POLITECNICO DI BARI

BIG DATA

Second Project
Machine Learning with Spark

Professor:
Prof. Ing. Simona COLUCCI

Students:
Benedetti Giuseppe
Fischetti Luca
Minervini Davide

Group: 23

Summary

1. Introduction and Objectives

This project's objective is to create a Machine Learning model to forecast the future air quality levels using historic air pollution data. In order to do so, we are basing our analysis on the following dataset that reports Air Quality Indexes from 2000 to 2016 in the U.S.

Link: https://www.kaggle.com/sogun3/uspollution

Each record of the dataset reports the AQI values registered by the EPA (Environmental Protection Agency) for each pollutant (NO_2, O_3, SO_2, CO) from a particular site located in the U.S. on a specific date.

The fields we used to reach our purpose are the following:

Field	***Description***
State	State where the site of measuring is located
State Code	State's numeric identifier
County	County where the site of measuring is located
County Code	County's numeric identifier
City	City where the site is located
Site Num	Site's numeric identifier
Date Local	Date of measuring
NO2 AQI	Air Quality Index for Nitrogen Dioxide (NO_2)
O3 AQI	Air Quality Index for Ozone (O_3)
SO2 AQI	Air Quality Index for Sulphur Dioxide (SO_2)
CO AQI	Air Quality Index for Carbon Monoxide (CO)

The Air Quality Index for each pollutant is an integer numeric value which represents how polluted the air is. Public health risks increase as the AQI rises. Different countries have their own air quality indexes, corresponding to different national air quality standards.
For more information on this topic:
https://en.wikipedia.org/wiki/Air_quality_index

For this project we needed to import different libraries to execute every task, such as:

- *Pyspark.sql* to submit query instructions on pyspark dataframes;

- *Pyspark.ml* to execute machine learning algorithms;
- *Time* to calculate execution time;
- *Statsmodels.tsa.seasonal.seasonal_decompose* to analyse the temporal trend of different AQIs;
- *Matplotlib.pyplot* to make plotting graphs easier and more effective;
- *Pandas* to be able to use the previous library;
- *Seaborn* for statistical data visualization;
- *Numpy,* which is a Python library used in domain of linear algebra, Fourier transform, and matrices;

In order to use the last four libraries, it is necessary to install Anaconda which is popular for its tools used in data science and machine learning.
We used machine learning algorithms to predict average AQI values for each pollutant. These types of algorithms need to handle consistent data within the dataset, in order to make the prediction as accurate as possible. To do so, we decided to consider only data from 2010 to 2016.

2. Data Analysis

2.1 Data Cleaning & Pre-processing

With the following code, we did a cleaning of the data from the original dataset to make it more suitable for this project. First of all, we selected only the fields we really needed (those reported in the table of the 1st Chapter) and rearranged their position for an easier consultation. Starting from the “Date Local” field, we created three new sub-fields (“Year”, “Month” and “Day”, keeping the original field) to ease querying. Furthermore, we decided to delete records with missing data using the dropna() method on the dataframe. Finally, we considered only stations that actually measured air quality levels with continuity in each year of the considered period (2010-2016) to have a more coherent forecast. Then we wrote a new CSV file called “Preprocessed_Dataset” containing the polished dataframe.

```
from pyspark.sql import *
from pyspark.sql.functions import substring
from pyspark.sql.functions import countDistinct
from pyspark.sql import functions as F
import time


start_time=time.time()
spark = SparkSession.builder \
.master("local") \
.appName("Job1(2)") \
.config("spark.some.config.option", "some-value") \
.getOrCreate()

df = spark.read.load("/home/giuseppe/Second_Project/Pollution2000_2016.csv" , format="csv", sep=",", header="true")

#Data Cleaning

df=df.select("State","State Code","County","County Code","City","Site Num","Date Local","NO2 AQI","O3 AQI","SO2 AQI","CO AQI")
df=df.withColumn("Year",substring("Date Local",0,4))
df=df.withColumn("Month",substring("Date Local",6,2))
df=df.withColumn("Day",substring("Date Local",9,2))
df=df.dropna()

df=df.filter(df["Year"]>2009)

df1=df
df1=df1.select("State","County","City","Site Num","Year").groupBy("State","County","City","Site Num").agg(countDistinct("Year").alias("Years"))
df1=df1.filter(df1["Years"]==7)
df1=df1.withColumnRenamed("City","City1")
df1=df1.withColumnRenamed("Years","Years1")

df=df.join(df1, on=["Site Num","State","County"], how="left_semi")

df.coalesce(1).write.option("header","true").format("csv").save("/home/giuseppe/Second_Project/Preprocessed_Dataset")

end_time=time.time()
print("Tempo impiegato:",end_time-start_time,"s")
spark.stop();
```

2.2 Data Exploration

2.2.1 Correlation Matrix

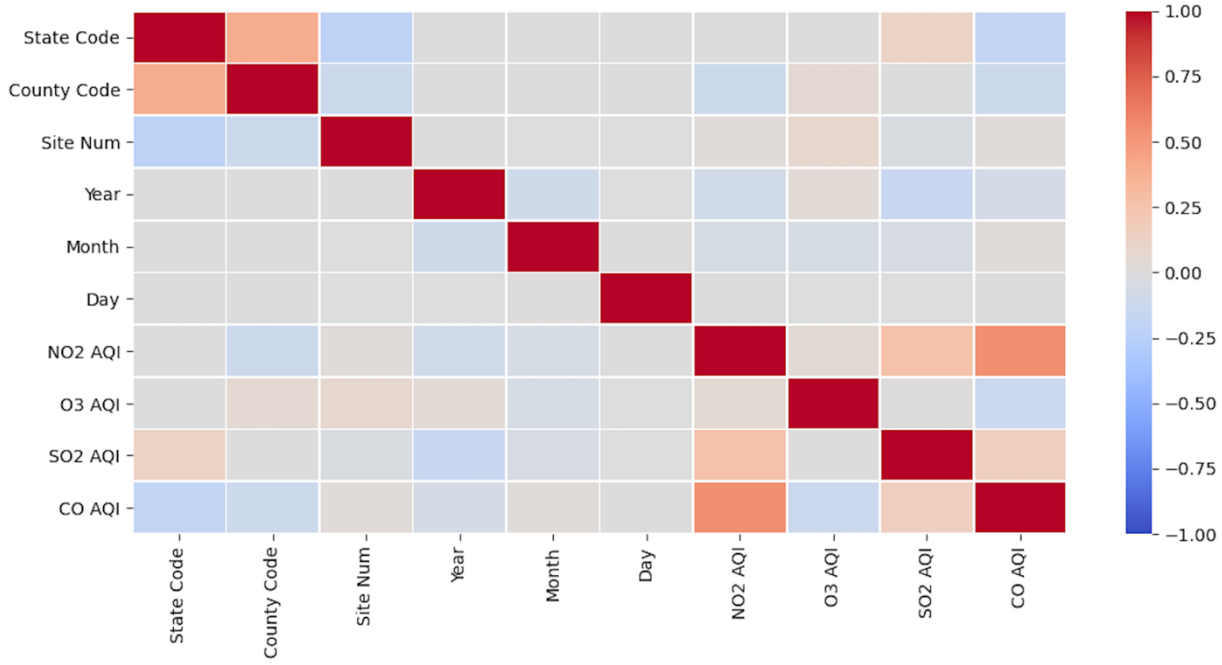
The correlation matrix is a square table containing the correlation indices between two or more variables.
Bivariate correlation coefficients are one of the most widely used statistical indices for assessing the relationship between variables. Within a research project, several correlation coefficients are often calculated and summarised in a single table, the correlation matrix.

To analyse this table in more detail, we can divide it into three parts: a main diagonal and two triangles, one at the bottom left and one at the top right.
The correlation indices on the diagonal running from the upper left-hand corner to the lower right-hand corner (i.e. on the main diagonal of the correlation matrix) are all equal to 1. These 1's on the main diagonal simply indicate that the correlation of a variable with itself is, by definition, always equal to 1.

As we can see in a correlation matrix the correlation index between each pair of variables always appears twice.

The values present in the lower left triangle of the correlation matrix are in fact the same as those present in the cells of the upper right triangle of the same matrix. This is because the correlation index is a symmetrical statistical measure that does not take into account the order in which the variables are inserted into the formula.

```
import time
import pandas as pd
import seaborn as sb
import matplotlib.pyplot as plt


start_time=time.time()

df= pd.read_csv('/home/giuseppe/Second_Project/Preprocessed_Dataset.csv')

corr = df.corr();
sb.heatmap(corr,xticklabels=corr.columns,yticklabels=corr.columns,vmax=1,vmin=-1,center=0,cmap='coolwarm',linewidths=0.5)
plt.show()

end_time=time.time()
print("Tempo impiegato:",end_time-start_time,"s")
```

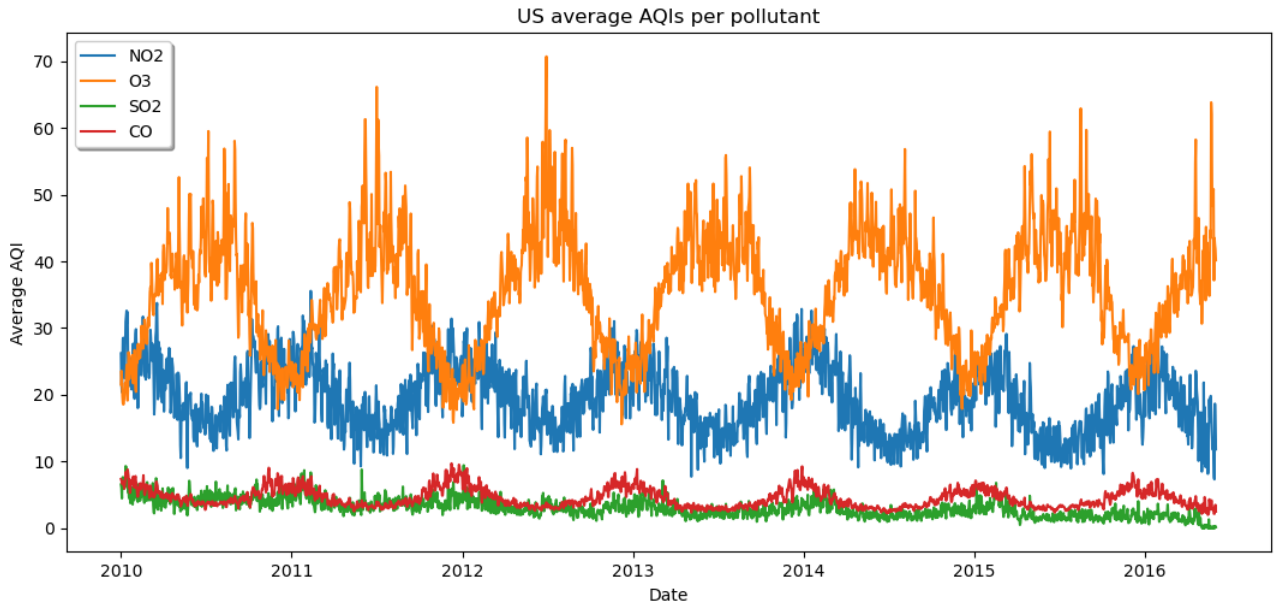


Looking at the correlation matrix, we notice that a strong correlation exists between the pair County Code - State Code (as expected) and NO_2 AQI – CO AQI. Also, a weak correlation between SO_2 AQI and NO_2 AQI can be noticed.
With these observations, we continue our data exploration keeping all the fields in the current dataset.

2.2.2: Average AQI Levels per pollutant

To represent the average variations of AQIs for each pollutant through the years, we built a plot with the following code:

```
df1 = spark.read.load("/home/giuseppe/Second_Project/Preprocessed_Dataset.csv" , format="csv", sep=",", header="true")

df1=df1.select((F.to_date("Date Local","yyyy-MM-dd").alias("Date")),"NO2 AQI","O3 AQI","SO2 AQI","CO AQI")

df1=df1.orderBy("Date")
df1=df1.groupBy("Date").agg(F.avg(F.col("NO2 AQI")).alias("NO2"),F.avg(F.col("O3 AQI")).alias("O3"),F.avg(F.col("SO2 AQI")).alias("SO2"),F.avg(F.col("CO AQI")).alias("CO"))
df1=df1.toPandas()

df1.plot(x='Date')
plt.title('US average AQIs per pollutant', fontdict=None, loc='center', pad=None)
plt.xlabel('Date')
plt.ylabel('Average AQI')
plt.legend(loc='upper left', ncol=1, fancybox=True, shadow=True)
plt.show()
```

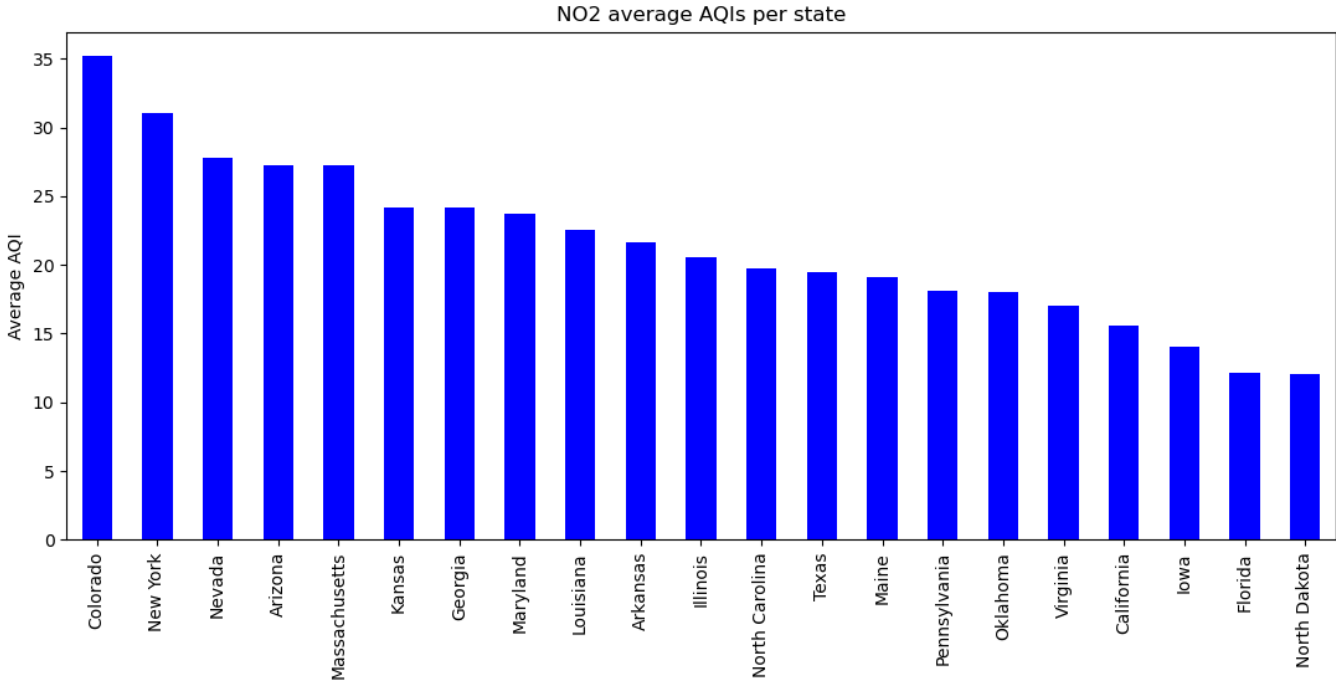


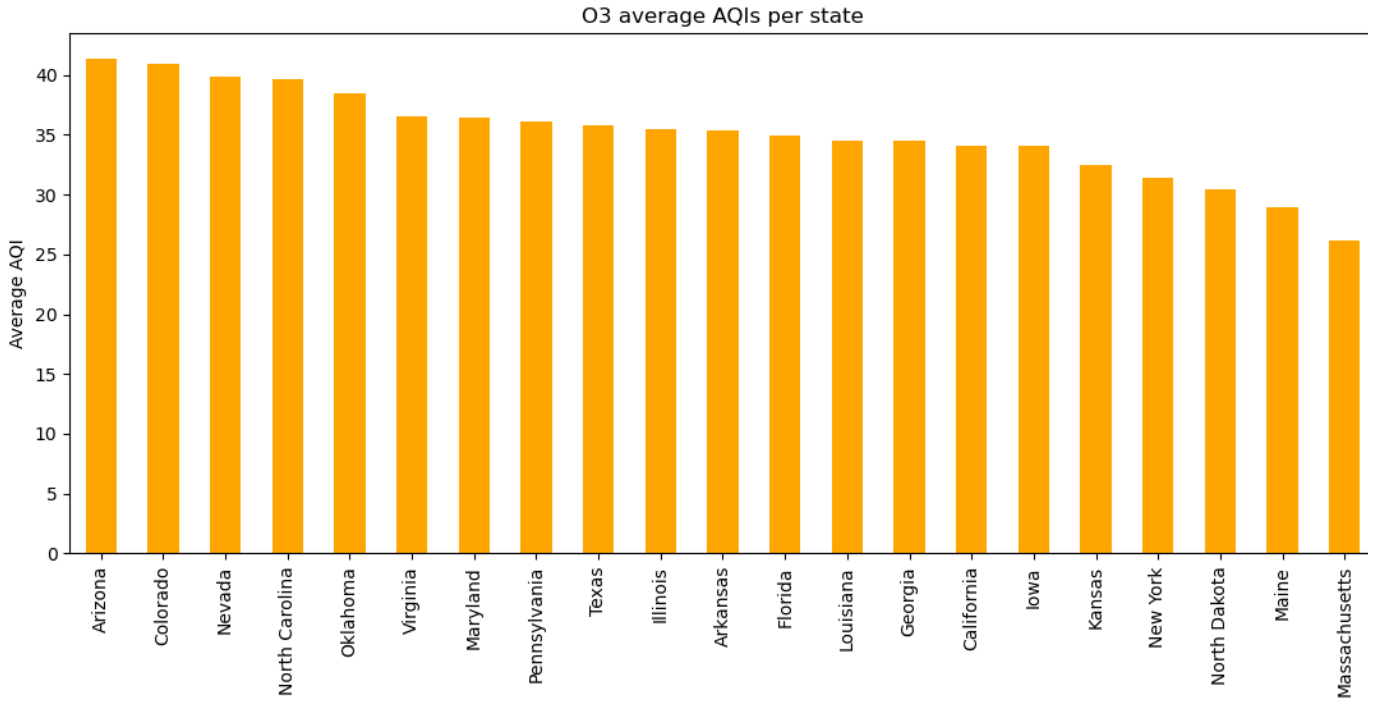
Looking at this plot, it's clear that the levels of each pollutant follow periodic patterns occurring each year. Specifically, the average O_3 AQI levels are clearly higher near the summer season, so they follow an opposite trend compared to the rest of pollutants. The difference in average NO_2 and O_3 AQIs value when compared to SO_2 and CO ones depends on the different scale of the measurements.

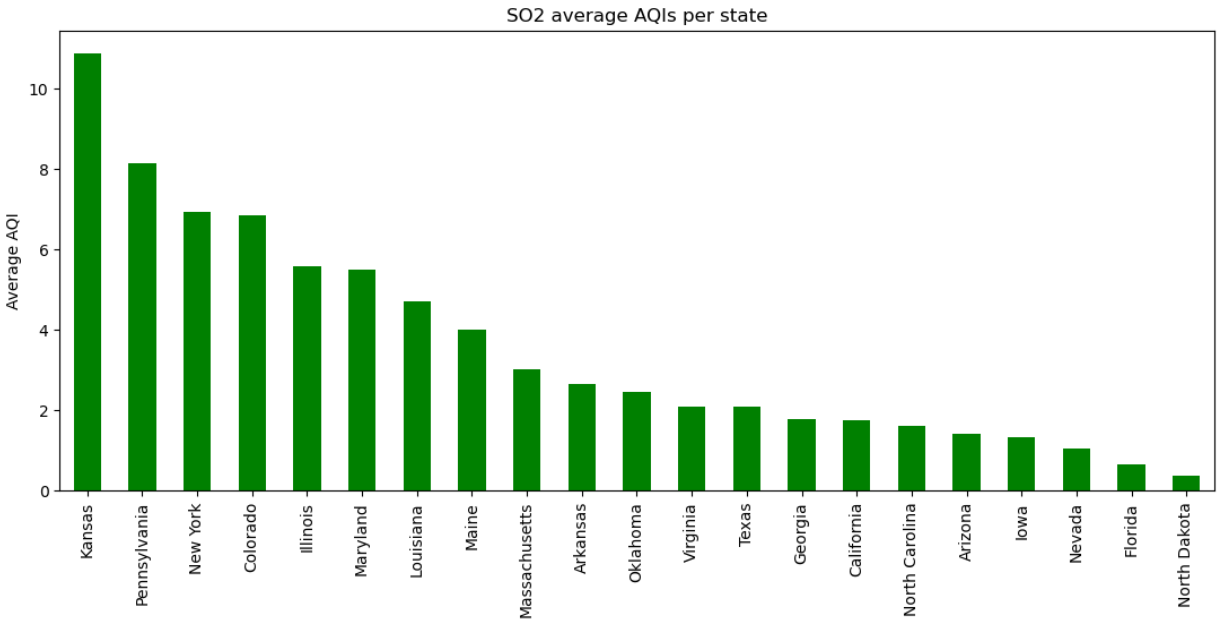
2.2.3: Average AQIs per State

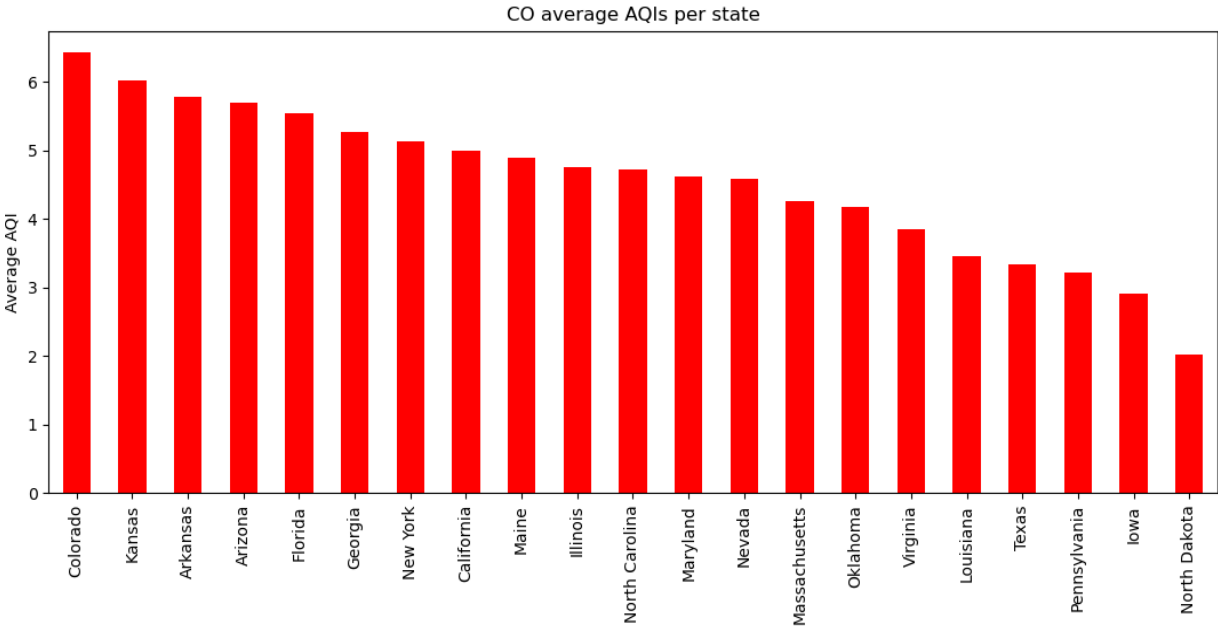
In this section, we grouped all records of the preprocessed dataset by State to do an average on AQIs measured by all stations in that State for each pollutant.

```
df = spark.read.load("/home/giuseppe/Second_Project/Preprocessed_Dataset.csv" , format="csv", sep=",", header="true")
df=df.select((F.to_date("Date Local","yyyy-MM-dd").alias("Date")),"State","NO2 AQI","O3 AQI","SO2 AQI","CO AQI")
df=df.groupBy("State").agg(F.avg(F.col("NO2 AQI")).alias("NO2"),F.avg(F.col("O3 AQI")).alias("O3"),
                           F.avg(F.col("SO2 AQI")).alias("SO2"),F.avg(F.col("CO AQI")).alias("CO"))

def AverageAQIsperState(pollutant,fig_number,color,dataframe):
 df=dataframe.orderBy(pollutant, ascending=False)
 df_pandas=df.toPandas()

 df_pandas.plot(x='State',y=pollutant,kind='bar',legend=None,color=color)
 plt.title(pollutant+' average AQIs per state', fontdict=None, loc='center', pad=None)
 plt.ylabel('Average AQI')
 plt.figure(fig_number)

AverageAQIsperState("NO2",1,"blue",df)
AverageAQIsperState("O3",2,"orange",df)
AverageAQIsperState("SO2",3,"green",df)
AverageAQIsperState("CO",4,"red",df)

plt.show()

end_time=time.time()
print("Tempo impiegato:",end_time-start_time,"s")
spark.stop();
```


NO2 average AQIs per state
Average AQI
0
5
10
15
20
25
30
35
Colorado
New York
Nevada
Arizona
Massachusetts
Kansas
Georgia
Maryland
Louisiana
Arkansas
Illinois
North Carolina
Texas
Maine
Pennsylvania
Oklahoma
Virginia
California
Iowa
Florida
North Dakota


O3 average AQIs per state
Average AQI
0
5
10
15
20
25
30
35
40
Arizona
Colorado
Nevada
North Carolina
Oklahoma
Virginia
Maryland
Pennsylvania
Texas
Illinois
Arkansas
Florida
Louisiana
Georgia
California
Iowa
Kansas
New York
North Dakota
Maine
Massachusetts





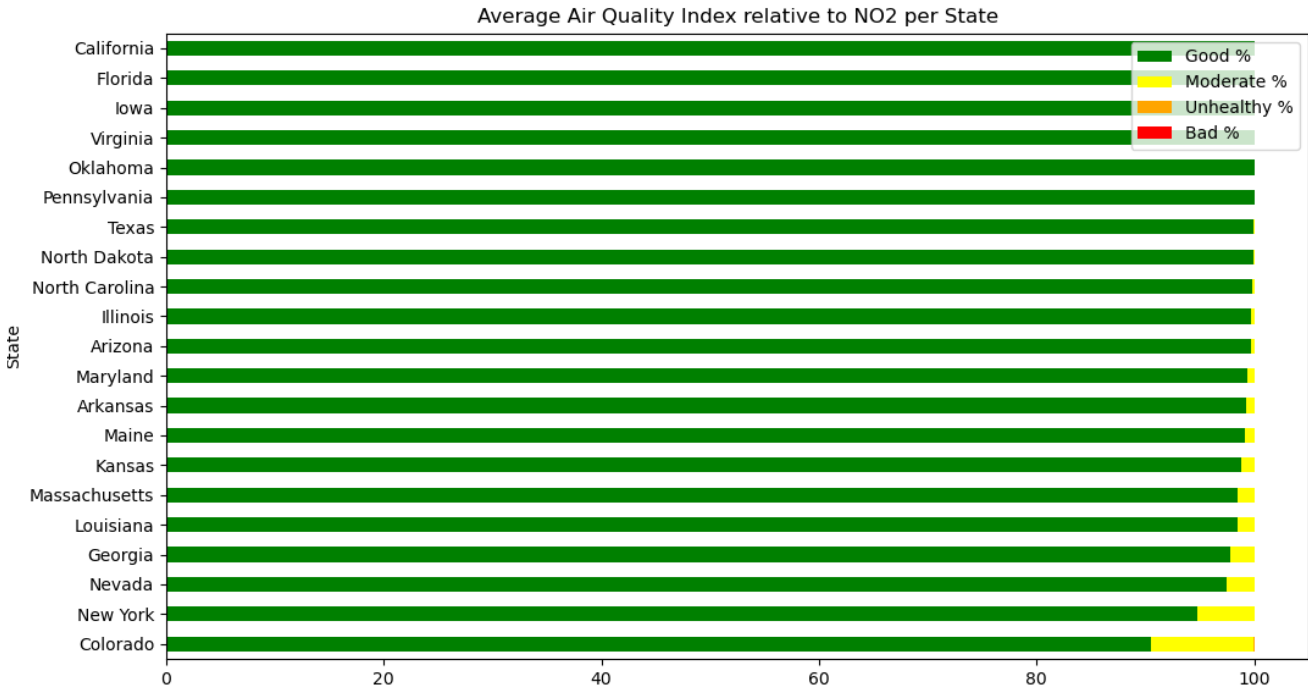
As we can see the SO_2 levels are strongly dependent on the State considered. This means that it may be linked to the geographical position, the industrialization index or to the State's population density. Instead, O_3 levels are pretty much constant on all U.S. areas and the CO emissions are concentrated in the middle-low area of the US.

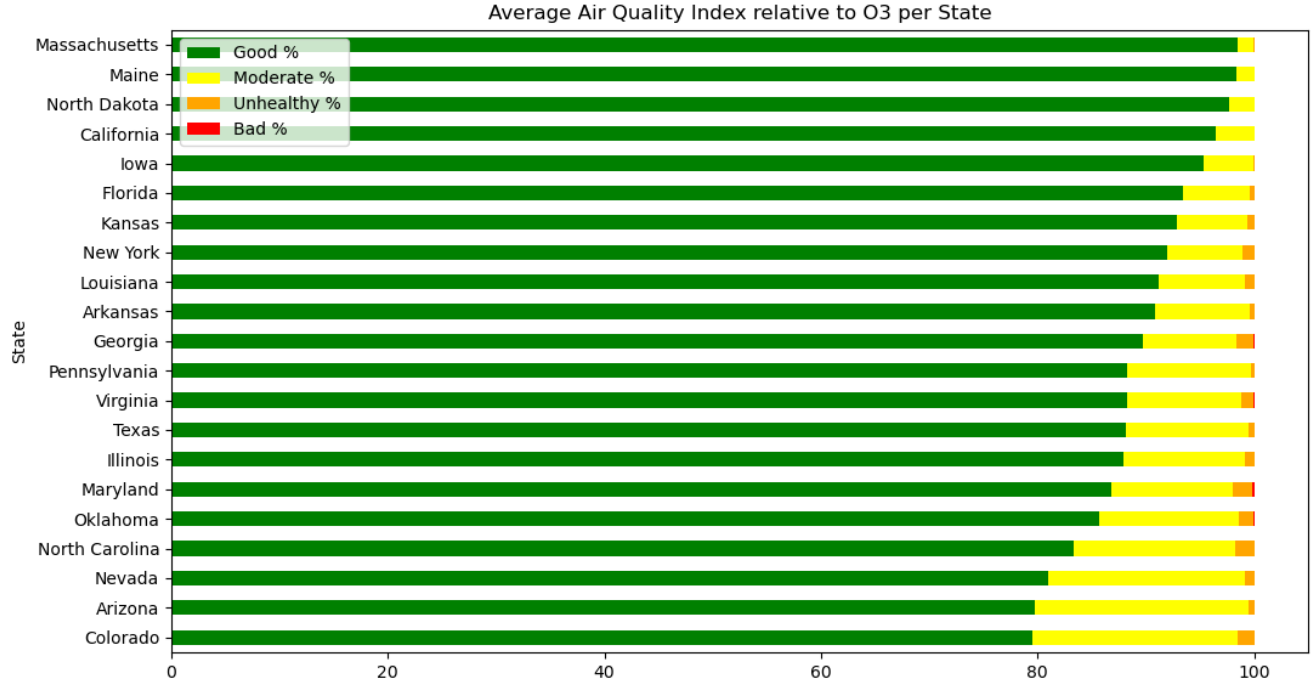
2.2.4: AQIs Evaluation

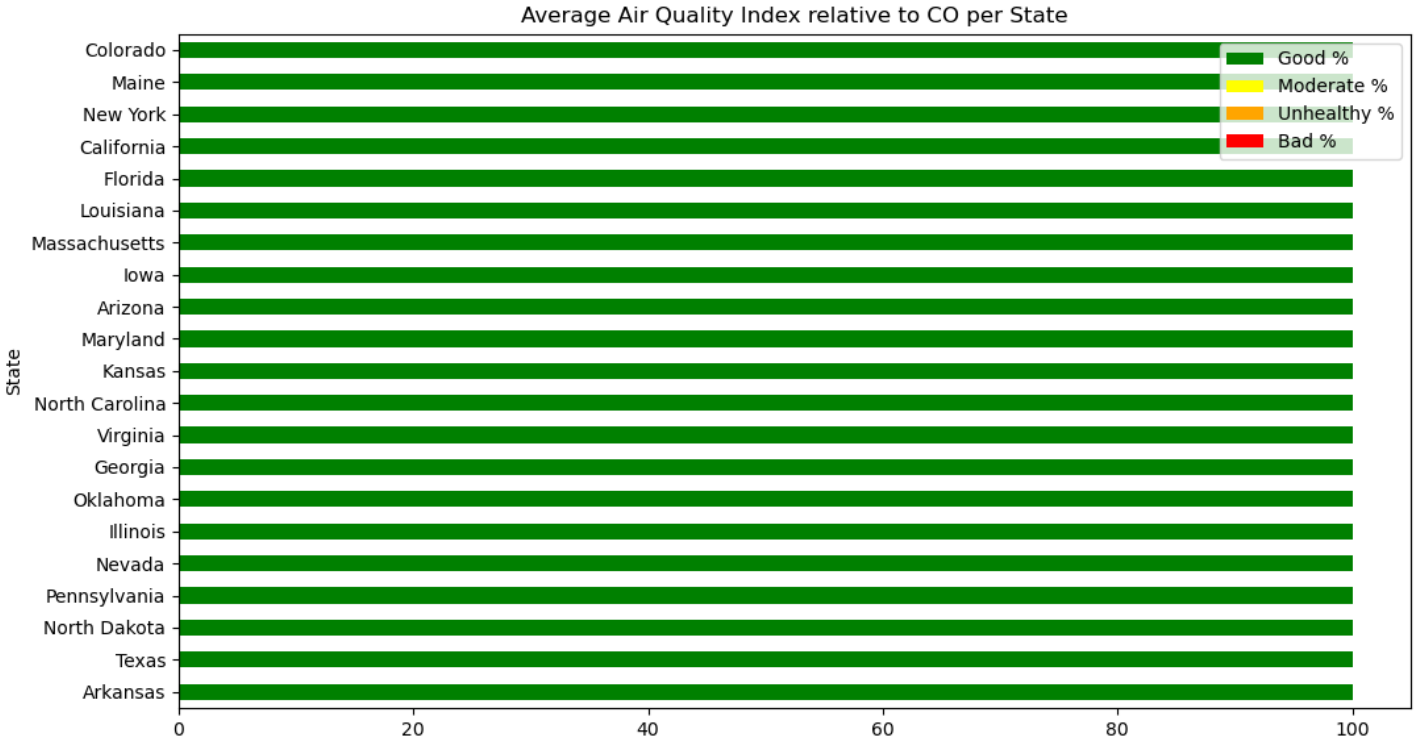
We decided to perform an analysis of the air quality indexes in order to evaluate the percentage of days with healthy or harmful levels for each pollutant in every State considered. We relied on a widely accepted standard to establish the right AQIs intervals to assess a particular health concern:

- 0 to 50 --> Good
- 51 to 100 --> Moderate
- 101 to 150 --> Unhealthy
- 151 or higher --> Bad

```
def plot_data (pollutant,fig_number):
 df_pyspark = spark.read.load("/home/giuseppe/Second_Project/Preprocessed_Dataset.csv" , format="csv", sep=",", header="true")
 df_pyspark=df_pyspark.select((F.to_date("Date Local","yyyy-MM-dd").alias("Date")),"State",pollutant+" AQI")
 df_pyspark1=df_pyspark.groupBy("State","Date").agg(F.avg(F.col(pollutant+" AQI")).alias(pollutant))
 df_pyspark2=df_pyspark1
 df_pyspark3=df_pyspark1
 df_pyspark4=df_pyspark1

 df_pyspark1=df_pyspark1.filter(df_pyspark1[pollutant]<50)
 df_pyspark1=df_pyspark1.groupBy("State").count()
 df_pyspark1=df_pyspark1.select(F.col("State"),F.col("count").alias("Good"))

 df_pyspark2=df_pyspark2.filter((df_pyspark2[pollutant]>=50) & (df_pyspark2[pollutant]<100))
 df_pyspark2=df_pyspark2.groupBy("State").count()
 df_pyspark2=df_pyspark2.select(F.col("State"),F.col("count").alias("Moderate"))

 df_pyspark3=df_pyspark3.filter((df_pyspark3[pollutant]>=100) & (df_pyspark3[pollutant]<150))
 df_pyspark3=df_pyspark3.groupBy("State").count()
 df_pyspark3=df_pyspark3.select(F.col("State"),F.col("count").alias("Unhealthy"))

 df_pyspark4=df_pyspark4.filter(df_pyspark4[pollutant]>=150)
 df_pyspark4=df_pyspark4.groupBy("State").count()
 df_pyspark4=df_pyspark4.select(F.col("State"),F.col("count").alias("Bad"))


 df_final=df_pyspark1.join(df_pyspark2,on="State", how="left_outer")
 df_final=df_final.join(df_pyspark3,on="State", how="left_outer")
 df_final=df_final.join(df_pyspark4, on="State" , how="left_outer")

 df_final=df_final.select("State",coalesce(df_final["Good"],F.lit(0)).alias("Good"),
 coalesce(df_final["Moderate"],F.lit(0)).alias("Moderate"),
 coalesce(df_final["Unhealthy"],F.lit(0)).alias("Unhealthy"),coalesce(df_final["Bad"], F.lit(0)).alias("Bad"))
 df_final=df_final.withColumn("Total Days",df_final["Good"]+df_final["Moderate"]+df_final["Unhealthy"]+df_final["Bad"])
 df_final=df_final.withColumn("Good %",df_final["Good"]*100/df_final["Total Days"])
 df_final=df_final.withColumn("Moderate %",df_final["Moderate"]*100/df_final["Total Days"])
 df_final=df_final.withColumn("Unhealthy %",df_final["Unhealthy"]*100/df_final["Total Days"])
 df_final=df_final.withColumn("Bad %",df_final["Bad"]*100/df_final["Total Days"])

 df_final=df_final.select("State","Good %","Moderate %","Unhealthy %","Bad %")
 df_final=df_final.orderBy("Good %")

 df_final_pandas=df_final.toPandas()
 df_final_pandas.plot.barh(x="State",stacked=True, color=["green","yellow","orange","red"])

 plt.title("Average Air Quality Index relative to "+ pollutant+" per State", fontdict=None, loc='center', pad=None)
 plt.figure(fig_number)

plot_data("NO2",1)
plot_data("O3",2)
plot_data("SO2",3)
plot_data("CO",4)
plt.show()

end_time=time.time()
print("Tempo impiegato:",end_time-start_time,"s")
spark.stop();
```


Average Air Quality Index relative to NO2 per State
State
California
Florida
Iowa
Virginia
Oklahoma
Pennsylvania
Texas
North Dakota
North Carolina
Illinois
Arizona
Maryland
Arkansas
Maine
Kansas
Massachusetts
Louisiana
Georgia
Nevada
New York
Colorado
Good %
Moderate %
Unhealthy %
Bad %
0
20
40
60
80
100


Average Air Quality Index relative to O3 per State
State
Massachusetts
Maine
North Dakota
California
Iowa
Florida
Kansas
New York
Louisiana
Arkansas
Georgia
Pennsylvania
Virginia
Texas
Illinois
Maryland
Oklahoma
North Carolina
Nevada
Arizona
Colorado
Good %
Moderate %
Unhealthy %
Bad %
0
20
40
60
80
100



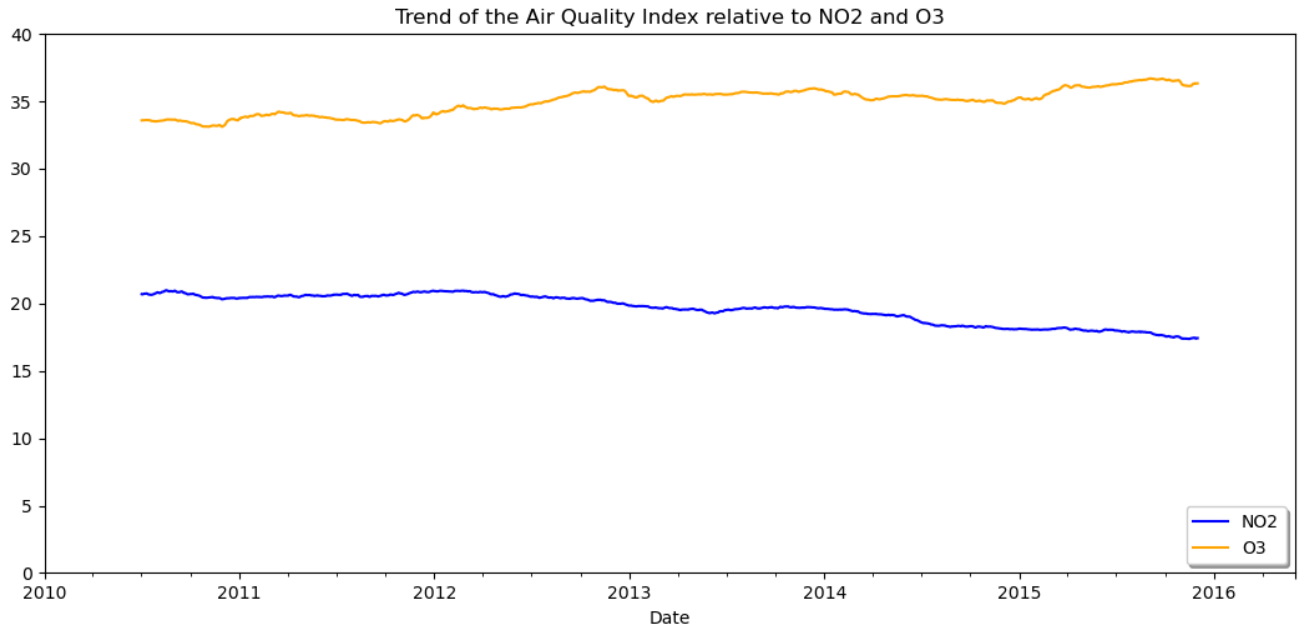
While NO_2, SO_2 and CO usually maintain good levels during the period considered, the same can't be said about O_3, which occasionally reaches worrying levels in different States like Maryland, Georgia etc.

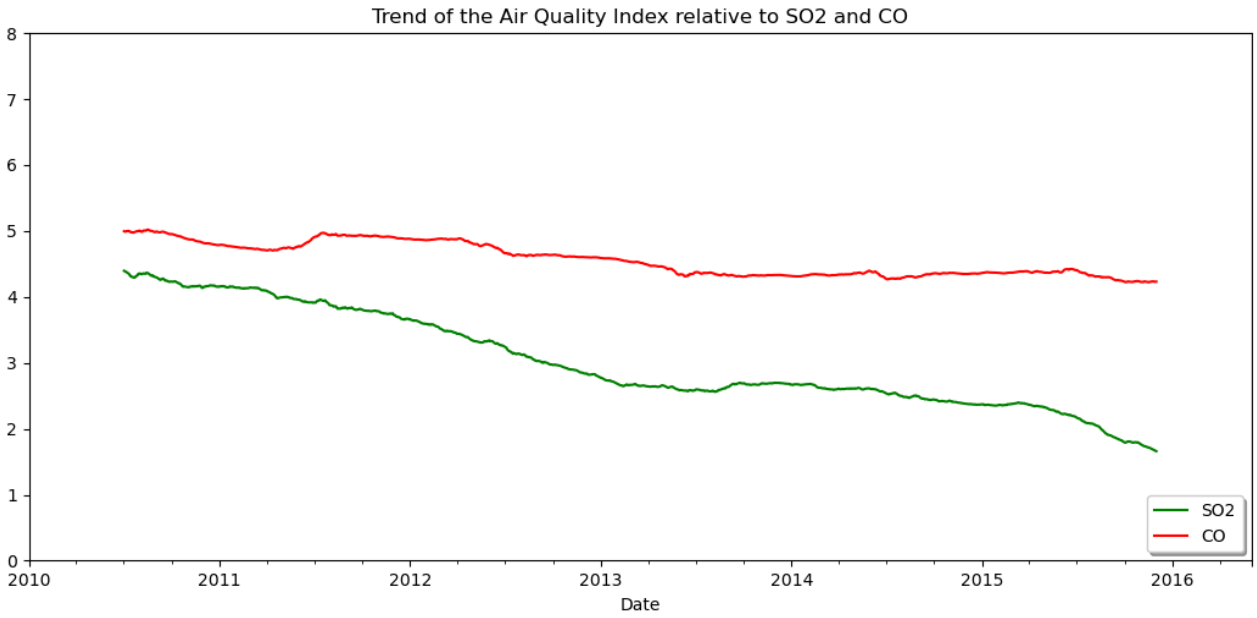
2.2.5: Trend Evaluation

To make a consistent forecast, we analyzed the trends of AQIs for each pollutant to evaluate their general behavior within the period of time considered and also to see if there's a predictable pattern that the forecast should follow.

```
def plot_data(pollutant1,pollutant2,fig_number,ylimit,color1,color2):
 df = spark.read.load("/home/giuseppe/Second_Project/Preprocessed_Dataset.csv" , format="csv", sep=",", header="true")

 df=df.select((F.to_date("Date Local","yyyy-MM-dd").alias("Date")),pollutant1+" AQI",pollutant2+" AQI")
 df=df.orderBy("Date")
 df=df.groupBy("Date").agg(F.avg(F.col(pollutant1+" AQI")).alias(pollutant1),F.avg(F.col(pollutant2+" AQI")).alias(pollutant2))
 df=df.toPandas()
 df.index = pd.to_datetime(df["Date"])
 plt.figure(fig_number)
 df1 = seasonal_decompose(df[pollutant1], model='additive', period=365)
 df1.trend.plot(x='Date', ylim=(0,ylimit),label=pollutant1,color=color1)
 df2 = seasonal_decompose(df[pollutant2], model='additive', period=365)
 df2.trend.plot(x='Date', ylim=(0,ylimit),label=pollutant2,color=color2)
 plt.title("Trend of the Air Quality Index relative to "+ pollutant1+ " and "+pollutant2, fontdict=None, loc='center', pad=None)
 plt.legend(loc='lower right', ncol=1, fancybox=True, shadow=True)


plot_data("NO2","O3",1,40,"blue","orange")
plot_data("SO2","CO",2,8,"green","red")
plt.show()


end_time=time.time()
print("Tempo impiegato:",end_time-start_time,"s")
spark.stop();
```

With the *seasonal_decompose* instruction, we deleted non-linearities from the functions represented by the AQIs variations, in order to emphasize the long term trend. Our data's granularity is in days and the seasonal patterns reappear yearly so we had to set the frequency to 365. We used an additive model because the time series' variance doesn't change significantly with an higher number of terms in the time series.

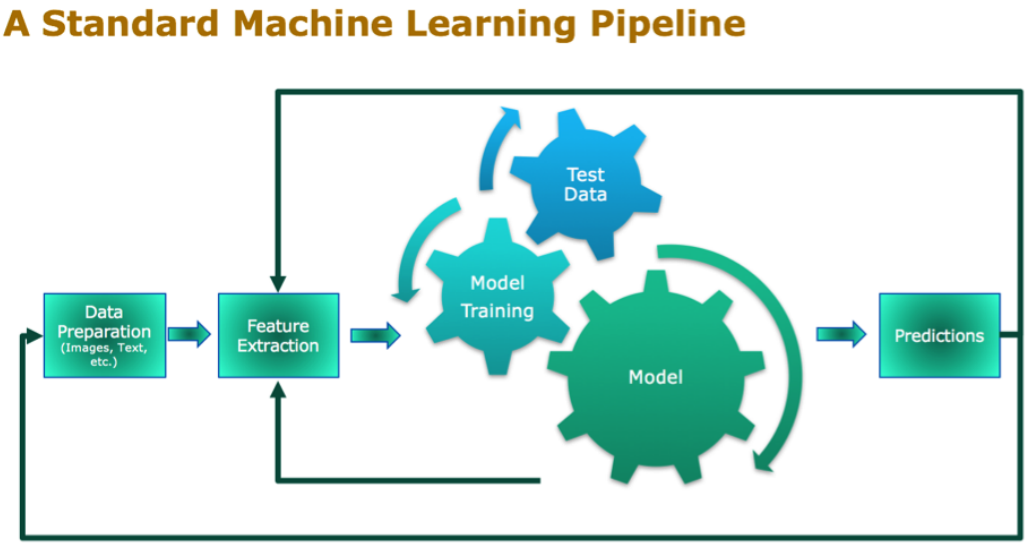




We noticed that O_3 AQIs are sightly increasing, while NO_2 and CO ones stay quite constant through the years, with a weak decrease in the last period. SO_2, instead, seems to have a heavily decreasing trend. This is surely an effect of the global limitations in force since 2006 to restrict emissions from power plants.
We represented pollutants on different figures to take into account the different scale of values measured.

3. Machine Learning

With the expression “Machine Learning” we mean the use and development of computer systems that are able to learn and adapt without following explicit instructions, by using algorithms and statistical models to analyze and draw inferences from patterns in data. A machine learning pipeline is used to help automate machine learning workflows. They operate by enabling a sequence of data to be transformed and correlated together into a model that can be tested and evaluated to achieve an outcome. The term “pipeline” is misleading as it implies a one-way flow of data. Instead, machine learning pipelines are cyclical and iterative as every step is repeated to continuously improve the accuracy of the model and achieve a successful result.



For this application we used a supervised learning approach. Supervised learning is about building a model that needs datasets to solve a particular problem using classification algorithms, and it is the most common use of machine learning. For its implementation, we used the Spark module MLlib, which provides common learning algorithms such as classification, regression, clustering, and collaborative filtering. With MLlib is possible to combine multiple algorithms into a single pipeline, or workflow.

Before discussing about the pipeline used, we needed to prepare data for the processing phase. Instead of taking into account the single measurements of different stations spread across the United States, we decided to consider the average AQIs of pollutants per day, just to make a more accurate prediction. To do a prediction of each pollutant, all machine learning operations are the same, so we executed them via a simple

procedure which takes in input the particular pollutant to consider, the dataframe on which to work, the number of figures where to represent a scatter and a temporal forecast plot and the color to distinguish the considered pollutant from the other ones.

```
from pyspark.ml.feature import StringIndexer,OneHotEncoder,VectorAssembler
from pyspark.ml.regression import DecisionTreeRegressor, DecisionTreeRegressionModel
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml import Pipeline
from pyspark.sql.functions import countDistinct
from pyspark.sql.functions import avg
from pyspark.sql import SparkSession
from pyspark.sql import functions as F
import datetime as dt
import time
import pandas as pd
import matplotlib.pyplot as plt


start_time=time.time()
spark = SparkSession.builder \
.master("local") \
.appName("Decision Tree Regressor") \
.config("spark.some.config.option", "some-value") \
.getOrCreate()

df=spark.read.load("/home/giuseppe/Second_Project/Preprocessed_Dataset.csv" , format="csv", sep=",", header="true")
df=df.select("State","County","Site Num","Day","Month","Year",(F.to_date("Date Local","yyyy-MM-dd").alias("Date")),"NO2 AQI","O3 AQI","SO2 AQI","CO AQI")
df=df.orderBy("Date")
df=df.groupBy("Date").agg(avg("NO2 AQI").alias("NO2"),avg("O3 AQI").alias("O3"),avg("SO2 AQI").alias("SO2"),
                          avg("CO AQI").alias("CO"),avg("Day").alias("Day"),avg("Month").alias("Month"),avg("Year").alias("Year"))

def Decision_Tree_Regressor(pollutant,fig_number1,fig_number2,color,dataframe):
 df=dataframe.select("Day","Month","Year","Date",pollutant)
 categoricalColumns = ['Day','Month','Year']
 stages = []
 for categoricalCol in categoricalColumns:
     stringIndexer = StringIndexer(inputCol = categoricalCol, outputCol = categoricalCol + 'Index',handleInvalid='keep')
     encoder = OneHotEncoder(inputCols=[stringIndexer.getOutputCol()], outputCols=[categoricalCol + "classVec"])
     stages += [stringIndexer, encoder]

 assemblerInputs = [c + "classVec" for c in categoricalColumns]
 assembler = VectorAssembler(inputCols=assemblerInputs, outputCol="features")
 stages += [assembler]
```

The next step is characterized by the definition of the vector “categoricalColumns”, which contains the dataframe’s fields that can be considered as a category for the data we are trying to predict (pollutants’ average AQIs). Since what we want to achieve is a temporal prediction of these values, the categorical columns selected are “Day”, “Month” and “Year”, which are temporal parameters. Then we encoded each of these columns to a column of label indices; with OneHotEncoder method, instead, we mapped these categorical features, represented as label indices, to a binary vector with at most a single one-value indicating the presence of a specific feature value from among the set of all feature values.

VectorAssembler is a transformer that combines a given list of columns into a single vector column. It is useful for combining raw features and also features generated by different feature transformers into a single feature vector, in order to train ML models. In each row, the values of the input columns will be concatenated into a vector in the specified order. In this case, we used the encoder output columns as input, naming the returned vector “features”.

```
df=df.withColumn(pollutant,df[pollutant].cast("double"))

decision_tree_regressor=DecisionTreeRegressor(featuresCol="features",labelCol=pollutant,maxDepth=20,maxBins=32)

stages+=[decision_tree_regressor]

train=df.where(df["Year"]<=2014)
test=df.where(df["Year"]>2014)
pipeline=Pipeline(stages=stages)
PipelineModel=pipeline.fit(train)
predictions=PipelineModel.transform(test)
predictions=predictions.select('Date',pollutant,'prediction')


evaluator_RMSE=RegressionEvaluator(labelCol=pollutant,predictionCol="prediction",metricName="rmse")
evaluator_MAE=RegressionEvaluator(labelCol=pollutant,predictionCol="prediction",metricName="mae")
rmse=evaluator_RMSE.evaluate(predictions)
mae=evaluator_MAE.evaluate(predictions)
```

At this point, we defined a particular regressor model. In the previous figure we used the Decision Tree Regressor as an example, but the overall structure of the pipeline doesn't change when using different regressors. Regardless of the regressor, we pass to its defining function the feature columns (the ones used to make the prediction) and the label column (the column we want to predict).
Then, we had to split the dataset in two parts: the training set, which contains every measure taken until 2014, and the testing set with every measure taken after 2014. We grouped all stages of the pipeline in a single variable, then we called the *fit()* method to produce a PipelineModel, which is a Transformer used at test time. The PipelineModel has the same number of stages as the original Pipeline, but all estimators in the original Pipeline have become transformers. When the PipelineModel's *transform()* method is called on the test dataset, the data are passed in order through the fitted pipeline. Each stage's *transform()* method updates the dataset and passes it to the next stage. Pipelines and PipelineModels help to ensure that training and test data go through identical feature processing steps.
This process produces a new dataframe, called "predictions", which is identical to the original one, but adds a column "prediction" that contains the predicted AQI value for each date of the test set.
To evaluate the models, we compared the column with the original AQI values (identified by the variable "pollutant" passed to the procedure) and the column with the predicted ones ("prediction") using two different error estimators:

RMSE (Root mean squared error): RMSE is a quadratic scoring rule that also measures the average magnitude of the error. It's the square root of the average of squared differences between prediction and actual observation.

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2}$$

MAE (Mean Absolute Error): MAE measures the average magnitude of the errors in a set of predictions, without considering their direction. It's the average over the test sample of the absolute differences between prediction and actual observation where all individual differences have equal weight.

$$\mathrm{MAE} = \frac{1}{n}\sum_{j=1}^{n}|y_j - \hat{y}_j|$$

```
predictions_Pandas=predictions.toPandas()
train_Pandas=train.toPandas()
test_Pandas=test.toPandas()

plt.figure(fig_number1)
plt.scatter(predictions_Pandas[pollutant],predictions_Pandas["prediction"],color=color)
plt.xlabel("Real "+pollutant+" values")
plt.ylabel(pollutant+" prediction")
plt.title('Predictions for '+pollutant+' with Decision Tree Regressor\nRoot Mean Square Error: '
                            +str(round(rmse,2))+'\nMean Absolute Error: '+str(round(mae,2)))

plt.figure(fig_number2)
plt.plot(train_Pandas["Date"],train_Pandas[pollutant],color="black",label="Train data")
plt.plot(test_Pandas["Date"],test_Pandas[pollutant],color="darkgray",label="Test data")
plt.plot(predictions_Pandas["Date"],predictions_Pandas["prediction"],color=color,label="Predicted data")
plt.legend(loc='upper left', ncol=1, fancybox=True, shadow=True)
plt.xlabel("Date")
plt.ylabel(pollutant+" AQI")
plt.title('Temporal forecasting for '+ pollutant+ ' with Decision Tree Regressor')


Decision_Tree_Regressor("NO2",1,2,"blue",df)
Decision_Tree_Regressor("O3",3,4,"orange",df)
Decision_Tree_Regressor("SO2",5,6,"green",df)
Decision_Tree_Regressor("CO",7,8,"red",df)

plt.show()

end_time=time.time()
print("Tempo impiegato:",end_time-start_time,"s")
spark.stop();
```

This code shows the last part of the sub-routine that plots the results of the prediction into two figures: the first one is a scatter plot that represents the offset of the predicted values from the actual ones; the second figure is a temporal forecast plot that shows the training data, the test data and the predicted data. We chose to follow the same color map used in the Data Exploration section for the different pollutants.

3.1: Regression Models

Regression analysis involves identifying the relationship between a dependent variable ("prediction" column) and one or more independent variables (those present in the "features" vector). A model of the relationship is hypothesized and estimates of the parameter values are used to develop an estimated regression equation. Various tests are then employed to determine if the model is satisfactory (in our case, RMSE and MAE have been used for this purpose). If the model is deemed satisfactory, the estimated regression equation can be used to predict the value of the dependent variable given values for the independent variables.
For our forecast, we used **Decision Tree**, **Gradient-Boosted Tree** and **Random Forest Regressors**.

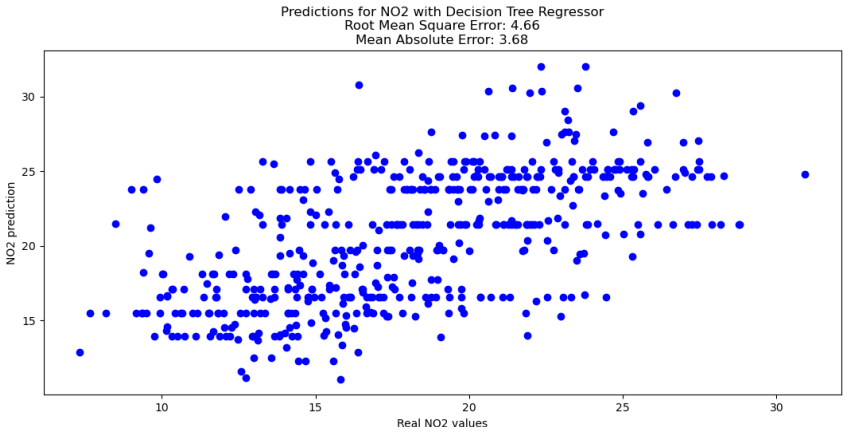
3.1.1: Decision Tree Regressor

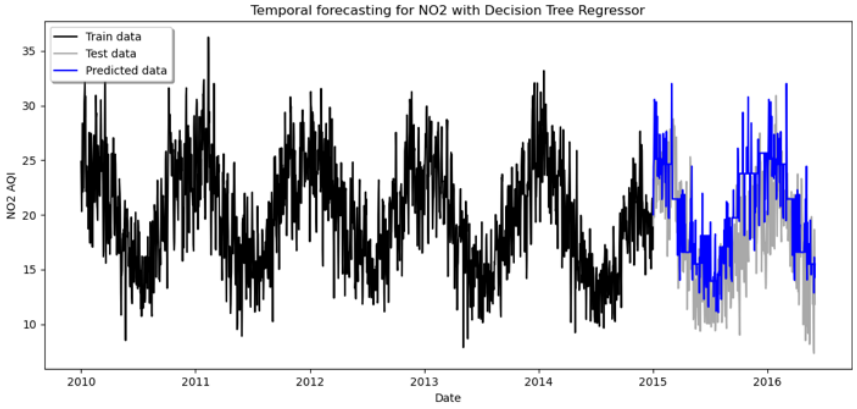
The decision tree is an algorithm that performs a recursive binary partitioning of the feature space. The tree predicts the same label for each bottommost (leaf) partition. Each partition is chosen greedily by selecting the *best split* from a set of possible splits, in order to maximize the information gain at a tree node.
To perform this kind of regression, we set these parameters:

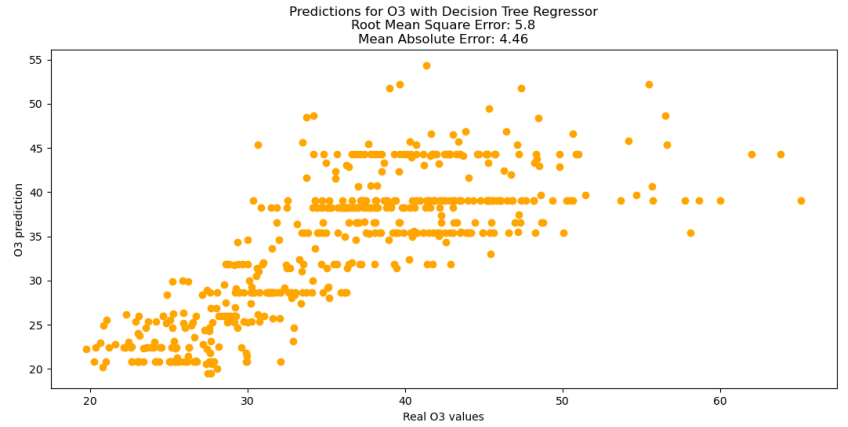
- **maxDepth:** maximum depth of a tree. Deeper trees are more expressive (potentially allowing higher accuracy), but they are also more costly to train and are more likely to overfit.

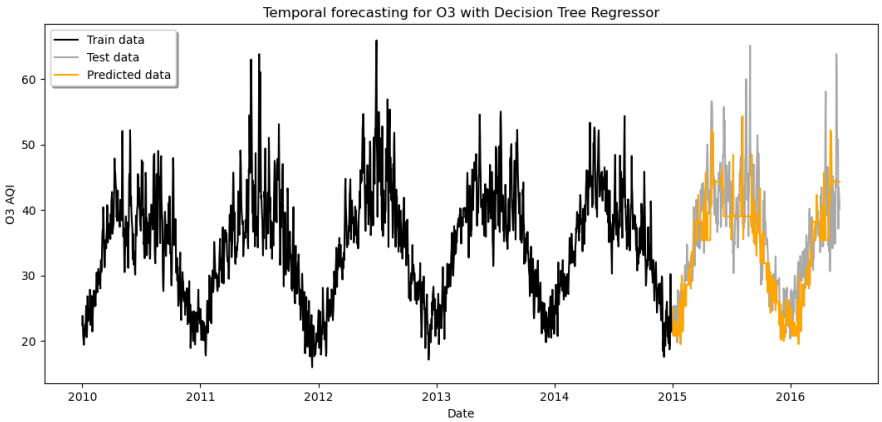
- **maxBins:** number of bins used when discretizing continuous features.
  Increasing maxBins allows the algorithm to consider more split candidates and make fine-grained split decisions. However, it also increases computation and communication. Note that the maxBins parameter must be at least the maximum number of categories for any categorical feature.

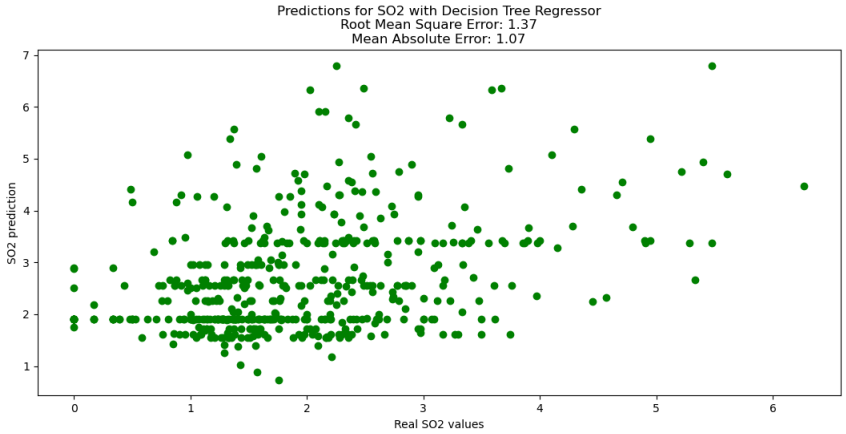
We set those parameters to 20 and 32 respectively, due to the fact that these values produced a more reliable model after evaluation (maxDepth produced results with slightly better error evaluations if set to 10, but the predictions were less diversified, so we preferred to keep the value to 20).

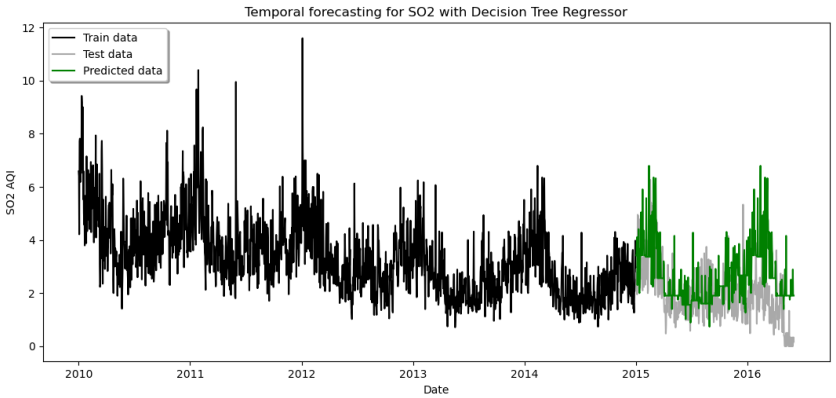

Predictions for NO2 with Decision Tree Regressor
Root Mean Square Error: 4.66
Mean Absolute Error: 3.68
NO2 prediction
Real NO2 values

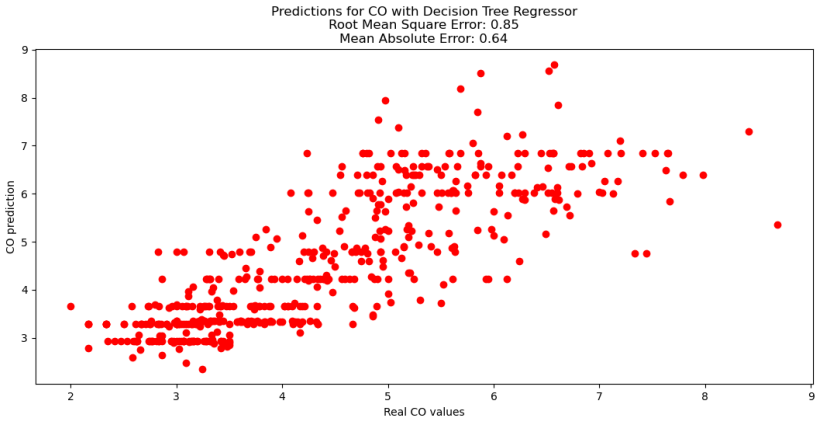

Temporal forecasting for NO2 with Decision Tree Regressor
Train data
Test data
Predicted data
NO2 AQI
Date

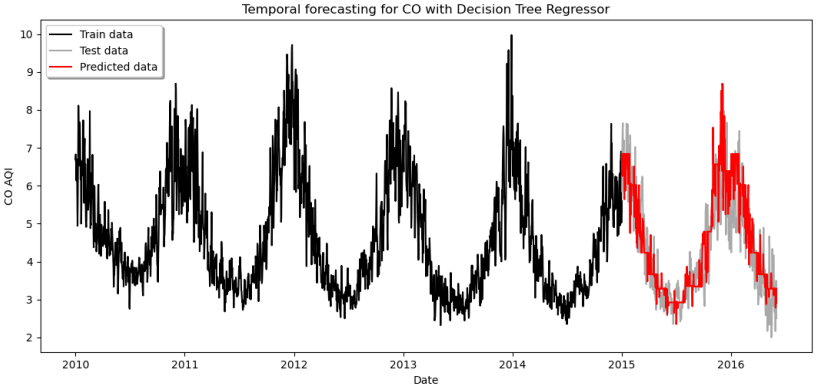

Predictions for O3 with Decision Tree Regressor
Root Mean Square Error: 5.8
Mean Absolute Error: 4.46
O3 prediction
Real O3 values


Temporal forecasting for O3 with Decision Tree Regressor
Train data
Test data
Predicted data
O3 AQI
Date


Predictions for SO2 with Decision Tree Regressor
Root Mean Square Error: 1.37
Mean Absolute Error: 1.07
SO2 prediction
Real SO2 values


Temporal forecasting for SO2 with Decision Tree Regressor
Train data
Test data
Predicted data
SO2 AQI
Date


Predictions for CO with Decision Tree Regressor
Root Mean Square Error: 0.85
Mean Absolute Error: 0.64
CO prediction
Real CO values

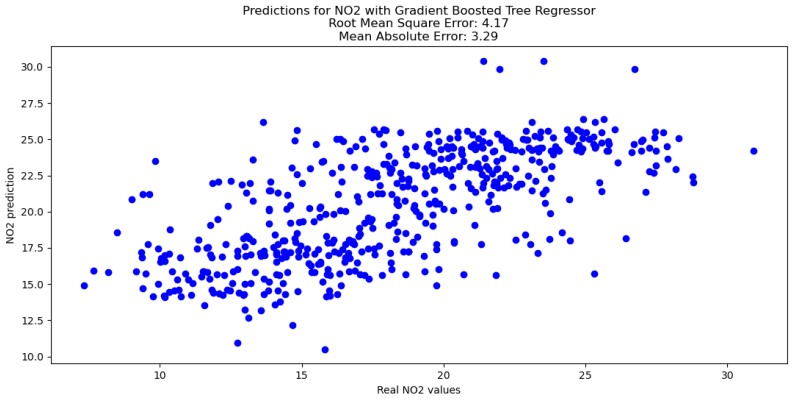

Temporal forecasting for CO with Decision Tree Regressor
Train data
Test data
Predicted data
CO AQI
Date

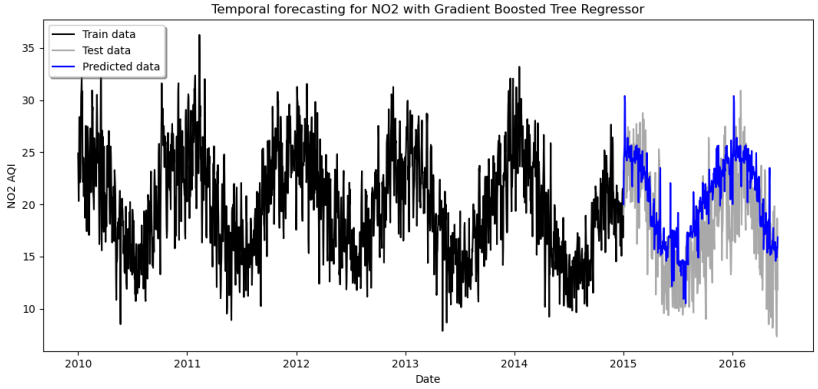
3.1.2 Gradient-Boosted Tree Regressor

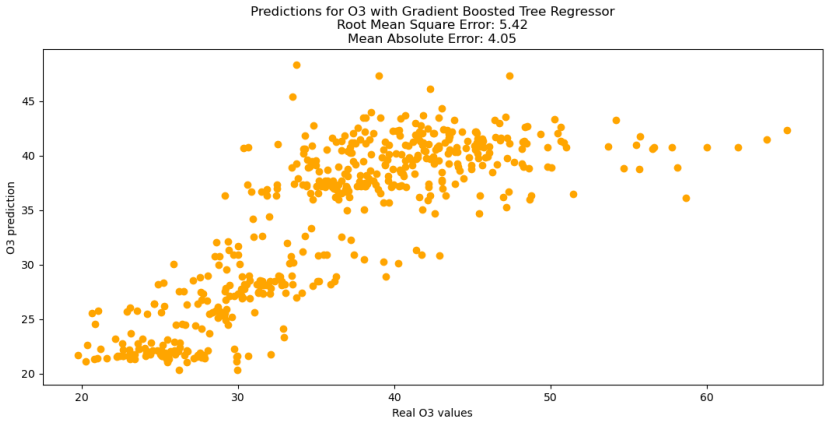
Gradient-Boosted Trees (GBTs) are ensembles of decision trees. GBTs iteratively train decision trees in order to minimize a loss function. Like decision trees, GBTs handle categorical features, extend to the multiclass classification setting, do not require feature scaling, and are able to capture non-linearities and feature interactions.
On each iteration, the algorithm uses the current ensemble to predict the label of each training instance and then compares the prediction with the true label. The dataset is re-labeled to put more emphasis on training instances with poor predictions. Thus, in the next iteration, the decision tree will help correct for previous mistakes.
The specific mechanism for re-labeling instances is defined by a loss function. With each iteration, GBTs further reduce this loss function on the training data.

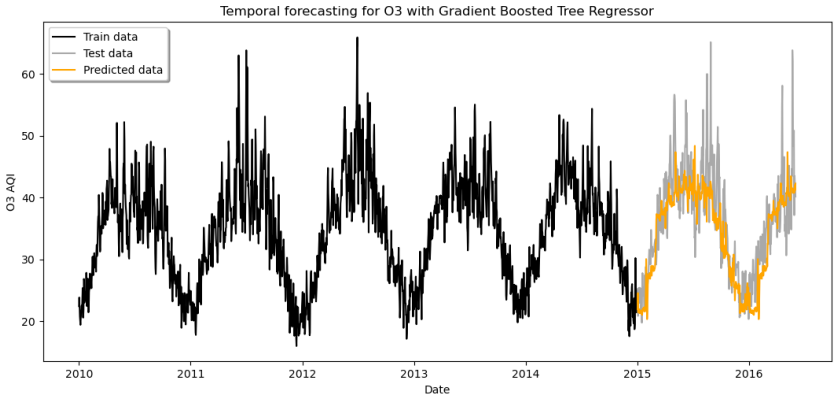
```
gbt_regressor=GBTRegressor(featuresCol="features",labelCol=pollutant,maxIter=20,maxDepth=10,maxBins=32)
```

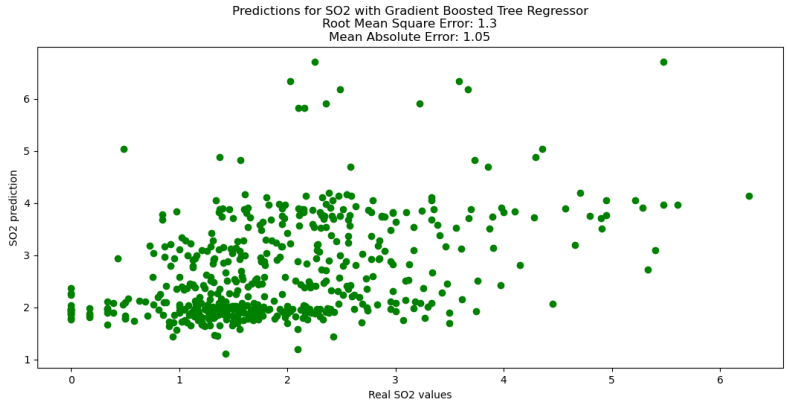
In addition to the parameters of the Decision Tree, GBT regressor has the **maxIter** parameter, which sets the number of trees in the ensemble. Each iteration produces one tree. Increasing this number makes the model more expressive, improving training data accuracy. However, test-time accuracy may suffer if this is too large.
As for the previous regressor, parameters were set accordingly to the best error evaluations.

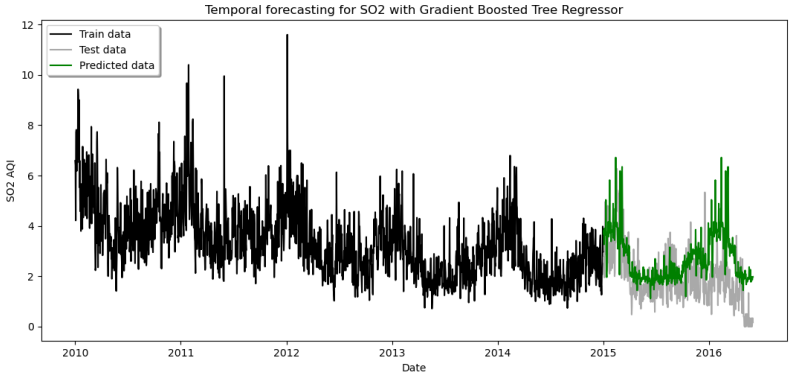


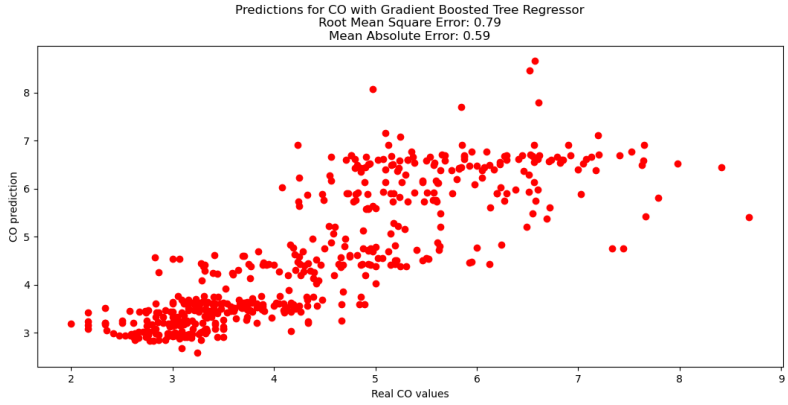

Temporal forecasting for NO2 with Gradient Boosted Tree Regressor
Train data
Test data
Predicted data
NO2 AQI
Date

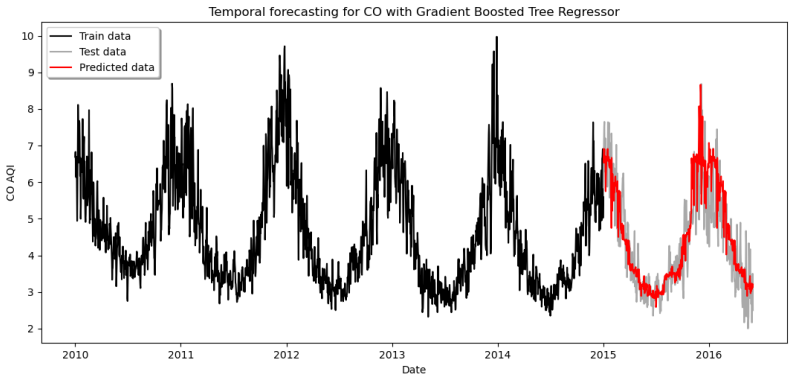

Predictions for O3 with Gradient Boosted Tree Regressor
Root Mean Square Error: 5.42
Mean Absolute Error: 4.05
O3 prediction
Real O3 values


Temporal forecasting for O3 with Gradient Boosted Tree Regressor
Train data
Test data
Predicted data
O3 AQI
Date


Predictions for SO2 with Gradient Boosted Tree Regressor
Root Mean Square Error: 1.3
Mean Absolute Error: 1.05
SO2 prediction
Real SO2 values


Temporal forecasting for SO2 with Gradient Boosted Tree Regressor
Train data
Test data
Predicted data
SO2 AQI
Date


Predictions for CO with Gradient Boosted Tree Regressor
Root Mean Square Error: 0.79
Mean Absolute Error: 0.59
CO prediction
Real CO values


Temporal forecasting for CO with Gradient Boosted Tree Regressor
Train data
Test data
Predicted data
CO AQI
Date

3.1.3: Random Forest Regressor

Random forests are ensembles of decision trees. They combine many decision trees in order to reduce the risk of overfitting. Like decision trees, random forests are able to handle categorical features and to capture non-linearities and feature interactions. Random forests train a set of decision trees separately, so the training can be done in parallel. The algorithm injects randomness into the training process so that each decision tree is a bit different. Combining the predictions from each tree reduces the variance of the predictions, improving the performance on test data.

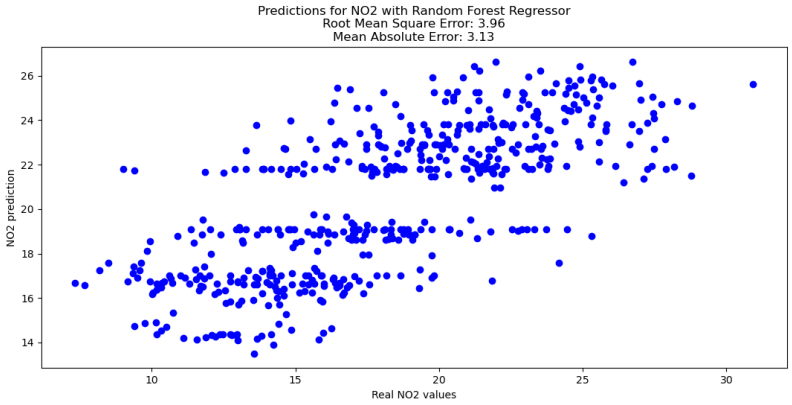
```
random_forest_regressor=RandomForestRegressor(featuresCol="features",labelCol=pollutant,numTrees=25,maxDepth=10,maxBins=32)
```

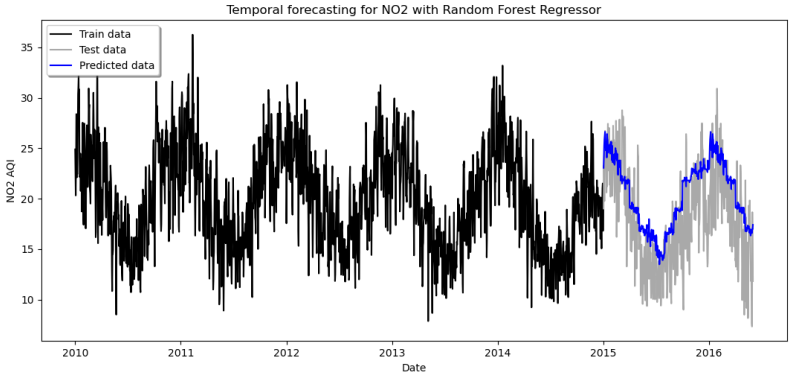
Particular parameters for this regressor are:

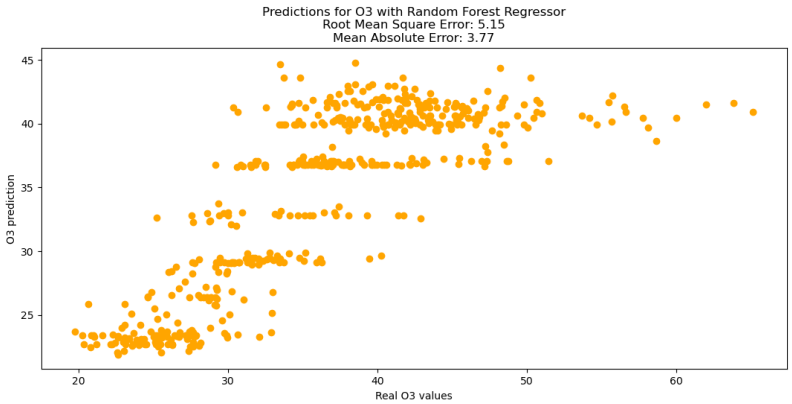
numTrees: number of trees in the forest. Increasing the number of trees will decrease the variance in predictions, improving the model's test-time accuracy.
Training time increases roughly linearly in the number of trees.

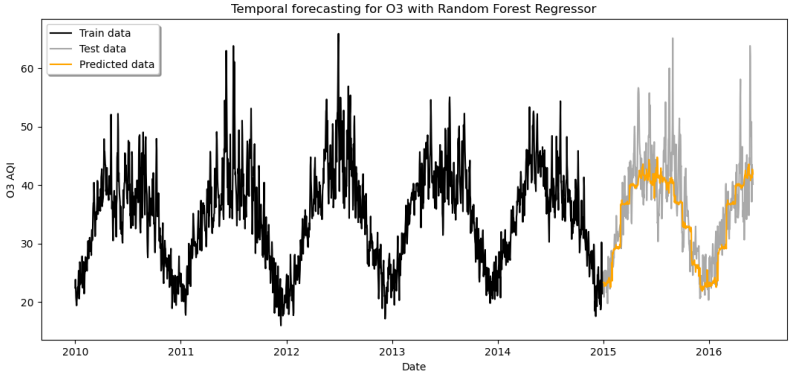
maxDepth: maximum depth of each tree in the forest.
Increasing the depth makes the model more expressive and powerful. However, deep trees take longer to train and are also more prone to overfitting.
In general, it is acceptable to train deeper trees when using random forests than when using a single decision tree. One tree is more likely to overfit than a random forest (because of the variance reduction from averaging multiple trees in the forest).

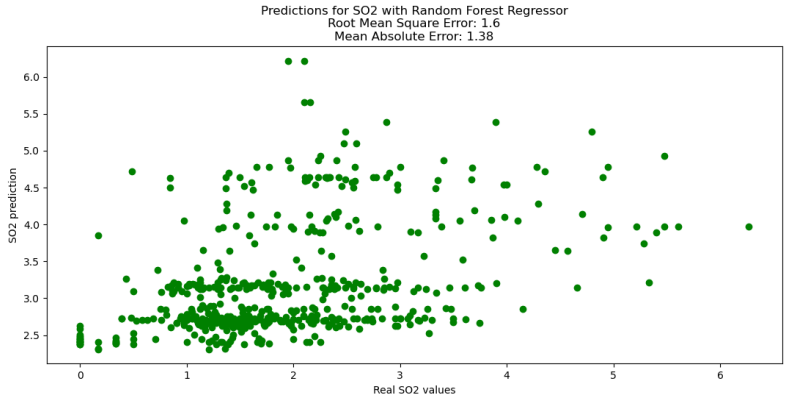
As usual, we set these parameters to have the best error evaluation.
Figures in the next page.

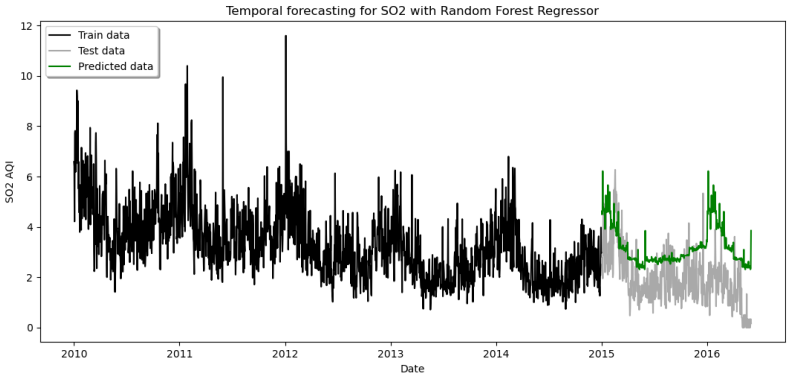

Predictions for NO2 with Random Forest Regressor
Root Mean Square Error: 3.96
Mean Absolute Error: 3.13
NO2 prediction
Real NO2 values

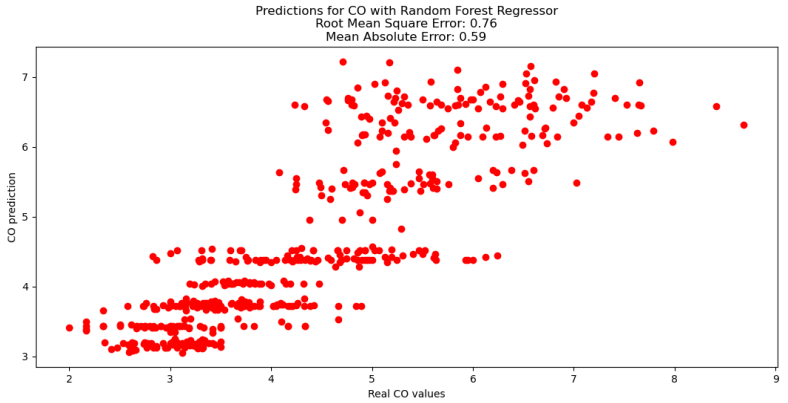

Temporal forecasting for NO2 with Random Forest Regressor
Train data
Test data
Predicted data
NO2 AQI
Date

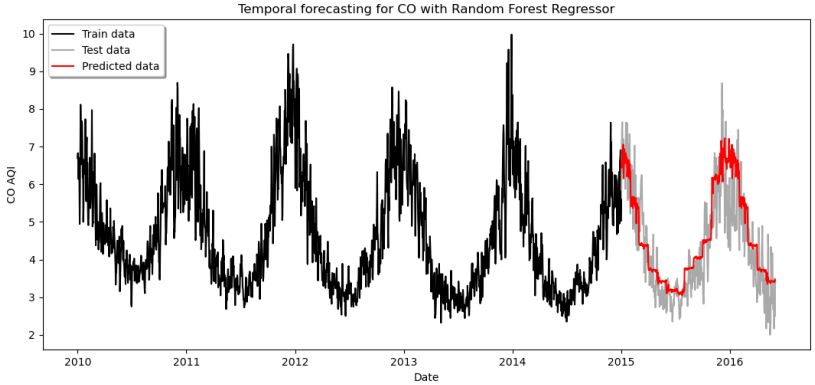

Predictions for O3 with Random Forest Regressor
Root Mean Square Error: 5.15
Mean Absolute Error: 3.77
O3 prediction
Real O3 values


Temporal forecasting for O3 with Random Forest Regressor
Train data
Test data
Predicted data
O3 AQI
Date


Predictions for SO2 with Random Forest Regressor
Root Mean Square Error: 1.6
Mean Absolute Error: 1.38
SO2 prediction
Real SO2 values


Temporal forecasting for SO2 with Random Forest Regressor
Train data
Test data
Predicted data
SO2 AQI
Date


Predictions for CO with Random Forest Regressor
Root Mean Square Error: 0.76
Mean Absolute Error: 0.59
CO prediction
Real CO values


Temporal forecasting for CO with Random Forest Regressor
Train data
Test data
Predicted data
CO AQI
Date

3.2 SARIMAX

The regression models used so far provided a forecast without considering seasonal trends or patterns. To take these characteristics into account, we tried to use another model called SARIMAX (Seasonal Autoregressive Integrated Moving-Average with Exogenous Regressors). SARIMAX is an extension of the SARIMA model which also includes the modeling of exogenous variables. Exogenous variables are also called covariates and can be thought of as parallel input sequences that have observations at the same time steps as the original series. The observations for exogenous variables are included in the model directly at each time step and are not modeled in the same way as the primary endogenous sequence.
The problem with SARIMAX is that it doesn't support high values of seasonal frequency (ours is 365). One way around that is by using Fourier Terms to add an exogenous variable that has the proper frequency we need.
First of all, we need to install *pmdarima* with the following command in order to be able to execute a SARIMAX prediction:

```
giuseppe@giuseppe-VirtualBox:~$ pip install pmdarima
```

3.2.1: Fourier Terms

```
def SARIMAX(pollutant,fig_number1,fig_number2,color,dataframe):
 df=dataframe.select("Date",pollutant)
 df=df.toPandas()
 df.index = pd.to_datetime(df["Date"])

 train = df[:'2014-12-31']
 test = df['2015-01-01':'2016-12-31']

 # Fourier terms
 fourier = pd.DataFrame(index=df.index)
 # Frequency is being set to 365.25 because we have leap years
 fourier['sin_1'] = np.sin(2 * np.pi * fourier.index.dayofyear / 365.25)
 fourier['cos_1'] = np.cos(2 * np.pi * fourier.index.dayofyear / 365.25)
 fourier['sin_2'] = np.sin(4 * np.pi * fourier.index.dayofyear / 365.25)
 fourier['cos_2'] = np.cos(4 * np.pi * fourier.index.dayofyear / 365.25)
 fourier_train = fourier.iloc[:len(train), :]
 fourier_test = fourier.iloc[len(train):(len(train)+len(test)), :]
```

Fourier terms are basically combinations of *sines* and *cosines* using the frequency of our data's seasonality. We will be using the first four terms, which means two sines and two cosines. Then, we will use the Fourier terms obtained as Exogenous values for our SARIMAX model.

3.2.2: Auto-Arima

There are many parameters to choose for a SARIMA model, so we will use pmdarima's *auto_arima* function that searches for the best sets of parameters. To use our fourier terms we need to feed the exogenous parameter with our fourier terms data.

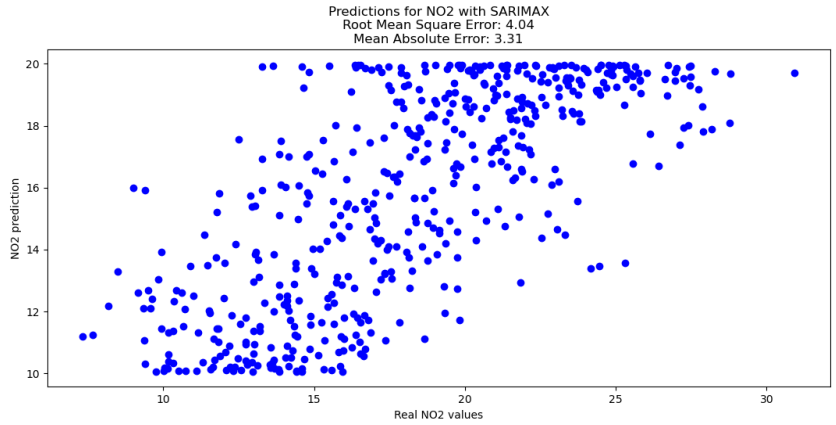
Then, we built our model and evaluated it, as usual, through RMSE and MAE:

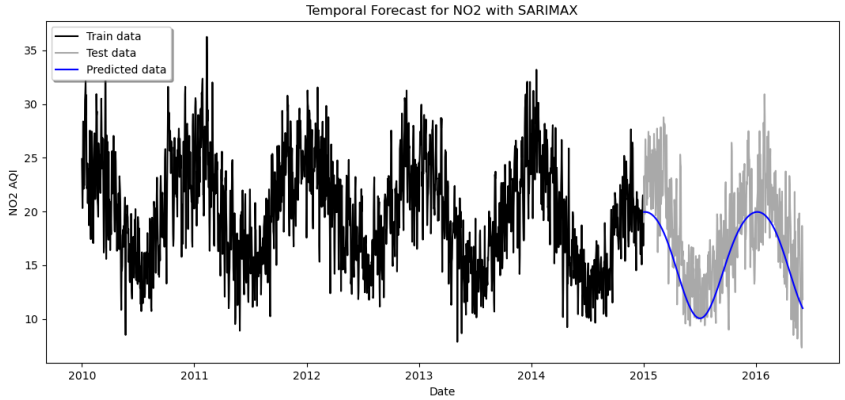
```
arima = pm.auto_arima(train[pollutant],exogenous=fourier_train, start_p=1, start_q=0,stepwise=True,suppress_warnings=True,error_action='ignore')

predictions = arima.predict(n_periods=len(test), exogenous=fourier_test)
sarima_pred = pd.DataFrame(np.c_[predictions],index=test.index)
rmse = np.sqrt(mean_squared_error(test[pollutant], sarima_pred))
mae = mean_absolute_error(test[pollutant], sarima_pred)
```

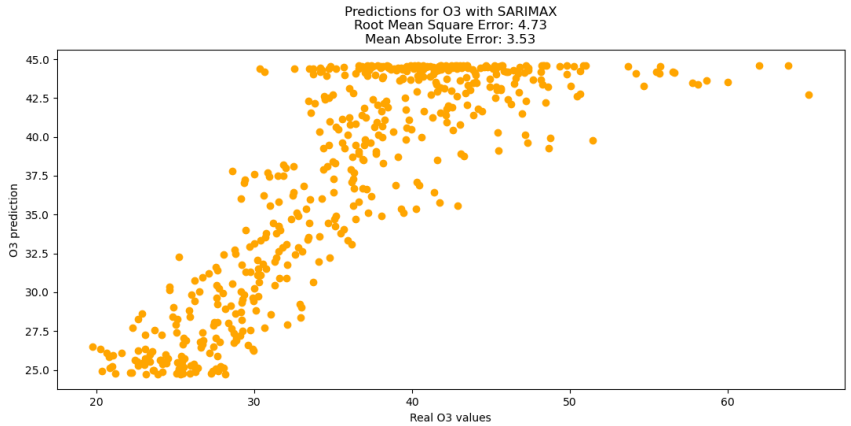
Finally, we plotted the forecast for each pollutant:

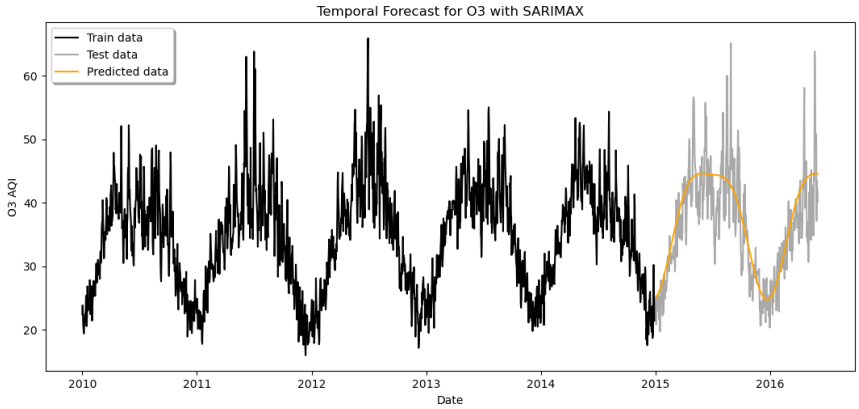
```
 plt.figure(fig_number1)
 plt.scatter(test[pollutant],sarima_pred,color=color)
 plt.xlabel("Real "+pollutant+" values")
 plt.ylabel(pollutant+" prediction")
 plt.title('Predictions for '+ pollutant+' with SARIMAX\nRoot Mean Square Error: '
                   +str(round(rmse,2))+'\nMean Absolute Error: '+str(round(mae,2)))

 plt.figure(fig_number2)
 plt.plot(train["Date"],train[pollutant],color="black",label="Train data")
 plt.plot(test["Date"],test[pollutant],color="darkgray",label="Test data")
 plt.plot(sarima_pred, color=color,label="Predicted data")
 plt.legend(loc='upper left', ncol=1, fancybox=True, shadow=True)
 plt.xlabel("Date")
 plt.ylabel(pollutant+" AQI")
 plt.title('Temporal Forecast for '+pollutant+' with SARIMAX')


SARIMAX("NO2",1,2,"blue",df)
SARIMAX("O3",3,4,"orange",df)
SARIMAX("SO2",5,6,"green",df)
SARIMAX("CO",7,8,"red",df)

plt.show()

end_time=time.time()
print("Tempo impiegato:",end_time-start_time,"s")
spark.stop();
```

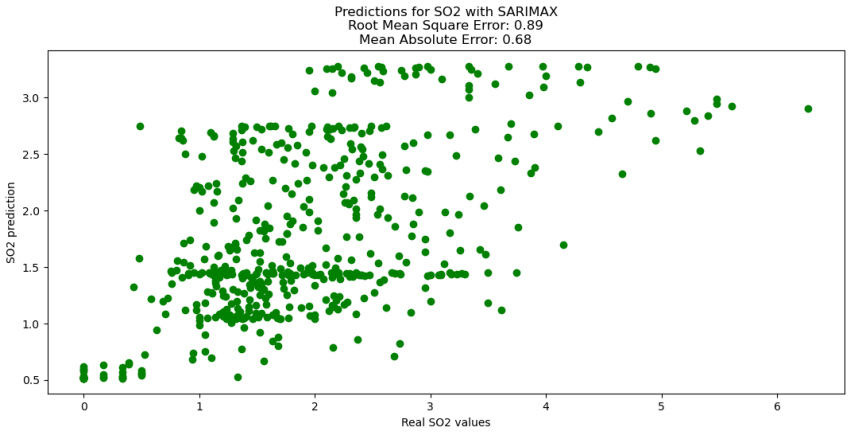
Figures in the next page.

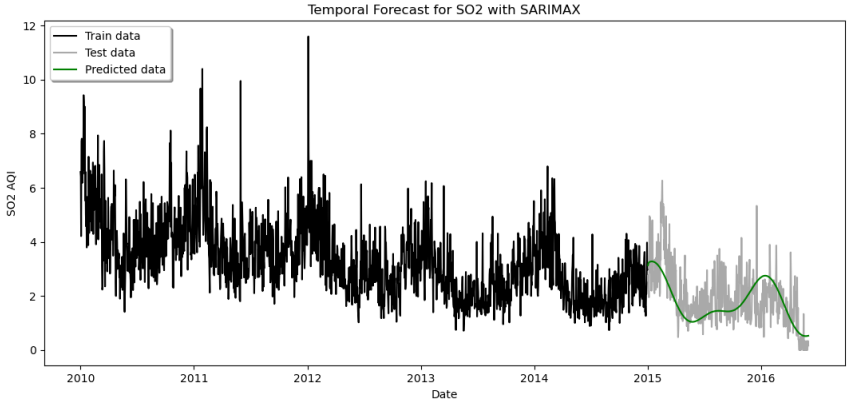

Predictions for NO2 with SARIMAX
Root Mean Square Error: 4.04
Mean Absolute Error: 3.31
NO2 prediction
Real NO2 values

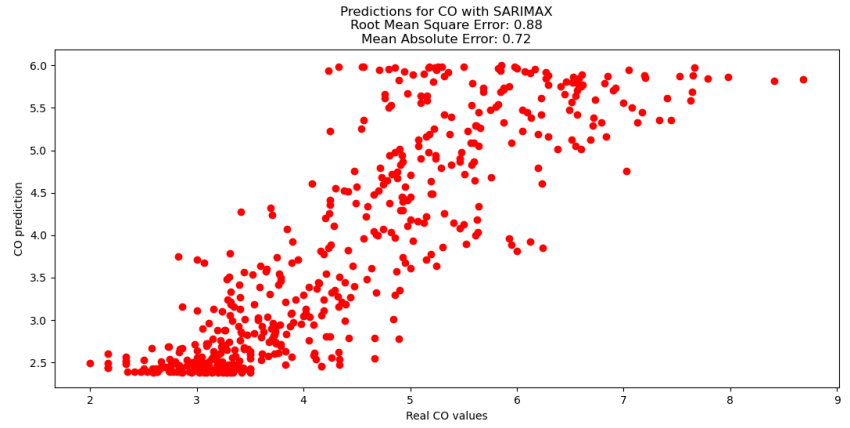

Temporal Forecast for NO2 with SARIMAX
Train data
Test data
Predicted data
NO2 AQI
Date

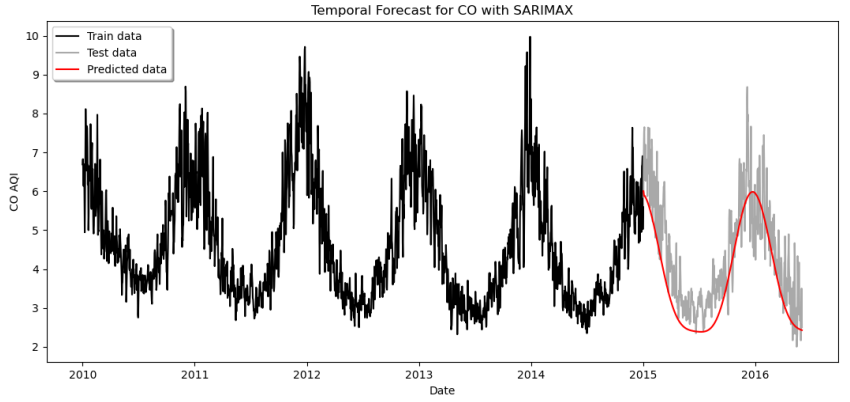

Predictions for O3 with SARIMAX
Root Mean Square Error: 4.73
Mean Absolute Error: 3.53
O3 prediction
Real O3 values


Temporal Forecast for O3 with SARIMAX
Train data
Test data
Predicted data
O3 AQI
Date


Predictions for SO2 with SARIMAX
Root Mean Square Error: 0.89
Mean Absolute Error: 0.68
SO2 prediction
Real SO2 values


Temporal Forecast for SO2 with SARIMAX
Train data
Test data
Predicted data
SO2 AQI
Date


Predictions for CO with SARIMAX
Root Mean Square Error: 0.88
Mean Absolute Error: 0.72
CO prediction
Real CO values


Temporal Forecast for CO with SARIMAX
Train data
Test data
Predicted data
CO AQI
Date

4. Conclusions

As we can see from the prediction results, there are substantial differences in errors and plots, varying between different pollutants and machine learning models.

NO2	Decision Tree	Gradient Boosted Tree	**Random Forest**	SARIMAX
RMSE	4,66	4,17	**3,96**	4,04
MAE	3,68	3,29	**3,13**	3,31

O3	Decision Tree	Gradient Boosted Tree	Random Forest	**SARIMAX**
RMSE	5,8	5,42	5,15	**4,73**
MAE	4,46	4,05	3,77	**3,53**

SO2	Decision Tree	Gradient Boosted Tree	Random Forest	**SARIMAX**
RMSE	1,37	1,3	1,6	**0,89**
MAE	1,07	1,05	1,38	**0,68**

CO	Decision Tree	Gradient Boosted Tree	**Random Forest**	SARIMAX
RMSE	0,85	0,79	**0,76**	0,88
MAE	0,64	0,59	**0,59**	0,72

According to these tables, we can make some considerations about the best model for each pollutant:

- **NO2** errors are high in general, but Random Forest regression model seems to be the more accurate one, due to its lower errors. The forecast will be inevitably imprecise because the pollutant's measurements have an high grade of volatility;

- **O3** has the same issues of NO2 in terms of accuracy, but for this pollutant SARIMAX model seems to be the more suitable, given the errors;

- For **SO2**, all Machine Learning algorithms provide a better prediction compared to the previous two pollutants, but the best one seems to be SARIMAX, also because it takes into account the decreasing trend of the pollutant measurements;

- **CO**, instead, thanks to its overall constant behavior, allows to all models applied to register extremely low errors. Anyway, the Random Forest regression model results the most appropriate one for this pollutant.

During the training phase of these models, we noticed a strong discrepancy in terms of execution times from one another. In particular, while Decision Tree, Random Forest and SARIMAX took about 10-15 minutes to complete, the Gradient Boosted Tree regressor needed about 30 minutes to finish the procedure. It also didn't report good results when compared to the other models. This means that it isn't an appropriate machine learning algorithm to execute a time series prediction of this type.
With all the results collected with the execution of this project, we can finally assert that it is possible to predict pretty accurately the future variations of air quality levels just by looking at historical data with the only condition being that the real measurements of the parameter of interest must be reliable and reported frequently and constantly. Combining this feature with basic machine learning algorithms, we were able to elaborate a forecast over a wide area like the US.
Possible future developments for a project like this, can be related to trying to predict air quality levels on a small geographical region, managing the lack of big amounts of data with the consideration of a lot of different factors linked to the emission of different pollutants (like population density, proximity of the area to the sea, industrialization ecc.) and with the power of more advanced machine learning methods.