Factors Influencing NO2 Air Quality in Beijing

Anandhan Manoharan

Virginia Tech University

CS 5805: Machine Learning I

Dr. Reza Jafari

December 8, 2024

Table of Contents

Abstract

Although now seeing improvements, over the years Beijing, China has been known to have some of the worst air pollution in the world with many different pollutants present in the air and with that we see constant smog present in the city and people typically wear masks to avoid breathing too many of these pollutants. To look at the different air pollutants present in Beijing and look at possible factors contributing to the prevalence of these pollutants in the air, the Beijing multi-site air quality data set was used for the analysis. Many factors will be looked at to see what contributes to the varying levels of pollutants and more specifically the NO2 pollutant will be looked at in more detail since through exploration phase that pollutant seemed to be the most spread out compared to the rest of the pollutants. Regression was run using NO2 as the dependent variable and factors such as temperature, pressure, dew point temperature, and wind speed as the independent variables. Various classifications were run and the performances of them were looked at and the best one was chosen to be used for future analysis. Finally, clustering and association independent research was conducted and observations from this research were noted.

Introduction

Beijing, China has always been one of the cities with the worst air pollution in the world with consistent smog present in the city. In the air there are many air pollutants present that with enough exposure can contribute to health problems. Analyzing these air pollutants and what factors may contribute to the prevalence of these pollutants can aid in setting up preventive measures or prepare for higher levels of these pollutants to keep the people safe . The objective of the final term project is to look at the factors that could be contributing to the prevalence of the various air pollutants. First step in the procedure is there was data exploration that was conducted before really jumping into the analysis. In the exploration phase, decided to look at the distribution of the different air pollutants in the data set and found that the NO2 pollutant was the most spread-out pollutant of them all so decided to direct further analysis on this one pollutant. Other exploration was done such as looking at the levels of NO2 in different seasons as well as looking at NO2 in relation to one of the variables in a scatterplot. The second step was conducting some feature engineering and data cleaning where I replaced the N/As with the mean since I want to all the information possible. Also I looked at what variables were most important or least important by conducting random forest which helped in determining important variables and least important variables. This was also supported by conducting VIF and covariance matrix. Next, using the some regression was conducted to look at how the dependent variable, the NO2,

was affected with a bunch of independent variables such as temperature, pressure, wind speed, and dewpoint were used as they were found to be the most important variables according to the analysis we did during the feature engineering. Third, I ran the model through different classifications to see which classifications had the best performance with the model by looking at things such as accuracy and plotting a graph showing all the performances. Finally, further analysis was conducted by doing the clustering and association with K-mean algorithm, DBSCAN algorithm, and Apriori algorithm.

Description of the Dataset

The data set being used in our analysis is called the Beijing Multi-Site Air Quality where it is from the area of climate and the environment. The data set consists of measurements of the air quality from twelve different stations that are spread out across the city of Beijing. This data set does meet the criteria since it has 420,768 observations which is well over the required 50,000 observations. The data set also consists of 18 features which are the six air pollutants such as PM2.5, PM10, SO2, NO2, CO, and O3. Then we also have time-based features such as year, month, day, and hour. Then you have some other weather-based factors as features such as temperature, pressure, dew point, precipitation, wind direction, and wind speed. Finally, there are some extra features such as the station name and row number. All the data sets were separated by station name with 12 different separate files. Within the data set, the feature that I chose to be the dependent variable was the NO2 feature which is one of the air pollutants in the air called Nitrogen Dioxide and the independent variables that I chose were some of the weather factors such as TEMP, PRES, DEWP, RAIN, wd, and WSPM. This data set would be important in

industry as it would aid in combating climate change. From this data set my goal was to see if there are certain factors that contribute to the prevalence of some of the air pollutants and from this analysis, we could apply to this to other areas of climate change and how to combat pollution levels all over the world.

Phase I: Feature Engineering & EDA

The data set came separated into 12 different data sets corresponding to a particular station. So in other words, each station had its own data set each with 18 features and 35,064 observations which is below the requirements that is needed. At first, I was thinking about using only two station's data but then I changed my mind and wanted to use all twelve stations to get as much information as possible. So, my first task that I worked on was to merge all twelve data sets together and as Figure 1 shows, we now have total observations of 420,767 and still have the 18 features and the station features has twelve different stations now listed. Next I decided to conduct some data

	No	year	month	day	hour	PM2.5	PM10	SO2	NO2	CO	O3	\
0	1	2013	3	1	0	3.0	6.0	13.0	7.0	300.0	85.0	
1	2	2013	3	1	1	3.0	3.0	6.0	6.0	300.0	85.0	
2	3	2013	3	1	2	3.0	3.0	22.0	13.0	400.0	74.0	
3	4	2013	3	1	3	3.0	6.0	12.0	8.0	300.0	81.0	
4	5	2013	3	1	4	3.0	3.0	14.0	8.0	300.0	81.0	
...	...	...	...	...	...	...	...	...	...	...	...	
420763	35060	2017	2	28	19	11.0	32.0	3.0	24.0	400.0	72.0	
420764	35061	2017	2	28	20	13.0	32.0	3.0	41.0	500.0	50.0	
420765	35062	2017	2	28	21	14.0	28.0	4.0	38.0	500.0	54.0	
420766	35063	2017	2	28	22	12.0	23.0	4.0	30.0	400.0	59.0	
420767	35064	2017	2	28	23	13.0	19.0	4.0	38.0	600.0	49.0	

	TEMP	PRES	DEWP	RAIN	wd	WSPM	station
0	-2.3	1020.8	-19.7	0.0	E	0.5	Changping
1	-2.5	1021.3	-19.0	0.0	ENE	0.7	Changping
2	-3.0	1021.3	-19.9	0.0	ENE	0.2	Changping
3	-3.6	1021.8	-19.1	0.0	NNE	1.0	Changping
4	-3.5	1022.3	-19.4	0.0	N	2.1	Changping
...	...	...	...	...	...	...	...
420763	12.5	1013.5	-16.2	0.0	NW	2.4	Wanshouxigong
420764	11.6	1013.6	-15.1	0.0	WNW	0.9	Wanshouxigong

Figure 1: Merged Data

```
Missing values after cleaning:
No            0
year          0
month         0
day           0
hour          0
PM2.5         0
PM10          0
SO2           0
NO2           0
CO            0
O3            0
TEMP          0
PRES          0
DEWP          0
RAIN          0
wd         1822
station       0
```

Figure 2: Cleaned Data

cleaning with he merged data set and since there were tons of time based data present and also I wanted to get as much information as possible, I decided to replace the missing values with the mean and as we can see with Figure 2, all the missing values have been replaced. The wind direction feature still has numbers listed as not replaced because the values in this feature are all strings and there was no missing numerical data in this feature so that's why that is the only feature with no replacements. Now after cleaning the data, I decided to conduct some exploratory

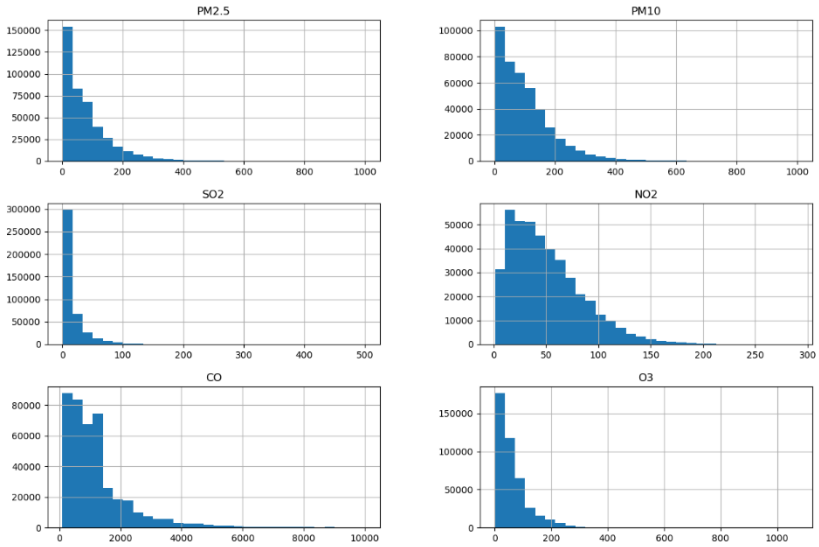


Figure 3: Histogram of Six Pollutants

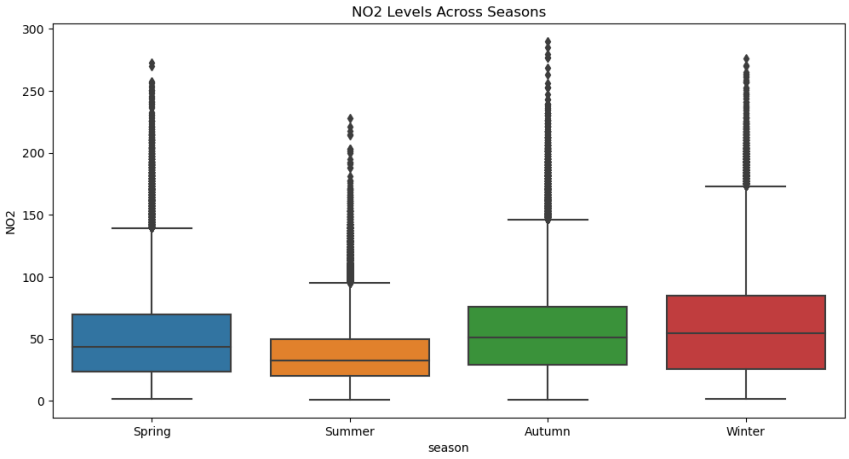


Figure 4: Box Plot of NO2 levels

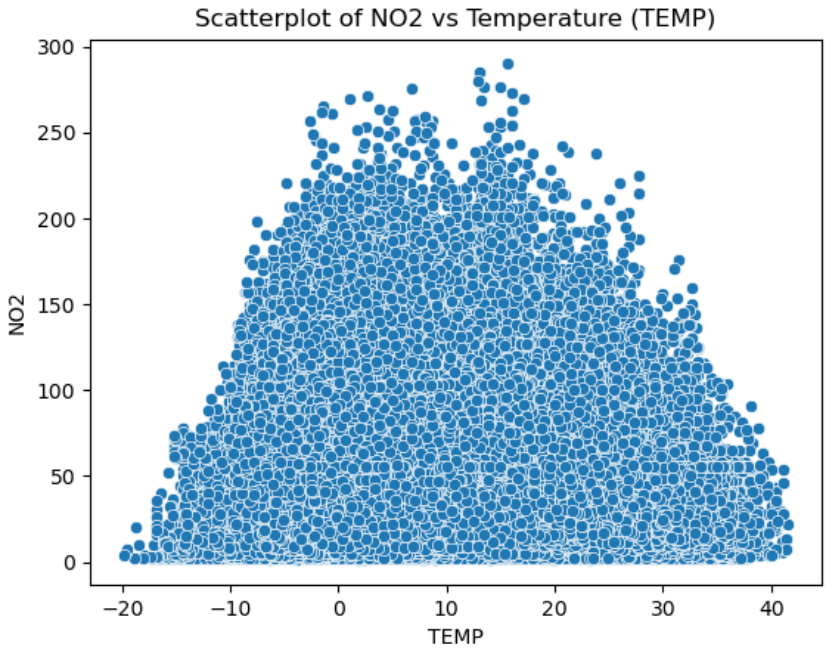


Figure 5: Scatterplots of NO2 levels vs Temperature

data analysis before moving onto more feature engineering. The first exploration I conducted was to look at the distributions of all six air pollutants by creating a histogram for each of them. Figure 3 shows these histograms and according to this it appears that the NO2 air pollutant has the most balanced distribution of all of the other pollutants which gave me a clue that maybe I should focus on this air pollutant for further analysis. Since I came to the decision to focus in on the NO2 air pollutant, I decided to focus the rest of my exploratory data analysis on this air pollutant. As Figure 4 shows, I decided to create a bunch of box plots showing the spread of the NO2 air pollutant levels across the different seasons and as we can see from the figure, it appears that the levels are most spread out in the autumn and next highest spread appears to be in the winter. Some possible things you can think about is perhaps NO2 pollutant is more prevalent in colder weather than in warmer weather. Next exploration I decided to create a scatterplot with NO2 levels vs Temperature. Figure 5 showcases this and as you can see it appears that my assertions

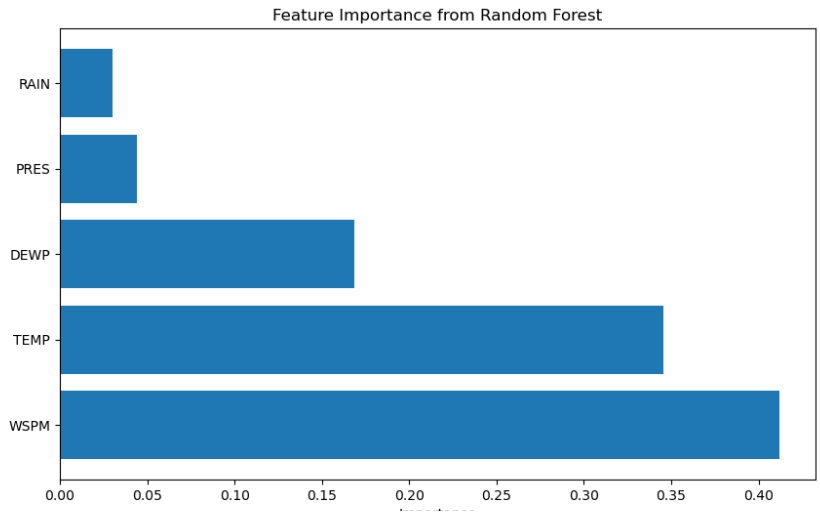


Figure 6 Random Forest Analysis for Feature Importance

from the box plot is followed here as it looks like the highest levels of NO2 are most prevalent among the colder temperatures.

Now I went onto conducting some feature engineering and the first thing I looked at was whether I wanted to do dimension reduction or feature selection. I decided to go with feature selection, where one would choose only certain features to focus on and since I already had an idea of which features to focus on I went with this and the features that I wanted to focus on was NO2 as the dependent variable and the independent variables being RAIN, wd, WSPM, TEMP, DEWP, and PRES. Those choosing to conduct feature selection, we typically would need to conduct a random forest analysis to look at the most important features among the independent variables. As we can see from the Figure 6, the least important features appear to be the RAIN and PRES features. Going off this random forest analysis, I decided to eliminate the RAIN feature from my feature selection and move on with

	Feature	Importance
4	WSPM	0.411965
0	TEMP	0.345327
2	DEWP	0.168640
1	PRES	0.043918
3	RAIN	0.030150

Figure 8: Feature Importance

	Feature	VIF
0	NO2	4.142522
1	TEMP	10.589062
2	PRES	12.305723
3	DEWP	4.783677
4	WSPM	4.627468

Figure 7 VIF analysis

Figure 8: VIF analysis

just WSPM, TEMP, DEWP, and PRES. Also, I wanted to make sure that there wasn't any multicollinearity present in the data so to look for this I decided to conduct the VIF analysis with the remaining variables. As we can see in Figure 8, we have two features with a VIF value of 10 or higher which are TEMP and PRES. This suggests that there is collinearity present in our data, and this would need to be addressed. The way I addressed this is to eliminate one of the two variables that have a VIF value higher than 10. Since PRES has the highest VIF value and in the Random Forest analysis from Figure 6 shows that it has the second lowest importance of all the features, I decided to eliminate the PRES feature and then decide the re run the VIF analysis to see if collinearity issue has been resolved. As we can see from Figure 9, all of the VIF values :for the remaining features is below 10 which suggests that there isn't any significant multicollinearity present and so I would say that the collinearity issue has been solved. After handling the feature selection and solving the collinearity issues, my next task was to conduct some discretization so that it would be easier to run the model through the classification. What I decided to do was to do some label encoding where a new feature was created called NO2 Safety to show whether the NO2 value is safe or unsafe and I set the threshold as 10 which is the guidelines that World Health Organization has set [WHO 2021]. Figure 10 shows the head of the data set with the new feature that has been created after label encoding called NO2_safety and we see that any value of

	Feature	VIF
0	NO2	1.586495
1	TEMP	7.453934
2	DEWP	4.221374
3	WSPM	3.911189

Figure 9 VIF analysis after elimination

	NO2	TEMP	DEWP	wd	WSPM	station	NO2_safety
0	7.0	-2.3	-19.7	E	0.5	Changping	safe
1	6.0	-2.5	-19.0	ENE	0.7	Changping	safe
2	13.0	-3.0	-19.9	ENE	0.2	Changping	unsafe
3	8.0	-3.6	-19.1	NNE	1.0	Changping	safe
4	8.0	-3.5	-19.4	N	2.1	Changping	safe

Figure 10 Data Head after Discretization

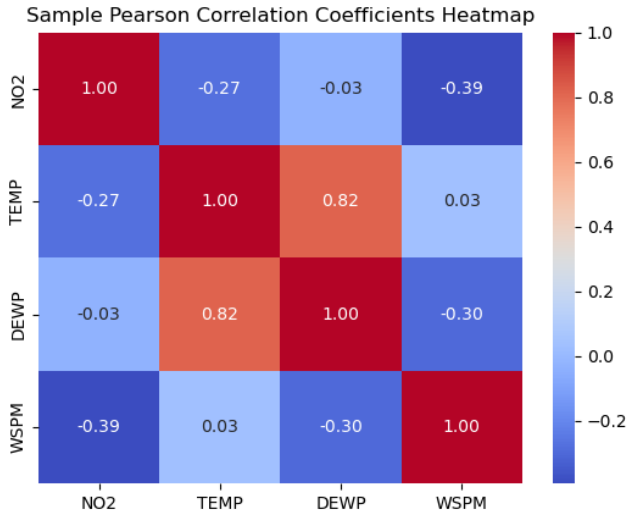


Figure 11Correlation Matrix heatmap

NO2 greater than 10 is labeled as unsafe and those below are labeled as safe. I didn't feel any variable transformations were necessary as the goal of my analysis wouldn't be affected by transforming the variables so I decided to leave all the variables as it is. I also didn't feel like an anomaly detection/outlier analysis was necessary either because there were no outliers present that would significantly affect my results. I felt I didn't need a covariance matrix either because I wasn't looking to see multiple variables relationship with each other and instead just looking at each variable independently and how they interact with each other and their direction and so I decided to create a correlation matrix. According to Figure 11 that shows the Sample Pearson Correlation Coefficients Heatmap, we can see that the two variables that tend to have the most positive relationship and move in the same direction, is the DEWP and TEMP feature, otherwise everything else don't seem to be as

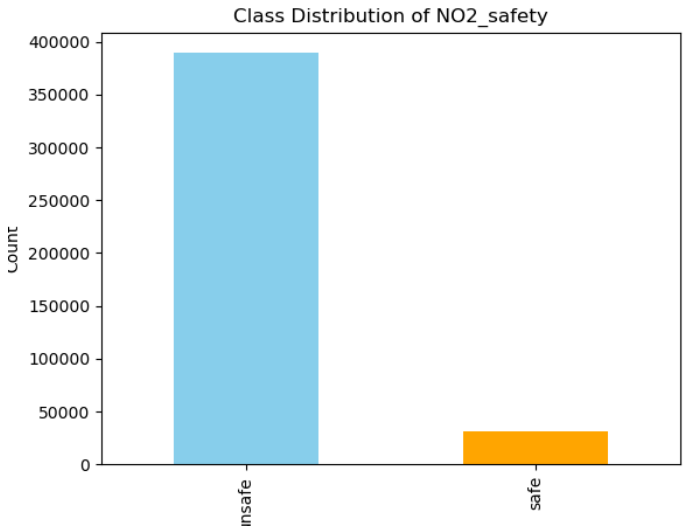


Figure 12 Imbalanced Data

correlated with one another. Finally, to finish off phase I of the project I went into balancing the data because as we can see in Figure 12, the NO2_safety feature is very imbalanced. So we want to balance the data in this feature and the method that was used in order to balance this data was the random over sampling using the imblearn package and importing RandomOverSampler and as a result I was able to get the data balanced as shown in Figure 13.

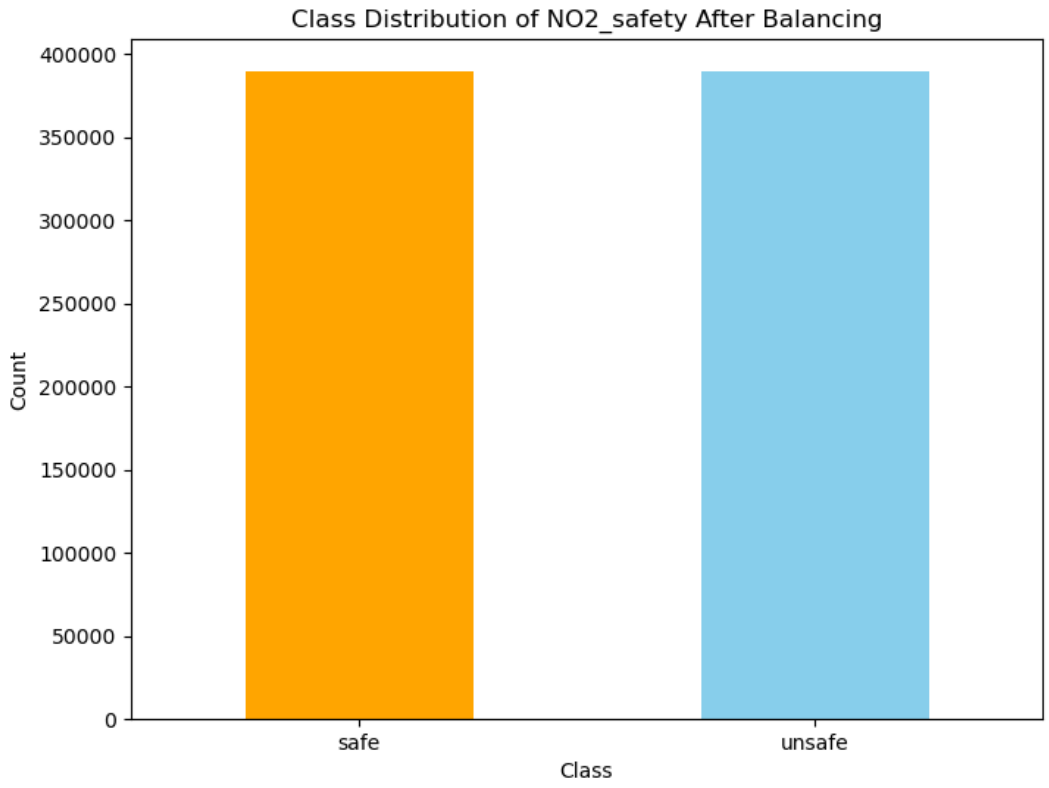


Figure 13 Balanced Data

```
Class Distribution:
unsafe    389416
safe       31352
Name: NO2_safety, dtype: int64
Class distribution after over-sampling: Counter({'safe': 389416, 'unsafe': 389416})
```

Figure 14 Results of oversampling

Phase II: Regression

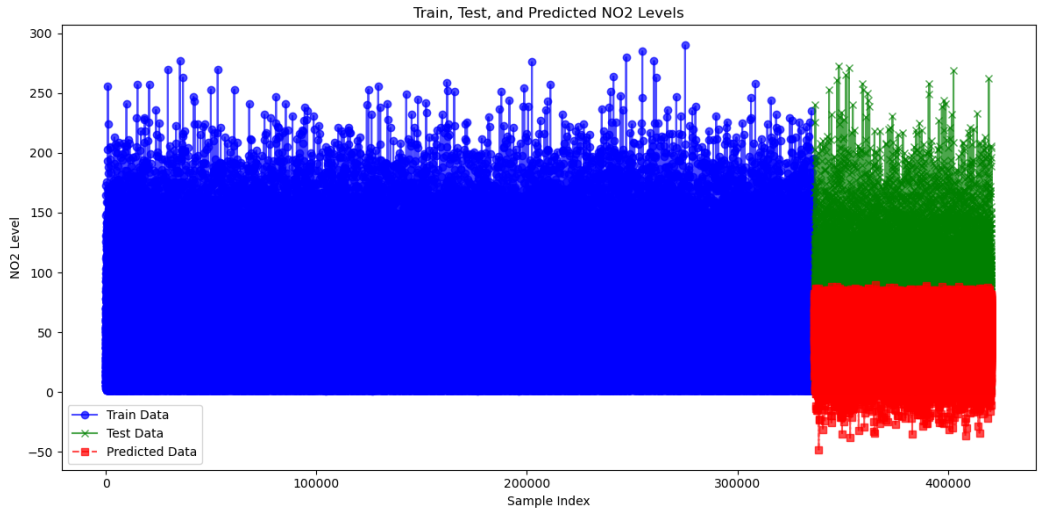
The next phase of the final project term project involved creating a regression model with the features from the data set. The feature that was chosen to be the dependent variable was the NO2 feature which is one of the pollutants and the independent variables after looking at feature importance and adjusting for collinearity are TEMP, DEWP, and WSPM. Here we have our final regression model:

$$NO_2 = TEMPx_1 + DEWP\ x_2 + WSPMx_3$$

Figure 15 Final Regression Model

Below we have a table containing the R-squared, adjusted R-square, AIC, BIC, and MSE from our final regression model shown in Figure 15.

Figure 16 Table of Metrics



	coef	std err	t	P>|t|	[0.025	0.975]
const	83.7196	0.114	736.555	0.000	83.497	83.942
TEMP	-1.5767	0.009	-171.171	0.000	-1.595	-1.559
DEWP	0.7823	0.008	97.924	0.000	0.767	0.798
WSPM	-7.9387	0.051	-157.049	0.000	-8.038	-7.840

Metric	**Value**
R- squared	2.454002e-01
Adjusted R-squared	2.453935e-01
AIC	3.248007e+06
BIC	3.248050e+06
MSE	9.077154e+02

During the Regression analysis I split the data into training and test sets and then plotted it along with the predicted variable which is the NO2 levels. . As shown in Figure 17, the blue portion of the plot is

Figure 2 1 T-test

our train data, the green portion is our test data, and the red portion is our predicted NO2 levels . Using the final regression model, I was able to conduct some T-test analysis . as shown in Figure 18. As you can see, all of the p-values are less than 0.05 which suggests that all of the t values are statistically significant. However, we can see that TEMP and WSPM are negative values which tell us that they have a negative relationship with NO2 levels . With the TEMP feature this was supported in my exploration when it showed NO2 levels were higher the colder the temperature was so this is another evidence of that. The DEWP variable tends to have a positive relationship with the NO2 levels which means the higher the NO2 levels, the higher the DEWP. Next up I also conducted a F-test analysis which tends to measure how the model as a whole is significant or not. In other words, it measures whether the explanatory variables relationship with the dependent variable is significant or not. Figure 19 shows the results of the F-test analysis and as we can see the F-statistic is 3.649e+04 which is a very large number. This signifies to us that the model has a high variance towards our dependent variable or in other words the model is highly significant in explaining our dependent variable . The only thing we need to confirm is the p-

OLS Regression Results

Dep. Variable:	NO2	R-squared:	0.245
Model:	OLS	Adj. R-squared:	0.245
Method:	Least Squares	F-statistic:	3.649e+04
Date:	Fri, 06 Dec 2024	Prob (F-statistic):	0.00
Time:	11:34:03	Log-Likelihood:	-1.6240e+06
No. Observations:	336614	AIC:	3.248e+06
Df Residuals:	336610	BIC:	3.248e+06
Df Model:	3		
Covariance Type:	nonrobust		

Figure 18 F test

Predicted NO2 values:

0	63.691684
1	13.584170
2	80.317964
3	56.191494
4	43.262968
...	...
84149	62.384518
84150	61.024882
84151	59.451738
84152	80.352540
84153	41.157801

Figure 19 Predictions

value which we see in the Prob (F-statistic) that it is 0 meaning that it is less than 0.05, hence telling us that the model has high statistical significance. From Figure 15 we have the final regression model and from that I was able to find some predicted values from this model. As we can see in Figure 20, these are the predicted values of our dependent variable from our final regression model and as we can see the predicted values tend to circulate around a certain range of values such as that 40 – 65 range with a few shooting up to a range of 80. Next was the confidence interval analysis that was conducted, and Figure 21 was the results of that analysis. As we can see, all the confidence intervals have a pretty small range which can tell us that the estimates of the variables are more accurate. We can also see which variables have a negative and positive correlation with the dependent variable which follows the other analysis we have done in earlier sections. Essentially, the confidence interval analysis has successfully reinforced

	Lower Bound	Upper Bound
const	83.496808	83.942363
TEMP	-1.594758	-1.558650
DEWP	0.766601	0.797915
WSPM	-8.037765	-7.839615

Figure 22 Confidence Interval

what was done in previous analysis such as T-test and F-test. For the stepwise regression, I chose a feature to use for this analysis and I chose the feature WSPM because in the random forest analysis from the phase I section of the project it was shown that it was the most important feature. As we can see from the output it shows that WSPM does indeed have a negative relationship with the dependent variable and with a p-value of less than 0.05 it is statistically significant. However, looking at the adjusted R-squared analysis, we have a value of 0.155 which tell us that this particular model only explains a little bit of the variance of the dependent variable and that other features would need to be added in to help in explaining the variance.

OLS Regression Results

Dep. Variable:	NO2	R-squared:	0.155
Model:	OLS	Adj. R-squared:	0.155
Method:	Least Squares	F-statistic:	7.736e+04
Date:	Fri, 06 Dec 2024	Prob (F-statistic):	0.00
Time:	11:34:04	Log-Likelihood:	-2.0536e+06
No. Observations:	420768	AIC:	4.107e+06
Df Residuals:	420766	BIC:	4.107e+06
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	69.5556	0.084	827.407	0.000	69.391	69.720
WSPM	-10.9685	0.039	-278.140	0.000	-11.046	-10.891

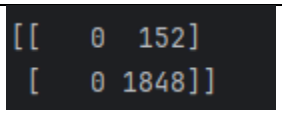
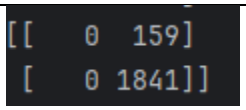
Omnibus:	68511.374	Durbin-Watson:	0.220
Prob(Omnibus):	0.000	Jarque-Bera (JB):	126107.396
Skew:	1.034	Prob(JB):	0.00
Kurtosis:	4.708	Cond. No.	4.21

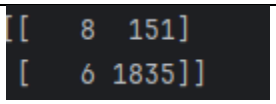
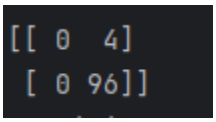
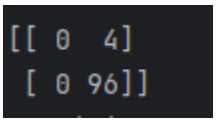
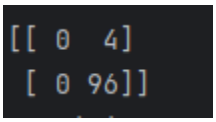
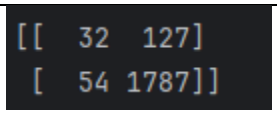
Figure 21 Stepwise and Adjusted R

Phase III: Classification Analysis

There were multiple classification analysis that was conducted with our model in mind and we ended up evaluating the performance of each classifier to see which one we would recommend. The performances of all the classifiers are summarized in the table below:

Classification Performance Table *Figure 23 Classification Table*

	Confusion Matrix	**Precision**	**Sensitivity or Recall**	**Specificity**	**F-score**	**ROC and AUC Curve**	**Stratified K-fold cross validation**
Decision Tree		0.92	1.00	0.00	0.96	0.77	0.73
Logistic Regression		0.92	1.00	0.00	0.96	0.64	0.92

KNN		0.92	1.00	0.05	0.96	0.73	0.92
SVM	Linear Kernel:  Poly Kernel:  Rbf Kernel: 	Linear Kernel: 0.96 Poly Kernel: 0.96 Rbf Kernel: 0.96	Linear Kernel: 1.00 Poly Kernel: 0.96 Rbf Kernel: 0.96	Linear Kernel: 0.00 Poly Kernel: 0.00 Rbf Kernel: 0.00	Linear Kernel: 0.98 Poly Kernel: 0.98 Rbf Kernel: 0.98	Linear Kernel: 0.51 Poly Kernel: 0.65 Rbf Kernel: 0.60	Linear Kernel: 0.94 Poly Kernel: 0.94 Rbf Kernel: 0.94
Naïve Bayes		0.93	0.97	0.20	0.95	0.70	0.91

Random Forest	[[3 156] [2 1839]]	0.92	0.9989	0.02	0.96	0.78	0.92
Neural Network	[[8 754] [9 9229]]	0.927	0.995	0.01	.9599	0.77	0.92

Figure 23: Table to show performances of all the classifiers

We also have all the ROC-AUC curves of all of the classifiers shown below:

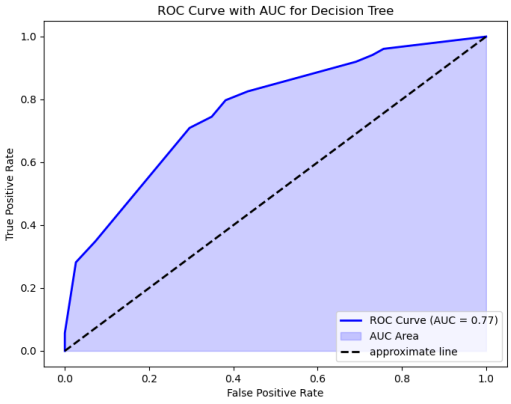


Figure 24 ROC Curve Decision Tree

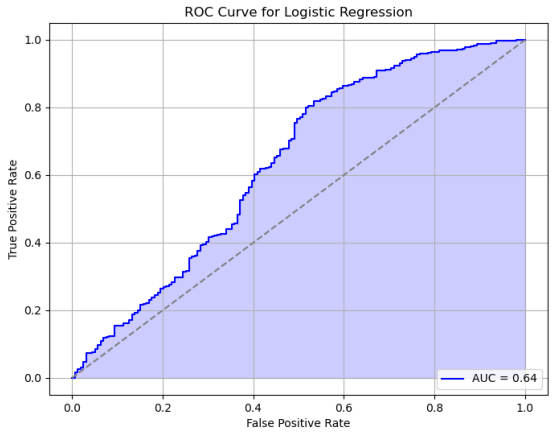


Figure 25 ROC curve Logistic Regression

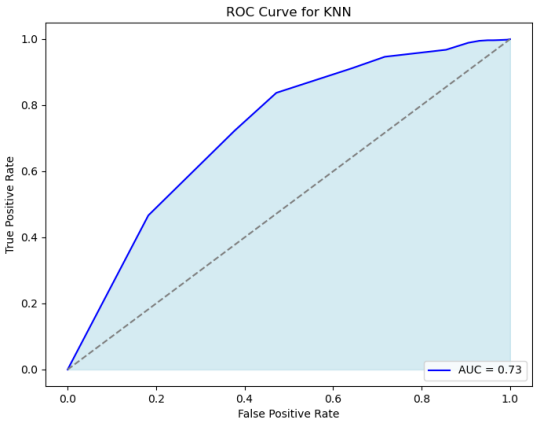


Figure 26 ROC curve KNN

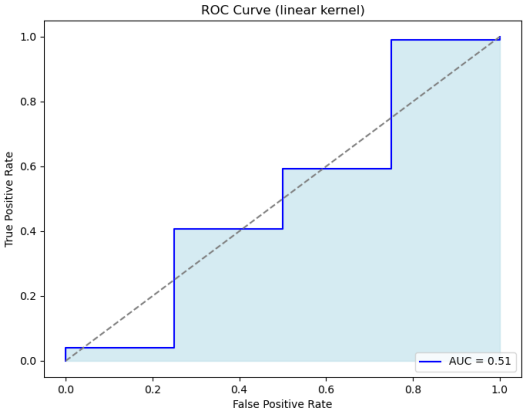


Figure 27 ROC for SVM linear

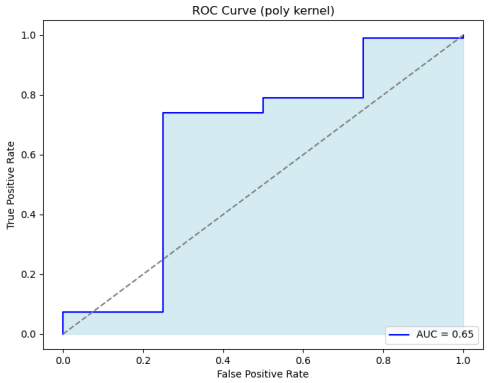


Figure 28 ROC cruve SVM Poly Kernel

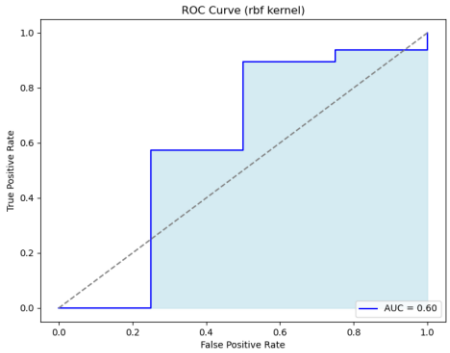


Figure 29 ROC for rbf kernel

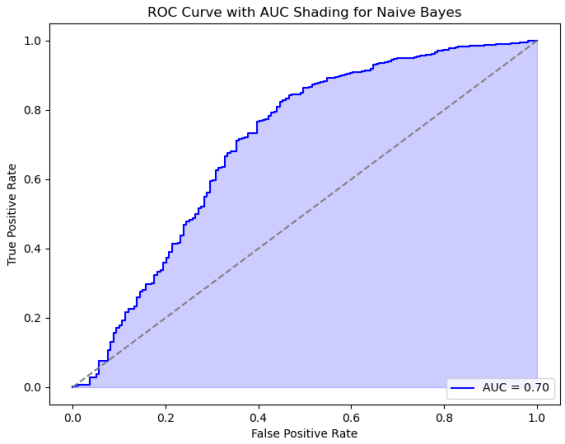


Figure 30 ROC for Naive Bayes

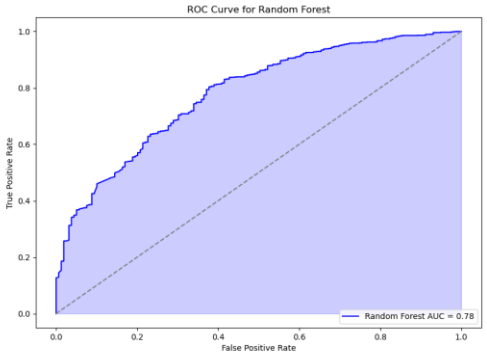


Figure 31 ROC for Random Forest

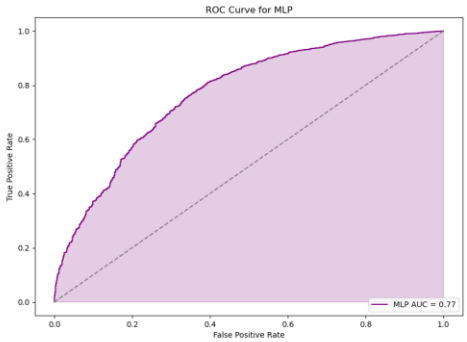


Figure 32 ROC for Neural Network

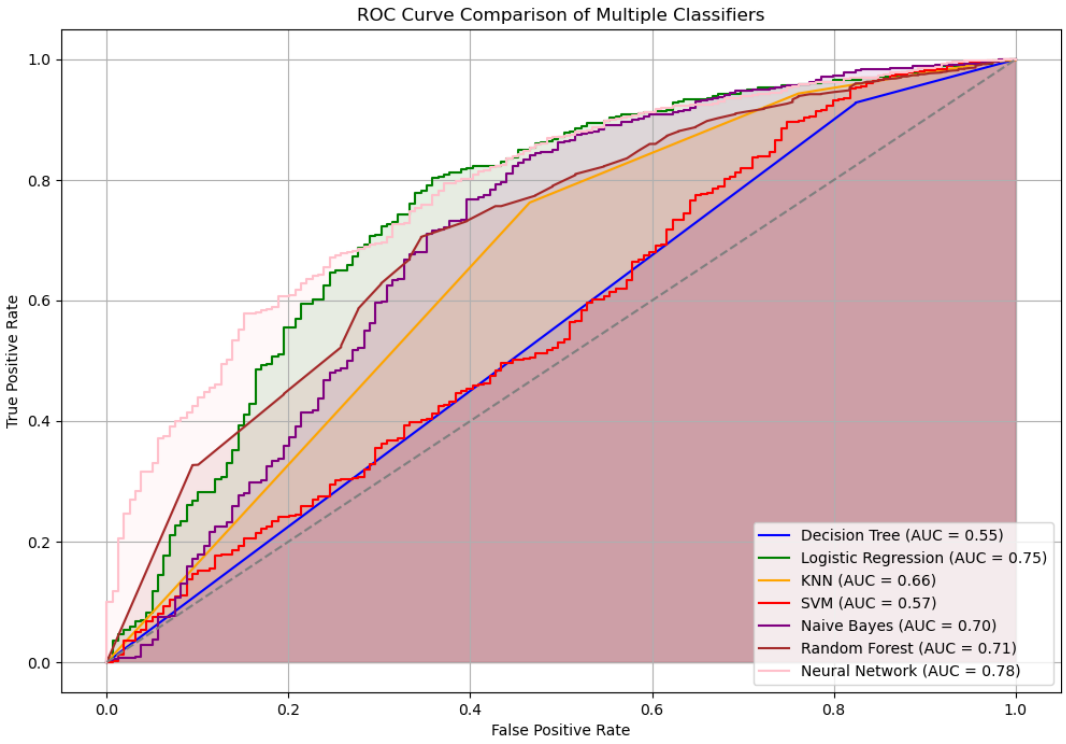


Figure 33 ROC combined plot

As we can see from Figures 24- 30, we have the individual ROC curves for all of the classifiers which also includes each kernel from the SVM classifier. These curves are one way to help in identifying the performance of the classifier and which one is the best one to use for this data set. To help me further in identifying the best performing classifier, I decided to plot all of the ROC curves in one combined plot as we can see in Figure 31 with each color corresponding to a different classifier's ROC curve. We want to look to see which curves are furthest away from the dashed line and as we can see it looks like Decision Tree and SVM classifiers are closest to the dashed lines so they are some of the worst performing classifiers where as we can see that Neural Network and Logistic Regression classifiers are best performing ones because

most of their curves are furthest away from the dashed line with Logistic Regression having the furthest point than any classifier away from the dashed line. This is the first clue, but the second clue is to look at the table that was created to showcase all the metrics of a classifier's performance in figure 23. I will focus on the Neural Network and Logistic Regression classifiers as they were the two curves the furthest away from the dashed line in the plot. We see in Figure 23 that the Neural Network AUC curve score is 0.77 and the Logistic Regression AUC score is 0.64. This score tells us which classifier has the better chance to detect safe and unsafe cases. Another metric to look at is recall because it tells us which classifier on how accurate it is in identifying the unsafe cases and Neural Network has 0.995 and Logistic Regression 1.00 giving the edge to Logistic Regression but not by much. Finally, another metric to look at is the F score which looks at the overall accuracy combining all the metrics and as we can see the two classifiers are about identical. It's pretty close but I would say that the Neural Network classifier performs the best as more of its curve is further away from the dashed line and has a higher AUC score .

Phase IV: Clustering and Association

In this last phase of the final term project, clustering and association rule mining was conducted using several different algorithms that were applied to my data set. The algorithms in question that were applied to the data set were K-mean, DBSCAN, and Apriori algorithms. The K-mean and DBSCAN algorithms helped to identify how many clusters were found when applied to the data set and the Apriori algorithm helped to identify some relationships in the data set where things could be associated with one another. Starting with the K-means algorithm, I had to down sample the data set that I was using because the algorithm was too computationally expensive, and I had to do the same thing for the DBSCAN algorithm. K-means algorithm is a clustering algorithm where it finds the optimal number of clusters from the data. In this algorithm, we are grouping the data based on how similar they are and by the end all the data form clusters with similar aspects among the data points that formed the cluster. One of the main objectives of the K-means algorithm is to minimize the distances between them. When running this algorithm I let it decide the optimal clusters based on a range that was give which was between 2 and 10 . As we can see from Figure 32, the optimal number of clusters that was chosen by the algorithm is 2 and the silhouette score, which measures how similar a point is to its cluster, is 0.464. These results tells us that having two clusters contains data points with the

```
Optimal number of clusters (k): 2
Silhouette Score for optimal k (2): 0.464
```

Figure 34 Clusters from KNN

most similarities . I would say this could make sense because in the data there was a feature called NO2_safety with safe and unsafe

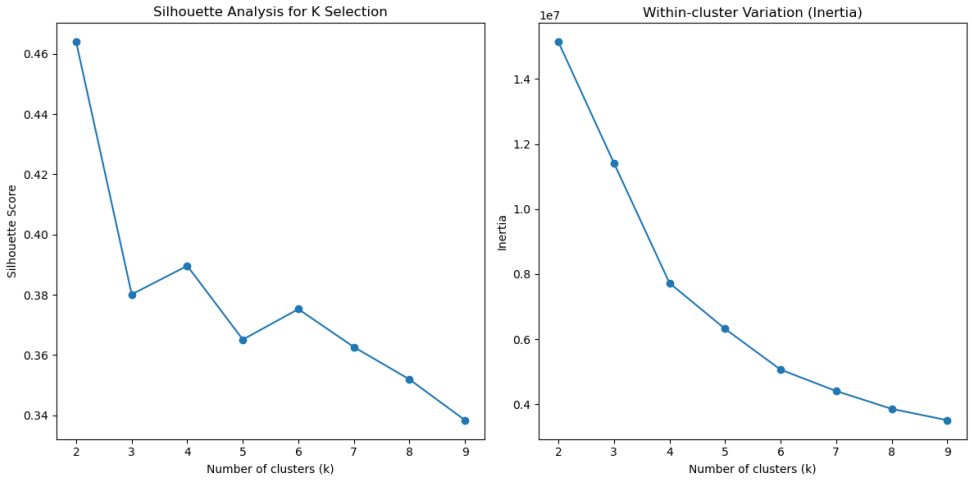


Figure 35 Silhouette

levels of NO2 so the two clusters may comprise of data with these values . The plots in Figure 33 also shows with the Silhouette score that the number of clusters of 2 has the highest silhouette score compared to the other cluster numbers . The silhouette score tells us how similar data points are to the cluster. We can see that the other cluster numbers from 3 to 9 have a much lower silhouette score compared to cluster number of 2. Also, in Figure 33 we are given the Inertia plot which helps to identify the elbow method and as we can see it is below 4 which also follows our optimal cluster being at 2. The next algorithm that was applied to my data set was the DBSCAN algorithm which is another algorithm that investigates the optimal clusters. The difference between DBSCAN and K-means algorithms is that the DBSCAN algorithms deals with the noise in the data better than that of the K-means. So, in other words we are looking to see how noise affects choosing the amount of clusters we may have. According to Figure 34, we see that the DBSCAN algorithm found double the number of clusters than the K-means algorithm. The K-means algorithm found 2 optimal

```
Number of clusters found by DBSCAN: 4
Number of outliers: 181
```

Figure 36 Clusters from DBSCAN

amount of clusters and the DBSCAN algorithm found 4 optimal amount of clusters which tells us that there are four clusters that have data points with similar attributes when we take noise into account. Figure 34 also shows the amount of noise that is present which is the number of outliers in which it mentions that there are 181 present in our data. Figure 35 shows the amount of clusters present in a scatterplot and it shows the clustering based on the temperature in relation with the NO2 levels and we see that one of the cluster labels are actually more of an outlier compared to the rest of the data points present. We notice that the red cluster labels are more of an outlier, but they also have higher levels of NO2 compared to the rest of the data points and other cluster labels. The final algorithm that was applied to our data set was the Apriori algorithm which is a type of association rule mining algorithm. Association rule mining tends to look at what data points go together. Its not a clustering algorithm but it tends to put data points together in association with each other because they have something in common or they complement each other in some way. First, I had to discretize all of my features so that they can be placed into bins. For example, for the temperature feature I discretized them into a low, medium, and high temperature binning. After running the Apriori algorithm, it appears that we

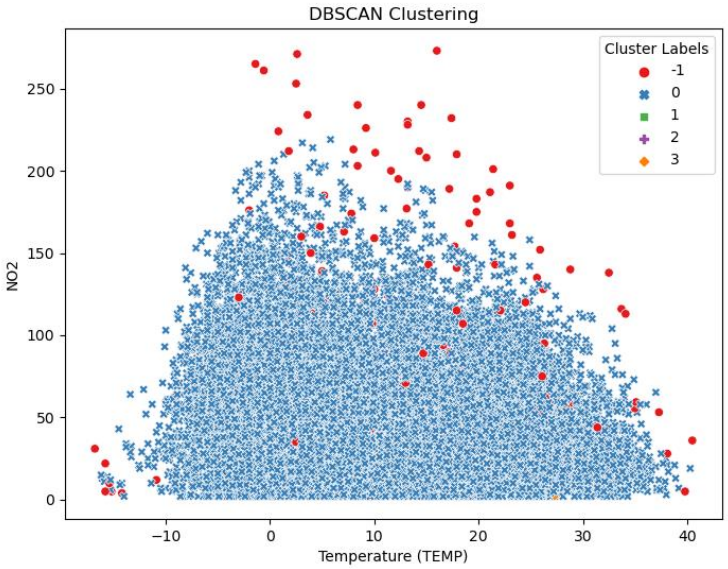


Figure 37 DBSCAN clustering and outliers

Number of association rules found: 72

	antecedents	consequents	support	confidence	lift
0	(NO2_Discretized_High)	(TEMP_Discretized_Warm)	0.053799	0.398333	1.346188
1	(TEMP_Discretized_Warm)	(NO2_Discretized_High)	0.053799	0.182087	1.346188
2	(WSPM_Discretized_Low)	(NO2_Discretized_High)	0.130022	0.137204	1.015871
3	(NO2_Discretized_High)	(WSPM_Discretized_Low)	0.130022	0.962695	1.015871
4	(WSPM_Discretized_Low)	(NO2_Discretized_Very High)	0.117157	0.123629	1.027068
5	(NO2_Discretized_Very High)	(WSPM_Discretized_Low)	0.117157	0.973386	1.027068
6	(WSPM_Discretized_Low)	(NO2_Discretized_Extremely High)	0.102883	0.108566	1.022408
7	(NO2_Discretized_Extremely High)	(WSPM_Discretized_Low)	0.102883	0.968890	1.022408
8	(DEWP_Discretized_Very Low)	(TEMP_Discretized_Very Cold)	0.054748	0.255876	1.643280
9	(TEMP_Discretized_Very Cold)	(DEWP_Discretized_Very Low)	0.054748	0.351598	1.643280
10	(TEMP_Discretized_Cold)	(DEWP_Discretized_Very Low)	0.091554	0.373448	1.745480
11	(DEWP_Discretized_Very Low)	(TEMP_Discretized_Cold)	0.091554	0.427980	1.745480
12	(DEWP_Discretized_Low)	(TEMP_Discretized_Cold)	0.052806	0.286198	1.167399
13	(TEMP_Discretized_Cold)	(DEWP_Discretized_Low)	0.052806	0.215394	1.167399
14	(DEWP_Discretized_Low)	(TEMP_Discretized_Mild)	0.080512	0.436362	1.822985
15	(TEMP_Discretized_Mild)	(DEWP_Discretized_Low)	0.080512	0.336355	1.822985
16	(DEWP_Discretized_Medium)	(TEMP_Discretized_Mild)	0.087644	0.318477	1.330495
17	(TEMP_Discretized_Mild)	(DEWP_Discretized_Medium)	0.087644	0.366151	1.330495
18	(TEMP_Discretized_Warm)	(DEWP_Discretized_Medium)	0.153978	0.521147	1.893709
19	(DEWP_Discretized_Medium)	(TEMP_Discretized_Warm)	0.153978	0.559515	1.893709
20	(TEMP_Discretized_Warm)	(DEWP_Discretized_High)	0.078611	0.266063	2.723086
21	(DEWP_Discretized_High)	(TEMP_Discretized_Warm)	0.078611	0.804539	2.723086
22	(WSPM_Discretized_Low)	(TEMP_Discretized_Warm)	0.283161	0.298801	1.011310
23	(TEMP_Discretized_Warm)	(WSPM_Discretized_Low)	0.283161	0.958374	1.011310
24	(WSPM_Discretized_Low)	(TEMP_Discretized_Hot)	0.059895	0.863204	1.840619
25	(TEMP_Discretized_Hot)	(WSPM_Discretized_Low)	0.059895	0.986148	1.040619
26	(WSPM_Discretized_Low)	(DEWP_Discretized_Very Low)	0.203369	0.214602	1.002995

Figure 38 Results of Apriori

got about 72 association rules that were found in our data set. Each rule has an antecedent which is something that is found in the data and a consequent which is something that is found in combination with the antecedent. Lets use the first entry from Figure 36 as an example of interpreting the association rule mining here. So we are saying that if the NO2 level is high then there is an approximately 40% confidence that the temperature will also be warm and it supported by 5% of our data. The lift for this rule is 1.35 and according to what we learned in Lecture 15, the two items are more likely to occur together since the lift is greater than 1.[Jafari 2024].

Recommendations

What I learned from this project is just how much detail goes into a machine learning based project. For example, the feature engineering aspect is absolutely crucial to get right before continuing on to the rest of the project. I found that the first time I was working through this project, I had to go back to do some more feature engineering in order to get some of the classification and clustering algorithms such as going back and doing some discretization for the rest of the features in order to put them in different bins so that I am able to successfully run Apriori algorithm. I felt that the phase of the project that took me the longest to implement was the classification phase because it took a while to solve the issues of computational expense as

some of these classifications can be expensive with very large data sets. Overall, I felt like I gained valueable experience in the whole machine learning project process. Out of all the classifications that were run on the data set, the classifications that performed the best were the Neural Network and the Logistic Regression. This was supported by the fact when I graphed all of the classifcations ROC curves under the same plot, it was shown that the Neural Network plot and the Logistic Regression plot were furthest from the dashed line which signifies better classifiers. This can be seen in Figure 31. The Neural Network AUC curve score is 0.77 and the Logistic Regression AUC score is 0.64. These scores were some of the highest compared to the other classifiers and it tells us which classifier has the better chance to detect safe and unsafe cases. I also looked at the recall score because it tells us which classifier was more accurate in identifying the unsafe cases and Neural Network has 0.995 and Logistic Regression 1.00 giving the edge to Logistic Regression but not by much and these were also some of the highest scores compared to other classifiers. Finally, the F score which looks at the overall accuracy combining all the metrics and as we can see the two classifiers are about identical between Neural Network and Logistic Regression. All the metrics were close, but I decided that the Neural Network classifier performs the best as more of its curve is further away from the dashed line and has a higher AUC score. I believe some ways that the classifications performance could improve is perhaps instead of using the entire data set of all 12 monitoring stations, I could use just two to three of the stations. Also, another possibility is to use other air pollutants as the dependent variables or look at how only the air pollutants affect each other and put those through the classifications. Finally, another thing I could have done was look at the time based features in the data set and that could have improved some of the classifications performances that do better

with time based data . All these strategies I just mentioned could be considerations for future work. My target variable was the NO2 feature, which was one of the air pollutants present in the data set and some of the features I decided to associate with it after going through feature selection and removing features due to collinearity issues, I ended up with having the TEMP, DEWP, and WSPM features which are temperature, dew point, and wind speed associated with the target variable. The number of clusters that were found in this feature space when running the K-means algorithm was 2 as the optimal number of clusters and when I ran the DBSCAN algorithm, which took into account the noise present in the data set, it gave me 4 clusters as the optimal amount of clusters in the feature space.

Appendix A – Phase I Feature Engineering/ EDA Code

```
import pandas as pd

from sklearn.ensemble import RandomForestRegressor

import matplotlib.pyplot as plt

import seaborn as sns


# List of files to load

files = [

'PRSA_Data_Changping_20130301-20170228.csv',

'PRSA_Data_Shunyi_20130301-20170228.csv',

'PRSA_Data_Aotizhongxin_20130301-20170228.csv',

'PRSA_Data_Dingling_20130301-20170228.csv',

'PRSA_Data_Dongsi_20130301-20170228.csv',

'PRSA_Data_Guanyuan_20130301-20170228.csv',

'PRSA_Data_Gucheng_20130301-20170228.csv',
```

```
'PRSA_Data_Huairou_20130301-20170228.csv',
'PRSA_Data_Nongzhanguan_20130301-20170228.csv',
'PRSA_Data_Tiantan_20130301-20170228.csv',
'PRSA_Data_Wanliu_20130301-20170228.csv',
'PRSA_Data_Wanshouxigong_20130301-20170228.csv'
]

# Initialize an empty list to store dataframes
df_list = []

# Read and clean datasets in batches
for file in files:
# Load each dataset in a chunk
chunk = pd.read_csv(file)

# Handle missing values for this chunk
chunk = chunk.fillna(chunk.mean())

# Append the cleaned chunk to the list
df_list.append(chunk)

# Concatenate the list of cleaned dataframes
```

```
merged_data_cleaned = pd.concat(df_list, ignore_index=True)

# Check for any remaining missing values
missing_values = merged_data_cleaned.isnull().sum()

# Display missing values count and final merged dataframe
print("Missing values after cleaning:")
print(missing_values)
# Set pandas to display all columns
pd.set_option('display.max_columns', None)
print(merged_data_cleaned)

#### EDA ##########################################################
pollutants = ['PM2.5', 'PM10', 'SO2', 'NO2', 'CO', 'O3']
merged_data_cleaned[pollutants].hist(bins=30, figsize=(15, 10))
plt.show()

# Create a 'season' column based on the month
merged_data_cleaned['season'] = merged_data_cleaned['month'].apply(lambda x:
```

```
'Winter' if x in [12, 1, 2] else
'Spring' if x in [3, 4, 5] else
'Summer' if x in [6, 7, 8] else
'Autumn')


plt.figure(figsize=(12, 6))
sns.boxplot(x='season', y='NO2', data=merged_data_cleaned)
plt.title('NO2 Levels Across Seasons')
plt.show()


### Scatterplot ###
# Scatterplot of NO2 vs WSPM
sns.scatterplot(x='TEMP', y='NO2', data=merged_data_cleaned)
plt.title('Scatterplot of NO2 vs Temperature (TEMP)')
plt.show()


##########################Random Forest ########################################
```

```
merged_data_cleaned = merged_data_cleaned.drop('season', axis=1)
```

```
# Select features and target variable
features = ['TEMP', 'PRES', 'DEWP', 'RAIN', 'WSPM']
target = 'NO2'

# Define X (features) and y (target)
X = merged_data_cleaned[features]
y = merged_data_cleaned[target]

# Train a Random Forest Regressor model with parallelism and other optimizations
rf = RandomForestRegressor(
    n_estimators=100,          # Number of trees (you can experiment with this)
    random_state=42,           # For reproducibility
    n_jobs=-1,                 # Use all available CPU cores for parallelism
    max_depth=10,              # Limit tree depth to reduce complexity and speed up training
    min_samples_split=10,      # Increase min samples required to split nodes
```

```
    warm_start=True           # Reuse previous models if needed
)

# Fit the model
rf.fit(X, y)

# Get feature importance from the trained model
importance = rf.feature_importances_

# Create a DataFrame to visualize the importance of each feature
importance_df = pd.DataFrame({
    'Feature': features,
    'Importance': importance
}).sort_values(by='Importance', ascending=False)

# Plotting feature importance
plt.figure(figsize=(10, 6))
plt.barh(importance_df['Feature'], importance_df['Importance'])
plt.xlabel('Importance')
plt.title('Feature Importance from Random Forest')
plt.show()
```

```
# Print out the sorted feature importance
print(importance_df)
```

```
######## Discretization ############
# Define a threshold for NO2 safety (let's use 10 µg/m³ as an example)
threshold_no2 = 10

# Label the values based on the threshold
merged_data_cleaned['NO2_safety'] = merged_data_cleaned['NO2'].apply(
lambda x: 'safe' if x <= threshold_no2 else 'unsafe'
)

# Now 'NO2_safe' will contain 'safe' or 'unsafe' instead of 1 or 0
print(merged_data_cleaned.head())
```

```
################ Subset the Data set #############
# Define the list of features you want to keep
subset_columns = ['NO2', 'TEMP', 'PRES', 'DEWP', 'wd', 'WSPM', 'station', 'NO2_safety']
```

```
# Create a subset of the dataset
subset_data = merged_data_cleaned[subset_columns]

# Display the first few rows of the subset
print(subset_data.head())

############### VIF ####################################################
from statsmodels.stats.outliers_influence import variance_inflation_factor
from statsmodels.tools.tools import add_constant

# Add constant to the features to account for the intercept
X = subset_data[['NO2', 'TEMP', 'PRES', 'DEWP', 'WSPM']]

# Calculate the VIF for each feature
vif_data = pd.DataFrame()
vif_data['Feature'] = X.columns
vif_data['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]

# Display the VIF
print(vif_data)
```

```
####### Second Subset of the Data ###################################

# Define the list of features you want to keep

subset_columns2 = ['NO2', 'TEMP', 'DEWP', 'wd', 'WSPM', 'station', 'NO2_safety']


# Create a subset of the dataset

subset_data2 = merged_data_cleaned[subset_columns2]


# Display the first few rows of the subset

print(subset_data2.head())


############### VIF ####################################################

from statsmodels.stats.outliers_influence import variance_inflation_factor

from statsmodels.tools.tools import add_constant


# Add constant to the features to account for the intercept

X = subset_data2[['NO2', 'TEMP', 'DEWP', 'WSPM']]


# Calculate the VIF for each feature

vif_data2 = pd.DataFrame()

vif_data2['Feature'] = X.columns

vif_data2['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
```

```
# Display the VIF
print(vif_data2)


############### Correlation Matrix ###################################
numerical_columns = subset_data2.select_dtypes(include=['float64', 'int64'])
correlation_matrix = numerical_columns.corr()
sns.heatmap(correlation_matrix, annot=True, fmt=".2f", cmap='coolwarm', cbar=True)
plt.title('Sample Pearson Correlation Coefficients Heatmap')
plt.show()


###### Balanced and imbalanced data ##################################
# Check the distribution of the target variable
class_distribution = subset_data2['NO2_safety'].value_counts()

print("Class Distribution:")
print(class_distribution)

# Visualize the class distribution
import matplotlib.pyplot as plt
```

```
class_distribution.plot(kind='bar', color=['skyblue', 'orange'])

plt.title('Class Distribution of NO2_safety')

plt.xlabel('Class')

plt.ylabel('Count')

plt.show()

###### Balancing Data ####

import imblearn


from imblearn.over_sampling import RandomOverSampler

from collections import Counter


# Define the feature matrix (X) and target vector (y)

X = subset_data2.drop('NO2_safety', axis=1)

y = subset_data2['NO2_safety']


# Perform random over-sampling

oversampler = RandomOverSampler(random_state=42)

X_resampled, y_resampled = oversampler.fit_resample(X, y)


# Check the new class distribution
```

```
print("Class distribution after over-sampling:", Counter(y_resampled))


### Show plot ####

# Visualize the balanced class distribution

balanced_class_distribution = pd.Series(y_resampled).value_counts()


# Plot the distribution after balancing

plt.figure(figsize=(8, 6))

balanced_class_distribution.plot(kind='bar', color=['orange', 'skyblue'])

plt.title('Class Distribution of NO2_safety After Balancing')

plt.xlabel('Class')

plt.ylabel('Count')

plt.xticks(ticks=[0, 1], labels=balanced_class_distribution.index, rotation=0)

plt.show()
```

Appendix B – Phase II: Regression Code

```
##############################################################################

##########

############################ Regression

################################################
```

```
##############################################################################
##########
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import statsmodels.api as sm
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from statsmodels.regression.linear_model import OLS
from itertools import combinations


# get variables to use for x and y
X_regs = subset_data2[['TEMP', 'DEWP', 'WSPM']]
y_regs = subset_data2['NO2']


print(X_regs.info())
print(y_regs.info())



# Add a constant for the intercept
X_with_const = sm.add_constant(X_regs)
```

```
# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X_with_const, y_regs, test_size=0.2,
random_state=42)

# Reset indices to avoid misalignment
X_train, X_test = X_train.reset_index(drop=True), X_test.reset_index(drop=True)
y_train, y_test = y_train.reset_index(drop=True), y_test.reset_index(drop=True)

# Train the regression model using statsmodels
air_NO2_model = sm.OLS(y_train, X_train).fit()

# Make predictions
y_pred_train = air_NO2_model.predict(X_train)
y_pred_test = air_NO2_model.predict(X_test)

# Calculate R-squared, Adjusted R-squared, AIC, BIC, and MSE
r_squared = air_NO2_model.rsquared
adj_r_squared = air_NO2_model.rsquared_adj
aic = air_NO2_model.aic
bic = air_NO2_model.bic
mse = mean_squared_error(y_test, y_pred_test)
```

```
# Display the predicted values
print("Predicted NO2 values:")
print(y_pred_test)

# Display the results
results = pd.DataFrame({
'Metric': ['R-squared', 'Adjusted R-squared', 'AIC', 'BIC', 'MSE'],
'Value': [r_squared, adj_r_squared, aic, bic, mse]
})
print(results)

# Display the summary for T-tests and F-test
print(air_NO2_model.summary())

# Display confidence intervals
confidence_intervals = air_NO2_model.conf_int()
confidence_intervals.columns = ['Lower Bound', 'Upper Bound']
print(confidence_intervals)
```

```
# function for stepwise regression
def stepwise_regression(X, y, initial_features=[], criterion='AIC'):

    remaining_features = list(X.columns)
    selected_features = list(initial_features)
    best_features = None
    best_model = None
    best_criterion = float('inf')

    while remaining_features:
        candidate_models = []

        # Add one feature at a time
        for feature in remaining_features:
            model_features = selected_features + [feature]
            X_selected = sm.add_constant(X[model_features])  # Add constant
            model = OLS(y, X_selected).fit()
            crit_value = model.aic if criterion == 'AIC' else model.bic
            candidate_models.append((model, model_features, crit_value))

        # Find the best candidate model
```

```
candidate_models.sort(key=lambda x: x[2])  # Sort by criterion
best_candidate_model, best_candidate_features, candidate_criterion = candidate_models[0]

# Stop if the criterion does not improve
if candidate_criterion < best_criterion:
    best_criterion = candidate_criterion
    best_model = best_candidate_model
    best_features = best_candidate_features
    selected_features = best_candidate_features[1:]  # Remove constant
    remaining_features = [f for f in remaining_features if f not in selected_features]
else:
    break

return best_model, best_features

# run stepwise regression
stepwise_model, stepwise_features = stepwise_regression(X_regs, y_regs, criterion='AIC')

# Display the final model summary
print("Selected Features:", stepwise_features)
print(stepwise_model.summary())
```

```
# Calculate adjusted R-squared
adjusted_r_squared = stepwise_model.rsquared_adj
print("Adjusted R-squared:", adjusted_r_squared)

# Visualize the train, test, and predicted values
plt.figure(figsize=(12, 6))

# Plot train data
plt.plot(range(len(y_train)), y_train, label="Train Data", color='blue', marker='o', linestyle='-',
alpha=0.7)

# Plot test data
plt.plot(range(len(y_train), len(y_train) + len(y_test)), y_test, label="Test Data", color='green',
marker='x', linestyle='-', alpha=0.7)

# Plot predictions on the test set
plt.plot(range(len(y_train), len(y_train) + len(y_test)), y_pred_test, label="Predicted Data",
color='red', marker='s', linestyle='--', alpha=0.7)

# Add labels, legend, and title
```

```
plt.title("Train, Test, and Predicted NO2 Levels")

plt.xlabel("Sample Index")

plt.ylabel("NO2 Level")
```

```
plt.legend()

plt.tight_layout()

# Display the plot

plt.show()
```

Appendix C: Phase III: Classification

```
##############################################################################
######
############################ Classification Analysis
###################################
##############################################################################
######


################## Decision Tree Classifier
##########################################
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split, GridSearchCV, StratifiedKFold,
cross_val_score
from sklearn.metrics import (
    confusion_matrix, classification_report, roc_auc_score, roc_curve, auc
)
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
```

```
import numpy as np

# Sampling the dataset
subset_data2_sampled = subset_data2.sample(n=10000, random_state=42)
X_sample = subset_data2_sampled[['TEMP', 'DEWP', 'WSPM']]
y_sample = subset_data2_sampled['NO2_safety'].map({'safe': 0, 'unsafe': 1})  # Convert to
numeric for classification

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X_sample, y_sample, test_size=0.2, random_state=42, stratify=y_sample
)

# 1. Decision Tree with Pre-pruning
dt_prepruned = DecisionTreeClassifier(
    criterion='gini', max_depth=5, min_samples_split=10, random_state=42
)
dt_prepruned.fit(X_train, y_train)

# 2. Post-pruning with Cost Complexity Pruning
path = dt_prepruned.cost_complexity_pruning_path(X_train, y_train)
ccp_alphas = path.ccp_alphas[:-1]  # Exclude the last alpha (results in an empty tree)
```

```
# Train trees with different alphas
trees = []
for alpha in ccp_alphas:
    tree = DecisionTreeClassifier(random_state=42, ccp_alpha=alpha)
    tree.fit(X_train, y_train)
    trees.append(tree)

# alpha with the highest cross-validated score
kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
tree_scores = [cross_val_score(tree, X_train, y_train, cv=kfold).mean() for tree in trees]
optimal_alpha = ccp_alphas[np.argmax(tree_scores)]

# Train final tree with the best alpha
dt_postpruned = DecisionTreeClassifier(random_state=42, ccp_alpha=optimal_alpha)
dt_postpruned.fit(X_train, y_train)

# Hyperparameter Tuning with GridSearchCV
param_grid = {
    'criterion': ['gini', 'entropy'],
    'splitter': ['best', 'random'],
    'max_depth': [5, 10, None],
```

```
    'min_samples_split': [2, 5, 10],
    'max_features': [None, 'sqrt', 'log2'],
    'ccp_alpha': [0, optimal_alpha]
}

grid_search = GridSearchCV(
    DecisionTreeClassifier(random_state=42),
    param_grid,
    cv=kfold,
    scoring='roc_auc',
    n_jobs=-1
)
grid_search.fit(X_train, y_train)
best_tree = grid_search.best_estimator_

# 4. Evaluation Metrics
def evaluate_model(model, X_eval, y_eval, title="Confusion Matrix"):
    # Predictions and probabilities
    y_pred = model.predict(X_eval)
    y_proba = model.predict_proba(X_eval)[:, 1]

    # Confusion matrix
```

```
cm = confusion_matrix(y_eval, y_pred)
tn, fp, fn, tp = cm.ravel()
print(cm)

# Metrics
precision = tp / (tp + fp)
recall = tp / (tp + fn)
specificity = tn / (tn + fp)
fscore = 2 * (precision * recall) / (precision + recall)
roc_auc = roc_auc_score(y_eval, y_proba)

print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"Specificity: {specificity:.2f}")
print(f"F-score: {fscore:.2f}")
print(f"ROC-AUC: {roc_auc:.2f}")



# ROC Curve with AUC
fpr, tpr, _ = roc_curve(y_eval, y_proba)
plt.figure(figsize=(8, 6))
```

```
    plt.plot(fpr, tpr, label=f"ROC Curve (AUC = {roc_auc:.2f})", color='blue', lw=2)
    plt.fill_between(fpr, tpr, alpha=0.2, color='blue', label="AUC Area")
    plt.plot([0, 1], [0, 1], "k--", lw=2, label="approximate line")  # Diagonal line
    plt.title("ROC Curve with AUC for Decision Tree")
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.legend(loc="lower right")
    plt.show()


# Evaluate the best tree
print("Evaluation for the Best Decision Tree")
evaluate_model(best_tree, X_test, y_test)

# Stratified K-Fold Cross-Validation for Best Tree
cv_scores = cross_val_score(best_tree, X_train, y_train, cv=kfold, scoring='roc_auc')
print(f"Stratified K-Fold ROC-AUC scores: {cv_scores}")
print(f"Mean ROC-AUC: {cv_scores.mean():.2f}, Std: {cv_scores.std():.2f}")

###################### Logistic Regression
#########################################################################
from sklearn.linear_model import LogisticRegression
```

```
from sklearn.model_selection import GridSearchCV, StratifiedKFold, train_test_split,
cross_val_score
from sklearn.metrics import confusion_matrix, precision_score, recall_score, f1_score,
roc_auc_score, roc_curve
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

# Sample the dataset
subset_data2_sampled = subset_data2.sample(n=10000, random_state=42)

# Prepare your data that was sampled by selecting features and target variable
X_sample = subset_data2_sampled[['TEMP', 'DEWP', 'WSPM']]  # Select relevant features
y_sample = subset_data2_sampled['NO2_safety'].map({'safe': 0, 'unsafe': 1})  # Convert target to
numeric

#  Split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X_sample, y_sample, test_size=0.2,
random_state=42)

# Set up Logistic Regression
log_reg = LogisticRegression(max_iter=1000)
```

```
# Define parameter grid for GridSearchCV
param_grid = {
    'C': [0.01, 0.1, 1, 10, 100],  # Regularization strength
    'penalty': ['l2'],  # Regularization type
    'solver': ['liblinear', 'saga']  # Solvers
}

# Grid search with Stratified K-Fold cross-validation
grid_search = GridSearchCV(log_reg, param_grid, cv=StratifiedKFold(n_splits=5),
scoring='accuracy')
grid_search.fit(X_train, y_train)

# best model from GridSearchCV
best_log_reg = grid_search.best_estimator_

# predictions
y_pred = best_log_reg.predict(X_test)
y_pred_prob = best_log_reg.predict_proba(X_test)[:, 1]  # Probability for ROC/AUC

print("Evaluate Logistic Regression")
# Evaluate performance metrics
```

```
# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print(conf_matrix)

# Precision, Recall, Specificity, F-Score
precision = precision_score(y_test, y_pred, pos_label=1)  # 'unsafe' is now 1
recall = recall_score(y_test, y_pred, pos_label=1)  # 'unsafe' is now 1
f_score = f1_score(y_test, y_pred, pos_label=1)  # 'unsafe' is now 1
specificity = recall_score(y_test, y_pred, pos_label=0)  # Specificity is recall for the 'safe' class
(0)

# ROC Curve and AUC
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob, pos_label=1)  # 'unsafe' is now 1
roc_auc = roc_auc_score(y_test, y_pred_prob)

# Display metrics
print(f"Precision: {precision:.2f}")
print(f"Recall (Sensitivity): {recall:.2f}")
print(f"Specificity: {specificity:.2f}")
print(f"F-Score: {f_score:.2f}")
print(f"AUC: {roc_auc:.2f}")
```

```
# Plot ROC Curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', label=f'AUC = {roc_auc:.2f}')
plt.fill_between(fpr, tpr, color='blue', alpha=0.2)  # Shading the AUC area
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.title('ROC Curve for Logistic Regression')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()

# Cross-validation
cross_val_scores = cross_val_score(best_log_reg, X_sample, y_sample,
cv=StratifiedKFold(n_splits=5), scoring='accuracy')
print(f"Cross-Validation Scores: {cross_val_scores}")
print(f"Average Cross-Validation Accuracy: {np.mean(cross_val_scores):.2f}")

################################## KNN
#############################################################################
####
from sklearn.neighbors import KNeighborsClassifier
```

```
from sklearn.model_selection import StratifiedKFold, train_test_split, cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import (accuracy_score, confusion_matrix,
                             precision_score, recall_score, f1_score, roc_auc_score, roc_curve)
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

# Prepare Data
subset_data2_sampled = subset_data2.sample(n=10000, random_state=42)
X_sample = subset_data2_sampled[['TEMP', 'DEWP', 'WSPM']]
y_sample = subset_data2_sampled['NO2_safety'].map({'safe': 0, 'unsafe': 1})  # Convert target to
numeric

# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X_sample, y_sample, test_size=0.2,
random_state=42)

# Standardize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
# Find Optimum K using the Elbow Method
error_rates = []
k_range = range(1, 21)

for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train_scaled, y_train)
    y_pred = knn.predict(X_test_scaled)
    error_rate = 1 - accuracy_score(y_test, y_pred)
    error_rates.append(error_rate)

# Plot the Elbow Method
plt.figure(figsize=(8, 6))
plt.plot(k_range, error_rates, marker='o', linestyle='-', color='blue')
plt.title('Elbow Method for Optimal K')
plt.xlabel('Number of Neighbors (K)')
plt.ylabel('Error Rate')
plt.xticks(k_range)
plt.grid(True)
plt.show()
```

```
# Train and Evaluate the Model with Optimal K
optimal_k = error_rates.index(min(error_rates)) + 1
print(f"Optimal K: {optimal_k}")

knn_best = KNeighborsClassifier(n_neighbors=optimal_k)
knn_best.fit(X_train_scaled, y_train)
y_pred_best = knn_best.predict(X_test_scaled)
y_pred_prob_best = knn_best.predict_proba(X_test_scaled)[:, 1]

print("Evaluation for KNN")
# Metrics and Visualizations
# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred_best)
print(conf_matrix)

# Precision, Recall (Sensitivity), Specificity, F-Score
precision = precision_score(y_test, y_pred_best)
recall = recall_score(y_test, y_pred_best)
specificity = recall_score(y_test, y_pred_best, pos_label=0)
f_score = f1_score(y_test, y_pred_best)

# ROC Curve and AUC
```

```
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob_best)
roc_auc = roc_auc_score(y_test, y_pred_prob_best)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', label=f'AUC = {roc_auc:.2f}')
plt.fill_between(fpr, tpr, color='lightblue', alpha=0.5)  # Shade the AUC curve
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.title('ROC Curve for KNN')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc='lower right')
plt.show()

# Display Metrics
print(f"Precision: {precision:.2f}")
print(f"Recall (Sensitivity): {recall:.2f}")
print(f"Specificity: {specificity:.2f}")
print(f"F-Score: {f_score:.2f}")
print(f"AUC: {roc_auc:.2f}")

print("Evaluation of KNN: ")
# Stratified K-Fold Cross-Validation
```

```
skf = StratifiedKFold(n_splits=5)
cv_scores = cross_val_score(knn_best, X_sample, y_sample, cv=skf, scoring='accuracy')

print(f"Cross-Validation Scores: {cv_scores}")
print(f"Average Cross-Validation Accuracy: {np.mean(cv_scores):.2f}")

################################### SVM
##############################################################################
##
from sklearn.svm import SVC
from sklearn.metrics import (confusion_matrix, precision_score, recall_score, f1_score,
roc_auc_score, roc_curve)
from sklearn.model_selection import StratifiedKFold, cross_val_score, train_test_split,
GridSearchCV
from sklearn.preprocessing import StandardScaler
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

# Prepare Data
subset_data2_sampled = subset_data2.sample(n=500, random_state=42)
X_sample = subset_data2_sampled[['TEMP', 'DEWP', 'WSPM']]
```

```
y_sample = subset_data2_sampled['NO2_safety'].map({'safe': 0, 'unsafe': 1})  # Convert target to numeric

# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X_sample, y_sample, test_size=0.2, random_state=42)

# Standardize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)


# Function to Train and Evaluate SVM
def evaluate_svm(kernel_type, degree=None, gamma=None, coef0=None, C=1.0):
    print(f"\n--- SVM with {kernel_type} kernel ---")


    svm_model = SVC(kernel=kernel_type, degree=degree if degree else 3, gamma=gamma if gamma else 'scale',
                    coef0=coef0 if coef0 else 0, C=C, probability=True, random_state=42)
    svm_model.fit(X_train_scaled, y_train)
```

```
    # Predictions and probabilities
    y_pred = svm_model.predict(X_test_scaled)
    y_pred_prob = svm_model.predict_proba(X_test_scaled)[:, 1]  # Probabilities for ROC and
AUC

    # Confusion Matrix
    conf_matrix = confusion_matrix(y_test, y_pred)
    print(conf_matrix)

    # Metrics
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    specificity = recall_score(y_test, y_pred, pos_label=0)
    f_score = f1_score(y_test, y_pred)
    roc_auc = roc_auc_score(y_test, y_pred_prob)

    print(f"Precision: {precision:.2f}")
    print(f"Recall (Sensitivity): {recall:.2f}")
    print(f"Specificity: {specificity:.2f}")
    print(f"F-Score: {f_score:.2f}")
    print(f"AUC: {roc_auc:.2f}")
```

```
    # ROC Curve
    fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)
    plt.figure(figsize=(8, 6))
    plt.plot(fpr, tpr, color='blue', label=f'AUC = {roc_auc:.2f}')
    plt.fill_between(fpr, tpr, color='lightblue', alpha=0.5)  # Shade the AUC curve
    plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
    plt.title(f'ROC Curve ({kernel_type} kernel)')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.legend(loc='lower right')
    plt.show()

    print("Evaluate SVM: ")
    # Stratified K-Fold Cross-Validation
    skf = StratifiedKFold(n_splits=3)
    cv_scores = cross_val_score(svm_model, X_sample, y_sample, cv=skf, scoring='accuracy')
    print(f"Cross-Validation Scores: {cv_scores}")
    print(f"Average Cross-Validation Accuracy: {np.mean(cv_scores):.2f}")


# Hyperparameter tuning using GridSearchCV for different kernels
```

```
def tune_svm(kernel_type):
    if kernel_type == 'linear':
        param_grid = {
            'C': [0.1, 1, 10],  # Regularization parameter
        }
    elif kernel_type == 'poly':
        param_grid = {
            'C': [0.1, 1, 10],
            'degree': [2, 3, 4],
            'gamma': ['scale', 'auto'],
            'coef0': [0, 1, 5]
        }
    elif kernel_type == 'rbf':
        param_grid = {
            'C': [0.1, 1, 10],
            'gamma': ['scale', 'auto']
        }
    else:
        raise ValueError("Invalid kernel type. Choose 'linear', 'poly', or 'rbf'.")

    #  SVM model
    svm_model = SVC(kernel=kernel_type, probability=True, random_state=42)
```

```
    #  GridSearchCV
    grid_search = GridSearchCV(estimator=svm_model, param_grid=param_grid, cv=3, n_jobs=-
1, scoring='accuracy')

    # Fit GridSearchCV
    grid_search.fit(X_train_scaled, y_train)

    # Print the best parameters and score
    print(f"\nBest parameters found by GridSearchCV for {kernel_type} kernel:")
    print(grid_search.best_params_)
    print(f"Best cross-validation accuracy: {grid_search.best_score_:.2f}")

    # Evaluate the best model
    best_model = grid_search.best_estimator_
    evaluate_svm(kernel_type=kernel_type, C=grid_search.best_params_['C'],
                 degree=grid_search.best_params_.get('degree', 3),
                 gamma=grid_search.best_params_.get('gamma', 'scale'),
                 coef0=grid_search.best_params_.get('coef0', 0))


# Tune and evaluate the SVM with different kernels
```

```
# Linear Kernel
tune_svm(kernel_type='linear')

# Polynomial Kernel
tune_svm(kernel_type='poly')

# Radial Basis Function (RBF) Kernel
tune_svm(kernel_type='rbf')

############################################## Naive Bayes
#####################################################################
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import GridSearchCV, StratifiedKFold
from sklearn.metrics import confusion_matrix, classification_report, roc_curve, auc,
precision_score, recall_score, f1_score
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

# Prepare the data
subset_data2_sampled = subset_data2.sample(n=10000, random_state=42)
X = subset_data2_sampled[['TEMP', 'DEWP', 'WSPM']]
```

```
y = subset_data2_sampled['NO2_safety'].map({'safe': 0, 'unsafe': 1})

# Split into training and test data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# StratifiedKFold cross-validation
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# Define the parameter grid for Naive Bayes
param_grid = {
    'var_smoothing': [1e-9, 1e-8, 1e-7]
}

#  Naive Bayes classifier
nb = GaussianNB()

# Set up GridSearchCV
grid_search = GridSearchCV(estimator=nb, param_grid=param_grid, scoring='accuracy', cv=cv,
n_jobs=-1)

# Fit the model
grid_search.fit(X_train, y_train)
```

```
# Get the best model and its hyperparameters
best_nb = grid_search.best_estimator_
print("Best hyperparameters:", grid_search.best_params_)

# Make predictions on the test set
y_pred = best_nb.predict(X_test)

# Evaluate the model
print("Evaluate Naive Bayes: ")
# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(conf_matrix)

# Precision, Recall, F-score
precision = precision_score(y_test, y_pred, pos_label=1)
recall = recall_score(y_test, y_pred, pos_label=1)
f_score = f1_score(y_test, y_pred, pos_label=1)
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F-score: {f_score}")
```

```
# Specificity (True Negative Rate)
specificity = recall_score(y_test, y_pred, pos_label=0)
print(f"Specificity: {specificity}")

# ROC and AUC
fpr, tpr, thresholds = roc_curve(y_test, best_nb.predict_proba(X_test)[:, 1])
roc_auc = auc(fpr, tpr)
print("AUC: ", roc_auc)
# Plot ROC Curve
# Plot ROC Curve with shading for AUC
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', label=f'AUC = {roc_auc:.2f}')
plt.fill_between(fpr, tpr, color='blue', alpha=0.2)  # Shade the area under the ROC curve
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.title('ROC Curve with AUC Shading for Naive Bayes')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc='lower right')
plt.show()

# Display classification report
```

```
print("Classification Report:")
print(classification_report(y_test, y_pred))

# Cross-validation performance
cv_results = grid_search.cv_results_
print("Grid Search CV results:")
for mean_score, params in zip(cv_results['mean_test_score'], cv_results['params']):
    print(f"Accuracy: {mean_score:.4f} for {params}")

################################ Random Forest
################################################################################


from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier,
StackingClassifier
from sklearn.model_selection import GridSearchCV, StratifiedKFold, train_test_split
from sklearn.metrics import confusion_matrix, classification_report, roc_curve, auc,
precision_score, recall_score, f1_score
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
import matplotlib.pyplot as plt
```

```
# Sample data
subset_data2_sampled = subset_data2.sample(n=10000, random_state=42)
X = subset_data2_sampled[['TEMP', 'DEWP', 'WSPM']]
y = subset_data2_sampled['NO2_safety'].map({'safe': 0, 'unsafe': 1})

# Split training and test data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize StratifiedKFold cross-validation
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

#  Random Forest Classifier
rf = RandomForestClassifier()

# Define the parameter grid for Random Forest
param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [None, 10],
    'min_samples_split': [2],
    'min_samples_leaf': [1],
    'bootstrap': [True]
```

```
}



# Parameter grid for Gradient Boosting
param_grid_gb = {
    'n_estimators': [50, 100],
    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [3, 5]
}

# Random Forest Model
rf_model = RandomForestClassifier(random_state=42)
grid_search_rf = GridSearchCV(estimator=rf_model, param_grid={'n_estimators': [50, 100],
'max_depth': [3, 5]}, scoring='accuracy', cv=cv, n_jobs=-1)
grid_search_rf.fit(X_train, y_train)

# Gradient Boosting Model
gb_model = GradientBoostingClassifier(random_state=42)
grid_search_gb = GridSearchCV(estimator=gb_model, param_grid=param_grid_gb,
scoring='accuracy', cv=cv, n_jobs=-1)
grid_search_gb.fit(X_train, y_train)
```

```
# Stacking Model with base models
base_learners = [
    ('dt', DecisionTreeClassifier(max_depth=3)),
    ('svc', SVC(probability=True)),
    ('rf', RandomForestClassifier(n_estimators=100, random_state=42))
]
meta_model = LogisticRegression()
stacking_model = StackingClassifier(estimators=base_learners, final_estimator=meta_model)
stacking_model.fit(X_train, y_train)

# Evaluate Random Forest
rf_best_model = grid_search_rf.best_estimator_
y_pred_rf = rf_best_model.predict(X_test)

# Evaluate Gradient Boosting
gb_best_model = grid_search_gb.best_estimator_
y_pred_gb = gb_best_model.predict(X_test)

# Evaluate Stacking
y_pred_stacking = stacking_model.predict(X_test)

# Confusion Matrix for Random Forest
```

```
conf_matrix_rf = confusion_matrix(y_test, y_pred_rf)
print("Random Forest Confusion Matrix:")
print(conf_matrix_rf)

# Confusion Matrix for Gradient Boosting
conf_matrix_gb = confusion_matrix(y_test, y_pred_gb)
print("Gradient Boosting Confusion Matrix:")
print(conf_matrix_gb)

# Confusion Matrix for Stacking
conf_matrix_stacking = confusion_matrix(y_test, y_pred_stacking)
print("Stacking Confusion Matrix:")
print(conf_matrix_stacking)

# Precision, Recall, F-score for Random Forest
precision_rf = precision_score(y_test, y_pred_rf, pos_label=1)
recall_rf = recall_score(y_test, y_pred_rf, pos_label=1)
f_score_rf = f1_score(y_test, y_pred_rf, pos_label=1)
# Specificity for Random Forest
TN_rf, FP_rf, FN_rf, TP_rf = conf_matrix_rf.ravel()
specificity_rf = TN_rf / (TN_rf + FP_rf)
print(f"Random Forest - Precision: {precision_rf}, Recall: {recall_rf}, F-score: {f_score_rf},
```

```
Specificity: {specificity_rf}")

# Precision, Recall, F-score for Gradient Boosting
precision_gb = precision_score(y_test, y_pred_gb, pos_label=1)
recall_gb = recall_score(y_test, y_pred_gb, pos_label=1)
f_score_gb = f1_score(y_test, y_pred_gb, pos_label=1)
# Specificity for Gradient Boosting
TN_gb, FP_gb, FN_gb, TP_gb = conf_matrix_gb.ravel()
specificity_gb = TN_gb / (TN_gb + FP_gb)
print(f"Gradient Boosting - Precision: {precision_gb}, Recall: {recall_gb}, F-score:
{f_score_gb}, Specificity: {specificity_gb}")

# Precision, Recall, F-score for Stacking
precision_stacking = precision_score(y_test, y_pred_stacking, pos_label=1)
recall_stacking = recall_score(y_test, y_pred_stacking, pos_label=1)
f_score_stacking = f1_score(y_test, y_pred_stacking, pos_label=1)
# Specificity for Stacking
TN_stacking, FP_stacking, FN_stacking, TP_stacking = conf_matrix_stacking.ravel()
specificity_stacking = TN_stacking / (TN_stacking + FP_stacking)
print(f"Stacking - Precision: {precision_stacking}, Recall: {recall_stacking}, F-score:
{f_score_stacking}, Specificity: {specificity_stacking}")
```

```
# ROC and AUC for Random Forest
fpr_rf, tpr_rf, thresholds_rf = roc_curve(y_test, rf_best_model.predict_proba(X_test)[:,1])
roc_auc_rf = auc(fpr_rf, tpr_rf)
print("AUC for Random Forest: ")
print(roc_auc_rf)
# ROC and AUC for Gradient Boosting
fpr_gb, tpr_gb, thresholds_gb = roc_curve(y_test, gb_best_model.predict_proba(X_test)[:,1])
roc_auc_gb = auc(fpr_gb, tpr_gb)

# ROC and AUC for Stacking
fpr_stacking, tpr_stacking, thresholds_stacking = roc_curve(y_test,
stacking_model.predict_proba(X_test)[:,1])
roc_auc_stacking = auc(fpr_stacking, tpr_stacking)

# Plot ROC Curve for Random Forest
plt.figure(figsize=(10, 7))
plt.plot(fpr_rf, tpr_rf, color='blue', label=f'Random Forest AUC = {roc_auc_rf:.2f}')
plt.fill_between(fpr_rf, tpr_rf, alpha=0.2, color='blue')  # AUC shading
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')

plt.title('ROC Curve for Random Forest')
plt.xlabel('False Positive Rate')
```

```
plt.ylabel('True Positive Rate')
plt.legend(loc='lower right')
plt.show()

# classification report for Random Forest
print("Random Forest Classification Report:")
print(classification_report(y_test, y_pred_rf))

# classification report for Gradient Boosting
print("Gradient Boosting Classification Report:")
print(classification_report(y_test, y_pred_gb))

# classification report for Stacking
print("Stacking Classification Report:")
print(classification_report(y_test, y_pred_stacking))


################################## Neural Networks
#############################################################################
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import StratifiedKFold, cross_val_score
from sklearn.metrics import precision_score, recall_score, f1_score, classification_report,
```

```
confusion_matrix, roc_curve, auc
import matplotlib.pyplot as plt

# Sample data
subset_data2_sampled = subset_data2.sample(n=10000, random_state=42)
X = subset_data2_sampled[['TEMP', 'DEWP', 'WSPM']]
y = subset_data2_sampled['NO2_safety'].map({'safe': 0, 'unsafe': 1})

# Initialize StratifiedKFold cross-validation
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# Initialize Multi-layer Perceptron (Neural Network)
mlp_model = MLPClassifier(hidden_layer_sizes=(50, 50), max_iter=1000, random_state=42)

# Perform Stratified K-fold cross-validation for the Neural Network (MLP)
print("\nEvaluating Multi-layer Perceptron (MLP) with Stratified K-fold Cross Validation:")

# Cross-validation scores (accuracy)
accuracy_scores = cross_val_score(mlp_model, X, y, cv=cv, scoring='accuracy')
print(f"Accuracy: {accuracy_scores.mean():.4f} (+/- {accuracy_scores.std():.4f})")

# Cross-validation scores (precision, recall, f1-score)
```

```
precision_scores = cross_val_score(mlp_model, X, y, cv=cv, scoring='precision')
recall_scores = cross_val_score(mlp_model, X, y, cv=cv, scoring='recall')
f1_scores = cross_val_score(mlp_model, X, y, cv=cv, scoring='f1')

print(f"Precision: {precision_scores.mean():.4f} (+/- {precision_scores.std():.4f})")
print(f"Recall: {recall_scores.mean():.4f} (+/- {recall_scores.std():.4f})")
print(f"F1 Score: {f1_scores.mean():.4f} (+/- {f1_scores.std():.4f})")

# Fit the model on the entire dataset
mlp_model.fit(X, y)
y_pred = mlp_model.predict(X)

# Confusion Matrix
conf_matrix_mlp = confusion_matrix(y, y_pred)
print("Confusion Matrix for Multi-layer Perceptron (MLP):")
print(conf_matrix_mlp)

# Extract TN, FP, FN, TP from the confusion matrix
tn, fp, fn, tp = conf_matrix_mlp.ravel()

# Calculate specificity
specificity = tn / (tn + fp)
```

```
print(f"Specificity: {specificity:.4f}")

# ROC Curve and AUC for Multi-layer Perceptron
fpr_mlp, tpr_mlp, thresholds_mlp = roc_curve(y, mlp_model.predict_proba(X)[:, 1])
roc_auc_mlp = auc(fpr_mlp, tpr_mlp)
print("AUC: ", roc_auc_mlp)

# Plot ROC Curve for Multi-layer Perceptron
plt.figure(figsize=(10, 7))
plt.plot(fpr_mlp, tpr_mlp, color='purple', label=f'MLP AUC = {roc_auc_mlp:.2f}')
plt.fill_between(fpr_mlp, tpr_mlp, alpha=0.2, color='purple')  # AUC shading
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.title('ROC Curve for MLP')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc='lower right')
plt.show()

# Display classification report for MLP
print("Classification Report for Multi-layer Perceptron (MLP):")
print(classification_report(y, y_pred))
```

```
##### Combined ROC Curve ##################################################
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# Sample data (assuming you have X_train, X_test, y_train, y_test ready)
# X_train, X_test, y_train, y_test already defined

# Initialize classifiers
classifiers = {
    "Decision Tree": DecisionTreeClassifier(),
    "Logistic Regression": LogisticRegression(),
    "KNN": KNeighborsClassifier(),
    "SVM": SVC(probability=True),  # SVM needs probability=True for predict_proba
    "Naive Bayes": GaussianNB(),
```

```
    "Random Forest": RandomForestClassifier(),
    "Neural Network": MLPClassifier(max_iter=1000)
}

# Colors for the ROC curves
colors = {
    "Decision Tree": "blue",
    "Logistic Regression": "green",
    "KNN": "orange",
    "SVM": "red",
    "Naive Bayes": "purple",
    "Random Forest": "brown",
    "Neural Network": "pink"
}

# Plot ROC curves
plt.figure(figsize=(12, 8))

for name, clf in classifiers.items():
    # Train the classifier
    clf.fit(X_train, y_train)
```

```
    # Predict probabilities
    y_prob = clf.predict_proba(X_test)[:, 1]

    # Compute ROC curve and AUC
    fpr, tpr, _ = roc_curve(y_test, y_prob)
    roc_auc = auc(fpr, tpr)

    # Plot ROC curve
    plt.plot(fpr, tpr, color=colors[name], label=f"{name} (AUC = {roc_auc:.2f})")
    plt.fill_between(fpr, tpr, alpha=0.1, color=colors[name])

# Plot diagonal line for random guessing
plt.plot([0, 1], [0, 1], color="gray", linestyle="--")

# Add labels, title, and legend
plt.title("ROC Curve Comparison of Multiple Classifiers")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend(loc="lower right")
```

```
plt.grid()

plt.show()
```

Appendix D: Phase IV Clustering and Association Rule Code

```
##############################################################################
########
############ Phase 4: Clustering and Association
#######################################
##############################################################################
########


############################## K Means
#################################################

from sklearn.cluster import KMeans

from sklearn.metrics import silhouette_score

import matplotlib.pyplot as plt

import seaborn as sns


# Down sample the data for faster run time

down_sample_data = subset_data2.sample(frac=0.05, random_state=42)  # Use 5% of the data
for faster processing

X_down = down_sample_data[['NO2', 'TEMP', 'DEWP', 'WSPM']]
```

```
#  range of k values to test for silhouette analysis
silhouette_scores = []
inertia_values = []
k_range = range(2, 10)  # Trying values of k from 2 to 10

#fit KMeans models for different k and calculate silhouette scores
for k in k_range:
kmeans = KMeans(n_clusters=k, init='k-means++',max_iter=100, random_state=42)
kmeans.fit(X_down)

# Calculate silhouette score
silhouette_avg = silhouette_score(X_down, kmeans.labels_)
silhouette_scores.append(silhouette_avg)

# Calculate inertia (within-cluster variation)
inertia_values.append(kmeans.inertia_)

# Plot Silhouette Scores and Inertia
plt.figure(figsize=(12, 6))
```

```
# Plot Silhouette Scores
plt.subplot(1, 2, 1)
plt.plot(k_range, silhouette_scores, marker='o')
plt.title('Silhouette Analysis for K Selection')
plt.xlabel('Number of clusters (k)')
plt.ylabel('Silhouette Score')

# Plot Inertia (Within-cluster variation)
plt.subplot(1, 2, 2)
plt.plot(k_range, inertia_values, marker='o')
plt.title('Within-cluster Variation (Inertia)')
plt.xlabel('Number of clusters (k)')
plt.ylabel('Inertia')

plt.tight_layout()
plt.show()

# Choose the best k based on silhouette score and fit the final KMeans model
best_k = k_range[silhouette_scores.index(max(silhouette_scores))]
print(f"Optimal number of clusters (k): {best_k}")

# Fit the final KMeans model with the best k
```

```
kmeans_final = KMeans(n_clusters=best_k, init='k-means++',max_iter=100, random_state=42)
kmeans_final.fit(X_down)

# Cluster labels
down_sample_data['KMeans_Labels'] = kmeans_final.labels_

optimal_silhouette_score = max(silhouette_scores)
print(f"Silhouette Score for optimal k ({best_k}): {optimal_silhouette_score:.3f}")

########################## DBSCAN
#########################################################################
from sklearn.cluster import DBSCAN
from sklearn.preprocessing import StandardScaler

down_sample_data = subset_data2.sample(frac=0.05, random_state=42)  # Use 5% of the data
for faster processing
X_down = down_sample_data[['NO2', 'TEMP', 'DEWP', 'WSPM']]
# Standardize the data for DBSCAN
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_down)

# Fit DBSCAN with a chosen epsilon and min_samples
```

```
dbscan = DBSCAN(eps=0.5, min_samples=5)
dbscan_labels = dbscan.fit_predict(X_scaled)

# Add DBSCAN labels to the dataset
down_sample_data['DBSCAN_Labels'] = dbscan_labels

# Number of clusters and outliers (-1 indicates outliers)
num_clusters = len(set(dbscan_labels)) - (1 if -1 in dbscan_labels else 0)
print(f"Number of clusters found by DBSCAN: {num_clusters}")
print(f"Number of outliers: {list(dbscan_labels).count(-1)}")

# Visualize DBSCAN clusters
plt.figure(figsize=(8, 6))
sns.scatterplot(x='TEMP', y='NO2', hue='DBSCAN_Labels', data=down_sample_data,
palette='Set1', style='DBSCAN_Labels')
plt.title('DBSCAN Clustering')
plt.xlabel('Temperature (TEMP)')
plt.ylabel('NO2')
plt.legend(title='Cluster Labels')
plt.show()

########################## Apriori
```

##3#####

```
from mlxtend.frequent_patterns import apriori, association_rules
import pandas as pd
"""
# Discretize continuous features
def discretize_feature(column, bins, labels):
return pd.cut(column, bins=bins, labels=labels)

# Discretize the features (NO2, TEMP, DEWP, WSPM)
subset_data2['NO2_Discretized'] = discretize_feature(subset_data2['NO2'], bins=[0, 10, 20, 30, 40, 50], labels=['Low', 'Medium', 'High', 'Very High', 'Extremely High'])
subset_data2['TEMP_Discretized'] = discretize_feature(subset_data2['TEMP'], bins=[-10, 0, 10, 20, 30, 40], labels=['Very Cold', 'Cold', 'Mild', 'Warm', 'Hot'])
subset_data2['DEWP_Discretized'] = discretize_feature(subset_data2['DEWP'], bins=[-10, 0, 10, 20, 30], labels=['Very Low', 'Low', 'Medium', 'High'])
subset_data2['WSPM_Discretized'] = discretize_feature(subset_data2['WSPM'], bins=[0, 5, 10, 15, 20], labels=['Low', 'Medium', 'High', 'Very High'])

# Convert to one-hot encoding for Apriori
subset_data2_onehot = pd.get_dummies(subset_data2[['NO2_Discretized', 'TEMP_Discretized', 'DEWP_Discretized', 'WSPM_Discretized']])
```

```
# Apply Apriori algorithm to find frequent item sets
frequent_itemsets = apriori(subset_data2_onehot, min_support=0.05, use_colnames=True)

# Generate association rules
rules = association_rules(frequent_itemsets, metric="lift", min_threshold=1.0,
num_itemsets=len(frequent_itemsets))

# Display rules
pd.set_option('display.max_rows', None)  # Display all rows
pd.set_option('display.width', None)  # Prevent line wrapping
```

```
print(f"Number of association rules found: {len(rules)}")

print(rules[['antecedents', 'consequents', 'support', 'confidence', 'lift']])
```

References

Jafari, R. (2024.). *CS5805 : Machine Learning I Lecture #2*. *Powerpoint Lecture Slides*. Virginia Polytechnic Institute and State University

Jafari, R. (2024.). *CS5805 : Machine Learning I Lecture #3*. *Powerpoint Lecture Slides*. Virginia Polytechnic Institute and State University

Jafari, R. (2024.). *CS5805 : Machine Learning I Lecture #4*. *Powerpoint Lecture Slides*. Virginia Polytechnic Institute and State University

Jafari, R. (2024.). *CS5805 : Machine Learning I Lecture #5*. *Powerpoint Lecture Slides*. Virginia Polytechnic Institute and State University

Jafari, R. (2024.). *CS5805 : Machine Learning I Lecture #6*. *Powerpoint Lecture Slides*. Virginia Polytechnic Institute and State University

Jafari, R. (2024.). *CS5805 : Machine Learning I Lecture #7*. *Powerpoint Lecture Slides*. Virginia Polytechnic Institute and State University

Jafari, R. (2024.). *CS5805 : Machine Learning I Lecture #8*. *Powerpoint Lecture Slides*. Virginia Polytechnic Institute and State University

Jafari, R. (2024.). *CS5805 : Machine Learning I Lecture #9*. *Powerpoint Lecture Slides*. Virginia Polytechnic Institute and State University

Jafari, R. (2024.). *CS5805 : Machine Learning I Lecture #10*. *Powerpoint Lecture Slides*. Virginia Polytechnic Institute and State University

Jafari, R. (2024.). *CS5805 : Machine Learning I Lecture #11*. *Powerpoint Lecture Slides*. Virginia Polytechnic Institute and State University

Jafari, R. (2024.). *CS5805 : Machine Learning I Lecture #12*. *Powerpoint Lecture Slides*. Virginia Polytechnic Institute and State University

Jafari, R. (2024.). *CS5805 : Machine Learning I Lecture #13*. *Powerpoint Lecture Slides*. Virginia Polytechnic Institute and State University

Jafari, R. (2024.). *CS5805 : Machine Learning I Lecture #14*. *Powerpoint Lecture Slides*. Virginia Polytechnic Institute and State University

Jafari, R. (2024.). *CS5805 : Machine Learning I Lecture #15*. *Powerpoint Lecture Slides*. Virginia Polytechnic Institute and State University

GeeksforGeeks. (2019, May 6). *DBSCAN Clustering in ML | Density based clustering*. GeeksforGeeks. https://www.geeksforgeeks.org/dbscan-clustering-in-ml-density-based-clustering/#

Alokpandey. (2024, October 11). *The Role of Inertia in K-means Clustering - Alokpandey - Medium*. Medium. https://medium.com/@alokpand885148/the-role-of-inertia-in-k-means-clustering-1987e9274a88

WHO. (2021, September). *C40 Knowledge Community*. Www.c40knowledgehub.org. https://www.c40knowledgehub.org/s/article/WHO-Air-Quality-Guidelines?language=en_US

collections — Container datatypes — Python 3.8.3 documentation. (n.d.). Docs.python.org. https://docs.python.org/3/library/collections.html