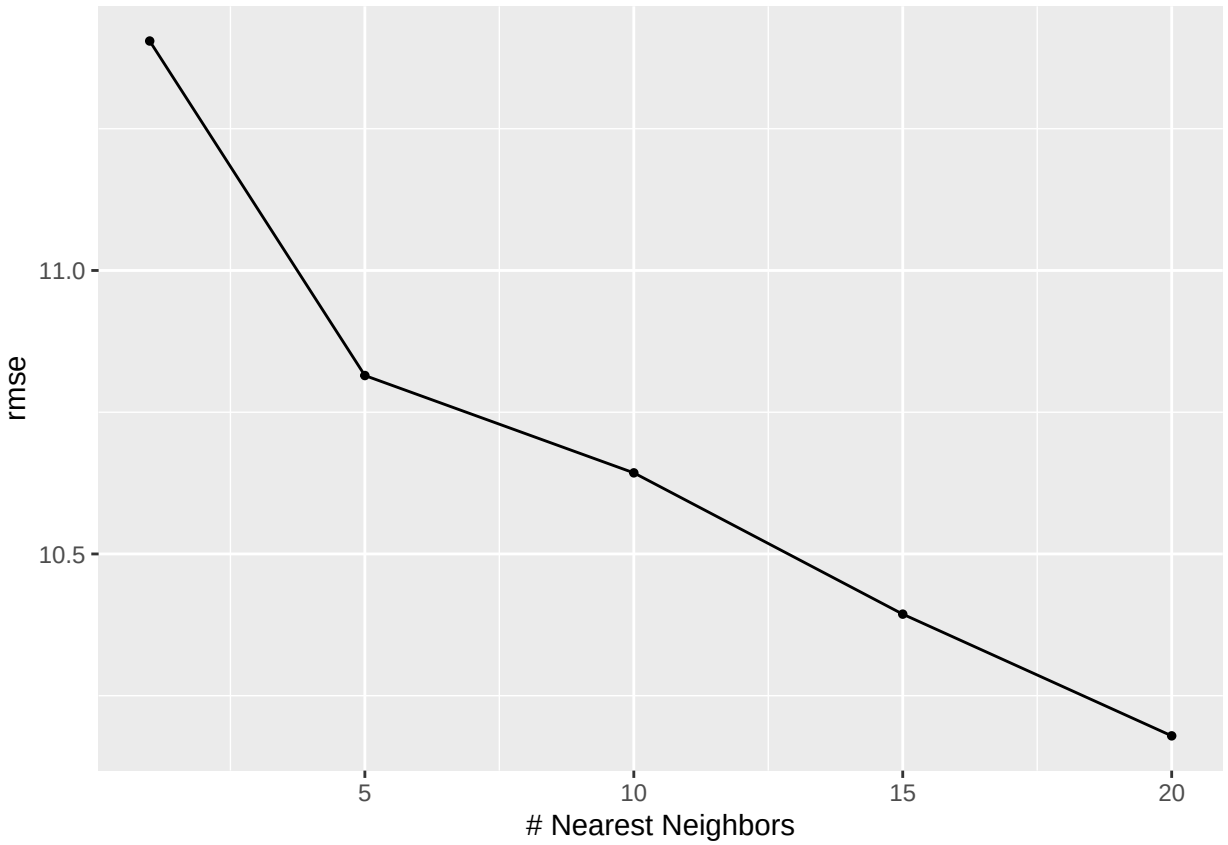
The best performing model (ie the one with the lowest RMSE) was the Random Forest model with an RMSE of 7.249566, and the second best was XGBoost with an RMSE of 8.893314. The worst performing model was Linear regression with an RMSE of 10.950933. To compare this visually, we can display a bar plot:

```
ggplot(training_results, aes(x = Model, y = RMSE, fill = Model)) +
  geom_bar(stat = "identity") +
  theme_bw() +
  labs(y = "RMSE", x = "Model") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



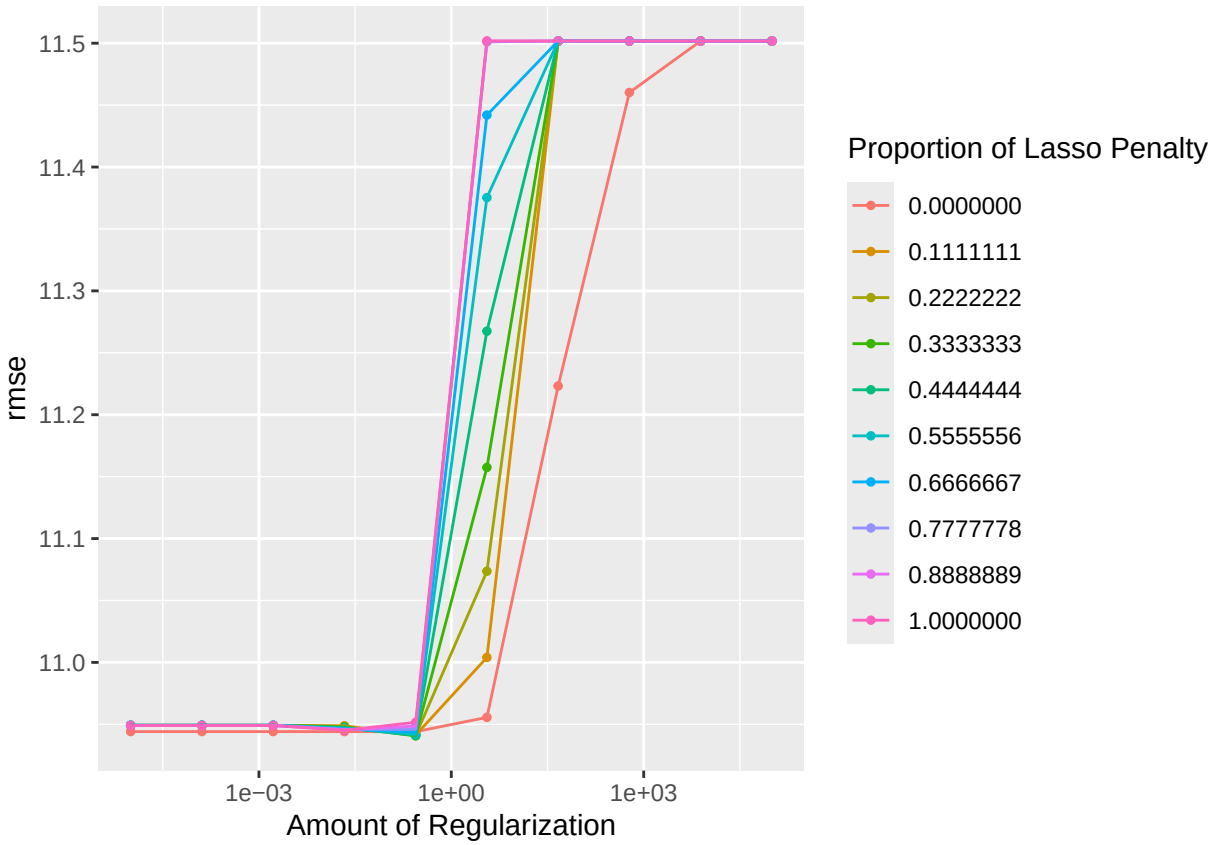
Additionally, we can visualize the performance of the models with different hyperparameters using autoplots.

```
autoplot(knn_tuned, metric="rmse")
```



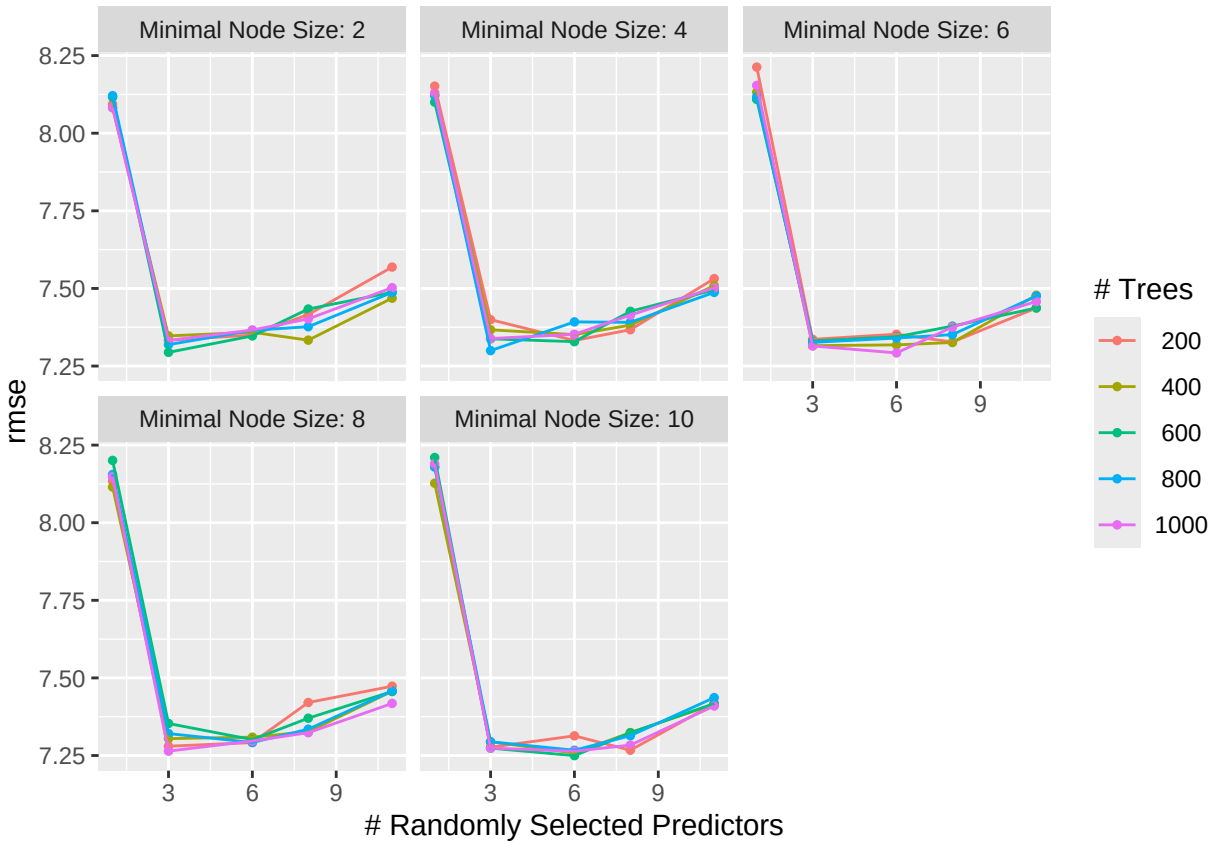
From the KNN autoplot, we can see a general decrease in RMSE as the value of k increases with the lowest RMSE associated with $k = 20$.

```
autoplot(elastic_tuned, metric="rmse")
```



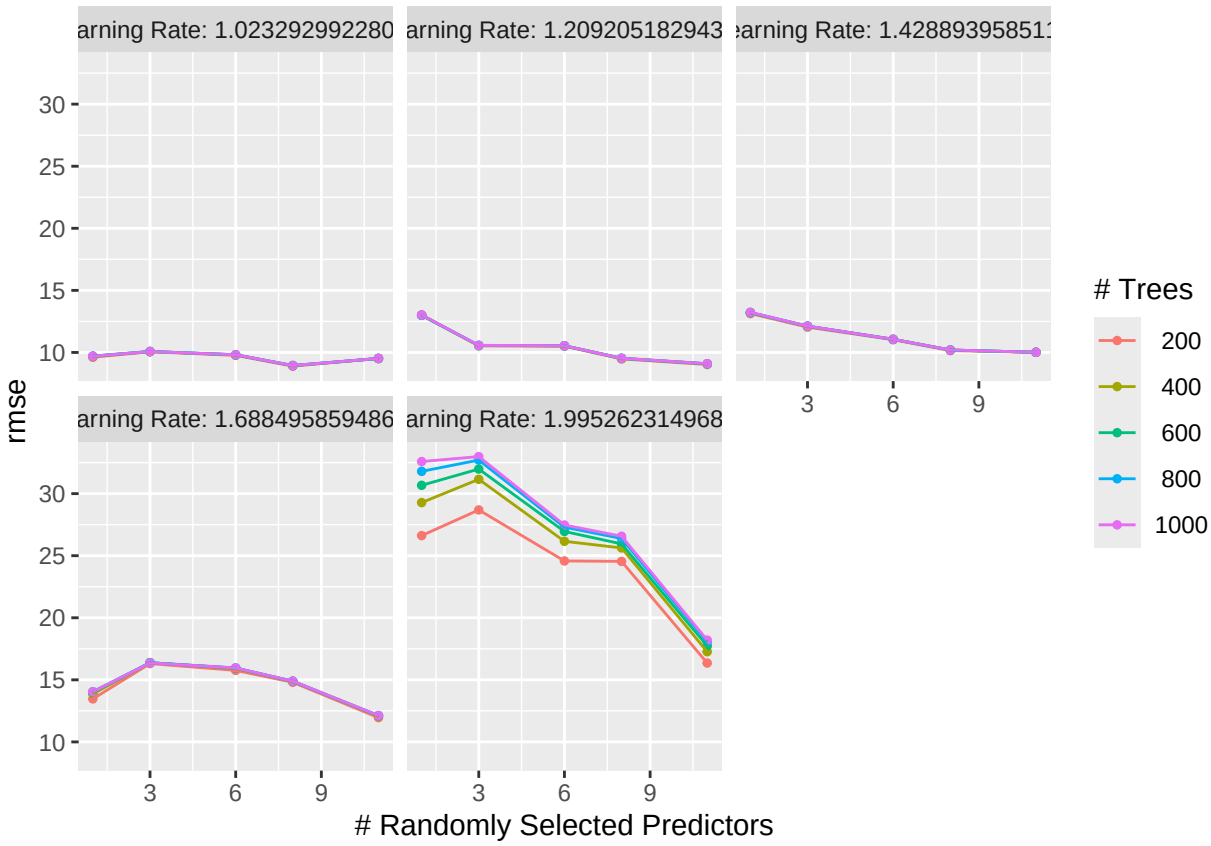
Finally, looking at the autoplot for Elastic Net, we can observe that lower values of regularization provides stable performance across all proportions of the Lasso penalty. As regularization increases, all models jump up in RMSE with higher proportions of Lasso penalty have the sharpest rises.

```
autoplot(rf_tuned, metric="rmse")
```



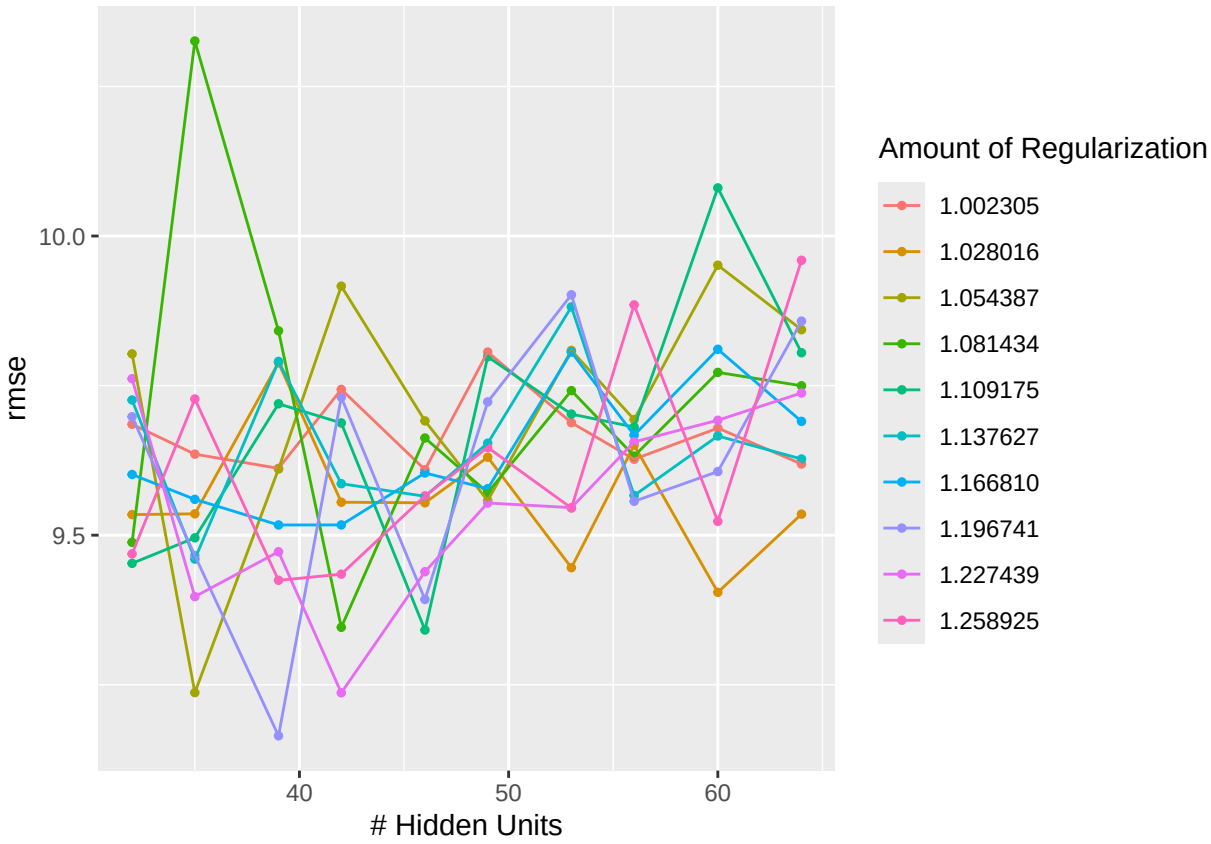
The Random Forest autoplot shows that the RMSE generally decreases when the number of randomly selected predictors is between 2.5 and 7.5, and starts increasing again with more predictors which is a consistent trend across all minimal node sizes. This shows that fewer randomly selected predictors (around 5) produce the best performance with a marginal difference in performance based on the number of trees.

```
autoplot(xgb_tuned, metric = 'rmse')
```



For the XGB model, learning rates around 1.02, 1.20, and 1.42, 1.69 cause the RMSE to remain stable and low regardless of the number of predictors or trees used. In contrast, the learning rate of 1.99 skyrocked the RMSE to over 30, which means lower learning rates provide better performance and stability while the higher learning rate leads to poor model performance.

```
autoplot(nn_tuned, metric="rmse")
```



The autoplot for the Neural Network shows some interesting patterns where the different amounts of regularization do not follow the same trend in contrast to the other autoplots have observed from the Random Forest and XGB. The fluctuations in RMSE across different hidden units and regularization values indicates that the models might be more sensitive to the hyperparameter choices and the initialization of the weights. From this plot, we can see that the model that had the lowest RMSE was the model with a regularization value of 1.196741 and around 39 hidden units.

Model Selection

The next thing to do is to choose the best model and proceed with evaluating it. From our results and analysis, the best performing model type was Random Forest. We can view the parameters of the model by extracting it:

```
best_rf <- rf_tuned %>%
  select_best(metric = "rmse")

best_rf
```

```
## # A tibble: 1 x 4
##    mtry trees min_n .config
##   <int> <int> <int> <chr>
## 1     6   600    10 Preprocessor1_Model113
```

The best Random Forest model was the one with 6 predictors randomly selected at each split, 600 trees grown in the forest, and a minimum of 10 data points required to form a node. Now we can fit it to the entire training set by finalizing the workflow:

```
final_rf_workflow <- rf_workflow %>%
  finalize_workflow(best_rf)
```

```
final_rf_model <- final_rf_workflow %>%
  fit(data = pm25_train)
```

Now we can see how the model performed on the entire training set.

```
rf_train_pred <- final_rf_model %>%
  predict(new_data = pm25_train)

rf_training_results <- pm25_train %>%
  bind_cols(rf_train_pred %>%
              rename(predicted_pm25 = .pred))

rf_training_results %>% metrics(truth = pm25, estimate = predicted_pm25)
```

```
## # A tibble: 3 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard       3.31
## 2 rsq     standard       0.929
## 3 mae     standard       1.32
```

From the results of fitting the Random Forest model on the entire training set, we can see that it had a lower RMSE than it did during cross validation. This is expected since the model is evaluated on the same data it was trained on, and since it has already seen the data it can fit more closely. During cross validation, since the model is trained on different folds and evaluated on a fold not used during training, the RMSE would be higher since some overfitting may occur.

Additionally, the model had a high R^2 of 0.9286769, meaning 92.87% of the variation in the training set could be explained by the model. This shows the model fit well to the training data, but we need to see how well it can generalize to unseen data.

Testing Results

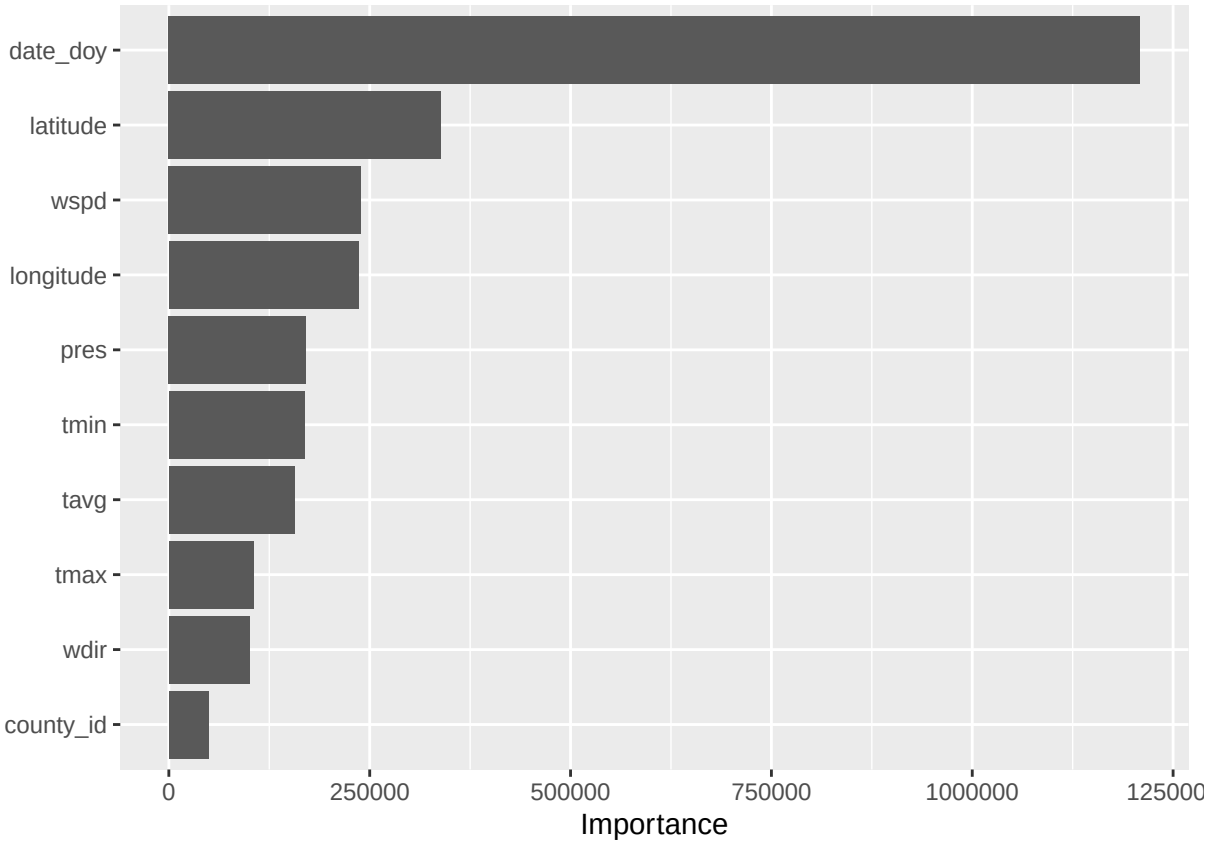
```
rf_predictions <- final_rf_model %>%
  predict(new_data = pm25_test)

rf_results <- pm25_test %>%
  bind_cols(rf_predictions %>%
              rename(predicted_pm25 = .pred))

rf_results %>% metrics(truth = pm25, estimate = predicted_pm25)
```

```
## # A tibble: 3 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard       5.89
## 2 rsq     standard       0.780
## 3 mae     standard       2.36
```

The results show that some slight overfitting occurred, but the model still achieved a somewhat low RMSE on the unseen data. Additionally, the R^2 value of 0.7803523 which means 78.03% of the total variation in the data could be explained by the model. This indicates that the model lost some predictive power compared to the training set, but it still was able to explain a large amount of the variation.

Next, we can see the importance of each predictor using the variable importance plot:

```
final_rf_model %>%
  extract_fit_engine() %>%
  vip()
```



From this plot we can see that the date (converted to the day of the year) was by far the most important feature for this model, followed by latitude and wind speed. This indicates there are strong temporal correlations, which matches our intuition because of the wildfires. Thus, date is an important predictor of PM2.5 concentrations for this model likely because certain periods of time during wildfire season are strongly associated with spikes in air pollution.

Conclusion

This project aimed to predict PM2.5 levels across different locations in California during the year 2020 using meteorological data and machine learning techniques. Among the models evaluated, Random Forest, XGBoost, and Neural Networks were the top performers, with Random Forest achieving the lowest RMSE. The reason for this could be its strong ability to capture the nonlinear relationships between air quality and weather patterns and the interactions between predictors.

In contrast, linear regression had the highest RMSE and struggled to accurately fit to the data. This could simply be due to the fact that the assumptions of linear regression were not met. In particular, the no collinearity assumption might not be met because some predictors are highly correlated (e.g. latitude and longitude have a correlation value of -0.85). There also might not be a linear relationship between all predictors and the outcome, which would violate the linear relationship assumption. Overall, more flexible models performed the best in capturing non-linear trends such as the impact of seasonal wildfires and spatial dependencies on PM2.5 levels.

This study has highlighted the ability of machine learning to provide a more robust framework for predicting

air quality during wildfires in California. By using models like Random Forest and XGBoost, researchers and professionals in the field could create better forecasting of dangerous PM2.5 levels, particularly during peak wildfire season. This can help ensure that people stay safe and prevent the harmful effects of PM2.5.

Future work should focus on handling both spatial and temporal dependencies in order to create more robust models that can achieve high accuracy on data from unseen locations. Incorporating spatial dependencies is particularly important because air quality is influenced by geographic factors such as elevation and proximity to pollution sources. Additionally, addressing multiple types of temporal dependencies such as seasonality, long term trends, and the effects of short term events like wildfires could potentially improve modeling changes in PM2.5 levels over time. Overall, this study has provided a good starting point into the uses of machine learning for forecasting air quality, especially after natural disaster events such as wildfires.

References

California Department of Forestry and Fire Protection (2021). 2020 Fire Season. Available at: https://www.fire.ca.gov/incidents/2020/

Martichoux, A. and Jue, T. (2020) Orange San Francisco sky looks even wilder in drone video, ABC7 News. Available at: https://abc7news.com/why-is-the-sky-orange-sf-yellow-california/6415705/

Sergent, J., Petras, G., Gelles, K. and Bacon, J. (2018) 3 startling facts about California's Camp Fire, USA Today. Available at: https://www.usatoday.com/story/news/2018/11/20/camp-fire-3-startling-facts/2064758002/

Stelloh, T. (2020) California exceeds 4 million acres burned by wildfires in 2020, NBC News. Available at: https://www.nbcnews.com/news/us-news/california-exceeds-4-million-acres-burned-wildfires-2020-n1242078

World Health Organization (WHO). (2021) Ambient (outdoor) air pollution. Available at: https://www.who.int/news-room/fact-sheets/detail/ambient-(outdoor)-air-quality-and-health