EDA using R

```r
# Calling the Libraries used
library(readxl)
library(dplyr)
library(lubridate)
library(readr)
library(ggplot2)
library(ggthemes)
library(tidyr)
library(DT)
library(scales)
library(stringr)
library(knitr)
library(FactoMineR)
library(ggpubr)
library(kableExtra)
library(magrittr)
library(ggfortify)
```

```r
# Reading the data from CSV file and loading into a Dataframe
ozone_df <- read.csv("daily_ozone_2022.csv")
# Reading few datasamples from the CSV file
ozone_df <- sample_n(ozone_df, 5000)
# Printing the Dimensions of the Dataframe
dim(ozone_df)
```

```
## [1] 5000   29
```

```r
# Examining the structure of the Dataframe
str(ozone_df)
```

```
## 'data.frame':    5000 obs. of  29 variables:
##  $ State.Code         : int  51 6 21 32 53 48 51 6 32 22 ...
##  $ County.Code        : int  113 77 185 33 73 355 113 25 3 57 ...
##  $ Site.Num           : int  3 1003 4 101 5 25 3 1003 299 4 ...
##  $ Parameter.Code     : int  44201 44201 44201 44201 44201 44201 44201 44201 44201 44201 ...
##  $ POC                : int  1 1 1 1 1 2 1 1 1 1 ...
##  $ Latitude           : num  38.5 38 38.4 39 49 ...
##  $ Longitude          : num  -78.4 -121.3 -85.4 -114.2 -122.6 ...
##  $ Datum              : chr  "WGS84" "WGS84" "WGS84" "WGS84" ...
##  $ Parameter.Name     : chr  "Ozone" "Ozone" "Ozone" "Ozone" ...
##  $ Sample.Duration    : chr  "8-HR RUN AVG BEGIN HOUR" "8-HR RUN AVG BEGIN HOUR" "8-HR RUN AVG BEGIN
##  $ Pollutant.Standard : chr  "Ozone 8-hour 2015" "Ozone 8-hour 2015" "Ozone 8-hour 2015" "Ozone 8-ho
##  $ Date.Local         : chr  "2022-10-29" "2022-05-12" "2022-10-18" "2022-10-29" ...
##  $ Units.of.Measure   : chr  "Parts per million" "Parts per million" "Parts per million" "Parts per
```

```
## $ Event.Type        : chr "None" "None" "None" "None" ...
## $ Observation.Count : int 17 3 17 17 17 17 17 17 17 17 ...
## $ Observation.Percent: num 100 18 100 100 100 100 100 100 100 100 ...
## $ Arithmetic.Mean   : num 0.0385 0.0387 0.0218 0.0398 0.0233 ...
## $ X1st.Max.Value    : num 0.042 0.04 0.023 0.042 0.04 0.011 0.044 0.034 0.028 0.035 ...
## $ X1st.Max.Hour     : int 20 9 12 20 10 21 18 9 9 9 ...
## $ AQI               : int 39 37 21 39 37 10 41 31 26 32 ...
## $ Method.Code       : int NA NA NA NA 87 87 47 NA 87 NA ...
## $ Method.Name       : chr " - " " - " " - " " - " ...
## $ Local.Site.Name   : chr "Shenandoah NP - Big Meadows" "Stockton - University Park" "BUCKNER" "G
## $ Address           : chr "SHENANDOAH NP BIG MEADOWS" "702 N Aurora Street, Stockton, CA 95202" "
## $ State.Name        : chr "Virginia" "California" "Kentucky" "Nevada" ...
## $ County.Name       : chr "Madison" "San Joaquin" "Oldham" "White Pine" ...
## $ City.Name         : chr "Not in a city" "Stockton" "Buckner" "Not in a city" ...
## $ CBSA.Name         : chr "" "Stockton-Lodi, CA" "Louisville/Jefferson County, KY-IN" "" ...
## $ Date.of.Last.Change: chr "2023-03-16" "2023-03-17" "2023-02-20" "2023-03-16" ...
```

```
# Examining the Dataframe, it appears come columns should be converted to Factors and dates
ozone_df$Parameter.Code <- as.factor(ozone_df$Parameter.Code)
ozone_df$Units.of.Measure <- as.factor(ozone_df$Units.of.Measure)
# Handling Date columns
ozone_df$Date.Local <- as.Date(ozone_df$Date.Local)
ozone_df$Date.of.Last.Change <- as.Date(ozone_df$Date.of.Last.Change)
# Printing sample rows from Dataframe
kbl(head(ozone_df[1:6,c(1:8)]), caption = "Ozone Pollutant Data",booktabs = T) %>% kable_styling(latex_
```

Table 1: Ozone Pollutant Data

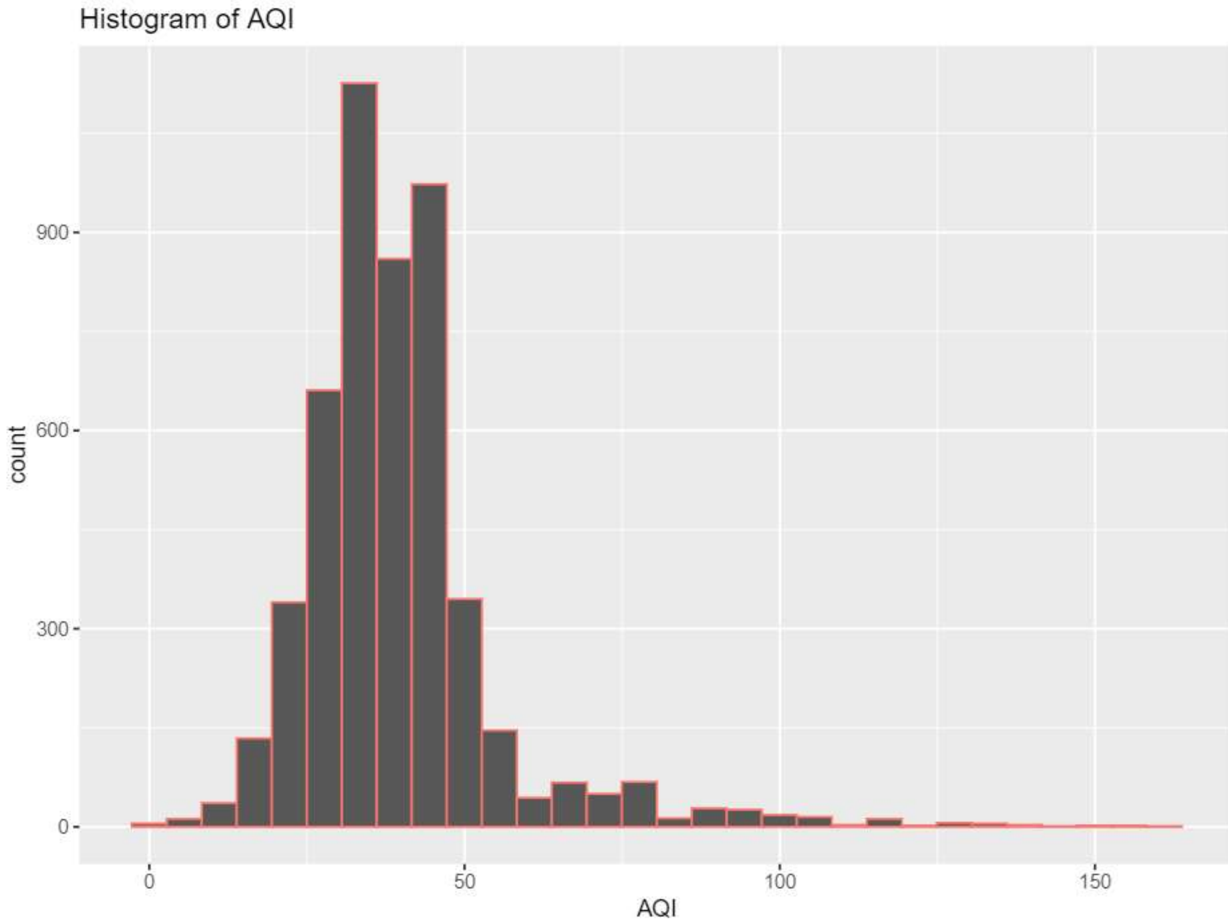
State.Code	County.Code	Site.Num	Parameter.Code	POC	Latitude	Longitude	Datum
51	113	3	44201	1	38.52310	-78.43471	WGS84
6	77	1003	44201	1	37.96158	-121.28141	WGS84
21	185	4	44201	1	38.40020	-85.44428	WGS84
32	33	101	44201	1	39.00512	-114.21593	WGS84
53	73	5	44201	1	48.95074	-122.55441	WGS84
48	355	25	44201	2	27.76534	-97.43426	WGS84

```
# Checking for Nulls in the Ozone Dataframe
colSums(is.na(ozone_df))
```

```
##        State.Code       County.Code          Site.Num     Parameter.Code
##                 0                 0                 0                  0
##               POC          Latitude         Longitude              Datum
##                 0                 0                 0                  0
##    Parameter.Name   Sample.Duration Pollutant.Standard         Date.Local
##                 0                 0                 0                  0
##  Units.of.Measure        Event.Type Observation.Count Observation.Percent
##                 0                 0                 0                  0
##   Arithmetic.Mean    X1st.Max.Value     X1st.Max.Hour                AQI
##                 0                 0                 0                  0
##       Method.Code       Method.Name    Local.Site.Name            Address
```

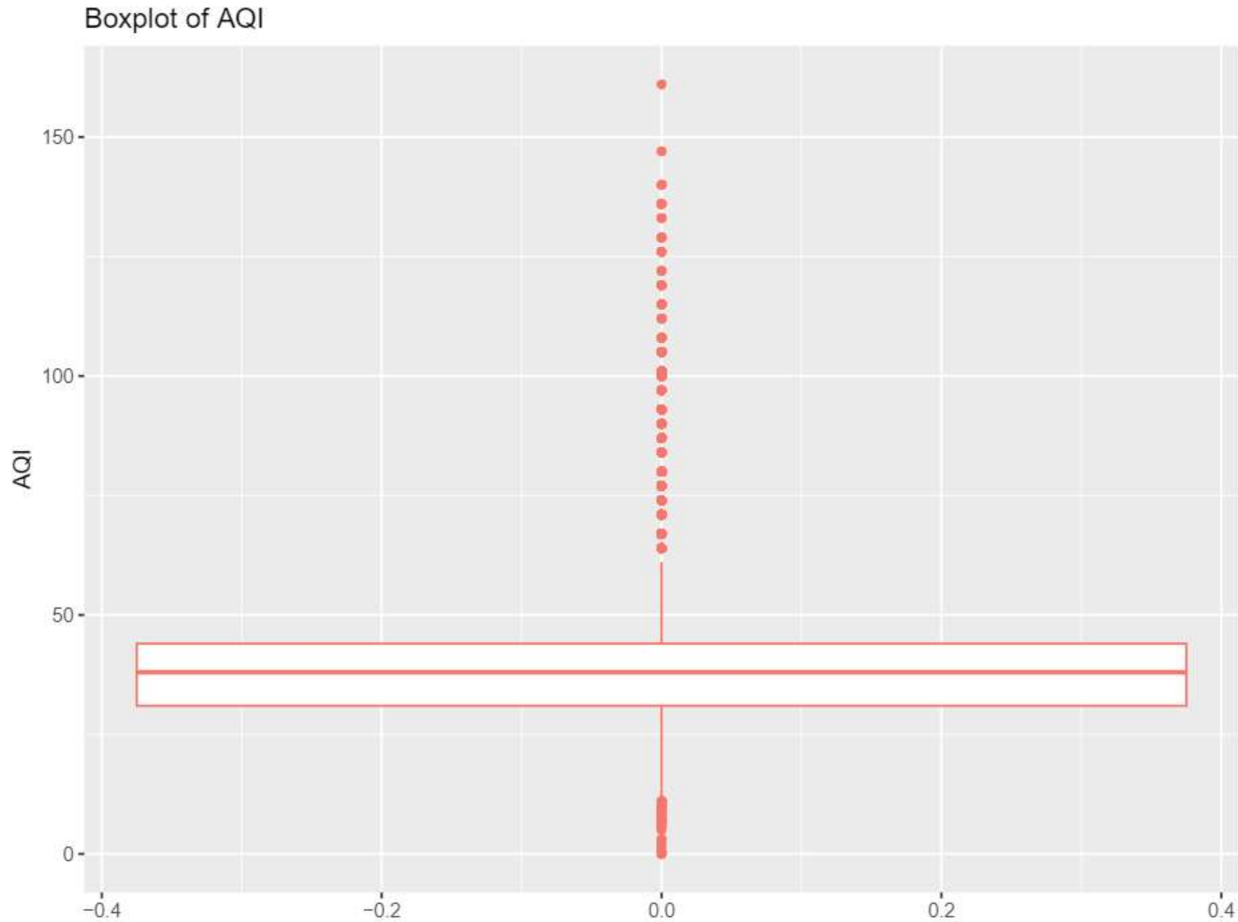
```
##                 1743                    0                    0                    0
##           State.Name          County.Name            City.Name            CBSA.Name
##                    0                    0                    0                    0
## Date.of.Last.Change
##                    0
```

```r
# Plotting Histogram to examine the distribution of AQI
ggplot(ozone_df,aes(x=AQI))+geom_histogram(aes(color="red"))+ggtitle(label = 'Histogram of AQI')+  them
```



Histogram Results: The Histogram results indicate that the AQI is heavily distributed between 20 and 65 and has a long tail on the right indicating outliers.

```r
# Plotting Boxplot to examine the distribution of AQI
ggplot(ozone_df,aes(y=AQI))+geom_boxplot(aes(color="red"))+ggtitle(label = 'Boxplot of AQI')+ theme(leg
```



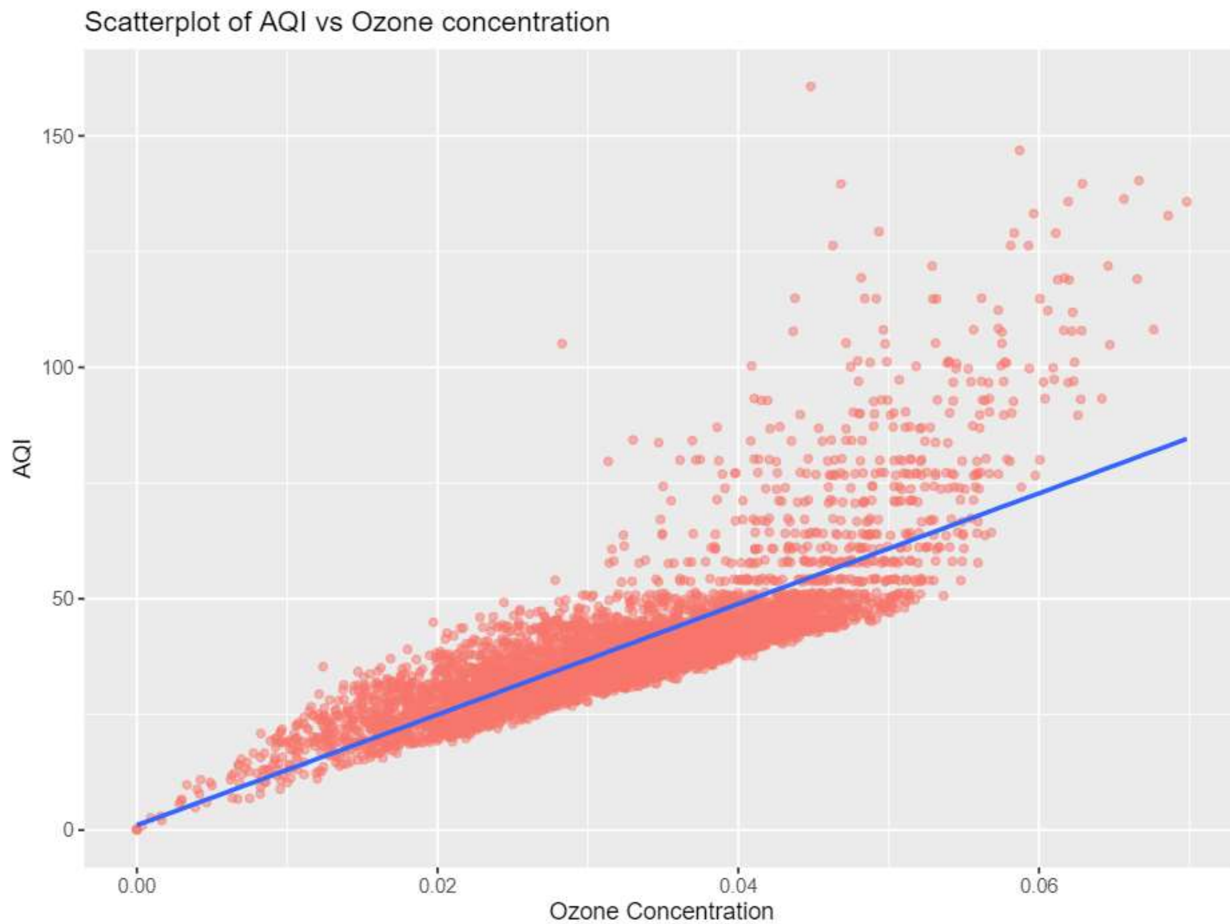
Boxplot Results: The Boxplot results indicate that the presence of Ouliers above an AQI of 65 and also below 20. Most of the Data is between 20 and 65.

```
# Computing the first and third quartiles
q1 <- quantile(ozone_df$AQI, 0.25)
q3 <- quantile(ozone_df$AQI, 0.75)
iqr <- q3 - q1
# Calculate the lower and upper cutoffs for outliers
lower <- q1 - 1.5 * iqr
upper <- q3 + 1.5 * iqr
# Filter AQI to find outliers
AQI_outliers <- ozone_df %>%
  filter(AQI > upper | AQI < lower)
# Printing the top few rows of Outliers
kbl(head(AQI_outliers[1:6,c(1:8)]), caption = "Outliers in the AQI",booktabs = T) %>% kable_styling(lat
```

```
ggplot(ozone_df,aes(x=Arithmetic.Mean,y=AQI))+geom_point(aes(color="Blue"),alpha=0.5,position="jitter")
```

Table 2: Outliers in the AQI

State.Code	County.Code	Site.Num	Parameter.Code	POC	Latitude	Longitude	Datum
48	355	25	44201	2	27.76534	-97.43426	WGS84
6	43	6	44201	1	37.54377	-119.83957	NAD83
48	201	1034	44201	2	29.76800	-95.22058	WGS84
15	3	1004	44201	2	21.30338	-157.87117	WGS84
39	167	4	44201	1	39.43212	-81.46044	NAD83
4	21	3003	44201	1	32.95436	-111.76225	WGS84



Scatterplot Results: The Scatterplot indicates a strong positive relation between the Ozone Concentration and AQI.

```
# Calculating the correlation coefficient
cor(ozone_df$Arithmetic.Mean,ozone_df$AQI)
```

```
## [1] 0.8041176
```

Correlation Results: The output of the correlation coefficient indicates a Strong positive correlation (0.8) between the Ozone concentration and the Air Quality Index values.

Data Preparation and Building Models using R

```r
# Calling the Libraries used
library(readxl)
library(dplyr)
library(lubridate)
library(readr)
library(ggplot2)
library(ggthemes)
library(tidyr)
library(scales)
library(stringr)
library(knitr)
library(FactoMineR)
library(ggpubr)
library(kableExtra)
library(magrittr)
library(ggfortify)
library(visdat)
library(usmap)
library(plotly)
library(leaflet)
library(magrittr)
library(treemap)
library(olsrr)
```

```r
# Function to create a dataframe by reading the contents from csv file
create_dataframe <- function(df,file_name){
    df <- read.csv(file_name,stringsAsFactors=FALSE)
    return(df)
}
# Function to rename the columns into required format
rename_columns <- function(df){
    df <- df %>% select(state_code=State.Code,county_code=County.Code,site_num=Site.Num,
                                    date_local=Date.Local,
                                   new_col=Arithmetic.Mean,AQI
                                   )
    return (df)
}
# Function to format the columns by changing the datatype to factors and Dates
format_columns <- function(df){
    df$state_code <- as.factor(df$state_code)
    df$county_code <-  as.factor(df$county_code)
    df$site_num <- as.factor(df$site_num)
    df$date_local <- as.Date(df$date_local)
    df$AQI[is.na(df$AQI)] <- 0
    return (df)
}
```

```r
# Function to find nulls in the dataframe by taking the column sums
find_nulls <- function(df){
    colSums(is.na(df))
}
# Function to find the correlation between the AQI and the field new_col
find_AQI_corr <- function(df){
    cor(df$new_col,df$AQI)
}
# Function to remove duplicates by grouping the data on common columns and retrieving only the distinct
remove_duplicates <- function(df) {
    df <- df %>% group_by(state_code,county_code,site_num,date_local) %>% distinct(state_code,county_co
    return (df)
}
```

```r
# Reading the contents of NO2 concentration into a dataframe for performing the EDA analysis
NO2_EDA_df <- create_dataframe(NO2_EDA_df,"daily_NO2_2022.csv")
# Selecting the required columns from the Dataframe
NO2_EDA_df <- NO2_EDA_df %>% select(state_code=State.Code,county_code=County.Code,site_num=Site.Num,
                                      date_local=Date.Local,
                                     NO2=Arithmetic.Mean,AQI,
                      state_name=State.Name,county_name=County.Name,city_name=City.Name)
# Converting the columns datatype to Factor and Date
NO2_EDA_df <- format_columns(NO2_EDA_df)
```

```r
# This function calls multiple functions to prepare the data for merging
data_preparation <- function(df,filename){
    # Calling function to create dataframe
    df <- create_dataframe(df,filename)
    # Calling function to rename the columns
    df <- rename_columns(df)
    # Function to Format the columns
    df <- format_columns(df)
    # Function to remove duplicates
    df <- remove_duplicates(df)
    return(df)
}
```

```r
# Calling Data preparation function on multiple source files each resulting in individual dataframes
NO2_df <- data_preparation(df=NO2_df,filename="daily_NO2_2022.csv")
ozone_df <- data_preparation(ozone_df,"daily_ozone_2022.csv")
SO2_df <- data_preparation(SO2_df,"daily_SO2_2022.csv")
CO_df <- data_preparation(CO_df,"daily_CO_2022.csv")
df_81102 <- data_preparation(df_81102,"daily_81102_2022.csv")
wind_df <- data_preparation(wind_df,"daily_WIND_2022.csv")
temp_df <- data_preparation(CO_df,"daily_TEMP_2022.csv")
press_df <- data_preparation(CO_df,"daily_PRESS_2022.csv")
rh_dp_df <- data_preparation(CO_df,"daily_RH_DP_2022.csv")
```

```r
# This function renames the "new_col" column into the desired name
rename_col2 <- function(df,col_name){
    df[[col_name]] <- df$new_col
    df <- df %>% select(-new_col)
    return(df)
```

```
}
# Function to join dataframes using the "left join"
merge_dataframes <- function(df1,df2,col_name){
    # Performs left join  and selects required columns
    merge_df <- left_join(df1,df2,by=c("state_code","county_code","site_num","date_local")) %>% rename(
    # Replaces the nulls in numeric columns after left join with median values
    merge_df[[col_name]][is.na(merge_df[[col_name]])]<-median(merge_df[[col_name]],na.rm=TRUE)
    return(merge_df)
}
```

```
# THis section calls the rename_col2 function to rename the "new_col" field with the desired name
ozone_df = rename_col2(ozone_df,col_name="ozone")
SO2_df = rename_col2(SO2_df,col_name="SO2")
NO2_df = rename_col2(NO2_df,col_name="NO2")
CO_df = rename_col2(CO_df,col_name="CO")
df_81102 = rename_col2(df_81102,col_name="PM")
wind_df = rename_col2(wind_df,col_name="wind")
temp_df = rename_col2(temp_df,col_name="temp")
press_df = rename_col2(press_df,col_name="press")
rh_dp_df = rename_col2(rh_dp_df,col_name="RH")
```

```
# Calling the Join function to perform the joins
merge1_df=merge_dataframes(NO2_df,ozone_df,col_name="ozone")
merge2_df=merge_dataframes(merge1_df,SO2_df,col_name="SO2")
merge3_df=merge_dataframes(merge2_df,CO_df,col_name="CO")
merge4_df=merge_dataframes(merge3_df,df_81102,col_name="PM")
merge5_df=merge_dataframes(merge4_df,wind_df,col_name="wind")
merge6_df=merge_dataframes(merge5_df,temp_df,col_name="temp")
merge7_df=merge_dataframes(merge6_df,press_df,col_name="press")
merge8_df=merge_dataframes(merge7_df,rh_dp_df,col_name="RH")
```

```
# Validating nulls in the final merged dataframe
find_nulls(merge8_df)
```

```
##  state_code county_code    site_num  date_local         AQI         NO2
##           0           0           0           0           0           0
##       ozone         SO2          CO          PM        wind        temp
##           0           0           0           0           0           0
##       press          RH
##           0           0
```

Visualizing the data

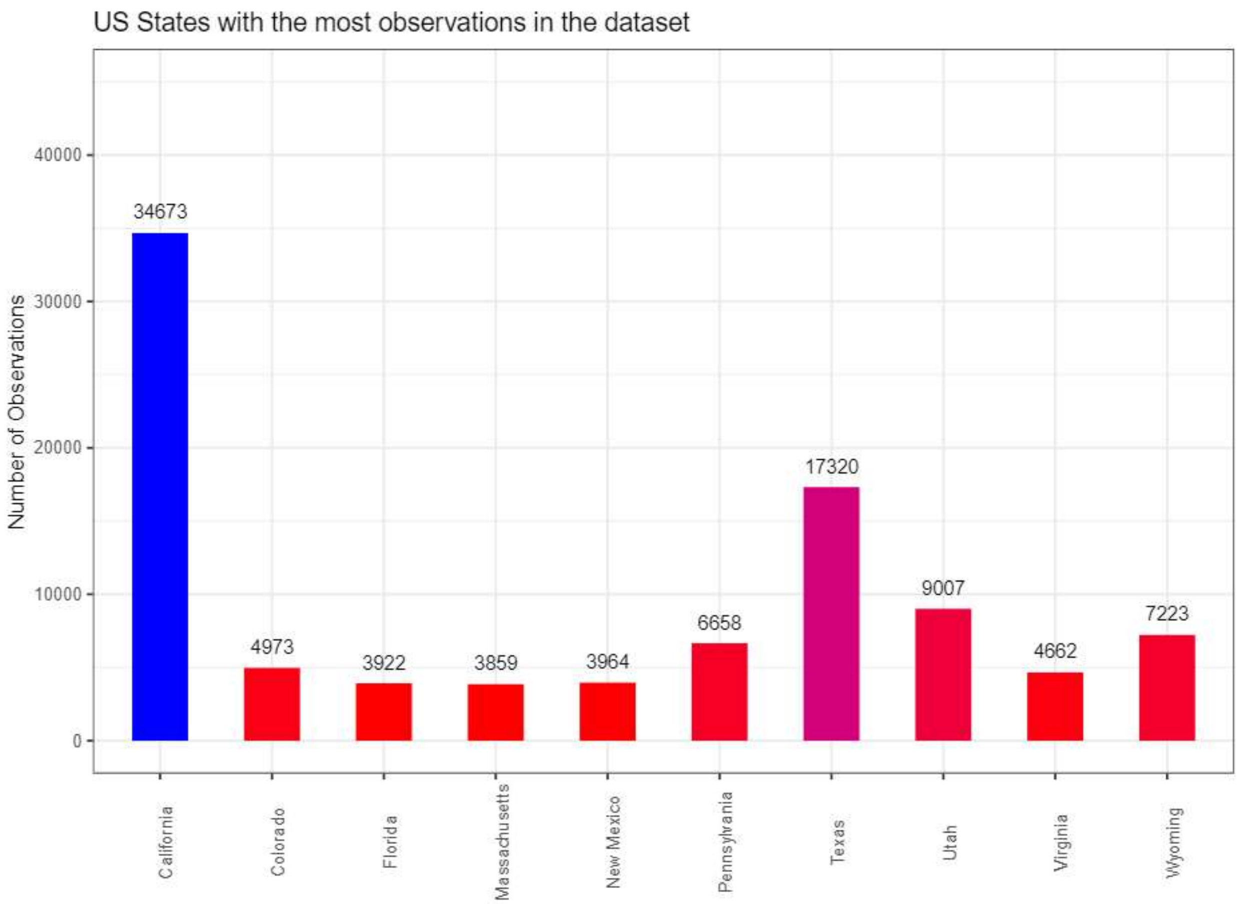
```
# Performing join with the EDA df to retrieve the state name, county name and location details
merge9_df <- merge8_df %>% inner_join(NO2_EDA_df,by=c("state_code","county_code","site_num","date_local
```

```
# Including Region details to the dataframe based on the state names
merge10_df <- merge9_df %>% mutate(
    region_name= case_when(
        state_name %in% c("Maine","Vermont","Massachusetts","Rhode Island","Connecticut","New Hampshire
```

```
        state_name %in% c("New York","Pennsylvania","New Jersey","Delaware","Maryland") ~ "Mid_Atlantic
        state_name %in% c("Arkansas","Louisiana","Mississippi","Alabama","Georgia","Florida","Tennessee
        state_name %in% c("North Dakota","South Dakota","Nebraska","Kansas","Missouri","Iowa","Minnesot
        state_name %in% c("Nevada","Utah","Colorado","Wyoming","Idaho","Montana") ~ "Rocky_Mountain_Sta
        state_name %in% c("Washington","California","Oregon","Alaska","Hawaii") ~ "Pacific_coastal",
        state_name %in% c("Arizona","New Mexico","Oklahoma","Texas") ~ "South_West",
        state_name %in% c("District Of Columbia") ~ "District Of Columbia"
    )
)
```

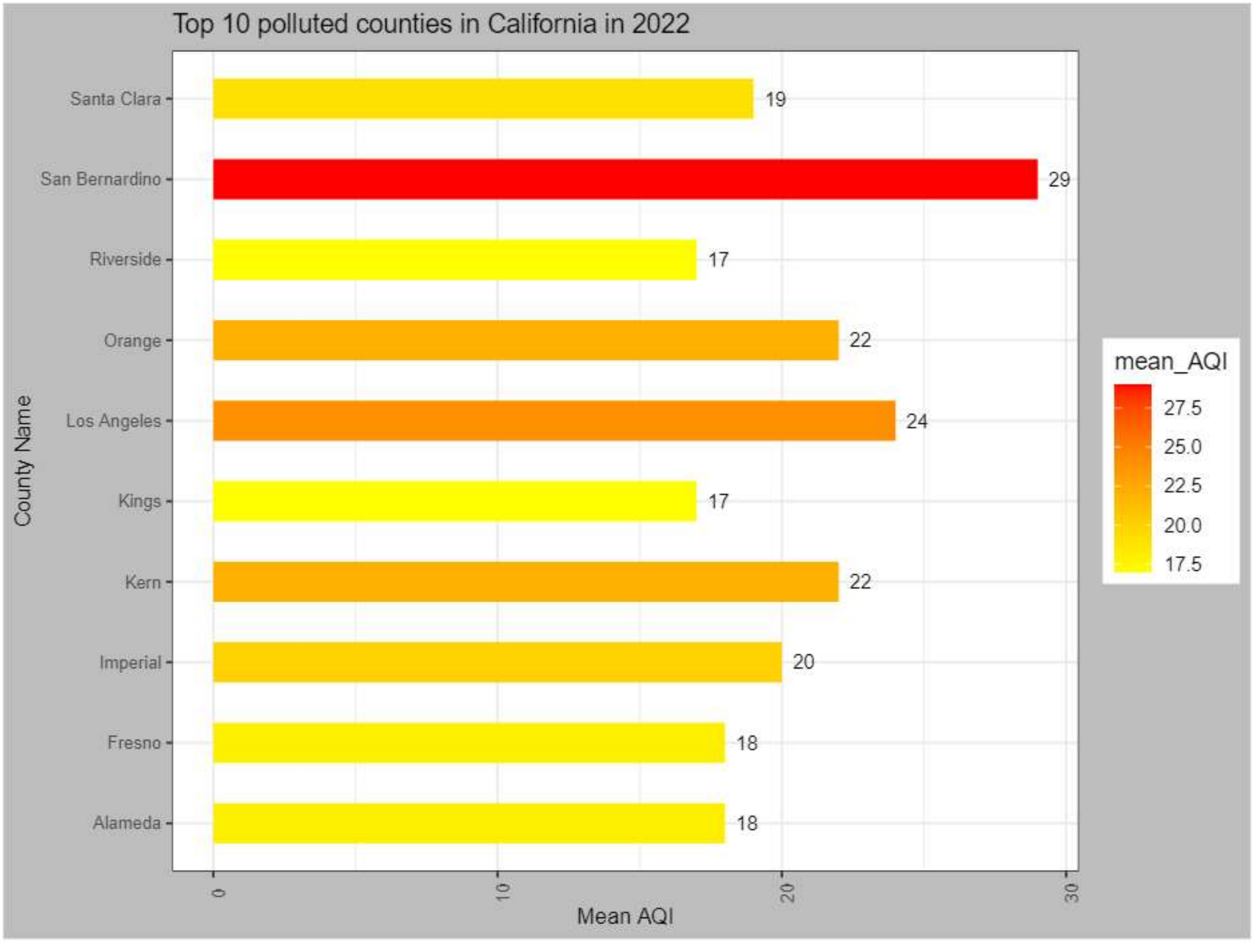
```
# Counting the number of entries in the final dataframe by state.
count_by_state_df <- merge10_df %>% group_by(state_name) %>% count() %>% arrange(desc(n)) %>% rename(ob
count_by_state_df <- head(count_by_state_df,10)
```

```
# Creating a Bar plot of the count based on the state count
ggplot(data = count_by_state_df, aes(state_name, observations,fill = observations)) + geom_bar(stat = "
geom_text(aes(label = observations), vjust = -1,
size = 3) + scale_fill_gradient(low = "Red", high = "blue") +
labs(x = "", y = "Number of Observations",
title = "US States with the most observations in the dataset") +
scale_y_continuous(labels = comma) + ylim(0, 45000) + theme_bw() + theme(plot.title = element_text(size
angle = 90), axis.text.y = element_text(size = 8),
axis.title = element_text(size = 10),
legend.position = "none")
```

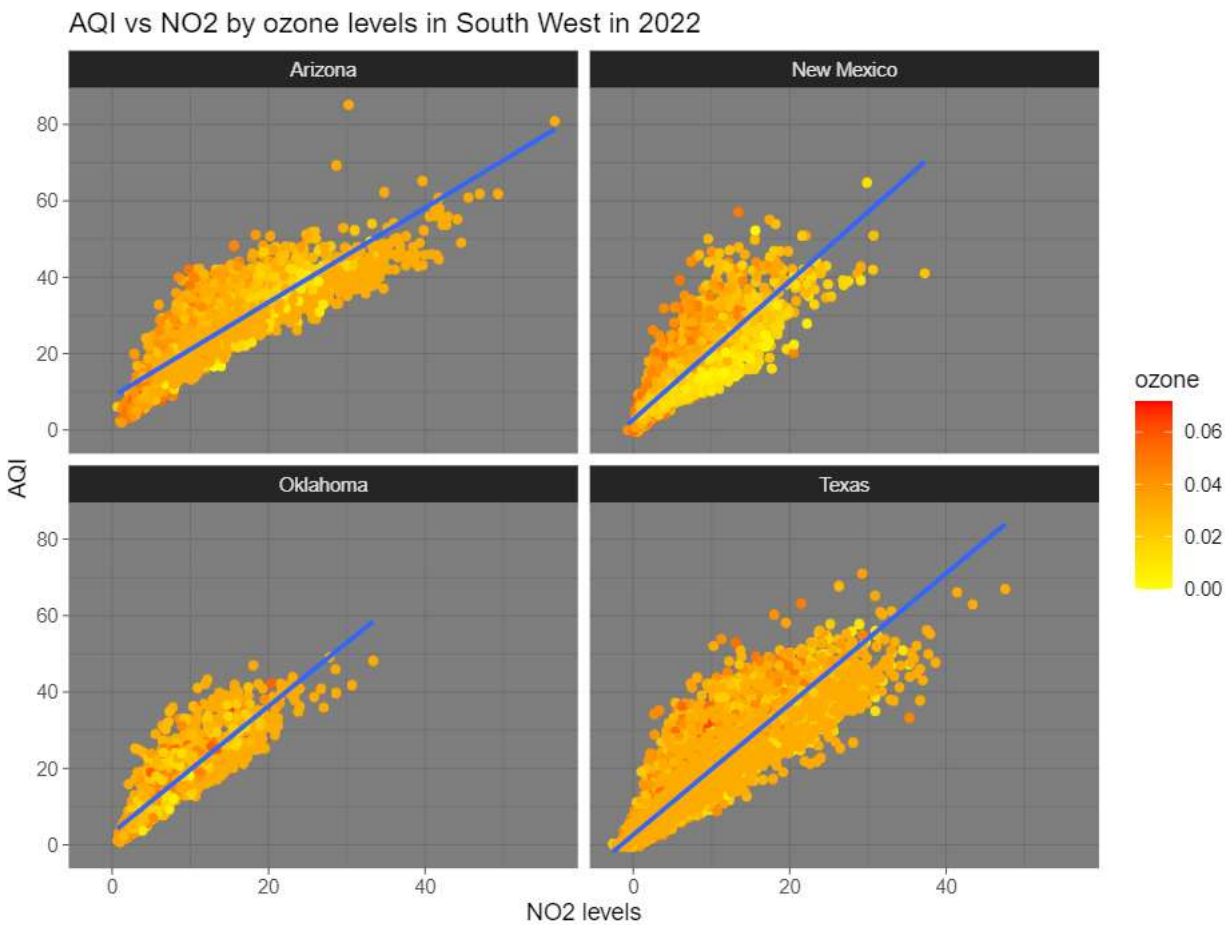



```r
# Creating a dataframe for California by identifying the most polluted counties based on the AQI
CA_AQI_df <- merge10_df %>% filter(state_name=="California") %>% group_by(county_name) %>% summarize(me
```

```r
# Creating a bar plot of CA_AQI_df with the coordinates flipped for better display
ggplot(data = CA_AQI_df, aes(county_name, mean_AQI)) + geom_bar(stat = "identity",
width = 0.5, aes(fill = mean_AQI)) + scale_fill_gradient(low = "Yellow",
high = "Red") + coord_flip() + labs(x = "County Name",
y = "Mean AQI", title = "Top 10 polluted counties in California in 2022") +
geom_text(aes(label = mean_AQI), hjust = -0.5,
size = 3) + theme_bw() + theme(plot.title = element_text(size = 12),
axis.text.x = element_text(size = 8, angle = 90), axis.text.y = element_text(size = 8),
axis.title = element_text(size = 10), plot.background = element_rect(fill = "Grey"))
```

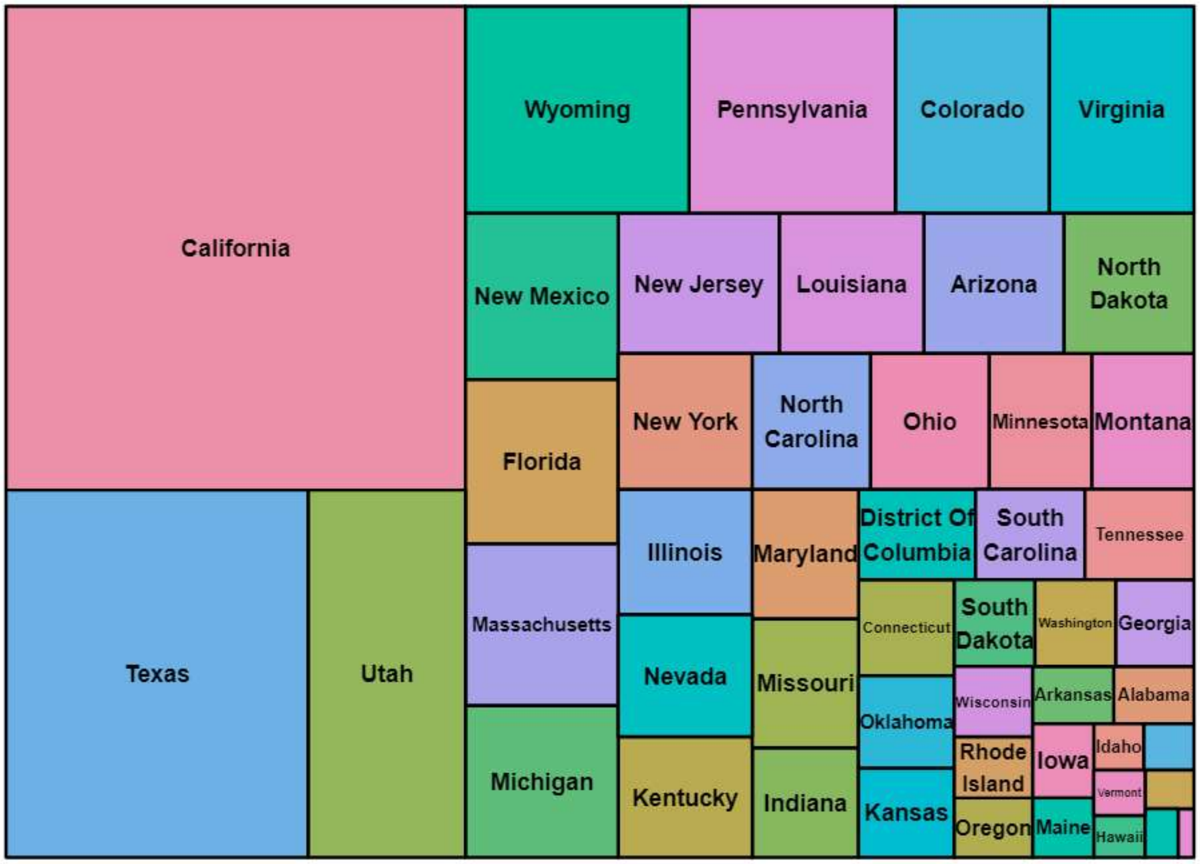



```
# Creating a dataframe for South west region data
SW_df1 <- merge10_df %>% filter(region_name=="South_West")
# Plotting a scatter plot with facet wrap to display the AQI vs NO2 levels based on Ozone concentration
ggplot(SW_df1, aes(x=NO2,y=AQI,color=ozone)) + facet_wrap(~state_name)+
  geom_point(alpha=0.5)+ geom_jitter()+ geom_smooth(method="lm",se=FALSE)+
    scale_color_gradient(low = 'yellow', high = 'red')+
    labs(y="AQI",
      x="NO2 levels",
      title="AQI vs NO2 by ozone levels in South West in 2022")+
      theme_dark()
```




```
# Creating a treemap of number of entries in the Ozone dataset
count_df <- merge10_df %>% group_by(state_name) %>% count()
treemap( count_df, index = "state_name", vSize = "n", type = "index", title = "Treemap of Ozone Observa
```

Treemap of Ozone Observations in each state




```
# Calculating the Median value of NO2 in the dataset across all states
median_NO2 <- median(merge10_df$NO2)
median_NO2
```

```
## [1] 5.85
```

```
# Creating a compare dataframe that compares the National median versus each state value
compare_df1 <- merge10_df %>% group_by(state_name) %>% summarize(median_state_NO2=median(NO2)) %>% muta
compare_df1 <- head(compare_df1,20)
```

```
# Plotting the compare dataframe displaying each state with above and below the national median values
ggplot(compare_df1, aes(x=state_name, y=difference)) +
  geom_bar(stat='identity', aes(fill=compare_val), width=.5)  +
  scale_fill_manual(name="NO2 Median Values",
                    labels = c("Above National Average", "Below National Average"),
                    values = c("above"="#00ba38", "below"="#f8766d")) +
  coord_flip()+theme_bw()+
    theme(plot.title = element_text(size=12),axis.text.x= element_text(size=8),axis.text.y= element_tex
      y="Difference from National Average",
     title="NO2 Median Concentration - National vs State Average")
```

Model Building

```r
# Standardizing the values to build the model using the scale function
merge8_df$NO2 <- scale(merge8_df$NO2)
merge8_df$ozone <- scale(merge8_df$ozone)
merge8_df$SO2 <- scale(merge8_df$SO2)
merge8_df$CO <- scale(merge8_df$CO)
merge8_df$wind <- scale(merge8_df$wind)
merge8_df$temp <- scale(merge8_df$temp)
merge8_df$press <- scale(merge8_df$press)
merge8_df$RH <- scale(merge8_df$RH)
```

```r
# Creating training and test datasets by splitting them in 75:25 ratio
gp <- runif(nrow(merge8_df))
train_df <- merge8_df[gp < 0.75, ]
test_df <- merge8_df[gp >= 0.75, ]
# Printing the rows in the train and test sets
nrow(train_df)
```

```
## [1] 116464
```

```r
nrow(test_df)
```

```
## [1] 39004
```

```r
# Creating a linear regression model on the train dataset
model1 <- lm(AQI ~ NO2+ozone+SO2+PM+CO+wind+temp+press+RH+date_local
             , data=train_df )
summary(model1)
```

```
##
## Call:
## lm(formula = AQI ~ NO2 + ozone + SO2 + PM + CO + wind + temp +
##     press + RH + date_local, data = train_df)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -27.727  -2.960  -1.008   2.157  78.442
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) 92.6116876  2.6086686  35.502  < 2e-16 ***
## NO2         10.7033274  0.0161371 663.273  < 2e-16 ***
## ozone        0.1098810  0.0154328   7.120 1.09e-12 ***
## SO2         -0.0909903  0.0144301  -6.306 2.88e-10 ***
## PM           0.0275975  0.0013108  21.054  < 2e-16 ***
## CO          -0.5877026  0.0151433 -38.810  < 2e-16 ***
## wind        -0.1810164  0.0139879 -12.941  < 2e-16 ***
## temp         0.0410445  0.0147863   2.776  0.00551 **
## press       -0.0870037  0.0140760  -6.181 6.39e-10 ***
## RH          -0.6743306  0.0146805 -45.934  < 2e-16 ***
## date_local  -0.0040564  0.0001361 -29.814  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.766 on 116453 degrees of freedom
## Multiple R-squared:  0.8305, Adjusted R-squared:  0.8304
## F-statistic: 5.704e+04 on 10 and 116453 DF,  p-value: < 2.2e-16
```

```r
# Using the backward fit method to identify the features that can be excluded from the model
backwardfit.p<-ols_step_backward_p(model1,prem=.05)
backwardfit.p
```

```
## [1] "No variables have been removed from the model."
```

```r
# Finding the best model using the ols_step_best_subset function
modcompare<-ols_step_best_subset(model1)
modcompare
```

```
##                    Best Subsets Regression
## ----------------------------------------------------------------
## Model Index    Predictors
## ----------------------------------------------------------------
##      1         NO2
##      2         NO2 RH
##      3         NO2 CO RH
##      4         NO2 CO RH date_local
```

```
##      5         NO2 PM CO RH date_local
##      6         NO2 PM CO wind RH date_local
##      7         NO2 ozone PM CO wind RH date_local
##      8         NO2 ozone SO2 PM CO wind RH date_local
##      9         NO2 ozone SO2 PM CO wind press RH date_local
##     10         NO2 ozone SO2 PM CO wind temp press RH date_local
## ----------------------------------------------------------------
##
##                                                Subsets Regression Summary
## -------------------------------------------------------------------------------------------------------------
##                        Adj.        Pred
## Model    R-Square    R-Square    R-Square      C(p)          AIC            SBIC            SBC
## -------------------------------------------------------------------------------------------------------------
##   1       0.8210      0.8210       0.821    6456.8733    700507.8046    369997.0744    700536.8006
##   2       0.8256      0.8256      0.8256    3328.8701    697505.7067    366995.0240    697544.3680
##   3       0.8279      0.8279      0.8277    1772.5573    695982.6800    365472.0455    696031.0067
##   4       0.8292      0.8292       0.829     847.7867    695068.1444    364557.5582    695126.1364
##   5       0.8300      0.8300      0.8298     319.1738    694542.1261    364031.5798    694609.7834
##   6       0.8302      0.8302        0.83     150.1134    694373.3907    363862.8597    694450.7134
##   7       0.8303      0.8303      0.8301      93.0517    694316.3830    363805.8577    694403.3710
##   8       0.8304      0.8304      0.8302      53.4082    694276.7595    363766.2394    694373.4129
##   9       0.8304      0.8304      0.8302      16.7054    694240.0620    363729.5479    694346.3807
##  10       0.8305      0.8304      0.8303      11.0000    694234.3562    363723.8436    694350.3402
## -------------------------------------------------------------------------------------------------------------
## AIC: Akaike Information Criteria
##  SBIC: Sawa's Bayesian Information Criteria
##  SBC: Schwarz Bayesian Criteria
##  MSEP: Estimated error of prediction, assuming multivariate normality
##  FPE: Final Prediction Error
##  HSP: Hocking's Sp
##  APC: Amemiya Prediction Criteria
```

```r
# Building a final model based on the above findings and printing the summary
model2 <- lm(AQI ~ NO2+ozone+SO2+PM+CO+wind+temp+press+RH+date_local
             , data=train_df )
summary(model2)
```

```
##
## Call:
## lm(formula = AQI ~ NO2 + ozone + SO2 + PM + CO + wind + temp +
##     press + RH + date_local, data = train_df)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -27.727  -2.960  -1.008   2.157  78.442
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) 92.6116876  2.6086686  35.502  < 2e-16 ***
## NO2         10.7033274  0.0161371 663.273  < 2e-16 ***
## ozone        0.1098810  0.0154328   7.120 1.09e-12 ***
## SO2         -0.0909903  0.0144301  -6.306 2.88e-10 ***
## PM           0.0275975  0.0013108  21.054  < 2e-16 ***
## CO          -0.5877026  0.0151433 -38.810  < 2e-16 ***
```

```
## wind        -0.1810164  0.0139879 -12.941  < 2e-16 ***
## temp         0.0410445  0.0147863   2.776  0.00551 **
## press       -0.0870037  0.0140760  -6.181 6.39e-10 ***
## RH          -0.6743306  0.0146805 -45.934  < 2e-16 ***
## date_local  -0.0040564  0.0001361 -29.814  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.766 on 116453 degrees of freedom
## Multiple R-squared:  0.8305, Adjusted R-squared:  0.8304
## F-statistic: 5.704e+04 on 10 and 116453 DF,  p-value: < 2.2e-16
```

```r
# Adding a new column for prediction in the Training and test dataset
train_df$pred_AQI <- predict(model2 ,train_df)
test_df$pred_AQI <- predict(model2 ,test_df)
```

```r
# Function to calculate the R-squared values manually
r_squared <- function(predcol, ycol) {
  tss = sum( (ycol - mean(ycol))^2 )
  rss = sum( (predcol - ycol)^2 )
  1 - rss/tss
}
# Function to calculate the Root mean square error values manually
rmse <- function(predcol, ycol) {
  res = predcol-ycol
  sqrt(mean(res^2))
}
# Calculating the RMSE for training and test datasets
rmse_train <- rmse(train_df$pred_AQI,train_df$AQI)
sprintf("The RMSE value of Training Dataset is %s", round(rmse_train,2) )
```

```
## [1] "The RMSE value of Training Dataset is 4.77"
```

```r
rmse_test <- rmse(test_df$pred_AQI,test_df$AQI)
sprintf("The RMSE value of Training Dataset is %s", round(rmse_test,2) )
```

```
## [1] "The RMSE value of Training Dataset is 4.77"
```

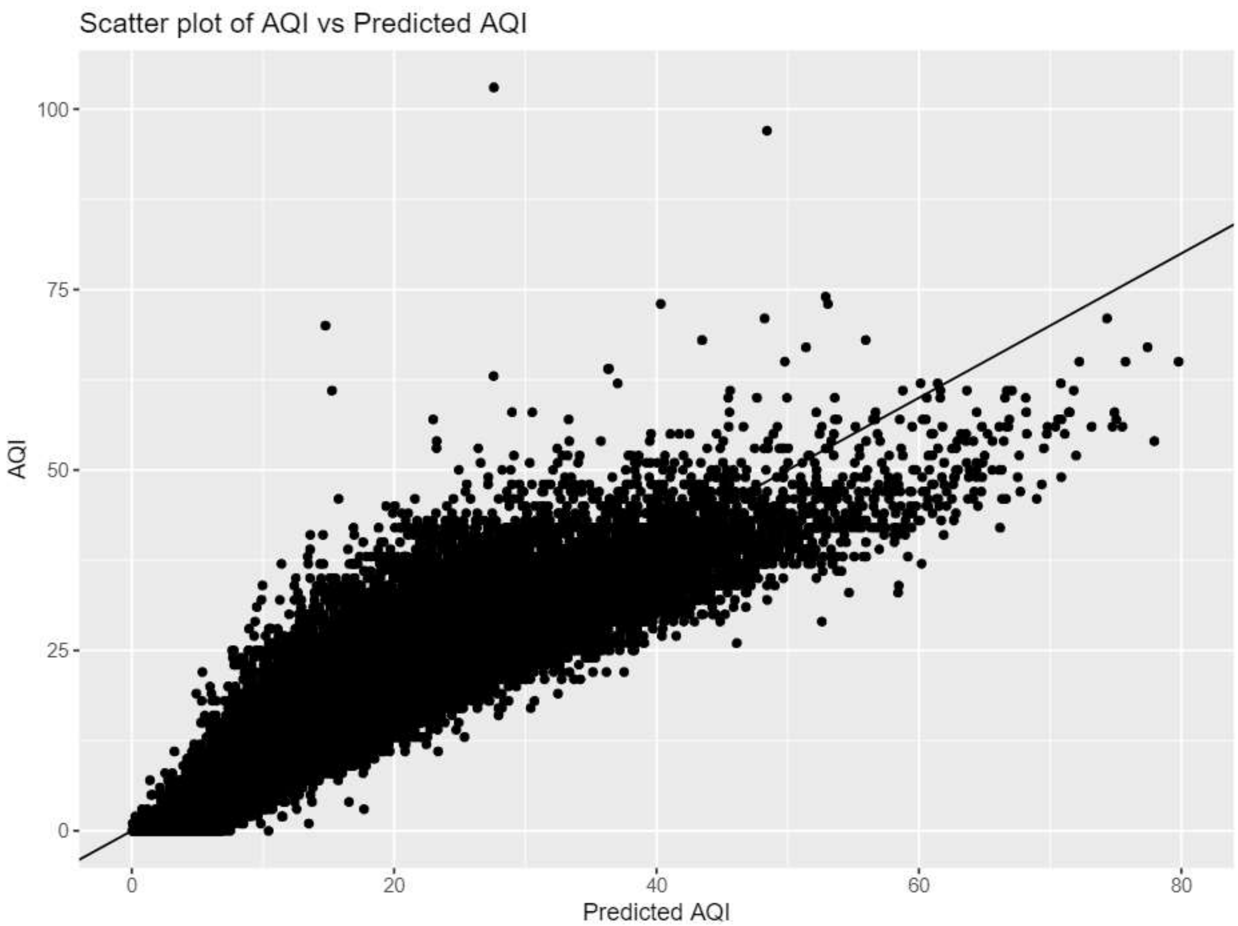
```r
# Evaluate the r-squared on both training and test data.and print them
rsq_train <- r_squared(train_df$pred_AQI,train_df$AQI)
sprintf("The R-squared value of Training Dataset is %s", round(rsq_train,2) )
```

```
## [1] "The R-squared value of Training Dataset is 0.83"
```

```r
rsq_test <- r_squared(test_df$pred_AQI,test_df$AQI)
sprintf("The R-squared value of Test Dataset is %s", round(rsq_test,2) )
```

```
## [1] "The R-squared value of Test Dataset is 0.83"
```

```r
# Plot the predictions (on the x-axis) against the outcome (AQI) on the test data
ggplot(test_df, aes(x = pred_AQI, y = AQI)) +
  geom_point() + xlim(0,80)+
  geom_abline()+labs(title="Scatter plot of AQI vs Predicted AQI",x="Predicted AQI", y="AQI")
```



Milestone4_Using_Python

February 11, 2024

[1]:
```python
# Importing required libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from sklearn.metrics import mean_squared_error
from math import sqrt

# Ignore warnings
import warnings
warnings.filterwarnings('ignore')

#For building ML model
from sklearn.model_selection import train_test_split

#Different Regressors for ML model
from sklearn import linear_model
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor,␣
↪AdaBoostRegressor
from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.gaussian_process.kernels import DotProduct, WhiteKernel , RBF
from sklearn.neighbors import KNeighborsRegressor
from sklearn.neural_network import MLPRegressor
from sklearn.svm import SVR
from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import LinearRegression

#For model evaluation
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
↪f1_score
```

[2]:
```python
# Creating Dataframe from the merged data
df_pollutants = pd.read_csv('merge10_df.csv')
df_pollutants
```

[2]:
```
        Unnamed: 0  state_code  county_code  site_num  date_local  AQI  \
0                1           1           73        23  2022-01-01    1
1                2           1           73        23  2022-01-02    4
2                3           1           73        23  2022-01-03    2
3                4           1           73        23  2022-01-04    8
4                5           1           73        23  2022-01-05   27
…              …           …            …         …           …  …
157739      157740          72           25         7  2022-09-14    5
157740      157741          72           25         7  2022-09-15    6
157741      157742          72           25         7  2022-09-16    8
157742      157743          72           25         7  2022-09-17    5
157743      157744          72           25         7  2022-09-18    2

              NO2     ozone       SO2        CO  PM       wind       temp  \
0        1.308333  0.024765 -0.191667  0.100000  14  21.525000  73.083333
1        1.954167  0.017824 -0.237500  0.145833   7  11.359091  55.141667
2        1.530000  0.031882 -0.380000  0.200000  18  49.100000  31.560000
3        5.455556  0.025353  0.555556  0.211111  18  12.266667  47.493333
4       16.493750  0.012563  0.550000  0.406667  21   5.995833  47.887500
…             …         …         …         …  ..        …
157739   3.466667  0.031882  0.345833  0.612500  18   3.729167  60.333333
157740   4.012500  0.031882  0.345833  0.600000  18   3.729167  60.333333
157741   4.575000  0.031882  0.345833  0.704167  18   3.729167  60.333333
157742   3.225000  0.031882  0.345833  0.458333  18   3.729167  60.333333
157743   1.580000  0.031882  0.345833  0.400000  18   3.729167  60.333333

              press         RH   state_name county_name   city_name  \
0        988.595833  56.191666      Alabama    Jefferson  Birmingham
1        991.304167  56.191666      Alabama    Jefferson  Birmingham
2       1002.450000  56.191666      Alabama    Jefferson  Birmingham
3       1001.420000  56.191666      Alabama    Jefferson  Birmingham
4        996.404167  56.191666      Alabama    Jefferson  Birmingham
…               …         …          …          …          …
157739   985.579167  56.191666  Puerto Rico       Caguas      Caguas
157740   985.579167  56.191666  Puerto Rico       Caguas      Caguas
157741   985.579167  56.191666  Puerto Rico       Caguas      Caguas
157742   985.579167  56.191666  Puerto Rico       Caguas      Caguas
157743   985.579167  56.191666  Puerto Rico       Caguas      Caguas

       region_name
0       South_East
1       South_East
2       South_East
3       South_East
4       South_East
…             …
157739         NaN
```

```
157740         NaN
157741         NaN
157742         NaN
157743         NaN

[157744 rows x 19 columns]
```

[3]:
```
# Defining the columns to select
y = df_pollutants['AQI']
columns_to_select = ['NO2', 'ozone', 'SO2', 'CO', 'PM', 'wind', 'temp', 'RH']
x = df_pollutants[columns_to_select]

#Split data into test and training sets
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size = 0.2,␣
 ↪random_state = 42)
X_train.shape,X_test.shape,y_train.shape,y_test.shape
```

[3]:
```
((126195, 8), (31549, 8), (126195,), (31549,))
```

[4]:
```
def model_assess(X_train, X_test, y_train, y_test, model, title ):
    """
    This function will be used to build the model. It takes train and test␣
 ↪attribues as the
    input and returns model parameters as output
    """
    model.fit(X_train, y_train)
    y_train_pred = model.predict(X_train)
    y_test_pred  = model.predict(X_test)

    train_mse = mean_squared_error(y_train, y_train_pred)
    train_r2 = r2_score(y_train, y_train_pred)
    test_mse = mean_squared_error(y_test, y_test_pred)
    test_r2 = r2_score(y_test, y_test_pred)

    r_squared = r2_score(y_test_pred,y_test)
    accuracy = round(r_squared*100,2)

    result = [str(title),test_mse, test_r2,r_squared,accuracy]

    return result
```

[5]:
```
# Creating a list of algorithms to perform the testing
algs = [LinearRegression(),
          KNeighborsRegressor(),
          RandomForestRegressor(),
          DecisionTreeRegressor(max_features = 'auto', max_depth=3,␣
 ↪random_state=42),
```

```
        GradientBoostingRegressor(n_estimators=100, max_depth=3,␣
 ↪random_state=42)]
```

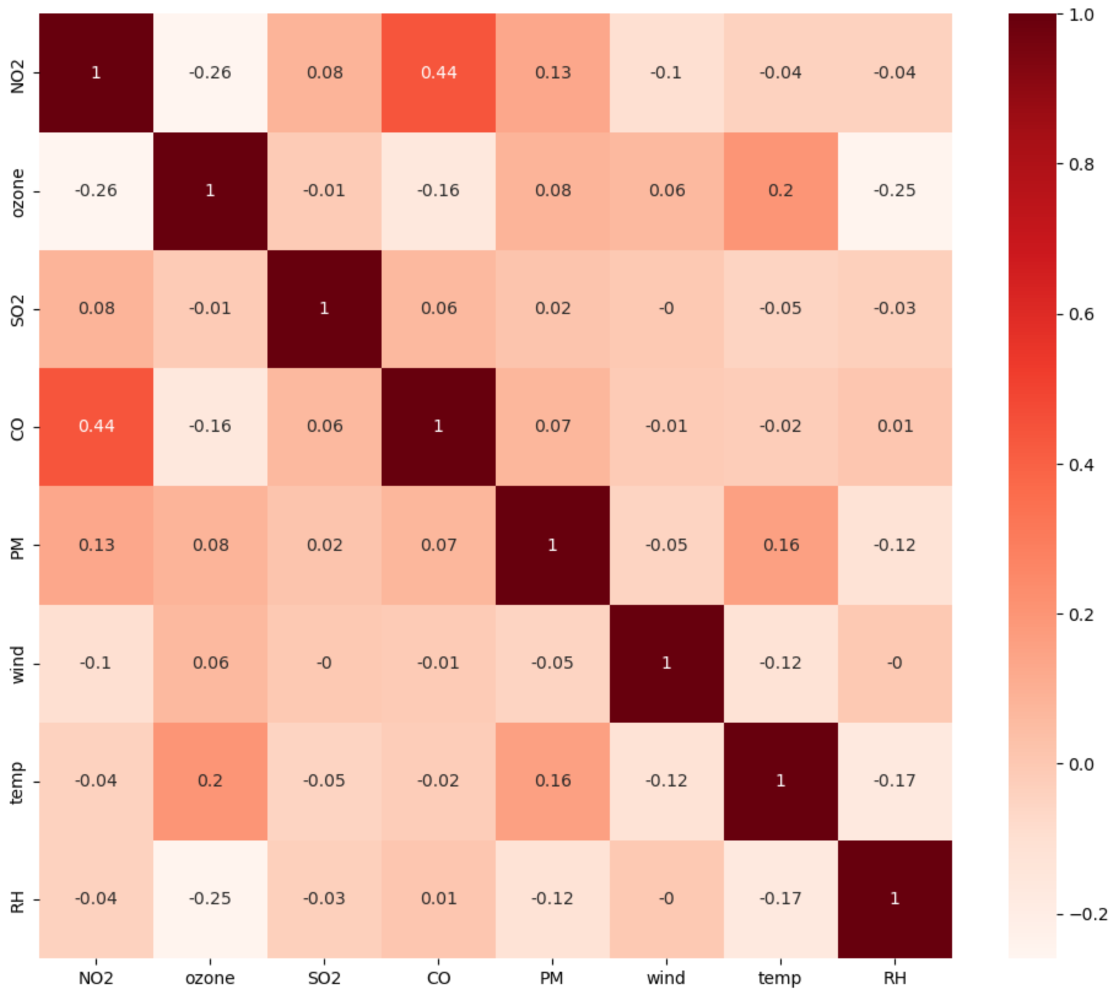
[6]:
```
# Creating a list where the results will be captured
results_list=[]
# Looping through each algorithm and build the model
for alg in algs:
   results_list.append(model_assess(X_train, X_test, y_train, y_test, alg,␣
 ↪title = alg) )
# Creating a Dataframe of the Model results
regression_results_df=pd.DataFrame(results_list, columns=['Algorithm','Test␣
 ↪MSE','Test R2','R Squared','Accuracy'])
regression_results_df
```

[6]:
```
                                          Algorithm   Test MSE   Test R2  \
0                                LinearRegression()  23.157948  0.826961
1                               KNeighborsRegressor()  21.482055  0.839484
2                             RandomForestRegressor()  17.442603  0.869667
3  DecisionTreeRegressor(max_depth=3, max_feature…  22.332338  0.833130
4         GradientBoostingRegressor(random_state=42)  17.199274  0.871485

   R Squared  Accuracy
0   0.793633     79.36
1   0.815963     81.60
2   0.854467     85.45
3   0.802207     80.22
4   0.851299     85.13
```

1 Feature Selection Using Pearson Correlation

[33]:
```
#Using Pearson Correlation
plt.figure(figsize=(12,10))
corr_cols=['NO2', 'ozone', 'SO2', 'CO', 'PM', 'wind', 'temp', 'RH']
# Creating a correlation matrix
corr_matrix = df_pollutants[corr_cols].corr().round(2)
sns.heatmap(corr_matrix, annot=True, cmap=plt.cm.Reds)
plt.show()
```



[30]:

```
#Correlation with output variable
cor_target = abs(cor["AQI"])
#Selecting highly correlated features
relevant_features = cor_target[cor_target>0.3]
relevant_features
```

[30]:

	NO2	ozone	SO2	CO	PM	wind	temp	RH
NO2	1.00	NaN	NaN	0.44	NaN	NaN	NaN	NaN
ozone	NaN	1.0	NaN	NaN	NaN	NaN	NaN	NaN
SO2	NaN	NaN	1.0	NaN	NaN	NaN	NaN	NaN
CO	0.44	NaN	NaN	1.00	NaN	NaN	NaN	NaN
PM	NaN	NaN	NaN	NaN	1.0	NaN	NaN	NaN
wind	NaN	NaN	NaN	NaN	NaN	1.0	NaN	NaN
temp	NaN	NaN	NaN	NaN	NaN	NaN	1.0	NaN
RH	NaN	NaN	NaN	NaN	NaN	NaN	NaN	1.0

[10]:
```
# The features NO2,CO are highly correlated with the output variable AQI,Hence␣
↪we will drop all other features apart from these

y = df_pollutants['AQI']
columns_to_select = ['NO2', 'CO']
x = df_pollutants[columns_to_select]

#Split data into test and training sets
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size = 0.2,␣
↪random_state = 42)
X_train.shape,X_test.shape,y_train.shape,y_test.shape
```

[10]:
```
((126195, 2), (31549, 2), (126195,), (31549,))
```

[11]:
```
# Creating a list where the results will be captured
reg_results_list1=[]
# Looping through each algorithm and build the model
for model in algs:
   reg_results_list1.append(model_assess(X_train, X_test, y_train, y_test,␣
↪model, title = model) )
```

[12]:
```
# Creating a Dataframe of the Model results
df_reg=pd.DataFrame(reg_results_list1, columns=['Algorithm','Test MSE','Test␣
↪R2','R Squared','Accuracy'])
df_reg
```

[12]:
```
                                          Algorithm   Test MSE   Test R2  \
0                                LinearRegression()  23.928997  0.821200
1                             KNeighborsRegressor()  22.712246  0.830292
2                           RandomForestRegressor()  22.510027  0.831803
3  DecisionTreeRegressor(max_depth=3, max_feature…  22.332338  0.833130
4         GradientBoostingRegressor(random_state=42)  18.987270  0.858125

   R Squared  Accuracy
0   0.785549     78.55
1   0.811406     81.14
2   0.814045     81.40
3   0.802207     80.22
4   0.835796     83.58
```

2 Feature Reduction by PCA

[13]:
```
# Selecting list of columns to build the model
columns_to_select = ['NO2', 'ozone', 'SO2', 'CO', 'PM', 'wind', 'temp', 'RH']
x = df_pollutants[columns_to_select]
```

[14]:
```
#Split data into test and training sets
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size = 0.2,␣
 ↪random_state = 42)
X_train.shape,X_test.shape,y_train.shape,y_test.shape
```

[14]: ((126195, 8), (31549, 8), (126195,), (31549,))

[15]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
# Standardize feature matrix
scaler_pca = StandardScaler()
 # Standardize the feature matrix
features = scaler_pca.fit_transform(X_train)
# Create a PCA that will retain 80 of variance
pca = PCA(n_components=0.80, whiten=True)
# Conduct PCA
features_pca = pca.fit_transform(features)
# Show results
print("Original number of features:", features.shape[1])
print("Reduced number of features:", features_pca.shape[1])
```

```
Original number of features: 8
Reduced number of features: 6
```

[17]:
```
# Standardize the feature matrix
features_test = scaler_pca.transform(X_test)
# Conduct PCA
features_pca_test = pca.transform(features_test)
# Show results
print("Original number of features:", features_test.shape[1])
print("Reduced number of features:", features_pca_test.shape[1])
X_test.shape[1]
```

```
Original number of features: 8
Reduced number of features: 6
```

[17]: 8

[18]:
```
# Create linear regression object
reg_pca = linear_model.LinearRegression()
reg_pca.fit(features_pca,y_train)

# Predicting the output

y_pred_pca = reg_pca.predict(features_pca_test)
y_pred_pca.shape
```

[18]: (31549,)

[19]:
```
from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score

# Mean Squared Error
mse = mean_squared_error(y_pred_pca,y_test)
# Root Mean Squared Error
rmse = np.sqrt(mse)
# Mean Absolute error
mae = mean_absolute_error(y_pred_pca,y_test)
# R2
r_squared = r2_score(y_pred_pca,y_test)
r_squared
print("Mean Squared Error:", mse)
print("Root Mean Squared Error:", rmse)
print("R-squared:", r_squared)
print("Accuracy",round(r_squared*100,2))
```

```
Mean Squared Error: 67.58882611476984
Root Mean Squared Error: 8.22124237051614
R-squared: 0.2458928916249895
Accuracy 24.59
```

[26]:
```
def model_assess_pca(features_pca, features_pca_test, y_train, y_test, model,␣
↪title = model):
    """
    This function will be used to build the model. It takes train and test␣
↪attribues as the
    input and returns model parameters as output
    """
    model.fit(features_pca, y_train)
    y_test_pred_pca = model.predict(features_pca_test)
    test_mse = mean_squared_error(y_test, y_test_pred_pca)
    test_r2 = r2_score(y_test, y_test_pred_pca)
    r_squared = r2_score(y_test_pred_pca,y_test)
    accuracy = round(r_squared*100,2)
    result = [str(title),test_mse, test_r2,r_squared,accuracy]
    return result
```

[27]:
```
# Creating a list where the results will be captured
results_list_pca=[]
# Looping through each algorithm and build the model
for model in algs:
   results_list_pca.append(model_assess_pca(features_pca, features_pca_test,␣
↪y_train, y_test, model, title = model) )
```

[28]:
```
# Create DataFrame to captue the model results
columns = ['Algorithm', 'Test MSE', 'Test R2', 'R Squared', 'Accuracy']
df_regression = pd.DataFrame(results_list_pca, columns=columns)
```

```
df_regression
```

[28]:

```
                                            Algorithm   Test MSE   Test R2  \
0                                  LinearRegression()  67.588826  0.494969
1                               KNeighborsRegressor()  27.964259  0.791048
2                             RandomForestRegressor()  26.463282  0.802263
3  DecisionTreeRegressor(max_depth=3, max_feature…  61.658604  0.539280
4         GradientBoostingRegressor(random_state=42)  38.324697  0.713634

   R Squared  Accuracy
0   0.245893     24.59
1   0.750876     75.09
2   0.753679     75.37
3   0.147110     14.71
4   0.568430     56.84
```